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

J. mes 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 17732\(_g\), 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 17732\(_g\),...17777\(_g\), 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 0-11777, 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 12000-17777 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:

![Diagram showing System states and tasks]

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 structure]

The symbols in the diagram are interpreted as follows:

- **q₀** = Address of the word queue. By convention, this is the address of the first control word.
- **q₁** = Pointer to the physically last data word in the queue.
- **q₂** = Pointer to the last word put into the queue (FIFO sense).
- **q₃** = 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

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

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

\[
qu'' = \begin{cases} 
qu - 1, & \text{if } q \neq q_0 + 3 \\
qu_1, & \text{if } q = q_0 + 3
\end{cases}
\]

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 \(n + 2\):

\[
Q \text{ $DC$ } *+\epsilon \\
$DS \epsilon
\]

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.)

<table>
<thead>
<tr>
<th>TASK</th>
<th>LAC = FIFO</th>
<th>GET ADDRESS OF FIFO QUEUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOOP1</td>
<td>LAC = COUNT</td>
<td>GET VALUE OF INTEGER</td>
</tr>
<tr>
<td></td>
<td>LMQ</td>
<td>SET UP PARAMETER</td>
</tr>
<tr>
<td></td>
<td>LAC = FIFO</td>
<td>GET ADDRESS OF FIFO QUEUE</td>
</tr>
</tbody>
</table>
3.2 Task Scheduling and I/O Device Allocation

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 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.
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 \( t_{14} \rightarrow \infty \) 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:**

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

**Calling sequence for T.U:**

```assembly
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:

```assembly
WAIT $DC 0
JMS* =T.L SET REENTRY LOCK
$DC 0 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 0-9 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.
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>Second Octal Digit</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td>3</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
</tr>
<tr>
<td>4</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>*</td>
<td>/</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>(</td>
<td>)</td>
<td>[</td>
<td>]</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>†</td>
</tr>
<tr>
<td>6</td>
<td>†</td>
<td>†</td>
<td>†</td>
<td>†</td>
<td>†</td>
<td>†</td>
<td>†</td>
<td>†</td>
</tr>
<tr>
<td>7</td>
<td>&quot;</td>
<td>$</td>
<td>#</td>
<td>&amp;</td>
<td>cr</td>
<td>lf</td>
<td>sp</td>
<td></td>
</tr>
</tbody>
</table>

cr = carriage return
lf = line feed
sp = space

All ASCII characters which do not appear in the table are mapped into 77. 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 ($026_{8}$). (At least two are required when receiving; eight are transmitted.)

2. A data link escape (DLE) character ($220_{8}$).

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 ($377_{8}$).

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\_8) or an end-of-transmission (204\_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):

<table>
<thead>
<tr>
<th>TASK</th>
<th>LAW</th>
<th>JMS*</th>
<th>DAC</th>
<th>READ</th>
<th>START</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIRROR</td>
<td>600</td>
<td>=T.A</td>
<td>COUNT</td>
<td>=B.FI</td>
<td>=B.FI</td>
</tr>
<tr>
<td></td>
<td>17700</td>
<td></td>
<td></td>
<td>JMP</td>
<td>JMP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>HELP</td>
<td>HELP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>=B.FI</td>
<td>=B.FI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>HELP</td>
<td>HELP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DATA SET NOT CONNECTED</td>
<td>DATA SET NOT CONNECTED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>START</td>
<td>READ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>=B.FO</td>
<td>NEXT RECORD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>JMP</td>
<td>GET INPUT IMAGE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>HELP</td>
<td>DATA SET NOT CONNECTED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ISZ</td>
<td>COUNT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>COUNT</td>
<td>DATA SET NOT CONNECTED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>JMP</td>
<td>READ</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>HELP</td>
<td>NEXT IMAGE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>=B.FI</td>
<td>GET INPUT IMAGE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>JMP</td>
<td>DATA SET NOT CONNECTED</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Control Bits</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>mode change</td>
</tr>
<tr>
<td>01</td>
<td>binary origin</td>
</tr>
<tr>
<td>10</td>
<td>binary data</td>
</tr>
<tr>
<td>11</td>
<td>alphanumeric data</td>
</tr>
</tbody>
</table>

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. (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 should contain 621365 and that location 23573 should contain 176234 is the following:

```
102  origin 23572
135
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 (000B).

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 (000B).

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  LAW  140  GET ALLOCATION MASK
      JMS* =T.A  ALLOCATE READER & PUNCH
      JMS* =B.R  GET IMAGE FROM READER
      JMP  RERR  READER OUT OF TAPE
      SNA  *+4   SKIP IF NOT END OF RECORD
      JMP  *+4   END OF RECORD
      JMS* =B.P  PUNCH IMAGE
      JMP  PERR  PUNCH OUT OF TAPE
      JMP  COPY+2 READ NEXT IMAGE
      JMS* =B.P  PUNCH END OF RECORD
      JMP  PERR  PUNCH OUT OF TAPE
      LAW  140  GET ALLOCATION MASK
      JMS* =T.R  RELEASE READER & PUNCH
      JMS* =T.F  TERMINATE TASK
RERR  LAC  =RERRT  GET ADDRESS OF TEXT LIST
      SKP  TYPE DIAGNOSTIC
PERR  LAC  =PERRT  GET ADDRESS OF TEXT LIST
      DAC  TEXT  SAVE ADDRESS OF TEXT LIST
      LAW  140  GET ALLOCATION MASK
      JMS* =T.R  RELEASE READER & PUNCH
      LAW  10   GET ALLOCATION MASK
      JMS* =T.A  ALLOCATE TELEPRINTER
      LAC  TEXT  GET ADDRESS OF TEXT LIST
      JMS* =L.T  TYPE DIAGNOSTIC
      LAC  =END  GET ADDRESS OF TEXT LIST
```
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 \(77_8\) 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:

```
ENCOD       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
DZM    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:

```assembly
JMS* =T.S SCHEDULE KEYBOARD CHECKER
LAC =TXT1 GET ADDRESS OF TEXT LIST
JMS* =L.T TYPE "PLEASE TYPE NOW"
LAW -1000 GET TIME PARAMETER
JMS* =N.C WAIT ABOUT 8 SECONDS
LAC DONE GET KEYBOARD RESPONSE SWITCH
SNA SKIP IF RESPONSE OBTAINED
JMP PROMPT+5 PROMPT OPERATOR AGAIN
JMS* =T.F TERMINATE EXECUTION

POLITE
JMS* =B.K GET KEYBOARD CHARACTER
XOR =777700 PRECEDE WITH NULL CHARACTERS
DAC DONE SET KEYBOARD RESPONSE SWITCH
SAD =777774 SKIP IF NOT CARRIAGE RETURN
JMP *+3 END OF INPUT LINE
JMS* =B.T ECHO CHARACTER ON TELEPRINTER
JMP POLITE GET ANOTHER CHARACTER
LAC =TXT2 GET ADDRESS OF TEXT LIST
JMS* =L.T TYPE "THANK YOU"
LAW 30 GET ALLOCATION MASK
JMS* =T.R RELEASE KEYBOARD AND TELEPRINTER
JMS* =T.F TERMINATE EXECUTION

TXT1 $DC 6
$TEXT "PLEASE TYPE NOW"
$DC 747577

TXT2 $DC 5
$DC 747577
$TEXT "THANK YOU"
$DC 747577
```

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    =STATE 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    =P.S  CLEAR BUTTONS
         CLA    =P.S  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:

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.)

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:
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

\[
\begin{align*}
(1) & \quad \forall x, y \in X \quad x + y \in X \\
(2) & \quad \forall x, y \in X \quad x \leq x + y \quad \text{and} \quad y \leq x + y \\
\text{and} & \quad \forall x, y, z \in X \quad x < z \quad \text{and} \quad y < z \implies x + y < z,
\end{align*}
\]

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:

![Drawing of a house with windows]

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,</td>
</tr>
<tr>
<td></td>
<td>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.
S.LN - 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
-45-

LAC* LVL4 GET POINTER TO OWNER
JMS* =S.TI INSERT CREATED LEVEL
JMP LVL1 DISPLAY STORAGE EXCEEDED
LAC LVL5 GET POINTER TO CREATED LEVEL
JMP LVL3+2 RETURN

LVL1
LAC LVL5 GET POINTER TO LEVEL
JMS* =S.TI SET UP PARAMETER
$DC 0 LVL PROGRAMMING ERROR
LMQ
LAC LVL5 GET POINTER TO LEVEL
JMS* =S.TR REMOVE ATTRIBUTE
$DC 0 LVL PROGRAMMING ERROR
LAC LVL5 GET POINTER TO LEVEL
JMS* =S.TD DESTROY LEVEL
$DC 0 LVL PROGRAMMING ERROR
JMP LVL3+3 RETURN

LVL2
LAC LVL5 GET POINTER TO LEVEL
JMS* =S.TD DESTROY LEVEL
$DC 0 LVL PROGRAMMING ERROR

LVL3
ISZ LVL+2 INCREMENT RETURN POINTER
ISZ LVL+2 INCREMENT RETURN POINTER
ISZ LVL+2 INCREMENT RETURN POINTER
ISZ LVL+2 INCREMENT RETURN POINTER
JMS* =T.U UNLOCK LVL & RETURN
$DC LVL

*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 SET REENTRY LOCK
    $DC  0
    DAC BUTN3 SAVE POINTER TO STRUCTURE
    LAW -4 GET LVL PARAMETER COUNT
    DAC BUTN4 INITIALIZE COUNTER
    LAC =BUTN1 GET ADDRESS OF FIRST LVL PARAMETER
    DAC BUTN5 INITIALIZE POINTER
    LAC* BUTN3 GET BUTN PARAMETER
    DAC* BUTN+2 STORE AS LVL PARAMETER
    ISZ BUTN+2 INCREMENT BUTN PARAMETER POINTER
    ISZ BUTN5 INCREMENT LVL PARAMETER POINTER
    ISZ BUTN4 INCREMENT COUNTER & SKIP IF DONE
    JMP *-5 COPY NEXT PARAMETER
    LAC BUTN3 GET POINTER TO STRUCTURE
    JMS LVL GENERATE INTERMEDIATE LEVEL

BUTN1 $DC  0 LOC CONTAINING POINTER TO OWNER
    $DC  0 Y COORDINATE
    $DC  0 X COORDINATE
    $DC  0 DISPLAY PARAMETER
    JMP BUTN2 DISPLAY STORAGE EXCEEDED
    DAC BUTN3 SAVE POINTER TO LEVEL
    LAC* BUTN+2 GET ADDRESS OF SERVICE TASK
    LMQ SET UP PARAMETER
    LAC BUTN3 GET POINTER TO LEVEL
    JMS* =S.LL SENITIZE LEVEL TO LPSI
    LAC BUTN3 GET POINTER TO LEVEL
    ISZ BUTN+2 INCREMENT RETURN POINTER

BUTN2 ISZ BUTN+2 INCREMENT RETURN POINTER
    JMS* =T.U UNLOCK BUTN & RETURN
    $DC BUTN
CALLING SEQUENCE:

* JMS BUTX
  (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:

* POINTING TO LIGHT BUTTON LEVEL

BUTX $DC 0
JMS* =T.L
$DC 0
LAC* BUTX+2
JMS* =L.D
JMP BUTX4
DAC BUTX7
LAW -6
DAC BUTX5
LAC =BUTX2
DAC BUTX6
BUTX1 ISZ BUTX+2
ISZ BUTX6
ISZ BUTX5
SKP
JMP BUTX2-1
LAC* BUTX+2
DAC* BUTX6
JMP BUTX1
LAC BUTX7
BUTX2 JMS BUTN
$DC 0
$DC 0
$DC 0 X COORDINATE
$DC 0 DISPLAY PARAMETER
$DC 0 SERVICE TASK ADDRESS
JMP BUTX3+2 DISPLAY STORAGE EXCEEDED
IS2 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.

CHEW $DC 0
JMS* =T.L SET REENTRY LOCK
$DC 0
DAC CHEW6 SAVE POINTER TO STRUCTURE
LAC =CHEWQ GET ADDRESS OF WORD QUEUE
JMS* =Q.C CLEAR WORD QUEUE
CHEW1 LAC* CHEW6 GET FIRST WORD FROM STRUCTURE
SNA SKIP IF ITEM NOT ALREADY DELETED
JMP CHEW5 GET NEXT ITEM FROM QUEUE
SAD =762010 SKIP IF NOT TEXT LEAF
JMP CHEW4 DESTROY TEXT LEAF
LAC CHEW6 GET POINTER TO STRUCTURE
AND =70000 GET BREAK FIELD BITS
SAD =10000 SKIP IF NOT LEVEL
SKP
JMP CHEW5
CHEW2
LAC CHEW5
JMP CHEW2
JMS Attr
JMP CHEW3
DAC CHEW7
LMQ
LAC CHEW6
JMP CHEW3
JMS* =S.TR
$DC 0
DAC CHEW6
JMP CHEW1
JMS* =T.U
$DC CHEW
CHEW6
JMP CHEW2
JMS* =Q.A
$DC 0
LOMQ
LAC =CHEWQ
JMP CHEW2
JMS* =Q.F
JMP **+3
DAC CHEW6
JMP CHEW1
JMS* =S.TR
$DC 0
$DC +200
CHEWQ
*CALLING SEQUENCE:
*     JMS Attr
*     ------  (RETURN IF NO MORE ATTRIBUTES)
*     ------  (RETURN IF ATTRIBUTE FOUND)
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:
The elements of this display are arranged in the following structure:

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

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:

```
            /
           /
          /
         /
        /
       /
      /
     /
    /
   /
  /
/
```

first vector

(0,0)

third vector

second vector
<table>
<thead>
<tr>
<th>SELGI</th>
<th>LAW</th>
<th>1</th>
<th>GET DISPLAY ALLOCATION MASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS*</td>
<td>=T.A</td>
<td></td>
<td>ALLOCATE DISPLAY</td>
</tr>
<tr>
<td>LAC</td>
<td>=LINES</td>
<td></td>
<td>GET ADDRESS OF LINE STORAGE AREA</td>
</tr>
<tr>
<td>DAC</td>
<td>DIS</td>
<td></td>
<td>SET STORAGE POINTER</td>
</tr>
<tr>
<td>LAW</td>
<td>-1000</td>
<td></td>
<td>LOAD AC WITH -512</td>
</tr>
<tr>
<td>DAC</td>
<td>FRM</td>
<td></td>
<td>SET STORAGE COUNTER</td>
</tr>
<tr>
<td>D2M*</td>
<td>DIS</td>
<td></td>
<td>CLEAR STORAGE LOCATION</td>
</tr>
<tr>
<td>ISZ</td>
<td>DIS</td>
<td></td>
<td>INCREMENT STORAGE POINTER</td>
</tr>
<tr>
<td>ISZ</td>
<td>FRM</td>
<td></td>
<td>SKIP IF STORAGE AREA CLEARED</td>
</tr>
<tr>
<td>JMP</td>
<td>*=3</td>
<td></td>
<td>CLEAR NEXT STORAGE LOCATION</td>
</tr>
<tr>
<td>JMS*</td>
<td>=S.LH</td>
<td></td>
<td>GET ADDRESS OF HIGHEST ACTIVE LEVEL</td>
</tr>
<tr>
<td>DAC</td>
<td>HAL</td>
<td></td>
<td>SAVE ADDRESS OF HIGHEST ACTIVE LEVEL</td>
</tr>
<tr>
<td>LAC</td>
<td>=TXT</td>
<td></td>
<td>GET ADDRESS OF TITLE TEXT LIST</td>
</tr>
<tr>
<td>JMS*</td>
<td>=L.D</td>
<td></td>
<td>CREATE TEXT LEAF</td>
</tr>
<tr>
<td>JMP</td>
<td>END</td>
<td></td>
<td>DISPLAY STORAGE EXCEEDED</td>
</tr>
<tr>
<td>DAC</td>
<td>DIS</td>
<td></td>
<td>SAVE POINTER TO TITLE LEAF</td>
</tr>
<tr>
<td>JMS</td>
<td>LVL</td>
<td></td>
<td>GENERATE SELGI DISPLAY LEVEL</td>
</tr>
<tr>
<td>$DC</td>
<td>HAL</td>
<td></td>
<td>POINTER TO HIGHEST ACTIVE LEVEL</td>
</tr>
<tr>
<td>$DC</td>
<td>360</td>
<td></td>
<td>Y COORDINATE</td>
</tr>
<tr>
<td>$DC</td>
<td>-34</td>
<td></td>
<td>X COORDINATE</td>
</tr>
<tr>
<td>$DC</td>
<td>500</td>
<td></td>
<td>SCALE X2</td>
</tr>
<tr>
<td>JMP</td>
<td>FAIL</td>
<td></td>
<td>DISPLAY STORAGE EXCEEDED</td>
</tr>
<tr>
<td>DAC</td>
<td>FRM</td>
<td></td>
<td>SAVE POINTER TO SELGI DISPLAY LEVEL</td>
</tr>
<tr>
<td>JMS</td>
<td>BUTX</td>
<td></td>
<td>GENERATE DRAW LIGHT BUTTON</td>
</tr>
<tr>
<td>$DC</td>
<td>TXT1</td>
<td></td>
<td>DRAW TEXT LIST</td>
</tr>
<tr>
<td>$DC</td>
<td>FRM</td>
<td></td>
<td>POINTER TO SELGI DISPLAY LEVEL</td>
</tr>
<tr>
<td>$DC</td>
<td>-750</td>
<td></td>
<td>Y COORDINATE</td>
</tr>
<tr>
<td>$DC</td>
<td>-344</td>
<td></td>
<td>X COORDINATE</td>
</tr>
<tr>
<td>$DC</td>
<td>0</td>
<td></td>
<td>NULL DISPLAY PARAMETER</td>
</tr>
<tr>
<td>$DC</td>
<td>DRAW</td>
<td></td>
<td>DRAW SERVICE TASK</td>
</tr>
<tr>
<td>JMP</td>
<td>END</td>
<td></td>
<td>DISPLAY STORAGE EXCEEDED</td>
</tr>
<tr>
<td>JMS</td>
<td>BUTX</td>
<td></td>
<td>GENERATE ERASE LIGHT BUTTON</td>
</tr>
<tr>
<td>$DC</td>
<td>TXT2</td>
<td></td>
<td>ERASE TEXT LIST</td>
</tr>
<tr>
<td>$DC</td>
<td>FRM</td>
<td></td>
<td>POINTER TO SELGI DISPLAY LEVEL</td>
</tr>
<tr>
<td>$DC</td>
<td>-750</td>
<td></td>
<td>Y COORDINATE</td>
</tr>
</tbody>
</table>
$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 FRM 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.IX 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
LAC  ESCAPE+1  GET ESCAPE SWITCH
S2A  SKIP IF ESCAPE NOT PENDING
JMP  END  TERMINATE SELGI
SKP  PREPARE TO SCHEDULE NEXT LOCATION
JMP  *-4  CHECK ESCAPE SWITCH
JMS* =T.P  SCHEDULE PREVIOUS LOCATION
FAIL  LAC  DIS  GET POINTER TO NONACTIVE STRUCTURE
   JMS  CHEW  DESTROY NONACTIVE STRUCTURE
END  LAC  HAL  GET POINTER TO HIGHEST ACTIVE LEVEL
   JMS  CHEW  DESTROY ACTIVE STRUCTURE
   CLA  GET NULL LIGHT PEN FLAG SERVICE
   JMS* =D.P  SET NULL LIGHT PEN SERVICE
   LAW  1  GET DISPLAY ALLOCATION MASK
   JMS* =T.R  RELEASE DISPLAY
   JMA* =T.F  TERMINATE
DRAW  LAW  -720  GET INITIAL X TRACKING COORDINATE
   LMQ  SET UP PARAMETER
   LAW  -730  GET INITIAL Y TRACKING COORDINATE
   JMS* =X.I  INITIALIZE TRACKING
   CLA  PREPARE TO READ OWNER 0 LEVELS BACK
   JMS* =D.O  READ ADDRESS OF DRAW LEVEL
   $DC  0  PROGRAMMING ERROR IF D.O FAILS
   JMS* =S.LBE  ENABLE BLINK ON DRAW LIGHT BUTTON
   JMS* =X.S  SKIP IF TRACKING HAS BEEN LOST
   JMS* =T.P  CHECK TRACKING AGAIN
   JMS* =X.Y  READ Y TRACKING COORDINATE
   TAD =700  FORM THRESHOLD CHECK
   SPA  SKIP IF LINE IS TO BE CREATED
   JMP  DRAW1  IGNORE ATTEMPT TO CREATE LINE
   LAC =LINES  GET POINTER TO LINE STORAGE
   DAC  FRM  SET STORAGE POINTER
   LAW  -100  GET MAXIMUM LINE COUNT
   DAC  CNT  SET LINE COUNTER
   LAC*  FRM  GET FIRST WORD OF LINE BLOCK
   SNA  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 GET VEC 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
$DC 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
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   MOVEY
DAC   MOVE4
LAC*  MOVE3
JMS*  =C.CB
LLSS  1
TAD   MOVEY
GSM   MOVEY
CMA   MOVEY
TAD   MOVE4
SMA   MOVEY
JMP   *+3
JMS   WATCH
JMS   FIXBGN
JMS   WATCH
JMS   FIXEND
LAC   MOVE1
XCT*  WATCH
JMS*  =X.S
JMP   *+6
LAW   -40
JMS*  =N.C
$DC   0
LAC   MOVE1
XCT*  WATCH
JMS*  =X.S
JMP   *+6
LAW   -40
JMS*  =N.C

-57-

PREPARE TO READ OWNER 0 LEVELS BACK
READ ADDRESS OF LINE LEAF
PROGRAMMING ERROR IF D.O FAILS
SAVE POINTER TO LINE LEAF
SAVE COY 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

FIXBGN

$DC 0  
SET UP POINTERS FOR FIXING LEAF
-59-

LAC  FIXY  GET Y TRACKING COORDINATE
DAC* FIX1  SET FIRST Y DISPLACEMENT
LAC  FIXX  GET X TRACKING COORDINATE
DAC* FIX2  SET FIRST X DISPLACEMENT
JMS  FIXFIX  CORRECT INTENSIFIED VECTOR
JMP* FIXBGN  RETURN

FIXEND  $DC  0

JMS  FIXRD  SET UP POINTERS FOR FIXING LEAF
LAC  FIXY  GET Y TRACKING COORDINATE
XOR  =2000  INVERT SIGN BIT
DAC* FIX5  SET THIRD Y DISPLACEMENT
LAC  FIXX  GET X TRACKING COORDINATE
XOR  =6000  INVERT SIGN BIT, SET ESCAPE BIT
DAC* FIX6  SET THIRD X DISPLACEMENT
JMS  FIXFIX  CORRECT INTENSIFIED VECTOR
JMP* FIXEND  RETURN

FIXRD  $DC  0

TAD  =1  FORM POINTER TO FIRST Y DISPLACEMENT
DAC  FIX1  SAVE
TAD  =1  FORM POINTER TO FIRST X DISPLACEMENT
DAC  FIX2  SAVE
TAD  =1  FORM POINTER TO SECOND Y DISPLACEMENT
DAC  FIX3  SAVE
TAD  =1  FORM POINTER TO SECOND X DISPLACEMENT
DAC  FIX4  SAVE
TAD  =1  FORM POINTER TO THIRD Y DISPLACEMENT
DAC  FIX5  SAVE
TAD  =1  FORM POINTER TO THIRD X DISPLACEMENT
DAC  FIX6  SAVE
JMS* =X.Y  READ Y TRACKING COORDINATE
LRSS  1  DIVIDE BY 2
JMS* =C.BC  CONVERT TO DISPLAY COORDINATE
DAC  FIXY  SAVE
JMS* =X.X  READ X TRACKING COORDINATE
LRSS  1  DIVIDE BY 2
-60-

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

\[ \text{A SIMPLE EXAMPLE} \]

is the following (in octal form):

\[
\begin{align*}
000006 \\
127634 \\
222631 \\
251676 \\
164112 \\
263125 \\
167777
\end{align*}
\]

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

\[
\text{LIST } \quad \$\text{DC} \quad 6 \\
\$\text{TEXT } \quad "\text{A SIMPLE EXAMPLE}" 
\]

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

\[
\begin{align*}
\text{TELETYPE} & \quad \text{FROM} \quad \text{PAPER TAPE} \quad \text{TO} \quad \text{TELETYPE} \\
\text{CORE} & \quad \text{PAPER TAPE} \quad \text{CORE} \quad \text{DISPLAY}
\end{align*}
\]

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**

<table>
<thead>
<tr>
<th>Location</th>
<th>Octal Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>23571</td>
<td>172356</td>
</tr>
<tr>
<td>23572</td>
<td>543125</td>
</tr>
<tr>
<td>23573</td>
<td>601241</td>
</tr>
<tr>
<td>23574</td>
<td>760001</td>
</tr>
<tr>
<td>23575</td>
<td>127123</td>
</tr>
</tbody>
</table>

(chars return)

**FROM TELETYPE TO CORE**

<table>
<thead>
<tr>
<th>Location</th>
<th>Octal Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>23571</td>
<td>172356</td>
</tr>
<tr>
<td>23572</td>
<td>543125</td>
</tr>
<tr>
<td>23574</td>
<td>760001</td>
</tr>
<tr>
<td>23575</td>
<td>127123</td>
</tr>
</tbody>
</table>

(chars return)

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 $2357_{10}$ through $23602_{10}$ on the teletype as follows:

FROM CORE TO TELETYPE
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 $20052_{10}$ through $20056_{10}$ may be moved into locations $21521_{10}$ through $21525_{10}$ 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 TELETYPE 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\textsubscript{g} into location 20001\textsubscript{g}, then it moves the zero from location 20001\textsubscript{g} into location 20002\textsubscript{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\textsubscript{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:

\texttt{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\textsubscript{g} may be scheduled by the following command:

\texttt{SCHEDULE 20571}

4.3 Clearing the Task Queue or Display Storage

The command

\texttt{CLEAR} \begin{verbatim} TASK QUEUE \end{verbatim}
\begin{verbatim} DISPLAY STORAGE \end{verbatim}

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

\texttt{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 dataphone to a remote computer from the teletype are record-oriented, rather than character-oriented, and since ASCII codes are accepted 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 timesharing 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 700000
HLT  $OPD 740040

IOT 3302  CLEAR ALL FLAGS
JMP SYSTEM  START SYSTEM

*1
JMS  .4  READ FIRST LINE OF 3-LINE BLOCK
SNA  SKIP IF NONBLANK TAPE
JMP  .2  BLANK TAPE -- TRY AGAIN
DAC  .5  SAVE FIRST LINE IMAGE
AND  .7  REMOVE DATA BITS
SAD  .8  SKIP IF DATA LINE
SKP  

*2
JMS  .3  FINISH ORIGIN WORD
DAC  .6  SET LOCATION COUNTER
JMP  .1  READ NEXT BLOCK

*3
SDC 0  READ SECOND LINE
LRS  6  SHIFT DATA BITS INTO MO
LAC  .5  LOAD AC WITH FIRST LINE IMAGE
LLS  6  SHIFT CONCATENATED IMAGE INTO AC
DAC  .5  SAVE CONCATENATED FIRST TWO LINES
JMS  .4  READ THIRD LINE
LRS  6  SHIFT DATA BITS INTO MO
LAC  .5  LOAD AC WITH CONCATENATED IMAGE
LLS  6  SHIFT COMPLETED WORD INTO AC
JMP* .3  RETURN

*4
SDC 0
IOT 104  SELECT READER
IOT 101  SKIP IF LINE READY
WAIT FOR FLAG
OVERRIDDEN "JMP .1-2" WHEN LOADED
RETURN

JMP .1
OVERRIDES BOOTSTRAP LOCATION 0
STITLE
SORG 1
JMP 1

SORG 21
JMP ET
JMP ES

SORG 100

Q.C SDC 0
JMP QC

Q.A SDC 0
JMP QA

Q.I SDC 0
JMP QI

Q.F SDC 0
JMP QF

T.S SDC 0
JMP TS

T.P SDC 0
JMP TP

T.F SDC 0
JMP TF

T.A SDC 0
JMP TA

T.R SDC 0
JMP TR

T.L SDC 0
JMP TL

T.U SDC 0
JMP TU

C.B6 SDC 0
JMP CB6

C.6A SDC 0
JMP C6A

C.A6 SDC 0
JMP CA6

C.CB SDC 0
JMP CCB

CONTROL DISPATCHER
INTERRUPT TRAP
ILLEGAL INSTRUCTION TRAP
SYSTEM RESTART
CLEAR QUEUE
ADD WORD TO BOTTOM OF QUEUE (F)
INSERT WORD ON TOP OF QUEUE (F)
FETCH WORD FROM TOP OF QUEUE (F)
SCHEDULE TASK
SCHEDULE PREVIOUS LOC & TERMINATE
TERMINATE CURRENT TASK
ALLOCATE I/O DEVICES UNDER MASK
RELEASE I/O DEVICES UNDER MASK
LOCK REENTRABLE SUBROUTINE
UNLOCK REENTRABLE SUBROUTINE
CONVERT BINARY TO 6-BIT OCTAL
CONVERT 6-BIT TO ASCII
CONVERT ASCII TO 6-BIT
CONVERT DISPLAY COORDINATE TO BINARY
CONVERT BINARY TO DISPLAY COORDINATE
GET IMAGE FROM 201 INPUT BUFFER (F)
