AN EXECUTIVE SYSTEM FOR A DEC 339 COMPUTER DISPLAY TERMINAL

James H. Jackson
THE UNIVERSITY OF MICHIGAN

Technical Report 15

AN EXECUTIVE SYSTEM
FOR A DEC 339 COMPUTER DISPLAY TERMINAL

James H. Jackson

CONCOMP: Research in Conversational Use of Computers
F. H. Westervelt, Project Director
ORA Project 07449

Supported by:
ADVANCED RESEARCH PROJECTS AGENCY
DEPARTMENT OF DEFENSE
WASHINGTON, D.C.

CONTRACT NO. DA-49-083 OSA-3050
ARPA ORDER NO. 716

Administered through:
OFFICE OF RESEARCH ADMINISTRATION ANN ARBOR

December 1968
ABSTRACT

This report describes a real-time multiprogramming software system for a DEC 339 computer display terminal, which may communicate with an external computer through a serial-synchronous data set. The system is designed to support both programs which require the attention of an external computer while they are being executed and programs which are independent of external computation service. For either type of program, the entire graphics support is provided by the 339 system, but the interpretation of the relations implied by the graphics may be performed either in the 339 or in an external computer. Multiprogramming facility is provided to facilitate effective use of I/O devices in order to cope with the demands of a real-time environment.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>iii</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. SYSTEM ORGANIZATION</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Bootstrap Arrangement</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Tasks</td>
<td>4</td>
</tr>
<tr>
<td>2.3 States of the System</td>
<td>4</td>
</tr>
<tr>
<td>2.4 Entering System State</td>
<td>5</td>
</tr>
<tr>
<td>3. SYSTEM SUBROUTINES</td>
<td>7</td>
</tr>
<tr>
<td>3.1 Word Queues</td>
<td>8</td>
</tr>
<tr>
<td>3.2 Task Scheduling and I/O Device Allocation</td>
<td>11</td>
</tr>
<tr>
<td>3.3 Format Conversions</td>
<td>16</td>
</tr>
<tr>
<td>3.4 Buffered I/O</td>
<td>17</td>
</tr>
<tr>
<td>3.4.1 Dataphone I/O</td>
<td>18</td>
</tr>
<tr>
<td>3.4.2 Paper Tape I/O</td>
<td>22</td>
</tr>
<tr>
<td>3.4.3 Teletype I/O</td>
<td>25</td>
</tr>
<tr>
<td>3.5 Nonbuffered I/O</td>
<td>26</td>
</tr>
<tr>
<td>3.6 Push-Button Processing</td>
<td>29</td>
</tr>
<tr>
<td>3.7 Display Control Communication</td>
<td>31</td>
</tr>
<tr>
<td>3.8 Light Pen Tracking</td>
<td>32</td>
</tr>
<tr>
<td>3.9 Display Structure Topology</td>
<td>35</td>
</tr>
<tr>
<td>3.10 Level Modification</td>
<td>40</td>
</tr>
<tr>
<td>3.11 Text List Manipulation</td>
<td>61</td>
</tr>
<tr>
<td>4. IDLE-TIME TASK</td>
<td>64</td>
</tr>
<tr>
<td>4.1 Copy Functions</td>
<td>64</td>
</tr>
<tr>
<td>4.2 Scheduling of User Tasks</td>
<td>67</td>
</tr>
<tr>
<td>4.3 Clearing the Task Queue or Display Storage</td>
<td>67</td>
</tr>
<tr>
<td>4.4 Teletype to Dataphone Transmission</td>
<td>68</td>
</tr>
<tr>
<td>4.5 Entering User State</td>
<td>68</td>
</tr>
<tr>
<td>5. SYSTEM CAPABILITY</td>
<td>69</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>70</td>
</tr>
<tr>
<td>APPENDICES</td>
<td></td>
</tr>
<tr>
<td>A LISTING OF THE EXECUTIVE SYSTEM</td>
<td>A-1</td>
</tr>
<tr>
<td>B SUMMARY OF SYSTEM SUBROUTINES</td>
<td>B-1</td>
</tr>
<tr>
<td>C SUMMARY OF IOT INSTRUCTIONS</td>
<td>C-1</td>
</tr>
<tr>
<td>D ASSEMBLY LANGUAGE</td>
<td>D-1</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

The objective of this report is to describe the conceptual organization of the SEL (Systems Engineering Laboratory's) Executive System for a 339 computer display terminal, as well as to provide a manual for its use. More specifically, the hardware configuration for which the System was designed consists of the following items (plus necessary interfaces, multiplexors, etc.):

DEC PDP-9 with at least two 8192-word memory banks
DEC KE09A extended arithmetic element
DEC 338 display control (less PDP-8)
DEC AF01B A/D converter
DEC AA01A D/A converter
AT&T 201A data set

The System provides both a multiprogramming capability (based on I/O slicing, rather than time-slicing) and a complete set of operators for maintaining a highly structured display file and for interrogating it for relational properties.

Since an on-line operator tends to produce a burst of inputs and then to be idle for a relatively long period of time, appropriate feedback to each input must be provided rapidly if the operator is to be allowed to proceed at his own rate. If the terminal were not multiprogrammed, the processing of one input would have to be completed before processing of the next could be begun. Consequently, bursts of operator activity could not be effectively handled by this scheme. However, if a multiprogramming system (where the users of the system are programs which respond to various inputs) were used, feedback to each input could be produced almost immediately, and the remaining (and usually time-consuming) part of the processing could be deferred until a later time.
Bandwidth limitations on the data link between the remote computer and the central timesharing system suggest that programs be distributed between the central computer and the remote computer such that dataphone traffic is minimized (subject to the constraint of the capacity of the remote machine). In terms of a remote display terminal, this usually means that the relations implied by a display file, rather than the display file itself, be transmitted. For this reason, the remote system should provide a facility for constructing a display file based partly on relational information, and for interrogating a display file for relational information.

A general discussion of the organization of the System and detailed discussions of the various system subroutines and the idle-time task follow. A complete listing of the System is given in Appendix A, a summary of system subroutines is given in Appendix B, a summary of all IOT instructions pertinent to the hardware configuration is given in Appendix C, and a brief description of the assembly language used in the examples is given in Appendix D.
2. SYSTEM ORGANIZATION

2.1 Bootstrap Arrangement

The System should be loaded by the following procedure:

1) Place the system tape in the reader.
2) Set all switches to 0 (down).
3) Depress the read-in key.

This procedure causes the first record, which is written in hardware RIM format, to be read, and the computer to be started at the last location loaded. The record read is the bootstrap loader represented by the following assembly code:

```
$ORG 0
IOT 144 SELECT READER IN BINARY MODE
IOT 101 SKIP ON READER FLAG
JMP *-1 WAIT FOR READER FLAG
IOT 112 READ READER BUFFER
DAC* 10 LOAD A WORD
JMP 0 READ NEXT WORD
HLT
HLT
$DC 17731 INITIAL INDEX VALUE
JMP 0 START BOOTSTRAP LOADER
```

The bootstrap loader is capable of loading one binary block (Section 3.4.2) starting at location 17732g, but is not capable of detecting the end of the block. However, the block which immediately follows the bootstrap loader on the system tape is loaded into locations 17732g, ..., 17777g, 0. The word loaded into location 0 is a JMP instruction to the beginning of a more sophisticated loader, which is contained in the block read by the bootstrap loader.

The loader loaded by the bootstrap loader is capable of loading an arbitrary number of binary blocks, and it is this
loader which loads the System. Immediately following the last block of the System is a one-word block which modifies the loader and causes execution of the System to begin.

At the end of the loading process, the System occupies locations 0-117778, and the bootstrap loader and system loader are no longer usable. (The storage occupied by the system loader is salvaged by the System for display structure use at a later time.)

2.2 *Tasks*

Each program written to run with the System is called a "task" and is identified by its entry point. The System maintains a task queue, each entry of which consists of the entry point for the task, together with other information required to determine the eligibility of the task or to restore the contents of certain registers before the task is executed. Whenever execution of a task is begun, the task is removed from the task queue.

A task is entered by a JMP instruction (rather than a JMS instruction, as in some other similar systems) and is subject to the following restrictions:

1) No user task may contain an IOT instruction.
2) No user task may store in core bank 0. (No user task should be loaded into core bank 0. Locations 120008-177778 are used by the System to store the display structure.)
3) A task which uses an allocatable I/O device (via system subroutines) must allocate the device before calling the system subroutine to use it, and must release the device before terminating. (The task may allocate and/or release the device implicitly by insuring that another task is scheduled to perform the function.)

2.3 *States of the System*

At any instant, the System is operating in one of two states:
1) System state—A special system task, called the idle-time task (Section 4), is executed. However, an incoming message from the 201A dataphone which is not directed to a user task will cause the 201-to-teleprinter task (Section 3.4.1) to be scheduled.

2) User state—All scheduled user tasks are executed and the idle-time task is not executed. The 201-to-teleprinter task is scheduled when necessary as in system state.

The states of the System may be depicted by the following diagram:

2.4 Entering System State

Whenever one of the following events occurs, the System is reinitialized (i.e., all I/O activity is stopped, the task queue and all buffers are cleared, and all I/O devices are
released), and system state is entered:

1) The System is reloaded.
2) The currently executing user task terminates with the 'k queue empty, and all output buffers become empty.
3) An unidentifiable interrupt occurs.
4) The manual interrupt button is pressed. (The manual interrupt is used by the operator to reinitialize the System in case of emergency.)
5) The task queue overflows.
6) The program is started at location 22g via the panel switches.
7) An illegal instruction (operation code 60g) is executed.

Immediately after system state is entered, a comment describing which one of the above events occurred is typed on the teletype, and, if enough free display storage remains, it is displayed on the screen. Reinitializing the System does not include clearing the display storage area, but it does cause the active structure to be detached from the highest active level (Section 3.9).
3. SYSTEM SUBROUTINES

Sections 3.1 through 3.11 describe the various system subroutines which are callable from user tasks. The entry point to each subroutine occupies a fixed position in a vector such that the actual code for the subroutine may be relocated (by some future modification of the System) without requiring user tasks to be reassembled. Since the System occupies core bank 0 and user tasks cannot be loaded into bank 0, system subroutines must be called via an indirect reference, i.e., if \( \alpha \) is the symbolic name of a system subroutine, a call to \( \alpha \) is written in the following form:

\[
\text{JMS*} = \alpha
\]

Most of the system subroutines return immediately after the JMS instructions which call them. (Parameters are passed in the AC and MQ.) However, several subroutines have "failure returns," i.e., a return is made immediately after the location containing the JMS instruction if the function which the subroutine must perform cannot be performed. If the subroutine succeeds, return is made to the next location. The two types of calling sequences may be illustrated as follows:

Subroutine with no failure return:

\[
\text{JMS*} = \alpha \\
------ (return)
\]

Subroutine with failure return:

\[
\text{JMS*} = \alpha \\
------ (failure return) \\
------ (success return)
\]

A subroutine which has a failure return is denoted by an asterisk (*) appended to its symbolic name in Sections 3.1 through 3.11. (The asterisk is not part of the symbolic name.)
3.1 Word Queues

The basic structure which supports cyclic I/O buffering and task scheduling in the System is a word queue. This structure consists of a block of three words, called control words, followed by n data words and has the properties of both a first-in first-out (FIFO) queue and a last-in first-out (LIFO) queue.

A word queue is represented in core as shown by the following diagram:

![Diagram of word queue](image)

The symbols in the diagram are interpreted as follows:

- \( q_0 \): Address of the word queue. By convention, this is the address of the first control word.
- \( q_1 \): Pointer to the physically last data word in the queue.
- \( q_2 \): Pointer to the last word put into the queue (FIFO sense).
- \( q_3 \): Pointer to the last word taken out of the queue.
The word queue is empty whenever \( q_2 = q_3 \), and it is full whenever \( q_3 = q_2 + 1 \) or \( q_3 = q_0 + 3 \) and \( q_2 = q_1 \). The maximum number of words which may be stored in the queue is then \( n - 1 \).

The cyclic nature of the word queue requires that the terms incrementing and decrementing a pointer be defined for this structure. A pointer \( q \) is "incremented" if it is modified so that it takes on the value

\[
q' = \begin{cases} 
q + 1, & \text{if } q \neq q_1 \\
q_0 + 3, & \text{if } q = q_1
\end{cases}
\]

A pointer \( q \) is "decremented" if it is modified so that it takes on the value

\[
q'' = \begin{cases} 
q + 1, & \text{if } q \neq q_0 + 3 \\
qu, & \text{if } q = q_0 + 3
\end{cases}
\]

The following system subroutines have been defined for managing word queues:

- **Q.C** - The word queue whose address is given in bits 3-17 of the AC is cleared. (\( q_2 \) and \( q_3 \) are both set equal to \( q_1 \).)

- **Q.I** - The word given in the MQ is added in LIFO fashion to the word queue whose address is given in bits 3-17 of the AC. (The word to be queued is stored in the location which \( q_3 \) references, and \( q_3 \) is decremented.) A failure return is made if the queue is full before the operation is attempted.

- **Q.A** - The word given in the MQ is added in FIFO fashion to the word queue whose address is given in bits 3-17 of the AC. (\( q_2 \) is incremented and the word to be queued is stored in the location which the resulting \( q_2 \) references.) A failure return is
made if the queue is full before the operation is attempted.

Q.F* - A word is fetched from the word queue whose address is given in bits 3-17 of the AC and is returned in the AC. \((q_3)\) is incremented, and the word stored in the location which the resulting \(q_3\) references is fetched.) A failure return is made if the word queue is empty before the operation is attempted.

A word queue may be constructed by defining only the pointers \(q_0\) and \(q_1\), since, if the queue is cleared (via Q.C) before it is used, the pointers \(q_2\) and \(q_3\) will be automatically established. For example, the word queue whose address is \(Q\) may be constructed by the following two statements, where \(\epsilon\) is an expression whose value is \(n + 2\):

\[
Q \quad \text{\$DC} \quad \epsilon \\
\text{\$DS} \quad \epsilon
\]

As an example of the manipulation (but not application) of word queues, consider a task, whose entry point is TASK, which stores sequential integers on a first-in, first-out basis in the word queue FIFO until the queue is full, and then copies words from FIFO into another word queue LIFO on a last-in, first-out basis. Both FIFO and LIFO will be assumed to have a capacity of \(X\) words, where \(X\) is a predefined symbol. An algorithm for this task is given below. (T.F is described in Section 3.2.)

\[
\begin{align*}
\text{TASK} & \quad \text{LAC} \quad =\text{FIFO} \quad \text{GET ADDRESS OF FIFO QUEUE} \\
& \quad \text{JMS*} \quad =\text{Q.C} \quad \text{CLEAR FIFO QUEUE} \\
& \quad \text{LAC} \quad =\text{LIFO} \quad \text{GET ADDRESS OF LIFO QUEUE} \\
& \quad \text{JMS*} \quad =\text{Q.C} \quad \text{CLEAR LIFO QUEUE} \\
& \quad \text{DZM} \quad \text{COUNT} \quad \text{START COUNTING AT ZERO} \\
\text{LOOP1} & \quad \text{LAC} \quad =\text{FIFO} \quad \text{GET ADDRESS OF FIFO QUEUE} \\
& \quad \text{LMQ} \quad \text{SET UP PARAMETER} \\
& \quad \text{LAC} \quad =\text{FIFO} \quad \text{GET ADDRESS OF FIFO QUEUE}
\end{align*}
\]
The following system subroutines have been defined for controlling task scheduling:

**T.S** - The task whose address appears in bits 3-17 in the AC is scheduled for execution.

**T.P** - The task whose entry point is the location immediately preceding the call to **T.P** is scheduled for execution, and execution of the task which called **T.P** is terminated.

**T.F** - Execution of the task which called **T.F** is terminated.

As an example of the use of these system subroutines, consider a task, whose entry point is **SCHED**, which schedules the two tasks **TASK1** and **TASK2** after a nonzero value is stored (by some other task) in location **SWITCH**. One algorithm for this task is the following:
The call to T.P is given whenever the subroutine CHECK produces a failure return (in the same sense that some system subroutines produce failure returns) to reschedule the call to CHECK. Because tasks are scheduled on a first-in first-out basis, the rescheduled call to CHECK is not executed until each other eligible task in the task queue has been executed.

A task allocates and releases I/O devices by calling appropriate system subroutines, supplying them with "allocation masks." An allocation mask is a representation of the set of I/O devices which are involved in an allocation operation. Each bit position in the mask is associated with one I/O device. If a bit position contains a 1, the corresponding I/O device is involved in the operation; otherwise, it is not. The bit position assignments are given by the following table:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>I/O Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>201 Dataphone Input</td>
</tr>
<tr>
<td>10</td>
<td>201 Dataphone Output</td>
</tr>
<tr>
<td>11</td>
<td>Reader</td>
</tr>
<tr>
<td>12</td>
<td>Punch</td>
</tr>
<tr>
<td>13</td>
<td>Keyboard</td>
</tr>
<tr>
<td>14</td>
<td>Teleprinter</td>
</tr>
<tr>
<td>15</td>
<td>D/A Converter</td>
</tr>
<tr>
<td>16</td>
<td>Push Buttons</td>
</tr>
<tr>
<td>17</td>
<td>Display</td>
</tr>
</tbody>
</table>
The following system subroutines have been defined for controlling I/O device allocation:

T.A - The I/O devices specified by the allocation mask in bits 9-17 of the AC are allocated. The calling task is terminated, and the return from this subroutine is scheduled as a task to be executed after the specified devices become available. Bits 0-4 of the AC are ignored.

T.R - The I/O devices specified by the allocation mask in bits 9-17 of the AC are released. Bits 0-4 of the AC are ignored.

In order to guarantee that all scheduled user tasks become eligible for execution in a finite amount of time, I/O device allocation must be performed according to the following rule:

Whenever an I/O device is allocated, all other I/O devices which are to be allocated before it is released must also be allocated.

As an example of I/O device allocation, consider two tasks, which are scheduled one immediately after the other, whose I/O device allocation activity is summarized by the following tables (where $t_{i,k+1} > t_{i,k}$):

Task #1:

<table>
<thead>
<tr>
<th>Time</th>
<th>Devices Allocated</th>
<th>Devices Released</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{11}$</td>
<td>A</td>
<td>-</td>
</tr>
<tr>
<td>$t_{12}$</td>
<td>B</td>
<td>-</td>
</tr>
<tr>
<td>$t_{13}$</td>
<td>-</td>
<td>A</td>
</tr>
<tr>
<td>$t_{14}$</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>$t_{15}$</td>
<td>-</td>
<td>B,C</td>
</tr>
</tbody>
</table>

Task #2:

<table>
<thead>
<tr>
<th>Time</th>
<th>Devices Allocated</th>
<th>Devices Released</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{21}$</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>$t_{22}$</td>
<td>B</td>
<td>-</td>
</tr>
<tr>
<td>$t_{23}$</td>
<td>-</td>
<td>B,C</td>
</tr>
</tbody>
</table>
Assume the rule given above is ignored, and the I/O devices are allocated precisely as shown in the above tables. Then, if \( t_{22} > t_{12} > t_{21} \) and \( t_{14} = \infty \) and \( t_{22} = \infty \) because Task #1 will not release device B until it can allocate device C, and Task #2 will not release device C until it can allocate device B.

By applying the allocation rule to the above tables, the following new tables are obtained:

Task #1:

<table>
<thead>
<tr>
<th>Time</th>
<th>Devices Allocated</th>
<th>Devices Released</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t'_{11} )</td>
<td>A,B,C</td>
<td>-</td>
</tr>
<tr>
<td>( t'_{12} )</td>
<td>-</td>
<td>B,C</td>
</tr>
<tr>
<td>( t'_{13} )</td>
<td>B,C</td>
<td>-</td>
</tr>
<tr>
<td>( t'_{14} )</td>
<td>-</td>
<td>A,C</td>
</tr>
<tr>
<td>( t'_{15} )</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>( t'_{16} )</td>
<td>-</td>
<td>B,C</td>
</tr>
</tbody>
</table>

Task #2:

<table>
<thead>
<tr>
<th>Time</th>
<th>Devices Allocated</th>
<th>Devices Released</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t'_{21} )</td>
<td>B,C</td>
<td>-</td>
</tr>
<tr>
<td>( t'_{22} )</td>
<td>-</td>
<td>B</td>
</tr>
<tr>
<td>( t'_{23} )</td>
<td>B</td>
<td>-</td>
</tr>
<tr>
<td>( t'_{24} )</td>
<td>-</td>
<td>B,C</td>
</tr>
</tbody>
</table>

With this modification, all tasks will become eligible for execution. (A new task is scheduled and the calling task is terminated each time I/O devices are allocated.)

A subroutine which may be called by several concurrently executing tasks and which allows tasks other than the one which called it to execute before it returns is in danger of being reentered from one task while it is servicing another. This event results in the loss of the return address for the subroutine and perhaps some of the data upon which the subroutine operates. To facilitate the writing of reentrant subroutines (i.e., subroutines which are protected against reentry), the following system subroutines have been defined:
T.L - Lock subroutine against reentry. If the location which immediately follows the call to T.L does not contain zero, the call to the subroutine whose entry point immediately precedes the call to T.L is rescheduled. Otherwise, the content of the location which immediately precedes the call to T.L is copied into the location which immediately follows the call to T.L.

T.U - Unlock reentrant subroutine. The location whose address is the address contained in the word which immediately follows the call to T.U plus 2 is zeroed, and a JMP to the address which was stored in that location before it was zeroed is executed.

Because both T.L and T.U must preserve the contents of the AC and MQ, these subroutines have the following special calling sequences:

Calling sequence for T.L:

---
$DC 0
JMS* =T.L
$DC 0
---
(reentrant subroutine entry point)
(save location for T.L)
(return)

Calling sequence for T.U:

JMS* =T.U
$DC -----
(subroutine entry point)

As an example of the use of T.L and T.U, consider the reentrant subroutine WAIT which returns to its calling task after all tasks on the task queue have had a chance to execute. An algorithm for this subroutine is the following:

WAIT
$DC 0
JMS* =T.L
$DC 0
SET REENTRY LOCK
SAVE LOC FOR T.L
3.3 Format Conversions

Characters are represented internally in the System by 6-bit codes to facilitate storage of three characters per word. Since ASCII character codes must be available for teletype, paper tape, and dataphone I/O, conversions between ASCII and 6-bit codes must be frequently performed. In addition, the 11-bit sign-magnitude coordinates required by the display control's vector mode must often be converted to and from 18-bit two's complement representation. To satisfy these requirements, the following system subroutines have been defined:

C.B6 - The binary number given in the AC is converted to its corresponding 6-bit octal representation, which is returned in the AC and MQ (high-order digits in AC, low-order digits in MQ).

C.6A - The 6-bit code given in bits 12-17 of the AC is converted to the corresponding ASCII code, which is returned in bits 10-17 of the AC, with bits 0-9 cleared and the parity bit of the ASCII code (i.e., bit 10 of the AC) set, regardless of the parity. Bits 0-11 of the AC are ignored on entry.

C.A6 - The ASCII code given in bits 10-17 of the AC is converted to the corresponding 6-bit code, which is returned in bits 12-17 of the AC, with bits 0-11 cleared. Bits 0-9 of the AC and the parity bit of the ASCII code (i.e., bit 10 of the AC) are ignored on entry.
-17-

C.CB - The vector mode sign-magnitude display coordinate given in bits 7-17 of the AC is converted to the corresponding two's complement representation, which is returned in the AC. Bits 0-6 of the AC are ignored on entry.

C.BC - The two's complement number in the AC is converted modulo $2^{10}$ to the corresponding vector mode sign-magnitude display coordinate representation, which is returned in bits 7-17 of the AC with bits 0-6 cleared.

The 6-bit codes used by the System may each be represented by two octal digits as shown by the following table:

<table>
<thead>
<tr>
<th>First Octal Digit</th>
<th>Second Octal Digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 1 2 3 4 5 6 7</td>
</tr>
<tr>
<td>1</td>
<td>8 9 A B C D E F</td>
</tr>
<tr>
<td>2</td>
<td>G H I J K L M N</td>
</tr>
<tr>
<td>3</td>
<td>O P Q R S T U V</td>
</tr>
<tr>
<td>4</td>
<td>W X Y Z * / + -</td>
</tr>
<tr>
<td>5</td>
<td>( ) [ ] &lt; = &gt; +</td>
</tr>
<tr>
<td>6</td>
<td>+ - . , : ; ? ! '</td>
</tr>
<tr>
<td>7</td>
<td>$ # &amp; cr $f sp</td>
</tr>
</tbody>
</table>

|     | cr = carriage return |
|     | $f = line feed       |
|     | sp = space           |

All ASCII characters which do not appear in the table are mapped into $77_8$. The only printing characters which are treated in this manner are "®", "©", and "".

3.4 Buffered I/O

Input data from the dataphone, the paper tape reader, and the keyboard, as well as output data to the dataphone,
paper tape punch, and teleprinter, are buffered by the System. In the event that an input buffer is empty or an output buffer is full and the system subroutine which transfers data between the buffer and a task is called, the return from the subroutine is scheduled as a task to be executed only after the state of the buffer changes, and execution of the calling task is terminated.

3.4.1 Dataphone I/O

The following system subroutines have been defined for managing the 201 dataphone buffers:

B.FI* - An image is fetched from the 201 dataphone input buffer and is returned in bits 10-17 of the AC. Bits 0-9 of the AC are cleared, unless the image is an end-of-record character in which case bits 0-4 are set and bits 5-9 are cleared. A failure return is made if the data set is not connected.

B.FO* - The image in bits 10-17 of the AC is sent to the 201 dataphone output buffer. If bit 0 of the AC is set, the image is interpreted as an end-of-record character, and transmission is begun. A failure return is made before the image is buffered if the data set is not connected.

Since actual dataphone transmission is record-oriented (although transfer of data between the dataphone buffers and tasks is not), the return from B.FI to the calling task is delayed until the dataphone input buffer contains a complete record, and the return from B.FO is delayed until the last record transmitted has been affirmatively acknowledged by the other party. In simpler terms, the dataphone input buffer is considered to be empty whenever it does not contain a complete record, and the dataphone output buffer is considered to be full whenever the last transmitted record has not been affirmatively acknowledged.
Dataphone records are formatted according to the conventions adopted by The University of Michigan Computing Center at the time of this report. Each record is formatted (if transmitted) or interpreted (if received) by the System and consists of the following sections:

1. Several synchronous idle (SYN) characters (026g). (At least two are required when receiving; eight are transmitted.)
2. A data link escape (DLE) character (220g).
3. Data. The 8-bit images in this section are arbitrary binary, with the exception that a DLE character (with either parity) is preceded by a DLE. The first DLE is ignored when the record is received, and serves only to cause the second one to be interpreted as data. (A pair of characters consisting of a DLE followed by a SYN is ignored when receiving, although this sequence is never transmitted.)
4. A DLE character.
5. An end-of-record character.
6. The high-order 8 bits of the block check (described below).
7. The low-order 8 bits of the block check (described below).
8. A pad character (377g).

In order to facilitate detection of burst errors, a 16-bit cyclic block check is included in each dataphone record. For purposes of computing this block check, the data sequence (consisting of the concatenation of the second through the last data images, plus the end-of-record character) is regarded as a cyclic polynomial code. The block check is obtained by simultaneously multiplying the polynomial representation of the data sequence by \( X^{16} \) and dividing it by \( X^{16} + X^{15} + X^2 + 1 \) (where the coefficients of the polynomials are taken from the field of two elements). The following diagram illustrates this operation:
Whenever a dataphone record is received by either party, the block check is computed and compared with the received block check. If the two block checks match, the dataphone record is assumed to have been received correctly, and an affirmative acknowledgment is returned when the receiving party is ready for the next record. However, if the two block checks do not match, a negative acknowledgment, which is a request for the record to be retransmitted, is returned, and the incorrectly received record is discarded. The System assumes complete responsibility for managing acknowledgments and retransmissions for the 339.

Whenever a dataphone record is received with a correct block check, the first data image is examined. If it is zero, user tasks are given access to it via the system subroutine B.FI. Otherwise, a special 201-to-teleprinter task is scheduled to type the record (interpreting it as a sequence of ASCII codes) as soon as the teleprinter becomes available. In
this way, unsolicited messages from the remote party are typed and routed clear of tasks which are using the dataphone.

Whenever the end-of-record character for either a transmitted or received record is an enquiry (005\textsubscript{8}) or an end-of-transmission (204\textsubscript{8}), both dataphone buffers (input and output) are cleared, and the last record transmitted is considered to have been affirmatively acknowledged. Note that transmitted records of this form will be processed normally by the System (except that immediate acknowledgment will be assumed), but received records of this form will be discarded once the end-of-record character is detected.

As an example of the use of B.FI and B.FO, consider the task MIRROR which receives 64 dataphone images in an arbitrary number of records (not including the zero images required to route records to tasks), transmits all of them in one dataphone output record, and ignores the remainder of the last dataphone input record which it examined. An algorithm for this task is the following (L.T is described in Section 3.11):

```
MIRROR
LAW 600    GET ALLOCATION MASK
JMS* =T.A ALLOCATE 201 INPUT & OUTPUT
LAW 17700 LOAD AC WITH -64
DAC COUNT INITIALIZE IMAGE COUNT
START
JMS* =B.FI GET REDUNDANT IMAGE
JMP HELP DATA SET NOT CONNECTED
READ
JMS* =B.FI GET INPUT IMAGE
JMP HELP DATA SET NOT CONNECTED
SPA SKIP IF NOT END OF RECORD
JMP START READ NEXT RECORD
JMS* =B.FO PUT IN OUTPUT BUFFER
JMP HELP DATA SET NOT CONNECTED
ISZ COUNT SKIP IF RECORD LONG ENOUGH
JMP READ READ NEXT IMAGE
JMS* =B.FI GET INPUT IMAGE
JMP HELP DATA SET NOT CONNECTED
```
3.4.2 Paper Tape I/O

The following system subroutines have been defined for managing the paper tape reader and punch buffers:

**B.R** - An image is fetched from the reader buffer and returned in bits 10-17 of the AC. Bits 0-9 of the AC are cleared. Only one end-of-record character (zero) may be returned by two successive calls to B.R. A failure return is made if the reader is out of tape and the reader buffer is empty.

**B.P** - The image in bits 10-17 in the AC is sent to the punch buffer. A failure return is made if the punch is out of tape and the punch buffer is full.
Paper tape formats are arbitrary, subject to the restriction that a zero image (i.e., a line of blank tape) which immediately follows a nonzero image is interpreted as an end-of-record character and all other zero images are ignored. However, the format which is read and punched by the data transfers of the idle-time task (Section 4.1) is recommended for compatibility reasons. In this format, the two high-order bits of each 8-bit tape image are interpreted as control information, and the remaining 6 bits are interpreted as data. The two control bits are interpreted as follows:

00  mode change
01  binary origin
10  binary data
11  alphanumeric data

There are 64 possible mode changes (designated by the low-order 6 bits of a mode change tape image), only one of which has been defined at the time of this writing, i.e., the end-of-record character 000\textsubscript{8}. (An example of possible future mode change assignments is a set of relocation modes for relocatable binary records.)

A binary block consists of three binary origin images followed by a multiple of three binary data images. The block represents a set of 18-bit words to be loaded starting at the address indicated by the data bits of the three origin images. For example, the binary block which indicates that location 23572\textsubscript{8} should contain 621365\textsubscript{8} and that location 23573\textsubscript{8} should contain 176234\textsubscript{8} is the following:

```
102
135 origin 23572
172
262
213 data 621365
265
217
262 data 176234
234
```
A binary record is a concatenation of binary blocks, followed by the end-of-record character (000\textsubscript{8}).

An alphanumeric record consists of an arbitrary number of alphanumeric tape images (where the 6 data bits in each image represent a 6-bit character code), followed by an end-of-record character (000\textsubscript{8}).

As an example of the use of the paper tape I/O system subroutines, consider a task COPY which copies one record of paper tape:

COPY

<table>
<thead>
<tr>
<th>LAW</th>
<th>140</th>
<th>GET ALLOCATION MASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS*</td>
<td>T.A</td>
<td>ALLOCATE READER &amp; PUNCH</td>
</tr>
<tr>
<td>JMS*</td>
<td>B.R</td>
<td>GET IMAGE FROM READER</td>
</tr>
<tr>
<td>JMP</td>
<td>RERR</td>
<td>READER OUT OF TAPE</td>
</tr>
<tr>
<td>SNA</td>
<td><strong>+4</strong></td>
<td>SKIP IF NOT END OF RECORD</td>
</tr>
<tr>
<td>JMP</td>
<td>B.P</td>
<td>PUNCH IMAGE</td>
</tr>
<tr>
<td>JMP</td>
<td>PERR</td>
<td>PUNCH OUT OF TAPE</td>
</tr>
<tr>
<td>JMP</td>
<td>COPY+2</td>
<td>READ NEXT IMAGE</td>
</tr>
<tr>
<td>JMS*</td>
<td>B.P</td>
<td>PUNCH END OF RECORD</td>
</tr>
<tr>
<td>JMP</td>
<td>PERR</td>
<td>PUNCH OUT OF TAPE</td>
</tr>
<tr>
<td>LAW</td>
<td>140</td>
<td>GET ALLOCATION MASK</td>
</tr>
<tr>
<td>JMS*</td>
<td>T.R</td>
<td>RELEASE READER &amp; PUNCH</td>
</tr>
<tr>
<td>JMS*</td>
<td>T.F</td>
<td>TERMINATE TASK</td>
</tr>
</tbody>
</table>

RERR

<table>
<thead>
<tr>
<th>LAC</th>
<th>RERRT</th>
<th>GET ADDRESS OF TEXT LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKP</td>
<td>TYPE DIAGNOSTIC</td>
<td></td>
</tr>
</tbody>
</table>

PERR

<table>
<thead>
<tr>
<th>LAC</th>
<th>PERRT</th>
<th>GET ADDRESS OF TEXT LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC</td>
<td>TEXT</td>
<td>SAVE ADDRESS OF TEXT LIST</td>
</tr>
<tr>
<td>LAW</td>
<td>140</td>
<td>GET ALLOCATION MASK</td>
</tr>
<tr>
<td>JMS*</td>
<td>T.R</td>
<td>RELEASE READER &amp; PUNCH</td>
</tr>
<tr>
<td>LAW</td>
<td>10</td>
<td>GET ALLOCATION MASK</td>
</tr>
<tr>
<td>JMS*</td>
<td>T.A</td>
<td>ALLOCATE TELEPRINTER</td>
</tr>
<tr>
<td>LAC</td>
<td>TEXT</td>
<td>GET ADDRESS OF TEXT LIST</td>
</tr>
<tr>
<td>JMS*</td>
<td>L.T</td>
<td>TYPE DIAGNOSTIC</td>
</tr>
<tr>
<td>LAC</td>
<td>END</td>
<td>GET ADDRESS OF TEXT LIST</td>
</tr>
</tbody>
</table>
3.4.3 Teletype I/O

The following system subroutines have been defined for managing the keyboard and teleprinter buffers:

- **B.K** - A 6-bit character is fetched from the keyboard buffer and returned in bits 12-17 of the AC. Bits 0-11 of the AC are cleared.
- **B.T** - The three six-bit characters in bits 0-5, 6-11, and 12-17 of the AC are sent to the teleprinter buffer to be typed in respective order. (The null character 77₁₀ will not be typed, even as a non-printing character.)

As an example of the use of these subroutines, consider the task ENCODE which accepts characters from the keyboard and types the octal representation of the corresponding 6-bit codes. When a null character is typed, the task is terminated. An algorithm for this task is the following:

```
  ENCODE
  LAW  30   GET ALLOCATION MASK
  JMS* =T.A   ALLOCATE KEYBOARD & TELEPRINTER
  JMS* =B.K   GET CHARACTER FROM KEYBOARD
  SAD  =77    SKIP IF NOT NULL CHARACTER
  JMP   END    TERMINATE TASK
```
3.5 Nonbuffered I/O

Three devices which might appear to require buffering are not buffered: the clock, the A/D converter, and the D/A converter. The clock, which is normally used in an interactive system to check for the occurrence of certain events within specified time intervals, is often programmed in a multiprogramming system such that any task may use it at any time. This is accomplished through the use of a buffer into which entries (each consisting of a return pointer and a time interval) may be inserted at arbitrary points. Since the buffer required is considerably more complicated than those used by other devices, the cost of programming the clock in this manner was found to be excessive.

