THE UNIVERSITY OF MICHIGAN

Technical Report 15

AN EXECUTIVE SYSTEM
FOR A DEC 339 COMPUTER DISPLAY TERMINAL

James H. Jackson

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

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

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

Administered through:
OFFICE OF RESEARCH ADMINISTRATION ANN ARBOR

December 1968
ABSTRACT

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

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

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

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

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

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

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

2.1 **Bootstrap Arrangement**

The System should be loaded by the following procedure:

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

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

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

The bootstrap loader is capable of loading one binary block (Section 3.4.2) **starting at location 17732_8**, 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_8, ..., 17777_8, 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 and user states]

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 task 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 22\textsubscript{8} via the panel switches.
7) An illegal instruction (operation code 00\textsubscript{8}) 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:

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

\[
------ (\text{return})
\]

Subroutine with failure return:

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

\[
------ (\text{failure return})
\]

\[
------ (\text{success return})
\]

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 a word queue]

The symbols in the diagram are interpreted as follows:

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

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

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

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

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

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. \( (q_2 \text{ and } q_3 \text{ are both set equal to } q_1) \).

**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. (q₃ is incremented, and the word stored in the location which the resulting q₃ 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 qₒ and q₁, since, if the queue is cleared (via Q.C) before it is used, the pointers q₂ and q₃ will be automatically established. For example, the word queue whose address is Q may be constructed by the following two statements, where ε is an expression whose value is n + 2:

Q $DC *+ε
$DS ε

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</th>
<th>=FIFO</th>
<th>GET ADDRESS OF FIFO QUEUE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JMS*</td>
<td>=Q.C</td>
<td>CLEAR FIFO QUEUE</td>
</tr>
<tr>
<td></td>
<td>LAC</td>
<td>=LIFO</td>
<td>GET ADDRESS OF LIFO QUEUE</td>
</tr>
<tr>
<td></td>
<td>JMS*</td>
<td>=Q.C</td>
<td>CLEAR LIFO QUEUE</td>
</tr>
<tr>
<td></td>
<td>D2M</td>
<td>COUNT</td>
<td>START COUNTING AT ZERO</td>
</tr>
<tr>
<td>LOOP1</td>
<td>LAC</td>
<td>COUNT</td>
<td>GET VALUE OF INTEGER</td>
</tr>
<tr>
<td></td>
<td>LMQ</td>
<td></td>
<td>SET UP PARAMETER</td>
</tr>
<tr>
<td></td>
<td>LAC</td>
<td>=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:
SCHED    JMS    CHECK    SKIP IF SWITCH IS SET
JMS*  =T.P    WAIT FOR SWITCH TO BE SET
LAC    =TASK1    GET ADDRESS OF FIRST TASK
JMS*  =T.S    SCHEDULE FIRST TASK
LAC    =TASK2    GET ADDRESS OF SECOND TASK
JMS*  =T.S    SCHEDULE SECOND TASK
JMS*  =T.F    TERMINATE TASK
CHECK    $DC  0
LAC    SWITCH    GET SWITCH VALUE
SZA    SKIP IF SWITCH NOT SET
ISZ    CHECK    INDICATE SUCCESS
JMP*   CHECK    RETURN

The call to T.P is given whenever the subroutine CHECK produces a failure return (in the same sense that some system subroutines produce failure returns) to reschedule the call to CHECK. Because tasks are scheduled on a first-in first-out basis, the rescheduled call to CHECK is not executed until each other eligible task in the task queue has been executed.

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

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

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

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

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

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

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

Task #1:

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

Task #2:

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

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

Task #1:

<table>
<thead>
<tr>
<th>Time</th>
<th>Devices Allocated</th>
<th>Devices Released</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{11}'$</td>
<td>A,B,C</td>
<td>-</td>
</tr>
<tr>
<td>$t_{12}'$</td>
<td>-</td>
<td>B,C</td>
</tr>
<tr>
<td>$t_{13}'$</td>
<td>B,C</td>
<td>-</td>
</tr>
<tr>
<td>$t_{14}'$</td>
<td>-</td>
<td>A,C</td>
</tr>
<tr>
<td>$t_{15}'$</td>
<td>C</td>
<td>-</td>
</tr>
<tr>
<td>$t_{16}'$</td>
<td>-</td>
<td>B,C</td>
</tr>
</tbody>
</table>

Task #2:

<table>
<thead>
<tr>
<th>Time</th>
<th>Devices Allocated</th>
<th>Devices Released</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{21}'$</td>
<td>B,C</td>
<td>-</td>
</tr>
<tr>
<td>$t_{22}'$</td>
<td>-</td>
<td>B</td>
</tr>
<tr>
<td>$t_{23}'$</td>
<td>B</td>
<td>-</td>
</tr>
<tr>
<td>$t_{24}'$</td>
<td>-</td>
<td>B,C</td>
</tr>
</tbody>
</table>

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

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

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

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

Calling sequence for T.L:

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

Calling sequence for T.U:

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

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

```
WAIT  $DC  0
    JMS* =T.L       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 0-11 cleared. Bits 0-9 of the AC and the parity bit of the ASCII code (i.e., bit 10 of the AC) are ignored on entry.
C.CB - The vector mode sign-magnitude display coordinate given in bits 7-17 of the AC is converted to the corresponding two's complement representation, which is returned in the AC. Bits 0-6 of the AC are ignored on entry.

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

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

<table>
<thead>
<tr>
<th>First Octal Digit</th>
<th>Second Octal Digit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 1 2 3 4 5 6 7</td>
</tr>
<tr>
<td>1</td>
<td>8 9 A B C D E F</td>
</tr>
<tr>
<td>2</td>
<td>G H I J K L M N</td>
</tr>
<tr>
<td>3</td>
<td>O P Q R S T U V</td>
</tr>
<tr>
<td>4</td>
<td>W X Y Z * / + -</td>
</tr>
<tr>
<td>5</td>
<td>( ) [ ] &lt; = &gt; †</td>
</tr>
<tr>
<td>6</td>
<td>+ - . : ; ? ! '</td>
</tr>
<tr>
<td>7</td>
<td>&quot; $ # &amp; cr lf sp</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 $77_8$. The only printing characters which are treated in this manner are "$", "", and "".

3.4 Buffered I/O

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

3.4.1 Dataphone I/O

The following system subroutines have been defined for managing the 20l dataphone buffers:

B.FI* - An image is fetched from the 20l 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 20l 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 (026₈). (At least two are required when receiving; eight are transmitted.)

2. A data link escape (DLE) character (220₈).

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 (377₈).

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:
16-bit block check at end of operation (shift register initially clear)

end-of-record character last data image third data image second data image

Data Sequence
(low-order bits of each image used first)

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 (0058) or an end-of-transmission (2048), 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>MIRROR</th>
<th>LAW 600</th>
<th>GET ALLOCATION MASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS*   =T.A</td>
<td>ALLOCATE 201 INPUT &amp; OUTPUT</td>
<td></td>
</tr>
<tr>
<td>LAW 17700</td>
<td>LOAD AC WITH -64</td>
<td></td>
</tr>
<tr>
<td>DAC COUNT</td>
<td>INITIALIZE IMAGE COUNT</td>
<td></td>
</tr>
<tr>
<td>START JMS* =B.FI</td>
<td>GET REDUNDANT IMAGE</td>
<td></td>
</tr>
<tr>
<td>JMP HELP</td>
<td>DATA SET NOT CONNECTED</td>
<td></td>
</tr>
<tr>
<td>READ JMS* =B.FI</td>
<td>GET INPUT IMAGE</td>
<td></td>
</tr>
<tr>
<td>JMP HELP</td>
<td>DATA SET NOT CONNECTED</td>
<td></td>
</tr>
<tr>
<td>SPA</td>
<td>SKIP IF NOT END OF RECORD</td>
<td></td>
</tr>
<tr>
<td>JMP START</td>
<td>READ NEXT RECORD</td>
<td></td>
</tr>
<tr>
<td>JMS* =B.FO</td>
<td>PUT IN OUTPUT BUFFER</td>
<td></td>
</tr>
<tr>
<td>JMP HELP</td>
<td>DATA SET NOT CONNECTED</td>
<td></td>
</tr>
<tr>
<td>ISZ COUNT</td>
<td>SKIP IF RECORD LONG ENOUGH</td>
<td></td>
</tr>
<tr>
<td>JMP READ</td>
<td>READ NEXT IMAGE</td>
<td></td>
</tr>
<tr>
<td>JMS* =B.FI</td>
<td>GET INPUT IMAGE</td>
<td></td>
</tr>
<tr>
<td>JMP HELP</td>
<td>DATA SET NOT CONNECTED</td>
<td></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:

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

There are 64 possible mode changes (designated by the low-order 6 bits of a mode change tape image), only one of which has been defined at the time of this writing, i.e., the end-of-record character 000<sub>8</sub>. (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<sub>8</sub> should contain 621365<sub>8</sub> and that location 23573<sub>8</sub> should contain 176234<sub>8</sub> is the following:

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

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

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  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:

```
ENCODE
LAW 30 GET ALLOCATION MASK
JMS* =T.A ALLOCATE KEYBOARD & TELEPRINTER
JMS* =B.K GET CHARACTER FROM KEYBOARD
SAD =77 SKIP IF NOT NULL CHARACTER
JMP END TERMINATE TASK
```
JMS* =C.B6 CONVERT TO 6-BIT OCTAL CODE
LACQ GET LOW-ORDER DIGITS
XOR =770000 REMOVE HIGH-ORDER ZERO
JMS* =B.T TYPE ENCODED CHARACTER
LAW 17475 GET CARRIAGE RETURN, LINE FEED CODE
JMS* =B.T TYPE CARRIAGE RETURN, LINE FEED
JMP ENCODE+2 PROCESS NEXT CHARACTER
END LAW 30 GET ALLOCATION MASK
JMS* =T.R RELEASE KEYBOARD & TELEPRINTER
JMS* =T.F 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
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 condi-
tion is satisfied, the comment "ANALOG INPUTS MATCH" is typed
on the teletype. An algorithm for this task is the following:
3.6 Push-Button Processing

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:

<table>
<thead>
<tr>
<th>BUTTON</th>
<th>LAW</th>
<th>22</th>
<th>GET ALLOCATION MASK</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS*</td>
<td>=T.A</td>
<td></td>
<td>ALLOCATE KEYBOARD &amp; PUSH BUTTONS</td>
</tr>
<tr>
<td>LAC</td>
<td>=SERV</td>
<td></td>
<td>GET ADDRESS OF SERVICE TASK</td>
</tr>
<tr>
<td>JMS*</td>
<td>=P.T</td>
<td></td>
<td>DECLARE SERVICE TASK</td>
</tr>
<tr>
<td>CLA</td>
<td></td>
<td></td>
<td>GET INITIAL PUSH BUTTON STATE</td>
</tr>
<tr>
<td>DAS</td>
<td></td>
<td></td>
<td>SAVE FOR USE BY SERV</td>
</tr>
<tr>
<td>JMS*</td>
<td>=P.S</td>
<td></td>
<td>SET INITIAL PUSH BUTTON STATE</td>
</tr>
<tr>
<td>JMS*</td>
<td>=P.E</td>
<td></td>
<td>ENABLE MANUAL OPERATION</td>
</tr>
<tr>
<td>JMS*</td>
<td>=B.K</td>
<td></td>
<td>GET KEYBOARD CHARACTER</td>
</tr>
<tr>
<td>JMS*</td>
<td>=P.D</td>
<td></td>
<td>DISABLE MANUAL OPERATION</td>
</tr>
<tr>
<td>CLA</td>
<td></td>
<td></td>
<td>GET FINAL PUSH BUTTON STATE</td>
</tr>
<tr>
<td>JMS*</td>
<td>=P.S</td>
<td></td>
<td>CLEAR PUSH BUTTONS</td>
</tr>
<tr>
<td>CLA</td>
<td></td>
<td></td>
<td>GET NULL SERVICE PARAMETER</td>
</tr>
<tr>
<td>JMS*</td>
<td>=P.T</td>
<td></td>
<td>DECLARE NULL SERVICE TASK</td>
</tr>
<tr>
<td>LAW</td>
<td>22</td>
<td></td>
<td>GET ALLOCATION MASK</td>
</tr>
<tr>
<td>JMS*</td>
<td>=T.R</td>
<td></td>
<td>RELEASE KEYBOARD &amp; PUSH BUTTONS</td>
</tr>
<tr>
<td>JMS*</td>
<td>=T.F</td>
<td></td>
<td>TERMINATE TASK</td>
</tr>
<tr>
<td>SERV</td>
<td>JMS*</td>
<td>=P.R</td>
<td>READ PUSH BUTTONS</td>
</tr>
</tbody>
</table>
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 on 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 x1 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
(1) \( \forall x \in L \quad x \preceq x \)
(2) \( \forall x, y \in L \quad x \preceq y \quad \text{and} \quad y \preceq x \Rightarrow x = y \)
(3) \( \forall x, y, z \in L \quad x \preceq y \quad \text{and} \quad y \preceq z \Rightarrow x \preceq z \)

The semantic interpretation of the expression \( x \preceq y \) 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 \( x \preceq y \), 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 \( x \preceq y \) and there does not exist a level \( z \) different from \( x \) and \( y \) such that \( x \preceq z \) and \( z \preceq y \).

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

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

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

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

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

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

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

$$
(1) \forall x, y \in X \quad x + y \in X \\
(2) \forall x, y \in X \quad x \leq x + y \quad \text{and} \quad y \leq x + y \\
\text{and} \quad (3) \forall x, y, z \in X \quad x \leq z \quad \text{and} \quad y \leq z \quad \implies \quad x + y \leq z ,
$$

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

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

![House drawing](image)

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

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

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

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

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

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

3.10 Level Modification

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

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

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

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

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

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

This subroutine has no effect on the highest active level, where the scale is \( x_8 \), the intensity is 7, and the light pen is disabled.

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

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

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

This subroutine has no effect on the highest active level.

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

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

S.LL - The task whose address is given in bits 3-17 of the MQ is scheduled whenever display interrupts are enabled (via D.E), the tail of the level whose address is given in bits 3-17 of the AC is executed, and the light pen sense indicator has been set during execution of that level. This subroutine has no effect on the highest active level.
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 ---- (Y COORDINATE)
* $DC ---- (X COORDINATE)
* $DC ---- (DISPLAY PARAMETER)
* ---- (RETURN IF DISPLAY STORAGE EXCEEDED)
* ---- (RETURN)
*AC CONTENT ON ENTRY:

* POINT TO ATTRIBUTE

*AC CONTENT ON RETURN:

* POINT 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
LAC LVL5 GET POINTER TO LEVEL
JMS* =S.TI INSERT ATTRIBUTE
JMP LVL2 DISPLAY STORAGE EXCEEDED
LAC* LVL+2 GET FIRST PARAMETER
DAC LVL+2 SAVE FIRST PARAMETER
ISZ LVL+2 ADVANCE TO NEXT PARAMETER
LAC* LVL+2 GET Y COORDINATE
LMQ
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
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
LAC LVL5 GET POINTER TO LEVEL
JMS* =S.LP SET DISPLAY PARAMETER
LAC LVL5 GET POINTER TO LEVEL
LMQ
SET UP PARAMETER
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 ATTR     GET FIRST ATTRIBUTE
$DC 0        LVL PROGRAMMING ERROR
LMQ          SET UP PARAMETER
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* BUTN+2  GET BUTN PARAMETER
DAC* BUTN5  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  SENSITIZE 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
* $DC ---- (ADDRESS OF TEXT LIST)
* $DC ---- (LOC CONTAINING POINTER TO OWNER)
* $DC ---- (Y COORDINATE)
* $DC ---- (X COORDINATE)
* $DC ---- (DISPLAY PARAMETER)
* $DC ---- (SERVICE TASK ADDRESS)
* ---- (RETURN IF DISPLAY STORAGE EXCEEDED)
* ---- (RETURN IF SUCCESSFUL)

*AC CONTENT ON RETURN:

* POINTER TO LIGHT BUTTON LEVEL

BUTX $DC 0
JMS* =T.L SET REENTRY LOCK
$DC 0
LAC* BUTX+2 GET ADDRESS OF TEXT LIST
JMS* =L.D CREATE TEXT LEAF
JMP BUTX4 DISPLAY STORAGE EXCEEDED
DAC BUTX7 SAVE POINTER TEXT LEAF
LAW -6 LOAD AC WITH -6
DAC BUTX5 SET PARAMETER COUNTER
LAC =BUTX2 GET ADDRESS OF BUTN CALL
DAC BUTX6 SET PARAMETER POINTER
BUTX1 ISZ BUTX+2 ADVANCE TO NEXT PARAMETER
ISZ BUTX6 INCREMENT PARAMETER POINTER
ISZ BUTX5 SKIP IF NOT PARAMETER
SKP MOVE PARAMETER
JMP BUTX2-1 CALL BUTN
LAC* BUTX+2 GET PARAMETER
DAC* BUTX6 STORE PARAMETER
JMP BUTX1 MOVE NEXT PARAMETER
LAC BUTX7 GET POINTER TO TEXT LEAF
BUTX2 JMS BUTN CREATE LIGHT BUTTON
$DC 0 LOC CONTAINING POINTER TO OWNER
$DC 0 Y COORDINATE
$DC 0 X COORDINATE
$DC 0 DISPLAY PARAMETER
$DC 0 SERVICE TASK ADDRESS
JMP BUTX3+2 DISPLAY STORAGE EXCEEDED
ISZ BUTX+2 INDICATE SUCCESS
BUTX3 JMS* =T.U UNLOCK BUTX & RETURN
$DC BUTX
LAC BUTX7 GET POINTER TO TEXT LEAF
JMS* =S.LL DESTROY TEXT LEAF
JMP BUTX3 RETURN
BUTX4 LAC BUTX+2 GET RETURN POINTER
TAD =6 ADVANCE PAST PARAMETER LIST
DAC BUTX+2 SET FAILURE RETURN POINTER
JMP BUTX3 RETURN

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

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

*TO BE CHEWEED
*
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 DESTROY LEVEL
JMP CHEW3 GET NEXT ITEM FROM QUEUE
LAC CHEW6 GET POINTER TO LEVEL
LAC CHEW6 GET FIRST ATTRIBUTE FROM LEVEL
JMS ATTR LEVEL IS EMPTY
JMS* =S.TR GET POINTER TO ATTRIBUTE
LAC CHEW7 SAVE POINTER TO ATTRIBUTE
LMQ SET UP PARAMETER
LAC CHEW6 GET POINTER TO LEVEL
JMS* =S.TR REMOVE ATTRIBUTE
$DC 0 CHEW PROGRAMMING ERROR
LAC CHEW7 GET POINTER TO ATTRIBUTE
LMQ SET UP PARAMETER
LAC =CHEWQ GET ADDRESS OF WORD QUEUE
JMS* =Q.A ADD ATTRIBUTE TO QUEUE
$DC 0 WORD QUEUE NOT LARGE ENOUGH
JMP CHEW2 PUT NEXT ATTRIBUTE IN QUEUE
CHEW3 LAC CHEW6 GET POINTER TO LEVEL
JMS* =S.TD DESTROY LEVEL
$DC 0 CHEW PROGRAMMING ERROR
JMP CHEW5 CHEW UP NEXT ITEM
CHEW4 LAC CHEW6 GET POINTER TO TEXT LEAF
JMS* =L.L DESTROY TEXT LEAF
CHEW5 LAC =CHEWQ GET ADDRESS OF WORD QUEUE
JMS* =Q.F GET NEXT ITEM FROM QUEUE
JMP *+3 QUEUE EMPTY
DAC CHEW6 SAVE POINTER TO ITEM
JMP CHEW1 CHEW UP ITEM FROM QUEUE
JMS* =T.U UNLOCK CHEW & RETURN
$DC CHEW

*CALLING SEQUENCE:
* JMS ATTR
* ---- (RETURN IF NO MORE ATTRIBUTES)
* ---- (RETURN IF ATTRIBUTE FOUND)
AC CONTENTS ON ENTRY:

ADDRESS OF LEVEL

ATTR $DC 0
TAD =7 FORM POINTER TO POINTER TO NODE
DAC ATTR2 SAVE POINTER TO POINTER TO NODE
LAC* ATTR2 GET POINTER TO NODE (OR TAIL)
DAC ATTR2 SAVE POINTER TO NODE (OR TAIL)
LAC* ATTR2 GET FIRST WORD FROM NODE (OR TAIL)
AND =777770 TRUNCATE BREAK FIELD
SAD =762010 SKIP IF NOT NODE
SKP NODE FOUND
JMP* ATTR NO MORE ATTRIBUTES
ISZ ATTR2 FORM POINTER TO SECOND LOC IN NODE
LAC* ATTR2 GET POINTER TO ATTRIBUTE
ISZ ATTR INDICATE SUCCESS
JMP* ATTR RETURN

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