SEND IMAGE TO 201 OUTPUT BUFFER (F)
GET IMAGE FROM READER BUFFER (F)
SEND IMAGE TO PUNCH BUFFER (F)
GET 6-BIT CHAR FROM KEYBOARD BUFFER
SEND 3 PACKED 6-BIT CHARs TO TP BUF
CONVERT ANALOG TO DIGITAL
SET CLOCK INTERVAL & SERVICE TASK
SELECT D/A CONVERTER #1
SELECT D/A CONVERTER #2
SELECT D/A CONVERTER #3
SET PUSH BUTTON SERVICE TASK
ENABLE MANUAL OPN OF PUSH BUTTONS
DISABLE MANUAL OPN OF PUSH BUTTONS
READ PUSH BUTTONS
SET PUSH BUTTONS
ENABLE DISPLAY INTERRUPTS
DISABLE DISPLAY INTERRUPTS
SET LIGHT PEN FLAG SERVICE TASK
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D.A</td>
<td>JMP DA READ DISPLAY ADR ON LAST INTERRUPT</td>
</tr>
<tr>
<td>D.Y</td>
<td>JMP DY READ Y DPY COORD ON LAST INTERRUPT</td>
</tr>
<tr>
<td>D.X</td>
<td>JMP DX READ X DPY COORD ON LAST INTERRUPT</td>
</tr>
<tr>
<td>D.O</td>
<td>JMP DO READ OWNER ON LAST INTERRUPT (F)</td>
</tr>
<tr>
<td>X.I</td>
<td>JMP XI INITIALIZE TRACKING AT GIVEN COORDS</td>
</tr>
<tr>
<td>X.R</td>
<td>JMP XR RESUME TRACKING</td>
</tr>
<tr>
<td>X.T</td>
<td>JMP XT TERMINATE TRACKING</td>
</tr>
<tr>
<td>X.S</td>
<td>JMP XS SKIP IF TRACKING NOT IN PROCESS (F)</td>
</tr>
<tr>
<td>X.Y</td>
<td>JMP XY READ Y TRACKING COORDINATE</td>
</tr>
<tr>
<td>X.X</td>
<td>JMP XX READ X TRACKING COORDINATE</td>
</tr>
<tr>
<td>S.TL</td>
<td>JMP STL CREATE A LEVEL (F)</td>
</tr>
<tr>
<td>S.TD</td>
<td>JMP STD DESTROY A LEVEL (F)</td>
</tr>
<tr>
<td>S.TI</td>
<td>JMP STI INSERT SUBSTRUCTURE INTO LEVEL (F)</td>
</tr>
<tr>
<td>S.TR</td>
<td>JMP STR REMOVE SUBSTRUCTURE FROM LEVEL (F)</td>
</tr>
<tr>
<td>S.LH</td>
<td>JMP SLH GET ADDRESS OF HIGHEST ACTIVE LEVEL</td>
</tr>
<tr>
<td>S.LY</td>
<td>JMP SLY TRANSLATE LEVEL IN Y DIRECTION</td>
</tr>
<tr>
<td>S.LX</td>
<td>JMP SLX TRANSLATE LEVEL IN X DIRECTION</td>
</tr>
<tr>
<td>S.LP</td>
<td>JMP SLP SET LEVEL PARAMETERS</td>
</tr>
<tr>
<td>S.LBE</td>
<td>JMP SLBE ENABLE BLINK ON LEVEL</td>
</tr>
<tr>
<td>S.LBD</td>
<td>JMP SLBD DISABLE BLINK ON LEVEL</td>
</tr>
<tr>
<td>Instruction</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>S.C SDC 0</td>
<td>COUNT SCALE AND/OR INTENSITY</td>
</tr>
<tr>
<td>S.LU SDC 0</td>
<td>INTERRUPT UNCONDITIONALLY ON LEVEL</td>
</tr>
<tr>
<td>S.LS SDC 0</td>
<td>INTERRUPT ON LEVEL IF ON SCREEN</td>
</tr>
<tr>
<td>S.LL SDC 0</td>
<td>INTERRUPT ON LEVEL IF LPSI SET</td>
</tr>
<tr>
<td>S.LN SDC 0</td>
<td>DISABLE INTERRUPT ON LEVEL</td>
</tr>
<tr>
<td>L.T SDC 0</td>
<td>SEND TEXT LIST TO TP BUFFER</td>
</tr>
<tr>
<td>L.D SDC 0</td>
<td>GENERATE TEXT LEAF (F)</td>
</tr>
<tr>
<td>L.L SDC 0</td>
<td>DESTROY TEXT LEAF</td>
</tr>
<tr>
<td>PDP1 SDS 204</td>
<td></td>
</tr>
<tr>
<td>PDP2 SDS 204</td>
<td></td>
</tr>
</tbody>
</table>
$TITLE DISPLAY CHARACTER GENERATOR

D00  INCR
SDC 1272
SDC 6251
SDC 6057
SDC 7516
SDC 1578
SDC 5172
SDC 3726
SDC 0
POP

D01  INCR
SDC 5160
SDC 1472
SDC 7255
SDC 3737
SDC 0
POP

D02  INCR
SDC 5271
SDC 5152
SDC 5364
SDC 5537
SDC 2774
SDC 5417
SDC 3020
SDC 0
POP

D03  INCR
SDC 1252
SDC 5760
SDC 5152
SDC 5354
SDC 1051
SDC 5253
SDC 6455
SDC 3737
SDC 1000
POP
D04    INCR
      SDC 1110
      SDC 5072
      SDC 7275
      SDC 6810
      SDC 5037
      SDC 1600
      POP

D05    INCR
      SDC 1252
      SDC 5760
      SDC 5162
      SDC 5374
      SDC 6270
      SDC 5037
      SDC 3616
      SDC 0
      POP

D06    INCR
      SDC 1252
      SDC 5760
      SDC 5152
      SDC 5364
      SDC 5572
      SDC 5170
      SDC 3736
      SDC 1600
      POP

D07    INCR
      SDC 5271
      SDC 5162
      SDC 7454
      SDC 5637
      SDC 3710
      SDC 0
      POP

D10    INCR
      SDC 1252
      SDC 5760
      SDC 5152
A-9

$DC 5364
$DC 5512
$DC 6251
$DC 6057
$DC 5637
$DC 2600
POP

D11  INCR
$DC 5270
$DC 5162
$DC 7453
$DC 5251
$DC 6057
$DC 5637
$DC 2600
POP

D12  INCR
$DC 7272
$DC 5160
$DC 5766
$DC 7632
$DC 7437
$DC 1720
$DC 0
POP

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

D14  INCR
$DC 1272
$DC 6251
$DC 6057
A-10

SDC 3656
SDC 5564
SDC 3020
SDC 1700
POP

D15 INCR
SDC 7272
SDC 6540
SDC 6766
SDC 5554
SDC 3020
SDC 1700

POP

D16 INCR
SDC 7272
SDC 5270
SDC 5025
SDC 5550
SDC 2516
SDC 3050
SDC 1720
SDC 0
POP

D17 INCR
SDC 7272
SDC 5270
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
D21  POP
    INCR
    SDC 7272
    SDC 5236
    SDC 7050
    SDC 7236
    SDC 7620
    SDC 1700
    POP

D22  INCR
    SDC 5160
    SDC 1472
    SDC 7254
    SDC 1050
    SDC 3736
    SDC 1700
    POP

D23  INCR
    SDC 2252
    SDC 5657
    SDC 6051
    SDC 7242
    SDC 3636
    SDC 1720
    SDC 0
    POP

D24  INCR
    SDC 7272
    SDC 5230
    SDC 5075
    SDC 7720
    SDC 1700
    POP

D25  INCR
    SDC 7272
    SDC 5236
    SDC 3670
    SDC 5020
    SDC 1700
    POP
D26  INCR
SDC 7272
SDC 5277
SDC 6176
SDC 7620
SDC 1700
POP
D27  INCR
SDC 7272
SDC 5277
SDC 3250
SDC 7676
SDC 1720
SDC 0
POP
D30  INCR
SDC 1272
SDC 6251
SDC 6057
SDC 7656
SDC 5564
SDC 3820
SDC 1700
POP
D31  INCR
SDC 7272
SDC 5270
SDC 5756
SDC 5564
SDC 3720
SDC 1700
POP
D32  INCR
SDC 1272
SDC 6251
SDC 6057
SDC 7656
SDC 5564
SDC 1022
SDC 7720
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
SDC 5076
SDC 5665
SDC 5327
SDC 3010
SDC 0
POP
D40 INCR
SDC 7272
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

SDC  5253
SDC  3727
SDC  2600
POP

D52  INCR
SDC  5172
SDC  7260
SDC  3636
SDC  5450
SDC  1730
SDC  0
POP

D53  INCR
SDC  5160
SDC  7272
SDC  6437
SDC  3716
SDC  0
POP

D54  INCR
SDC  3051
SDC  7371
SDC  3736
SDC  1600
POP

D55  INCR
SDC  3150
SDC  7454
SDC  1252
SDC  7050
SDC  3726
SDC  0
POP

D56  INCR
SDC  5271
SDC  7337
SDC  3717
SDC  0
POP

D57  INCR
$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 7210
$DC 5066
$DC 3717
$DC 1600
POP
D71 INCR
$DC 1252
$DC 5760
$DC 5152
$DC 5364
$DC 5352
$DC 5160
$DC 5714
$DC 5456
$DC 1666
$DC 2730
$DC 0
POP

D72  INCR
$DC 5172
$DC 7210
$DC 5076
$DC 7612
$DC 5314
$DC 5412
$DC 5210
$DC 5010
$DC 5016
$DC 5637
$DC 0
POP

D73  INCR
$DC 5153
$DC 5261
$DC 5355
$DC 1777
$DC 1454
$DC 1151
$DC 3700
POP

D75  VEC
$DC 2020
$DC 4000
POP

D76  SVEC
$DC 50
POP
<table>
<thead>
<tr>
<th>STITLE</th>
<th>TRACKING PATTERN GENERATOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>XP</td>
<td>LAW 3000</td>
</tr>
<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></td>
<td>SVEC</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></td>
<td>SVEC</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></td>
<td>SVEC</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></td>
<td>SVEC</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>
</tbody>
</table>
SDC 0
SDC 4
SDC 4
SDC 4000
SDC 2060
SDC 6060
SDC 0
SDC 4000
SDC 60
SDC 4060
SDC 0
SDC 4
SDC 4
SDC 4000
SDC 2070
SDC 6070
SDC 0
SDC 4000
SDC 70
SDC 4070
SDC 0
SDC 4
SDC 4
SDC 4000
SDC 2100
SDC 6100
SDC 0
SDC 4000
SDC 100
SDC 4100
SDC 4000
SDC 1400
SDC X5
POP
$TITLE

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
    SKP  TEST NEXT 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 IKB  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
    IOT 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 IIS</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 LAC 2</td>
<td>GET SC CONTENTS</td>
</tr>
<tr>
<td>XOR =77</td>
<td>COMPLEMENT SHIFT COUNT</td>
</tr>
<tr>
<td>TAD =640402</td>
<td>FORM NORM INSTRUCTION</td>
</tr>
<tr>
<td>AND =640477</td>
<td>TRUNCATE CARRY</td>
</tr>
<tr>
<td>DAC +=1</td>
<td>STORE NORM INSTRUCTION</td>
</tr>
<tr>
<td>HLT</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 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        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 785       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.B6     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
SAVE ADDRESS OF DIAGNOSTIC
ENTER EXTEND MODE
READ 201 STATUS
GET RECEIVE STATE BIT
SKIP IF NOT RECEIVING
WAIT FOR END OF RECORD
CLEAR 201 INTERFACE
GET TERM RDY BIT & FRAME SIZE 8
SET INITIAL 201 INTERFACE STATE
GET PUNCH STATUS SWITCH
SKIP IF PUNCH ACTIVE
PUNCH NOT ACTIVE
SKIP ON PUNCH FLAG
WAIT FOR PUNCH FLAG
GET TELEPRINTER STATUS SWITCH
SKIP IF TELEPRINTER ACTIVE
TELEPRINTER NOT ACTIVE
SKIP ON TELEPRINTER FLAG
WAIT FOR TELEPRINTER FLAG
READ DISPLAY STATUS
GET DISPLAY FLAG BITS
SKIP IF DISPLAY STOPPED
WAIT FOR DISPLAY TO STOP
CLEAR PUSH BUTTONS
DISABLE CLOCK
CLEAR ALL FLAGS
INDICATE PUNCH IDLE
INDICATE TELEPRINTER IDLE
DISABLE OPERATION OF PUSH BUTTONS
DISABLE DISPLAY INTERRUPTS
CLEAR TRANSLATION VALUE
UNLOCK N.A
UNLOCK N.C
UNLOCK S.TRD
UNLOCK S.TRR
UNLOCK S.LY
UNLOCK S.LX
GET TELEPRINTER MASK
DAC STATUS  ALLOCATE TELEPRINTER ONLY
DAC BFTTY2  SET BFTTY ALLOCATION MASK
LAC TQ  GET POINTER TO END OF TASK QUEUE
DAC TQ+1  RESET INPUT POINTER
DAC TQ+2  RESET OUTPUT POINTER
DAC BRQ  SET RECORD SEEK SWITCH
DZM BRO  INDICATE NEW RECORD NEEDED
LAC BPQ  GET POINTER TO END OF PUNCH BUFFER
DAC BPQ+1  RESET INPUT POINTER
DAC BPQ+2  RESET OUTPUT POINTER
LAC BKQ  GET POINTER TO END OF KB BUFFER
DAC BKQ+1  RESET INPUT POINTER
DAC BKQ+2  RESET OUTPUT POINTER
LAC BTQ  GET POINTER TO END OF TP BUFFER
DAC BTQ+1  RESET INPUT POINTER
DAC BTQ+2  RESET OUTPUT POINTER
LAC =DN  GET ADDRESS OF NULL DISPLAY SERVICE
DAC DPT  SET NULL LIGHT PEN SERVICE
LAC =PN  GET ADDRESS OF NULL PB SERVICE
DAC PTT  SET NULL PUSH BUTTON SERVICE
LAW 3000  GET POP INSTRUCTION
DAC XP  INHIBIT TRACKING PROCESS
LAC =D  GET ADDRESS OF HIGHEST ACTIVE LEVEL
DAC DHAL+7  REMOVE EVERYTHING FROM HAL
LAW PDPI  GET ADDRESS OF PUSH DOWN LIST
IOT 645  SET PUSH DOWN POINTER
LAW 7763  GET INITIAL DISPLAY CONDITIONS
IOT 665  SET INITIAL DISPLAY CONDITIONS
LAW 4400  GET BREAK FIELD 1 PARAMETER
IOT 765  LOAD BREAK FIELD
LAW D  GET ADDRESS OF SYSTEM DISPLAY FILE
IOT 1605  START DISPLAY
LAC =BFENG  GET ENQUIRY CHARACTER
JMS BFENGS  INITIALIZE 201 TASKS
IOT 42  ENABLE INTERRUPTS
LAW BFENG  GET ENQUIRY CHARACTER
JMS B*FO  SEND ATTENTION INTERRUPT
NOP  DATA SET NOT CONNECTED

E1 DZM 26  CLEAR POINTER TO DIAGNOSTIC LEVEL
DZM 27  CLEAR POINTER TO DIAGNOSTIC LEAF
JMS S.TL CREATE TITLE LEAF
JMP E2 USE TELETYPWR ONLY
DAC E3 SAVE POINTER TO TITLE LEAF
LAC =EF GET ADDRESS OF TEXT LIST
JMS L.D CREATE TITLE LEAF
JMP E2 USE TELETYPWR ONLY
LMQ
LAC E3 GET POINTER TO TITLE LEVEL
JMS S.TI INSERT TITLE LEAF
JMP E2 USE TELETYPWR ONLY
LAC =370 GET Y TITLE COORDINATE
LMQ
LAC E3 GET POINTER TO TITLE LEVEL
JMS S.LY SET Y TITLE COORDINATE
LAW -144 GET X TITLE COORDINATE
LMQ
LAC E3 GET POINTER TO TITLE LEVEL
JMS S.LX SET X TITLE COORDINATE
LAW 500 GET SCALE X2 PARAMETER
LMQ
LAC E3 GET POINTER TO TITLE LEVEL
JMS S.LP SET TITLE SCALE
LAC E3 GET POINTER TO TITLE LEVEL
LMQ
LAC =DHAL GET ADDRESS OF HIGHEST ACTIVE LEVEL
JMS S.TI INSERT TITLE LEVEL
JMP E2 USE TELETYPWR ONLY
JMS S.TL CREATE DIAGNOSTIC LEVEL
JMP E2 USE TELETYPWR ONLY
DAC 26 SET POINTER TO DIAGNOSTIC LEVEL
LAC =200 GET Y DIAGNOSTIC DISPLACEMENT
LMQ
LAC 26 GET ADDRESS OF DIAGNOSTIC LEVEL
JMS S.LY TRANSLATE LEVEL IN Y DIRECTION
LAW -400 GET X DIAGNOSTIC DISPLACEMENT
LMQ
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
A-30

LAC 26
JMS S.LP
LAC 26
LMQ