Since A/D converter data should be interpreted in real time, these data are not buffered. Instead, whenever a task calls the system subroutine to obtain data from the A/D converter, the device is selected, the return from the subroutine is scheduled as a task to be executed after the conversion is complete, and execution of the calling task is terminated.

The D/A converter requires only two microseconds to produce an output after it is selected, whereas the minimum time between selections of a particular D/A channel is four microseconds. Consequently, the System does not buffer D/A converter data.
The following system subroutines have been defined for nonbuffered I/O:

N.C - Execution of the calling task is terminated and the return from N.C is scheduled as a task to be executed at least the number of sixtieths of a second later which is the two's complement of the number given in the AC.

N.A - The channel of the A/D converter specified in bits 12-17 of the AC is selected, and the converted value, when obtained, is returned in bits 0-11 of the AC. Bits 12-17 of the AC are cleared. The returned value, if interpreted as an ordinary two's complement number, is \(-2^{17}(1+V/5)\), where \(V\) is the applied input voltage (which ranges from 0 to -10 volts).

N.D1- D/A converter channel #1 is selected. The output of the channel is set to \(-5(1+2^{-17}A)\) volts, where \(A\) is the content of the AC.

N.D2- D/A converter channel #2 is selected. The output of the channel is set to \(-5(1+2^{-17}A)\) volts, where \(A\) is the content of the AC.

N.D3- D/A converter channel #3 is selected. The output of the channel is set to \(-5(1+2^{-17}A)\) volts, where \(A\) is the content of the AC.

As an example of a use of N.C, consider the task PROMPT which types "PLEASE TYPE NOW" once about every eight seconds until the operator types something on the keyboard, and types "THANK YOU" when the operator finishes typing a line. An algorithm for this task is the following:

```
PROMPT    LAW    30  GET ALLOCATION MASK
JMS*   =T.A  ALLOCATE KEYBOARD & TELEPRINTER
DZN    DONE  INDICATE NO KEYBOARD RESPONSE
LAC    =POLITE GET ADDRESS OF KEYBOARD CHECKER
```
As an example of the use of N.A, consider the task

COMPAR which samples channels 0 and 1 of the A/D converter until the inputs on the two channels are close enough to each other that the same value is read from each channel. When this condition is satisfied, the comment "ANALOG INPUTS MATCH" is typed on the teletype. An algorithm for this task is the following:
The following system subroutines have been defined for managing the push buttons which are associated with the display control:

**P.T** - The task whose address is given in bits 3-17 of the AC is declared to be the service task for manual operation of the push buttons (i.e., this task is scheduled whenever the state of the push buttons is altered by the operator). If the AC contains zero when P.T is called, a null service task (i.e., one which calls P.E and terminates) is used.

**P.E** - Manual operation of the push buttons is enabled (i.e., the state of the push buttons may be changed by the operator).
P.D - Manual operation of the push buttons is disabled (i.e., the state of the push buttons may not be changed by the operator). A call to P.D is effected whenever the operator changes the state of the push buttons.

P.R - Push buttons 0-11 are read into bits 6-17 of the AC, and bits 0-5 of the AC are cleared.

P.S - Push buttons 0-11 are set according to bits 6-17 of the AC.

As an example of the use of these subroutines, consider the task BUTTON which enables manual operation of the push buttons and sets the button numbered one greater (modulo 12) than the number of the one pushed by the operator. The procedure is terminated and all push buttons are cleared when a keyboard character is struck. An algorithm for this task is the following:

```
BUTTON LAW 22 GET ALLOCATION MASK
        JMS* =T.A ALLOCATE KEYBOARD & PUSH BUTTONS
        LAC =SERV GET ADDRESS OF SERVICE TASK
        JMS* =P.T DECLARE SERVICE TASK
        CLA GET INITIAL PUSH BUTTON STATE
        DAC STATE SAVE FOR USE BY SERV
        JMS* =P.S SET INITIAL PUSH BUTTON STATE
        JMS* =P.E ENABLE MANUAL OPERATION
        JMS* =B.K GET KEYBOARD CHARACTER
        JMS* =P.D DISABLE MANUAL OPERATION
        CLA GET FINAL PUSH BUTTON STATE
        JMS* =P.S CLEAR PUSH BUTTONS
        CLA GET NULL SERVICE PARAMETER
        JMS* =P.T DECLARE NULL SERVICE TASK
        LAW 22 GET ALLOCATION MASK
        JMS* =T.R RELEASE KEYBOARD & PUSH BUTTONS
        JMS* =T.F TERMINATE TASK
SERV JMS* =P.R READ PUSH BUTTONS
```
3.7 Display Control Communication

The following system subroutines have been defined for communicating with the display control:

D.E - Display interrupts are enabled (i.e., a light pen flag interrupt or an internal stop interrupt will cause the System to read the display status information required for D.A, D.Y, D.X, and D.O and to schedule the appropriate service task).

D.D - Display interrupts are disabled (i.e., the System will ignore light pen flag and internal stop interrupts). A call to D.D is effected whenever a display interrupt occurs.

D.P - The task whose address is given in bits 3-17 of the AC is declared to be the service task for light pen flags. This task is scheduled whenever the light pen sees an intensified portion of the display on which the light pen is enabled (see Section 3.10), providing that display interrupts are enabled (via D.E). If the AC contains zero when D.P is called, a null service task (i.e., one which calls D.E and terminates) is used.

D.A - The address of the display on the last display interrupt is returned in bits 3-17 of the AC with bits 0-2 clear.
D.Y - The y coordinate of the display (measured relative to the center of the screen in scale xl) on the last display interrupt is returned in the AC as a two's complement number.

D.X - The x coordinate of the display (measured relative to the center of the screen in scale xl) on the last display interrupt is returned in the AC as a two's complement number.

D.O*- The address which is the operand of the push jump instruction which was the number of entries given in bits 12-17 of the AC above the last entry in the display control's push-down list or the last display interrupt is returned in bits 3-17 of the AC with bits 0-2 clear. (A more meaningful interpretation of this subroutine may be obtained from the examples in Section 3.10.) A failure return is made if the indicated push jump instruction does not exist.

The external stop interrupt and the edge flag interrupt are not used. The function of the external stop interrupt may be performed via an unconditional internal stop interrupt (via S.LU, which is described in Section 3.10). Since the virtual display area established by the System is 75 inches by 75 inches, the edge flags, if used, would occur on the left and lower edges of the screen, but not on the upper or right edges. Because of this inconsistency, the edge flags are not used.

3.8 Light Pen Tracking

A light pen tracking algorithm is supplied with the System to enable user tasks to follow the motion of the light pen. This algorithm has been empirically determined to track the light pen at any attainable speed, and it is insensitive to changes in direction because it does not involve prediction.
The tracking algorithm may be described with the aid of the following diagram:

When the display for the tracking algorithm is begun, strokes 1 and 2 are drawn. (Strokes 1 and 2 are actually coincident.) The x coordinate of the first light pen hit on each stroke is recorded. If both x coordinates are obtained, a new x coordinate for the tracking cross is computed as their average. Strokes 3 and 4 are then drawn, and a new y coordinate for the tracking cross is computed in similar manner if both y coordinates are obtained.
If any one of the four coordinates required to compute a new position of the tracking cross is not obtained, a search pattern consisting of concentric squares 5 through 12 is drawn. When a light pen hit is detected on any one of these squares, the search pattern is terminated, and the tracking cross is placed at the coordinates of the hit. If square 12 is completed and no light pen hit is detected, the tracking process is terminated.

Whenever the tracking cross is positioned via the search pattern, rather than by averaging coordinates, the tracking display is immediately repeated. The remainder of the active display structure (Section 3.9) is not displayed until the tracking cross can be positioned by averaging coordinates. In this way, the tracking display is given priority over all other displays whenever the light pen is being moved rapidly and tracking is in process.

The following system subroutines have been defined for controlling the tracking process:

**X.I** - The tracking cross is placed at the y coordinate given in the AC and the x coordinate given in the MQ, and the tracking process is begun. The coordinates, which are given as two's complement numbers, are interpreted modulo $2^{10}$ measured in scale xL relative to the center of the screen.

**X.R** - The tracking process is resumed with the tracking cross at the coordinates where tracking was last terminated (by X.T or by completion of square 12).

**X.T** - The tracking process is terminated. (The tracking cross is removed from the screen.)

**X.S** - A failure return is made if tracking is in process.

**X.Y** - The y tracking coordinate is returned in the AC as a two's complement number measured in scale xL relative to the center of the screen. If tracking
is not in process, the y coordinate where tracking was last terminated is returned.

X.X - The x tracking coordinate is returned in the AC as a two's complement number measured in scale xl relative to the center of the screen. If tracking is not in process, the x coordinate where tracking was last terminated is returned.

The tracking algorithm is independent of D.E and D.D.

3.9 Display Structure Topology

Each entity to be displayed is represented in the display structure provided by the System as a position in the hierarchy of the entities which constitute the picture. Each position in the hierarchy is implemented as a display subroutine which is called a level. A level which is being executed by the display control at least once on every frame is called an active level. One particular level, which is always active and is an integral part of the system, represents the 75 inch by 75 inch virtual display area of the display control and is called the highest active level.

A display subroutine which is not itself a level and which contains no calls to levels is called a leaf. All of the drawing of visible portions of the picture is accomplished by leaves. A leaf is subject to the restriction that the state of the display (coordinates, light pen status, scale, intensity, blink status, light pen sense indicator) must be the same when the subroutine returns as when it is entered. Consequently, because the display control's POP instruction does not restore coordinates, the only data modes which are useful in leaves are vector mode, short vector mode, and increment mode.

The set L of all levels and leaves (both active and non-active) is partially ordered, i.e., there exists a relation "<" defined on L such that
The semantic interpretation of the expression $x \leq y$ is that any modification of the entity represented by the level $x$ (or in the drawing produced by the leaf $x$, if $x$ is a leaf) will effect a corresponding modification in the entity represented by the level $y$. When $x \leq y$, the level $y$ is said to own the level or leaf $x$. An attribute of a level $y$ is a level or leaf $x$ such that $x \leq y$ and there does not exist a level $z$ different from $x$ and $y$ such that $x \leq z$ and $z \leq y$.

As an example of this interpretation of the relation "\leq", consider a triangle which is to be represented internally as a set of three lines:

```
   c
  /|
 / \
 b
  |
 a
```

A display structure for this triangle may be represented by the following diagram. (In the diagram, $x \leq y$ is represented by a line joining $x$ and $y$ such that $y$ appears above $x$ in the diagram.)

```
triangle
/  \
/    \line a  line b  line c
triangle
```

Note from the diagram that the triangle owns each of its sides (lines $a$, $b$, and $c$). If line $b$ is now deleted, the display structure assumes the following form:

```
triangle
/  \
/    \line a  line c
triangle
```
The triangle is obviously modified by this operation (in fact, it is no longer a triangle). However, the fact that the triangle has been modified does not imply that all of its attributes have been modified. In this example, lines a and c remain unchanged.

The set $X$ of all active levels and the leaves which they own is also partially ordered, since $X \subseteq L$ and $L$ is partially ordered. Because the highest active level represents the virtual display area of the display control, it owns every element of $X$. Consequently, if the operator "+" is defined by the conditions

1. $\forall x, y \in X \quad x + y \in X$
2. $\forall x, y \in X \quad x \leq x + y$ and $y \leq x + y$
3. $\forall x, y, z \in X \quad x \leq z$ and $y \leq z \implies x + y \leq z$,

the pair $(X, +)$ is a semilattice. The semantic interpretation of the expression $x + y$ is that $x + y$ is a level which represents the most primitive entity which owns both of the entities represented by the levels $x$ and $y$.

As an example of the interpretation of the operator "+", consider the following drawing of one exterior wall of a house:

For purposes of illustration, assume that all three windows in the picture are identical, each instance of each entity in the drawing is represented by a separate level, and the drawing shown is the only one being displayed. The display structure, then, assumes the following form:
Assume that a task which records two references to the picture with the light pen is being executed, and that the most primitive entity which owns both items referenced is to be deleted. Clearly, the portion of the structure which should be removed consists of everything which $x+y$ owns, where $x$ and $y$ are the two levels which represent the entities referenced with the light pen. For example, if the door perimeter and a window in the wall of the house are referenced, the entire wall of the house is deleted, but if the door perimeter and the window in the door are referenced, only the door is deleted.

A level is implemented as the data structure shown by the following diagram (all numbers are octal):
The following system subroutines have been defined for managing the display structure topology. (Examples of their use are given in Section 3.10.)

S.TL*- A level is created and its address (i.e., the address of the first location in its head) is returned in bits 3-17 of the AC with bits 0-2 clear. A failure return is made if the level cannot be created because of insufficient free display storage.

S.TD*- The non-active level whose address is given in bits 3-17 of the AC is destroyed. A failure return is made if the level has attributes.

S.TI*- The level or leaf whose address is given in bits 3-17 of the MQ is inserted into (i.e., made an attribute of) the level whose address is given in bits 3-17 of the AC. The created node is inserted immediately after the head.
in the level data structure. A failure return is made if the required node cannot be created because of insufficient free display storage.

S.TR* - The attribute whose address is given in bits 3-17 of the MQ is removed from the level whose address is given in bits 3-17 of the AC. This subroutine does not return until the display control has completed the current frame. (Tasks other than the calling task are executed during this delay.) A failure return is made if the specified attribute is not found in the specified level.

3.10 **Level Modification**

The following system subroutines have been defined for modifying existing levels:

- **S.LH** - The address of the highest active level is returned in bits 3-17 of the AC with bits 0-2 clear.

- **S.LY** - The y coordinate of the level whose address is given in bits 3-17 of the AC is set to the value given in the MQ. The given coordinate is interpreted as a two's complement number in the scale of the specified level, measured relative to the y coordinate of each level of which the specified level is an attribute. This subroutine has no effect on the highest active level, where the coordinates are at the center of the screen.

- **S.LX** - The x coordinate of the level whose address is given in bits 3-17 of the AC is set to the value given in the MQ. The given coordinate is interpreted as a two's complement number in
the scale of the specified level, measured relative to the x coordinate of each level of which the specified level is an attribute. This subroutine has no effect on the highest active level, where the coordinates are at the center of the screen.

S.LP - The scale, intensity, and light pen status are set on the level whose address is given in bits 3-17 of the AC according to bits 9-17 of the MQ. The content of the MQ is interpreted as follows:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>set scale according to bits 10-11</td>
</tr>
<tr>
<td>10-11</td>
<td>n, where scale is $x^{2^n}$</td>
</tr>
<tr>
<td>12</td>
<td>set light pen status according to bit 13</td>
</tr>
<tr>
<td>13</td>
<td>light pen status (1 = enabled, 0 = disabled)</td>
</tr>
<tr>
<td>14</td>
<td>set intensity according to bits 15-17</td>
</tr>
<tr>
<td>15-17</td>
<td>intensity value</td>
</tr>
</tbody>
</table>

This subroutine has no effect on the highest active level, where the scale is $x^8$, the intensity is 7, and the light pen is disabled.

S.LBE- The displays generated by calls (either direct or indirect) to leaves from the level whose address is given in bits 3-17 of the AC are caused to blink with a 0.5-second period. Because the 339 POP instruction does not restore the blink status, care must be taken to insure that this blink is not simultaneously effective on any level of which the given level is an owner. This subroutine has no effect on the highest active level, where blink is disabled.

S.LBD- Blinking of the level whose address is given in bits 3-17 of the AC is disabled (i.e., the effect of a call to S.LBE is removed).
S.LC - The scale and/or intensity is counted up or down one unit on the level whose address is given in bits 3-17 of the AC according to bits 12-15 of the MQ, which are interpreted as follows:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Count scale according to bit 13</td>
</tr>
<tr>
<td>13</td>
<td>1 = multiply scale by 2, 0 = divide scale by 2</td>
</tr>
<tr>
<td>14</td>
<td>Count intensity according to bit 15</td>
</tr>
<tr>
<td>15</td>
<td>1 = increment intensity by unity, 0 = decrement intensity by unity.</td>
</tr>
</tbody>
</table>

This subroutine has no effect on the highest active level.

S.LU - An unconditional scheduling of the task whose address is given in bits 3-17 of the MQ is effected whenever display interrupts are enabled (via D.E) and the tail of the level whose address is given in bits 3-17 of the AC is executed. This subroutine has no effect on the highest active level.

S.LS - The task whose address is given in bits 3-17 of the MQ is scheduled whenever display interrupts are enabled (via D.E), the tail of the level whose address is given in bits 3-17 of the AC is executed, and the coordinates of that level are on the screen. This subroutine has no effect on the highest active level.

S.LL - The task whose address is given in bits 3-17 of the MQ is scheduled whenever display interrupts are enabled (via D.E), the tail of the level whose address is given in bits 3-17 of the AC is executed, and the light pen sense indicator has been set during execution of that level. This subroutine has no effect on the highest active level.
The effect of S.LU, S.LS, or S.LL is removed from the level whose address is given in bits 3-17 of the AC.

Whenever the scale, light pen status, intensity, blink status, or coordinates are not set on a level, the quantities which are not set on that level are the same as those on the level of which it is an attribute.

Some user subroutines which call these system subroutines, as well as those in Section 3.9, are given below. LVL generates a level, inserts a specified attribute into it, sets the x and y coordinates and display parameters on the generated level, and inserts the generated level into a specified owner level. BUTN calls on LVL, and then establishes a task to be scheduled whenever the light pen sense indicator is set while the display control is executing the generated level. BUTX generates a text leaf from a specified text list, and then calls on BUTN, using the generated text leaf as the attribute parameter. CHEW (which calls on ATTR to find the first attribute of a level) destroys a given display structure, and salvages all storage from the destroyed levels and text leaves. The display structure on which CHEW operates must satisfy two conditions:

1. It must assume the form of a semilattice.
2. The maximum element of the display structure must not be owned by any level (other than itself, if it itself is a level). (L.D and L.L are described in Section 3.11.)

*CALLING SEQUENCE:
* JMS LVL
* $DC ---- (LOC CONTAINING POINTER TO OWNER)
* $DC ---- (X COORDINATE)
* $DC ---- (Y COORDINATE)
* $DC ---- (DISPLAY PARAMETER)
* ---- (RETURN IF DISPLAY STORAGE EXCEEDED)
* ---- (RETURN)
AC CONTENT ON ENTRY:
* POINTER TO ATTRIBUTE

AC CONTENT ON RETURN:
* POINTER TO CREATED LEVEL

LVL
$DC 0
JMS* =T.L  SET REENTRY LOCK
$DC 0
DAC LVL4  SAVE POINTER TO ATTRIBUTE
JMS* =S.TL  CREATE A LEVEL
JMP LVL3  DISPLAY STORAGE EXCEEDED
DAC LVL5  SAVE POINTER TO LEVEL
LAC LVL4  GET POINTER TO ATTRIBUTE
LMQ  SET UP PARAMETER
LAC LVL5  GET POINTER TO LEVEL
JMS* =S.TI  INSERT ATTRIBUTE
JMP LVL2  DISPLAY STORAGE EXCEEDED
LAC* LVL+2  GET FIRST PARAMETER
DAC LVL4  SAVE FIRST PARAMETER
ISZ LVL+2  ADVANCE TO NEXT PARAMETER
LAC* LVL+2  GET Y COORDINATE
LMQ  SET UP PARAMETER
LAC LVL5  GET POINTER TO LEVEL
JMS* =S.LY  SET Y COORDINATE
ISZ LVL+2  ADVANCE TO NEXT PARAMETER
LAC* LVL+2  GET X COORDINATE
LMQ  SET UP PARAMETER
LAC LVL5  GET POINTER TO LEVEL
JMS* =S.LX  SET X COORDINATE
ISZ LVL+2  ADVANCE TO NEXT PARAMETER
LAC* LVL+2  GET DISPLAY PARAMETER
LMQ  SET UP PARAMETER
LAC LVL5  GET POINTER TO LEVEL
JMS* =S.LP  SET DISPLAY PARAMETER
LAC LVL5  GET POINTER TO LEVEL
LMQ  SET UP PARAMETER
CALLING SEQUENCE:

* JMS BUTN  
* $DC ----  (LOC CONTAINING POINTER TO OWNER)  
* $DC ----  (Y COORDINATE)  
* $DC ----  (X COORDINATE)  
* $DC ----  (DISPLAY PARAMETER)  
* $DC ----  (SERVICE TASK ADDRESS)  
* ----  (RETURN IF DISPLAY STORAGE EXCEEDED)  
* ----  (RETURN IF SUCCESSFUL)

*AC CONTENT ON ENTRY:
POINTER TO STRUCTURE FOR BUTTON DISPLAY

*AC CONTENT ON RETURN:

*POINTER TO LIGHT BUTTON LEVEL

BUTN $DC 0
JMS* =T.L
$DC 0
DAC BUTN3
LAW -4
DAC BUTN4
LAC =BUTN1
DAC BUTN5
LAC* BUTN2
DAC* BUTN+2
ISZ BUTN+2
ISZ BUTN5
ISZ BUTN4
JMP *-5
LAC BUTN3
JMS LVL
BUTN1 $DC 0
$DC 0
$DC 0
$DC 0
JMP BUTN2
DAC BUTN3
LAC* BUTN+2
LMQ
LAC BUTN3
JMS* =S.LL
LAC BUTN3
ISZ BUTN+2
BUTN2 ISZ BUTN+2
JMS* =T.U
$DC BUTN
*CALLING SEQUENCE:

* JMS  BUTX
* $DC  ----  (ADDRESS OF TEXT LIST)
* $DC  ----  (LOC CONTAINING POINTER TO OWNER)
* $DC  ----  (Y COORDINATE)
* $DC  ----  (X COORDINATE)
* $DC  ----  (DISPLAY PARAMETER)
* $DC  ----  (SERVICE TASK ADDRESS)
* ----  (RETURN IF DISPLAY STORAGE EXCEEDED)
* ----  (RETURN IF SUCCESSFUL)

*AC CONTENT ON RETURN:

* POINTER TO LIGHT BUTTON LEVEL

BUTX $DC  0
    JMS* =T.L  SET REENTRY LOCK
    $DC  0
    LAC* BUTX+2  GET ADDRESS OF TEXT LIST
    JMS* =L.D  CREATE TEXT LEAF
    JMP BUTX4  DISPLAY STORAGE EXCEEDED
    DAC BUTX7  SAVE POINTER TEXT LEAF
    LAW -6  LOAD AC WITH -6
    DAC BUTX5  SET PARAMETER COUNTER
    LAC =BUTX2  GET ADDRESS OF BUTN CALL
    DAC BUTX6  SET PARAMETER POINTER
    BUTX1  ISZ BUTX+2  ADVANCE TO NEXT PARAMETER
     ISZ BUTX6  INCREMENT PARAMETER POINTER
     ISZ BUTX5  SKIP IF NOT PARAMETER
     SKP
    JMP BUTX2-1  CALL BUTN
    LAC* BUTX+2  GET PARAMETER
    DAC* BUTX6  STORE PARAMETER
    JMP BUTX1  MOVE NEXT PARAMETER
    LAC BUTX7  GET POINTER TO TEXT LEAF

BUTX2  JMS  BUTN  CREATE LIGHT BUTTON
    $DC  0  LOC CONTAINING POINTER TO OWNER
    $DC  0  Y COORDINATE
$DC 0 X COORDINATE
$DC 0 DISPLAY PARAMETER
$DC 0 SERVICE TASK ADDRESS
JMP BUTX3+2 DISPLAY STORAGE EXCEEDED
ISZ BUTX+2 INDICATE SUCCESS
BUTX3 JMS* =T.U UNLOCK BUTX & RETURN
$DC BUTX
LAC BUTX7 GET POINTER TO TEXT LEAF
JMS* =S.LL DESTROY TEXT LEAF
JMP BUTX3 RETURN
BUTX4 LAC BUTX+2 GET RETURN POINTER
TAD =6 ADVANCE PAST PARAMETER LIST
DAC BUTX+2 SET FAILURE RETURN POINTER
JMP BUTX3 RETURN

**CALLING SEQUENCE:**
* JMS CHEW
* ----- (RETURN)

**AC CONTENT ON ENTRY:**
* POINTER TO MAXIMUM ELEMENT IN THE STRUCTURE

**TO BE CHEWED**

**THE MAXIMUM ELEMENT SPECIFIED MUST OWN ALL LEVELS**

**WHICH OWN ELEMENTS OF THE STRUCTURE.**

<table>
<thead>
<tr>
<th>CHEW</th>
<th>$DC 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS*</td>
<td>=T.L</td>
</tr>
<tr>
<td>DAC</td>
<td>CHEW6</td>
</tr>
<tr>
<td>LAC</td>
<td>CHEW4</td>
</tr>
<tr>
<td>SNA</td>
<td>CHEW5</td>
</tr>
<tr>
<td>JMP</td>
<td>CHEW6</td>
</tr>
<tr>
<td>JMP</td>
<td>CHEW4</td>
</tr>
<tr>
<td>SAD</td>
<td>=762010</td>
</tr>
<tr>
<td>SAD</td>
<td>=70000</td>
</tr>
<tr>
<td>SAD</td>
<td>=10000</td>
</tr>
</tbody>
</table>
SKP
JMP CHEW5
CHEW2
LAC CHEW6
JMS ATTR
JMP CHEW3
DAC CHEW7
LMQ
LAC CHEW6
JMS* =S.TR
$DC 0
LAC CHEW7
LMQ
LAC =CHEWQ
JMS* =Q.A
$DC 0
JMP CHEW2
CHEW3
LAC CHEW6
JMS* =S.TD
$DC 0
JMP CHEW5
CHEW4
LAC CHEW6
JMS* =L.L
CHEW5
LAC =CHEWQ
JMS* =Q.F
JMP **+3
DAC CHEW6
JMP CHEW1
JMS* =T.U
$DC CHEW
$DC *+200
$DC 200

*CALLING SEQUENCE:
* JMS ATTR
* -----
* -----

(RETURN IF NO MORE ATTRIBUTES)
(RETURN IF ATTRIBUTE FOUND)
*AC CONTENTS ON ENTRY:

ADDRESS OF LEVEL

** ATTR $DC 0

TAD = 7 FORM POINTER TO POINTER TO NODE

DAC ATTR2 SAVE POINTER TO POINTER TO NODE

LAC* ATTR2 GET POINTER TO NODE (OR TAIL)

DAC ATTR2 SAVE POINTER TO NODE (OR TAIL)

LAC* ATTR? GET FIRST WORD FROM NODE (OR TAIL)

AND =777770 TRUNCATE BREAK FIELD

SAD =762010 SKIP IF NOT NODE

SKP NODE FOUND

JMP* ATTR NO MORE ATTRIBUTES

ISZ ATTR2 FORM POINTER TO SECOND LOC IN NODE

LAC* ATTR2 GET POINTER TO ATTRIBUTE

ISZ ATTR INDICATE SUCCESS

JMP* ATTR RETURN

As an example of how these subroutines might be used, consider a task called SELGI which allows the operator to draw unrelated straight lines on the display with the light pen. More specifically, when the task is begun, it allocates the display and displays the following:

---

**SELGI** title (insensitive to light pen)

**DRAW** ERASE ESCAPE

light buttons

---

**threshold** (imaginary line)
The elements of this display are arranged in the following structure:

```
  highest active level
       
  SELGI display level
       
  line level
    draw level erase level escape level title leaf

  draw leaf erase leaf escape leaf
```

The SELGI display level is set to scale x2, each light button level is sensitized to the light pen sense indicator, and the line level (into which all lines drawn by the operator will be inserted) has coordinates at the center of the screen.

When the light pen is pointed at the DRAW light button, the task DRAW is scheduled. The task DRAW then starts tracking on the DRAW light button, and waits (through the use of T.P) until the operator loses tracking. Then, if the Y tracking coordinate is above the threshold line, a line of length one point (which appears as a point on the display) is inserted into the line level such that it appears at the coordinates where tracking was lost. Otherwise, no line is generated. (The DRAW light button blinks while tracking is in process for this operation.) Up to 64 lines may be created in this manner.

If the light pen is now pointed at any of the unit-length lines (points) on the screen, tracking is started, and one end of the line is affixed to the tracking cross. The line
then may be stretched by moving the affixed end point to some other position on the screen. If the light pen is now pointed at any line which is longer than one point, tracking is started, and the end point of the line which is closer to the tracking cross is affixed to the tracking cross and may be moved to any position on the screen.

If the light pen is pointed at the ERASE light button, this light button starts blinking. If, while the ERASE light button is blinking, the light pen is pointed at some line on the screen, the line is removed from the line level, the storage which it occupied is salvaged, and the blinking of the ERASE light button is stopped.

If the light pen is pointed at the ESCAPE light button, the entire display structure created by SELGI is destroyed via the subroutine CHEW. The task SELGI then releases the display and terminates.

Lines are represented internally in this program by leaves which have the following format:

```
VEC     ENTER VECTOR MODE
----    Y DISPLACEMENT (NONINTENSIFIED)
----    X DISPLACEMENT (NO ESCAPE)
----    Y DISPLACEMENT (INTENSIFIED)
----    X DISPLACEMENT (NO ESCAPE)
----    Y DISPLACEMENT (NONINTENSIFIED)
----    X DISPLACEMENT (ESCAPE)
POP     END OF LEAF
```

Each leaf actually represents a triangle with two nonintensified sides. This scheme permits the end points of the line to occur anywhere on the screen:
-53-

SELGI LAW 1 GET DISPLAY ALLOCATION MASK
JMS* =T.A ALLOCATE DISPLAY
LAC =LINES GET ADDRESS OF LINE STORAGE AREA
DAC DIS SET STORAGE POINTER
LAW -1000 LOAD AC WITH -512
DAC FRM SET STORAGE COUNTER
DZM* DIS CLEAR STORAGE LOCATION
ISZ DIS INCREMENT STORAGE POINTER
ISZ FRM SKIP IF STORAGE AREA CLEARED
JMP *-3 CLEAR NEXT STORAGE LOCATION
JMS* =S.LH GET ADDRESS OF HIGHEST ACTIVE LEVEL
DAC HAL SAVE ADDRESS OF HIGHEST ACTIVE LEVEL
LAC =TXT GET ADDRESS OF TITLE TEXT LIST
JMS* =L.D CREATE TEXT LEAF
JMP END DISPLAY STORAGE EXCEEDED
DAC DIS SAVE POINTER TO TITLE LEAF
JMS LVL GENERATE SELGI DISPLAY LEVEL
$DC HAL pointer to highest active level
$DC 360 Y COORDINATE
$DC -34 X COORDINATE
$DC 500 SCALE X2
JMP FAIL DISPLAY STORAGE EXCEEDED
DAC FRM SAVE POINTER TO SELGI DISPLAY LEVEL
JMS BUTX GENERATE DRAW LIGHT BUTTON
$DC TXT1 DRAW TEXT LIST
$DC FRM POINTER TO SELGI DISPLAY LEVEL
$DC -750 Y COORDINATE
$DC -344 X COORDINATE
$DC 0 NULL DISPLAY PARAMETER
$DC DRAW DRAW SERVICE TASK
JMP END DISPLAY STORAGE EXCEEDED
JMS BUTX GENERATE ERASE LIGHT BUTTON
$DC TXT2 ERASE TEXT LIST
$DC FRM POINTER TO SELGI DISPLAY LEVEL
$DC -750 Y COORDINATE
$DC 10 X COORDINATE
$DC 0 NULL DISPLAY PARAMETER
$DC ERASE ERASE SERVICE TASK
JMP END DISPLAY STORAGE EXCEEDED
JMS BUTX GENERATE ESCAPE LIGHT BUTTON
$DC TXT3 ESCAPE TEXT LIST
$DC FR1 POINTER TO SEIGI DISPLAY LEVEL
$DC -750 Y COORDINATE
$DC 354 X COORDINATE
$DC 0 NULL DISPLAY PARAMETER
$DC ESCAPE ESCAPE SERVICE TASK
JMP END DISPLAY STORAGE EXCEEDED
JMS* =S.TL CREATE LINE LEVEL
JMP END DISPLAY STORAGE EXCEEDED
DAC DIS SAVE POINTER TO LINE LEVEL
LMQ SET UP PARAMETER
LAC FRM GET POINTER TO SEIGI DISPLAY LEVEL
JMS* =S.TI INSERT LINE LEVEL
JMP FAIL DISPLAY STORAGE EXCEEDED
LAW 60 GET LIGHT PEN ON PARAMETER
LMQ SET UP PARAMETER
LAC LIS GET POINTER TO LINE LEVEL
JMS* =S.LP ENABLE LIGHT PEN ON LINE LEVEL
LAW -360 GET Y COORDINATE
LMQ SET UP PARAMETER
LAC DIS GET POINTER TO LINE LEVEL
JMS* =S.LY SET Y COORDINATE OF LINE LEVEL
LAC -34 GET X COORDINATE
LMQ SET UP PARAMETER
LAC DIS GET POINTER TO LINE LEVEL
JMS* =S.LX SET X COORDINATE OF LINE LEVEL
LAC =MOVE GET ADDRESS OF LINE MOVING TASK
JMS* =D.P SET LIGHT PEN FLAG SERVICE
JMS* =D.E ENABLE DISPLAY INTERRUPTS
D2M ESCAPE+1 CLEAR ESCAPE SWITCH
GET ESCAPE SWITCH
SKIP IF ESCAPE NOT PENDING
TERMINATE SELGI
PREPARE TO SCHEDULE NEXT LOCATION
CHECK ESCAPE SWITCH
SCHEDULE PREVIOUS LOCATION
GET POINTER TO NONACTIVE STRUCTURE
DESTROY NONACTIVE STRUCTURE
GET POINTER TO HIGHEST ACTIVE LEVEL
DESTROY ACTIVE STRUCTURE
GET NULL LIGHT PEN FLAG SERVICE
SET NULL LIGHT PEN SERVICE
GET DISPLAY ALLOCATION MASK
RELEASE DISPLAY
TERMINATE
GET INITIAL X TRACKING COORDINATE
SET UP PARAMETER
GET INITIAL Y TRACKING COORDINATE
INITIALIZE TRACKING
PREPARE TO READ OWNER 0 LEVELS BACK
READ ADDRESS OF DRAW LEVEL
PROGRAMMING ERROR IF D.O FAILS
ENABLE BLINK ON DRAW LIGHT BUTTON
SKIP IF TRACKING HAS BEEN LOST
CHECK TRACKING AGAIN
READ Y TRACKING COORDINATE
FORM THRESHOLD CHECK
SKIP IF LINE IS TO BE CREATED
IGNORE ATTEMPT TO CREATE LINE
GET POINTER TO LINE STORAGE
SET STORAGE POINTER
GET MAXIMUM LINE COUNT
SET LINE COUNTER
GET FIRST WORD OF LINE BLOCK
SKIP IF LINE BLOCK IN USE
JMP  *+7  LINE BLOCK IS AVAILABLE
LAC  FRM  GET STORAGE POINTER
TAD  =10  FORM ADDRESS OF NEXT LINE BLOCK
DAC  FRM  SET STORAGE POINTER TO NEXT BLOCK
ISZ  CNT  SKIP IF NO MORE LINE STORAGE
JMP  *-7  CHECK AVAILABILITY OF LINE BLOCK
JMP  DRAW1  IGNORE ATTEMPT TO CREATE LINE
IAW  1121  GETVEC INSTRUCTION
DAC*  FRM  STORE IN FIRST LOCATION OF LINE BLOCK
LAC  FRM  GET POINTER TO LINE BLOCK
TAD  =7  FORM POINTER TO LAST WORD IN BLOCK
DAC  CNT  SAVE POINTER TO LAST WORD IN BLOCK
LAW  3000  GET POP INSTRUCTION
DAC*  CNT  STORE IN LAST WORD IN BLOCK
LAC  FRM  GET POINTER TO LINE BLOCK
JMS  FIXBGN  SET 1ST END POINT TO TRACKING COORD
LAC  FRM  GET POINTER TO LINE BLOCK
JMS  FIXEND  SET 2ND END POINT TO TRACKING COORD
LAC  FRM  GET POINTER TO LINE BLOCK
LMQ  SET UP PARAMETER
LAC  DIS  GET POINTER TO LINE LEVEL
JMS*  =S.TI  INSERT LINE BLOCK
NOP  DISPLAY STORAGE EXCEEDED
DRAW1  CLA  PREPARE TO READ OWNER 0 LEVELS BACK
JMS*  =D.O  READ ADDRESS OF DRAW LEVEL
SDC  0  PROGRAMMING ERROR IF D.O FAILS
JMS*  =S.LBD  STOP BLINK OF DRAW LIGHT BUTTON
JMS*  =D.E  ENABLE DISPLAY INTERRUPTS
JMS*  =T.F  TERMINATE
MOVE  JMS*  =D.Y  READ Y DISPLAY COORDINATE
DAC  MOVEY  SAVE Y DISPLAY COORDINATE
JMS*  =D.X  READ X DISPLAY COORDINATE
LMQ  SET UP PARAMETER
LAC  MOVEY  GET Y DISPLAY COORDINATE
JMS*  =X.I  INITIALIZE TRACKING
-57-