```
+-----+-----+-----+
|     |     |     |
|     |     |     |
| SELGI |
|     |
+-----+-----+-----+
```

- title (insensitive to light pen)
- threshold (imaginary line)
- light buttons
- DRAW
- ERASE
- ESCAPE
The elements of this display are arranged in the following structure:

\[
\text{highest active level} \\
\quad \text{SELGI display level} \\
\quad \quad \quad \text{line level} \\
\quad \quad \quad \text{draw level} \\
\quad \quad \quad \quad \quad \quad \text{draw leaf} \\
\quad \quad \text{erase level} \\
\quad \quad \quad \quad \quad \quad \text{erase leaf} \\
\quad \quad \text{escape level} \\
\quad \quad \quad \quad \quad \quad \text{escape leaf} \\
\quad \text{title leaf}
\]

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

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

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

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

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

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

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

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

```
first vector

(0,0)

third vector

second vector
```
SELG1
LAW 1 GET DISPLAY ALLOCATION MASK
JMS* =T.A ALLOCATE DISPLAY
LAC =LINES GET ADDRESS OF LINE STORAGE AREA
DAC DIS SET STORAGE POINTER
LAW -1000 LOAD AC WITH -512
DAC FRM SET STORAGE COUNTER
DZM* DIS CLEAR STORAGE LOCATION
ISZ DIS INCREMENT STORAGE POINTER
ISZ FRM SKIP IF STORAGE AREA CLEARED
JMP *-3 CLEAR NEXT STORAGE LOCATION
JMS* =S.LH GET ADDRESS OF HIGHEST ACTIVE LEVEL
DAC HAL SAVE ADDRESS OF HIGHEST ACTIVE LEVEL
LAC =TXT GET ADDRESS OF TITLE TEXT LIST
JMS* =L.D CREATE TEXT LEAF
JMP END DISPLAY STORAGE EXCEEDED
DAC DIS SAVE POINTER TO TITLE LEAF
JMS LVL GENERATE SELGI DISPLAY LEVEL
$DC HAL POINTER TO HIGHEST ACTIVE LEVEL
$DC 360 Y COORDINATE
$DC -34 X COORDINATE
$DC 500 SCALE X2
JMP FAIL DISPLAY STORAGE EXCEEDED
DAC FRM SAVE POINTER TO SELGI DISPLAY LEVEL
JMS BUTX GENERATE DRAW LIGHT BUTTON
$DC TXT1 DRAW TEXT LIST
$DC FRM POINTER TO SELGI DISPLAY LEVEL
$DC -750 Y COORDINATE
$DC -344 X COORDINATE
$DC 0 NULL DISPLAY PARAMETER
$DC DRAW DRAW SERVICE TASK
JMP END DISPLAY STORAGE EXCEEDED
JMS BUTX GENERATE ERASE LIGHT BUTTON
$DC TXT2 ERASE TEXT LIST
$DC FRM POINTER TO SELGI DISPLAY LEVEL
$DC -750 Y COORDINATE
$DC 10 X COORDINATE
$DC 0 NULL DISPLAY PARAMETER
$DC ERASE ERASE SERVICE TASK
JMP END DISPLAY STORAGE EXCEEDED
JMS BUTX GENERATE ESCAPE LIGHT BUTTON
$DC TXT3 ESCAPE TEXT LIST
$DC FRM POINTER TO SELGI 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 DIS SET UP PARAMETER
LAC FRM GET POINTER TO SELGI DISPLAY LEVEL
JMS* =S.TI INSERT LINE LEVEL
JMP FAIL DISPLAY STORAGE EXCEEDED
LAW 60 GET LIGHT PEN ON PARAMETER
LMQ DIS SET UP PARAMETER
LAC DIS GET POINTER TO LINE LEVEL
JMS* =S.LP ENABLE LIGHT PEN ON LINE LEVEL
LAW -360 GET Y COORDINATE
LMQ DIS 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 DIS SET UP PARAMETER
LAC DIS GET POINTER TO LINE LEVEL
JMS* =S.LX SET X COORDINATE OF LINE LEVEL
LAC =MOVE GET ADDRESS OF LINE MOVING TASK
JMS* =D.P SET LIGHT PEN FLAG SERVICE
JMS* =D.E ENABLE DISPLAY INTERRUPTS
DZM ESCAPE+1 CLEAR ESCAPE SWITCH
LAC    ESCAPE+1    GET ESCAPE SWITCH
SZA    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
LAW 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 MOVEX 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
DAC MOVE4
LAC* MOVE3
JMS* =C.CB
LLSS 1
TAD MOVEY
GSM
CMA
TAD MOVE4
SMA
JMP *+3
JMS
JMS
JMS
$DC 0
LAC MOVE1
XCT* WATCH
JMS* =X.S
JMP *+6
LAW -40
JMS* =N.C
JMS* =D.E ENABLE DISPLAY INTERRUPTS
JMS* =T.F TERMINATE
JMP WATCH+1 UPDATE END POINT
JMS* =T.P SCHEDULE PREVIOUS LOCATION
ERASE
LAC =DELETE GET ADDRESS OF LINE DELETE TASK
JMS* =D.P SET LIGHT PEN FLAG SERVICE
CLA PREPARE TO READ OWNER 0 LEVELS BACK
JMS* =D.O GET POINTER TO ERASE LEVEL
$DC 0 PROGRAMMING ERROR IF D.O FAILS
DAC ERS SAVE POINTER TO ERASE LEVEL
JMS* =S.LBE START BLINKING ERASE LIGHT BUTTON
JMS* =D.E ENABLE DISPLAY INTERRUPTS
JMS* =T.F TERMINATE
DELETE
LAC ERS GET POINTER TO ERASE LEVEL
JMS* =S.LBD STOP BLINKING ERASE LIGHT BUTTON
CLA PREPARE TO READ OWNER 0 LEVELS BACK
JMS* =D.O GET POINTER TO LINE LEAF
$DC 0 PROGRAMMING ERROR IF D.O FAILS
DAC FRM SAVE POINTER TO LINE LEAF
LMQ SET UP PARAMETER
LAC DIS GET POINTER TO LINE LEVEL
JMS* =S.TR REMOVE LINE LEAF
$DC 0 PROGRAMMING ERROR IF S.TR FAILS
DZM* FRM DESTROY LINE LEAF
LAC =MOVE GET ADDRESS OF LINE MOVING TASK
JMS* =D.P SET LIGHT PEN SERVICE
LAW -40 LOAD AC WITH -32
JMS* =N.C WAIT ABOUT HALF A SECOND
JMS* =D.E ENABLE DISPLAY INTERRUPTS
JMS* =T.F TERMINATE
ESCAPE
JMS *+1 SET ESCAPE SWITCH
$DC 0 ESCAPE SWITCH
JMS* =T.F TERMINATE
FIXBGN
$DC 0
JMS FIXRD SET UP POINTERS FOR FIXING LEAF
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAC FIXY</td>
<td>GET Y TRACKING COORDINATE</td>
</tr>
<tr>
<td>DAC* FIX1</td>
<td>SET FIRST Y DISPLACEMENT</td>
</tr>
<tr>
<td>LAC FIXX</td>
<td>GET X TRACKING COORDINATE</td>
</tr>
<tr>
<td>DAC* FIX2</td>
<td>SET FIRST X DISPLACEMENT</td>
</tr>
<tr>
<td>JMS FIXFIX</td>
<td>CORRECT INTENSIFIED VECTOR</td>
</tr>
<tr>
<td>JMP* FIXBGN</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

**FIXEND**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS FIXRD</td>
<td>SET UP POINTERS FOR FIXING LEAF</td>
</tr>
<tr>
<td>LAC FIXY</td>
<td>GET Y TRACKING COORDINATE</td>
</tr>
<tr>
<td>XOR =2000</td>
<td>INVERT SIGN BIT</td>
</tr>
<tr>
<td>DAC* FIX5</td>
<td>SET THIRD Y DISPLACEMENT</td>
</tr>
<tr>
<td>LAC FIXX</td>
<td>GET X TRACKING COORDINATE</td>
</tr>
<tr>
<td>XOR =6000</td>
<td>INVERT SIGN BIT, SET ESCAPE BIT</td>
</tr>
<tr>
<td>DAC* FIX6</td>
<td>SET THIRD X DISPLACEMENT</td>
</tr>
<tr>
<td>JMS FIXFIX</td>
<td>CORRECT INTENSIFIED VECTOR</td>
</tr>
<tr>