LAC =DHAL
JMS S.TI
JMP E2
LAC 25
SZA
JMS L.D
JMP E2
DAC 27
LMQ
LAC 26
JMS S.TI
NOP
LAC 25
SNA
JMP IDLE
LAC =747575
JMS B.T
LAC 25
JMS L.T
LAC =747575
JMS B.T
JMP IDLE

E2

GET ADDRESS OF DIAGNOSTIC LEVEL
SET DIAGNOSTIC SCALE
GET ADDRESS OF DIAGNOSTIC LEVEL
SET UP PARAMETER
GET ADDRESS OF HIGHEST ACTIVE LEVEL
INSERT DIAGNOSTIC LEVEL
DISPLAY STORAGE EXCEEDED
GET ADDRESS OF TEXT LIST
SKIP IF DISP STORAGE BEING CLEARED
CREATE DIAGNOSTIC LEAF
USE TELETYPE ONLY
SET POINTER TO DIAGNOSTIC LEAF
SET UP PARAMETER
GET ADDRESS OF DIAGNOSTIC LEVEL
INSERT DIAGNOSTIC LEAF
USE TELETYPE ONLY
GET POINTER TO TEXT LIST
SKIP IF COMMENT TO BE TYPED
BEGIN IDLE-TIME TASK
GET TELEPRINTER POSITIONING CODE
POSITION TELEPRINTER
GET ADDRESS OF TEXT LIST
TYPE DIAGNOSTIC
GET TELEPRINTER POSITIONING CODE
POSITION TELEPRINTER
BEGIN IDLE-TIME TASK

EF

$DC 11
$TEXT "SEL EXECUTIVE SYSTEM (01)"
<table>
<thead>
<tr>
<th>TITLE</th>
<th>DISPLAY STRUCTURE STORAGE MANAGER</th>
</tr>
</thead>
<tbody>
<tr>
<td>STORE</td>
<td>SEQU 12000</td>
</tr>
<tr>
<td>B</td>
<td>SDC 0</td>
</tr>
<tr>
<td></td>
<td>CMA</td>
</tr>
<tr>
<td></td>
<td>TAD = 1</td>
</tr>
<tr>
<td></td>
<td>DAC T1</td>
</tr>
<tr>
<td></td>
<td>DAC T2</td>
</tr>
<tr>
<td></td>
<td>LAC = STORE</td>
</tr>
<tr>
<td>B1</td>
<td>DAC T3</td>
</tr>
<tr>
<td></td>
<td>DAC T4</td>
</tr>
<tr>
<td></td>
<td>SAD = 20000</td>
</tr>
<tr>
<td></td>
<td>JMP* B</td>
</tr>
<tr>
<td></td>
<td>LAC* T4</td>
</tr>
<tr>
<td></td>
<td>SNA</td>
</tr>
<tr>
<td></td>
<td>JMP B2</td>
</tr>
<tr>
<td></td>
<td>LAC T2</td>
</tr>
<tr>
<td></td>
<td>DAC T1</td>
</tr>
<tr>
<td></td>
<td>LAC T4</td>
</tr>
<tr>
<td></td>
<td>TAD = 4</td>
</tr>
<tr>
<td></td>
<td>JMP B1</td>
</tr>
<tr>
<td>B2</td>
<td>ISZ T1</td>
</tr>
<tr>
<td></td>
<td>JMP ++4</td>
</tr>
<tr>
<td></td>
<td>LAC T3</td>
</tr>
<tr>
<td></td>
<td>ISZ B</td>
</tr>
<tr>
<td></td>
<td>JMP* B</td>
</tr>
<tr>
<td></td>
<td>LAC T4</td>
</tr>
<tr>
<td></td>
<td>TAD = 4</td>
</tr>
<tr>
<td></td>
<td>JMP B1 +1</td>
</tr>
<tr>
<td>B3</td>
<td>SDC 0</td>
</tr>
<tr>
<td></td>
<td>LAC = 1</td>
</tr>
<tr>
<td></td>
<td>JMS B</td>
</tr>
<tr>
<td></td>
<td>JMP* B3</td>
</tr>
<tr>
<td></td>
<td>ISZ B3</td>
</tr>
<tr>
<td></td>
<td>JMP* B3</td>
</tr>
<tr>
<td>B4</td>
<td>SDC 0</td>
</tr>
<tr>
<td></td>
<td>LAC = 2</td>
</tr>
<tr>
<td></td>
<td>GET SINGLE BLOCK PARAMETER</td>
</tr>
<tr>
<td></td>
<td>FIND SINGLE BLOCK</td>
</tr>
<tr>
<td></td>
<td>NO SINGLE BLOCK AVAILABLE</td>
</tr>
<tr>
<td></td>
<td>INDICATE SUCCESS</td>
</tr>
<tr>
<td></td>
<td>RETURN</td>
</tr>
<tr>
<td></td>
<td>GET DOUBLE BLOCK PARAMETER</td>
</tr>
</tbody>
</table>

**LOWER LIMIT OF DISPLAY STORAGE**

- STORE VALUE FOR RESETTING COUNTER
- SET POINTER TO CANDIDATE
- SET POINTER TO NEW BLOCK
- SKIP IF STORAGE NOT EXCEEDED
- NOT ENOUGH FREE STORAGE
- GET FIRST WORD FROM BLOCK
- SKIP IF BLOCK NOT AVAILABLE
- ADD BLOCK TO CANDIDATE
- GET INITIAL VALUE OF COUNTER
- 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
- GET ADDRESS OF BLOCK JUST ADDED
- FORM ADDRESS OF NEXT BLOCK
- ADD ANOTHER BLOCK
- GET SINGLE BLOCK PARAMETER
- FIND SINGLE BLOCK
- NO SINGLE BLOCK AVAILABLE
- INDICATE SUCCESS
- RETURN
- GET DOUBLE BLOCK PARAMETER
JMS B  FIND DOUBLE BLOCK
JMP* B4  NO DOUBLE BLOCK AVAILABLE
ISZ B4  INDICATE SUCCESS
JMP* B4  RETURN
**STITLE**  

**WORD QUEUE MANAGER**

| QC  | IOT 2 | DISABLE INTERRUPTS  
|     | JMS QS | SET CONTROL POINTERS  
|     | LAC* QP | GET POINTER TO END OF QUEUE  
|     | DAC* QIP | SET INPUT POINTER  
|     | DAC* QOP | SET OUTPUT POINTER  
|     | IOT 42 | ENABLE INTERRUPTS  
|     | JMP* Q.C | RETURN  

| QA  | IOT 2 | DISABLE INTERRUPTS  
|     | JMS QA1 | ADD WORD TO QUEUE  
|     | SKP | OVERFLOW  
|     | ISZ Q.A | INDICATE SUCCESS  
|     | IOT 42 | ENABLE INTERRUPTS  
|     | JMP* Q.A | RETURN  

| QI  | IOT 2 | DISABLE INTERRUPTS  
|     | JMS QS | SET CONTROL POINTERS  
|     | LAC* QOP | GET OUTPUT POINTER  
|     | DAC 23 | SAVE OUTPUT POINTER  
|     | TAD =-3 | SUBTRACT 3  
|     | SAD QP | SKIP IF NO WRAP-AROUND  
|     | LAC* QP | GET POINTER TO END OF QUEUE  
|     | SAD* QP | SKIP IF NO WRAP-AROUND  
|     | SKP | CHECK FOR OVERFLOW  
|     | TAD =2 | FORM NEW OUTPUT POINTER  
|     | SAD* QIP | SKIP IF NO OVERFLOW  
|     | JMP *+5 | OVERFLOW  
|     | DAC* QOP | SET NEW OUTPUT POINTER  
|     | LACQ | GET VALUE TO BE STORED  
|     | DAC* 23 | STORE VALUE IN QUEUE  
|     | ISZ Q.1 | INDICATE SUCCESS  
|     | IOT 42 | ENABLE INTERRUPTS  
|     | JMP* Q.1 | RETURN  

| QF  | IOT 2 | DISABLE INTERRUPTS  
|     | JMS QF1 | FETCH WORD FROM QUEUE  
|     | SKP | QUEUE EMPTY  
|     | ISZ Q.F | 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 QOP
TAD =1

SKIP IF NOT END OF QUEUE
WRAP AROUND TO BEGINNING OF QUEUE
INCREMENT
| TS | IOT 2 | DISABLE INTERRUPTS |
|    | AND =77777 | TRUNCATE HIGH ORDER BITS |
|    | JMS TII | PUT TASK ADDRESS ON QUEUE |
|    | IOT 42 | ENABLE INTERRUPTS |
|    | JMP* T.S | RETURN |
| TP | LAW 17776 | LOAD AC WITH -2 |
|    | TAD T.P | FORM ADDRESS OF NEW TASK |
|    | JMS T.S | SCHEDULE NEW TASK |
| TF | IOT 2 | DISABLE INTERRUPTS |
|    | JMS TIO | READ WORD FROM TASK QUEUE |
|    | DAC 23 | SAVE TASK ADDRESS |
|    | RAL | SHIFT TYPE BITS INTO LINK & SIGN |
|    | S2L | SKIP IF NOT REENTRY DELAY |
|    | JMP TF1 | RESTORE MQ & AC AND EXECUTE |
|    | SPA | SKIP IF NOT ALLOCATION DELAY |
|    | JMP TF2 | CHECK ELIGIBILITY |
|    | IOT 42 | ENABLE INTERRUPTS |
|    | JMP* 23 | EXECUTE TASK |
| TF1 | JMS TIO | READ WORD FROM TASK QUEUE |
|    | LMQ | RESTORE MQ |
|    | JMS TIO | READ WORD FROM TASK QUEUE |
|    | JMP TF1-2 | EXECUTE TASK |
| TF2 | JMS TIO | READ WORD FROM TASK QUEUE |
|    | AND STATUS | FORM ELIGIBILITY CHECK |
|    | SNA | SKIP IF TASK NOT ELIGIBLE |
|    | JMP TF3 | MODIFY STATUS & EXECUTE |
|    | LAC 23 | GET ADDRESS OF TASK |
|    | JMS TII | PUT BACK ON TASK QUEUE |
|    | LAC T9+2 | GET ALLOCATION MASK |
|    | JMS TII | PUT BACK ON TASK QUEUE |
|    | IOT 42 | ENABLE INTERRUPTS |
|    | JMP TF | GET ANOTHER TASK |
| TF3 | LAC T9+2 | GET ALLOCATION MASK |
|    | XOR STATUS | OR WITH STATUS WORD |
|    | DAC STATUS | STORE NEW STATUS WORD |
|    | JMP TF1-2 | EXECUTE TASK |
TA
AND =17777  TRUNCATE HIGH ORDER BITS
IOT 2  DISABLE INTERRUPTS
DAC 23  SAVE ALLOCATION MASK
LAC  T.A  GET ADDRESS OF RETURN
AND =77777  TRUNCATE HIGH ORDER BITS
XOR =200000  INDICATE ALLOCATION DELAY
JMS TII  PUT TASK ADDRESS ON QUEUE
LAC 23  GET ALLOCATION MASK
JMS TII  PUT ALLOCATION MASK ON QUEUE
JMP TF+1  GET ANOTHER TASK

TR
CMA  COMPLEMENT RELEASE MASK
AND STATUS  MODIFY ALLOCATION STATUS
DAC STATUS  STORE NEW ALLOCATION STATUS
JMP* T.R  RETURN

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

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

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

TIO
DAC T0+2
LAC TQ+2
SAD TQ+1
JMP EE
JMS TI
DAC TQ+2
LAC TQ+2
GET OUTPUT POINTER
SKIP IF TASK QUEUE NOT EMPTY
TASK QUEUE EMPTY
INCREMENT
STORE NEW OUTPUT POINTER
GET WORD FROM TASK QUEUE
RETURN

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

TI
DAC T0
LAC T0+1
JMP T0
GET VALUE TO BE STORED
PUT IN TASK QUEUE
RETURN

TIL
DAC T2
LAC T0+1
JMP TII
SAVE INPUT POINTER
INCREMENT
STORE NEW INPUT POINTER
GET VALUE TO BE STORED
PUT IN TASK QUEUE
RETURN
SAD TQ
LAC = TQ + 2
TAD = 1
JMP* TI

TQ  $DC *+200
    $DS 200

SKIP IF NO WRAP-AROUND
GET ADDRESS BEFORE FIRST DATA WORD
INCREMENT POINTER
RETURN
### STITLE

**FORMAT CONVERTER**

<table>
<thead>
<tr>
<th>CB6</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLL</td>
<td>USE ZERO'S TO FILL HOLES</td>
</tr>
<tr>
<td>LRS 14</td>
<td>SHIFT DIGITS 2, 3, 4, &amp; 5 INTO MQ</td>
</tr>
<tr>
<td>ALS 3</td>
<td>CONVERT DIGIT 2</td>
</tr>
<tr>
<td>LRS 6</td>
<td>SHIFT DIGIT 2 INTO MQ</td>
</tr>
<tr>
<td>ALS 3</td>
<td>CONVERT DIGIT 1</td>
</tr>
<tr>
<td>LLS 11</td>
<td>SHIFT DIGITS 0, 1, &amp; 2 INTO AC</td>
</tr>
<tr>
<td>DAC T1</td>
<td>STORE HIGH ORDER DIGITS</td>
</tr>
<tr>
<td>LLS 6</td>
<td>SHIFT DIGITS 3 &amp; 4 INTO AC</td>
</tr>
<tr>
<td>ALS 3</td>
<td>CONVERT DIGIT 5</td>
</tr>
<tr>
<td>LRS 6</td>
<td>SHIFT DIGIT 4 INTO MQ</td>
</tr>
<tr>
<td>ALS 3</td>
<td>CONVERT DIGIT 4</td>
</tr>
<tr>
<td>AND #77</td>
<td>CONVERT DIGIT 3</td>
</tr>
<tr>
<td>LRS 11</td>
<td>SHIFT LOW ORDER DIGITS INTO MQ</td>
</tr>
<tr>
<td>LAC* T1</td>
<td>GET HIGH ORDER DIGITS</td>
</tr>
<tr>
<td>JMP* C.B6</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C6A</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND #77</td>
<td>TRUNCATE HIGH ORDER BITS</td>
</tr>
<tr>
<td>TAD =C6A1</td>
<td>ADD ADDRESS OF TABLE</td>
</tr>
<tr>
<td>DAC T1</td>
<td>SAVE TEMPORARILY</td>
</tr>
<tr>
<td>LAC* T1</td>
<td>GET CONVERTED VALUE</td>
</tr>
<tr>
<td>JMP* C.6A</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C6A1</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<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>
A-42

SDC 242
SDC 244
SDC 243
SDC 246
SDC 215
SDC 212
SDC 240
SDC 377

CA6 AND =177 TRUNCATE HIGH ORDER BITS
TAD =CA61 ADD ADDRESS OF TABLE
DAC T1 SAVE TEMPORARILY
LAC* T1 GET CONVERTED VALUE
JMP* C.A6 RETURN

CA61 SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 74
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 77
SDC 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 FORM 1'S COMPLEMENT
TAD =2001 GET MAGNITUDE
AND =3777 SET SIGN BIT & FORM 2'S COMPLEMENT
JMP* C.BC CLEAR ESCAPE/INTENSITY BIT
CBC1 AND =1777 RETURN
JMP* C.BC CONVERT TO MODULO 2^10
RETURN
$TITLE 201 DATAPHONE BUFFER MANAGER

BFDLE $SEQU 220 DATA LINK ESCAPE
BFSYN $SEQU 26 SYNCHRONOUS IDLE
BFACK $SEQU 6 POSITIVE ACKNOWLEDGEMENT
BFNAK $SEQU 225 NEGATIVE ACKNOWLEDGEMENT
BFEOF $SEQU 204 END OF TRANSMISSION
BFENQ $SEQU 5 ENQUIRY
BFETB $SEQU 27 END OF TEXT BLOCK
BFETX $SEQU 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 JMP BFI1 DATA SET CONNECTED
JMP* BFI
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 BFI1 GET FIRST RECEIVED CHARACTER
SZA 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
<table>
<thead>
<tr>
<th>BF11</th>
<th>ISZ B.FI</th>
<th>INDICATE SUCCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JMP B.FI</td>
<td>RETURN</td>
</tr>
<tr>
<td></td>
<td>JMP BFI</td>
<td>GET CHARACTER FROM INPUT BUFFER</td>
</tr>
<tr>
<td>BF12</td>
<td>JMS T.P</td>
<td>SCHEDULE PREVIOUS LOC &amp; TERMINATE</td>
</tr>
</tbody>
</table>

| BF0  | DAC BFO3 | SAVE CHARACTER TO BE BUFFERED |
| IOT 1412 | AND =1000 | READ 201 STATUS |
| SNA  | GET SET READY BIT |
|      | JMP B+F0  | DATA SET NOT CONNECTED |
| LAC BFS | 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 BFOI | ISZ BFOI | PUT CHARACTER IN OUTPUT BUFFER |
| SMA  | INCREMENT INPUT POINTER |
|      | JMP BFO1 | SKIP IF END-OF-RECORD CHARACTER |
| LAC BFS | RETURN |
| XOR #20 | GET 201 TASK STATE |
| DAC BFS | SET NEW 201 TASK STATE |
| LAC =BFXMT | GET ADDRESS OF TRANSMISSION TASK |
| JMS T+5 | SCHEDULE TRANSMISSION TASK |