CLA
JMS* =D.O
$DC 0
DAC MOVE1
DAC MOVE2
ISZ MOVE2
TAD =5
DAC MOVE3
LAC* MOVE2
XOR =2000
JMS* =C.CB
LLSS 1
TAD MOVEY
GSM
DAC MOVE4
LAC* MOVE3
JMS* =C.CB
LLSS 1
TAD MOVEY
GSM
CMA
TAD MOVE4
SMA
JMP *+3
JMS WATCH
JMS FIXBGN
JMS WATCH
JMS FIXEND

WATCH
$DC 0
LAC MOVE1
XCT* WATCH
JMS* =X.S
JMP *+6
LAW -40
JMS* =N.C

PREPARE TO READ OWNER 0 LEVELS BACK
READ ADDRESS OF LINE LEAF
PROGRAMMING ERROR IF D.O FAILS
SAVE POINTER TO LINE LEAF
SAVE CO-Y OF POINTER TO LINE LEAF
FORM POINTER TO FIRST Y DISPLACEMENT
FORM POINTER TO THIRD Y DISPLACEMENT
SAVE POINTER TO THIRD Y DISPLACEMENT
GET FIRST Y DISPLACEMENT
INVERT SIGN BIT
CONVERT TO TWO'S COMPLEMENT
MULTIPLY BY 2
ADD Y DISPLAY COORDINATE
FORM ABSOLUTE VALUE
SAVE FOR LATER COMPARISON
GET THIRD Y DISPLACEMENT
CONVERT TO TWO'S COMPLEMENT
MULTIPLY BY 2
ADD Y DISPLAY COORDINATE
FORM ABSOLUTE VALUE
FORM NEGATIVE OF ABSOLUTE VALUE
ADD DISPLACEMENT FROM OTHER END
SKIP IF CLOSER TO FIRST Y DISPLACEMENT
CLOSER TO SECOND Y DISPLACEMENT
ENTER UPDATING TASK
PARAMETER FOR UPDATING TASK
ENTER UPDATING TASK
PARAMETER FOR UPDATING TASK
GET POINTER TO LINE LEAF
UPDATE AFFIXED END POINT
SKIP IF TRACKING NOT IN PROCESS
SCHEDULE NEXT UPDATING
LOAD AC WITH -32
WAIT ABOUT HALF A SECOND
JMS* = D.E  ENABLE DISPLAY INTERRUPTS
JMS* = T.F  TERMINATE
JMP  WATCH+1  UPDATE END POINT
JMS* = T.P  SCHEDULE PREVIOUS LOCATION

**ERASE**

LAC  = DELETE  GET ADDRESS OF LINE DELETE TASK
JMS* = D.P  SET LIGHT PEN FLAG SERVICE
CLA  PREPARE TO READ OWNER 0 LEVELS BACK
JMS* = D.O  GET POINTER TO ERASE LEVEL
$DC  0  PROGRAMMING ERROR IF D.O FAILS
DAC  ERS  SAVE POINTER TO ERASE LEVEL
JMS* = S.LBE  START BLINKING ERASE LIGHT BUTTON
JMS* = D.E  ENABLE DISPLAY INTERRUPTS
JMS* = T.F  TERMINATE

**DELETE**

LAC  ERS  GET POINTER TO ERASE LEVEL
JMS* = S.LBD  STOP BLINKING ERASE LIGHT BUTTON
CLA  PREPARE TO READ OWNER 0 LEVELS BACK
JMS* = D.O  GET POINTER TO LINE LEAF
$DC  0  PROGRAMMING ERROR IF D.O FAILS
DAC  FRM  SAVE POINTER TO LINE LEAF
LMQ  SET UP PARAMETER
LAC  DIS  GET POINTER TO LINE LEVEL
JMS* = S.TR  REMOVE LINE LEAF
$DC  0  PROGRAMMING ERROR IF S.TR FAILS
DZM*  FRM  DESTROY LINE LEAF
LAC  = MOVE  GET ADDRESS OF LINE MOVING TASK
JMS* = D.P  SET LIGHT PEN SERVICE
LAW  -40  LOAD AC WITH -32
JMS* = N.C  WAIT ABOUT HALF A SECOND
JMS* = D.E  ENABLE DISPLAY INTERRUPTS
JMS* = T.F  TERMINATE

**ESCAPE**

JMS  *+1  SET ESCAPE SWITCH
$DC  0  ESCAPE SWITCH
JMS* = T.F  TERMINATE

**FIXBGN**

$DC  0
JMS  FIXRD  SET UP POINTERS FOR FIXING LEAF
GET Y TRACKING COORDINATE
SET FIRST Y DISPLACEMENT
GET X TRACKING COORDINATE
SET FIRST X DISPLACEMENT
CORRECT INTENSIFIED VECTOR
RETURN

SET UP POINTERS FOR FIXING LEAF
GET Y TRACKING COORDINATE
INVERT SIGN BIT
SET THIRD Y DISPLACEMENT
GET X TRACKING COORDINATE
INVERT SIGN BIT, SET ESCAPE BIT
SET THIRD X DISPLACEMENT
CORRECT INTENSIFIED VECTOR
RETURN

FORM POINTER TO FIRST Y DISPLACEMENT
SAVE
FORM POINTER TO FIRST X DISPLACEMENT
SAVE
FORM POINTER TO SECOND Y DISPLACEMENT
SAVE
FORM POINTER TO SECOND X DISPLACEMENT
SAVE
FORM POINTER TO THIRD Y DISPLACEMENT
SAVE
FORM POINTER TO THIRD X DISPLACEMENT
SAVE
READ Y TRACKING COORDINATE
DIVIDE BY 2
READ X TRACKING COORDINATE
DIVIDE BY 2
JMS* =C.BC CONVERT TO DISPLAY COORDINATE
DAC FIXX SAVE
JMP* FIXRD RETURN

FIXFIX $DC 0

LAC* FIX1 GET FIRST Y DISPLACEMENT
JMS* =C.CB CONVERT TO TWOS COMPLEMENT
DAC FIXY SAVE
LAC* FIX5 GET THIRD Y DISPLACEMENT
JMS* =C.CB CONVERT TO TWOS COMPLEMENT
TAD FIXY ADD FIRST Y DISPLACEMENT
JMS* =C.BC CONVERT TO DISPLAY COORDINATE
SZA SKIP IF Y DISPLACEMENTS WERE EQUAL
JMP *+7 CONVERTED VALUE IS NONZERO
LAC* FIX5 GET THIRD Y DISPLACEMENT
JMS* =C.CB CONVERT TO TWOS COMPLEMENT
TAD =1 MAKE DIFFERENT FROM 1ST Y DISPLACEMENT
JMS* =C.BC CONVERT TO DISPLAY COORDINATE
DAC* FIX5 STORE MODIFIED THIRD Y DISPLACEMENT
LAW 1 GET DISPLACEMENT OF 1
XOR =6000 SET ESCAPE BIT, INVERT SIGN BIT
DAC* FIX3 STORE SECOND Y DISPLACEMENT
LAC* FIX2 GET FIRST X DISPLACEMENT
JMS* =C.CB CONVERT TO TWOS COMPLEMENT
DAC FIXX SAVE
LAC* FIX6 GET THIRD X DISPLACEMENT
JMS* =C.CB CONVERT TO TWOS COMPLEMENT
TAD FIXX ADD FIRST X DISPLACEMENT
JMS* =C.BC CONVERT TO DISPLAY COORDINATE
XOR =2000 INVERT SIGN BIT
DAC* FIX4 SET SECOND X DISPLACEMENT
JMP* FIXFIX RETURN

TXT $DC 2
$TEXT "SELGI"

TXT1 $DC 2
$TEXT "DRAW"
3.11 Text List Manipulation

A structure which may be used to represent efficiently strings of text in core is called a "text list." A text list consists of a word which contains a number m which represents the length of the list, followed by m words, each of which contains three 6-bit characters. As an example, a text list which represents the string

A SIMPLE EXAMPLE

is the following (in octal form):

```
000006
127634
222631
251676
164112
263125
167777
```

This text list may easily be represented in assembly language via the TEXT pseudo-op:

```
LIST $DC 6
$TEXT "A SIMPLE EXAMPLE"
```

The address of the text list is the address of its first word. In this example, LIST is a symbol whose value is the address of the text list.

A "text leaf" is a representation of a text list as a display leaf. The leaf is composed of a series of push jumps to various character generation subroutines within the System.
A carriage return, however, is represented explicitly in the text leaf by three words which generate a vector which restores the X coordinate to its value just before the display control enters the text leaf. An additional vector is included at the end of the text leaf to restore both the X and Y coordinates. The high-order six bits of the second word of each push jump contain the 6-bit code for the character which the push jump represents. Each character is drawn in increment mode and is 7 points high by 5 points wide. The trailing space, which is produced by each character generation subroutine, is 3 points wide.

As an example of a text leaf, consider the following text list:

LEAF
$DC 10
$TEXT "EXAMPLE OF"
$DC 747577
$TEXT "2 LINES"

The text leaf which would be produced from this text list is the following:

762010
16----
762010
41----
762010
12----
762010
26----
762010
31----
762010
25----
762010
16----
762010
76----
762010
30----
762010
17----
761121
400000
006120
The following system subroutines have been defined for manipulating text lists and text leaves:

L.T - The text list whose address is given in bits 3-17 of the AC is typed on the teletype.

L.D* - A text leaf is generated from the text list whose address is given in bits 3-17 of the AC. The address of the generated text leaf is returned in bits 3-17 of the AC. A failure return is made if the text leaf cannot be generated because of insufficient free display storage.

L.L - The text leaf whose address is given in bits 3-17 of the AC is destroyed, and the storage which it occupied is salvaged by the System.
4. IDLE-TIME TASK

The idle-time task, which is executed whenever the System is in system state (Section 2.3), interprets various keyboard commands which provide some functions which are useful for testing and modifying user tasks. These commands are described in Sections 4.1 through 4.5. Each command is given by typing only the underlined characters; the System will type all other characters shown.

4.1 Copy Functions

The command

```
FROM {TELETYPE \ PAPER TAPE \ CORE} TO {TELETYPE \ PAPER TAPE \ CORE \ DISPLAY}
```

allows the operator to transfer data from teletype, paper tape, or core to teletype, paper tape, core, or the display. Many of these copy functions normally are specified by other names. For example, a copy from paper tape to core is called loading, a copy from core to teletype or from core to display is called a dump, a copy from teletype to core is called altering, etc.

When a transfer from teletype to any device other than core is specified, everything typed on the teletype up to the next character which maps into a 6-bit null character (Section 3.3) is transferred to the device specified. After a null character is typed, the idle-time task is ready for a new command. When copying from teletype to core, the following sequence of events occurs:

1. The operator types a 5-digit octal address on the keyboard. If one character which he types is not an octal digit, it is interpreted as the first character of a new command, and the copy from teletype to core is terminated.
(2) The idle-time task types the content of the location specified on the current line of text.

(3) The operator types a 6-digit octal content to replace the content of the location specified on the current line of text. If he types a carriage return in place of one of the octal digits, the content of the location is left unchanged. If he types a character which is neither an octal digit nor a carriage return, the copy task proceeds with Step 1.

(4) The address of the location which immediately follows the one which was just examined (and perhaps modified) is typed. The copy task then proceeds with Step 2.

As an example of a copy from teletype to core, consider setting the content of location \(23571_8\) to \(547521_8\) and the content of location \(23574_8\) to \(607213_8\). This may be accomplished by either of the following procedures:

**FROM TELETYPE TO CORE**

\[
\begin{array}{ccc}
23571 & 172356 & 547521 \\
23572 & 543125 & \text{(carriage return)} \\
23573 & 601241 & \text{(carriage return)} \\
23574 & 760001 & 607213 \\
23575 & 127123 & \text{(rubout)} \\
\text{FROM ----} & & \text{(new command)}
\end{array}
\]

**FROM TELETYPE TO CORE**

\[
\begin{array}{ccc}
23571 & 172356 & 547521 \\
23572 & 543125 & \text{(rubout)} \\
23574 & 760001 & 607213 \\
23575 & 127123 & \text{(rubout)} \\
\text{FROM ----} & & \text{(new command)}
\end{array}
\]

When a copy from paper tape to any device other than core is specified, the next alphanumeric record (Section 3.4.2) is read, and all binary records which are encountered before it
are ignored. (However, if the alphanumeric record is too long for the display, and a copy from paper tape to display is specified, only part of the alphanumeric record is read. The next part of the record may be displayed by another copy from paper tape to display.) Similarly, whenever a copy from paper tape to core is specified, the next binary record is read, and all alphanumeric records which are encountered before it are ignored.

When a copy from core to any device is specified, the specification of a block of core locations is requested from the operator. For example, the operator may dump locations $2357_8$ through $23602_8$ on the teletype as follows:

```
FROM CORE TO TELETYPET
BLOCK (23571, 23602)
23571 172356 543125 601241 760001 127123 127124 000200 000001
23601 000236 777777
```

A copy from core to core will also request the address of the first location in the block into which the information is to be moved. For example, locations $20052_8$ through $20056_8$ may be moved into locations $21521_8$ through $21525_8$ by the following command:

```
FROM CORE TO CORE
BLOCK (20052, 20056) TO 21521
```

Since the words in a block to be moved by a copy from core to core are moved one at a time, starting with the lowest address of the specified block, the following sequence of commands may be used to store zeros in all of core bank 1. (This is sometimes a useful operation to perform before loading a program to be debugged, since it stores illegal instructions throughout core bank 1.)

```
FROM TELETYPET TO CORE
20000 172132 000000
20001 172312 (rubout)
```
FROM CORE TO CORE
BLOCK (20000, 37776) TO 20001

The copy from core to core in this example moves the zero from location 20000\(_g\) into location 20001\(_g\), then it moves the zero from location 20001\(_g\) into location 20002\(_g\), etc.

Copy functions to the display are constrained to a maximum of 64 characters per line and to 10 lines. For this reason, a maximum of 100\(_g\) locations may be dumped on the screen at one time, and a copy from paper tape or teletype to display will be terminated at the end of 10 lines.

4.2 Scheduling of User Tasks

User tasks may be scheduled while in system state, but they will not be executed until user state is entered (Section 4.5). The command which accomplishes this is the following:

```
SCHEDULE ___
```

In the blanks after the word "schedule" the operator should type a 5-digit octal address where the task which he is scheduling begins. For example, a user task which starts at location 20571\(_g\) may be scheduled by the following command:

```
SCHEDULE 20571
```

4.3 Clearing the Task Queue or Display Storage

The command

```
CLEAR TASK QUEUE DISPLAY STORAGE
```

allows the operator to remove all user tasks scheduled by the command described in Section 4.2 from the task queue, or to clear the display storage area. When a copy function to the display is performed, the comment

```
NOT ENOUGH DISPLAY STORAGE
```
may be printed on the teletype, and the copy function will not
be completed. The facility of clearing the display storage
area is provided to allow the operator to destroy all display
structures to provide display storage for copy functions to
the display.

4.4 Teletype to Dataphone Transmission

Since most messages to be sent over the 201A data-
phone to a remote computer from the teletype are record-oriented,
rather than character-oriented, and since ASCII codes are ac-
cepted as standards for this type of communication, a copy from
the teletype to the dataphone is handled in a different manner
from other copy functions. If the command "#" is typed, all
succeeding characters typed on the keyboard, up to the first
carriage return, are sent over the dataphone as one record of
ASCII characters. (Of course, any response to such a record
which does not begin with the 8-bit character 000₈ will be typed
by the 201-to-teleprinter task.) However, a rubout will delete
a partially typed line, and the character "••-" will delete the
previous character on the line, if it exists. This command is
terminated when the line is terminated or deleted. A record
consisting of an enquiry (used as an end-of-record character)
may be sent from the teletype by striking the "WRU" key when
the idle-time task is expecting a new command.

4.5 Entering User State

The command

RUN

causes all user tasks which have been scheduled by the command
described in Section 4.2 to become eligible for execution, and
the idle-time task to be terminated. This causes the System
to enter user state (Section 2.3).
5. SYSTEM CAPABILITY

The System was designed primarily to support user tasks which provide communication between the operator and the 339 via network diagrams and between the 339 and a large time-sharing system. As can be determined by examination of the display structure, the display support provided by the System is easily applied to almost any display-oriented task which is two-dimensional in nature (e.g., network diagrams, two-dimensional Sketchpad programs, line-oriented text editing, etc.). The System offers no support for tasks which involve three-dimensional projection in that: (1) floating point arithmetic (which is almost essential for this type of task) is not provided, and (2) the display structure has no provision for storing the extra information required for three-dimensional projection.

Because a timesharing system is not always available to support preparation and testing of remote terminal programs, the philosophy behind the design of the system was to consider the remote terminal as an independent unit which considers the large timesharing system to be an I/O device. This differs from the philosophy, which is commonly applied to the design of remote terminal software systems, that the large timesharing system must be available to support the remote terminal system whenever the remote system is operating.
BIBLIOGRAPHY


APPENDIX A -- LISTING OF THE EXECUTIVE SYSTEM

$ORG 17732
$TITLE
IOT $OPDM 70000
HLT $OPD 740040

IOT 3302
JMP SYSTEM

*1 JMS 4
SNA
JMP 2
DAC .5
AND .7
SAD .8
SKP
JMP 2
JMS 3
DAC .6
JMP 1

*2 JMS 3
DAC* .6
ISZ .6
JMP 1

*3 SDC 0
JMS 4
LRS 6
LAC .5
LLS 6
DAC .5
JMS 4
LRS 6
LAC .5
LLS 6
JMP 3

*4 SDC 0
IOT 104
IOT 101

SEL EXECUTIVE SYSTEM LOADER
CLEAR ALL FLAGS
START SYSTEM
READ FIRST LINE OF 3-LINE BLOCK
SKIP IF NONBLANK TAPE
BLANK TAPE -- TRY AGAIN
SAVE FIRST LINE IMAGE
REMOVE DATA BITS
SKIP IF DATA LINE
ORIGIN LINE
DATA LINE
FINISH ORIGIN WORD
SET LOCATION COUNTER
READ NEXT BLOCK
FINISH DATA WORD
LOAD DATA WORD
INCREMENT LOCATION COUNTER
READ NEXT BLOCK
READ SECOND LINE
SHIFT DATA BITS INTO MQ
LOAD AC WITH FIRST LINE IMAGE
SHIFT CONCATENATED IMAGE INTO AC
SAVE CONCATENATED FIRST TWO LINES
READ THIRD LINE
SHIFT DATA BITS INTO MQ
LOAD AC WITH CONCATENATED IMAGE
SHIFT COMPLETED WORD INTO AC
RETURN
SELECT READER
SKIP IF LINE READY
<table>
<thead>
<tr>
<th>Address</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP *-1</td>
<td>WAIT FOR FLAG</td>
</tr>
<tr>
<td>IOT 112</td>
<td>OVERRIDDEN &quot;JMP .1-2&quot; WHEN LOADED</td>
</tr>
<tr>
<td>JMP .4</td>
<td>RETURN</td>
</tr>
<tr>
<td>.5</td>
<td>SDC 0</td>
</tr>
<tr>
<td>.6</td>
<td>SDC 0</td>
</tr>
<tr>
<td>.7</td>
<td>SDC 300</td>
</tr>
<tr>
<td>.8</td>
<td>SDC 100</td>
</tr>
<tr>
<td>JMP .1</td>
<td>OVERRIDES BOOTSTRAP LOCATION 0</td>
</tr>
</tbody>
</table>
$TITLE

CONTROL DISPATCHER

$SORG 1
JMP I

INTERRUPT TRAP

$SORG 21
JMP ET
JMP ES

ILLEGAL INSTRUCTION TRAP

SYSTEM RESTART

$SORG 100
Q.C SDC 0
JMP QC

CLEAR QUEUE

Q.A SDC 0
JMP QA

ADD WORD TO BOTTOM OF QUEUE (F)

Q.I SDC 0
JMP QI

INSERT WORD ON TOP OF QUEUE (F)

Q.F SDC 0
JMP QF

FETCH WORD FROM TOP OF QUEUE (F)

T.S SDC 0
JMP TS

SCHEDULE TASK

T.P SDC 0
JMP TP

SCHEDULE PREVIOUS LOC & TERMINATE

T.F SDC 0
JMP TF

TERMINATE CURRENT TASK

T.A SDC 0
JMP TA

ALLOCATE I/O DEVICES UNDER MASK

T.R SDC 0
JMP TR

RELEASE I/O DEVICES UNDER MASK

T.L SDC 0
JMP TL

LOCK REENTRABLE SUBROUTINE

T.U SDC 0
JMP TU

UNLOCK REENTRABLE SUBROUTINE

C.B6 SDC 0
JMP CB6

CONVERT BINARY TO 6-BIT OCTAL

C.6A SDC 0
JMP C6A

CONVERT 6-BIT TO ASCII

C.A6 SDC 0
JMP CA6

CONVERT ASCII TO 6-BIT

C.CB SDC 0
JMP CCB

CONVERT DISPLAY COORDINATE TO BINARY
A-4

<table>
<thead>
<tr>
<th>Code</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.BC</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPCBC</td>
<td></td>
</tr>
<tr>
<td>B.FI</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPBFI</td>
<td></td>
</tr>
<tr>
<td>B.FO</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPBFO</td>
<td></td>
</tr>
<tr>
<td>B.R</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPBR</td>
<td></td>
</tr>
<tr>
<td>B.P</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPP</td>
<td></td>
</tr>
<tr>
<td>B.K</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPBK</td>
<td></td>
</tr>
<tr>
<td>B.T</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPBT</td>
<td></td>
</tr>
<tr>
<td>N.A</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPPNA</td>
<td></td>
</tr>
<tr>
<td>N.C</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPNC</td>
<td></td>
</tr>
<tr>
<td>N.D1</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPND1</td>
<td></td>
</tr>
<tr>
<td>N.D2</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPND2</td>
<td></td>
</tr>
<tr>
<td>N.D3</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPND3</td>
<td></td>
</tr>
<tr>
<td>P.T</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPPT</td>
<td></td>
</tr>
<tr>
<td>P.E</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPP</td>
<td></td>
</tr>
<tr>
<td>P.D</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPPD</td>
<td></td>
</tr>
<tr>
<td>P.R</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPIRP</td>
<td></td>
</tr>
<tr>
<td>P.S</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPIPS</td>
<td></td>
</tr>
<tr>
<td>D.E</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPDE</td>
<td></td>
</tr>
<tr>
<td>D.D</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPDD</td>
<td></td>
</tr>
<tr>
<td>D.P</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMPIPD</td>
<td></td>
</tr>
<tr>
<td>Symbol</td>
<td>Opcode</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>D.A</td>
<td>SDC 0</td>
</tr>
<tr>
<td>D.Y</td>
<td>SDC 0</td>
</tr>
<tr>
<td>D.X</td>
<td>SDC 0</td>
</tr>
<tr>
<td>D.O</td>
<td>SDC 0</td>
</tr>
<tr>
<td>X.I</td>
<td>SDC 0</td>
</tr>
<tr>
<td>X.R</td>
<td>SDC 0</td>
</tr>
<tr>
<td>X.T</td>
<td>SDC 0</td>
</tr>
<tr>
<td>X.S</td>
<td>SDC 0</td>
</tr>
<tr>
<td>X.Y</td>
<td>SDC 0</td>
</tr>
<tr>
<td>X.X</td>
<td>SDC 0</td>
</tr>
<tr>
<td>S.TL</td>
<td>SDC 0</td>
</tr>
<tr>
<td>S.TD</td>
<td>SDC 0</td>
</tr>
<tr>
<td>S.TI</td>
<td>SDC 0</td>
</tr>
<tr>
<td>S.TR</td>
<td>SDC 0</td>
</tr>
<tr>
<td>S.LH</td>
<td>SDC 0</td>
</tr>
<tr>
<td>S.LY</td>
<td>SDC 0</td>
</tr>
<tr>
<td>S.LX</td>
<td>SDC 0</td>
</tr>
<tr>
<td>S.LP</td>
<td>SDC 0</td>
</tr>
<tr>
<td>S.LBE</td>
<td>SDC 0</td>
</tr>
<tr>
<td>S.LBD</td>
<td>SDC 0</td>
</tr>
<tr>
<td></td>
<td>Count Scale and/or Intensity</td>
</tr>
<tr>
<td></td>
<td>Interrupt Unconditionally on Level</td>
</tr>
<tr>
<td></td>
<td>Interrupt on Level if On Screen</td>
</tr>
<tr>
<td></td>
<td>Interrupt on Level if LPSI Set</td>
</tr>
<tr>
<td></td>
<td>Disable Interrupt on Level</td>
</tr>
<tr>
<td></td>
<td>Send Text List to TP Buffer</td>
</tr>
<tr>
<td></td>
<td>Generate Text Leaf (F)</td>
</tr>
<tr>
<td></td>
<td>Destroy Text Leaf</td>
</tr>
</tbody>
</table>

|  |                          |
|  |                            |
|  |                            |

### Memory Locations

<table>
<thead>
<tr>
<th>Location</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPPI</td>
<td>204</td>
</tr>
<tr>
<td>PDP2</td>
<td>204</td>
</tr>
<tr>
<td>PDP1</td>
<td></td>
</tr>
</tbody>
</table>

**Instructions:**

- **S.C:** SDC 0
- **S.LU:** SDC 0
- **S.LS:** SDC 0
- **S.LL:** SDC 0
- **S.LN:** SDC 0
- **L.T:** SDC 0
- **L.D:** SDC 0
- **L.L:** SDC 0
- **PDPI:** SDS 204
- **PDP2:** SDS 204
<table>
<thead>
<tr>
<th>D00</th>
<th>INCR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SDC 1272</td>
</tr>
<tr>
<td></td>
<td>SDC 6251</td>
</tr>
<tr>
<td></td>
<td>SDC 6057</td>
</tr>
<tr>
<td></td>
<td>SDC 7516</td>
</tr>
<tr>
<td></td>
<td>SDC 1570</td>
</tr>
<tr>
<td></td>
<td>SDC 5172</td>
</tr>
<tr>
<td></td>
<td>SDC 3726</td>
</tr>
<tr>
<td></td>
<td>SDC 0</td>
</tr>
<tr>
<td></td>
<td>POP</td>
</tr>
<tr>
<td>D01</td>
<td>INCR</td>
</tr>
<tr>
<td></td>
<td>SDC 5160</td>
</tr>
<tr>
<td></td>
<td>SDC 1472</td>
</tr>
<tr>
<td></td>
<td>SDC 7255</td>
</tr>
<tr>
<td></td>
<td>SDC 3737</td>
</tr>
<tr>
<td></td>
<td>SDC 0</td>
</tr>
<tr>
<td></td>
<td>POP</td>
</tr>
<tr>
<td>D02</td>
<td>INCR</td>
</tr>
<tr>
<td></td>
<td>SDC 5271</td>
</tr>
<tr>
<td></td>
<td>SDC 5152</td>
</tr>
<tr>
<td></td>
<td>SDC 5364</td>
</tr>
<tr>
<td></td>
<td>SDC 5537</td>
</tr>
<tr>
<td></td>
<td>SDC 2774</td>
</tr>
<tr>
<td></td>
<td>SDC 5417</td>
</tr>
<tr>
<td></td>
<td>SDC 3020</td>
</tr>
<tr>
<td></td>
<td>SDC 0</td>
</tr>
<tr>
<td></td>
<td>POP</td>
</tr>
<tr>
<td>D03</td>
<td>INCR</td>
</tr>
<tr>
<td></td>
<td>SDC 1252</td>
</tr>
<tr>
<td></td>
<td>SDC 5760</td>
</tr>
<tr>
<td></td>
<td>SDC 5152</td>
</tr>
<tr>
<td></td>
<td>SDC 5354</td>
</tr>
<tr>
<td></td>
<td>SDC 1051</td>
</tr>
<tr>
<td></td>
<td>SDC 5253</td>
</tr>
<tr>
<td></td>
<td>SDC 6455</td>
</tr>
<tr>
<td></td>
<td>SDC 3737</td>
</tr>
<tr>
<td></td>
<td>SDC 1000</td>
</tr>
<tr>
<td></td>
<td>POP</td>
</tr>
<tr>
<td>D04</td>
<td>INCR</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>SDC</td>
<td>1110</td>
</tr>
<tr>
<td>SDC</td>
<td>5072</td>
</tr>
<tr>
<td>SDC</td>
<td>7275</td>
</tr>
<tr>
<td>SDC</td>
<td>6110</td>
</tr>
<tr>
<td>SDC</td>
<td>5037</td>
</tr>
<tr>
<td>SDC</td>
<td>1600</td>
</tr>
<tr>
<td>POP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D05</th>
<th>INCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDC</td>
<td>1252</td>
</tr>
<tr>
<td>SDC</td>
<td>5760</td>
</tr>
<tr>
<td>SDC</td>
<td>5162</td>
</tr>
<tr>
<td>SDC</td>
<td>5374</td>
</tr>
<tr>
<td>SDC</td>
<td>6270</td>
</tr>
<tr>
<td>SDC</td>
<td>5037</td>
</tr>
<tr>
<td>SDC</td>
<td>3616</td>
</tr>
<tr>
<td>SDC</td>
<td>0</td>
</tr>
<tr>
<td>POP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D06</th>
<th>INCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDC</td>
<td>1252</td>
</tr>
<tr>
<td>SDC</td>
<td>5760</td>
</tr>
<tr>
<td>SDC</td>
<td>5152</td>
</tr>
<tr>
<td>SDC</td>
<td>5364</td>
</tr>
<tr>
<td>SDC</td>
<td>5572</td>
</tr>
<tr>
<td>SDC</td>
<td>5170</td>
</tr>
<tr>
<td>SDC</td>
<td>3736</td>
</tr>
<tr>
<td>SDC</td>
<td>1600</td>
</tr>
<tr>
<td>POP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D07</th>
<th>INCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDC</td>
<td>5271</td>
</tr>
<tr>
<td>SDC</td>
<td>5162</td>
</tr>
<tr>
<td>SDC</td>
<td>7454</td>
</tr>
<tr>
<td>SDC</td>
<td>5637</td>
</tr>
<tr>
<td>SDC</td>
<td>3710</td>
</tr>
<tr>
<td>SDC</td>
<td>0</td>
</tr>
<tr>
<td>POP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D10</th>
<th>INCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDC</td>
<td>1252</td>
</tr>
<tr>
<td>SDC</td>
<td>5760</td>
</tr>
<tr>
<td>SDC</td>
<td>5152</td>
</tr>
</tbody>
</table>
SDC 5364
SDC 5512
SDC 6251
SDC 6057
SDC 5637
SDC 2600
POP

D11  INCR
SDC 5270
SDC 5162
SDC 7453
SDC 5251
SDC 6057
SDC 5637
SDC 2600
POP

D12  INCR
SDC 7272
SDC 5160
SDC 5766
SDC 7632
SDC 7437
SDC 1720
SDC 0
POP

D13  INCR
SDC 7272
SDC 5270
SDC 5756
SDC 5564
SDC 2057
SDC 5655
SDC 6430
SDC 1720
SDC 0
POP

D14  INCR
SDC 1272
SDC 6251
SDC 6057
A-10

```
SDC 3656
SDC 5564
SDC 3020
SDC 1700
POP

D15 INCR
SDC 7272
SDC 5025
SDC 5550
SDC 2516
SDC 1720
SDC 0
POP

D16 INCR
SDC 7272
SDC 5025
SDC 5550
SDC 2516
SDC 1720
SDC 0
POP

D17 INCR
SDC 7272
SDC 5025
SDC 5550
SDC 3717
SDC 1000
POP

D20 INCR
SDC 1272
SDC 6051
SDC 6057
SDC 3570
SDC 5655
SDC 6430
SDC 1720
SDC 0
```
<table>
<thead>
<tr>
<th>D21</th>
<th>INCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>$DC$ 7272</td>
<td></td>
</tr>
<tr>
<td>$DC$ 7236</td>
<td></td>
</tr>
<tr>
<td>$DC$ 7050</td>
<td></td>
</tr>
<tr>
<td>$DC$ 7236</td>
<td></td>
</tr>
<tr>
<td>$DC$ 7620</td>
<td></td>
</tr>
<tr>
<td>$DC$ 1700</td>
<td></td>
</tr>
<tr>
<td>POP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D22</th>
<th>INCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>$DC$ 5160</td>
<td></td>
</tr>
<tr>
<td>$DC$ 1472</td>
<td></td>
</tr>
<tr>
<td>$DC$ 7254</td>
<td></td>
</tr>
<tr>
<td>$DC$ 1050</td>
<td></td>
</tr>
<tr>
<td>$DC$ 3736</td>
<td></td>
</tr>
<tr>
<td>$DC$ 1700</td>
<td></td>
</tr>
<tr>
<td>POP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D23</th>
<th>INCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>$DC$ 2252</td>
<td></td>
</tr>
<tr>
<td>$DC$ 5657</td>
<td></td>
</tr>
<tr>
<td>$DC$ 6051</td>
<td></td>
</tr>
<tr>
<td>$DC$ 7242</td>
<td></td>
</tr>
<tr>
<td>$DC$ 3636</td>
<td></td>
</tr>
<tr>
<td>$DC$ 1720</td>
<td></td>
</tr>
<tr>
<td>$DC$ 0</td>
<td></td>
</tr>
<tr>
<td>POP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D24</th>
<th>INCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>$DC$ 7272</td>
<td></td>
</tr>
<tr>
<td>$DC$ 5230</td>
<td></td>
</tr>
<tr>
<td>$DC$ 5075</td>
<td></td>
</tr>
<tr>
<td>$DC$ 7720</td>
<td></td>
</tr>
<tr>
<td>$DC$ 1700</td>
<td></td>
</tr>
<tr>
<td>POP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D25</th>
<th>INCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>$DC$ 7272</td>
<td></td>
</tr>
<tr>
<td>$DC$ 5236</td>
<td></td>
</tr>
<tr>
<td>$DC$ 3670</td>
<td></td>
</tr>
<tr>
<td>$DC$ 5020</td>
<td></td>
</tr>
<tr>
<td>$DC$ 1700</td>
<td></td>
</tr>
<tr>
<td>POP</td>
<td></td>
</tr>
<tr>
<td>D26</td>
<td>INCR</td>
</tr>
<tr>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>SDC 7272</td>
</tr>
<tr>
<td></td>
<td>SDC 5267</td>
</tr>
<tr>
<td></td>
<td>SDC 6176</td>
</tr>
<tr>
<td></td>
<td>SDC 7620</td>
</tr>
<tr>
<td></td>
<td>SDC 1700</td>
</tr>
<tr>
<td>POP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D27</th>
<th>INCR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SDC 7272</td>
</tr>
<tr>
<td></td>
<td>SDC 5277</td>
</tr>
<tr>
<td></td>
<td>SDC 3250</td>
</tr>
<tr>
<td></td>
<td>SDC 7676</td>
</tr>
<tr>
<td></td>
<td>SDC 1720</td>
</tr>
<tr>
<td></td>
<td>SDC 0</td>
</tr>
<tr>
<td>POP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D30</th>
<th>INCR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SDC 1272</td>
</tr>
<tr>
<td></td>
<td>SDC 6251</td>
</tr>
<tr>
<td></td>
<td>SDC 6057</td>
</tr>
<tr>
<td></td>
<td>SDC 7656</td>
</tr>
<tr>
<td></td>
<td>SDC 5564</td>
</tr>
<tr>
<td></td>
<td>SDC 3020</td>
</tr>
<tr>
<td></td>
<td>SDC 1700</td>
</tr>
<tr>
<td>POP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D31</th>
<th>INCR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SDC 7272</td>
</tr>
<tr>
<td></td>
<td>SDC 5270</td>
</tr>
<tr>
<td></td>
<td>SDC 5756</td>
</tr>
<tr>
<td></td>
<td>SDC 5564</td>
</tr>
<tr>
<td></td>
<td>SDC 3720</td>
</tr>
<tr>
<td></td>
<td>SDC 1700</td>
</tr>
<tr>
<td>POP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D32</th>
<th>INCR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SDC 1272</td>
</tr>
<tr>
<td></td>
<td>SDC 6251</td>
</tr>
<tr>
<td></td>
<td>SDC 6057</td>
</tr>
<tr>
<td></td>
<td>SDC 7656</td>
</tr>
<tr>
<td></td>
<td>SDC 5564</td>
</tr>
<tr>
<td></td>
<td>SDC 1022</td>
</tr>
<tr>
<td></td>
<td>SDC 7720</td>
</tr>
</tbody>
</table>
A-13