<td>JMP* FIXEND</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

**FIXRD**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$DC 0</td>
<td></td>
</tr>
<tr>
<td>TAD =1</td>
<td>FORM POINTER TO FIRST Y DISPLACEMENT</td>
</tr>
<tr>
<td>DAC FIX1</td>
<td>SAVE</td>
</tr>
<tr>
<td>TAD =1</td>
<td>FORM POINTER TO FIRST X DISPLACEMENT</td>
</tr>
<tr>
<td>DAC FIX2</td>
<td>SAVE</td>
</tr>
<tr>
<td>TAD =1</td>
<td>FORM POINTER TO SECOND Y DISPLACEMENT</td>
</tr>
<tr>
<td>DAC FIX3</td>
<td>SAVE</td>
</tr>
<tr>
<td>TAD =1</td>
<td>FORM POINTER TO SECOND X DISPLACEMENT</td>
</tr>
<tr>
<td>DAC FIX4</td>
<td>SAVE</td>
</tr>
<tr>
<td>TAD =1</td>
<td>FORM POINTER TO THIRD Y DISPLACEMENT</td>
</tr>
<tr>
<td>DAC FIX5</td>
<td>SAVE</td>
</tr>
<tr>
<td>TAD =1</td>
<td>FORM POINTER TO THIRD X DISPLACEMENT</td>
</tr>
<tr>
<td>DAC FIX6</td>
<td>SAVE</td>
</tr>
<tr>
<td>JMS* =X.Y</td>
<td>READ Y TRACKING COORDINATE</td>
</tr>
<tr>
<td>LRSS 1</td>
<td>DIVIDE BY 2</td>
</tr>
<tr>
<td>JMS* =C.BC</td>
<td>CONVERT TO DISPLAY COORDINATE</td>
</tr>
<tr>
<td>DAC FIXY</td>
<td>SAVE</td>
</tr>
<tr>
<td>JMS* =X.X</td>
<td>READ X TRACKING COORDINATE</td>
</tr>
<tr>
<td>LRSS 1</td>
<td>DIVIDE BY 2</td>
</tr>
</tbody>
</table>
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:

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

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

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

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 \textit{loading}, a copy from core to teletype or from core to display is called a \textit{dump}, a copy from teletype to core is called \textit{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 TELETYPETO CORE**

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

**FROM TELETYPETO CORE**

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

When a copy from paper tape to any device other than core is specified, the next alphanumeric record (Section 3.4.2) is read, and all binary records which are encountered before it
are ignored. (However, if the alphanumerical record is too long for the display, and a copy from paper tape to display is specified, only part of the alphanumerical 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 alphanumerical 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 23571\textsubscript{8} through 23602\textsubscript{8} on the teletype as follows:

\texttt{FROM\ CORE\ TO\ TELETYP\E}
\texttt{BLOCK \ (23571, \ 23602)}
\texttt{23571 \ 172356 \ 543125 \ 601241 \ 760001 \ 127123 \ 127124 \ 000200 \ 000001}
\texttt{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\textsubscript{8} through 20056\textsubscript{8} may be moved into locations 21521\textsubscript{8} through 21525\textsubscript{8} by the following command:

\texttt{FROM\ CORE\ TO\ CORE}
\texttt{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.)

\texttt{FROM\ TELETYP\E\ TO\ CORE}
\texttt{20000 \ 172132 \ 000000}
\texttt{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{8} into location 20001\textsubscript{8}, then it moves the zero from location 20001\textsubscript{8} into location 20002\textsubscript{8}, 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{8} 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{8} may be scheduled by the following command:

\texttt{SCHEDULE 20571}

4.3 Clearing the Task Queue or Display Storage

The command

\texttt{CLEAR \{\texttt{TASK QUEUE}}

\texttt{\texttt{DISPLAY STORAGE}}

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

\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 \text{000}_{8} 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

\text{RUN}

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

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

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


APPENDIX A -- LISTING OF THE EXECUTIVE SYSTEM

$ORG 17732
$TITLE
IOT $OPDM 700000
HLT $OPD 740040

IOT 3302
JMP SYSTEM

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

*2

.3

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

.4

$DC 0
IOT 104
IOT 101

CLEAR ALL FLAGS
START SYSTEM
READ FIRST LINE OF 3-LINE BLOCK
SKIP IF NONBLANK TAPE
BLANK TAPE -- TRY AGAIN
SAVE FIRST LINE IMAGE
REMOVE DATA BITS
SKIP IF DATA LINE
ORIGIN LINE
DATA LINE
FINISH ORIGIN WORD
SET LOCATION COUNTER
READ NEXT BLOCK
FINISH DATA WORD
LOAD DATA WORD
INCREMENT LOCATION COUNTER
READ NEXT BLOCK
READ SECOND LINE
SHIFT DATA BITS INTO MQ
LOAD AC WITH FIRST LINE IMAGE
SHIFT CONCATENATED IMAGE INTO AC
SAVE CONCATENATED FIRST TWO LINE
READ THIRD LINE
SHIFT DATA BITS INTO MQ
LOAD AC WITH CONCATENATED IMAGE
SHIFT COMPLETED WORD INTO AC
RETURN
SELECT READER
SKIP IF LINE READY
JMP *-1
IOT 112
JMP* *4

.5 $DC 0
.6 $DC 0
.7 $DC 300
.8 $DC 100

JMP *1

WAIT FOR FLAG
OVERRIDDEN "JMP .1-2" WHEN LOADED
RETURN

OVERRIDES Bootstrap LOCATION 0
$TITLE  CONTROL DISPATCHER
SORG 1  INTERRUPT TRAP
JMP I

SORG 21  ILLEGAL INSTRUCTION TRAP
JMP ET
JMP ES  SYSTEM RESTART

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

SCHEDULE TASK

T.P  SDC 0
JMP TP

SCHEDULE PREVIOUS LOC & TERMINATE

T.F  SDC 0
JMP TF

TERMINATE CURRENT TASK

T.A  SDC 0
JMP TA

ALLOCATE I/O DEVICES UNDER MASK

T.R  SDC 0
JMP TR

RELEASE I/O DEVICES UNDER MASK

T.L  SDC 0
JMP TL

LOCK REENTRABLE SUBROUTINE

T.U  SDC 0
JMP TU

UNLOCK REENTRABLE SUBROUTINE

C.B6  SDC 0
JMP CB6

CONVERT BINARY TO 6-BIT OCTAL

C.6A  SDC 0
JMP C6A

CONVERT 6-BIT TO ASCII

C.A6  SDC 0
JMP CA6

CONVERT ASCII TO 6-BIT

C.CB  SDC 0
JMP CCB

CONVERT DISPLAY COORDINATE TO BINARY
| C.BC | SDC 0 | JMP CBC | CONVERT BINARY TO DISPLAY COORDINATE |
| B.FI | SDC 0 | JMP BFI | GET IMAGE FROM 201 INPUT BUFFER (F) |
| B.FO | SDC 0 | JMP BFO | SEND IMAGE TO 201 OUTPUT BUFFER (F) |
| B.R  | SDC 0 | JMP BR  | GET IMAGE FROM READER BUFFER (F)   |