| BF01 | ISZ B+F0 | INDICATE SUCCESS |
|      | JMP B+F0 | RETURN |
|      | JMP BFO+1 | PUT CHARACTER IN BUFFER |
| BF02 | JMS T+P | SCHEDULE PREVIOUS LOC & TERMINATE |

| BFXMT | IOT 1412 | READ 201 STATUS |
| IOT 1412 | 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 |
| SZA+RAR | SKIP IF ACK NOT EXPECTED |
|      | JMP BFXMT4 | RESCHEDULE BFXMT |
| SZA+RAR | SKIP IF INPUT BUFFER EMPTY |
|      | JMP BFXMT4 | RESCHEDULE BFXMT |
| SNA+RAR | SKIP IF ACK OUTPUT PENDING |
A-49

JMP BFXMT1
LAC BFS
AND =33
DAC BFS
LAC =BFIB
DAC BFII
DAC BFIO
LAC =BFACKR
JMP BFXMT3

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

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

BFXMT3 DAC BF00
LAW -10
DAC BFC
LAW BFBSYN
DAC 5
LAC =200005
DAC IFO
LAC =20000
IOT 1404
JMS T.F
JMP BFXMT

BFXMT4 JMS T.P

BFACKR LAW BFACK
BFNAKR LAW BFNAK

BFTTY LAC BFTTY2
JMS T.A

ACK RECORD
NAK RECORD

GET 201 TASK STATE
CLEAR ACK OUTPUT PENDING BIT
SET NEW 201 TASK STATE
GET ADDRESS OF INPUT BUFFER
RESET INPUT POINTER
RESET OUTPUT POINTER
GET POINTER TO ACK RECORD
TRANSMIT ACK
SKIP IF NAK OUTPUT PENDING
CHECK FOR DATA OUTPUT PENDING
GET 201 TASK STATE
CLEAR NAK OUTPUT PENDING BIT
SET NEW 201 TASK STATE
GET POINTER TO NAK RECORD
TRANSMIT NAK
SKIP IF DATA OUTPUT PENDING
NO OUTPUT PENDING
GET 201 TASK STATE
CLEAR DATA BIT, SET ACK EXP BIT
SET NEW 201 TASK STATE
GET POINTER TO OUTPUT BUFFER
SET OUTPUT POINTER
LOAD AC WITH -8
SET SYN COUNT
GET SYN CHARACTER
SET TRANSMIT IMAGE
GET LAC 5 INSTRUCTION
INITIALIZE XMT INTERRUPT SERVICE
GET XMT REQ BIT MASK
SET XMT REQ BIT
TERMINATE
START TRANSMISSION, IF APPLICABLE
SCHEDULE PREVIOUS LOC & TERMINATE

GET CONDITIONAL TELEPRINTER MASK
ALLOCATE TELEPRINTER, IF NECESSARY
PREPARE FOR POSSIBLE ENQUIRY
GET CHARACTER FROM BUFFER
SKIP IF NOT END-OF-RECORD CHARACTER
TERMINATE LINE
CONVERT TO 6-BIT CODE
PRECEDE WITH NULL CHARACTERS
TYPE CHARACTER
INCREMENT INPUT POINTER
TYPE NEXT CHARACTER
GET CARRIAGE RETURN, LINE FEED CODE
TYPE CARRIAGE RETURN, LINE FEED
GET 201 TASK STATE
FORM ACK PENDING STATE
SET NEW 201 TASK STATE
GET ALLOCATION MASK
SET BFTTY ALLOCATION MASK
RELEASE TELEPRINTER
ACKNOWLEDGE RECORD
GET RECEIVED CHARACTER
SHIFT INTO POSITION
TRUNCATE HIGH ORDER BITS
STATE VARIABLE
SKIP IF NOT SYN
FIND NEXT SYN & CHANGE STATE
IGNORE CHARACTER
READ 201 STATUS
GET TEXT BIT
CLEAR TEXT BIT
GET JMP IFI1 INSTRUCTION
MODIFY INTERRUPT SERVICE
CLEAR 201 FLAGS
RETURN FROM INTERRUPT
SKIP IF NOT SYN
IGNORE SYN
SKIP IF NOT DLE (EVEN PARITY)
BUFFER RECEIVED RECORD
GET NOP INSTRUCTION
MODIFY INTERRUPT SERVICE
GET JMP IFI2 INSTRUCTION
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP IFI1-3</td>
<td>MODIFY INTERRUPT SERVICE</td>
</tr>
<tr>
<td>SAD = BFDE</td>
<td>SKIP IF NOT DLE (EVEN PARITY)</td>
</tr>
<tr>
<td>JMP IFI3-2</td>
<td>CHANGE STATE FOR NEXT CHARACTER</td>
</tr>
<tr>
<td>SAD = BFDE-200</td>
<td>SKIP IF NOT DLE (ODD PARITY)</td>
</tr>
<tr>
<td>JMP IFI3-2</td>
<td>CHANGE STATE FOR NEXT CHARACTER</td>
</tr>
<tr>
<td>JMS BFIS</td>
<td>PUT CHARACTER IN BUFFER</td>
</tr>
<tr>
<td>JMP IFI1-2</td>
<td>CLEAR FLAGS AND RETURN</td>
</tr>
<tr>
<td>LAC = 600000+IFI3</td>
<td>GET JMP IFI3 INSTRUCTION</td>
</tr>
<tr>
<td>JMP IFI1-3</td>
<td>MODIFY INTERRUPT SERVICE</td>
</tr>
<tr>
<td>SAD = BFDE</td>
<td>SKIP IF NOT DLE (EVEN PARITY)</td>
</tr>
<tr>
<td>JMP IFI31-3</td>
<td>PUT DLE IN BUFFER</td>
</tr>
<tr>
<td>SAD = BFDE-200</td>
<td>SKIP IF NOT DLE (ODD PARITY)</td>
</tr>
<tr>
<td>JMP IFI31-3</td>
<td>PUT DLE IN BUFFER</td>
</tr>
<tr>
<td>SAD = BFSYN</td>
<td>SKIP IF NOT SYN</td>
</tr>
<tr>
<td>JMP IFI31-2</td>
<td>IGNORE SYN</td>
</tr>
<tr>
<td>DAC BFEOR</td>
<td>SAVE END-OF-RECORD CHARACTER</td>
</tr>
<tr>
<td>SAD = BFACK</td>
<td>SKIP IF NOT ACK</td>
</tr>
<tr>
<td>JMP IFI31</td>
<td>CLEAR OUTPUT BUFFER</td>
</tr>
<tr>
<td>SAD = BFNK</td>
<td>SKIP IF NOT NAK</td>
</tr>
<tr>
<td>JMP IFI32</td>
<td>RETRANSMIT LAST DATA RECORD</td>
</tr>
<tr>
<td>XOR = 760000</td>
<td>INDICATE END-OF-RECORD CHARACTER</td>
</tr>
<tr>
<td>JMS BFIS</td>
<td>PUT CHARACTER IN BUFFER</td>
</tr>
<tr>
<td>LAC = 600000+IFI4</td>
<td>GET JMP IFI4 INSTRUCTION</td>
</tr>
<tr>
<td>JMP IFI1-3</td>
<td>MODIFY INTERRUPT SERVICE</td>
</tr>
<tr>
<td>JMS BFIS</td>
<td>PUT DLE CHARACTER IN BUFFER</td>
</tr>
<tr>
<td>LAC = 600000+IFI2</td>
<td>GET JMP IFI2 INSTRUCTION</td>
</tr>
<tr>
<td>JMP IFI1-3</td>
<td>MODIFY INTERRUPT SERVICE</td>
</tr>
<tr>
<td>LAC = BF0B</td>
<td>GET ADDRESS OF OUTPUT BUFFER</td>
</tr>
<tr>
<td>DAC BF01</td>
<td>RESET INPUT POINTER</td>
</tr>
<tr>
<td>LAC BFS</td>
<td>GET 201 TASK STATE</td>
</tr>
<tr>
<td>AND = 26</td>
<td>INDICATE ACK NOT EXPECTED</td>
</tr>
<tr>
<td>DAC BFS</td>
<td>STORE NEW TASK STATE</td>
</tr>
<tr>
<td>LAC = 740000</td>
<td>GET NOP INSTRUCTION</td>
</tr>
<tr>
<td>JMP IFI1-3</td>
<td>MODIFY INTERRUPT SERVICE</td>
</tr>
<tr>
<td>LAC BFS</td>
<td>GET 201 TASK STATE</td>
</tr>
<tr>
<td>XOR = 21</td>
<td>FORM STATE FOR RETRANSMISSION</td>
</tr>
<tr>
<td>DAC BFS</td>
<td>STORE NEW TASK STATE</td>
</tr>
<tr>
<td>LAC = BF0XMT</td>
<td>GET ADDRESS OF TRANSMISSION TASK</td>
</tr>
<tr>
<td>JMS TII</td>
<td>SCHEDULE TRANSMISSION</td>
</tr>
<tr>
<td>JMP IFI32-2</td>
<td>MODIFY INTERRUPT SERVICE</td>
</tr>
</tbody>
</table>
IF14 CLL
ALS 10
DAC BFCKR
LAC =600000+IF15
JMP IF11-3

IFI5 XOR BFCKR
DAC BFII
LAC BFS
XOR =10
DAC BFS
JMP IF14-3

IF151 LAC BFS
XOR =2
DAC BFS
LAC BFEO
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 IFO1
SAD =760000+BFDLE
JMP IFO3
SMA
JMP IFO4
JMP IFO5

IFO1 ISZ BFC
JMP IFO2+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
GET INITIAL DLE
GET DLE CHARACTER
GET TRANSMIT IMAGE
GET TRANSMIT IMAGE
CLEAR 201 FLAGS
RETURN FROM INTERRUPT
GET CHARACTER FROM BUFFER
UPDATE BLOCK CHECK
GET OUTPUT POINTER
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 IF06 INSTRUCTION
MODIFY INTERRUPT SERVICE
GET BLOCK CHECK
SHIFT HIGH ORDER PART INTO POSITION
TRANSMIT HIGH ORDER BLOCK CHECK
GET JMP INSTRUCTION
MODIFY INTERRUPT SERVICE
GET BLOCK CHECK
TRANSMIT LOW ORDER PART
GET JMP IF08 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

BFENQS  SDC 0
SAD =BFENQ
JMP **4
SAD =BFENT
SKP
JMP+ BFENQS
LAC =BF0B
DAC BF0I
LAC =BFIB
DAC BFII
DAC BFIO
DZM BFIB
DZM BFS
LAC =740000
DAC IFI+3
JMP+ BFENGS
SKIP IF NOT ENQUIRY
PROCESS ENQUIRY
SKIP IF NOT END-OF-TRANSMISSION
REGARD AS ENQUIRY
RETURN
GET ADDRESS OF OUTPUT BUFFER
RESET INPUT POINTER
GET ADDRESS OF INPUT BUFFER
RESET INPUT POINTER
RESET OUTPUT POINTER
DO NOT SCHEDULE BYPASS TASK
STOP 201 TASK ACTIVITY
GET NOP INSTRUCTION
MODIFY INTERRUPT SERVICE
RETURN

BFIS  SDC 0
DAC+ BFII
JMS BFCKS
LAC BFII
SAD =BFIB
DZM BFCK
ISZ BFII
JMP+ BFIS
PUT CHARACTER IN INPUT BUFFER
UPDATE BLOCK CHECK
GET INPUT POINTER
SKIP IF INPUT BUFFER NON-EMPTY
CLEAR BLOCK CHECK
INCREMENT INPUT POINTER
RETURN

BFCKS  SDC 0
DAC 23
LAW -10
DAC 24
LAC BFCK
GET FORMER BLOCK CHECK
SAVE CHARACTER
LOAD AC WITH -8
SET COUNTER
GET FORMER BLOCK CHECK
BFCKSI RCR
DAC BFCK
CLQ
LAC 23
LRS 1
DAC 23
LACQ
ROTATE LOW ORDER BIT INTO LINK
STORE NEW LOW ORDER 15 BITS
PREPARE TO GET LOW ORDER CHAR BIT
GET CHARACTER REMAINS
SHIFT LOW ORDER BIT INTO MQ
STORE CHARACTER REMAINS
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
READER BUFFER MANAGER

**BR**
- LAC BRO
- SNA
- JMP BR2
- SAD BRI
- JMP BR1
- LAC* BRO
- ISZ BRO
- SNA
- DZM BRO
- ISZ B.R
- JMP* B.R
- RETURN

**BR1**
- SAD =BRQ+200
- JMP BR2
- IOT 314
- AND #1000
- SNA
- JMP BR2-1
- DZM BRO
- RETURN

**BR2**
- LAC =BRQ
- DAC BRI
- DAC BRO
- IOT 104
- 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

- 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
BRS $DC 0
DZM* BRI
ISZ BRI
JMP IR
IRD1 IOT 102
JMP IR
IRD2 DAC* BRI
ISZ BRI
DZM BRS
LAC BRI
SAD =BRQ+200
JMP IR
IRD3 IOT 104
JMP IR
BRQ $DS 200

RECORD SEEK SWITCH
STORE END-OF-RECORD IMAGE
INCREMENT INPUT POINTER
RETURN FROM INTERRUPT
CLEAR READER FLAG
RETURN FROM INTERRUPT
STORE IN READER BUFFER
INCREMENT INPUT POINTER
CLEAR RECORD SEEK SWITCH
GET INPUT POINTER
SKIP IF NOT END OF BUFFER
RETURN FROM INTERRUPT
SELECT READER
RETURN FROM INTERRUPT
**STITLE**

**PUNCH BUFFER MANAGER**

**BP**
- **DAC BP4**
  - SAVE PUNCH IMAGE
- **JMS BP2**
  - START PUNCH, IF POSSIBLE
- **LAC BP4**
  - GET PUNCH IMAGE
- **LMQ**
  - SET UP PUNCH IMAGE AS PARAMETER
- **LAC *BP6**
  - GET ADDRESS OF PUNCH BUFFER
- **JMS G.A**
  - PUT PUNCH IMAGE IN BUFFER
- **JMP BP1**
  - PUNCH BUFFER FULL
- **JMS BP2**
  - START PUNCH, IF POSSIBLE
- **ISZ B.P**
  - INDICATE SUCCESS
- **JMP* B.P**
  - RETURN

**BP1**
- **IOT 314**
  - READ STATUS
- **AND = 400**
  - GET PUNCH OUT-OF-TAPE FLAG
- **SZA**
  - SKIP IF PUNCH CONTAINS TAPE
- **JMP* B.P**
  - PUNCH OUT OF TAPE
- **SKP**
  - PREPARE TO SCHEDULE NEXT LOCATION
- **JMP BP+2**
  - TRY AGAIN TO PUT IMAGE IN BUFFER
- **JMS T.P**
  - SCHEDULE PREVIOUS LOC & TERMINATE

**BP2**
- **$DC 0**
  - GET PUNCH STATUS SWITCH
- **LAC BP3**
  - SKIP IF PUNCH IS IDLE
- **SZA**
  - PUNCH IS ACTIVE
- **JMP* BP2**
  - GET ADDRESS OF PUNCH BUFFER
- **LAC *BP9**
  - FETCH IMAGE FROM BUFFER
- **JMS G.F**
  - PUNCH BUFFER EMPTY
- **IOT 204**
  - SELECT PUNCH
- **JMP BP+2**
  - SET PUNCH STATUS SWITCH
- **JMS T.P**
  - PUNCH STATUS SWITCH
- **JMP BP+2**
  - RETURN

**BP3**
- **$DC 0**
  - GET PUNCH STATUS SWITCH
- **JMP BP2**
  - PUNCH STATUS SWITCH
- **JMP BP+2**
  - RETURN

**IPC**
- **IOT 314**
  - READ STATUS
- **AND = 400**
  - GET PUNCH OUT-OF-TAPE FLAG
- **SZA**
  - SKIP IF PUNCH CONTAINS TAPE
- **JMP IPC1**
  - PUNCH OUT OF TAPE
- **LAC *BP9**
  - GET ADDRESS OF PUNCH BUFFER
- **JMS G.F1**
  - GET IMAGE FROM PUNCH BUFFER
- **JMP IPC1**
  - PUNCH BUFFER EMPTY
- **IOT 204**
  - SELECT PUNCH
A-59

IPC1
JMP IR
IOT 202
DZM BP3
JMP IR
RETURN FROM INTERRUPT
CLEAR PUNCH FLAG
INDICATE PUNCH IDLE
RETURN FROM INTERRUPT

BPO
SDC **100
SDS 100
STITLE

KEYBOARD BUFFER MANAGER

BK
LAC = BKG
JMS Q.F
JMP BK1
DAC BKF
JMS C+A6
JMP* B+K
JMP BK

JMP* B.K

JMP

JMS

T.P

SCHEDULE NEW ATTEMPT

BK1

IKB
IOT 312
LMO
LAC = BKG
JMS QA1
NOP
JMP IR

READ KEYBOARD BUFFER
SET UP PARAMETER
GET ADDRESS OF KEYBOARD BUFFER
PUT CHARACTER IN BUFFER
BUFFER FULL -- IGNORE CHARACTER
RETURN FROM INTERRUPT

SDC ++100
SDS 100
STitle

Teleprinter Buffer Manager

BT
DAC BT5
LAC BT5
LNR
LAC = BT9
JMP Q - A
JMP BT2
LAC BT1
SAZ
JMP - B - T
IOT 2
LAC B - T
DAC 0
JMS BT1
JMP ITP
JMP BT+1
BT3
SDC 77
BT4
SDC 77

ITP
LAC BT4
SAD = 77
SKP
JMP ITP3
LAC BT3
SAD = 77
SKP
JMP ITP4
LAC = BT9
JMS QF1
JMP ITP2
DAC BT3
LRS 6
DAC BT4
LRS 6

ITP1
AND = 77
SAD = 77

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
Disable Interrupts
Get Return Address
Store Interrupt Return
Set Teleprinter Status Switch
Teleprinter Status Switch
Fake A Teleprinter Interrupt
Try Again To Put Char In Buffer
Schedule New Attempt
Get Second Character
Skip If Not Null Character
Look At Third Character
Type Second Character
Get Third Character
Skip If Not Null Character
Type First Character
Type Third Character
Get Address Of Teleprinter Buffer
Get Packed Word From TP Buffer