SDC 0
POP
D33
INCR
SDC 7272
SDC 5270
SDC 5756
SDC 5564
SDC 7720
SDC 1700
POP
D34
INCR
SDC 1252
SDC 5760
SDC 5152
SDC 5364
SDC 5352
SDC 5160
SDC 5737
SDC 3600
POP
D35
INCR
SDC 1150
SDC 7272
SDC 6420
SDC 6036
SDC 1637
SDC 0
POP
D36
INCR
SDC 1272
SDC 7230
SDC 5076
SDC 6655
SDC 6430
SDC 1720
SDC 0
POP
D37
INCR
SDC 2272
SDC 6230
A-14

SDC 5076
SDC 5665
SDC 5327
SDC 3010
SDC 0
POP
D40
INCR
SDC 7272
SDC 5230
SDC 5076
SDC 7663
SDC 5527
SDC 3010
SDC 0
POP
D41
INCR
SDC 6271
SDC 5152
SDC 3454
SDC 5657
SDC 1767
SDC 5617
SDC 2000
POP
D42
INCR
SDC 1150
SDC 7261
SDC 5234
SDC 5456
SDC 5737
SDC 2710
SDC 0
POP
D43
INCR
SDC 6271
SDC 5152
SDC 7454
SDC 3727
SDC 5574
SDC 3020
SDC 1700
POP
D44 INCR
SDC 1251
SDC 7151
SDC 3454
SDC 5717
SDC 6727
SDC 0
POP
D45 INCR
SDC 1252
SDC 7151
SDC 3736
SDC 0
POP
D46 INCR
SDC 1151
SDC 7252
SDC 1555
SDC 5010
SDC 6037
SDC 1600
POP
D47 INCR
SDC 3252
SDC 7050
SDC 3716
SDC 0
POP
D50 INCR
SDC 1150
SDC 5372
SDC 5251
SDC 3727
SDC 2600
POP
D51 INCR
SDC 1150
SDC 5172
A-16

$DC 5253
$DC 3727
$DC 2600
POP

D52 INCR
$DC 5172
$DC 7260
$DC 3636
$DC 5450
$DC 1730
$DC 0
POP

D53 INCR
$DC 5160
$DC 7272
$DC 6437
$DC 3716
$DC 0
POP

D54 INCR
$DC 3051
$DC 7371
$DC 3736
$DC 1600
POP

D55 INCR
$DC 3150
$DC 7454
$DC 1252
$DC 7050
$DC 3726
$DC 0
POP

D56 INCR
$DC 5271
$DC 7337
$DC 3717
$DC 0
POP

D57 INCR
$DC 1150
$DC 7272
$DC 6520
$DC 5157
$DC 3726
$DC 0
POP
D60 INCR
$DC 1151
$DC 6361
$DC 1655
$DC 7037
$DC 1600
POP
D61 INCR
$DC 6250
$DC 5630
$DC 1720
$DC 0
POP
D62 INCR
$DC 6250
$DC 6655
$DC 1130
$DC 3000
POP
D63 INCR
$DC 1262
$DC 1262
$DC 5056
$DC 1666
$DC 2710
$DC 3000
POP
D64 INCR
$DC 1262
$DC 1262
$DC 5056
$DC 1676
$DC 5530
$DC 3010
$DC 0
POP

D65  INCR
$DC 1150
$DC 1262
$DC 5051
$DC 5253
$DC 6455
$DC 3737
$DC 1000
POP

D66  INCR
$DC 1150
$DC 1272
$DC 6237
$DC 2726
$DC 0
POP

D67  INCR
$DC 2122
$DC 7237
$DC 2726
$DC 0
POP

D70  INCR
$DC 1132
$DC 1210
$DC 5066
$DC 3717
$DC 1600
POP

D71  INCR
$DC 1252
$DC 5760
$DC 5152
$DC 5364
$DC 5352
$DC 5160
$DC 5714
A-19

```
SDC 5456
SDC 1666
SDC 2730
SDC 0
POP
D72 INCR
SDC 5172
SDC 7210
SDC 5076
SDC 7612
SDC 5314
SDC 5412
SDC 5210
SDC 5010
SDC 5016
SDC 5637
SDC 0
POP
D73 INCR
SDC 5153
SDC 5261
SDC 5355
SDC 1777
SDC 1454
SDC 1151
SDC 3700
POP
D75 VEC
SDC 2020
SDC 4000
POP
D76 SVEC
SDC 50
POP
```
<table>
<thead>
<tr>
<th>XP</th>
<th>LAW 3000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SDC 1105</td>
</tr>
<tr>
<td>XPY</td>
<td>SDC 1000</td>
</tr>
<tr>
<td>XPX</td>
<td>SDC 5000</td>
</tr>
<tr>
<td></td>
<td>SDC 1400</td>
</tr>
<tr>
<td></td>
<td>SDC X1</td>
</tr>
<tr>
<td></td>
<td>SDC 60</td>
</tr>
<tr>
<td>SVEC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SDC 24</td>
</tr>
<tr>
<td></td>
<td>SDC 4047</td>
</tr>
<tr>
<td></td>
<td>SDC 1400</td>
</tr>
<tr>
<td></td>
<td>SDC X2</td>
</tr>
<tr>
<td></td>
<td>SDC 60</td>
</tr>
<tr>
<td>SVEC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SDC 1</td>
</tr>
<tr>
<td></td>
<td>SDC 4067</td>
</tr>
<tr>
<td></td>
<td>SDC 1400</td>
</tr>
<tr>
<td></td>
<td>SDC X3</td>
</tr>
<tr>
<td></td>
<td>SDC 60</td>
</tr>
<tr>
<td>SVEC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SDC 2403</td>
</tr>
<tr>
<td></td>
<td>SDC 4740</td>
</tr>
<tr>
<td></td>
<td>SDC 1400</td>
</tr>
<tr>
<td></td>
<td>SDC X2</td>
</tr>
<tr>
<td></td>
<td>SDC 60</td>
</tr>
<tr>
<td>SVEC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SDC 100</td>
</tr>
<tr>
<td></td>
<td>SDC 6700</td>
</tr>
<tr>
<td></td>
<td>SDC 340</td>
</tr>
<tr>
<td></td>
<td>SDC 1400</td>
</tr>
<tr>
<td></td>
<td>SDC X4</td>
</tr>
<tr>
<td>XPS</td>
<td>POP</td>
</tr>
<tr>
<td></td>
<td>SVEC</td>
</tr>
<tr>
<td></td>
<td>SDC 404</td>
</tr>
<tr>
<td></td>
<td>SDC 4030</td>
</tr>
<tr>
<td></td>
<td>SDC 7000</td>
</tr>
<tr>
<td></td>
<td>SDC 4010</td>
</tr>
<tr>
<td></td>
<td>SDC 5040</td>
</tr>
<tr>
<td>VEC</td>
<td>SDC 4</td>
</tr>
<tr>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>SDC 4000</td>
</tr>
<tr>
<td></td>
<td>SDC 6020</td>
</tr>
<tr>
<td></td>
<td>SDC 4000</td>
</tr>
<tr>
<td></td>
<td>SDC 4020</td>
</tr>
<tr>
<td></td>
<td>SDC 4</td>
</tr>
<tr>
<td></td>
<td>SDC 4000</td>
</tr>
<tr>
<td></td>
<td>SDC 6030</td>
</tr>
<tr>
<td></td>
<td>SDC 4000</td>
</tr>
<tr>
<td></td>
<td>SDC 4030</td>
</tr>
<tr>
<td></td>
<td>SDC 4</td>
</tr>
<tr>
<td></td>
<td>SDC 4000</td>
</tr>
<tr>
<td></td>
<td>SDC 6040</td>
</tr>
<tr>
<td></td>
<td>SDC 4000</td>
</tr>
<tr>
<td></td>
<td>SDC 4040</td>
</tr>
<tr>
<td></td>
<td>SDC 4</td>
</tr>
<tr>
<td></td>
<td>SDC 4000</td>
</tr>
<tr>
<td></td>
<td>SDC 6050</td>
</tr>
<tr>
<td></td>
<td>SDC 4000</td>
</tr>
<tr>
<td></td>
<td>SDC 4050</td>
</tr>
</tbody>
</table>
$TITLE

INTERRUPT DISPATCHER

1 DAC 6      SAVE AC CONTENTS
      LACQ       GET MQ CONTENTS
      DAC 3      SAVE MQ CONTENTS
      LACS       GET SC CONTENTS
      DAC 2      SAVE SC CONTENTS
      IOT 1441   SKIP ON DATAPHONE RECEIVE FLAG
      JMP IFI    SERVICE DATAPHONE INPUT INTERRUPT
      IOT 1401   SKIP ON DATAPHONE TRANSMIT FLAG
      SKP        TEST NEXT FLAG
      JMP IFO    SERVICE DATAPHONE OUTPUT INTERRUPT
      IOT 101    SKIP ON READER FLAG
      SKP        TEST NEXT FLAG
      JMP IRD    SERVICE READER INTERRUPT
      IOT 1301   SKIP ON A/D CONVERTER FLAG
      SKP        TEST NEXT FLAG
      JMP IAD    SERVICE A/D CONVERTER INTERRUPT
      IOT 301    SKIP ON KEYBOARD FLAG
      SKP        TEST NEXT FLAG
      JMP IBK    SERVICE KEYBOARD INTERRUPT
      IOT 201    SKIP ON PUNCH FLAG
      SKP        TEST NEXT FLAG
      JMP IPC    SERVICE PUNCH INTERRUPT
      IOT 401    SKIP ON TELEPRINTER FLAG
      SKP        TEST NEXT FLAG
      JMP ITP    SERVICE TELEPRINTER INTERRUPT
      IOT 1      SKIP ON CLOCK FLAG
      SKP        TEST NEXT FLAG
      JMP ICK    SERVICE CLOCK INTERRUPT
      IOT 612    READ DISPLAY STATUS
      DAC DSS    SAVE DISPLAY STATUS WORD 1
      AND =20    GET PUSH BUTTON FLAG
      SZA        SKIP ON NO PUSH BUTTON FLAG
      JMP IPB    SERVICE PUSH BUTTON INTERRUPT
      IOT 702    SKIP ON EDGE FLAG
      JMP **3    TEST NEXT FLAG
      IOT 724    RESUME DISPLAY
      JMP IR     RETURN FROM INTERRUPT
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOT 642</td>
<td>SKIP ON LIGHT PEN FLAG</td>
</tr>
<tr>
<td>SKP</td>
<td>TEST NEXT FLAG</td>
</tr>
<tr>
<td>JMP ILP</td>
<td>SERVICE LIGHT PEN INTERRUPT</td>
</tr>
<tr>
<td>IOT 721</td>
<td>SKIP ON INTERNAL STOP FLAG</td>
</tr>
<tr>
<td>SKP</td>
<td>TEST NEXT FLAG</td>
</tr>
<tr>
<td>JMP II5</td>
<td>SERVICE INTERNAL STOP INTERRUPT</td>
</tr>
<tr>
<td>IOT 722</td>
<td>SKIP ON MANUAL INTERRUPT FLAG</td>
</tr>
<tr>
<td>JMP EI</td>
<td>INVALID INTERRUPT</td>
</tr>
<tr>
<td>JMP EM</td>
<td>EMERGENCY REINITIALIZATION</td>
</tr>
<tr>
<td>IR</td>
<td>GET SC CONTENTS</td>
</tr>
<tr>
<td>LAC 2</td>
<td>COMPLEMENT SHIFT COUNT</td>
</tr>
<tr>
<td>XOR =#77</td>
<td>FORM NORM INSTRUCTION</td>
</tr>
<tr>
<td>TAD =#640402</td>
<td>TRUNCATE CARRY</td>
</tr>
<tr>
<td>AND =#640477</td>
<td>STORE NORM INSTRUCTION</td>
</tr>
<tr>
<td>DAC *=#1</td>
<td>RESTORE SC CONTENTS</td>
</tr>
<tr>
<td>LAC 3</td>
<td>GET MQ CONTENTS</td>
</tr>
<tr>
<td>LMQ</td>
<td>RESTORE MQ CONTENTS</td>
</tr>
<tr>
<td>LAC 6</td>
<td>RESTORE AC CONTENTS</td>
</tr>
<tr>
<td>IOT 42</td>
<td>ENABLE INTERRUPTS</td>
</tr>
<tr>
<td>IOT 3344</td>
<td>DEBREAK AND RESTORE</td>
</tr>
<tr>
<td>JMP# 0</td>
<td>RETURN TO INTERRUPTED PROGRAM</td>
</tr>
</tbody>
</table>
$TITLE SYSTEM DIAGNOSTICS

SYSTEM LAW 4400 GET BREAK FIELD I PARAMETER
  IOT 705 LOAD BREAK FIELD
  LAW =1400 GET ADDRESS OF INTERNAL STOP
  IOT 1605 INITIALIZE DISPLAY
  LAC **+2 GET ADDRESS OF TEXT LIST
  JMP E INITIALIZE SYSTEM
  $DC 5

$TEXT "SYSTEM RELOADED"

EE IOT 42 ENABLE INTERRUPTS
  LAC BP3 GET PUNCH STATUS SWITCH
  SZA SKIP IF PUNCH IS IDLE
  JMP *-2 WAIT FOR PUNCH TO FINISH
  LAC BT1 GET TELEPRINTER STATUS SWITCH
  SZA SKIP IF TELEPRINTER IS IDLE
  JMP *-2 WAIT FOR TELEPRINTER TO FINISH
  IOT 1412 READ 201 STATUS
  AND =2 GET TRANSMIT STATE BIT
  SZA SKIP IF NOT TRANSMITTING
  JMP *-3 WAIT FOR END OF TRANSMISSION
  IOT 2 DISABLE INTERRUPTS
  LAC **+2 GET ADDRESS OF TEXT LIST
  JMP E REINITIALIZE SYSTEM
  $DC 6

$TEXT "TASK QUEUE EMPTY"

EI LAC **+2 GET ADDRESS OF TEXT LIST
  JMP E REINITIALIZE SYSTEM
  $DC 6

$TEXT "INVALID INTERRUPT"

EM LAC **+2 GET ADDRESS OF TEXT LIST
  JMP E REINITIALIZE SYSTEM
  $DC 6

$TEXT "MANUAL INTERRUPT"

EQ LAC **+2 GET ADDRESS OF TEXT LIST
  JMP E REINITIALIZE SYSTEM
$DC 7
$TEXT "TASK QUEUE OVERFLOW"

ES
DZM BP3 CLEAR PUNCH STATUS SWITCH
DZM BT1 CLEAR TELEPRINTER STATUS SWITCH
LAW 4400 GET BREAK FIELD 1 PARAMETER
IOT 705 LOAD BREAK FIELD
LAW #1400 GET ADDRESS OF INTERNAL STOP
IOT 1605 INITIALIZE DISPLAY
LAC ***2 GET ADDRESS OF TEXT LIST
JMP E REINITIALIZE SYSTEM
$DC 5
$TEXT "PANEL RECOVERY"

ET
IOT 2 DISABLE INTERRUPTS
CLC LOAD AC WITH -1
TAD 20 ADD PROGRAM COUNTER DURING TRAP
JMS C.86 CONVERT TO 6-BIT CODE
AND =7777 TRUNCATE HIGH ORDER DIGIT
TAD =760000 USE BLANK AS HIGH ORDER CHARACTER
DAC ET1 STORE HIGH ORDER CHARACTERS
LACQ GET LOW ORDER DIGITS
DAC ET2 STORE LOW ORDER DIGITS
LAC ***2 GET ADDRESS OF TEXT LIST
JMP E REINITIALIZE SYSTEM
$DC 13
$TEXT "ILLEGAL INSTRUCTION AT LOC"

ET1 $DC 0
ET2 $DC 0
**STITLE**

A-27

**SYSTEM INITIALIZER**

**E**

- **DAC 25**: SAVE ADDRESS OF DIAGNOSTIC
- **IOT 7702**: ENTER EXTEND MODE
- **IOT 1412**: READ 201 STATUS
- **AND #1**: GET RECEIVE STATE BIT
- **SZA**: SKIP IF NOT RECEIVING
- **JMP *-3**: WAIT FOR END OF RECORD
- **IOT 1444**: CLEAR 201 INTERFACE
- **LAC #440**: GET TERM RDY BIT & FRAME SIZE 8
- **IOT 1404**: SET INITIAL 201 INTERFACE STATE
- **LAC BP3**: GET PUNCH STATUS SWITCH
- **SNA**: SKIP IF PUNCH ACTIVE
- **JMP *+3**: PUNCH NOT ACTIVE
- **IOT 201**: SKIP ON PUNCH FLAG
- **JMP *-1**: WAIT FOR PUNCH FLAG
- **LAC BT1**: GET TELEPRINTER STATUS SWITCH
- **SNA**: SKIP IF TELEPRINTER ACTIVE
- **JMP *+3**: TELEPRINTER NOT ACTIVE
- **IOT 401**: SKIP ON TELEPRINTER FLAG
- **JMP *-1**: WAIT FOR TELEPRINTER FLAG
- **IOT 612**: READ DISPLAY STATUS
- **AND #7400**: GET DISPLAY FLAG BITS
- **SNA+CLA**: SKIP IF DISPLAY STOPPED
- **JMP *-3**: WAIT FOR DISPLAY TO STOP
- **JMS PS1**: CLEAR PUSH BUTTONS
- **IOT 4**: DISABLE CLOCK
- **IOT 3302**: CLEAR ALL FLAGS
- **DZM BP3**: INDICATE PUNCH IDLE
- **DZM BT1**: INDICATE TELEPRINTER IDLE
- **DZM PE+1**: DISABLE OPERATION OF PUSH BUTTONS
- **DZM DE+1**: DISABLE DISPLAY INTERRUPTS
- **DZM DWV**: CLEAR TRANSLATION VALUE
- **DZM NA+2**: UNLOCK N.A
- **DZM NC+2**: UNLOCK N.C
- **DZM STRD+2**: UNLOCK S.TRD
- **DZM STRR+2**: UNLOCK S.TRR
- **DZM SLY+2**: UNLOCK S.LY
- **DZM SLX+2**: UNLOCK S.LX
- **LAW 10**: GET TELEPRINTER MASK
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC STATUS</td>
<td>ALLOCATE TELEPRINTER ONLY</td>
</tr>
<tr>
<td>DAC BTTY2</td>
<td>SET BTTY ALLOCATION MASK</td>
</tr>
<tr>
<td>LAC TQ</td>
<td>GET POINTER TO END OF TASK QUEUE</td>
</tr>
<tr>
<td>DAC TQ+1</td>
<td>RESET INPUT POINTER</td>
</tr>
<tr>
<td>DAC TQ+2</td>
<td>RESET OUTPUT POINTER</td>
</tr>
<tr>
<td>DAC BRS</td>
<td>SET RECORD SEEK SWITCH</td>
</tr>
<tr>
<td>DZM BRO</td>
<td>INDICATE NEW RECORD NEEDED</td>
</tr>
<tr>
<td>LAC BPQ</td>
<td>GET POINTER TO END OF PUNCH BUFFER</td>
</tr>
<tr>
<td>DAC BPQ+1</td>
<td>RESET INPUT POINTER</td>
</tr>
<tr>
<td>DAC BPQ+2</td>
<td>RESET OUTPUT POINTER</td>
</tr>
<tr>
<td>LAC BKQ</td>
<td>GET POINTER TO END OF KB BUFFER</td>
</tr>
<tr>
<td>DAC BKQ+1</td>
<td>RESET INPUT POINTER</td>
</tr>
<tr>
<td>DAC BKQ+2</td>
<td>RESET OUTPUT POINTER</td>
</tr>
<tr>
<td>LAC BTQ</td>
<td>GET POINTER TO END OF TP BUFFER</td>
</tr>
<tr>
<td>DAC BTQ+1</td>
<td>RESET INPUT POINTER</td>
</tr>
<tr>
<td>DAC BTQ+2</td>
<td>RESET OUTPUT POINTER</td>
</tr>
<tr>
<td>LAC =DN</td>
<td>GET ADDRESS OF NULL DISPLAY SERVICE</td>
</tr>
<tr>
<td>DAC DPT</td>
<td>SET NULL LIGHT PEN SERVICE</td>
</tr>
<tr>
<td>LAC =PN</td>
<td>GET ADDRESS OF NULL PB SERVICE</td>
</tr>
<tr>
<td>DAC PTT</td>
<td>SET NULL PUSH BUTTON SERVICE</td>
</tr>
<tr>
<td>LAW 3000</td>
<td>GET POP INSTRUCTION</td>
</tr>
<tr>
<td>DAC XP</td>
<td>INHIBIT TRACKING PROCESS</td>
</tr>
<tr>
<td>LAC =D</td>
<td>GET ADDRESS OF HIGHEST ACTIVE LEVEL</td>
</tr>
<tr>
<td>DAC DHAL+7</td>
<td>REMOVE EVERYTHING FROM HAL</td>
</tr>
<tr>
<td>LAW PDP1</td>
<td>GET ADDRESS OF PUSH DOWN LIST</td>
</tr>
<tr>
<td>IOT 645</td>
<td>SET PUSH DOWN POINTER</td>
</tr>
<tr>
<td>LAW 7763</td>
<td>GET INITIAL DISPLAY CONDITIONS</td>
</tr>
<tr>
<td>IOT 665</td>
<td>SET INITIAL DISPLAY CONDITIONS</td>
</tr>
<tr>
<td>LAW 4000</td>
<td>GET BREAK FIELD 1 PARAMETER</td>
</tr>
<tr>
<td>IOT 705</td>
<td>LOAD BREAK FIELD</td>
</tr>
<tr>
<td>LAW D</td>
<td>GET ADDRESS OF SYSTEM DISPLAY FILE</td>
</tr>
<tr>
<td>IOT 1605</td>
<td>START DISPLAY</td>
</tr>
<tr>
<td>LAC =BFENQ</td>
<td>GET ENQUIRY CHARACTER</td>
</tr>
<tr>
<td>JMS BFENGS</td>
<td>INITIALIZE 201 TASKS</td>
</tr>
<tr>
<td>IOT 42</td>
<td>ENABLE INTERRUPTS</td>
</tr>
<tr>
<td>LAW BFENQ</td>
<td>GET ENQUIRY CHARACTER</td>
</tr>
<tr>
<td>JMS B+FO</td>
<td>SEND ATTENTION INTERRUPT</td>
</tr>
<tr>
<td>NOP</td>
<td>DATA SET NOT CONNECTED</td>
</tr>
<tr>
<td>E1</td>
<td>CLEAR POINTER TO DIAGNOSTIC LEVEL</td>
</tr>
<tr>
<td>DZM 26</td>
<td>CLEAR POINTER TO DIAGNOSTIC LEAF</td>
</tr>
<tr>
<td>DZM 27</td>
<td>CLEAR POINTER TO DIAGNOSTIC LEAF</td>
</tr>
</tbody>
</table>
JMS S.TL  CREATE TITLE LEAF
JMP E2  USE TELETYPETO ONLY
DAC E3  SAVE POINTER TO TITLE LEAF
LAC =EF  GET ADDRESS OF TEXT LIST
JMS L.D  CREATE TITLE LEAF
JMP E2  USE TELETYPETO ONLY
LMQ  SET UP PARAMETER
LAC E3  GET POINTER TO TITLE LEVEL
JMS S.TI  INSERT TITLE LEAF
JMP E2  USE TELETYPETO ONLY
LAC =370  GET Y TITLE COORDINATE
LMQ  SET UP PARAMETER
LAC E3  GET POINTER TO TITLE LEVEL
JMS S.LY  SET Y TITLE COORDINATE
LAW -144  GET X TITLE COORDINATE
LMQ  SET UP PARAMETER
LAC E3  GET POINTER TO TITLE LEVEL
JMS S.LX  SET X TITLE COORDINATE
LAW 500  GET SCALE X2 PARAMETER
LMQ  SET UP PARAMETER
LAC E3  GET POINTER TO TITLE LEVEL
JMS S.LP  SET TITLE SCALE
LAC E3  GET POINTER TO TITLE LEVEL
LMQ  SET UP PARAMETER
LAC =DHAL  GET ADDRESS OF HIGHEST ACTIVE LEVEL
JMS S.TI  INSERT TITLE LEVEL
JMP E2  USE TELETYPETO ONLY
JMS S.TL  CREATE DIAGNOSTIC LEVEL
JMP E2  USE TELETYPETO ONLY
DAC 26  SET POINTER TO DIAGNOSTIC LEVEL
LAC =200  GET Y DIAGNOSTIC DISPLACEMENT
LMQ  SET UP PARAMETER
LAC 26  GET ADDRESS OF DIAGNOSTIC LEVEL
JMS S.LY  TRANSLATE LEVEL IN Y DIRECTION
LAW -400  GET X DIAGNOSTIC DISPLACEMENT
LMQ  SET UP PARAMETER
LAC 26  GET ADDRESS OF DIAGNOSTIC LEVEL
JMS S.LX  TRANSLATE LEVEL IN X DIRECTION
LAW 500  GET SCALE X2 PARAMETER
LMQ  SET UP PARAMETER
LAC 26  GET ADDRESS OF DIAGNOSTIC LEVEL
JMS S.LP  SET DIAGNOSTIC SCALE
LAC 26  GET ADDRESS OF DIAGNOSTIC LEVEL
LMQ  SET UP PARAMETER
LAC =DHAL  GET ADDRESS OF HIGHEST ACTIVE LEVEL
JMS S.TI  INSERT DIAGNOSTIC LEVEL
JMP E2  DISPLAY STORAGE EXCEEDED
LAC 25  GET ADDRESS OF TEXT LIST
SZA  SKIP IF DISP STORAGE BEING CLEARED
JMS L.D  CREATE DIAGNOSTIC LEAF
JMP E2  USE TELETYPewriter ONLY
DAC 27  SET POINTER TO DIAGNOSTIC LEAF
LMQ  SET UP PARAMETER
LAC 26  GET ADDRESS OF DIAGNOSTIC LEVEL
JMS S.TI  INSERT DIAGNOSTIC LEAF
NOP  USE TELETYPewriter ONLY
E2
  LAC 25  GET POINTER TO TEXT LIST
SNA  SKIP IF COMMENT TO BE TYPED
JMP IDLE  BEGIN IDLE-TIME TASK
LAC =747575  GET TELEPRINTER POSITIONING CODE
JMS B.T  POSITION TELEPRINTER
LAC 25  GET ADDRESS OF TEXT LIST
JMS L.T  TYPE DIAGNOSTIC
LAC =747575  GET TELEPRINTER POSITIONING CODE
JMS B.T  POSITION TELEPRINTER
JMP IDLE  BEGIN IDLE-TIME TASK
EF
  $DC 11
  $TEXT "SEL EXECUTIVE SYSTEM (01)"
DISPLAY STRUCTURE STORAGE MANAGER

STORE SEQU 12000

LOWER LIMIT OF DISPLAY STORAGE

B

$DC 0
CMA
TAD =1
DAC T1
DAC T2
LAC =STORE

FORM 1'S COMP OF NUMBER OF BLOCKS
FORM 2'S COMP OF NUMBER OF BLOCKS
INITIALIZE COUNTER
STORE VALUE FOR resetting COUNTER
GET LOWER LIMIT OF DISPLAY STORAGE

B1

DAC T3
DAC T4
SAD =20000
JMP* B
LAC* T4
SNA
JMP
LAC T2
DAC T1
LAC T4
TAD
JMP
JMP
LAC B2
SNA
JMP
DAC T2
LAC T1
LAC T4
TAD
JMP

REinitialize COUNTER
GET ADDRESS OF UNAVAILABLE BLOCK
FORM ADDRESS OF NEXT BLOCK
PROCEED WITH NEXT CANDIDATE
INCREMENT COUNTER & SKIP IF DONE
PREPARE TO ADD ANOTHER BLOCK
GET ADDRESS OF ACQUIRED STORAGE
INDICATE SUCCESS
RETURN

B2

ISZ T1
JMP ++4
LAC T3
ISZ B
JMP
LAC T4
TAD =4
JMP B1+1

INCearation COUNTER & SKIP IF DONE
PREPARE TO ADD ANOTHER BLOCK
GET ADDRESS OF ACQUIRED STORAGE
INDICATE SUCCESS
RETURN

B3

$DC 0
LAC =1
JMS B
JMP B3
ISZ B3
JMP B3

GET SINGLE BLOCK PARAMETER
FIND SINGLE BLOCK
NO SINGLE BLOCK AVAILABLE
INDICATE SUCCESS
RETURN

B4

$DC 0
LAC =2

GET DOUBLE BLOCK PARAMETER
JMS B
JMP* B4
ISZ B4
JMP* B4

FIND DOUBLE BLOCK
NO DOUBLE BLOCK AVAILABLE
INDICATE SUCCESS
RETURN
STITLE

WORD QUEUE MANAGER

QC
IOT 2
JMS QS
LAC* QP
DAC* QIP
DAC* QQP
IOT 42
JMP* Q.C

DISABLE INTERRUPTS
SET CONTROL POINTERS
GET POINTER TO END OF QUEUE
SET INPUT POINTER
SET OUTPUT POINTER
ENABLE INTERRUPTS
RETURN

QA
IOT 2
JMS QA1
SKP
ISZ Q.A
IOT 42
JMP* Q.A

DISABLE INTERRUPTS
ADD WORD TO QUEUE
OVERFLOW
INDICATE SUCCESS
ENABLE INTERRUPTS
RETURN

QI
IOT 2
JMS QS
LAC* QQP
DAC 23
TAD = -3
SAD QP
LAC* QP
SAD* QP
SKP
TAD = 2
SAD* QIP
JMP * + 5
OVERFLOW
DAC* QQP
DAC* 23
ISZ Q.I
IOT 42
JMP* Q.I

DISABLE INTERRUPTS
SET CONTROL POINTERS
GET OUTPUT POINTER
SAVE OUTPUT POINTER
SUBTRACT 3
SKIP IF NO WRAP-AROUND
GET POINTER TO END OF QUEUE
SKIP IF NO WRAP-AROUND
CHECK FOR OVERFLOW
FORM NEW OUTPUT POINTER
SKIP IF NO OVERFLOW
OVERFLOW
SET NEW OUTPUT POINTER
GET VALUE TO BE STORED
STORE VALUE IN QUEUE
INDICATE SUCCESS
ENABLE INTERRUPTS
RETURN

QF
IOT 2
JMS QF1
SKP
ISZ Q.F

DISABLE INTERRUPTS
FETCH WORD FROM QUEUE
QUEUE EMPTY
INDICATE SUCCESS
IOT 42
JMP* Q.F

ENABLE INTERRUPTS
RETURN

QA1
SDC 0
JMS QS
LAC* QIP
JMS QINC
SAD* QOP
JMP* QA1
DAC* QIP
DAC 23
LACQ
DAC* 23
ISZ QA1
JMP* QA1

SET CONTROL POINTERS
GET INPUT POINTER
INCREMENT
SKIP IF NO OVERFLOW
OVERFLOW
SET NEW INPUT POINTER
SAVE COPY OF POINTER
GET WORD TO BE STORED
STORE WORD IN QUEUE
INDICATE SUCCESS
RETURN

QF1
SDC 0
JMS QS
LAC* QOP
SAD* QIP
JMP* QF1
JMS QINC
DAC* QOP
DAC 23
LAC 23
ISZ QF1
JMP* QF1

SET CONTROL POINTERS
GET OUTPUT POINTER
SKIP IF QUEUE NOT EMPTY
QUEUE EMPTY
INCREMENT
SET NEW OUTPUT POINTER
SAVE COPY OF POINTER
GET WORD FROM QUEUE
INDICATE SUCCESS
RETURN

QS
SDC 0
DAC QP
TAD #1
DAC QIP
TAD #1
DAC QOP
JMP* QS

SET POINTER TO QUEUE
COMPUTE ADDRESS OF NEXT LOCATION
SET POINTER TO INPUT POINTER
COMPUTE ADDRESS OF NEXT LOCATION
SET POINTER TO OUTPUT POINTER
RETURN