| B.P  | SDC 0 | JMP BP  | SEND IMAGE TO PUNCH BUFFER (F)     |
| B.K  | SDC 0 | JMP BK  | GET 6-BIT CHAR FROM KEYBOARD BUFFER |
| B.T  | SDC 0 | JMP BT  | SEND 3 PACKED 6-BIT CHARs TO TP BUF |
| N.A  | SDC 0 | JMP NA  | CONVERT ANALOG TO DIGITAL           |
| N.C  | SDC 0 | JMP NC  | SET CLOCK INTERVAL & SERVICE TASK   |
| N.D1 | SDC 0 | JMP ND1 | SELECT D/A CONVERTER #1             |
| N.D2 | SDC 0 | JMP ND2 | SELECT D/A CONVERTER #2             |
| N.D3 | SDC 0 | JMP ND3 | SELECT D/A CONVERTER #3             |
| P.T  | SDC 0 | JMP PT  | SET PUSH BUTTON SERVICE TASK        |
| P.E  | SDC 0 | JMP PE  | ENABLE MANUAL OPN OF PUSH BUTTONS    |
| P.D  | SDC 0 | JMP PD  | DISABLE MANUAL OPN OF PUSH BUTTONS   |
| P.R  | SDC 0 | JMP PR  | READ PUSH BUTTONS                    |
| P.S  | SDC 0 | JMP PS  | SET PUSH BUTTONS                     |
| D.E  | SDC 0 | JMP DE  | ENABLE DISPLAY INTERRUPTS             |
| D.D  | SDC 0 | JMP DD  | DISABLE DISPLAY INTERRUPTS            |
| D.P  | SDC 0 | JMP DP  | SET LIGHT PEN FLAG SERVICE TASK      |
D.A  $DC 0
   JMP DA
READ DISPLAY ADR ON LAST INTERRUPT
D.Y  $DC 0
   JMP DY
READ Y DPY COORD ON LAST INTERRUPT
D.X  $DC 0
   JMP DX
READ X DPY COORD ON LAST INTERRUPT
D.O  $DC 0
   JMP DO
READ OWNER ON LAST INTERRUPT (F)
X.I  $DC 0
   JMP XI
INITIALIZE TRACKING AT GIVEN COORDS
X.R  $DC 0
   JMP XR
RESUME TRACKING
X.T  $DC 0
   JMP XT
TERMINATE TRACKING
X.S  $DC 0
   JMP XS
SKIP IF TRACKING NOT IN PROCESS (F)
X.Y  $DC 0
   JMP XY
READ Y TRACKING COORDINATE
X.X  $DC 0
   JMP XX
READ X TRACKING COORDINATE
S.TL $DC 0
   JMP STL
CREATE A LEVEL (F)
S.TD $DC 0
   JMP STD
DESTROY A LEVEL (F)
S.TI $DC 0
   JMP STI
INSERT SUBSTRUCTURE INTO LEVEL (F)
S.TR $DC 0
   JMP STR
REMOVE SUBSTRUCTURE FROM LEVEL (F)
S.LH $DC 0
   JMP SLH
GET ADDRESS OF HIGHEST ACTIVE LEVEL
S.LY $DC 0
   JMP SLY
TRANSLATE LEVEL IN Y DIRECTION
S.LX $DC 0
   JMP SLX
TRANSLATE LEVEL IN X DIRECTION
S.LP $DC 0
   JMP SLP
SET LEVEL PARAMETERS
S.LBE $DC 0
   JMP SLBE
ENABLE BLINK ON LEVEL
S.LBD $DC 0
   JMP SLBD
DISABLE BLINK ON LEVEL
S.LC   $DC 0
   JMP SLC
COUNT SCALE AND/OR INTENSITY

S.LU   $DC 0
   JMP SLU
INTERRUPT UNCONDITIONALLY ON LEVEL

S.LS   $DC 0
   JMP SLS
INTERRUPT ON LEVEL IF ON SCREEN

S.LL   $DC 0
   JMP SLL
INTERRUPT ON LEVEL IF LPSI SET

S.LN   $DC 0
   JMP SLN
DISABLE INTERRUPT ON LEVEL

L.T    $DC 0
   JMP LT
SEND TEXT LIST TO TP BUFFER

L.D    $DC 0
   JMP LD
GENERATE TEXT LEAF (F)

L.L    $DC 0
   JMP LL
DESTROY TEXT LEAF

PDP1   $DS 204
PDP2   $DS 204
DISPLAY CHARACTER GENERATOR

$TITLE

D00  INCR
$DC 1272
$DC 6251
$DC 6057
$DC 7516
$DC 1570
$DC 5172
$DC 3726
$DC 0
POP

D01  INCR
$DC 5160
$DC 1472
$DC 7255
$DC 3737
$DC 0
POP

D02  INCR
$DC 5271
$DC 5152
$DC 5364
$DC 5537
$DC 2774
$DC 5417
$DC 3020
$DC 0
POP

D03  INCR
$DC 1252
$DC 5760
$DC 5152
$DC 5354
$DC 1051
$DC 5253
$DC 6455
$DC 3737
$DC 1000
POP
D04  INCR  
  SDC  1110  
  SDC  5072  
  SDC  7275  
  SDC  6010  
  SDC  5937  
  SDC  1600  
  POP  

D05  INCR  
  SDC  1252  
  SDC  5760  
  SDC  5162  
  SDC  5374  
  SDC  6270  
  SDC  5937  
  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  

$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
$DC 3656
$DC 5564
$DC 3020
$DC 1700
POP
D15  INCR
$DC 7272
$DC 5260
$DC 6766
$DC 6554
$DC 3020
$DC 1700
POP

D16  INCR
$DC 7272
$DC 5273
$DC 5025
$DC 5550
$DC 2516
$DC 7050
$DC 1720
$DC 0
POP

D17  INCR
$DC 7272
$DC 5270
$DC 5025
$DC 5550
$DC 3717
$DC 1000
POP

D20  INCR
$DC 1272
$DC 6251
$DC 6057
$DC 3570
$DC 5655
$DC 6430
$DC 1720
$DC 0
D21
  INCR
  $DC 7272
  $DC 5236
  $DC 7050
  $DC 7236
  $DC 7620
  $DC 1700
  POP

D22
  INCR
  $DC 5160
  $DC 1472
  $DC 7254
  $DC 1050
  $DC 3736
  $DC 1700
  POP

D23
  INCR
  $DC 2252
  $DC 5657
  $DC 6051
  $DC 7262
  $DC 3636
  $DC 1720
  $DC 0
  POP

D24
  INCR
  $DC 7272
  $DC 5230
  $DC 5075
  $DC 7720
  $DC 1700
  POP

D25
  INCR
  $DC 7272
  $DC 5236
  $DC 3670
  $DC 5020
  $DC 1700
  POP
D26
INCR
$DC 7272
$DC 5267
$DC 6176
$DC 7620
$DC 1700
POP

D27
INCR
$DC 7272
$DC 5277
$DC 3250
$DC 7676
$DC 1720
$DC 0
POP

D30
INCR
$DC 1272
$DC 6251
$DC 6057
$DC 7656
$DC 5564
$DC 3020
$DC 1700
POP

D31
INCR
$DC 7272
$DC 5270
$DC 5756
$DC 5564
$DC 3720
$DC 1700
POP

D32
INCR
$DC 1272
$DC 6251
$DC 6057
$DC 7656
$DC 5564
$DC 1022
$DC 7720
$DC \ 0
POP
D33  INCR
$DC 7272
$DC 5270
$DC 5756
$DC 5564
$DC 7720
$DC 1700
POP
D34  INCR
$DC 1252
$DC 5760
$DC 5152
$DC 5364
$DC 5352
$DC 5160
$DC 5737
$DC 3600
POP
D35  INCR
$DC 1150
$DC 7272
$DC 6420
$DC 6036
$DC 1637
$DC 0
POP
D36  INCR
$DC 1272
$DC 7230
$DC 5076
$DC 6655
$DC 6430
$DC 1720
$DC 0
POP
D37  INCR
$DC 2272
$DC 6230
$DC 5076
$DC 5665
$DC 5327
$DC 3010
$DC 0
POP

D40 INCR
$DC 7272
$DC 5230
$DC 5076
$DC 7663
$DC 5527
$DC 3010
$DC 0
POP

D41 INCR
$DC 6271
$DC 5152
$DC 3454
$DC 5657
$DC 1767
$DC 5617
$DC 2000
POP

D42 INCR
$DC 1150
$DC 7261
$DC 5234
$DC 5456
$DC 5737
$DC 2710
$DC 0
POP

D43 INCR
$DC 6271
$DC 5152
$DC 7454
$DC 3727
$DC 5574
$DC 3020
$DC 1700
POP

D44  INCR
$DC 1251
$DC 7151
$DC 3454
$DC 5717
$DC 6727
$DC 0
POP

D45  INCR
$DC 1252
$DC 7151
$DC 3736
$DC 0
POP

D46  INCR
$DC 1151
$DC 7252
$DC 1555
$DC 5010
$DC 6037
$DC 1600
POP

D47  INCR
$DC 3252
$DC 7050
$DC 3716
$DC 0
POP

D50  INCR
$DC 1150
$DC 5372
$DC 5251
$DC 3727
$DC 2600
POP

D51  INCR
$DC 1150
$DC 5172
$DC 5253
$DC 3727
$DC 2600
POP

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

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

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

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

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

D57  INCR