Clear Flag & Return
Set Up Third Character
Shift Second Character Into Place
Set Up Second Character
Shift First Character Into Place
Truncate High Order Bits
Skip If Not Null Character
JMP ITP
TAD =C6A1
DAC 23
LAC* 23
IOT 406
JMP 1R
ITP2
IOT 402
D2M BT1
JMP 1R
ITP3
DAC 23
LAC* 77
DAC BT4
LAC 23
JMP ITP1
ITP4
DAC 23
LAC* 77
DAC BT3
LAC 23
JMP ITP1
BT0
SDC ++100
SDS 100

TYPE NEXT CHARACTER
ADD ADDRESS OF 6-BIT TO ASCII TABLE
SAVE TEMPORARILY
GET CONVERTED ASCII VALUE
SEND CHARACTER TO TELEPRINTER
RETURN FROM INTERRUPT
CLEAR TELEPRINTER FLAG
INDICATE TELEPRINTER IDLE
RETURN FROM INTERRUPT
SAVE TEMPORARILY
GET NULL CHARACTER
STORE AS SECOND CHARACTER
GET CHARACTER TO BE TYPED
TYPE SECOND CHARACTER
SAVE TEMPORARILY
GET NULL CHARACTER
STORE AS THIRD CHARACTER
GET CHARACTER TO BE TYPED
TYPE THIRD CHARACTER

TABLE
- SELECT D/A CONVERTER #3
- RETURN
- SELECT D/A CONVERTER #6
- RETURN
- SELECT D/A CONVERTER #1
- SCHEDULE A LATER CHECK
- CHECK ELAPSED TIME
- UNLOCK N.C
- WAIT A LITTLE LONGER
- SKIP IF TIME INTERVAL HAS ELAPSED
- GET CLOCK SWITCH
- CLEAR CLOCK SWITCH
- ENABLE CLOCK
- SET CLOCK INTERVAL
- PROTECT AGAINST REENTRY
- SCHEDULE A CONVERSION CHECK
- CHECK FOR CONVERSION COMPLETE
- UNLOCK N.A
- GET CONVERTED VALUE
- WAIT FOR CONVERSION TO BE COMPLETED
- SKIP IF CONVERSION COMPLETE
- GET CONVERSION SWITCH
- SELECT A/D CONVERTER
- SELECT A/D CONVERTER CHANNEL
- TRUNCATE HIGH ORDER BITS
- PROTECT AGAINST REENTRY
- NONBUFFERED I/O MANAGER

A-63
A-64

IAD 10T 1312
DAC NA3
JMS IAD1

IAD1 SDC 0
JMP IR

ICK JMS 0+1
SDC 0
1OT 4
JMP IR

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

SET CLOCK SWITCH
CLOCK SWITCH
CLEAR CLOCK FLAG
RETURN FROM INTERRUPT
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
A-66

LAC PTT        GET ADDRESS OF PUSH BUTTON SERVICE
AND #77777     TRUNCATE HIGH ORDER BITS
JMS TII        SCHEDULE PUSH BUTTON SERVICE
IOT 631        READ PUSH BUTTONS
DAC PRG        MODIFY PUSH BUTTON STATUS WORD
DZM PE+1       DISABLE PUSH BUTTONS
JMP IR         RETURN FROM INTERRUPT

IPBI LAC PRG   GET FORMER PUSH BUTTON STATUS
JMS PS1        SET PUSH BUTTONS
JMP IR         RETURN FROM INTERRUPT
$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
A-68

TAD = -PDP2 FORM VALIDITY CHECK
SPA SKIP IF PARAMETER VALID
JMP* D.O NOT ENOUGH OWNERS
LAC* T1 GET FIRST PUSH DOWN WORD
LLS 3 SHIFT BREAK FIELD INTO POSITION
AND =70000 REMOVE ALL BUT BREAK FIELD
ISZ T1 SET POINTER TO SECOND PD ENTRY
TAD* T1 COMBINE FIRST & SECOND ENTRIES
TAD =7777 FORM ADDRESS IN OWNER OF OWNER
DAC T1 SAVE TEMPORARILY
LAC* T1 GET ADDRESS OF DESIRED OWNER
ISZ D.O INDICATE SUCCESS
JMP* D.O RETURN

DN JMS D.E ENABLE DISPLAY INTERRUPTS
JMS T.F TERMINATE TASK

DW SDC 0 CLEAR DISPLAY READY SWITCH
DZM DWT GET DISPLAY READY SWITCH
LAC DWT SKIP IF SET
SNA JMP ++3 WAIT FOR DISPLAY TO FINISH FRAME
JMP* DW RETURN
JMP DW+2 CHECK DISPLAY READY SWITCH
JMS T.P SCHEDULE NEW SWITCH CHECK

DWT SDC 0 DISPLAY READY SWITCH
LAC DWV GET TRANSLATION VALUE
SNA SKIP IF TRANSLATION PENDING
JMP XIS1 RESUME DISPLAY & RETURN
DAC* DWMD STORE DISPLACEMENT
XOR =2000 INVERT SIGN BIT
DAC* DWTL STORE COUNTERDISPLACEMENT
DZM DWV INDICATE TRANSLATION PERFORMED
JMP XIS1 RESUME DISPLAY & RETURN

ILP LAC DSS GET DISPLAY STATUS WORD 1
AND =7 GET BREAK FIELD
SZA SKIP IF ZERO BREAK FIELD
JMP ++5 USER FILE INTERRUPT
IOT 611 READ DISPLAY ADDRESS
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAD =-XP</td>
<td>FORM ADDRESS CHECK</td>
</tr>
<tr>
<td>SMA</td>
<td>SKIP IF USER FILE INTERRUPT</td>
</tr>
<tr>
<td>JMP XLP</td>
<td>TRACKING INTERRUPT</td>
</tr>
<tr>
<td>LAC DE+1</td>
<td>GET DISPLAY INT ENABLE SWITCH</td>
</tr>
<tr>
<td>SZA</td>
<td>SKIP IF DISPLAY INTERRUPTS DISABLED</td>
</tr>
<tr>
<td>JMP ++3</td>
<td>GET STATUS FOR USER</td>
</tr>
<tr>
<td>IOT 724</td>
<td>RESUME DISPLAY</td>
</tr>
<tr>
<td>JMP IR</td>
<td>RETURN FROM INTERRUPT</td>
</tr>
<tr>
<td>LAC UPT</td>
<td>GET ADDRESS OF SERVICE TASK</td>
</tr>
<tr>
<td>JMS DS</td>
<td>SCHEDULE SERVICE &amp; READ STATUS</td>
</tr>
<tr>
<td>JMP *-4</td>
<td>RESUME DISPLAY &amp; RETURN</td>
</tr>
<tr>
<td>IIS</td>
<td></td>
</tr>
<tr>
<td>LAC DSS</td>
<td>GET DISPLAY STATUS WORD 1</td>
</tr>
<tr>
<td>AND =7</td>
<td>GET BREAK FIELD</td>
</tr>
<tr>
<td>SNA</td>
<td>SKIP IF USER FILE INTERRUPT</td>
</tr>
<tr>
<td>JMP X1S</td>
<td>TRACKING INTERRUPT</td>
</tr>
<tr>
<td>IOT 611</td>
<td>READ DISPLAY ADDRESS</td>
</tr>
<tr>
<td>XOR =10000</td>
<td>INTERPRET WITH BREAK FIELD 1</td>
</tr>
<tr>
<td>SAD =D+4</td>
<td>SKIP IF NOT DISPLAY SYNC INTERRUPT</td>
</tr>
<tr>
<td>JMS DWT</td>
<td>SET DISPLAY READY SWITCH</td>
</tr>
<tr>
<td>LAC DE+1</td>
<td>GET DISPLAY INT ENABLE SWITCH</td>
</tr>
<tr>
<td>SZA</td>
<td>SKIP IF DISPLAY INTERRUPTS DISABLED</td>
</tr>
<tr>
<td>JMP ++5</td>
<td>GET STATUS FOR USER</td>
</tr>
<tr>
<td>IOT 611</td>
<td>READ DISPLAY ADDRESS</td>
</tr>
<tr>
<td>TAD =1</td>
<td>FORM RESUME ADDRESS</td>
</tr>
<tr>
<td>IOT 1605</td>
<td>RESUME DISPLAY</td>
</tr>
<tr>
<td>JMP IR</td>
<td>RETURN FROM INTERRUPT</td>
</tr>
<tr>
<td>LAC DSS</td>
<td>GET DISPLAY STATUS WORD 1</td>
</tr>
<tr>
<td>LLS 14</td>
<td>SHIFT BREAK FIELD INTO POSITION</td>
</tr>
<tr>
<td>AND =70000</td>
<td>REMOVE ALL BUT BREAK FIELD</td>
</tr>
<tr>
<td>IOT 601</td>
<td>FORM DISPLAY ADDRESS</td>
</tr>
<tr>
<td>DAC 23</td>
<td>SAVE TEMPORARILY</td>
</tr>
<tr>
<td>LAC* 23</td>
<td>GET ADDRESS OF SERVICE TASK</td>
</tr>
<tr>
<td>JMS DS</td>
<td>SCHEDULE SERVICE &amp; READ STATUS</td>
</tr>
<tr>
<td>JMP *-13</td>
<td>RESUME DISPLAY &amp; RETURN</td>
</tr>
<tr>
<td>DS</td>
<td></td>
</tr>
<tr>
<td>SDC 0</td>
<td></td>
</tr>
<tr>
<td>AND =77777</td>
<td>TRUNCATE HIGH ORDER BITS</td>
</tr>
<tr>
<td>JMS TII</td>
<td>SCHEDULE SERVICE TASK</td>
</tr>
<tr>
<td>DZM DE+1</td>
<td>DISABLE DISPLAY INTERRUPTS</td>
</tr>
</tbody>
</table>
LAC DSS
DAC DSI
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
SAVE
READ STATUS WORD 2
SAVE
READ DISPLAY ADDRESS
SAVE
READ Y DISPLAY COORDINATE
SAVE
READ X DISPLAY COORDINATE
SAVE
READ PUSH DOWN POINTER
SAVE
GET ADDRESS OF PUSH DOWN LIST
SET AUTOINDEX REGISTER
GET ADDRESS OF PUSH DOWN SAVE AREA
SET AUTOINDEX REGISTER
GET WORD FROM PUSH DOWN LIST
STORE IN PUSH-DOWN SAVE AREA
GET SOURCE POINTER
SKIP IF NOT END OF LIST
RETURN
COPY NEXT WORD
STITLE  TRACKING CONTROLLER

XI  TAD = 1000
    AND = 1777
    DAC XPY
    LAC9
    TAD = 1000
    AND = 1777
    XOR = 4000
    DAC XPX
    DZM XP
    JMP* X*I

XR  DZM XP
    JMP* X*R

XT  LAW 3000
    DAC XP
    JMP* X*T

XS  LAW 3000
    SAD XP
    ISZ X*S
    JMP* X*S

XY  LAC XPY
    TAD = -1000
    JMP* X+Y

XX  LAC XPX
    AND = 1777
    TAD = -1000
    JMP* X+X

XLP  HLT
    HLT
    HLT
    JMP **3
    AND = 1777
    DAC XPY
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 XPX 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+XLP1 GET JMP XLP1 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
SNA
JMP X31
TAD XL
SAD XH
JMP X31
RCR
TAD = -2000
SMA
JMP X31+2
AND = 1777
XOR = 4000
DAC XPX
JMP X31+2
LAW 777
DAC XPS
DZM XL
DZM XH
LAC = 701612
DAC XLP
LAC = 40000+XL
DAC XLP+1
JMP XIS1

X4
LAC XH
SNA
JMP X41
TAD XL
SAD XH
JMP X41
RCR
TAD = -2000
SMA
JMP X41+2
AND = 1777
DAC XPY
JMP X41+2
LAC XLP+7
DAC XLP+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 10T 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 XISI

GET SMA INSTRUCTION
MODIFY INTERRUPT SERVICE
RESUME DISPLAY & RETURN

X5
LAW 3000
DAC XP
JMP XISI

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 64

JMP STL1

DAC* 12

TAD =-1

DAC 12

LAW 640

DAC 12

LAW 1400

DAC* 12

DAC* 12

DZM* 12

DZM* 12

LAW 1121

DAC* 12

DAC* 12

LAW 4000

DAC* 12

LAW 3000

DAC* 12

LAC 75

ISZ S.TL

JMP* S.TL

STRTLE

GET 8-WORD BLOCK

NOT ENOUGH STORAGE

SAVE ADDRESS FOR RETURN

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

GETVEC INSTRUCTION

STORE IN FOURTH LOCATION IN HEAD

STORE IN FIFTH LOCATION IN HEAD

STORE IN SIXTH LOCATION IN HEAD

GETJUMP1 INSTRUCTION

STORE IN SEVENTH LOCATION IN HEAD

GET 8-WORD BLOCK

NOT ENOUGH STORAGE

STORE ADDRESS OF TAIL IN HEAD

GET UNCONDITIONAL DISPLAY SKIP

SET AUTOINDEX REGISTER

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

GETVEC 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

GETADDRESS 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* 75
DAC* 12
LAC T1
DAC* 75
ISZ S.TI
JMP* S.TI

TRUNCATE HIGH ORDER BITS
GET COMPLETE ADDRESS
STORE IN SECOND LOC IN BLOCK
GET JUMP1 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

PROTECT AGAINST REENTRY

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
ISZ T2
LACQ
SAD* T2
JMP STR1
LAC T2
TAD = 2
JMP STR+4

SUBSTRUCTURE NOT IN LEVEL
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

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
A-78

SDC STR
STITLE

SLH
- LAC = DHAL
- JMP S.LH

SLY
- JMS TV
- $DC S.LY
- $DC 0
- SAD = DHAL
- JMP SLY1 + 3
- TAD = 4
- DAC DWHD
- TAD = 3
- JMS SLT
- TAD = 5
- DAC DWTL
- LACO
- JMS T.BC
- XOR #760000
- DAC DWV

SLY1
- LAC DWV
- SZA
- JMP **4
- JMS T.U
- $DC SLY
- JMP SLY1
- JMS T.P

SLX
- JMS TV
- $DC S.LX
- $DC 0
- SAD = DHAL
- JMP SLX1 + 3
- TAD = 5
- DAC DWHD
- TAD = 2
- JMS SLT
- TAD = 6
- DAC DWTL
- LACO

LEVEL MODIFICATION OPERATORS

GET ADDRESS OF HIGHEST ACTIVE LEVEL
RETURN

PROTECT AGAINST REENTRY

SKIP IF NOT HIGHEST ACTIVE LEVEL
RETURN

FORM POINTER TO Y COORD IN HEAD
SAVE

FORM POINTER TO END OF HEAD
GET ADDRESS OF TAIL

FORM POINTER TO Y COORD IN TAIL
SAVE

GET Y INCREMENT

CONVERT TO DISPLAY COORDINATE
INDICATE STORAGE OCCUPIED
SAVE TRANSLATION VALUE

GET TRANSLATION VALUE

SKIP IF TRANSLATION COMPLETE
RESCHEDULE COMPLETION CHECK
UNLOCK S.LY

CHECK FOR TRANSLATION COMPLETE
SCHEDULE COMPLETION CHECK

PROTECT AGAINST REENTRY

SKIP IF NOT HIGHEST ACTIVE LEVEL
RETURN

FORM POINTER TO X COORD IN HEAD
SAVE

FORM POINTER TO END OF HEAD
GET ADDRESS OF TAIL

FORM POINTER TO X COORD IN TAIL
SAVE

GET X INCREMENT
A-80

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

SLX1

SLP
SAD = DHAL
JMP S.LP
TAD = 2
DAC T1
LACQ
AND = 777
DAC T1
JMP S.LP

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

SLBD
SAD = DHAL
JMP S.LBD
TAD = 1
DAC T2
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 6220
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 SLSL

SKIP IF NOT HIGHEST ACTIVE LEVEL
RETURN
REMOVE INTERRUPT FROM END OF LEVEL
GET LPSI CLEAR INSTRUCTION
STORE IN HEAD
GET SKIP-ON-NO-LPSI INSTRUCTION
STORE IN TAIL
RETURN

SLN
SAD =DHAL
JMP* S.LLN
JMS S.LSP
LAW 0
DAC* T1
JMP* S.LLN
RETURN

SLT
$DC 0
DAC T1
LAC* T1
DAC T1
LAC* T1
AND =777770
SAD =762010
JMP ++3
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
TAD =7
JMS SLT
TAD =2
DAC T3
LAW 6240
DAC* T1
LACQ
DAC* T3
JMP* SLSP
RETURN
SAVE ADDRESS OF LEVEL
FORM POINTER TO END OF HEAD
GET ADDRESS OF TAIL
FORM POINTER TO TASK ADDRESS
SAVE TEMPORARILY
GET SKIP INSTRUCTION
STORE IN TAIL
GET NEW SERVICE TASK ADDRESS
STORE IN TAIL
RETURN
STITLE TEXT OPERATORS

**LT**
DAC LT2
LAC* LT2
CMA