QINC
SDC 0
SAD* QP
LAC QQP
TAD #1

SKIP IF NOT END OF QUEUE
WRAP AROUND TO BEGINNING OF QUEUE
INCREMENT
JMP QC
RETURN
$TITLE TASK SCHEDULER

TS
IOT 2
AND =77777
JMS TII
IOT 42
JMP* T.S

TP
LAW 17776
TAD T.*P
JMS T.*S

TF
IOT 2
JMS TIO
DAC 23
RAL
SZL
JMP TF1
SPA
JMP TF2
IOT 42
JMP* 23

TF1
JMS TIO
LMO
JMS TIO
JMP TF1-2

TF2
JMS TIO
AND STATUS
SNA
JMP TF3

TF3
LAC 23
JMS TII
LAC T9+2
JMS TII
IOT 42
JMP TF

LAC T9+2
XOR STATUS
DAC STATUS
JMP TF1-2
<table>
<thead>
<tr>
<th>TA</th>
<th>AND =17777</th>
<th>TRUNCATE HIGH ORDER BITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOT 2</td>
<td>DISABLE INTERRUPTS</td>
<td></td>
</tr>
<tr>
<td>DAC 23</td>
<td>SAVE ALLOCATION MASK</td>
<td></td>
</tr>
<tr>
<td>LAC T-A</td>
<td>GET ADDRESS OF RETURN</td>
<td></td>
</tr>
<tr>
<td>AND =77777</td>
<td>TRUNCATE HIGH ORDER BITS</td>
<td></td>
</tr>
<tr>
<td>XOR =200000</td>
<td>INDICATE ALLOCATION DELAY</td>
<td></td>
</tr>
<tr>
<td>JMS TII</td>
<td>PUT TASK ADDRESS ON QUEUE</td>
<td></td>
</tr>
<tr>
<td>LAC 23</td>
<td>GET ALLOCATION MASK</td>
<td></td>
</tr>
<tr>
<td>JMS TII</td>
<td>PUT ALLOCATION MASK ON QUEUE</td>
<td></td>
</tr>
<tr>
<td>JMP TF+1</td>
<td>GET ANOTHER TASK</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TR</th>
<th>CMA</th>
<th>COMPLEMENT RELEASE MASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND STATUS</td>
<td>MODIFY ALLOCATION STATUS</td>
<td></td>
</tr>
<tr>
<td>DAC STATUS</td>
<td>STORE NEW ALLOCATION STATUS</td>
<td></td>
</tr>
<tr>
<td>JMP* T.R</td>
<td>RETURN</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TL</th>
<th>DAC T1</th>
<th>SAVE AC CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAW 17776</td>
<td>LOAD AC WITH -2</td>
<td></td>
</tr>
<tr>
<td>TAD T-L</td>
<td>FORM ADDRESS OF SUBROUTINE ENTRY</td>
<td></td>
</tr>
<tr>
<td>DAC T2</td>
<td>SAVE ADDRESS OF SUBROUTINE ENTRY</td>
<td></td>
</tr>
<tr>
<td>LAC* T-L</td>
<td>GET SAVED RETURN POINTER</td>
<td></td>
</tr>
<tr>
<td>SZA</td>
<td>SKIP IF SUBROUTINE ENTERABLE</td>
<td></td>
</tr>
<tr>
<td>JMP TL1</td>
<td>RESCHEDULE SUBROUTINE CALL</td>
<td></td>
</tr>
<tr>
<td>LAC* T2</td>
<td>GET RETURN POINTER</td>
<td></td>
</tr>
<tr>
<td>DAC* T-L</td>
<td>SAVE AND LOCK SUBROUTINE</td>
<td></td>
</tr>
<tr>
<td>LAC T1</td>
<td>RESTORE AC CONTENTS</td>
<td></td>
</tr>
<tr>
<td>ISZ T-L</td>
<td>ADVANCE PAST SAVED RETURN POINTER</td>
<td></td>
</tr>
<tr>
<td>JMP* T.L</td>
<td>RETURN</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TL1</th>
<th>CLC</th>
<th>LOAD AC WITH -1</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAD* T2</td>
<td>FORM ADDRESS OF SUBROUTINE CALL</td>
<td></td>
</tr>
<tr>
<td>AND =77777</td>
<td>TRUNCATE HIGH ORDER BITS</td>
<td></td>
</tr>
<tr>
<td>XOR =400000</td>
<td>INDICATE REENTRY DELAY</td>
<td></td>
</tr>
<tr>
<td>IOT 2</td>
<td>DISABLE INTERRUPTS</td>
<td></td>
</tr>
<tr>
<td>JMS TII</td>
<td>PUT TASK ADDRESS ON QUEUE</td>
<td></td>
</tr>
<tr>
<td>LACQ</td>
<td>GET CONTENTS OF MQ</td>
<td></td>
</tr>
<tr>
<td>JMS TII</td>
<td>PUT ON TASK QUEUE</td>
<td></td>
</tr>
<tr>
<td>LAC T1</td>
<td>RESTORE AC CONTENTS</td>
<td></td>
</tr>
<tr>
<td>JMS TII</td>
<td>PUT ON TASK QUEUE</td>
<td></td>
</tr>
<tr>
<td>JMP TF+1</td>
<td>GET A NEW TASK</td>
<td></td>
</tr>
<tr>
<td>TU</td>
<td>DAC T1</td>
<td>SAVE AC CONTENTS</td>
</tr>
<tr>
<td></td>
<td>LAC* T.U</td>
<td>GET ADDRESS OF SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td>TAD =2</td>
<td>FORM ADDRESS OF SAVED RETURN</td>
</tr>
<tr>
<td></td>
<td>DAC T2</td>
<td>SAVE ADDRESS OF SAVED RETURN</td>
</tr>
<tr>
<td></td>
<td>LAC* T2</td>
<td>GET SAVED RETURN</td>
</tr>
<tr>
<td></td>
<td>DAC T3</td>
<td>SAVE TEMPORARILY</td>
</tr>
<tr>
<td></td>
<td>DZM* T2</td>
<td>UNLOCK SUBROUTINE</td>
</tr>
<tr>
<td></td>
<td>LAC T1</td>
<td>RESTORE AC CONTENTS</td>
</tr>
<tr>
<td></td>
<td>JMP* T3</td>
<td>RETURN FROM SUBROUTINE</td>
</tr>
</tbody>
</table>

| TV | SDC 0 | SAVE AC CONTENTS |
|    | DAC T1 | GET POINTER TO SUBROUTINE |
|    | LAC* TV | SAVE POINTER TO SUBROUTINE |
|    | DAC T2 | FORM POINTER TO SAVED RETURN |
|    | ISZ TV | GET POINTER TO SAVED RETURN |
|    | LAC TV | SIMULATE CALL TO T.L |
|    | DAC T.L | FAKE AN ENTRY TO T.L |
|    | JMP TL+4 | |

| TIO | SDC 0 | GET OUTPUT POINTER |
|     | LAC TQ+2 | SKIP IF TASK QUEUE NOT EMPTY |
|     | SAD TQ+1 | TASK QUEUE EMPTY |
|     | JMP EE | INCREMENT |
|     | JMS TI | |
|     | DAC TQ+2 | STORE NEW OUTPUT POINTER |
|     | LAC* TQ+2 | GET WORD FROM TASK QUEUE |
|     | JMP TIO | RETURN |

| TII | SDC 0 | SAVE VALUE TO BE STORED |
|     | DAC 24 | GET INPUT POINTER |
|     | LAC TQ+1 | INCREMENT |
|     | JMS TI | STORE NEW INPUT POINTER |
|     | DAC TQ+1 | SKIP IF NO TASK QUEUE OVERFLOW |
|     | SAD TQ+2 | TASK QUEUE OVERFLOW |
|     | JMP EQ | |
|     | LAC 24 | GET VALUE TO BE STORED |
|     | DAC* TQ+1 | PUT IN TASK QUEUE |
|     | JMP TII | RETURN |

| TI | SDC 0 | |
|    | | |
SAD TQ
LAC = TQ+2
TAD = 1
JMP+ TI

TQ SDC **200
SDS 200

SKIP IF NO WRAP-AROUND
GET ADDRESS BEFORE FIRST DATA WORD
INCREMENT POINTER
RETURN
<table>
<thead>
<tr>
<th>ASSEMBLER MACHINE CODE</th>
<th>ASSEMBLER MACHINE CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CB6</td>
<td></td>
</tr>
<tr>
<td>GCLL</td>
<td></td>
</tr>
<tr>
<td>LRS 14</td>
<td></td>
</tr>
<tr>
<td>ALS 3</td>
<td></td>
</tr>
<tr>
<td>LRS 6</td>
<td></td>
</tr>
<tr>
<td>ALS 3</td>
<td></td>
</tr>
<tr>
<td>LLS 11</td>
<td></td>
</tr>
<tr>
<td>DAC T1</td>
<td></td>
</tr>
<tr>
<td>LLS 6</td>
<td></td>
</tr>
<tr>
<td>ALS 3</td>
<td></td>
</tr>
<tr>
<td>LRS 6</td>
<td></td>
</tr>
<tr>
<td>ALS 3</td>
<td></td>
</tr>
<tr>
<td>AND #77</td>
<td></td>
</tr>
<tr>
<td>LRS 11</td>
<td></td>
</tr>
<tr>
<td>LAC* T1</td>
<td></td>
</tr>
<tr>
<td>JMP* C.B6</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ASSEMBLER MACHINE CODE</th>
<th>ASSEMBLER MACHINE CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C6A</td>
<td></td>
</tr>
<tr>
<td>AND #77</td>
<td></td>
</tr>
<tr>
<td>TAD =C6A1</td>
<td></td>
</tr>
<tr>
<td>DAC T1</td>
<td></td>
</tr>
<tr>
<td>LAC* T1</td>
<td></td>
</tr>
<tr>
<td>JMP* C.6A</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ASSEMBLER MACHINE CODE</th>
<th>ASSEMBLER MACHINE CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C6A1</td>
<td></td>
</tr>
<tr>
<td>$DC 260</td>
<td></td>
</tr>
<tr>
<td>$DC 261</td>
<td></td>
</tr>
<tr>
<td>$DC 262</td>
<td></td>
</tr>
<tr>
<td>$DC 263</td>
<td></td>
</tr>
<tr>
<td>$DC 264</td>
<td></td>
</tr>
<tr>
<td>$DC 265</td>
<td></td>
</tr>
<tr>
<td>$DC 266</td>
<td></td>
</tr>
<tr>
<td>$DC 267</td>
<td></td>
</tr>
<tr>
<td>$DC 270</td>
<td></td>
</tr>
<tr>
<td>$DC 271</td>
<td></td>
</tr>
<tr>
<td>$DC 301</td>
<td></td>
</tr>
<tr>
<td>$DC 302</td>
<td></td>
</tr>
<tr>
<td>$DC 303</td>
<td></td>
</tr>
<tr>
<td>$DC 304</td>
<td></td>
</tr>
<tr>
<td>$DC 305</td>
<td></td>
</tr>
<tr>
<td>$DC 306</td>
<td></td>
</tr>
</tbody>
</table>

**FORMAT CONVERTER**

USE ZEROs TO FILL HOLES
SHIFT DIGITS 2, 3, 4, & 5 INTO MQ
CONVERT DIGIT 2
SHIFT DIGIT 2 INTO MQ
CONVERT DIGIT 1
SHIFT DIGITS 0, 1, & 2 INTO AC
STORE HIGH ORDER DIGITS
SHIFT DIGITS 3 & 4 INTO AC
CONVERT DIGIT 5
SHIFT DIGIT 4 INTO MQ
CONVERT DIGIT 4
CONVERT DIGIT 3
SHIFT LOW ORDER DIGITS INTO MQ
GET HIGH ORDER DIGITS
RETURN

TRUNCATE HIGH ORDER BITS
ADD ADDRESS OF TABLE
SAVE TEMPORARILY
GET CONVERTED VALUE
RETURN
SDC 307
SDC 310
SDC 311
SDC 312
SDC 313
SDC 314
SDC 315
SDC 316
SDC 317
SDC 320
SDC 321
SDC 322
SDC 323
SDC 324
SDC 325
SDC 326
SDC 327
SDC 330
SDC 331
SDC 332
SDC 252
SDC 257
SDC 253
SDC 255
SDC 250
SDC 251
SDC 333
SDC 335
SDC 274
SDC 275
SDC 276
SDC 336
SDC 337
SDC 256
SDC 254
SDC 272
SDC 273
SDC 277
SDC 241
SDC 247
TRUNCATE HIGH ORDER BITS
ADD ADDRESS OF TABLE
SAVE TEMPORARILY
GET CONVERTED VALUE
RETURN

A-42
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 76
$DC 66
$DC 70
$DC 72
$DC 71
$DC 77
$DC 73
$DC 67
$DC 50
$DC 51
$DC 44
$DC 46
$DC 62
$DC 47
$DC 61
$DC 45
$DC 00
$DC 01
$DC 02
$DC 03
$DC 04
$DC 05
$DC 06
$DC 07
$DC 10
$DC 11
$DC 63
$DC 64
$DC 54
$DC 55
$DC 56
$DC 65
$DC 77
CCB        DAC T1             SAVE VALUE TO BE CONVERTED
          AND =1777            GET MAGNITUDE
          DAC T2             SAVE MAGNITUDE
          LAC T1             GET VALUE TO BE CONVERTED
          AND =2000            GET SIGN BIT
          SNA                 SKIP IF NEGATIVE
          JMP CCB1            DO NOT MODIFY MAGNITUDE
          LAC T2             GET MAGNITUDE
          CMA                 FORM 1'S COMPLEMENT
          TAD =1              FORM 2'S COMPLEMENT
          JMP* C.CB          RETURN
CCB1       LAC T2             GET CONVERTED VALUE
          JMP* C.CB          RETURN
CBC         SMA                 SKIP IF NEGATIVE
            JMP CBC1         DO NOT FORM NEGATIVE
CMA
AND #1777
TAD #0001
AND #3777
JMP* C+BC

FORM 1'S COMPLEMENT
GET MAGNITUDE
SET SIGN BIT & FORM 2'S COMPLEMENT
CLEAR ESCAPE/INTENSITY BIT
RETURN

CBC1
AND #1777
JMP* C+BC

CONVERT TO MODULO 2^10
RETURN
STITLE 201 DATAPHONE BUFFER MANAGER

BFdle $equ 220  DATA LINK ESCAPE
BFsyn $equ 26   SYNCHRONOUS IDLE
BFack $equ 6    POSITIVE ACKNOWLEDGEMENT
BFnak $equ 225  NEGATIVE ACKNOWLEDGEMENT
BFetot $equ 204 END OF TRANSMISSION
BFenq $equ 5    ENQUIRY
BFetb $equ 27   END OF TEXT BLOCK
BFetx $equ 3    END OF TEXT

* STATE BITS (LOW ORDER 5 BITS OF BFS):
* 01   ACK OUTSTANDING
* 02   LAST INPUT RECORD COMPLETELY RECEIVED
* 04   ACK OUTPUT PENDING
* 10   NAK OUTPUT PENDING
* 20   DATA OUTPUT PENDING

Bfi  Iot 1412 READ 201 STATUS
     and #1000 GET SET READY BIT
      snA  SKIP IF DATA SET CONNECTED
     jmp BFI     DATA SET NOT CONNECTED
     lac bfs   GET 201 TASK STATE
     and #2    GET INPUT RECORD AVAILABLE BIT
       snA  SKIP IF INPUT RECORD AVAILABLE
     jmp BFI2  WAIT FOR INPUT RECORD
     lac BFS1 GET FIRST RECEIVED CHARACTER
      saZ  SKIP IF USER RECORD
     jmp BFI2  WAIT FOR RECORD TO BE TYPED
     lac BFI0 GET CHARACTER FROM INPUT BUFFER
      dAC BFI3  SAVE INPUT CHARACTER
     isZ BFI0  INCREMENT INPUT POINTER
       smA  SKIP IF END OF RECORD
     jmp BFI1  RETURN
     lac BFS   GET 201 TASK STATE
      xor #6  FORM ACK PENDING STATE
     dAC BFS  SET NEW STATE
     lac =BFXMT GET ADDRESS OF TRANSMISSION TASK
     jms t+s SCHEDULE TRANSMISSION TASK
     lac BFI3 GET END OF RECORD CHARACTER
BFI1  ISZ B.FI  INDICATE SUCCESS
      JMP* B.FI  RETURN
      JMP BFI  GET CHARACTER FROM INPUT BUFFER
BFI2  JMS T.P  SCHEDULE PREVIOUS LOC & TERMINATE

BFO  DAC BFO3  SAVE CHARACTER TO BE BUFFERED
     IOT 1412  READ 201 STATUS
     AND =1000  GET SET READY BIT
     SNA  SKIP IF DATA SET CONNECTED
     JMP* B.FO  DATA SET NOT CONNECTED
     LAC BFS  GET 201 TASK STATE
     AND =21  GET DATA OUTPUT & ACK EXP BITS
     SZA  SKIP IF OUTPUT BUFFER IS FREE
     JMP BFO2  PUT CHARACTER INTO BUFFER LATER
     LAC BFO3  GET CHARACTER TO BE BUFFERED
     DAC* BFO1  PUT CHARACTER IN OUTPUT BUFFER
     ISZ BFO1  INCREMENT INPUT POINTER
     SMA  SKIP IF END-OF-RECORD CHARACTER
     JMP BFO1  RETURN
     LAC BFS  GET 201 TASK STATE
     XOR =20  SET DATA OUTPUT PENDING BIT
     DAC BFS  SET NEW 201 TASK STATE
     LAC =BFXMT  GET ADDRESS OF TRANSMISSION TASK
     JMS T.S  SCHEDULE TRANSMISSION TASK
BFO1  ISZ B.FO  INDICATE SUCCESS
      JMP* B.FO  RETURN
      JMP BFO+1  PUT CHARACTER IN BUFFER
BFO2  JMS T.P  SCHEDULE PREVIOUS LOC & TERMINATE

BFXMT  IOT 1412  READ 201 STATUS
     AND =60100  GET CAR DET, XMT REQ, CLR SEND BITS
     SZA  SKIP IF ABLE TO TRANSMIT
     JMP BFXMT4  RESCHEDULE BFXMT
     LAC BFS  GET 201 TASK STATE
     RAR  SHIFT ACK EXPECTED BIT INTO LINK
     SZL+RAR  SKIP IF ACK NOT EXPECTED
     JMP BFXMT4  RESCHEDULE BFXMT
     SZL+RAR  SKIP IF INPUT BUFFER EMPTY
     JMP BFXMT4  RESCHEDULE BFXMT
     SNL+RAR  SKIP IF ACK OUTPUT PENDING
JMP BFXMT1
LAC BFS
AND =33
DAC BFS
LAC =BFIB
DAC BFII
DAC BFIO
LAC =BFACKR
JMP BFXMT2

BFXMT1 SNL+RAR
JMP BFXMT2
LAC BFS
AND =27
DAC BFS
LAC =BFNAKR
JMP BFXMT3

BFXMT2 SNL
JMS T.F
LAC BFS
XOR =21
DAC BFS
LAC =BFACKR

BFXMT3 DAC BFIO
LAW -10
DAC BFC
LAW BF SYN
DAC 5
LAC =200005
DAC IFO
LAC =20000
IOT 1404
JMS T.F
JMP BFXMT4

BFXMT4 JMS T.P

BFACKR LAW BFACK
BFNAKR LAW BFNAK

BFTTY LAC BFTTY2
JMS T.A
IF1
LAC 4
GET RECEIVED CHARACTER
LRS 4
SHIFT INTO POSITION
AND =377
TRUNCATE HIGH ORDER BITS
HLT
STATE VARIABLE
SAD =BFSYN
SKIP IF NOT SYN
SKP
FIND NEXT SYN & CHANGE STATE
JMP IF16
IGNORE CHARACTER
IOT 1412
READ 201 STATUS
AND =2000
GET TEXT BIT
IOT 1404
CLEAR TEXT BIT
LAC =60000+IF11
GET JMP IF11 INSTRUCTION
DAC IF1+3
MODIFY INTERRUPT SERVICE
IOT 1442
CLEAR 201 FLAGS
JMP IR
RETURN FROM INTERRUPT
IF11
SAD =BFSYN
SKIP IF NOT SYN
JMP IF11-2
IGNORE SYN
SAD =BFDE
SKIP IF NOT DLE (EVEN PARITY)
JMP =3
BUFFER RECEIVED RECORD
LAC =740000
GET NOP INSTRUCTION
JMP IF11-3
MODIFY INTERRUPT SERVICE
LAC =60000+IF12
GET JMP IF12 INSTRUCTION
A-51

JMP IFI1-3
MODIFY INTERRUPT SERVICE

IFI2
SAD = BFDLE
MODIFY INTERRUPT SERVICE
JMP IFI3-2
SKIP IF NOT DLE (EVEN PARITY)
SAD = BFDLE-200
CHANGE STATE FOR NEXT CHARACTER
JMP IFI3-2
SKIP IF NOT DLE (ODD PARITY)
JMS BFIS
PUT CHARACTER IN BUFFER
JMP IFI1-2
CLEAR FLAGS AND RETURN
LAC = 600000+IFI3
GET JMP IFI3 INSTRUCTION
JMP IFI1-3
MODIFY INTERRUPT SERVICE

IFI3
SAD = BFDLE
MODIFY INTERRUPT SERVICE
JMP IFI31-3
SKIP IF NOT DLE (EVEN PARITY)
SAD = BFDLE-200
PUT DLE IN BUFFER
JMP IFI31-3
SKIP IF NOT DLE (ODD PARITY)
JMP IFI31-2
PUT DLE IN BUFFER
SAD = BFSYN
SKIP IF NOT SYN
JMP IFI31-2
IGNORE SYN
DAC BFEOR
SAVE END-OF-RECORD CHARACTER
SAD = BFACK
SKIP IF NOT ACK
JMP IFI31
CLEAR OUTPUT BUFFER
SAD = BFNAK
SKIP IF NOT NAK
JMP IFI32
RETRANSMIT LAST DATA RECORD
XOR = 760000
INDICATE END-OF-RECORD CHARACTER
JMS BFIS
PUT CHARACTER IN BUFFER
LAC = 600000+IFI4
GET JMP IFI4 INSTRUCTION
JMP IFI1-3
MODIFY INTERRUPT SERVICE
JMS BFIS
PUT DLE CHARACTER IN BUFFER
LAC = 600000+IFI2
GET JMP IFI2 INSTRUCTION
JMP IFI1-3
MODIFY INTERRUPT SERVICE

IFI31
LAC = BFOB
GET ADDRESS OF OUTPUT BUFFER
DAC BFOI
RESET INPUT POINTER
LAC BFS
GET 201 TASK STATE
AND = A6
INDICATE ACK NOT EXPECTED
DAC BFS
STORE NEW TASK STATE
LAC = 740000
GET NOP INSTRUCTION
JMP IFI1-3
MODIFY INTERRUPT SERVICE

IFI32
LAC BFS
GET 201 TASK STATE
XOR = 21
FORM STATE FOR RETRANSMISSION
DAC BFS
STORE NEW TASK STATE
LAC = BFMT
GET ADDRESS OF TRANSMISSION TASK
JMS TII
SCHEDULE TRANSMISSION
JMP IFI32-2
MODIFY INTERRUPT SERVICE
IFI4  CLL
AL5  10
DAC  BFCKR
LAC  =600000+IFI5
JMP  IF11-3

IFI5  XOR  BFCKR
SAD  BFCK
JMP  IF151
LAC  =BF1B
DAC  BF11
DAC  BF10
LAC  BFS
XOR  =10
DAC  BFS
JMP  IF14-3

IFI51  LAC  BFS
XOR  =2
DAC  BFS
LAC  BFEOB
JMS  BFENOS
LAC  BFIB
SNA
JMP  IF132-2
LAC  =BFTTY
JMP  IF14-2

IFI6  IOT  1412
AND  =2000
IOT  1404
JMP  IF11-2

IFO  HLT
SAD  =760000+BFSYN
JMP  IF01
SAD  =760000+BFDLE
JMP  IF03
SMA
JMP  IF04
JMP  IF05

IFO1  ISZ  BFC
JMP  IF02+2

PREPARE TO SHIFT ZEROS INTO AC
SHIFT HIGH ORDER CHECK INTO POSITION
SAVE HIGH ORDER BLOCK CHECK
GET JMP IF15 INSTRUCTION
MODIFY INTERRUPT SERVICE
FORM COMPLETE BLOCK CHECK
SKIP IF BAD RECORD
INDICATE INPUT BUFFER FULL
GET ADDRESS OF INPUT BUFFER
RESET INPUT POINTER
RESET OUTPUT POINTER
GET 201 TASK STATE
INDICATE NAK PENDING
SET NEW 201 TASK STATE
SCHEDULE TRANSMISSION TASK
GET 201 TASK STATE
INDICATE INPUT BUFFER FULL
SET NEW 201 TASK STATE
GET END-OF-RECORD CHARACTER
PROCESS ENQUIRY IF PRESENT
GET FIRST RECEIVED CHARACTER
SKIP IF UNSOLICITED RECORD
MODIFY INTERRUPT SERVICE
GET ADDRESS OF BYPASS TASK
SCHEDULE BYPASS TASK
READ 201 STATUS
GET TEXT BIT
CLEAR TEXT BIT
CLEAR FLAGS AND RETURN
STATE VARIABLE
SKIP IF NOT SYN
SYN SENT LAST TIME
SKIP IF NOT DLE
DLE SENT LAST TIME
SKIP IF END-OF-RECORD CHARACTER
TEXT CHARACTER SENT LAST TIME
ENTER BLOCK CHECK PROCEDURE
SKIP IF LAST SYN SENT
CLEAR FLAGS AND RETURN
A-53

```
LAC = BFDLE
SKP

IF02
LAW BFDLE
DAC 5
IOT 1442
JMP IR

IF03
LAC* BF00
JMS BFCKS
LAC BF00
SAD = BF0B
DZM BFCK
LAC* BF00
ISZ BF00
JMP IFO2+1

IF04
LAC* BF00
SAD = BFDLE
JMP IFO2
SAD = BFDLE-200
JMP IFO2

IF05
LAC = 600000+IFO6
DAC BF0
LAC BFCK
LRS 10
JMP IFO2+1

IF06
LAC = 600000+IFO7
DAC BF0
LAC BFCK
JMP IFO2+1

IF07
LAC = 600000+IFO8
DAC BF0
CLC
JMP IFO2+1

IF08
IOT 1412
AND = 20000
IOT 1404
GET INITIAL DLE
SET TRANSMIT IMAGE
GET DLE CHARACTER
SET TRANSMIT IMAGE
CLEAR 201 FLAGS
RETURN FROM INTERRUPT
GET CHARACTER FROM BUFFER
UPDATE BLOCK CHECK
GET OUTPUT POINTER
SKIP IF NOT FIRST CHARACTER
CLEAR BLOCK CHECK
GET CHARACTER FROM OUTPUT BUFFER
INCREMENT OUTPUT POINTER
TRANSMIT CHARACTER
GET CHARACTER FROM BUFFER
SKIP IF NOT DLE (EVEN PARITY)
PRECEDE WITH DLE
SKIP IF NOT DLE (ODD PARITY)
PRECEDE WITH DLE
SKIP IF END OF RECORD
SEND CHARACTER FROM BUFFER
TRUNCATE HIGH ORDER BITS
PROCESS ENQUIRY, IF PRESENT
PRECEDE WITH DLE
GET JMP IFO6 INSTRUCTION
MODIFY INTERRUPT SERVICE
GET BLOCK CHECK
SHIFT HIGH ORDER PART INTO POSITION
TRANSMIT HIGH ORDER BLOCK CHECK
GET JMP IFO7 INSTRUCTION
MODIFY INTERRUPT SERVICE
GET BLOCK CHECK
TRANSMIT LOW ORDER PART
GET JMP IFO8 INSTRUCTION
MODIFY INTERRUPT SERVICE
GET PAD CHARACTER
TRANSMIT PAD
READ 201 STATUS
GET XMT REQ BIT
CLEAR XMT REQ BIT
```
JMP IF02+2
CLEAR 201 FLAGS AND RETURN

BFENQ SDC 0
SAD =BFENQ
JMP ++4
SAD =BFENQ
SKIP IF NOT ENQUIRY
JMP BFENQ
SKIP IF NOT END-OF-TRANSMISSION
LAC =BFENQ
RETURN
DAC BFENQ
GET ADDRESS OF OUTPUT BUFFER
DAC BFEE
REJECT INPUT POINTER
DAC BFIE
GET ADDRESS OF INPUT BUFFER
DAC BFIO
RESET INPUT POINTER
DAC BFVE
RESET OUTPUT POINTER
DAC BFIF
DO NOT SCHEDULE BYPASS TASK
DAC BE0
STOP 201 TASK ACTIVITY
DAC =740000
GET NOP INSTRUCTION
DAC IFI+3
MODIFY INTERRUPT SERVICE
JMP BFENQ
RETURN

BFIS SDC 0
DAC* BFII
JMS BFCKS
DAC BFII
UPDATE BLOCK CHECK
DAC BFIO
GET INPUT POINTER
DAC BFIB
SKIP IF INPUT BUFFER NON-EMPTY
DAC BFCK
CLEAR BLOCK CHECK
DAC BFIF
INCREMENT INPUT POINTER
DAC BFIS
RETURN

BFCKS SDC 0
DAC 23
LAW -10
DAC 24
LAC BFCK
SAVE CHARACTER
DAC 24
SET COUNTER
LAC BFCK
GET FORMER BLOCK CHECK
DAC BFCK
ROTATE LOW ORDER BIT INTO LINK
DAC BFCK
STORE NEW LOW ORDER 15 BITS
DAC BFCK
PREPARE TO GET LOW ORDER CHAR BIT
LAC 23
GET CHARACTER REMAINS
DAC 23
SHIFT LOW ORDER BIT INTO MQ
DAC 23
STORE CHARACTER REMAINS
DAC 23
GET LOW ORDER CHARACTER BIT
SZA
CML
LAC BFCK
SZL
XOR =120001
ISZ 24
JMP BFCKS1
DAC BFCK
JMP BFCKS

BFIB $DS 200
BFOB $DS 200

SKIP IF NOT SET
OR CHECK BIT WITH CHARACTER BIT
GET LOW ORDER 15 BITS OF CHECK
SKIP IF LOW ORDER BIT WAS 0
INVERT FEEDBACK BITS
INCREMENT COUNT & SKIP IF DONE
PROCESS NEXT CHARACTER BIT
STORE NEW BLOCK CHECK
RETURN
STITLE

READER BUFFER MANAGER

BR
LAC BR0
SNA
JMP BR2
SAD BRI
JMP BR1
LAC+ BR0
ISZ BRI
SNA
DZM BRO
ISZ B.R
JMP* B.R

BR1
SAD =BRQ+200
JMP BR2
IOT 314
AND =1000
SNA
JMP BR2-1
DZM BRO
JMP B.R
RETURN
JMP BR
JMS T.P

BR2
LAC =BRQ
DAC BRI
DAC BRO
IOT 184
JMP BR2-1

IRD
IOT 314
AND =1000
SZA
JMP IRD1
IOT 112
SZA
JMP IRD2
LAC BRS
SZA
JMP IRD3
JMS BRS

GET OUTPUT POINTER
SKIP IF NOT START OF NEW RECORD
CLEAR BUFFER & START READER
SKIP IF BUFFER NOT EMPTY
WAIT FOR MORE INPUT
GET IMAGE FROM BUFFER
INCREMENT OUTPUT POINTER
SKIP IF NOT END OF RECORD
INDICATE NEW RECORD NEEDED
INDICATE SUCCESS
RETURN
SKIP IF NOT END OF BUFFER
CLEAR BUFFER & START READER
READ STATUS
GET READER OUT-OF-TAPE FLAG
SKIP IF READER OUT OF TAPE
SCHEDULE NEW ATTEMPT
INDICATE NEW RECORD NEEDED
RETURN
TRY AGAIN TO GET IMAGE
SCHEDULE NEW ATTEMPT
GET ADDRESS OF READER BUFFER
SET INPUT POINTER
SET OUTPUT POINTER
SELECT READER
SCHEDULE NEW ATTEMPT
READ STATUS
GET READER OUT-OF-TAPE FLAG
SKIP IF TAPE IS IN READER
READER OUT OF TAPE
READ READER BUFFER
SKIP IF BLANK TAPE
PUT IMAGE IN BUFFER
GET RECORD SEEK SWITCH
SKIP IF END OF RECORD
IGNORE BLANK TAPE
SET RECORD SEEK SWITCH
<table>
<thead>
<tr>
<th>BRQ</th>
<th>SDS 200</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRS</td>
<td>SDC 0</td>
</tr>
<tr>
<td>DZM* BRI</td>
<td>RECORD SEEK SWITCH</td>
</tr>
<tr>
<td>ISZ BRI</td>
<td>STORE END-OF-RECORD IMAGE</td>
</tr>
<tr>
<td>JMP IR</td>
<td>INCREMENT INPUT POINTER</td>
</tr>
<tr>
<td>IRD1</td>
<td>IOT 102</td>
</tr>
<tr>
<td></td>
<td>RETURN FROM INTERRUPT</td>
</tr>
<tr>
<td>IRD2</td>
<td>DAC* BRI</td>
</tr>
<tr>
<td></td>
<td>STORE IN READER BUFFER</td>
</tr>
<tr>
<td>ISZ BRI</td>
<td>INCREMENT INPUT POINTER</td>
</tr>
<tr>
<td>DZM BRS</td>
<td>CLEAR RECORD SEEK SWITCH</td>
</tr>
<tr>
<td>LAC BRI</td>
<td>GET INPUT POINTER</td>
</tr>
<tr>
<td>SAD = BRQ+200</td>
<td>SKIP IF NOT END OF BUFFER</td>
</tr>
<tr>
<td>JMP IR</td>
<td>RETURN FROM INTERRUPT</td>
</tr>
<tr>
<td>IRD3</td>
<td>IOT 104</td>
</tr>
<tr>
<td></td>
<td>SELECT READER</td>
</tr>
<tr>
<td>BRQ</td>
<td>SDS 200</td>
</tr>
<tr>
<td></td>
<td>RETURN FROM INTERRUPT</td>
</tr>
</tbody>
</table>
SAVE PUNCH IMAGE
START PUNCH, IF POSSIBLE
GET PUNCH IMAGE
SET UP PUNCH IMAGE AS PARAMETER
GET ADDRESS OF PUNCH BUFFER
PUT PUNCH IMAGE IN BUFFER
PUNCH BUFFER FULL
START PUNCH, IF POSSIBLE
INDICATE SUCCESS
RETURN
READ STATUS
GET PUNCH OUT-OF-TAPE FLAG
SKIP IF PUNCH CONTAINS TAPE
PUNCH OUT OF TAPE
PREPARE TO SCHEDULE NEXT LOCATION
TRY AGAIN TO PUT IMAGE IN BUFFER
SCHEDULE PREVIOUS LOC & TERMINATE

GET PUNCH STATUS SWITCH
SKIP IF PUNCH IS IDLE
PUNCH IS ACTIVE
GET ADDRESS OF PUNCH BUFFER
FETCH IMAGE FROM BUFFER
PUNCH BUFFER EMPTY
SELECT PUNCH
SET PUNCH STATUS SWITCH

GET PUNCH STATUS SWITCH
PUNCH STATUS SWITCH
RETURN

READ STATUS
GET PUNCH OUT-OF-TAPE FLAG
SKIP IF PUNCH CONTAINS TAPE
PUNCH OUT OF TAPE
GET ADDRESS OF PUNCH BUFFER
GET IMAGE FROM PUNCH BUFFER
PUNCH BUFFER EMPTY
SELECT PUNCH
JMP IR
RETURN FROM INTERRUPT
CLEAR PUNCH FLAG
INDICATE PUNCH IDLE
RETURN FROM INTERRUPT

BPO SDC ++100
SDS 100
A-60

STITLE

BK      LAC =BK0
        JMP BKF
        JMS Q.F
        DAC BKF
        JMS C+A6
        JMP+ B+K
        JMS BK
        JMP BKF

BK1   JMS T+P

IKB   IOT 312
      LMO
      LAC =BK0
      JMS GA1
      NOP
      JMP IR

BK0   SDC ++100
      SDS 100

KEYBOARD BUFFER MANAGER

GET ADDRESS OF KEYBOARD BUFFER
GET CHARACTER FROM KEYBOARD BUFFER
WAIT FOR MORE INPUT
SAVE ASCII FOR SYSTEM USAGE
CONVERT TO 6-BIT CODE
RETURN
TRY AGAIN TO RETURN CHARACTER
SCHEDULE NEW ATTEMPT
READ KEYBOARD BUFFER
SET UP PARAMETER
GET ADDRESS OF KEYBOARD BUFFER
PUT CHARACTER IN BUFFER
BUFFER FULL -- IGNORE CHARACTER
RETURN FROM INTERRUPT
TITLE

TELEPRINTER BUFFER MANAGER

BT
DAC BT5
LAC BT5
LMQ
LAC = BT9
JMS Q-A
JMP BT2
LAC BT1
SZA
JMP* B.T

BT5

SAVE TEMPORARILY
GET PACKED WORD TO BE BUFFERED
SET UP PARAMETER
GET ADDRESS OF TELEPRINTER BUFFER
PUT PACKED WORD INTO TP BUFFER
TRY AGAIN LATER
GET TELEPRINTER STATUS SWITCH
SKIP IF TELEPRINTER IDLE
RETURN