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$DC $1,150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $7272</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $6520</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $5157</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $3726</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $0 $</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$POP $</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**D60 INCR**

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$DC $1,151</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $6361</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $1,655</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $7037</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $1,600</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$POP $</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**D61 INCR**

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$DC $6,250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $5,630</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $1,720</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $0 $</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$POP $</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**D62 INCR**

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$DC $6,250</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $6,655</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $1,130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $3,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$POP $</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**D63 INCR**

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$DC $1,262</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $1,262</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $5,056</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $1,666</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $2,710</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $3,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$POP $</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**D64 INCR**

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$DC $1,262</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $1,262</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $5,056</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $1,676</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$DC $5,530</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
$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
$TITLE

TRACKING PATTERN GENERATOR

XP LAW 3000
$DC 1105

XPY $DC 1000

XPX $DC 5000
$DC 1400
$DC X1
$DC 60
SVEC
$DC 24
$DC 4047
$DC 1400
$DC X2
$DC 60
SVEC
$DC 1
$DC 4067
$DC 1400
$DC X3
$DC 60
SVEC
$DC 2403
$DC 4740
$DC 1400
$DC X2
$DC 60
SVEC
$DC 100
$DC 6700
$DC 340
$DC 1400
$DC X4

XPS POP
SVEC
$DC 404
$DC 4030
$DC 7000
$DC 4010
$DC 5040
VEC
$DC$ 4
$DC$ 4
$DC$ 4000
$DC$ 2020
$DC$ 6020
$DC$ 0
$DC$ 4000
$DC \ 0$
$DC \ 4$
$DC \ 4$
$DC \ 4000$
$DC \ 2060$
$DC \ 6060$
$DC \ 0$
$DC \ 4000$
$DC \ 60$
$DC \ 4060$
$DC \ 0$
$DC \ 4$
$DC \ 4$
$DC \ 4000$
$DC \ 2070$
$DC \ 6070$
$DC \ 0$
$DC \ 4000$
$DC \ 70$
$DC \ 4070$
$DC \ 0$
$DC \ 4$
$DC \ 4$
$DC \ 4000$
$DC \ 2100$
$DC \ 6100$
$DC \ 0$
$DC \ 4000$
$DC \ 100$
$DC \ 4100$
$DC \ 4000$
$DC \ 1400$
$DC \ X5$
$POP$
STITLE

INTERRUPT DISPATCHER

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
JMP IR RETURN FROM INTERRUPT
IOT 642  SKIP ON LIGHT PEN FLAG
SKP      TEST NEXT FLAG
JMP ILP  SERVICE LIGHT PEN INTERRUPT
IOT 721  SKIP ON INTERNAL STOP FLAG
SKP      TEST NEXT FLAG
JMP IIS  SERVICE INTERNAL STOP INTERRUPT
IOT 722  SKIP ON MANUAL INTERRUPT FLAG
JMP E1   INVALID INTERRUPT
JMP EM   EMERGENCY REINITIALIZATION
IR
LAC 2    GET SC CONTENTS
XOR =77  COMPLEMENT SHIFT COUNT
TAD =640402 FORM NORM INSTRUCTION
AND =640477 TRUNCATE CARRY
DAC ++1  STORE NORM INSTRUCTION
HLT      RESTORE SC CONTENTS
LAC 3    GET MQ CONTENTS
LMQ      RESTORE MQ CONTENTS
LAC 6    RESTORE AC CONTENTS
IOT 42   ENABLE INTERRUPTS
IOT 3344 DEBREAK AND RESTORE
JMP* 0   RETURN TO INTERRUPTED PROGRAM
$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 705   LOAD BREAK FIELD
LAW =1400  GET ADDRESS OF INTERNAL STOP
IOT 1605  INITIALIZE DISPLAY
LAC =**2   GET ADDRESS OF TEXT LIST
JMP E     REINITIALIZE SYSTEM
$DC 5
$TEXT "PANEL RECOVERY"

ET
IOT 2     DISABLE INTERRUPTS
CLC       LOAD AC WITH -1
TAD 20    ADD PROGRAM COUNTER DURING TRAP
JMS C.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
<table>
<thead>
<tr>
<th>STITLE</th>
<th>SYSTEM INITIALIZER</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC 25</td>
<td>SAVE ADDRESS OF DIAGNOSTIC</td>
</tr>
<tr>
<td>IOT 7702</td>
<td>ENTER EXTEND MODE</td>
</tr>
<tr>
<td>IOT 1412</td>
<td>READ 201 STATUS</td>
</tr>
<tr>
<td>AND =1</td>
<td>GET RECEIVE STATE BIT</td>
</tr>
<tr>
<td>SZA</td>
<td>SKIP IF NOT RECEIVING</td>
</tr>
<tr>
<td>JMP *-3</td>
<td>WAIT FOR END OF RECORD</td>
</tr>
<tr>
<td>IOT 1444</td>
<td>CLEAR 201 INTERFACE</td>
</tr>
<tr>
<td>LAC =440</td>
<td>GET TERM RDY BIT &amp; FRAME SIZE 8</td>
</tr>
<tr>
<td>IOT 1404</td>
<td>SET INITIAL 201 INTERFACE STATE</td>
</tr>
<tr>
<td>LAC BP3</td>
<td>GET PUNCH STATUS SWITCH</td>
</tr>
<tr>
<td>SNA</td>
<td>SKIP IF PUNCH ACTIVE</td>
</tr>
<tr>
<td>JMP ++3</td>
<td>PUNCH NOT ACTIVE</td>
</tr>
<tr>
<td>IOT 201</td>
<td>SKIP ON PUNCH FLAG</td>
</tr>
<tr>
<td>JMP *-1</td>
<td>WAIT FOR PUNCH FLAG</td>
</tr>
<tr>
<td>LAC BT1</td>
<td>GET TELEPRINTER STATUS SWITCH</td>
</tr>
<tr>
<td>SNA</td>
<td>SKIP IF TELEPRINTER ACTIVE</td>
</tr>
<tr>
<td>JMP ++3</td>
<td>TELEPRINTER NOT ACTIVE</td>
</tr>
<tr>
<td>IOT 401</td>
<td>SKIP ON TELEPRINTER FLAG</td>
</tr>
<tr>
<td>JMP *-1</td>
<td>WAIT FOR TELEPRINTER FLAG</td>
</tr>
<tr>
<td>IOT 612</td>
<td>READ DISPLAY STATUS</td>
</tr>
<tr>
<td>AND =7400</td>
<td>GET DISPLAY FLAG BITS</td>