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
CMA
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.D
DAC T1

SAVE ADDRESS OF TEXT LIST
GET TEXT WORD COUNT
FORM 1'S COMPLEMENT
STORE COMPLEMENTED WORD COUNT
FORM 2'S COMPLEMENT OF WORD COUNT
WORD COUNT NOT ZERO
RETURN
SET POINTER TO NEXT TEXT WORD
GET TEXT WORD
SEND TO TELEPRINTER BUFFER
INCREMENT COUNT & SKIP IF DONE
PROCESS NEXT TEXT WORD
RETURN
SAVE ADDRESS OF TEXT LIST
SET POINTER TO TEXT LIST
GET WORD COUNT
FORM 1'S COMPLEMENT
STORE COMPLEMENTED WORD COUNT
GET INITIAL VALUE OF LEAF LENGTH
SET INITIAL VALUE OF LEAF LENGTH
FORM 2'S COMPLEMENT OF WORD COUNT
WORD COUNT NOT ZERO
RETURN NULL TEXT LEAF
SET POINTER TO NEXT TEXT WORD
GET TEXT WORD
SHIFT FIRST CHARACTER INTO POSITION
MODIFY LEAF LENGTH COUNT
MODIFY LEAF LENGTH COUNT
MODIFY LEAF LENGTH COUNT
INCREMENT WORD COUNT & SKIP IF DONE
PROCESS NEXT TEXT WORD
GET SIZE OF TEXT LEAF
DIVIDE BY 2
DIVIDE BY 2
GET STORAGE FOR TEXT LEAF
NOT ENOUGH STORAGE
SAVE ADDRESS OF TEXT LEAF AREA
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
A-85

LD6
SAD = 77
JMP ++5
SAD = 74
ISZ T3
ISZ T3
ISZ T3
LLS 6
JMP* LD5

SAD = 77
JMP* ++5
SAD = 74
ISZ T3
ISZ T3
ISZ T3
LLS 6
JMP* LD5

LD6  SDC 0
AND = 77
SAD = 77
JMP LD7-2
SAD = 74
JMP LD7
SAD = 75
SKP
JMP **3
ISZ T4
SKP
ISZ T3
TAD = LB8
DAC T7
LAN 2010
DAC* T1
ISZ T1
LAC* T7
DAC* T1
ISZ T1
LLS 6
JMP* LD6

LD7  LACQ
DAC T7
LAN 1121
DAC* T1
ISZ T1
LAW 0
DAC* T1
ISZ T1
LAC T3
A-86

PREPARE TO SHIFT ZEROS INTO AC
MULTIPLY BY 8
CONVERT MODULO 2^10
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

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

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+100000
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
<table>
<thead>
<tr>
<th>$DC</th>
<th>D25+250000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$DC</td>
<td>D26+260000</td>
</tr>
<tr>
<td>$DC</td>
<td>D27+270000</td>
</tr>
<tr>
<td>$DC</td>
<td>D30+300000</td>
</tr>
<tr>
<td>$DC</td>
<td>D31+310000</td>
</tr>
<tr>
<td>$DC</td>
<td>D32+320000</td>
</tr>
<tr>
<td>$DC</td>
<td>D33+330000</td>
</tr>
<tr>
<td>$DC</td>
<td>D34+340000</td>
</tr>
<tr>
<td>$DC</td>
<td>D35+350000</td>
</tr>
<tr>
<td>$DC</td>
<td>D36+360000</td>
</tr>
<tr>
<td>$DC</td>
<td>D37+370000</td>
</tr>
<tr>
<td>$DC</td>
<td>D40+400000</td>
</tr>
<tr>
<td>$DC</td>
<td>D41+410000</td>
</tr>
<tr>
<td>$DC</td>
<td>D42+420000</td>
</tr>
<tr>
<td>$DC</td>
<td>D43+430000</td>
</tr>
<tr>
<td>$DC</td>
<td>D44+440000</td>
</tr>
<tr>
<td>$DC</td>
<td>D45+450000</td>
</tr>
<tr>
<td>$DC</td>
<td>D46+460000</td>
</tr>
<tr>
<td>$DC</td>
<td>D47+470000</td>
</tr>
<tr>
<td>$DC</td>
<td>D50+500000</td>
</tr>
<tr>
<td>$DC</td>
<td>D51+510000</td>
</tr>
<tr>
<td>$DC</td>
<td>D52+520000</td>
</tr>
<tr>
<td>$DC</td>
<td>D53+530000</td>
</tr>
<tr>
<td>$DC</td>
<td>D54+540000</td>
</tr>
<tr>
<td>$DC</td>
<td>D55+550000</td>
</tr>
<tr>
<td>$DC</td>
<td>D56+560000</td>
</tr>
<tr>
<td>$DC</td>
<td>D57+570000</td>
</tr>
<tr>
<td>$DC</td>
<td>D60+600000</td>
</tr>
<tr>
<td>$DC</td>
<td>D61+610000</td>
</tr>
<tr>
<td>$DC</td>
<td>D62+620000</td>
</tr>
<tr>
<td>$DC</td>
<td>D63+630000</td>
</tr>
<tr>
<td>$DC</td>
<td>D64+640000</td>
</tr>
<tr>
<td>$DC</td>
<td>D65+650000</td>
</tr>
<tr>
<td>$DC</td>
<td>D66+660000</td>
</tr>
<tr>
<td>$DC</td>
<td>D67+670000</td>
</tr>
<tr>
<td>$DC</td>
<td>D70+700000</td>
</tr>
<tr>
<td>$DC</td>
<td>D71+710000</td>
</tr>
<tr>
<td>$DC</td>
<td>D72+720000</td>
</tr>
<tr>
<td>$DC</td>
<td>D73+730000</td>
</tr>
</tbody>
</table>

NOP
$\text{SDC D75+750000}$
$\text{SDC D76+760000}$
STITLE

IDLE-TIME TASK

IDLE LAW 17475
JMS B.T
LAW 10
JMS T.R
JMS B.K
DAC T3
LAW 10
JMS T.A
LAC BKF
SAD =205
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

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
STEXT "CLEAR"

IDLER SDC RUN
SDC 2
STEXT "RUN"
SDC 747575

IDLES SDC SCHED
SDC 3
STEXT "SCHEDULE"

IDLE1 SDC TTY201
SDC 0

IDLEF SDC FROM
SDC 2
STEXT "FROM"

CLEAR JMS B.K
SAD =15
JMP CLEARI
SAD =35
SKIP IF NOT D
JMP =IDLEQ
JMP LAC
LAC =CLEARD
JMS L-T
LAC =SCHEDQ
LAC =SCHEDQ
JMS Q-C
JMP IDLE
JMP IDLE
GET KEYBOARD CHARACTER
CLEAR DISPLAY STORAGE
CLEAR DISPLAY STORAGE
CLEAR TASK QUEUE
CLEAR TASK QUEUE
CANCEL COMMAND
GET ADDRESS OF TEXT LIST
GET ADDRESS OF TEXT LIST
GET ADDRESS OF FROZEN TASK QUEUE
GET ADDRESS OF FROZEN TASK QUEUE
GET NEW COMMAND
GET NEW COMMAND
CREATE TASK QUEUE
CREATE TASK QUEUE
GET ADDRESS OF TEXT LIST
GET ADDRESS OF TEXT LIST
CLEAR DISPLAY STORAGE
CLEAR DISPLAY STORAGE
CLEAR DISPLAY STORAGE
CLEAR DISPLAY STORAGE
INDICATE NO DIAGNOSTIC
RE-ESTABLISH DISPLAYED TITLE

CLEARD SDC 5
STEXT "DISPLAY STORAGE"

CLEART SDC 4
STEXT "TASK QUEUE"

RUN LAW 10
JMS T.R
JMS STC
GET TELEPRINTER MASK
RELEASE TELEPRINTER
CLEAR DISPLAY STORAGE
A-91

RUN1
LAC = SCHEDQ
JMS Q.F
JMS T.F
JMS T.S
JMP RUN1

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

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

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

TTY201 LAW 17772
JMS B.T
JMS ECHO
LAC BKF
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 ADDRESS OF FROZEN TASK QUEUE
GET TASK FROM FROZEN TASK QUEUE
TERMINATE IDLE-TIME EXECUTION
SCHEDULE TASK FROM FROZEN QUEUE
ENABLE NEXT TASK

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

GET # CODE
TYPE #
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
TTY1
JMP IDLE
LAW BFETX
JMS B.FO
JMP IDLE
TTY2
LAC =BF OB
SAD BF0I
JMP TTY201+2
JMS B.FO
JMP TTY1-3
LAW -2
TAD BF0I
JMP TTY201+2
TTY3
JMS B.FO
NOP
LAC =BFOB
DAC BF0I
LAC =TTY6
JMS L.T
JMP IDLE
TTY4
LAW BFENG
JMS B.FO
JMP TTY1+1
TTY5
SDC 11
STEXT "-- DATA SET NOT CONNECTED"
TTY6
SDC 4
STEXT "-- DELETED"
FROM
JMS B.K
DZM T3
SAD =14
JMP FROM1
ISZ T3
SAD =31
JMP FROM2
ISZ T3
SAD =35
JMP FROM3
JMP IDLE9
GET NEW COMMAND
GET END OF TEXT CHARACTER
SEND TO 201 OUTPUT BUFFER
DATA SET NOT CONNECTED
GET NEW COMMAND
GET ADDRESS OF OUTPUT BUFFER
SKIP IF BUFFER NON-EMPTY
IGNORE CHARACTER DELETE
WAIT FOR ACK TO LAST RECORD
DATA SET NOT CONNECTED
LOAD AC WITH -2
COMPUTE NEW VALUE OF INPUT POINTER
BACKSPACE OUTPUT BUFFER
PROCESS NEXT CHARACTER
WAIT FOR ACK TO LAST RECORD
DATA SET NOT CONNECTED
GET ADDRESS OF 201 OUTPUT BUFFER
RESET INPUT POINTER
TYPE DIAGNOSTIC
GET NEW COMMAND
GET ENQUIRY
SEND TO 201 OUTPUT BUFFER
GET KEYBOARD CHARACTER
CLEAR DATA TRANSFER POINTER
SKIP IF NOT C
FROM CORE
INCREMENT DATA TRANSFER POINTER
SKIP IF NOT P
FROM PAPER TAPE
INCREMENT DATA TRANSFER POINTER
SKIP IF NOT T
FROM TELETYPENO CANCEL COMMAND
FROM1  LAC =FROMC
      JMP FROM4  GET ADDRESS OF TEXT LIST
FROM2  LAC =FROMP
      SKP     TYPE TEXT LIST
FROM3  LAC =FROMT
      GET ADDRESS OF TEXT LIST
FROM4  JMS L.T
      GET ADDRESS OF TEXT LIST
      LAC T3
      TYPE TEXT LIST
      CALL+RTL
      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
      JMP FROM9
      GET ADDRESS OF TEXT LIST
FROM6  LAC =FROMP
      JMP FROM9
      GET ADDRESS OF TEXT LIST
FROM7  LAC =FROMT
      JMP FROM9
      GET ADDRESS OF TEXT LIST
      SKP
      TYPE TEXT LIST
FROM8  LAC =FROMMD
      GET ADDRESS OF TEXT LIST
FROM9  JMS L.T
      TYPE TEXT LIST
      LAC* T3
      GET ADDRESS OF DATA TRANSFER
      DAC T3
      SAVE TEMPORARILY
      JMP* T3
      BEGIN DATA TRANSFER
FROM11  SDC TRCC
FROM11  SDC TRCP
FROM11  SDC TRCT
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
<table>
<thead>
<tr>
<th>LAW 100</th>
<th>GET ORIGIN CONTROL BIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC T3</td>
<td>SET CONTROL MASK</td>
</tr>
<tr>
<td>LAC TRBKL</td>
<td>GET ORIGIN OF BLOCK</td>
</tr>
<tr>
<td>JMS TRCP2</td>
<td>PUNCH ORIGIN</td>
</tr>
<tr>
<td>LAW 200</td>
<td>GET DATA CONTROL BIT</td>
</tr>
<tr>
<td>DAC T3</td>
<td>SET CONTROL MASK</td>
</tr>
<tr>
<td>TRCP1 LAC TRBKL</td>
<td>GET DATA WORD</td>
</tr>
<tr>
<td>JMS TRCP2</td>
<td>PUNCH DATA WORD</td>
</tr>
<tr>
<td>ISZ TRBKL</td>
<td>INCREMENT POINTER</td>
</tr>
<tr>
<td>ISZ TRBKC</td>
<td>INCREMENT COUNT &amp; SKIP IF DONE</td>
</tr>
<tr>
<td>JMP TRCP1</td>
<td>PUNCH NEXT WORD</td>
</tr>
<tr>
<td>JMP IDLE</td>
<td>GET NEW COMMAND</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TRCP2 SDC 0</th>
<th>SAVE WORD TO BE PUNCHED</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC T4</td>
<td>SHIFT HIGH ORDER 'HITS INTO POSITION</td>
</tr>
<tr>
<td>LRS 14</td>
<td>TRUNCATE BITS FROM LINK</td>
</tr>
<tr>
<td>AND #77</td>
<td>SET CONTROL BIT</td>
</tr>
<tr>
<td>XOR T3</td>
<td>PUNCH IMAGE</td>
</tr>
<tr>
<td>JMS PUNCH</td>
<td>GET WORD TO BE PUNCHED</td>
</tr>
<tr>
<td>LAC T4</td>
<td>SHIFT MIDDLE BITS INTO POSITION</td>
</tr>
<tr>
<td>LRS 6</td>
<td>TRUNCATE HIGH ORDER BITS</td>
</tr>
<tr>
<td>AND #77</td>
<td>SET CONTROL BIT</td>
</tr>
<tr>
<td>XOR T3</td>
<td>PUNCH IMAGE</td>
</tr>
<tr>
<td>JMS PUNCH</td>
<td>GET WORD TO BE PUNCHED</td>
</tr>
<tr>
<td>LAC T4</td>
<td>TRUNCATE HIGH ORDER BITS</td>
</tr>
<tr>
<td>AND #77</td>
<td>SET CONTROL BIT</td>
</tr>
<tr>
<td>XOR T3</td>
<td>PUNCH IMAGE</td>
</tr>
<tr>
<td>JMP TRCP2</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TRCT JMS TRBK</th>
<th>GET CORE BLOCK FROM KEYBOARD</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAW 17475</td>
<td>GET CARRIAGE RETURN, LINE FEED CODE</td>
</tr>
<tr>
<td>JMS B.T</td>
<td>TYPE CARRIAGE RETURN, LINE FEED</td>
</tr>
<tr>
<td>LAC TRBKL</td>
<td>GET ADDRESS TO BE TYPED</td>
</tr>
<tr>
<td>JMS C.B6</td>
<td>CONVERT TO 6-BIT CODE</td>
</tr>
<tr>
<td>TAD =770000</td>
<td>REMOVE HIGH ORDER ZERO</td>
</tr>
<tr>
<td>JMS TRKT</td>
<td>TYPE ADDRESS</td>
</tr>
<tr>
<td>LAW 17676</td>
<td>GET CODE FOR TWO SPACES</td>
</tr>
<tr>
<td>JMS B.T</td>
<td>TYPE TWO SPACES</td>
</tr>
<tr>
<td>LAW 17770</td>
<td>LOAD AC WITH -8</td>
</tr>
</tbody>
</table>
A-96

DAC T3
TRCT1 LAW 17677 GET CODE FOR ONE SPACE
JMS B+T TYPE SPACE
LAC* TRBKL GET WORD TO BE TYPED
JMS C+6 CONVERT TO 6-BIT CODE
JMS TRK INITIALIZE LOCATION POINTER
ISZ TRBKL INCREMENT COUNT & SKIP IF DONE
ISZ TRBKC TYPE NEXT WORD
JMP IDLE GET NEW COMMAND
ISZ T3 SKIP IF END OF LINE
JMP TRCT1 TYPE NEXT WORD
JMP TRCT+1 BEGIN NEW LINE

TRCD JMS TRBK GET CORE BLOCK FROM KEYBOARD
LAC TRBKc GET WORD COUNT
TAD =100 MAKE POSITIVE IF NOT TOO LARGE
SMA SKIP IF TOO LARGE
JMP ++3 WORD COUNT OK
LAW 17700 LOAD AC WITH -64
DAC TRBKc ADJUST WORD COUNT
JMS TRD1 INITIALIZE TEXT LIST FOR DISPLAY
LAC TRBKl GET ADDRESS TO BE DISPLAYED
JMS C+6 CONVERT TO 6-BIT CODE
TAD =770000 REMOVE HIGH ORDER ZERO
JMS TRD2 PUT HIGH ORDER DIGITS IN TEXT LIST
LACQ GET LOW ORDER DIGITS
JMS TRD2 PUT LOW ORDER DIGITS IN TEXT LIST
LAW 17676 GET CODE FOR TWO SPACES
JMS TRD2 PUT IN TEXT LIST
LAW 17770 LOAD AC WITH -8
DAC T3 SET WORD COUNTER
TRCD2 LAW 17677 GET CODE FOR ONE SPACE
JMS TRD2 PUT IN TEXT LIST
LAC* TRBKl GET WORD TO BE DISPLAYED
JMS C+6 CONVERT TO 6-BIT CODE
JMS TRD2 PUT HIGH ORDER DIGITS IN TEXT LIST
LACQ GET LOW ORDER DIGITS
JMS TRD2 PUT LOW ORDER DIGITS IN TEXT LIST
ISZ TRBKl INCREMENT LOCATION POINTER
INCREMENT COUNT & SKIP IF DONE
PREPARE NEXT WORD
GET THREE NULL CHARACTERS