IOT 2
DISABLE INTERRUPTS
LAC B.T
GET RETURN ADDRESS
DAC 0
STORE INTERRUPT RETURN
JMS BT1
SET TELEPRINTER STATUS SWITCH

BT1
SDC 0
TELEPRINTER STATUS SWITCH
JMP ITP
FAKE A TELEPRINTER INTERRUPT
JMP BT+1
TRY AGAIN TO PUT CHAR IN BUFFER

BT2
JMS T.P
SCHEDULE NEW ATTEMPT

BT3
SDC 77
BT4
SDC 77

ITP
LAC BT4
SAD = 77
SKP
JMP ITP3
TYPE SECOND CHARACTER
LAC BT3
GET THIRD CHARACTER
SAD = 77
SKIP IF NOT NULL CHARACTER
SKP
JMP ITP4
TYPE THIRD CHARACTER
LAC = BT9
GET ADDRESS OF TELEPRINTER BUFFER
JMS QF1
GET PACKED WORD FROM TP BUFFER
JMP ITP1
CLEAR FLAG & RETURN
DAC BT3
SET UP THIRD CHARACTER
LRS 6
SHIFT SECOND CHARACTER INTO PLACE
DAC BT4
SET UP SECOND CHARACTER
LRS 6
SHIFT FIRST CHARACTER INTO PLACE

ITPI
AND = 77
SAD = 77

TRUNCATE HIGH ORDER BITS
SKIP IF NOT NULL CHARACTER
<table>
<thead>
<tr>
<th>Label</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP ITP</td>
<td>TYPE NEXT CHARACTER</td>
<td></td>
</tr>
<tr>
<td>TAD = C6A1</td>
<td>ADD ADDRESS OF 6-BIT TO ASCII TABLE</td>
<td></td>
</tr>
<tr>
<td>DAC 23</td>
<td>SAVE TEMPORARILY</td>
<td></td>
</tr>
<tr>
<td>LAC* = 23</td>
<td>GET CONVERTED ASCII VALUE</td>
<td></td>
</tr>
<tr>
<td>IOT 406</td>
<td>SEND CHARACTER TO TELEPRINTER</td>
<td></td>
</tr>
<tr>
<td>JMP 1R</td>
<td>RETURN FROM INTERRUPT</td>
<td></td>
</tr>
<tr>
<td>ITP2 IOT 402</td>
<td>CLEAR TELEPRINTER FLAG</td>
<td></td>
</tr>
<tr>
<td>DAC BT1</td>
<td>INDICATE TELEPRINTER IDLE</td>
<td></td>
</tr>
<tr>
<td>JMP 1R</td>
<td>RETURN FROM INTERRUPT</td>
<td></td>
</tr>
<tr>
<td>ITP3 DAC 23</td>
<td>SAVE TEMPORARILY</td>
<td></td>
</tr>
<tr>
<td>LAC = 77</td>
<td>GET NULL CHARACTER</td>
<td></td>
</tr>
<tr>
<td>DAC BT4</td>
<td>STORE AS SECOND CHARACTER</td>
<td></td>
</tr>
<tr>
<td>LAC 23</td>
<td>GET CHARACTER TO BE TYPED</td>
<td></td>
</tr>
<tr>
<td>JMP ITP1</td>
<td>TYPE SECOND CHARACTER</td>
<td></td>
</tr>
<tr>
<td>ITP4 DAC 23</td>
<td>SAVE TEMPORARILY</td>
<td></td>
</tr>
<tr>
<td>LAC = 77</td>
<td>GET NULL CHARACTER</td>
<td></td>
</tr>
<tr>
<td>DAC BT3</td>
<td>STORE AS THIRD CHARACTER</td>
<td></td>
</tr>
<tr>
<td>LAC 23</td>
<td>GET CHARACTER TO BE TYPED</td>
<td></td>
</tr>
<tr>
<td>JMP ITP1</td>
<td>TYPE THIRD CHARACTER</td>
<td></td>
</tr>
</tbody>
</table>

**Table:**
- **SDC:** Entry point for the program.
- **SDS:** Entry point for the program.

---

A-62
STITLE NONBUFFERED I/O MANAGER

NA JMS TV
SDC N.A
SDC 0
AND =77
IOT 1103
IOT 1304
DZM IAD1
LAC IAD1
SNA
JMP NA2
LAC NA3
JMS T.U
SDC NA
JMP NA1
NA2 JMS T.P

NC JMS TV
SDC N.C
SDC 0
DAC 7
IOT 44
DZM ICK+1
LAC ICK+1
SNA
JMP NC1
JMS T.U
SDC NC
JMP =-5
NC1 JMS T.P

ND1 IOT 5101
JMP* N.D1
RETURN

ND2 IOT 5102
JMP* N.D2
RETURN

ND3 IOT 5104
JMP* N.D3
RETURN

PROTECT AGAINST REENTRY
TRUNCATE HIGH ORDER BITS
SELECT A/D CONVERTER CHANNEL
SELECT A/D CONVERTER
CLEAR CONVERSION SWICTH
GET CONVERSION SWITCH
SKIP IF CONVERSION COMPLETE
WAIT FOR CONVERSION TO BE COMPLETED
GET CONVERTED VALUE
UNLOCK N.A
CHECK FOR CONVERSION COMPLETE
SCHEDULE CONVERSION CHECK

PROTECT AGAINST REENTRY
SET CLOCK INTERVAL
ENABLE CLOCK
CLEAR CLOCK SWITCH
GET CLOCK SWITCH
SKIP IF TIME INTERVAL HAS ELAPSED
WAIT A LITTLE LONGER
UNLOCK N.C
CHECK ELAPSED TIME
SCHEDULE A LATER CHECK

SELECT D/A CONVERTER #1
RETURN

SELECT D/A CONVERTER #2
RETURN

SELECT D/A CONVERTER #3
RETURN
READ A/D CONVERTER
STORE CONVERTED VALUE
SET CONVERSION SWITCH
CONVERSION SWITCH
RETURN FROM INTERRUPT

READ A/D CONVERTER
STORE CONVERTED VALUE
SET CONVERSION SWITCH
CONVERSION SWITCH
RETURN FROM INTERRUPT

READ A/D CONVERTER
STORE CONVERTED VALUE
SET CONVERSION SWITCH
CONVERSION SWITCH
RETURN FROM INTERRUPT

READ A/D CONVERTER
STORE CONVERTED VALUE
SET CONVERSION SWITCH
CONVERSION SWITCH
RETURN FROM INTERRUPT
STITLE

PUSH BUTTON PROCESSOR

PT
SNA
LAC =PN
DAC PTT
JMP* P.T

PE
JMS *+1
SDC 0
JMP* P.E

PD
DZM PE+1
JMP* P.D

PR
IOT 631
JMP* P.R

PS
IOT 2
JMS PS1
IOT 42
JMP* P.S

PN
JMS P.E
JMS T.F

PS1
SDC 0
DAC PRG
LRS 6
AND =77
TAD =200
IOT 765
LLS 6
AND =77
TAD =300
IOT 765
JMP* PS1

IPB
LAC PE+1
SNA
JMP IPB1

SKIP IF NOT NULL TASK
GET ADDRESS OF NULL TASK
SAVE ADDRESS OF PUSH BUTTON SERVICE
RETURN

SET PUSH BUTTON ENABLE SWITCH
PUSH BUTTON ENABLE SWITCH
RETURN

CLEAR PUSH BUTTON ENABLE SWITCH
RETURN

READ PUSH BUTTONS
RETURN

DISABLE INTERRUPTS
SET PUSH BUTTONS
ENABLE INTERRUPTS
RETURN

ENABLE MANUAL OPN OF PUSH BUTTONS
TERMINATE TASK

STORE NEW PUSH BUTTON STATUS
SHIFT BITS 0-5 INTO POSITION
TRUNCATE HIGH ORDER BITS
SET BITS 0-5 ENABLE BIT
SET PUSH BUTTONS 0-5

SHIFT BITS 6-11 INTO POSITION
TRUNCATE HIGH ORDER BITS
SET BITS 6-11 ENABLE BITS
SET PUSH BUTTONS 6-11
RETURN

GET PUSH BUTTON ENABLE SWITCH
SKIP IF PUSH BUTTONS ARE ENABLED
RESTORE PUSH BUTTON STATUS
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAC PTT</td>
<td>GET ADDRESS OF PUSH BUTTON SERVICE</td>
</tr>
<tr>
<td>AND #77777</td>
<td>TRUNCATE HIGH ORDER BITS</td>
</tr>
<tr>
<td>JMS TII</td>
<td>SCHEDULE PUSH BUTTON SERVICE</td>
</tr>
<tr>
<td>IOT 631</td>
<td>READ PUSH BUTTONS</td>
</tr>
<tr>
<td>DAC PRG</td>
<td>MODIFY PUSH BUTTON STATUS WORD</td>
</tr>
<tr>
<td>DZM PE+1</td>
<td>DISABLE PUSH BUTTONS</td>
</tr>
<tr>
<td>JMP IR</td>
<td>RETURN FROM INTERRUPT</td>
</tr>
<tr>
<td>IPBI LAC PRG</td>
<td>GET FORMER PUSH BUTTON STATUS</td>
</tr>
<tr>
<td>JMS PSI</td>
<td>SET PUSH BUTTONS</td>
</tr>
<tr>
<td>JMP IR</td>
<td>RETURN FROM INTERRUPT</td>
</tr>
</tbody>
</table>
$TITLE

DISPLAY COMMUNICATOR

DE
JMS **1
SDC 0
JMP* D.E

SET DISPLAY INT ENABLE SWITCH
DISPLAY INT ENABLE SWITCH
RETURN

DD
DZM DE+1
JMP* D.D

CLEAR INT ENABLE SWITCH
RETURN

DP
SNA
LAC =DN
DAC DPT
JMP* D.P

SKIP IF NOT NULL SERVICE
GET ADDRESS OF NULL SERVICE
STORE ADDRESS OF SERVICE TASK
RETURN

DA
LAC DS1
LLS 14
AND =70000
XOR DSA
JMP* D.A

GET STATUS WORD 1
SHIFT BREAK FIELD INTO POSITION
REMOVE ALL BUT BREAK FIELD
FORM 15-BIT ADDRESS
RETURN

DY
LAC DS2
LLS 3
AND =10000
XOR DSY
TAD =-1000
JMP* D.Y

GET STATUS WORD 2
SHIFT HIGH ORDER BIT INTO POSITION
REMOVE OTHER BITS
FORM 13-BIT Y COORDINATE
CONVERT RELATIVE TO SCREEN CENTER
RETURN

DX
LAC DS2
LLS 4
AND =10000
XOR DSX
TAD =-1000
JMP* D.X

GET STATUS WORD 2
SHIFT HIGH ORDER BIT INTO POSITION
REMOVE OTHER BITS
FORM 13-BIT X COORDINATE
CONVERT RELATIVE TO SCREEN CENTER
RETURN

DO
AND =77
RCL
CMA
TAD DSP
TAD =PDP2-PDP1-1
DAC T1

TRUNCATE HIGH ORDER BITS
MULTIPLY PARAMETER BY 2
FORM 1'S COMPLEMENT
ADD PUSH DOWN POINTER
COMPUTE ADDRESS OF PUSH DOWN ENTRY
SAVE TEMPORARILY
TAD =-PDP2
SPA
JMP* D.O
LAC* T1
LLS 3
AND =70000
ISZ T1
TAD* T1
TAD =7777
DAC* T1
ISZ D.O
JMP* D.O

DN JMS D.E
JMS T.F

FORM VALIDITY CHECK
SKIP IF PARAMETER VALID
NOT ENOUGH OWNERS
GET FIRST PUSH DOWN WORD
SHIFT BREAK FIELD INTO POSITION
REMOVE ALL BUT BREAK FIELD
SET POINTER TO SECOND PD ENTRY
COMBINE FIRST & SECOND ENTRIES
FORM ADDRESS IN OWNER OF OWNER
SAVE TEMPORARILY
GET ADDRESS OF DESIRED OWNER
INDICATE SUCCESS
RETURN

ENABLE DISPLAY INTERRUPTS
TERMINATE TASK

CLEAR DISPLAY READY SWITCH
GET DISPLAY READY SWITCH
SKIP IF SET
WAIT FOR DISPLAY TO FINISH FRAME
RETURN
CHECK DISPLAY READY SWITCH
SCHEDULE NEW SWITCH CHECK

DISPLAY READY SWITCH
GET TRANSLATION VALUE
SKIP IF TRANSLATION PENDING
RESUME DISPLAY & RETURN
STORE DISPLACEMENT
INVERT SIGN BIT
STORE COUNTERDISPLACEMENT
INDICATE TRANSLATION PERFORMED
RESUME DISPLAY & RETURN

GET DISPLAY STATUS WORD 1
GET BREAK FIELD
SKIP IF ZERO BREAK FIELD
USER FILE INTERRUPT
READ DISPLAY ADDRESS
A-69

TAD =-XP
SMA
JMP XLP
LAC DE+1
SZA
JMP ++3
IOT 724
JMP IR
LAC UPT
JMS DS
JMP *-4

IIS
LAC DSS
AND =7
SNA
JMP X1S
IOT 611
XOR #10000
SAD =D+4
JMS DWT
LAC DE+1
SZA
JMP ++5
IOT 611
TAD =1
IOT 1605
JMP IR
LAC DSS
LLS 14
AND #70000
IOT 601
DAC 23
LAC* 23
JMS DS
JMP *-13

DS
SDC 0
AND #77777
JMS TII
DZM DE+1

FORM ADDRESS CHECK
SKIP IF USER FILE INTERRUPT
TRACKING INTERRUPT
GET DISPLAY INT ENABLE SWITCH
SKIP IF DISPLAY INTERRUPTS DISABLED
GET STATUS FOR USER
RESUME DISPLAY
RETURN FROM INTERRUPT
GET ADDRESS OF SERVICE TASK
SCHEDULE SERVICE & READ STATUS
RESUME DISPLAY & RETURN
GET DISPLAY STATUS WORD 1
GET BREAK FIELD
SKIP IF USER FILE INTERRUPT
TRACKING INTERRUPT
READ DISPLAY ADDRESS
INTERPRET WITH BREAK FIELD 1
SKIP IF NOT DISPLAY SYNC INTERRUPT
SET DISPLAY READY SWITCH
GET DISPLAY INT ENABLE SWITCH
SKIP IF DISPLAY INTERRUPTS DISABLED
GET STATUS FOR USER
READ DISPLAY ADDRESS
FORM RESUME ADDRESS
RESUME DISPLAY
RETURN FROM INTERRUPT
GET DISPLAY STATUS WORD 1
SHIFT BREAK FIELD INTO POSITION
REMOVE ALL BUT BREAK FIELD
FORM DISPLAY ADDRESS
SAVE TEMPORARILY
GET ADDRESS OF SERVICE TASK
SCHEDULE SERVICE & READ STATUS
RESUME DISPLAY & RETURN
TRUNCATE HIGH ORDER BITS
SCHEDULE SERVICE TASK
DISABLE DISPLAY INTERRUPTS
LAC DSS
DAC DS1
IOT 1632
DAC DS2
IOT 611
DAC DSA
IOT 1612
DAC DSY
IOT 512
DAC DSX
IOT 511
DAC DSP
LAC =PDP1-1
DAC 10
LAC =PDP2-1
DAC 11
LAC* 10
DAC* 11
LAC 10
SAD DSP
JMP* DS
JMP *-5

GET DISPLAY STATUS WORD 1
GET ADDRESS OF PUSH DOWN LIST
GET ADDRESS OF PUSH DOWN SAVE AREA
GET WORD FROM PUSH DOWN LIST
GET SOURCE POINTER
COPY NEXT WORD

SAVE
READ STATUS WORD 2
READ DISPLAY ADDRESS
READ Y DISPLAY COORDINATE
READ X DISPLAY COORDINATE
READ PUSH DOWN POINTER
SAVE
GET ADDRESS OF PUSH DOWN LIST
SET AUTOINDEX REGISTER
GET ADDRESS OF PUSH DOWN SAVE AREA
SET AUTOINDEX REGISTER
STORE IN PUSH-DOWN SAVE AREA
GET SOURCE POINTER
SKIP IF NOT END OF LIST
RETURN
| X1 | TAD = 1000 AND = 1777 | CONVERT RELATIVE TO ORIGIN CONVERT MODULO 2\*10 | DAC XPY | SET Y TRACKING COORDINATE LAC0 | GET X COORDINATE TAD = 1000 AND = 1777 | CONVERT RELATIVE TO ORIGIN XOR = 4000 | SET ESCAPE BIT DAC XPX | SET X TRACKING COORDINATE DZM XP | ENABLE TRACKING JMP* X*I RETURN |
|----|----------------------|-------------------------------------------|--------|--------------------------------|--------------------------------|------------------------|-----------------|------------------------|--------|--------------------------------|
| XR | DZM XP | ENABLE TRACKING | JMP* X*R | RETURN |
| XT | LAW 3000 DAC XP | GET POP INSTRUCTION | JMP* X*T | RETURN |
| XS | LAW 3000 SAD XP | GET POP INSTRUCTION ISZ X*S | SKIP IF TRACKING ENABLED | INDICATE SUCCESS | JMP* X*S | RETURN |
| XY | LAC XPY | GET Y TRACKING COORDINATE | TAD = -1000 | CONVERT RELATIVE TO SCREEN CENTER | JMP* X*Y | RETURN |
| XX | LAC XPX AND = 1777 TAD = -1000 | GET X TRACKING COORDINATE TRUNCATE ESCAPE | JMP* X*X | CONVERT RELATIVE TO SCREEN CENTER | RETURN |
| XLP | HLT | STATE VARIABLE | HLT | STATE VARIABLE | HLT | STATE VARIABLE | JMP **3 | DO NOT CHANGE Y TRACKING COORDINATE | AND = 1777 | TRUNCATE HIGH ORDER BITS DAC XPY | SET Y TRACKING COORDINATE |
IOT 512 READ X COORDINATE
TAD = -2000 SUBTRACT 1024
SMA SKIP IF COORDINATE ON SCREEN
JMP *+4 DO NOT CHANGE X COORDINATE
AND =1777 TRUNCATE HIGH ORDER BITS
XOR =4000 SET ESCAPE BIT
DAC XPK SET X TRACKING COORDINATE
LAW XP GET ADDRESS OF TRACKING PATTERN
IOT 1605 RESTART TRACKING PROCESS
JMP IR RETURN FROM INTERRUPT
XLP1 IOT 724 RESUME DISPLAY
JMP IR RETURN

XIS IOT 611 READ DISPLAY ADDRESS
DAC 23 SAVE TEMPORARILY
LAC* 23 GET ADDRESS OF SERVICE
DAC 23 SAVE TEMPORARILY
JMP* 23 SERVICE INTERRUPT
XIS1 IOT 611 READ DISPLAY ADDRESS
TAD =1 FORM RESUME ADDRESS
IOT 1605 RESUME DISPLAY
JMP IR RETURN FROM INTERRUPT

X1 LAW 3000 GET POP INSTRUCTION
DAC XPS INHIBIT SEARCH PATTERN
LAC =700512 GET IOT 512 INSTRUCTION
DAC XLP MODIFY INTERRUPT SERVICE
LAC =40000+XL GET DAC XL INSTRUCTION
DAC XLP+1 MODIFY INTERRUPT SERVICE
LAC =600000+XLPI GET JMP XLPI INSTRUCTION
DAC XLP+2 MODIFY INTERRUPT SERVICE
DZM XL CLEAR LOW COORDINATE
DZM XH CLEAR HIGH COORDINATE
JMP XIS1 RESUME DISPLAY & RETURN

X2 LAC =40000+XH GET DAC XH INSTRUCTION
DAC XLP+1 MODIFY INTERRUPT SERVICE
JMP XIS1 RESUME DISPLAY & RETURN

X3 LAC XH GET HIGH COORDINATE
SNAX31
JMPX31
TADXL
SADXH
JMPX31
RCR
TAD=-2000
SMA
JMPX31+2
AND=1777
XOR=4000
DACXPX
JMPX31+2
LAW777
DACXPS
DZMXL
DZMXH
LAC=701612
DACXLPMODIFY INTERRUPT SERVICE
LAC=40000+XL
DACXLP+1
JMPXIS1
SNAX4
JMPX41
TADXL
SADXH
JMPX41
RCR
TAD=-2000
SMA
JMPX41+2
AND=1777
DACXPY
JMPX41+2
LAW777
DACXPS
LACXLP+7
DACXLP+1
LACXH
SNAX4
JMPX41
TADXL
SADXH
JMPX41
RCR
TAD=-2000
SMA
JMPX41+2
AND=1777
DACXPY
JMPX41+2
LAW777
DACXPS
LACXLP+7
DACXLP+1
LACXH
SNAX4
JMPX41
TADXL
SADXH
JMPX41
RCR
TAD=-2000
SMA
JMPX41+2
AND=1777
DACXPY
JMPX41+2
LAW777
DACXPS
LACXLP+7
DACXLP+1
LACXH
SNAX4
JMPX41
TADXL
SADXH
JMPX41
RCR
TAD=-2000
SMA
JMPX41+2
AND=1777
DACXPY
JMPX41+2
LAW777
DACXPS
LACXLP+7
DACXLP+1

SKIP IF VALID
ENABLE SEARCH PATTERN
ADD LOW COORDINATE
SKIP IF VALID
ENABLE SEARCH PATTERN
DIVIDE BY 2
SUBTRACT 1024
SKIP IF COORDINATE ON SCREEN
DO NOT CHANGE X COORDINATE
CONVERT MODULO 2^10
SET ESCAPE BIT
SET X TRACKING COORDINATE
LEAVE SEARCH PATTERN INHIBITED

GET SEARCH ENABLE WORD
ENABLE SEARCH PATTERN
CLEAR LOW COORDINATE
CLEAR HIGH COORDINATE
GET IOT 1612 INSTRUCTION
MODIFY INTERRUPT SERVICE
GET DAC XL INSTRUCTION
MODIFY INTERRUPT SERVICE
RESUME DISPLAY & RETURN

GET HIGH COORDINATE
SKIP IF NOT VALID
ENABLE SEARCH PATTERN
ADD LOW COORDINATE
SKIP IF VALID
ENABLE SEARCH PATTERN
DIVIDE BY 2
SUBTRACT 1024
SKIP IF COORDINATE ON SCREEN
DO NOT CHANGE Y TRACKING COORDINATE
CONVERT MODULO 2^10
SET Y TRACKING COORDINATE
LEAVE SEARCH PATTERN INHIBITED

GET SEARCH ENABLE WORD
ENABLE SEARCH PATTERN
GET TAD=-2000 INSTRUCTION
MODIFY INTERRUPT SERVICE
LAC = 740100
DAC XLP+2
JMP XIS1

GET SMA INSTRUCTION
MODIFY INTERRUPT SERVICE
RESUME DISPLAY & RETURN

X5
LAW 3000
DAC XP
JMP XIS1

GET POP INSTRUCTION
DISABLE TRACKING
RESUME DISPLAY & RETURN
STITLE

STL

JMS B4
JMP* S.TL
DAC T5
TAD =-1
DAC 12
LAW 0
DAC* 12
DAC* 12
DAC* 12
LAW 1121
DAC* 12
LAW 0
DAC* 12
LAW 4000
DAC* 12
LAW 2001
DAC* 12
JMS B4
JMP STL1
DAC* 12
DAC* 12
DAC* 12
DAC* 12
LAW 640
DAC 12
LAW 1400
DAC* 12
DZM* 12
LAW 1121
DAC* 12
DZM* 12
LAW 4000
DAC* 12
LAW 3000
DAC* 12
LAC T5
ISZ S.TL
JMP* S.TL

STRUCTURE TOPOLOGY OPERATORS

GET 8-WORD BLOCK
NOT ENOUGH STORAGE
SAVE ADDRESS FOR RETURN
COMPUTE INITIAL INDEX VALUE
SET AUTOINDEX REGISTER
GET DISPLAY NOP INSTRUCTION
STORE IN FIRST LOCATION IN HEAD
STORE IN SECOND LOCATION IN HEAD
STORE IN THIRD LOCATION IN HEAD
GET VEC INSTRUCTION
STORE IN FOURTH LOCATION IN HEAD
GET DISPLAY NOP INSTRUCTION
STORE IN FIFTH LOCATION IN HEAD
GET ZERO X COORD WITH ESCAPE BIT
STORE IN SIXTH LOCATION IN HEAD
GET JUMPI INSTRUCTION
STORE IN SEVENTH LOCATION IN HEAD
GET 8-WORD BLOCK
NOT ENOUGH STORAGE
STORE ADDRESS OF TAIL IN HEAD
COMPUTE INITIAL INDEX VALUE
SET AUTOINDEX REGISTER
GET UNCONDITIONAL DISPLAY SKIP
STORE IN FIRST LOCATION IN TAIL
GET INTERNAL STOP INSTRUCTION
STORE IN SECOND LOCATION IN TAIL
ZERO IN THIRD LOCATION IN TAIL
STORE DISPLAY NOP IN FOURTH LOCATION
GET VEC INSTRUCTION
STORE IN FIFTH LOC IN TAIL
STORE IN SIXTH LOC IN TAIL
GET ZERO X COORD WITH ESCAPE BIT
STORE IN SEVENTH LOCATION IN TAIL
GET POP INSTRUCTION
STORE IN EIGHTH LOC IN TAIL
GET ADDRESS OF CREATED LEVEL
INDICATE SUCCESS
RETURN
FREE FIRST 4-WORD BLOCK IN HEAD
GET ADDRESS OF 8-WORD BLOCK
FORM ADDRESS OF SECOND 4-WORD BLOCK
SAVE TEMPORARILY
FREE SECOND 4-WORD BLOCK IN HEAD
FAILURE RETURN

SKIP IF NOT HIGHEST ACTIVE LEVEL
HIGHEST ACTIVE LEVEL
SAVE ADDRESS OF FIRST HEAD BLOCK
FORM ADDRESS OF SECOND HEAD BLOCK
SAVE ADDRESS OF SECOND HEAD BLOCK
FORM POINTER TO LAST LOC IN HEAD
SAVE TEMPORARILY
GET ADDRESS OF TAIL (OR NODE)
SAVE ADDRESS OF TAIL (OR NODE)
FORM ADDRESS OF SECOND TAIL BLOCK
SAVE
GET FIRST WORD OF TAIL (OR NODE)
TRUNCATE BREAK FIELD
SKIP IF NOT NODE
LEVEL NOT EMPTY
RELEASE FIRST HEAD BLOCK
RELEASE SECOND HEAD BLOCK
RELEASE FIRST TAIL BLOCK
RELEASE SECOND TAIL BLOCK
INDICATE SUCCESS
RETURN

FORM POINTER TO LAST LOC IN HEAD
SAVE
CREATE 4-WORD BLOCK
NOT ENOUGH STORAGE
SET POINTER TO BLOCK
COMPUTE INITIAL INDEX VALUE
SET AUTOINDEX REGISTER
SHIFT BREAK FIELD INTO AC
TRUNCATE HIGH ORDER BITS
FORM PUSH JUMP INSTRUCTION
STORE IN FIRST LOC IN BLOCK
AND =7
LLS 14
DAC* 12
LAW 2001
DAC* 12
LAC+ T5
DAC* 12
LAC T1
DAC+ T5
ISZ S.T1
JMP* S.T1
TRUNCATE HIGH ORDER BITS
GET COMPLETE ADDRESS
STORE IN SECOND LOC IN BLOCK
GET JUMP INSTRUCTION
STORE IN THIRD LOC IN BLOCK
GET ADR OF FIRST ELEMENT IN LEVEL
STORE AS SUCCESSOR TO NEW NODE
GET ADDRESS OF NEW NODE
INSERT NEW NODE INTO LEVEL
INDICATE SUCCESS
RETURN

STR
JMS TV
SDC S.TR
SDC 0
TAD =7
DAC T1
LAC* T1
DAC T2
DAC STR2
LAC* T2
AND =777770
SAD =762010
SKP
JMP STR1+7
ISTR T2
LACQ
SAD* T2
JMP STR1
LAC T2
TAD =2
JMP STR+4
ISTR T2
ISTR T2
LAC* T2
DAC* T1
JMS DW
DZM* STR2
ISTR STR+2
JMS T.U
PROTECT AGAINST REENTRY
GET POINTER TO END OF HEAD
SAVE TEMPORARILY
GET ADDRESS OF FIRST ELEMENT
SAVE TEMPORARILY
SAVE ADDRESS FOR REMOVAL
GET FIRST WORD OF FIRST ELEMENT
TRUNCATE BREAK FIELD
SKIP IF NOT NODE
NODE
SUBSTRUCTURE NOT IN LEVEL
FORM POINTER TO ADR OF SUBSTRUCTURE
GET ADDRESS OF GIVEN SUBSTRUCTURE
SKIP IF NO MATCH
SUBSTRUCTURE FOUND
FORM POINTER TO END OF NODE
TRY NEXT NODE
INCREMENT POINTER TO LOC IN NODE
INCREMENT POINTER TO LOC IN NODE
GET ADR OF SUCCESSOR TO NODE
STORE IN PREVIOUS NODE (OR HEAD)
WAIT FOR DISPLAY TO SETTLE DOWN
RELEASE NODE TO FREE STORAGE
INDICATE SUCCESS
UNLOCK S.TRD
STITLE

LEVEL MODIFICATION OPERATORS

SLH   LAC = DHAL
      JMP* S.LH
      GET ADDRESS OF HIGHEST ACTIVE LEVEL
      RETURN

SLY   JMS TV
      SDC S.LY
      SDC 0
      SAD = DHAL
      JMP SLY1+3
      TAD = 4
      DAC DWHD
      TAD = 3
      JMS SLT
      TAD = 5
      DAC DWTI
      LACQ
      JMS C+BC
      XOR = 760000
      DAC DWV
      SLY1   LAC DWV
      SZA
      JMP **4
      JMS T.U
      SDC SLY
      JMP SLY1
      JMS T.P
      CHECK FOR TRANSLATION COMPLETE
      SCHEDULE COMPLETION CHECK

SLX   JMS TV
      SDC $LX
      SDC 0
      SAD = DHAL
      JMP SLX1+3
      TAD = 5
      DAC DWHD
      TAD = 2
      JMS SLT
      TAD = 6
      DAC DWTI
      LACQ
      GET X INCREMENT
A-80

JMS C•BC
XOR •4000
DAC DWV
SLX1 LAC DWV
SZA
JMP •••4
JMS T•U
SDC SLX
JMP SLX1
JMS T•P

CONVERT TO DISPLAY COORDINATE
SET ESCAPE BIT
SAVE TRANSLATION VALUE
GET TRANSLATION VALUE
SKIP IF TRANSLATION COMPLETE
RESCHEDULE COMPLETION CHECK
UNLOCK S•LX
CHECK FOR TRANSLATION COMPLETE
SCHEDULE COMPLETION CHECK

SLP SAD •DHAL
JMP• S•LP
TAD •2
DAC T1
LAC8
AND •777
DAC• T1
JMP• S•LP

SKIP IF NOT HIGHEST ACTIVE LEVEL
RETURN
GET ADDRESS OF PARAMETER SLOT
SAVE TEMPORARILY
GET PARAMETERS
TRUNCATE HIGH ORDER BITS
STORE PARAMETERS IN LEVEL
RETURN

SLBE SAD •DHAL
JMP• S•LBE
TAD •1
DAC T2
TAD •6
JMS SLT
TAD •3
DAC T1
LAW 6301
DAC• T1
LAC• T2
AND •74
TAD •6302
DAC• T2
JMP• S•LBE

SKIP IF NOT HIGHEST ACTIVE LEVEL
RETURN
FORM POINTER TO COUNT SLOT
SAVE TEMPORARILY
FORM POINTER TO END OF HEAD
GET ADDRESS OF TAIL
FORM POINTER TO BLINK OFF SLOT
SAVE TEMPORARILY
GET BLINK OFF INSTRUCTION
STORE IN TAIL
GET COUNT INSTRUCTION
GET COUNT BITS
FORM NEW COUNT INSTRUCTION
STORE NEW COUNT INSTRUCTION
RETURN

SLBD SAD •DHAL
JMP• S•LBD
TAD •1
DAC T2

SKIP IF NOT HIGHEST ACTIVE LEVEL
RETURN
FORM POINTER TO COUNT SLOT
SAVE TEMPORARILY
TAD = 6
JMS SLT
TAD = 3
DAC T1
LAC* T2
AND =7777
DAC* T2
DZM* T1
JMP* S.LBD

FORM POINTER TO END OF HEAD
GET ADDRESS OF TAIL
FORM POINTER TO BLINK OFF SLOT
SAVE TEMPORARILY
GET COUNT INSTRUCTION
FORM NEW COUNT INSTRUCTION
STORE NEW COUNT INSTRUCTION
REMOVE BLINK OFF INSTRUCTION
RETURN

SLC
SAD =DHAL
JMP* S.LC
TAD =1
DAC T1
LAC* T1
AND =2
DAC T2
LACQ
AND =74
XOR T2
XOR =6300
DAC* T1
JMP* S.LC

SKIP IF NOT HIGHEST ACTIVE LEVEL
RETURN
FORM POINTER TO COUNT SLOT
SAVE TEMPORARILY
GET COUNT INSTRUCTION
GET BLINK BIT
SAVE TEMPORARILY
GET COUNT BITS
TRUNCATE OTHER BITS
CONCATENATE COUNT BITS & BLINK BIT
FORM NEW COUNT INSTRUCTION
STORE NEW COUNT INSTRUCTION
RETURN

SLU
SAD =DHAL
JMP* S.LU
JMS S.LN
DAC* T1
JMP* S.LU

SKIP IF NOT HIGHEST ACTIVE LEVEL
RETURN
REMOVE INTERRUPT AT END OF LEVEL
REMOVE SKIP INSTRUCTION FROM TAIL
RETURN

SLS
SAD =DHAL
JMP* S.LS
JMS S.LN
LAW 6280
DAC* T1
JMP* S.LS

SKIP IF NOT HIGHEST ACTIVE LEVEL
RETURN
REMOVE INTERRUPT AT END OF LEVEL
GET SKIP-IF-OFF-SCREEN INSTRUCTION
STORE IN TAIL
RETURN

SLL
SAD =DHAL
JMP* S.LL
JMS SLSP

SKIP IF NOT HIGHEST ACTIVE LEVEL
RETURN
REMOVE INTERRUPT FROM END OF LEVEL
| LAW 6201 | GET LPSI CLEAR INSTRUCTION |
| DAC* T2 | STORE IN HEAD |
| LAW 6202 | GET SKIP-ON-NO-LPSI INSTRUCTION |
| DAC* T1 | STORE IN TAIL |
| JMP* S.LL | RETURN |

| SLN | SAD =DHAL |
| JMP* S.LN | SKIP IF NOT HIGHEST ACTIVE LEVEL |
| JMS SLSP | RETURN |
| LAW 0 | REMOVE INTERRUPT AT END OF LEVEL |
| DAC* T2 | GET DISPLAY NOP INSTRUCTION |
| JMP* S.LN | REMOVE LPSI CLEAR |
| RETURN |

| SLT | $DC 0 |
| DAC T1 | STORE POINTER TO END OF BLOCK |
| LAC* T1 | GET POINTER TO NEXT NODE (OR TAIL) |
| DAC T1 | SAVE TEMPORARILY |
| LAC* T1 | GET FIRST WORD FROM NODE (OR TAIL) |
| AND =777770 | TRUNCATE BREAK FIELD |
| SAD =762010 | SKIP IF NOT NODE |
| JMP ++3 | TAIL NOT FOUND |
| LAC T1 | GET ADDRESS OF TAIL |
| JMP* SLT | RETURN |
| LAC T1 | GET POINTER TO NODE |
| TAD =3 | FORM POINTER TO END OF NODE |
| JMP SLT+1 | LOOK AT NEXT NODE (OR TAIL) |

| SLSP | $DC 0 |
| DAC T2 | SAVE ADDRESS OF LEVEL |
| TAD =7 | FORM POINTER TO END OF HEAD |
| JMS SLT | GET ADDRESS OF TAIL |
| TAD =2 | FORM POINTER TO TASK ADDRESS |
| DAC T3 | SAVE TEMPORARILY |
| LAW 6240 | GET SKIP INSTRUCTION |
| DAC* T1 | STORE IN TAIL |
| LACQ | GET NEW SERVICE TASK ADDRESS |
| DAC* T3 | STORE IN TAIL |
| JMP* SLSP | RETURN |
STITLE