</tr>
<tr>
<td>SNA+CLA</td>
<td>SKIP IF DISPLAY STOPPED</td>
</tr>
<tr>
<td>JMP *-3</td>
<td>WAIT FOR DISPLAY TO STOP</td>
</tr>
<tr>
<td>JMS PS1</td>
<td>CLEAR PUSH BUTTONS</td>
</tr>
<tr>
<td>IOT 4</td>
<td>DISABLE CLOCK</td>
</tr>
<tr>
<td>IOT 3302</td>
<td>CLEAR ALL FLAGS</td>
</tr>
<tr>
<td>DZM BP3</td>
<td>INDICATE PUNCH IDLE</td>
</tr>
<tr>
<td>DZM BT1</td>
<td>INDICATE TELEPRINTER IDLE</td>
</tr>
<tr>
<td>DZM PE+1</td>
<td>DISABLE OPERATION OF PUNCH BUTTON</td>
</tr>
<tr>
<td>DZM DE+1</td>
<td>DISABLE DISPLAY INTERRUPTS</td>
</tr>
<tr>
<td>DZM DWV</td>
<td>CLEAR TRANSLATION VALUE</td>
</tr>
<tr>
<td>DZM NA+2</td>
<td>UNLOCK N.A</td>
</tr>
<tr>
<td>DZM NC+2</td>
<td>UNLOCK N.C</td>
</tr>
<tr>
<td>DZM STRD+2</td>
<td>UNLOCK S.TRD</td>
</tr>
<tr>
<td>DZM STRR+2</td>
<td>UNLOCK S.TRR</td>
</tr>
<tr>
<td>DZM SLY+2</td>
<td>UNLOCK S.LY</td>
</tr>
<tr>
<td>DZM SLX+2</td>
<td>UNLOCK S.LX</td>
</tr>
<tr>
<td>LAW 10</td>
<td>GET TELEPRINTER MASK</td>
</tr>
<tr>
<td>Instruction</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>DAC STATUS</td>
<td>ALLOCATE TELEPRINTER ONLY</td>
</tr>
<tr>
<td>DAC BFTTY2</td>
<td>SET BFTTY ALLOCATION MASK</td>
</tr>
<tr>
<td>LAC TQ</td>
<td>GET POINTER TO END OF TASK QUEUE</td>
</tr>
<tr>
<td>DAC TQ+1</td>
<td>RESET INPUT POINTER</td>
</tr>
<tr>
<td>DAC TQ+2</td>
<td>RESET OUTPUT POINTER</td>
</tr>
<tr>
<td>DAC BRS</td>
<td>SET RECORD SEEK SWITCH</td>
</tr>
<tr>
<td>DZM BRO</td>
<td>INDICATE NEW RECORD NEEDED</td>
</tr>
<tr>
<td>LAC BPQ</td>
<td>GET POINTER TO END OF PUNCH BUFFER</td>
</tr>
<tr>
<td>DAC BPQ+1</td>
<td>RESET INPUT POINTER</td>
</tr>
<tr>
<td>DAC BPQ+2</td>
<td>RESET OUTPUT POINTER</td>
</tr>
<tr>
<td>LAC BKQ</td>
<td>GET POINTER TO END OF KB BUFFER</td>
</tr>
<tr>
<td>DAC BKQ+1</td>
<td>RESET INPUT POINTER</td>
</tr>
<tr>
<td>DAC BKQ+2</td>
<td>RESET OUTPUT POINTER</td>
</tr>
<tr>
<td>LAC BTQ</td>
<td>GET POINTER TO END OF TP BUFFER</td>
</tr>
<tr>
<td>DAC BTQ+1</td>
<td>RESET INPUT POINTER</td>
</tr>
<tr>
<td>DAC BTQ+2</td>
<td>RESET OUTPUT POINTER</td>
</tr>
<tr>
<td>LAC =DN</td>
<td>GET ADDRESS OF NULL DISPLAY SERVICE</td>
</tr>
<tr>
<td>DAC DPT</td>
<td>SET NULL LIGHT PEN SERVICE</td>
</tr>
<tr>
<td>LAC =PN</td>
<td>GET ADDRESS OF NULL PB SERVICE</td>
</tr>
<tr>
<td>DAC PTT</td>
<td>SET NULL PUSH BUTTON SERVICE</td>
</tr>
<tr>
<td>LAW 3000</td>
<td>GET POP INSTRUCTION</td>
</tr>
<tr>
<td>DAC XP</td>
<td>INHIBIT TRACKING PROCESS</td>
</tr>
<tr>
<td>LAC =D</td>
<td>GET ADDRESS OF HIGHEST ACTIVE LEVEL</td>
</tr>
<tr>
<td>DAC DHAL+7</td>
<td>REMOVE EVERYTHING FROM HAL</td>
</tr>
<tr>
<td>LAW PDP1</td>
<td>GET ADDRESS OF PUSH DOWN LIST</td>
</tr>
<tr>
<td>IOT 645</td>
<td>SET PUSH DOWN POINTER</td>
</tr>
<tr>
<td>LAW 7763</td>
<td>GET INITIAL DISPLAY CONDITIONS</td>
</tr>
<tr>
<td>IOT 665</td>
<td>SET INITIAL DISPLAY CONDITIONS</td>
</tr>
<tr>
<td>LAW 4400</td>
<td>GET BREAK FIELD 1 PARAMETER</td>
</tr>
<tr>
<td>IOT 705</td>
<td>LOAD BREAK FIELD</td>
</tr>
<tr>
<td>LAW D</td>
<td>GET ADDRESS OF SYSTEM DISPLAY FILE</td>
</tr>
<tr>
<td>IOT 1605</td>
<td>START DISPLAY</td>
</tr>
<tr>
<td>LAC =BFENQ</td>
<td>GET ENQUIRY CHARACTER</td>
</tr>
<tr>
<td>JMS BFENQS</td>
<td>INITIALIZE 201 TASKS</td>
</tr>
<tr>
<td>IOT 42</td>
<td>ENABLE INTERRUPTS</td>
</tr>
<tr>
<td>LAW BFENQ</td>
<td>GET ENQUIRY CHARACTER</td>
</tr>
<tr>
<td>JMS B•FO</td>
<td>SEND ATTENTION INTERRUPT</td>
</tr>
<tr>
<td>NOP</td>
<td>DATA SET NOT CONNECTED</td>
</tr>
<tr>
<td>E1</td>
<td>CLEAR POINTER TO DIAGNOSTIC LEVEL</td>
</tr>
<tr>
<td>DZM 26</td>
<td>CLEAR POINTER TO DIAGNOSTIC LEAF</td>
</tr>
<tr>
<td>DZM 27</td>
<td>CLEAR POINTER TO DIAGNOSTIC LEAF</td>
</tr>
</tbody>
</table>
JMS S.TL
JMP E2
DAC E3
LAC =EF
JMS L.D
JMP E2
LMQ
LAC E3
JMS S.TI
JMP E2
LAC =370
LMQ
LAC E3
JMS S.LY
LAW -144
LMQ
LAC E3
JMS S.LX
LAW 500
LMQ
LAC E3
JMS S.LP
LMQ
LAC E3
JMS S.TI
LAC =DHAL
JMS S.TL
JMP E2
DAC 26
LAC =200
LMQ
LAC 26
JMS S.LY
LAW -400
LMQ
LAC 26
JMS S.LX
LAW 500
LMQ
CREATE TITLE LEAF
USE TELETYPewriter ONLY
SAVE POINTER TO TITLE LEAF
GET ADDRESS OF TEXT LIST
CREATE TITLE LEAF
USE TELETYPewriter ONLY
GET POINTER TO TITLE LEVEL
INSERT TITLE LEAF
USE TELETYPewriter ONLY
GET Y TITLE COORDINATE
SET UP PARAMETER
GET POINTER TO TITLE LEVEL
SET Y TITLE COORDINATE
GET X TITLE COORDINATE
SET UP PARAMETER
GET POINTER TO TITLE LEVEL
SET X TITLE COORDINATE
GET SCALE X2 PARAMETER
SET UP PARAMETER
GET POINTER TO TITLE LEVEL
SET TITLE SCALE
GET POINTER TO TITLE LEVEL
SET UP PARAMETER
GET ADDRESS OF HIGHEST ACTIVE LEVEL
INSERT TITLE LEVEL
USE TELETYPewriter ONLY
CREATE DIAGNOSTIC LEVEL
USE TELETYPewriter ONLY
SET POINTER TO DIAGNOSTIC LEVEL
GET Y DIAGNOSTIC DISPLACEMENT
SET UP PARAMETER
GET ADDRESS OF DIAGNOSTIC LEVEL
TRANSLATE LEVEL IN Y DIRECTION
GET X DIAGNOSTIC DISPLACEMENT
SET UP PARAMETER
GET ADDRESS OF DIAGNOSTIC LEVEL
TRANSLATE LEVEL IN X DIRECTION
GET SCALE X2 PARAMETER
SET UP PARAMETER
LAC  26       GET ADDRESS OF DIAGNOSTIC LEVEL
JMS S•LP     SET DIAGNOSTIC SCALE
LAC  26       GET ADDRESS OF DIAGNOSTIC LEVEL
LMQ          SET UP PARAMETER
LAC =DHAL    GET ADDRESS OF HIGHEST ACTIVE LEVEL
JMS S•TI     INSERT DIAGNOSTIC LEVEL
JMP E2       DISPLAY STORAGE EXCEEDED
LAC  25       GET ADDRESS OF TEXT LIST
SZA          SKIP IF DISP STORAGE BEING CLEARED
JMS L•D      CREATE DIAGNOSTIC LEAF
JMP E2       USE TELETYPE ONLY
DAC  27       SET POINTER TO DIAGNOSTIC LEAF
LMQ          SET UP PARAMETER
LAC  26       GET ADDRESS OF DIAGNOSTIC LEVEL
JMS S•TI     INSERT DIAGNOSTIC LEAF
NOP          USE TELETYPE ONLY
E2
LAC  25       GET POINTER TO TEXT LIST
SNA          SKIP IF COMMENT TO BE TYPED
JMP IDLE     BEGIN IDLE-TIME TASK
LAC =747575   GET TELEPRINTER POSITIONING CODE
JMS B•T      POSITION TELEPRINTER
LAC  25       GET ADDRESS OF TEXT LIST