NULLIFY ACCUMULATED CHARACTERS
DISPLAY TEXT LIST
GET NEW COMMAND
SKIP IF END OF LINE
PREPARE NEXT WORD
GET CARRIAGE RETURN, LINE FEED CODE
PUT IN TEXT LIST
BEGIN NEW LINE
READ ONE TAPE IMAGE
SKIP IF NOT END OF RECORD
GET NEW COMMAND
SAVE TAPE LINE
GET CONTROL BITS
SKIP IF NOT ORIGIN
COMPLETE ORIGIN
SKIP IF NOT BINARY DATA
COMPLETE DATA WORD
READ A TAPE IMAGE
SKIP IF NOT END OF RECORD
RESTART DATA TRANSFER
COMPLETE DATA WORD
READ A TAPE IMAGE
GET NEXT WORD FROM TAPE
SET ORIGIN
FINISH READING ORIGIN
GET NEXT WORD FROM TAPE
FINISH READING DATA WORD
LOAD DATA WORD
INCREMENT LOCATION COUNTER
GET NEXT WORD FROM TAPE
GET SECOND IMAGE FROM TAPE
SHIFT DATA BITS INTO MQ
GET HIGH ORDER 6 BITS
SHIFT HIGH ORDER 12 BITS INTO AC
SAVE HIGH ORDER 12 BITS
GET THIRD IMAGE FROM TAPE
A-98

LRS 6  SHIFT DATA BITS INTO MO
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  SKIP IF NOT END-OF-TAPE GARBAGE
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  SKIP IF NOT END OF RECORD
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
TITLE: 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

**TRPT2**
- XOR = 777400
- JMS B.T
- JMS READ
- 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
- 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
- RESTART DATA TRANSFER
- SAVE TEMPORARILY
- GET CONTROL BITS
- SKIP IF NOT END OF RECORD GARBAGE
- RESTART DATA TRANSFER
- 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:
SKIP  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
A-101

TRTC2  DCT  T5
      ISZ  T5
      LAW  17475
      JMS  B.T
      LAC  T5
      JMS  C86
      TAD  =770000
      JMS  TRKT
      JMP  TRTC1

TRTC3  SAD  =74
      JMP  TRTC2
      JMP  TRTC

TRTC4  DCT  T3
      LAW  10
      JMS  T.R
      JMP  IDLE+6

TRTP  CLA
      JMS  PUNCH
      LAW  17475
      JMS  B.T

TRTP1  JMS  ECHO
       SAD  =77
      JMP  TRTP2
      XOR  =300
      JMS  PUNCH
      JMP  TRTP1

TRTP2  CLA
       JMS  PUNCH
      JMP  IDLE

TRTT  LAW  17475
      JMS  B.T
      JMS  ECHO
      SAD  =77
      JMP  IDLE
      JMP  =-3

TRTD  LAW  17766
      DAC  T4

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 IT
ECHO KEYBOARD CHARACTER
SKIP IF NOT NULL CHARACTER
TERMINATE TRANSFER
SET ALPHANUMERIC CONTROL BITS
PUNCH CHARACTER
GET NEXT CHARACTER
GET END-OF-RECORD MARK
PUNCH IT
GET NEW COMMAND
LOAD AC WITH -10
INITIALIZE LINE COUNTER
A-102

TRTD1
JMS TRD1
INITIALIZE TEXT LIST
LAW 17475
GET CARRIAGE RETURN, LINE FEED CODE
JMS B.T
TYPE CARRIAGE RETURN, LINE FEED
LAW 17700
LOAD AC WITH -64
DAC T5
INITIALIZE CHARACTER COUNTER

TRTD2
JMS ECHO
ECHO KEYBOARD CHARACTER
SAD =77
SKIP IF NOT NULL CHARACTER
JMP TRTD4
DISPLAY TEXT LIST
SAD =74
SKIP IF NOT CARRIAGE RETURN
JMP TRTD3
TERMINATE LINE
JMS TRD4
ADD CHARACTER TO TEXT LIST
ISZ T5
SKIP IF END OF LINE
JMP TRTD2
GET NEXT CHARACTER

TRTD3
ISZ T4
INCREMENT LINE COUNT & SKIP IF DONE
SKP
TERMINATE LINE
JMP TRTD4
TERMINATE TRANSFER
LAW 74
GET CARRIAGE RETURN CODE
JMS TRD4
ADD TO TEXT LIST
LAW 75
GET LINE FEED CODE
JMS TRD4
ADD TO TEXT LIST
LAW 75
GET LINE FEED CODE
JMS TRD4
ADD TO TEXT LIST
JMP TRTD1
BEGIN NEW LINE

TRTD4
JMS TRD3
DISPLAY TEXT LIST
JMP IDLE
GET NEW COMMAND

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

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

PUNCH1 $DC 7
$DC 7 47531
$TEXT "UNCH OUT OF TAPE"

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

READ1 $DC 7
$DC 7 47533
$TEXT "READER OUT OF TAPE"

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

TRD2 $DC 0
ISZ TRDT
INCREMENT TEXT LIST COUNT
ISZ TRDP
INCREMENT TEXT LIST POINTER
DAC* TRDP
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 TRD3
DAC 27
LMO
LAC 26
JMS S=TI
JMP TRD3
JMP* TRD3

TRD3I LAC ■•♦3
JMS L=T
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  SDS 351
STC  $DC @  GET ADDRESS OF HIGHEST ACTIVE LEVEL
    LAC =D  REMOVE ALL NODES FROM HAL
    DAC DHAL+7  WAIT FOR DISPLAY TO RECOVER
    JMS DW  GET INITIAL COUNTER VALUE
    LAW STORE  SET POINTER & COUNTER
    IOT 7704  LEAVE EXTEND MODE
    DZM* T1  CLEAR STORAGE LOCATION
    ISZ T1  INCREMENT POINTER & COUNTER
    JMP -2  CLEAR NEXT STORAGE LOCATION
    IOT 7702  ENTER EXTEND MODE
    JMP* STC  RETURN

OCTAL1 SDC @  GET KEYBOARD CHARACTER
    JMS B.K  MAKE NEGATIVE IF OCTAL
    TAD =-10  SKIP IF NOT OCTAL DIGIT
    SPA  OCTAL DIGIT TYPED
    JMP ++3  RESTORE CHARACTER
    TAD =10  INDICATE FAILURE
    JMP* OCTAL1  CONVERT TO 6-BIT CODE
    DAC T3  TYPE OCTAL DIGIT
    XOR =70  GET OCTAL INFORMATION
    JMS B.T  SHIFT DIGIT INTO MQ
    LAC T3  GET RECORDED DIGITS
    LRS 3  CONCATENATE NEW DIGIT
    LAC T4  RECORD NEW WORD
    ISZ OCTAL1  INDICATE SUCCESS
    JMP* OCTAL1  RETURN

OCTAL5 SDC @  CLEAR OCTAL RECORDING WORD
    DZM T4  GET OCTAL DIGIT FROM KEYBOARD
    JMS OCTAL1  NON-OCTAL CHARACTER TYPED
    JMP* OCTAL5  GET OCTAL DIGIT FROM KEYBOARD
    JMS OCTAL1  NON-OCTAL CHARACTER TYPED
    JMP* OCTAL5  GET OCTAL DIGIT FROM KEYBOARD
    JMS OCTAL1  NON-OCTAL CHARACTER TYPED
    JMP* OCTAL5  GET OCTAL DIGIT FROM KEYBOARD
    JMS OCTAL1  GET OCTAL DIGIT FROM KEYBOARD
A-106

JMP* OCTAL5       NON-OCTAL CHARACTER TYPED
JMS OCTAL1        GET OCTAL CHARACTER FROM KEYBOARD
JMP* OCTAL5       NON-OCTAL CHARACTER TYPED
ISZ OCTAL5        INDICATE SUCCESS
JMP* OCTAL5       RETURN

OCTAL6 $DC 0
JMS OCTAL5        GET 5 OCTAL DIGITS FROM KEYBOARD
JMP* OCTAL6       NON-OCTAL CHARACTER TYPED
JMS OCTAL1        GET OCTAL DIGIT FROM KEYBOARD
JMP* OCTAL6       NON-OCTAL CHARACTER TYPED
ISZ OCTAL6        INDICATE SUCCESS
JMP* OCTAL6       RETURN

TRKT $DC 0
DAC T1            SAVE HIGH ORDER DIGITS
LACQ              GET LOW ORDER DIGITS
DAC T6            SAVE LOW ORDER DIGITS
LAC T1            GET HIGH ORDER DIGITS
JMS B*T           TYPE HIGH ORDER DIGITS
LAC T6            GET LOW ORDER DIGITS
JMS B*T           TYPE LOW ORDER DIGITS
JMP* TRKT         RETURN

TRBK $DC 0
LAC =TRBKFL       GET ADDRESS OF TEXT LIST
JMS L*T           TYPE TEXT LIST
JMS OCTAL5        GET LOW ADDRESS FROM KEYBOARD
JMP IDLEQ         CANCEL COMMAND
DAC TRBKL         STORE LOW ADDRESS
LAW 16277         GET COMMA CODE
JMS B*T           TYPE COMMA
JMS OCTAL5        GET HIGH ADDRESS FROM KEYBOARD
JMP IDLEQ         CANCEL COMMAND
CMA                FORM ONE'S COMPLEMENT
TAD TRBKL         ADD LOW ADDRESS
SMA                SKIP IF PROPERLY ORDERED ADDRESSES
JMP IDLEQ         CANCEL COMMAND
DAC TRBKC         STORE LOCATION COUNT
LAW 15177         GET RIGHT PARENTHESIS CODE
JMS B.T
JMP* TRBK

TYPE RIGHT PARENTHESIS
RETURN

TRBK$ $DC 3
$DC 747513
$TEXT "LOCK"
<table>
<thead>
<tr>
<th>TITLE</th>
<th>HIGHEST ACTIVE LEVEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>$DC 757</td>
</tr>
<tr>
<td></td>
<td>$DC 6201</td>
</tr>
<tr>
<td></td>
<td>$DC 6301</td>
</tr>
<tr>
<td></td>
<td>$DC 1400</td>
</tr>
<tr>
<td>DHAL</td>
<td>$DC DWT</td>
</tr>
<tr>
<td></td>
<td>$DC 2010</td>
</tr>
<tr>
<td></td>
<td>$DC XP</td>
</tr>
<tr>
<td></td>
<td>$DC 1105</td>
</tr>
<tr>
<td></td>
<td>$DC 1000</td>
</tr>
<tr>
<td></td>
<td>$DC 5000</td>
</tr>
<tr>
<td></td>
<td>$DC 2001</td>
</tr>
<tr>
<td></td>
<td>$DC D</td>
</tr>
</tbody>
</table>
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>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.F0</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.CB</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>
<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>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.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>
</tbody>
</table>
BIT INTERPRETATION
---
16 TRANSMIT STATE
17 RECEIVE STATE

DISPLAY STATUS WORD 1
BIT INTERPRETATION
---
6 LIGHT PEN FLAG
7 VERTICAL EDGE FLAG
8 HORIZONTAL EDGE FLAG
9 INTERNAL STOP FLAG
10 SECTOR 0 FLAG (DISPLAY COORDINATES ARE ON SCREEN)
11 CONTROL STATE
12 MANUAL INTERRUPT FLAG
13 PUSH BUTTON FLAG
14 DISPLAY INTERRUPT PENDING
15 BREAK FIELD REGISTER BIT 0
16 BREAK FIELD REGISTER BIT 1
17 BREAK FIELD REGISTER BIT 2

DISPLAY STATUS WORD 2
BIT INTERPRETATION
---
6 0 -- LEFT HAND INCREMENT BEING EXECUTED
  1 -- RIGHT HAND INCREMENT BEING EXECUTED
7 LIGHT PEN ENABLED
8 BIT 0 OF Y POSITION REGISTER
9 BIT 0 OF X POSITION REGISTER
10 SCALE BIT 0
11 SCALE BIT 1
12 MODE BIT 0
13 MODE BIT 1
14 MODE BIT 2
15 INTENSITY BIT 0
16 INTENSITY BIT 1
17 INTENSITY BIT 2
### 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>
<tr>
<td>8</td>
<td>0 -- DO NOT DISABLE LIGHT PEN AFTER RESUMING DISPLAY</td>
</tr>
<tr>
<td></td>
<td>1 -- ENABLE LIGHT PEN ACCORDING TO BIT 9</td>
</tr>
<tr>
<td>9</td>
<td>0 -- ENABLE LIGHT PEN AFTER FIRST DATA REQUEST AFTER RESUMING DISPLAY</td>
</tr>
<tr>
<td></td>
<td>1 -- DO NOT ENABLE LIGHT PEN AFTER RESUMING DISPLAY</td>
</tr>
<tr>
<td>10</td>
<td>BIT 0 OF Y DIMENSION</td>
</tr>
<tr>
<td>11</td>
<td>BIT 1 OF Y DIMENSION</td>
</tr>
<tr>
<td>12</td>
<td>BIT 0 OF X DIMENSION</td>
</tr>
<tr>
<td>13</td>
<td>BIT 1 OF X DIMENSION</td>
</tr>
<tr>
<td>14</td>
<td>INTENSIFY ALL POINTS</td>
</tr>
<tr>
<td>15</td>
<td>INHIBIT EDGE FLAGS</td>
</tr>
<tr>
<td>16</td>
<td>ENABLE PUSH BUTTON INTERRUPT</td>
</tr>
<tr>
<td>17</td>
<td>ENABLE INTERNAL STOP INTERRUPT</td>
</tr>
</tbody>
</table>

### 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>
<tr>
<td>11</td>
<td>0 -- LOAD PUSH BUTTONS 0-5</td>
</tr>
<tr>
<td></td>
<td>1 -- LOAD PUSH BUTTONS 6-11</td>
</tr>
<tr>
<td>12</td>
<td>PUSH BUTTON 0 OR 6</td>
</tr>
<tr>
<td>13</td>
<td>PUSH BUTTON 1 OR 7</td>
</tr>
<tr>
<td>14</td>
<td>PUSH BUTTON 2 OR 8</td>
</tr>
<tr>
<td>15</td>
<td>PUSH BUTTON 3 OR 9</td>
</tr>
<tr>
<td>16</td>
<td>PUSH BUTTON 4 OR 10</td>
</tr>
<tr>
<td>17</td>
<td>PUSH BUTTON 5 OR 11</td>
</tr>
</tbody>
</table>
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 TELETYPewriter 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.

THE OPERAND FIELD CONTAINS A SET OF PROGRAM SYMBOLS AND/OR NUMBERS SEPARATED BY THE BINARY OPERATOR SYMBOLS "•" (2'S COMPLEMENT ADDITION) AND/OR "-" (2'S COMPLEMENT SUBTRACTION). IN ADDITION, THE FIRST PROGRAM SYMBOL OR NUMBER MAY BE PRECEDED BY EITHER OF THE UNARY OPERATORS "+" (UNARY PLUS) OR "-" (2'S COMPLEMENT). LITERALS ARE DENOTED BY AN EQUAL SIGN (=) APPENDED TO THE LEFT END OF THE OPERAND FIELD. AN ASTERISK (*) REPRESENTS A MNEMONIC Whose VALUE IS THE ADDRESS OF THE LOCATION Which THE SOURCE LINE IN WHICH IT APPEARS REPRESENTS (IN THE OPERAND FIELD ONLY). THE LOW ORDER 13 BITS OF THE VALUE OF THE EXPRESSION IN THE OPERAND FIELD ARE ADDED TO THE VALUE REPRESENTED BY THE INSTRUCTION FIELD.

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:
$DC A word which contains the full 18-bit value of the expression in the operand field is produced.

$DS 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.)

$END The end of the source program is declared.

$EOU 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 operandless 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 teletype 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.