TEXT OPERATORS

LT
DAC LT2
LAC LT2
CHM
DAC LT3
ISZ LT3
SKP
JMP L.T

LT1
ISZ LT2
LAC LT2
JMS B.T
ISZ LT3
JMP LT1
JMP L.T

LD
DAC T5
DAC T1
LAC T1
CHM
DAC T2
LAC =7
DAC T3
ISZ T2
SKP
JMP LD4

LD1
ISZ T1
LAC T1
LRS 14
JMS LD5
JMS LD5
JMS LD5
ISZ T2
JMP LD1
LAC T3
RCR
RCR
JMS B
JMP L.T
DAC T1
SAVE ADDRESS FOR RETURN
GET WORD COUNT
FORM 1'S COMPLEMENT
FORM 2'S COMPLEMENT
STORE COMPLEMENTED WORD COUNT
CLEAR HORIZONTAL COUNT
CLEAR VERTICAL COUNT
SET POINTER TO NEXT TEXT WORD
GET NEXT TEXT WORD
SHIFT FIRST CHARACTER INTO POSITION
PUT FIRST CHARACTER INTO LEAF
PUT SECOND CHARACTER INTO LEAF
PUT THIRD CHARACTER INTO LEAF
INCREMENT COUNT & SKIP IF DONE
PROCESS NEXT TEXT WORD
GET VEC INSTRUCTION
STORE IN TEXT LEAF
INCREMENT POINTER TO LOC IN LEAF
GET VERTICAL COUNT
PREPARE TO SHIFT ZEROS INTO AC
MULTIPLY BY 16
SET TO NONZERO VALUE
STORE IN TEXT LEAF
INCREMENT POINTER TO LOC IN LEAF
GET HORIZONTAL COUNT
MULTIPLY BY 8
CONVERT MODULO 2^10
SET ESCAPE BIT & MINUS SIGN
STORE IN TEXT LEAF
INCREMENT POINTER TO LOC IN LEAF
GET ADDRESS OF TEXT LEAF
INDICATE SUCCESS
RETURN
GET ADDRESS OF POP INSTRUCTION
INDICATE SUCCESS & RETURN
TRUNCATE HIGH ORDER BITS
SAD = 77  SKIP IF NOT NULL CHARACTER
JMP #5  NULL CHARACTER -- RETURN
SAD = 74  SKIP IF NOT CARRIAGE RETURN
ISZ T3  INCREMENT LEAF SIZE EXTRA TIME
ISZ T3  INCREMENT LEAF SIZE
ISZ T3  INCREMENT LEAF SIZE
LLS 6  SHIFT NEXT CHARACTER INTO POSITION
JMP* LD5  RETURN

LD6  $DC 0
AND = 77  TRUNCATE HIGH ORDER BITS
SAD = 77  SKIP IF NOT NULL CHARACTER
JMP LD7-2  NULL CHARACTER -- RETURN
SAD = 74  SKIP IF NOT CARRIAGE RETURN
JMP LD7  PUT CARRIAGE RETURN INTO LEAF
SAD = 75  SKIP IF NOT LINE FEED
SKP  LINE FEED -- INCREMENT VERT COUNT
JMP #3  NORMAL CHARACTER
ISZ T4  INCREMENT VERTICAL COUNT
SKP  LEAVE HORIZONTAL COUNT ALONE
ISZ T3  INCREMENT HORIZONTAL COUNT
TAD = LD8  ADD ADDRESS OF CONVERSION TABLE
DAC T7  SAVE TEMPORARILY
LAW 2018  GET PUSH JUMP INSTRUCTION
DAC* T1  STORE IN TEXT LEAF
ISZ T1  INCREMENT POINTER TO LOC IN LEAF
LAC* T7  GET ADDRESS OF DISPLAY FOR CHAR
DAC* T1  STORE IN TEXT LEAF
ISZ T1  INCREMENT POINTER TO LOC IN LEAF
LLS 6  SHIFT NEXT CHARACTER INTO POSITION
JMP* LD6  RETURN

LD7  LACQ  GET MQ CONTENTS
DAC T7  SAVE TEMPORARILY
LAW 1121  GET VEC INSTRUCTION
DAC* T1  STORE IN TEXT LEAF
ISZ T1  INCREMENT POINTER TO LOC IN LEAF
LAW 0  GET ZERO Y DISPLACEMENT
DAC* T1  STORE ZERO Y DISPLACEMENT IN LEAF
ISZ T1  INCREMENT POINTER TO LOC IN LEAF
LAC T3  GET HORIZONTAL DISPLACEMENT
CLQ
LLS 3
AND #$177
XOR #$6000
DAC* T1
ISZ T1
DZM T3
LAC T7
LRS 14
JMP* LD6

PREPARE TO SHIFT ZEROS INTO AC
MULTIPLY BY 8
CONVERT MODULO $2\times 10$
SET ESCAPE BIT & MINUS SIGN
STORE IN TEXT LEAF
INCREMENT POINTER TO LOC IN LEAF
CLEAR HORIZONTAL COUNT
GET PREVIOUS MQ CONTENTS
SHIFT NEXT CHARACTER INTO POSITION
RETURN

DAC T1
LAC* T1
DZM* T1
SAD $763000$
JMP* L+L
ISZ T1
JMP LL+1

STORE ADDRESS OF TEXT LEAF
GET VALUE FROM LEAF
FREE STORAGE LOCATION
SKIP IF NOT END OF TEXT LEAF
RETURN
SET POINTER TO NEXT LOC IN LEAF
FREE NEXT LOCATION

LD8
SDC D00
SDC D01+10000
SDC D02+20000
SDC D03+30000
SDC D04+40000
SDC D05+50000
SDC D06+60000
SDC D07+70000
SDC D10+10000
SDC D11+110000
SDC D12+120000
SDC D13+130000
SDC D14+140000
SDC D15+150000
SDC D16+160000
SDC D17+170000
SDC D20+200000
SDC D21+210000
SDC D22+220000
SDC D23+230000
SDC D24+240000

A-86
$DC D25+250000
$DC D26+260000
$DC D27+270000
$DC D30+300000
$DC D31+310000
$DC D32+320000
$DC D33+330000
$DC D34+340000
$DC D35+350000
$DC D36+360000
$DC D37+370000
$DC D40+400000
$DC D41+410000
$DC D42+420000
$DC D43+430000
$DC D44+440000
$DC D45+450000
$DC D46+460000
$DC D47+470000
$DC D50+500000
$DC D51+510000
$DC D52+520000
$DC D53+530000
$DC D54+540000
$DC D55+550000
$DC D56+560000
$DC D57+570000
$DC D60+600000
$DC D61+610000
$DC D62+620000
$DC D63+630000
$DC D64+640000
$DC D65+650000
$DC D66+660000
$DC D67+670000
$DC D70+700000
$DC D71+710000
$DC D72+720000
$DC D73+730000
NOP
A-89

STITLE

IDLE-LAW 17475
JMS B-T
LAW 10
JMS T-R
JMS B-K
DAC T3
LAW 10
JMS T-A
LAC BKF
SAD =285
JMP TTY4
LAC T3
SAD =14
LAC =IDLEC
SAD =33
LAC =IDLER
SAD =34
LAC =IDLES
SAD =72
LAC =IDLE1
SAD =17
LAC =IDLEF
DAC T3
AND =777700
SNA
JMP IDLEQ
LAC T3
TAD =1
JMS L-T
LAC T3
DAC T3
JMP T3

IDLEQ LAC =657475
JMP IDLE+1

IDLEC
SDC CLEAR
SDC 2

IDLE-TIME TASK

GET CARRIAGE RETURN, LINE FEED CODE
TYPE CARRIAGE RETURN, LINE FEED
GET TELEPRINTER ALLOCATION MASK
RELEASE TELEPRINTER
GET KEYBOARD CHARACTER
SAVE KEYBOARD CHARACTER
GET TELEPRINTER ALLOCATION MASK
ALLOCATE TELEPRINTER
GET ASCII FORM OF CHARACTER
SKIP IF NOT ENQUIRY
SEND ENQUIRY RECORD
GET KEYBOARD CHARACTER
SKIP IF NOT C
GET "CLEAR" RESPONSE POINTER
SKIP IF NOT R
GET "RUN" RESPONSE POINTER
SKIP IF NOT S
GET "SCHEDULE" RESPONSE POINTER
SKIP IF NOT #
GET TTY/201 RESPONSE POINTER
SKIP IF NOT F
GET "FROM" RESPONSE POINTER
SAVE SELECTED RESPONSE POINTER
TRUNCATE LOW ORDER BITS
SKIP IF LEGAL COMMAND
CANCEL COMMAND
GET RESPONSE POINTER
COMPUTE ADDRESS OF TEXT LIST
TYPE TEXT LIST
GET ADDRESS OF RESPONSE
SAVE TEMPORARILY
EXECUTE RESPONSE

GET QUESTION MARK CODE
TYPE & GET NEW COMMAND
<table>
<thead>
<tr>
<th>IDLER</th>
<th>SDC RUN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SDC 2</td>
</tr>
<tr>
<td></td>
<td>STEXT &quot;RUN&quot;</td>
</tr>
<tr>
<td></td>
<td>SDC 747575</td>
</tr>
<tr>
<td>IDLES</td>
<td>SDC SCHED</td>
</tr>
<tr>
<td></td>
<td>SDC 3</td>
</tr>
<tr>
<td></td>
<td>STEXT &quot;SCHEDULE &quot;</td>
</tr>
<tr>
<td>IDLE1</td>
<td>SDC TTY201</td>
</tr>
<tr>
<td></td>
<td>SDC 0</td>
</tr>
<tr>
<td>IDLEF</td>
<td>SDC FROM</td>
</tr>
<tr>
<td></td>
<td>SDC 2</td>
</tr>
<tr>
<td></td>
<td>STEXT &quot;FROM &quot;</td>
</tr>
</tbody>
</table>

| CLEAR | JMS B+K |
|       | GET KEYBOARD CHARACTER |
|       | SAD =15 |
|       | SKIP IF NOT D |
|       | JMP CLEAR1 |
|       | CLEAR DISPLAY STORAGE |
|       | SAD =35 |
|       | SKIP IF NOT T |
|       | SKP |
|       | CLEAR TASK QUEUE |
|       | JMP IDLEQ |
|       | CANCEL COMMAND |
|       | LAC =CLEARD |
|       | GET ADDRESS OF TEXT LIST |
|       | JMS L+T |
|       | TYPE TEXT LIST |
|       | LAC =SCHEDQ |
|       | GET ADDRESS OF FROZEN TASK QUEUE |
|       | JMS Q+C |
|       | CLEAR FROZEN TASK QUEUE |
|       | JMP IDLE |
|       | GET NEW COMMAND |

| CLEAR1 | LAC =CLEARD |
|        | GET ADDRESS OF TEXT LIST |
|        | JMS L+T |
|        | TYPE TEXT LIST |
|        | JMS STC |
|        | CLEAR DISPLAY STORAGE |
|        | DZM 25 |
|        | INDICATE NO DIAGNOSTIC |
|        | JMP E1 |
|        | RE-ESTABLISH DISPLAYED TITLE |

| CLEARD | SDC 5 |
|        | STEXT "DISPLAY STORAGE" |

| CLEART | SDC 4 |
|        | STEXT "TASK QUEUE" |

| RUN    | LAW 10 |
|        | GET TELEPRINTER MASK |
|        | JMS T.R |
|        | RELEASE TELEPRINTER |
|        | JMS STC |
|        | CLEAR DISPLAY STORAGE |
RUN1
LAC = SCHEDQ
JMS Q.F
JMS T.F
JMS T.S
JMP RUN1

GET ADDRESS OF FROZEN TASK QUEUE
GET TASK FROM FROZEN TASK QUEUE
TERMINATE IDLE-TIME EXECUTION
SCHEDULE TASK FROM FROZEN QUEUE
ENABLE NEXT TASK

SCHED
JMS OCTAL5
JMP IDLEQ
LMQ
LAC = SCHEDQ
JMS Q.A
SKP
JMP IDLE
LAC = SCHED1
JMS L.T
JMP IDLE

GET ADDRESS FROM KEYBOARD
CANCEL COMMAND
SET UP PARAMETER
GET ADDRESS OF FROZEN TASK QUEUE
ADD TASK TO FROZEN QUEUE
TYPE DIAGNOSTIC
GET NEW COMMAND
GET ADDRESS OF TEXT LIST
TYPE TEXT LIST
GET NEW COMMAND

SCHED1 $DC 11
$TEXT " -- NO ROOM FOR THIS TASK"

SCHEDQ $DC ++37
$DC ++36
$DC ++35
$ORG ++35

TTY201 LAW 17772
JMS B.T
JMS ECHO
LAC ECHO
SAD = 215
JMP TTY1
SAD = 337
JMP TTY2
SAD = 377
JMP TTY3
JMS B.F0
SKP
JMP TTY201+2
LAC = TTY5
JMS L.T

GET # CODE
ECHO KEYBOARD CHARACTER
GET ASCII FORM OF CHARACTER
SKIP IF NOT CARRIAGE RETURN
TERMINATE RECORD WITH ETX
SKIP IF NOT BACK ARROW
DELETE CHARACTER
SKIP IF NOT RUBOUT
CLEAR 201 OUTPUT BUFFER
SEND CHARACTER TO 201 OUTPUT BUFFER
DATA SET NOT CONNECTED
PROCESS NEXT CHARACTER
GET ADDRESS OF TEXT LIST
TYPE DIAGNOSTIC
A-92

TTY1
JMP IDLE
GET NEW COMMAND
LAW B FETX
GET END OF TEXT CHARACTER
JMS B • F0
SEND TO 201 OUTPUT BUFFER
JMP * - 5
DATA SET NOT CONNECTED
JMP IDLE
GET NEW COMMAND

TTY2
LAC = B FOB
GET ADDRESS OF OUTPUT BUFFER
SAD B FOI
SKIP IF BUFFER NON-EMPTY
JMP TTY201 + 2
IGNORE CHARACTER DELETE
JMS B • FO
WAIT FOR ACK TO LAST RECORD
JMP TTY1 - 3
DATA SET NOT CONNECTED
LAW - 2
LOAD AC WITH - 2
TAD BF0I
COMPUTE NEW VALUE OF INPUT POINTER
DAC BF0I
BACKSPACE OUTPUT BUFFER
JMP TTY201 + 2
PROCESS NEXT CHARACTER

TTY3
JMS B • FO
WAIT FOR ACK TO LAST RECORD
NOP
DATA SET NOT CONNECTED
LAC = B FOB
GET ADDRESS OF 201 OUTPUT BUFFER
DAC BF0I
RESET INPUT POINTER
LAC = TTY6
GET ADDRESS OF TEXT LIST
JMS L • T
TYPE DIAGNOSTIC
JMP IDLE
GET NEW COMMAND

TTY4
LAW B FENG
GET ENQUIRY
JMP TTY1 + 1
SEND TO 201 OUTPUT BUFFER

TTY5
SDC 11
TEXT "-- DATA SET NOT CONNECTED"

TTY6
SDC 4
TEXT "-- DELETED"

FROM
JMS B • K
GET KEYBOARD CHARACTER
DZM T3
CLEAR DATA TRANSFER POINTER
SAD = 14
SKIP IF NOT C
JMP FROM1
FROM CORE
ISZ T3
INCREMENT DATA TRANSFER POINTER
SAD = 31
SKIP IF NOT P
JMP FROM2
FROM PAPER TAPE
ISZ T3
INCREMENT DATA TRANSFER POINTER
SAD = 35
SKIP IF NOT T
JMP FROM3
FROM TELETYPE
JMP IDLE9
CANCEL COMMAND
A-93

FROM1 LAC = FROMC
JMP FROM4
GET ADDRESS OF TEXT LIST
TYPE TEXT LIST

FROM2 LAC = FROMP
SKP
GET ADDRESS OF TEXT LIST
TYPE TEXT LIST

FROM3 LAC = FROMT
LAC T3
GET ADDRESS OF TEXT LIST
TYPE TEXT LIST

FROM4 JMS L T
LL + RTL
MULTIPLY BY 4
TAD = FROM11
ADD ADDRESS OF TABLE
DAC T3
STORE REFINED DATA TRANSFER POINTER
LAC = FROMTO
GET ADDRESS OF TEXT LIST
JMS L T
TYPE TEXT LIST
JMS B K
GET KEYBOARD CHARACTER
SAD = 14
SKIP IF NOT C
JMP FROM5
TO CORE
ISZ T3
INCREMENT DATA TRANSFER POINTER
SAD = 31
SKIP IF NOT P
JMP FROM6
TO PAPER TAPE
ISZ T3
INCREMENT DATA TRANSFER POINTER
SAD = 35
SKIP IF NOT T
JMP FROM7
TO TELETYPE
ISZ T3
INCREMENT DATA TRANSFER POINTER
SAD = 15
SKIP IF NOT D
JMP FROM8
TO DISPLAY
JMP IDLEQ
CANCEL COMMAND

FROM5 LAC = FROMC
GET ADDRESS OF TEXT LIST
JMP FROM9
TYPE TEXT LIST

FROM6 LAC = FROMP
GET ADDRESS OF TEXT LIST
JMP FROM9
TYPE TEXT LIST

FROM7 LAC = FROMT
SKP
GET ADDRESS OF TEXT LIST
TYPE TEXT LIST

FROM8 LAC = FROMMD
GET ADDRESS OF TEXT LIST

FROM9 JMS L T
LAC T3
GET ADDRESS OF DATA TRANSFER
DAC T3
SAVE TEMPORARILY
JMP T3
BEGIN DATA TRANSFER

FROM11 SDC TRCC
SDC TRCP
SDC TRCT
$DC $TRCD
$DC $TRPC
$DC $TRPP
$DC $TRPT
$DC $TRPD
$DC $TPTC
$DC $TRIP
$DC $TRTT
$DC $TRTD

FROMC $DC 2
$TEXT "CORE"

FROMP $DC 4
$TEXT "PAPER TAPE"

FROMT $DC 3
$TEXT "TELETYPewriter"

FROMD $DC 3
$TEXT "DISPLAY"

FROMTO $DC 2
$TEXT " TO "

TRCC  JMS TRBK               GET CORE BLOCK FROM KEYBOARD
      LAC =FROMTO             GET ADDRESS OF TEXT LIST
      JMS L+T                 TYPE TEXT LIST
      JMS OCTAL5               GET ADDRESS FROM KEYBOARD
      JMP IDLEQ                CANCEL COMMAND
      DAC Ti                   SAVE ADDRESS

TRCC1 LAC* TRBKl              GET WORD TO BE MOVED
       DAC* Ti                STORE IN NEW LOCATION
       ISZ TRBKl               INCREMENT SOURCE POINTER
       ISZ Ti                  INCREMENT SINK POINTER
       ISZ TRBKc               INCREMENT LOC COUNT & SKIP IF DONE
       JMP TRCC1               MOVE NEXT WORD
       JMP IDLE                GET NEW COMMAND

TRCP  JMS TRBK               GET CORE BLOCK FROM KEYBOARD
GET ORIGIN CONTROL BIT
SET CONTROL MASK
GET ORIGIN OF BLOCK
PUNCH ORIGIN
GET DATA CONTROL BIT
SET CONTROL MASK
GET DATA WORD
PUNCH DATA WORD
INCREMENT POINTER
INCREMENT COUNT & SKIP IF DONE
PUNCH NEXT WORD
GET NEW COMMAND

SAVE WORD TO BE PUNCHED
SHIFT HIGH ORDER BITS INTO POSITION
TRUNCATE BITS FROM LINK
SET CONTROL BIT
PUNCH IMAGE
GET WORD TO BE PUNCHED
SHIFT MIDDLE BITS INTO POSITION
TRUNCATE HIGH ORDER BITS
SET CONTROL BIT
PUNCH IMAGE
GET WORD TO BE PUNCHED
TRUNCATE HIGH ORDER BITS
SET CONTROL BIT
PUNCH IMAGE
RETURN

GET CORE BLOCK FROM KEYBOARD
GET CARRIAGE RETURN, LINE FEED CODE
TYPE CARRIAGE RETURN, LINE FEED
GET ADDRESS TO BE TYPED
CONVERT TO 6-BIT CODE
REMOVE HIGH ORDER ZERO
TYPE ADDRESS
GET CODE FOR TWO SPACES
TYPE TWO SPACES
LOAD AC WITH -8
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC T3</td>
<td>SET WORD COUNTER</td>
</tr>
<tr>
<td>TRCTI</td>
<td>GET CODE FOR ONE SPACE</td>
</tr>
<tr>
<td>LAW 17677</td>
<td>TYPE SPACE</td>
</tr>
<tr>
<td>JMS B+T</td>
<td>GET WORD TO BE TYPED</td>
</tr>
<tr>
<td>LAC* TRBKL</td>
<td>CONVERT TO 6-BIT CODE</td>
</tr>
<tr>
<td>JMS C+B6</td>
<td>TYPE WORD</td>
</tr>
<tr>
<td>ISZ TRBKL</td>
<td>INCREMENT LOCATION POINTER</td>
</tr>
<tr>
<td>ISZ TRBKC</td>
<td>INCREMENT COUNT &amp; SKIP IF DONE</td>
</tr>
<tr>
<td>SKP</td>
<td>TYPE NEXT WORD</td>
</tr>
<tr>
<td>JMP IDLE</td>
<td>GET NEW COMMAND</td>
</tr>
<tr>
<td>ISZ T3</td>
<td>SKIP IF END OF LINE</td>
</tr>
<tr>
<td>JMP TRCT1</td>
<td>TYPE NEXT WORD</td>
</tr>
<tr>
<td>JMP TRCT+1</td>
<td>BEGIN NEW LINE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRCD</td>
<td>GET CORE BLOCK FROM KEYBOARD</td>
</tr>
<tr>
<td>JMS TRBK</td>
<td>GET WORD COUNT</td>
</tr>
<tr>
<td>LAC TRBKC</td>
<td>MAKE POSITIVE IF NOT TOO LARGE</td>
</tr>
<tr>
<td>TAD =100</td>
<td>SKIP IF TOO LARGE</td>
</tr>
<tr>
<td>SMA</td>
<td>WORD COUNT OK</td>
</tr>
<tr>
<td>JMP **3</td>
<td>LOAD AC WITH -64</td>
</tr>
<tr>
<td>LAW 17700</td>
<td>ADJUST WORD COUNT</td>
</tr>
<tr>
<td>DAC TRBKC</td>
<td>INITIALIZE TEXT LIST FOR DISPLAY</td>
</tr>
<tr>
<td>JMS TRD1</td>
<td>GET ADDRESS TO BE DISPLAYED</td>
</tr>
<tr>
<td>LAC TRBKL</td>
<td>CONVERT TO 6-BIT CODE</td>
</tr>
<tr>
<td>TAD =770000</td>
<td>REMOVE HIGH ORDER ZERO</td>
</tr>
<tr>
<td>JMS TRD2</td>
<td>PUT HIGH ORDER DIGITS IN TEXT LIST</td>
</tr>
<tr>
<td>LACQ</td>
<td>GET LOW ORDER DIGITS</td>
</tr>
<tr>
<td>JMS TRD2</td>
<td>PUT LOW ORDER DIGITS IN TEXT LIST</td>
</tr>
<tr>
<td>LAW 17676</td>
<td>GET CODE FOR TWO SPACES</td>
</tr>
<tr>
<td>JMS TRD2</td>
<td>PUT IN TEXT LIST</td>
</tr>
<tr>
<td>LAW 17770</td>
<td>LOAD AC WITH -8</td>
</tr>
<tr>
<td>DAC T3</td>
<td>SET WORD COUNTER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRCD2</td>
<td>GET CODE FOR ONE SPACE</td>
</tr>
<tr>
<td>LAW 17677</td>
<td>PUT IN TEXT LIST</td>
</tr>
<tr>
<td>JMS TRD2</td>
<td>GET WORD TO BE DISPLAYED</td>
</tr>
<tr>
<td>LAC* TRBKL</td>
<td>CONVERT TO 6-BIT CODE</td>
</tr>
<tr>
<td>JMS C+B6</td>
<td>PUT HIGH ORDER DIGITS IN TEXT LIST</td>
</tr>
<tr>
<td>JMS TRD2</td>
<td>GET LOW ORDER DIGITS</td>
</tr>
<tr>
<td>LACQ</td>
<td>PUT LOW ORDER DIGITS IN TEXT LIST</td>
</tr>
<tr>
<td>JMS TRD2</td>
<td>INCREMENT LOCATION POINTER</td>
</tr>
<tr>
<td>Instruction</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>ISZ TRBKJC</td>
<td>Increment count &amp; skip if done</td>
</tr>
<tr>
<td>JMP TRCD3</td>
<td>Prepare next word</td>
</tr>
<tr>
<td>CLC</td>
<td>Get three null characters</td>
</tr>
<tr>
<td>DAC T3</td>
<td>Nullify accumulated characters</td>
</tr>
<tr>
<td>JMS TRD3</td>
<td>Display text list</td>
</tr>
<tr>
<td>JMP IDLE</td>
<td>Get new command</td>
</tr>
<tr>
<td>TRCD3</td>
<td>ISZ T3</td>
</tr>
<tr>
<td>JMP TRCD2</td>
<td>Prepare next word</td>
</tr>
<tr>
<td>LAC =747575</td>
<td>Get carriage return, line feed code</td>
</tr>
<tr>
<td>JMS TRD2</td>
<td>Put in text list</td>
</tr>
<tr>
<td>JMP TRCD1</td>
<td>Begin new line</td>
</tr>
<tr>
<td>TRPC</td>
<td>JMS READ</td>
</tr>
<tr>
<td>SNA</td>
<td>Skip if not end of record</td>
</tr>
<tr>
<td>JMP IDLE</td>
<td>Get new command</td>
</tr>
<tr>
<td>DAC T3</td>
<td>Save tape line</td>
</tr>
<tr>
<td>AND =300</td>
<td>Get control bits</td>
</tr>
<tr>
<td>SAD =100</td>
<td>Skip if not origin</td>
</tr>
<tr>
<td>JMP TRPC1</td>
<td>Complete origin</td>
</tr>
<tr>
<td>SAD =200</td>
<td>Skip if not binary data</td>
</tr>
<tr>
<td>JMP TRPC2</td>
<td>Complete data word</td>
</tr>
<tr>
<td>JMS READ</td>
<td>Read a tape image</td>
</tr>
<tr>
<td>SNA</td>
<td>Skip if not end of record</td>
</tr>
<tr>
<td>JMP TRPC</td>
<td>Restart data transfer</td>
</tr>
<tr>
<td>JMP *-3</td>
<td>Ignore tape image</td>
</tr>
<tr>
<td>TRPC1</td>
<td>JMS TRPC3</td>
</tr>
<tr>
<td>DAC T4</td>
<td>Set origin</td>
</tr>
<tr>
<td>JMP TRPC</td>
<td>Get next word from tape</td>
</tr>
<tr>
<td>TRPC2</td>
<td>JMS TRPC3</td>
</tr>
<tr>
<td>DAC* T4</td>
<td>Load data word</td>
</tr>
<tr>
<td>ISZ T4</td>
<td>Increment location counter</td>
</tr>
<tr>
<td>JMP TRPC</td>
<td>Get next word from tape</td>
</tr>
<tr>
<td>TRPC3</td>
<td>SDC 0</td>
</tr>
<tr>
<td>JMS READ</td>
<td>Shift data bits into MQ</td>
</tr>
<tr>
<td>LRS 6</td>
<td>Get high order 6 bits</td>
</tr>
<tr>
<td>LAC T3</td>
<td>Shift high order 12 bits into AC</td>
</tr>
<tr>
<td>LLS 6</td>
<td>Save high order 12 bits</td>
</tr>
<tr>
<td>DAC T3</td>
<td>Get third image from tape</td>
</tr>
<tr>
<td>JMS READ</td>
<td>Get third image from tape</td>
</tr>
</tbody>
</table>
LRS 6       SHIFT DATA BITS INTO M0
LAC T3      GET HIGH ORDER 12 BITS
LLS 6       SHIFT COMPLETED WORD INTO AC
JMP TRPC3   RETURN

TRPP JMS READ       GET IMAGE FROM PAPER TAPE
SAD =377 JMP TRPP    RESTART DATA TRANSFER
DAC T3       SAVE TEMPORARILY
AND =300      GET CONTROL BITS
SAD =300      SKIP IF NOT ALPHANUMERIC
JMS TRPP3     PUNCH END-OF-RECORD MARK
LAC T3       GET IMAGE READ

TRPP1 JMS PUNCH    PUNCH IMAGE
JMS READ       GET IMAGE FROM PAPER TAPE
SNA       JMP TRPP2 PUNCH END-OF-RECORD IF NECESSARY
DAC T3       SAVE TEMPORARILY
JMP TRPP1     PUNCH IMAGE

TRPP2 LAC T3      GET LAST IMAGE PUNCHED
AND =300      GET CONTROL BITS
SAD =300      SKIP IF NOT ALPHANUMERIC
JMS TRPP3     PUNCH END-OF-RECORD MARK
JMP IDLE      GET NEW COMMAND

TRPP3 $DC 0       GET END-OF-RECORD MARK
CLA
JMS PUNCH     PUNCH END-OF-RECORD MARK
JMP TRPP3     RETURN
IDLE-TIME TASK (CONTINUED)

TRPT JMS READ
SAD #377
JMP TRPT
DAC T3
AND #300
SAD #300
JMP TRPT1
JMS READ
SNA
JMP TRPT
JMP *-3

TRPT1 LAW 17475
JMS B.T
LAC T3
XOR #777400
JMS B.T
SNA
JMP IDLE
JMP TRPT2

TRPD JMS READ
SNA
JMP TRPD
SAD #375
JMP TRPD
SAD #377
JMP TRPD
DAC T6
AND #300
SAD #300
JMP TRPD1
JMS READ
SNA
JMP TRPD
JMP *-3

TRPD1 JMS TRD1
LAW 17766

GET IMAGE FROM PAPER TAPE
SKIP IF NOT END-OF-TAPE GARBAGE
RESTART DATA TRANSFER
SAVE TEMPORARILY
GET CONTROL BITS
SKIP IF BINARY INFORMATION
RECORD IS ALPHANUMERIC
GET IMAGE FROM PAPER TAPE
SKIP IF NOT END OF RECORD
TRY TRANSFER AGAIN
GET NEXT IMAGE
GET CARRIAGE RETURN, LINE FEED CODE
TYPE CARRIAGE RETURN, LINE FEED
GET FIRST IMAGE FROM TAPE
PRECEDE WITH NULL CHARACTERS
TYPE CHARACTER FROM TAPE
GET IMAGE FROM TAPE
SKIP IF NOT END OF RECORD
GET NEW COMMAND
TYPE CHARACTER
READ IMAGE FROM TAPE
SKIP IF NOT END-OF-RECORD CHARACTER
RESTART DATA TRANSFER
SKIP IF NOT LINE FEED
RESTART DATA TRANSFER
SKIP IF NOT END-OF-TAPE GARBAGE
RESTART DATA TRANSFER
SAVE TEMPORARILY
GET CONTROL BITS
SKIP IF BINARY
RECORD OK
READ IMAGE FROM TAPE
SKIP IF NOT END OF RECORD
TRY TRANSFER AGAIN
IGNORE IMAGE
INITIALIZE TEXT LIST
LOAD AC WITH -10
DAC T4  SET LINE COUNTER
LAW 17676  LOAD AC WITH -66
DAC T5  SET CHARACTER COUNTER
LAC T6  GET FIRST CHARACTER
JMP TRPD2+3  ADD TO TEXT LIST

TRPD2 LAW 17676  LOAD AC WITH -66
DAC T5  SET CHARACTER COUNTER
JMS READ  READ IMAGE FROM TAPE
SAD =374  SKIP IF NOT CARRIAGE RETURN
JMP TRPD3  TERMINATE LINE
SNA  SKIP IF NOT END OF RECORD
JMP TRPD4  TERMINATE TRANSFER
JMS TRD4  ADD CHARACTER TO TEXT LIST
ISZ T5  INCREMENT CHAR COUNT & SKIP IF DONE
JMP TRPD2+2  GET NEXT CHARACTER

TRPD3 ISZ T4  INCREMENT COUNTER & SKIP IF DONE:
SKP  GET ANOTHER LINE
JMP TRPD4  TERMINATE TRANSFER
LAW 74  GET CARRIAGE RETURN
JMS TRD4  ADD TO TEXT LIST
LAW 75  GET LINE FEED
JMS TRD4  ADD TO TEXT LIST
JMP TRPD2  BEGIN NEW LINE

TRPD4 JMS TRD3  DISPLAY TEXT LIST
JMP IDLE  GET NEW COMMAND

TRTC LAW 17475  GET CARRIAGE RETURN, LINE FEED CODE
JMS B.T  TYPE IT
JMS OCTAL5  GET ADDRESS FROM KEYBOARD
JMP TRTC4  INTERPRET AS COMMAND
DAC T5  STORE ADDRESS

TRTC1 LAW 17677  GET CODE FOR ONE SPACE
JMS B.T  TYPE IT
LAC* T5  GET CURRENT CONTENT OF WORD
JMS C.B6  CONVERT TO 6-BIT CODE
JMS TRK1  TYPE CURRENT CONTENTS
LAW 17677  GET CODE FOR ONE SPACE
JMS B.T  TYPE IT
JMS OCTAL6  GET NEW CONTENTS FROM KEYBOARD
JMP TRTC3  DETERMINE NATURE OF FAILURE
DAC* T5
TRTC2 ISZ T5
LAW 17475
JMS B.T
LAC T5
JMS C.B6
TAD =778000
JMS TRKT
JMP TRTC1
TRTC3 SAD =74
JMP TRTC2
JMP TRTC
TRTC4 DAC T3
LAW 10
JMS T.R
JMP IDLE+6

STORE NEW CONTENTS
INCREMENT STORED ADDRESS
GET CARRIAGE RETURN, LINE FEED CODE
TYPE CARRIAGE RETURN, LINE FEED
GET CURRENT ADDRESS
CONVERT TO 6-BIT CODE
REMOVE HIGH ORDER ZERO
TYPE CURRENT ADDRESS
TYPE CONTENTS OF CURRENT LOCATION
SKIP IF NOT CARRIAGE RETURN
LEAVE WORD UNCHANGED
BEGIN INTERPRETATION OF NEW BLOCK
SAVE KEYBOARD CHARACTER
GET TELEPRINTER MASK
RELEASE TELEPRINTER
INTERPRET CHARACTER AS COMMAND
GET END-OF-RECORD MARK
PUNCH IT
GET CARRIAGE RETURN, LINE FEED CODE
TYPE CARRIAGE RETURN, LINE FEED
ECHO KEYBOARD CHARACTER
SKIP IT NOT NULL CHARACTER
TERMINATE TRANSFER
SET ALPHANUMERIC CONTROL BITS
PUNCH CHARACTER
GET NEXT CHARACTER
GET END-OF-RECORD MARK
PUNCH IT
GET NEW COMMAND
GET CARRIAGE RETURN, LINE FEED CODE
TYPE IT
ECHO KEYBOARD CHARACTER
SKIP IF NOT NULL CHARACTER
GET NEW COMMAND
GET NEXT CHARACTER
LOAD AC WITH -10
INITIALIZE LINE COUNTER
TRTD1  LAW 17475  JMS  B.T  LAW 17700  DTC  T5  JMS  B.T  JMS  B.T
        INITIALIZE TEXT LIST  GET CARRIAGE RETURN, LINE FEED CODE  TYPE CARRIAGE RETURN, LINE FEED  LOAD AC WITH -64  INITIALIZE CHARACTER COUNTER
TRTD2  JMS  ECHO  SAD  =77  JMP  TRTD4  SAD  =74  JMP  TRTD3  JMS  TRD4  ISZ  T5  JMP  TRTD2
        ECHO KEYBOARD CHARACTER  SKIP IF NOT NULL CHARACTER  DISPLAY TEXT LIST  SKIP IF NOT CARRIAGE RETURN  TERMINATE LINE  ADD CHARACTER TO TEXT LIST  SKIP IF END OF LINE  GET NEXT CHARACTER
TRTD3  ISZ  T4  SKP  JMP  TRTD4  LAW  74  JMS  TRD4  LAW  75  JMS  TRD4  LAW  75  JMS  TRD4  JMS  TRD4  JMP  TRT01
        INCREMENT LINE COUNT & SKIP IF DONE  TERMINATE LINE  GET CARRIAGE RETURN CODE  ADD TO TEXT LIST  GET LINE FEED CODE  ADD TO TEXT LIST  GET LINE FEED CODE  ADD TO TEXT LIST  BEGIN NEW LINE
TRTD4  JMS  TRD3  JMP  IDLE
        DISPLAY TEXT LIST  GET NEW COMMAND

ECH0  SDC  0  JMS  B.K  DAC  T6  XOR  =777700  JMS  B.T  LAC  T6  JMP  ECHO
        GET CHARACTER FROM KEYBOARD  SAVE TEMPORARILY  PRECEDE WITH NULL CHARACTERS  ECHO CHARACTER ON TELEPRINTER  GET CHARACTER FOR RETURN  RETURN

PUNCH  SDC  0  JMS  B.P  SKP  JMP*  PUNCH  LAC  =PUNCH1  JMS  L.T
        SEND IMAGE TO PUNCH  PUNCH OUT OF TAPE  RETURN  GET ADDRESS OF TEXT LIST  TYPE TEXT LIST
JMP IDLE GET NEW COMMAND

PUNCH
$DC 7
$DC 747531
$TEXT "UNCH OUT OF TAPE"

READ
$DC 0
JMP B.R
SKP
JMP* READ
LAC =READ
JMP IDLE
GET IMAGE FROM READER BUFFER
READER OUT OF TAPE
RETURN
GET ADDRESS OF TEXT LIST
TYPE TEXT LIST
GET NEW COMMAND

READ1
$DC 7
$DC 747533
$TEXT "READER OUT OF TAPE"

TRD1
$DC 0
LAC 27
LAC 26
SZA
JMS S.TR
NOP
LAC 27
SZA
JMS L.L
DZM 27
DZM TRDT
LAC =TRDT
DAC TRDP
CLC
DAC T3
JMP* TRD1
GET POINTER TO LEAF
SET UP PARAMETER
GET POINTER TO LEVEL
SKIP IF NO LEVEL
REMOVE LEAF FROM LEVEL
LEAF OR LEVEL DIDN'T EXIST
GET ADDRESS OF LEAF
SKIP IF NO LEAF
DESTROY LEAF
INDICATE NO LEAF
CLEAR TEXT LIST COUNT
GET ADDRESS OF TEXT LIST
INITIALIZE TEXT LIST POINTER
GET 3 NULL CHARACTERS
STORE NULL CHARACTERS
RETURN

TRD2
$DC 0
ISZ TRDT
ISZ TRDP
DAC* TRDP
INCREMENT TEXT LIST COUNT
INCREMENT TEXT LIST POINTER
STORE NEW TEXT WORD
JMP* TRD2
RETURN

TRD3 SDC 0
LAC T3
JMS TRD2
LAC 26
SNA
JMP TRD3
LAC *TRDT
JMS L+D
JMP TRD31
DAC 27
LMQ
LAC 26
JMP TRD3
TRD3I
JMS
LAC T3
JMS
LAC T3
JMP IDLE
SDC 12
SDC 747577
$TEXT "NOT ENOUGH DISPLAY STORAGE"

TRD4 SDC 0
LRS 6
LAC T3
LLS 6
DAC T3
AND =770000
SAD =770000
JMP* TRD4
LAC T3
JMS TRD2
CLC
DAC T3
JMP* TRD4

TRDT $DS 351
A-105

**STC**

- SDC 0
- LAC = D
- DAC DHAL+7
- JMS DW
- LAW STORE
- IOT 7704
- DAC T1
- DZM* T1
- ISZ T1
- JMP =-2
- IOT 7702
- JMP* STC

Get address of highest active level
Remove all nodes from HAL
Wait for display to recover
Get initial counter value
Set pointer & counter
Leave extend mode
Clear storage location
Increment pointer & counter
Clear next storage location
Enter extend mode
Return

**OCTAL1**

- SDC 0
- JMS B=K
- TAD =-10
- SPA
- JMP ++3
- TAD =10
- JMP* OCTAL1
- DAC T3
- XOR =70
- JMS B=T
- LAC T3
- LRS 3
- LAC T4
- LLS 3
- ISZ OCTAL1
- DAC T4
- JMP* OCTAL1

Get keyboard character
Make negative if octal
Skip if not octal digit
Octal digit typed
Restore character
Indicate failure
Save octal information
Convert to 6-bit code
Type octal digit
Get octal information
Shift digit into MO
Get recorded digits
Concatenate new digit
Record new word
Indicate success
Return

**OCTAL5**

- SDC 0
- DZM T4
- JMS OCTAL1
- JMP* OCTAL5
- JMS OCTAL1
- JMP* OCTAL5
- JMS OCTAL1
- JMP* OCTAL5
- JMS OCTAL1

Clear octal recording word
Get octal digit from keyboard
Non-octal character typed
Get octal digit from keyboard
Non-octal character typed
Get octal digit from keyboard
Non-octal character typed
Get octal digit from keyboard
NON-OCTAL CHARACTER TYPED
GET OCTAL CHARACTER FROM KEYBOARD
NON-OCTAL CHARACTER TYPED
INDICATE SUCCESS
RETURN
GET 5 OCTAL DIGITS FROM KEYBOARD
NON-OCTAL CHARACTER TYPED
GET OCTAL DIGIT FROM KEYBOARD
NON-OCTAL CHARACTER TYPED
INDICATE SUCCESS
RETURN
SAVE HIGH ORDER DIGITS
GET LOW ORDER DIGITS
SAVE LOW ORDER DIGITS
GET HIGH ORDER DIGITS
TYPE HIGH ORDER DIGITS
GET LOW ORDER DIGITS
TYPE LOW ORDER DIGITS
RETURN
GET ADDRESS OF TEXT LIST
TYPE TEXT LIST
GET LOW ADDRESS FROM KEYBOARD
CANCEL COMMAND
STORE LOW ADDRESS
GET COMMA CODE
TYPE COMMA
GET HIGH ADDRESS FROM KEYBOARD
CANCEL COMMAND
FORM ONE'S COMPLEMENT
ADD LOW ADDRESS
SKIP IF PROPERLY ORDERED ADDRESSES
CANCEL COMMAND
STORE LOCATION COUNT
GET RIGHT PARENTHESIS CODE
JMS B.T
JMP* TRBK

TRBKF $DC 3
$DC 747513
STEXT "LOCK('"

TYPE RIGHT PARENTHESIS
RETURN
<table>
<thead>
<tr>
<th>TITLE</th>
<th>HIGHEST ACTIVE LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>SDC 757</td>
</tr>
<tr>
<td></td>
<td>SDC 6201</td>
</tr>
<tr>
<td></td>
<td>SDC 6301</td>
</tr>
<tr>
<td></td>
<td>SDC 1400</td>
</tr>
<tr>
<td>DHAL</td>
<td>SDC DWT</td>
</tr>
<tr>
<td></td>
<td>SDC 2010</td>
</tr>
<tr>
<td></td>
<td>SDC XP</td>
</tr>
<tr>
<td></td>
<td>SDC 1105</td>
</tr>
<tr>
<td></td>
<td>SDC 1000</td>
</tr>
<tr>
<td></td>
<td>SDC 5000</td>
</tr>
<tr>
<td></td>
<td>SDC 2001</td>
</tr>
<tr>
<td></td>
<td>SDC D</td>
</tr>
</tbody>
</table>

SEND
APPENDIX B -- SUMMARY OF SYSTEM SUBROUTINES

The following table of system subroutines is provided as a reference to facilitate the writing of user programs. The various columns are interpreted as follows:

<table>
<thead>
<tr>
<th>NAME --</th>
<th>SYMBOLIC NAME OF THE SYSTEM SUBROUTINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTRY POINT --</td>
<td>ADDRESS AT WHICH THE SUBROUTINE STARTS</td>
</tr>
<tr>
<td>SECTION --</td>
<td>SECTION OF THE REPORT IN WHICH THE SUBROUTINE IS DESCRIBED</td>
</tr>
<tr>
<td>FAILURE RETURN --</td>
<td>WHETHER OR NOT A FAILURE RETURN EXISTS</td>
</tr>
<tr>
<td>DELAY POSSIBLE --</td>
<td>WHETHER OR NOT OTHER TASKS MAY BE EXECUTED BEFORE THE SUBROUTINE RETURNS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>ENTRY POINT</th>
<th>SECTION</th>
<th>FAILURE RETURN</th>
<th>DELAY POSSIBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.FI</td>
<td>140</td>
<td>3.4.1</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>B.FO</td>
<td>142</td>
<td>3.4.1</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>B.K</td>
<td>150</td>
<td>3.4.3</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>B.P</td>
<td>146</td>
<td>3.4.2</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>B.R</td>
<td>144</td>
<td>3.4.2</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>B.T</td>
<td>152</td>
<td>3.4.3</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>C.6A</td>
<td>130</td>
<td>3.3</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>C.A6</td>
<td>132</td>
<td>3.3</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>C.B6</td>
<td>126</td>
<td>3.3</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>C.BC</td>
<td>136</td>
<td>3.3</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>C.CH</td>
<td>134</td>
<td>3.3</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>D.A</td>
<td>206</td>
<td>3.7</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>D.D</td>
<td>202</td>
<td>3.7</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>D.E</td>
<td>200</td>
<td>3.7</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>D.Q</td>
<td>214</td>
<td>3.7</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>D.P</td>
<td>204</td>
<td>3.7</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>D.X</td>
<td>212</td>
<td>3.7</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>D.Y</td>
<td>210</td>
<td>3.7</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

B-1
<table>
<thead>
<tr>
<th>NAME</th>
<th>ENTRY POINT</th>
<th>SECTION</th>
<th>FAILURE RETURN</th>
<th>DELAY POSSIBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td>272</td>
<td>3.11</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>L.L</td>
<td>274</td>
<td>3.11</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>L.T</td>
<td>270</td>
<td>3.11</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>N.A</td>
<td>154</td>
<td>3.5</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>N.C</td>
<td>156</td>
<td>3.5</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>N.D1</td>
<td>160</td>
<td>3.5</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>N.D2</td>
<td>162</td>
<td>3.5</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>N.D3</td>
<td>164</td>
<td>3.5</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>P.D</td>
<td>172</td>
<td>3.6</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>P.E</td>
<td>170</td>
<td>3.6</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>P.R</td>
<td>174</td>
<td>3.6</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>P.S</td>
<td>176</td>
<td>3.6</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>P.T</td>
<td>166</td>
<td>3.6</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Q.A</td>
<td>102</td>
<td>3.1</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Q.C</td>
<td>100</td>
<td>3.1</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Q.F</td>
<td>106</td>
<td>3.1</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Q.I</td>
<td>104</td>
<td>3.1</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>S.LBD</td>
<td>254</td>
<td>3.10</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S.LBE</td>
<td>252</td>
<td>3.10</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S.LC</td>
<td>256</td>
<td>3.10</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S.LH</td>
<td>242</td>
<td>3.10</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S.LL</td>
<td>264</td>
<td>3.10</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S.LN</td>
<td>266</td>
<td>3.10</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S.LP</td>
<td>250</td>
<td>3.10</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S.LS</td>
<td>262</td>
<td>3.10</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S.LU</td>
<td>260</td>
<td>3.10</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S.LX</td>
<td>246</td>
<td>3.10</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>S.LY</td>
<td>244</td>
<td>3.10</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>S.TD</td>
<td>234</td>
<td>3.9</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>S.TI</td>
<td>236</td>
<td>3.9</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>S.TL</td>
<td>232</td>
<td>3.9</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>S.TR</td>
<td>240</td>
<td>3.9</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>T.A</td>
<td>116</td>
<td>3.2</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>T.F</td>
<td>114</td>
<td>3.2</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>T.L</td>
<td>122</td>
<td>3.2</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>T.P</td>
<td>112</td>
<td>3.2</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>T.R</td>
<td>120</td>
<td>3.2</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>NAME</td>
<td>ENTRY POINT</td>
<td>SECTION</td>
<td>FAILURE RETURN</td>
<td>DELAY POSSIBLE</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>---------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>T.S</td>
<td>110</td>
<td>3.2</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>T.U</td>
<td>124</td>
<td>3.2</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>X.I</td>
<td>216</td>
<td>3.8</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>X.R</td>
<td>220</td>
<td>3.8</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>X.S</td>
<td>224</td>
<td>3.8</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>X.T</td>
<td>222</td>
<td>3.8</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>X.X</td>
<td>230</td>
<td>3.8</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>X.Y</td>
<td>226</td>
<td>3.8</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>
APPENDIX C -- SUMMARY OF IOT INSTRUCTIONS

STATUS WORDS

ALL BITS WHOSE INTERPRETATIONS ARE NOT SPECIFIED BELOW ARE NOT USED.

PDP-9 I/O STATUS

<table>
<thead>
<tr>
<th>BIT</th>
<th>INTERPRETATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>INTERRUPTS ARE ENABLED</td>
</tr>
<tr>
<td>1</td>
<td>READER FLAG</td>
</tr>
<tr>
<td>2</td>
<td>PUNCH FLAG</td>
</tr>
<tr>
<td>3</td>
<td>KEYBOARD FLAG</td>
</tr>
<tr>
<td>4</td>
<td>TELEPRINTER FLAG</td>
</tr>
<tr>
<td>6</td>
<td>CLOCK FLAG</td>
</tr>
<tr>
<td>7</td>
<td>CLOCK ENABLED</td>
</tr>
<tr>
<td>8</td>
<td>READER OUT-OF-TAPE FLAG</td>
</tr>
<tr>
<td>9</td>
<td>PUNCH OUT-OF-TAPE FLAG</td>
</tr>
<tr>
<td>11</td>
<td>201 DATAPHONE TRANSMIT FLAG</td>
</tr>
<tr>
<td>12</td>
<td>201 DATAPHONE RECEIVE FLAG</td>
</tr>
</tbody>
</table>

201 DATAPHONE STATUS

<table>
<thead>
<tr>
<th>BIT</th>
<th>INTERPRETATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>INTERRUPT PENDING</td>
</tr>
<tr>
<td>1</td>
<td>DATA LOST</td>
</tr>
<tr>
<td>2</td>
<td>PARITY ERROR</td>
</tr>
<tr>
<td>3</td>
<td>REQUEST TO SEND</td>
</tr>
<tr>
<td>4</td>
<td>TRANSMIT REQUEST</td>
</tr>
<tr>
<td>5</td>
<td>CLEAR TO SEND</td>
</tr>
<tr>
<td>6</td>
<td>CHECK PARITY</td>
</tr>
<tr>
<td>7</td>
<td>TEXT MODE</td>
</tr>
<tr>
<td>8</td>
<td>SET READY</td>
</tr>
<tr>
<td>9</td>
<td>TERMINAL READY</td>
</tr>
<tr>
<td>10</td>
<td>RING</td>
</tr>
<tr>
<td>11</td>
<td>CARRIER DETECTED</td>
</tr>
<tr>
<td>12</td>
<td>FRAME SIZE REGISTER BIT 0</td>
</tr>
<tr>
<td>13</td>
<td>FRAME SIZE REGISTER BIT 1</td>
</tr>
<tr>
<td>14</td>
<td>FRAME SIZE REGISTER BIT 2</td>
</tr>
<tr>
<td>15</td>
<td>FRAME SIZE REGISTER BIT 3</td>
</tr>
<tr>
<td>BIT</td>
<td>INTERPRETATION</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------------------------</td>
</tr>
<tr>
<td>16</td>
<td>TRANSMIT STATE</td>
</tr>
<tr>
<td>17</td>
<td>RECEIVE STATE</td>
</tr>
</tbody>
</table>

**DISPLAY STATUS WORD 1**

<table>
<thead>
<tr>
<th>BIT</th>
<th>INTERPRETATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>LIGHT PEN FLAG</td>
</tr>
<tr>
<td>7</td>
<td>VERTICAL EDGE FLAG</td>
</tr>
<tr>
<td>8</td>
<td>HORIZONTAL EDGE FLAG</td>
</tr>
<tr>
<td>9</td>
<td>INTERNAL STOP FLAG</td>
</tr>
<tr>
<td>10</td>
<td>SECTOR 0 FLAG (DISPLAY COORDINATES ARE ON SCREEN)</td>
</tr>
<tr>
<td>11</td>
<td>CONTROL STATE</td>
</tr>
<tr>
<td>12</td>
<td>MANUAL INTERRUPT FLAG</td>
</tr>
<tr>
<td>13</td>
<td>PUSH BUTTON FLAG</td>
</tr>
<tr>
<td>14</td>
<td>DISPLAY INTERRUPT PENDING</td>
</tr>
<tr>
<td>15</td>
<td>BREAK FIELD REGISTER BIT 0</td>
</tr>
<tr>
<td>16</td>
<td>BREAK FIELD REGISTER BIT 1</td>
</tr>
<tr>
<td>17</td>
<td>BREAK FIELD REGISTER BIT 2</td>
</tr>
</tbody>
</table>

**DISPLAY STATUS WORD 2**

<table>
<thead>
<tr>
<th>BIT</th>
<th>INTERPRETATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0 -- LEFT HAND INCREMENT BEING EXECUTED</td>
</tr>
<tr>
<td></td>
<td>1 -- RIGHT HAND INCREMENT BEING EXECUTED</td>
</tr>
<tr>
<td>7</td>
<td>LIGHT PEN ENABLED</td>
</tr>
<tr>
<td>8</td>
<td>BIT 0 OF Y POSITION REGISTER</td>
</tr>
<tr>
<td>9</td>
<td>BIT 0 OF X POSITION REGISTER</td>
</tr>
<tr>
<td>10</td>
<td>SCALE BIT 0</td>
</tr>
<tr>
<td>11</td>
<td>SCALE BIT 1</td>
</tr>
<tr>
<td>12</td>
<td>MODE BIT 0</td>
</tr>
<tr>
<td>13</td>
<td>MODE BIT 1</td>
</tr>
<tr>
<td>14</td>
<td>MODE BIT 2</td>
</tr>
<tr>
<td>15</td>
<td>INTENSITY BIT 0</td>
</tr>
<tr>
<td>16</td>
<td>INTENSITY BIT 1</td>
</tr>
<tr>
<td>17</td>
<td>INTENSITY BIT 2</td>
</tr>
</tbody>
</table>
### Display Initial Conditions

<table>
<thead>
<tr>
<th>BIT</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Enable Edge Flag Interrupt</td>
</tr>
<tr>
<td>7</td>
<td>Enable Light Pen Flag Interrupt</td>
</tr>
</tbody>
</table>
| 8   | 0 -- Do not disable Light Pen after resuming display  
     | 1 -- Enable Light Pen according to Bit 9 |
| 9   | 0 -- Enable Light Pen after first data request after resuming display  
     | 1 -- Do not enable Light Pen after resuming display |
| 10  | Bit 0 of Y Dimension |
| 11  | Bit 1 of Y Dimension |
| 12  | Bit 0 of X Dimension |
| 13  | Bit 1 of X Dimension |
| 14  | Intensify All Points |
| 15  | Inhibit Edge Flags |
| 16  | Enable Push Button Interrupt |
| 17  | Enable Internal Stop Interrupt |

### Break Field Load Parameter

<table>
<thead>
<tr>
<th>BIT</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Load Break Field according to bits 7-9</td>
</tr>
<tr>
<td>7</td>
<td>Break Field Bit 0</td>
</tr>
<tr>
<td>8</td>
<td>Break Field Bit 1</td>
</tr>
<tr>
<td>9</td>
<td>Break Field Bit 2</td>
</tr>
<tr>
<td>10</td>
<td>Load Push Buttons according to bits 11-17</td>
</tr>
</tbody>
</table>
| 11  | 0 -- Load Push Buttons 0-5  
     | 1 -- Load Push Buttons 6-11 |
| 12  | Push Button 0 or 6 |
| 13  | Push Button 1 or 7 |
| 14  | Push Button 2 or 8 |
| 15  | Push Button 3 or 9 |
| 16  | Push Button 4 or 10 |
| 17  | Push Button 5 or 11 |
IOT INSTRUCTIONS

Each IOT instruction is formed by adding the code from the table below to 700000. The AC may be cleared at event time 1 of the IOT instruction by setting bit 14 in the instruction.

<table>
<thead>
<tr>
<th>CODE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0002</td>
<td>ENABLE INTERRUPTS</td>
</tr>
<tr>
<td>0042</td>
<td>DISABLE INTERRUPTS</td>
</tr>
<tr>
<td>0001</td>
<td>SKIP IF CLOCK FLAG IS SET</td>
</tr>
<tr>
<td>0004</td>
<td>CLEAR CLOCK FLAG AND DISABLE CLOCK</td>
</tr>
<tr>
<td>0044</td>
<td>CLEAR CLOCK FLAG AND ENABLE CLOCK</td>
</tr>
<tr>
<td>0101</td>
<td>SKIP IF READER FLAG IS SET</td>
</tr>
<tr>
<td>0102</td>
<td>CLEAR READER FLAG, INCLUSIVE OR CONTENT OF READER BUFFER INTO AC</td>
</tr>
<tr>
<td>0104</td>
<td>SELECT READER IN ALPHANUMERIC MODE</td>
</tr>
<tr>
<td>0144</td>
<td>SELECT READER IN BINARY MODE</td>
</tr>
<tr>
<td>0201</td>
<td>SKIP IF PUNCH FLAG IS SET</td>
</tr>
<tr>
<td>0202</td>
<td>CLEAR PUNCH FLAG</td>
</tr>
<tr>
<td>0206</td>
<td>PUNCH TAPE IMAGE FROM BITS 10-17 OF AC</td>
</tr>
<tr>
<td>0244</td>
<td>PUNCH TAPE IMAGE IN BINARY MODE FROM BITS 12-17 OF AC</td>
</tr>
<tr>
<td>0301</td>
<td>SKIP IF KEYBOARD FLAG IS SET</td>
</tr>
<tr>
<td>0302</td>
<td>OR CONTENT OF KEYBOARD BUFFER INTO BITS 10-17 OF AC</td>
</tr>
<tr>
<td>0304</td>
<td>OR I/O STATUS WORD INTO AC</td>
</tr>
<tr>
<td>CODE</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>0401</td>
<td>SKIP IF TELEPRINTER FLAG IS SET</td>
</tr>
<tr>
<td>0402</td>
<td>CLEAR TELEPRINTER FLAG</td>
</tr>
<tr>
<td>0406</td>
<td>LOAD TELEPRINTER BUFFER FROM BITS 10-17 OF THE AC</td>
</tr>
<tr>
<td>0501</td>
<td>OR DISPLAY PUSH-DOWN POINTER INTO BITS 6-17 OF THE AC</td>
</tr>
<tr>
<td>0502</td>
<td>OR BITS 1-12 OF THE DISPLAY CONTROL X POSITION REGISTER INTO BITS 6-17 OF THE AC</td>
</tr>
<tr>
<td>0601</td>
<td>OR BITS 3-14 OF THE DISPLAY ADDRESS COUNTER INTO BITS 6-17 OF THE AC</td>
</tr>
<tr>
<td>0602</td>
<td>OR DISPLAY STATUS WORD 1 INTO BITS 6-17 OF THE AC</td>
</tr>
<tr>
<td>0621</td>
<td>OR PUSH BUTTONS 0-11 INTO BITS 6-17 OF THE AC</td>
</tr>
<tr>
<td>0642</td>
<td>SKIP IF THE LIGHT PEN FLAG IS SET</td>
</tr>
<tr>
<td>0645</td>
<td>SET DISPLAY PUSH DOWN POINTER FROM BITS 6-17 OF THE AC</td>
</tr>
<tr>
<td>0665</td>
<td>SET DISPLAY INITIAL CONDITIONS FROM BITS 6-17 OF THE AC</td>
</tr>
<tr>
<td>0701</td>
<td>SKIP IF DISPLAY EXTERNAL STOP FLAG IS SET</td>
</tr>
<tr>
<td>0702</td>
<td>SKIP IF EITHER THE VERTICAL OR HORIZONTAL EDGE FLAG IS SET</td>
</tr>
<tr>
<td>0704</td>
<td>STOP DISPLAY (EXTERNAL)</td>
</tr>
<tr>
<td>0705</td>
<td>LOAD BREAK FIELD AND/OR PUSH BUTTONS FROM THE BREAK FIELD PARAMETER IN BITS 6-17 OF THE AC</td>
</tr>
<tr>
<td>0721</td>
<td>SKIP IF DISPLAY INTERNAL STOP FLAG IS SET</td>
</tr>
<tr>
<td>0722</td>
<td>SKIP IF MANUAL INTERRUPT FLAG IS SET</td>
</tr>
<tr>
<td>CODE</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
</tr>
<tr>
<td>1103</td>
<td>SET THE A/D CONVERTER MULTIPLEXOR TO THE CHANNEL SPECIFIED IN BITS 12-17 OF THE AC</td>
</tr>
<tr>
<td>1201</td>
<td>INCREMENT THE A/D CONVERTER MULTIPLEXOR CHANNEL NUMBER (CHANNEL 0 FOLLOWS CHANNEL 77)</td>
</tr>
<tr>
<td>1202</td>
<td>OR A/D CONVERTER MULTIPLEXOR CHANNEL NUMBER INTO BITS 12-17 OF THE AC</td>
</tr>
<tr>
<td>1301</td>
<td>SKIP IF THE A/D CONVERTER FLAG IS SET</td>
</tr>
<tr>
<td>1302</td>
<td>OR A/D CONVERTER BUFFER INTO BITS 0-11 OF THE AC</td>
</tr>
<tr>
<td>1304</td>
<td>SELECT THE A/D CONVERTER</td>
</tr>
<tr>
<td>1401</td>
<td>SKIP IF THE DATAPHONE TRANSMIT FLAG IS SET</td>
</tr>
<tr>
<td>1402</td>
<td>OR THE DATAPHONE STATUS WORD INTO THE AC</td>
</tr>
<tr>
<td>1404</td>
<td>INVERT THE DATAPHONE STATUS BITS WHEREVER A 1 APPEARS IN THE CORRESPONDING POSITION IN THE AC</td>
</tr>
<tr>
<td>1421</td>
<td>SKIP IF DATAPHONE MASK SKIP FLAG IS SET</td>
</tr>
<tr>
<td>1422</td>
<td>SET THE DATAPHONE MASK SKIP FLAG IF ALL BITS IN THE DATAPHONE STATUS WORD ARE 1'S WHEREVER A 1 APPEARS IN THE CORRESPONDING POSITION IN THE AC</td>
</tr>
<tr>
<td>1424</td>
<td>CLEAR DATAPHONE MASK SKIP FLAG</td>
</tr>
<tr>
<td>1441</td>
<td>SKIP IF THE DATAPHONE RECEIVE FLAG IS SET</td>
</tr>
<tr>
<td>1442</td>
<td>CLEAR THE DATAPHONE TRANSMIT AND RECEIVE FLAGS</td>
</tr>
<tr>
<td>1444</td>
<td>CLEAR ALL DATAPHONE FLAGS AND REGISTERS</td>
</tr>
<tr>
<td>1601</td>
<td>CLEAR DISPLAY FLAGS</td>
</tr>
<tr>
<td>CODE</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
</tr>
<tr>
<td>1602</td>
<td>OR BITS 1-12 OF THE DISPLAY Y POSITION REGISTER INTO BITS 6-17 OF THE AC</td>
</tr>
<tr>
<td>1604</td>
<td>RESUME DISPLAY AFTER INTERNAL STOP</td>
</tr>
<tr>
<td>1605</td>
<td>INITIALIZE DISPLAY AT ADDRESS GIVEN IN BITS 6-17 OF THE AC</td>
</tr>
<tr>
<td>1622</td>
<td>OR DISPLAY STATUS WORD 2 INTO BITS 6-17 OF THE AC</td>
</tr>
<tr>
<td>3301</td>
<td>SKIP IF THE TELETYPE IS CONNECTED</td>
</tr>
<tr>
<td>3302</td>
<td>CLEAR ALL FLAGS</td>
</tr>
<tr>
<td>3344</td>
<td>RESTORE THE LINK AND EXTEND MODE STATUS FROM INFORMATION CONTAINED IN THE LOCATION WHOSE ADDRESS IS GIVEN IN BITS 5-17 OF THE FOLLOWING WORD IN MEMORY</td>
</tr>
<tr>
<td>5101</td>
<td>LOAD D/A CONVERTER CHANNEL #1 FROM BITS 0-11 OF THE AC</td>
</tr>
<tr>
<td>5102</td>
<td>LOAD D/A CONVERTER CHANNEL #2 FROM BITS 0-11 OF THE AC</td>
</tr>
<tr>
<td>5104</td>
<td>LOAD D/A CONVERTER CHANNEL #3 FROM BITS 0-11 OF THE AC</td>
</tr>
<tr>
<td>7701</td>
<td>SKIP IF IN EXTEND MODE</td>
</tr>
<tr>
<td>7702</td>
<td>ENTER EXTEND MODE</td>
</tr>
<tr>
<td>7704</td>
<td>LEAVE EXTEND MODE</td>
</tr>
</tbody>
</table>
APPENDIX D -- ASSEMBLY LANGUAGE

THE ASSEMBLY LANGUAGE WHICH IS USED IN THE EXAMPLES IN THE REPORT IS THE SOURCE LANGUAGE FOR THE ASSEMBLER (TO BE DESCRIBED IN A FORTHCOMING REPORT) WHICH RUNS UNDER THE EXECUTIVE SYSTEM. THIS LANGUAGE IS DESCRIBED BRIEFLY BELOW.

ALL MNEMONICS ARE FROM ONE TO SIX CHARACTERS LONG. THE FIRST CHARACTER MUST BE AN ALPHABETIC CHARACTER OR A PERIOD (.), AND ALL OTHER CHARACTERS MUST BE ALPHANUMERIC OR PERIODS. A MNEMONIC MAY REPRESENT ANY ONE OF THE FOLLOWING ENTITIES:

(1) A PROGRAM SYMBOL (I. E., A SYMBOL WHOSE VALUE IS USED TO COMPUTE THE OPERAND OF AN INSTRUCTION),
(2) AN INSTRUCTION CODE, OR
(3) A PSEUDO-OP (I. E., AN INSTRUCTION TO THE ASSEMBLER).

IF A MNEMONIC IS USED TO REPRESENT MORE THAN ONE OF THESE ENTITIES, THE ASSEMBLER WILL RESOLVE THE AMBIGUITY FROM CONTEXT.

ALL NUMBERS ARE INTERPRETED AS OCTAL NUMBERS. NUMBERS MAY REPRESENT VALUES OF PROGRAM SYMBOLS ONLY.

A SOURCE LINE IS COMPOSED OF UP TO FOUR FIELDS. EACH FIELD IS DELIMITED BY SPACES. (SEVERAL CONSECUTIVE SPACES ARE INTERPRETED AS A SINGLE SPACE BY THE ASSEMBLER, EXCEPT IN TEXT PSEUDO-OP OPERANDS.) THE FOUR POSSIBLE FIELDS (FROM LEFT TO RIGHT ON THE SOURCE LINE) ARE THE FOLLOWING:

(1) LOCATION FIELD
(2) INSTRUCTION FIELD
(3) OPERAND FIELD
(4) COMMENT FIELD

THE LOCATION FIELD CONTAINS A MNEMONIC WHICH IS ASSIGNED...

THE INSTRUCTION FIELD CONTAINS ONE OF THE FOLLOWING:

(1) A PSEUDO-OP SYMBOL,

(2) A MNEMONIC WHICH REPRESENTS AN INSTRUCTION WHICH REQUIRES AN OPERAND, OR

(3) AN OPERANDLESS INSTRUCTION MNEMONIC OR A SET OF THESE MNEMONICS SEPARATED BY PLUS SIGNS (+), WHICH DENOTE "INCLUSIVE OR" IN THIS FIELD.

IF THE INSTRUCTION FIELD CONTAINS AN OPERANDLESS INSTRUCTION THE OPERAND FIELD IS NOT PRESENT. INDIRECT ADDRESSING IS INDICATED BY AN ASTERISK (*) APPENDED TO THE RIGHT OF A MNEMONIC WHICH REPRESENTS AN INSTRUCTION WHICH REQUIRES AN OPERAND.


PSEUDO-OP SYMBOLS ARE WRITTEN IN THE INSTRUCTION FIELD AND CONSIST OF A DOLLAR SIGN ($) APPENDED TO THE LEFT OF THE PSEUDO-OP MNEMONIC. THE FOLLOWING SYMBOLS ARE ACCEPTED BY THE ASSEMBLER:
SDC A WORD WHICH CONTAINS THE FULL 18-BIT VALUE OF THE
EXPRESSION IN THE OPERAND FIELD IS PRODUCED.

SDS THE 18-BIT VALUE OF THE EXPRESSION IN THE OPERAND FIELD
IS ADDED INTO THE LOCATION COUNTER WITHIN THE ASSEMBLER
(BY TWO'S COMPLEMENT ADDITION). (ALL MNEMONICS IN THE
OPERAND FIELD MUST BE PREDEFINED.)

SEND THE END OF THE SOURCE PROGRAM IS DECLARED.

SEOU THE PROGRAM SYMBOL MNEMONIC IN THE LOCATION FIELD IS
ASSIGNED THE 18-BIT VALUE OF THE EXPRESSION IN THE
OPERAND FIELD. (ALL MNEMONICS IN THE OPERAND FIELD MUST
BE PREDEFINED.)

$OPD THE OPERAND-LESS INSTRUCTION MNEMONIC IN THE LOCATION
FIELD IS ASSIGNED THE 18-BIT VALUE OF THE EXPRESSION IN
THE OPERAND FIELD. (ALL MNEMONICS IN THE OPERAND FIELD
MUST BE PREDEFINED.)

$OPDM THE OPERAND-REQUIRING INSTRUCTION MNEMONIC IN THE
LOCATION FIELD IS ASSIGNED THE 18-BIT VALUE OF THE
EXPRESSION IN THE OPERAND FIELD. (ALL MNEMONICS IN THE
OPERAND FIELD MUST BE PREDEFINED.)

$ORG THE LOCATION COUNTER WITHIN THE ASSEMBLER IS SET TO THE
18-BIT VALUE OF THE EXPRESSION IN THE OPERAND FIELD. (ALL
MNEMONICS IN THE OPERAND FIELD MUST BE PREDEFINED.)

$TEXT THE FIRST CHARACTER IN THE OPERAND FIELD IS TAKEN AS A
BREAK CHARACTER, AND ALL CHARACTERS TO THE RIGHT OF IT
UP TO THE NEXT BREAK CHARACTER ARE PACKED AS 3 6-BIT
CHARACTER CODES PER WORD. IF THE NUMBER OF CHARACTERS
BETWEEN THE BREAK CHARACTERS IS NOT A MULTIPLE OF 3, THE
LAST WORD GENERATED IS PADDED WITH NULL CHARACTER CODES
(77).

$TITLE ALL CHARACTERS TO THE RIGHT OF THIS PSEUDO-OP ARE TAKEN
TO BE THE TITLE OF THE CURRENT SECTION OF THE PROGRAM.
(THIS TITLE IS TYPED ON THE TELETYPewriter DURING PASS 1 OF THE
ASSEMBLY, BEGINNING WITH THE FIRST NON-BLANK CHARACTER.)

THE ASSEMBLER IGNORES SOURCE LINES WHICH BEGIN WITH AN
ASTERISK (*), SOURCE LINES WHICH HAVE NO FIELDS, AND COMMENT
FIELDS.