JMS L•T      TYPE DIAGNOSTIC
LAC =747575   GET TELEPRINTER POSITIONING CODE
JMS B•T      POSITION TELEPRINTER
JMP IDLE     BEGIN IDLE-TIME TASK

EF
SDC  11
$TEXT "SEL EXECUTIVE SYSTEM (01)"
$TITLE DISPLAY STRUCTURE STORAGE MANAGER

STORE $SEGU 12000
LOWER LIMIT OF DISPLAY STORAGE

B
$DC 0
CMA
FORM 1'S COMP OF NUMBER OF BLOCKS
TAD =1
FORM 2'S COMP OF NUMBER OF BLOCKS
DAC T1
INITIALIZE COUNTER
DAC T2
STORE VALUE FOR/resetting COUNTER
LAC =STORE
GET LOWER LIMIT OF DISPLAY STORAGE

B1
DAC T3
SET POINTER TO CANDIDATE
DAC T4
SET POINTER TO NEW BLOCK
SAD =20000
SKIP IF STORAGE NOT EXCEEDED
JMP* B
NOT ENOUGH FREE STORAGE
LAC* T4
GET FIRST WORD FROM BLOCK
SNA
SKIP IF BLOCK NOT AVAILABLE
JMP B2
ADD BLOCK TO CANDIDATE
LAC T2
GET INITIAL VALUE OF COUNTER
DAC T1
REINITIALIZE COUNTER
LAC T4
GET ADDRESS OF UNAVAILABLE BLOCK
TAD =4
FORM ADDRESS OF NEXT BLOCK
JMP B1
PROCEED WITH NEXT CANDIDATE

B2
ISZ T1
INCREMENT COUNTER & SKIP IF DONE
JMP **+4
PREPARE TO ADD ANOTHER BLOCK
LAC T3
GET ADDRESS OF ACQUIRED STORAGE
ISZ B
INDICATE SUCCESS
JMP* B
RETURN
LAC T4
GET ADDRESS OF BLOCK JUST ADDED
TAD =4
FORM ADDRESS OF NEXT BLOCK
JMP B1+1
ADD ANOTHER BLOCK

B3
$DC 0
LAC =1
GET SINGLE BLOCK PARAMETER
JMS B
FIND SINGLE BLOCK
JMP* B3
NO SINGLE BLOCK AVAILABLE
ISZ B3
INDICATE SUCCESS
JMP* B3
RETURN

B4
$DC 0
LAC =2
GET DOUBLE BLOCK PARAMETER
JMS B
JMP* B4
ISZ B4
JMP* B4

FIND DOUBLE BLOCK
NO DOUBLE BLOCK AVAILABLE
INDICATE SUCCESS
RETURN
$TITLE 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
     LAC0            GET VALUE TO BE STORED
     DAC* 23         STORE VALUE IN QUEUE
     ISZ Q.I         INDICATE SUCCESS
     IOT 42         ENABLE INTERRUPTS
     JMP* Q.I       RETURN

QF  IOT 2           DISABLE INTERRUPTS
     JMS QF1         FETCH WORD FROM QUEUE
     SKP             QUEUE EMPTY
     ISZ Q.F         INDICATE SUCCESS
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOT 42</td>
<td>ENABLE INTERRUPTS</td>
</tr>
<tr>
<td>JMP* Q.F</td>
<td>RETURN</td>
</tr>
<tr>
<td>QA1</td>
<td>SET CONTROL POINTERS</td>
</tr>
<tr>
<td>$DC 0</td>
<td></td>
</tr>
<tr>
<td>JMS QS</td>
<td>SET CONTROL POINTERS</td>
</tr>
<tr>
<td>LAC* QIP</td>
<td>GET INPUT POINTER</td>
</tr>
<tr>
<td>JMS QINC</td>
<td>INCREMENT</td>
</tr>
<tr>
<td>SAD* QOP</td>
<td>SKIP IF NO OVERFLOW</td>
</tr>
<tr>
<td>JMP* QA1</td>
<td>OVERFLOW</td>
</tr>
<tr>
<td>DAC* QIP</td>
<td>SET NEW INPUT POINTER</td>
</tr>
<tr>
<td>DAC 23</td>
<td>SAVE COPY OF POINTER</td>
</tr>
<tr>
<td>LACQ</td>
<td>GET WORD TO BE STORED</td>
</tr>
<tr>
<td>DAC* 23</td>
<td>STORE WORD IN QUEUE</td>
</tr>
<tr>
<td>ISZ QA1</td>
<td>INDICATE SUCCESS</td>
</tr>
<tr>
<td>JMP* QA1</td>
<td>RETURN</td>
</tr>
<tr>
<td>QF1</td>
<td>SET CONTROL POINTERS</td>
</tr>
<tr>
<td>$DC 0</td>
<td></td>
</tr>
<tr>
<td>JMS QS</td>
<td>SET CONTROL POINTERS</td>
</tr>
<tr>
<td>LAC* QOP</td>
<td>GET OUTPUT POINTER</td>
</tr>
<tr>
<td>SAD* QIP</td>
<td>SKIP IF QUEUE NOT EMPTY</td>
</tr>
<tr>
<td>JMP* QF1</td>
<td>QUEUE EMPTY</td>
</tr>
<tr>
<td>JMS QINC</td>
<td>INCREMENT</td>
</tr>
<tr>
<td>DAC* QOP</td>
<td>SET NEW OUTPUT POINTER</td>
</tr>
<tr>
<td>DAC 23</td>
<td>SAVE COPY OF POINTER</td>
</tr>
<tr>
<td>LAC* 23</td>
<td>GET WORD FROM QUEUE</td>
</tr>
<tr>
<td>ISZ QF1</td>
<td>INDICATE SUCCESS</td>
</tr>
<tr>
<td>JMP* QF1</td>
<td>RETURN</td>
</tr>
<tr>
<td>QS</td>
<td>SET POINTER TO QUEUE</td>
</tr>
<tr>
<td>$DC 0</td>
<td></td>
</tr>
<tr>
<td>DAC QP</td>
<td>SET POINTER TO QUEUE</td>
</tr>
<tr>
<td>TAD =1</td>
<td>COMPUTE ADDRESS OF NEXT LOCAT</td>
</tr>
<tr>
<td>DAC QIP</td>
<td>SET POINTER TO INPUT POINTER</td>
</tr>
<tr>
<td>TAD =1</td>
<td>COMPUTE ADDRESS OF NEXT LOCATION</td>
</tr>
<tr>
<td>DAC QOP</td>
<td>SET POINTER TO OUTPUT POINTER</td>
</tr>
<tr>
<td>JMP* QS</td>
<td>RETURN</td>
</tr>
<tr>
<td>QINC</td>
<td>SKIP IF NOT END OF QUEUE</td>
</tr>
<tr>
<td>$DC 0</td>
<td></td>
</tr>
<tr>
<td>SAD* QP</td>
<td>Wrap around to beginning of queue</td>
</tr>
<tr>
<td>LAC QQP</td>
<td></td>
</tr>
<tr>
<td>TAD =1</td>
<td>INCREMENT</td>
</tr>
</tbody>
</table>
JMP* QINC

RETURN
$TITLE

TASK SCHEDULER

TS
IOT 2
AND =77777
JMS TII
IOT 42
JMP* T.S
DISABLE INTERRUPTS
TRUNCATE HIGH ORDER BITS
PUT TASK ADDRESS ON QUEUE
ENABLE INTERRUPTS
RETURN

TP
LAW 17776
TAD T.P
JMS T.S
LOAD AC WITH -2
FORM ADDRESS OF NEW TASK
SCHEDULE NEW TASK

TF
IOT 2
JMS TIO
DAC 23
RAL
JMP TF1
JMP TF2
DISABLE INTERRUPTS
READ WORD FROM TASK QUEUE
SAVE TASK ADDRESS
SHIFT TYPE BITS INTO LINK & SIGN
SKIP IF NOT REENTRY DELAY
RESTORE MQ & AC AND EXECUTE
SKIP IF NOT ALLOCATION DELAY
CHECK ELIGIBILITY
ENABLE INTERRUPTS
EXECUTE TASK

TF1
JMS TIO
LMQ
JMS TIO
JMP TF1-2
READ WORD FROM TASK QUEUE
RESTORE MQ
EXECUTE TASK

TF2
JMS TIO
AND STATUS
SNA
JMP TF3
READ WORD FROM TASK QUEUE
SKIP IF TASK NOT ELIGIBLE
FORM ELIGIBILITY CHECK
MODIFY STATUS & EXECUTE

TF3
LAC* TQ+2
JMS TII
IOT 42
LAC* TQ+2
JMP TF
GET Allocation MASK
PUT BACK ON TASK QUEUE
ENABLE INTERRUPTS
GET ANOTHER TASK

XOR STATUS
DAC STATUS
JMP TF1-2
STORE NEW STATUS WORD
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 | SAVE AC CONTENTS  |
|    | LAC* T.U | GET ADDRESS OF SUBROUTINE |
|    | TAD =2   | FORM ADDRESS OF SAVED RETURN |
|    | DAC T2   | SAVE ADDRESS OF SAVED RETURN |
|    | LAC* T2  | GET SAVED RETURN |
|    | DAC T3   | SAVE TEMPORARILY |
|    | DZM* T2  | UNLOCK SUBROUTINE |
|    | LAC T1   | RESTORE AC CONTENTS |
|    | JMP* T3  | RETURN FROM SUBROUTINE |

| TV | $DC 0   | SAVE AC CONTENTS  |
|    | DAC T1  | GET POINTER TO SUBROUTINE |
|    | LAC* TV | SAVE POINTER TO SUBROUTINE |
|    | DAC T2  | FORM POINTER TO SAVED RETURN |
|    | ISZ TV  | GET POINTER TO SAVED RETURN |
|    | LAC TV  | SIMULATE CALL TO T.L |
|    | DAC T.L | FAKE AN ENTRY TO T.L |

| TIO | $DC 0 | GET OUTPUT POINTER |
|     | LAC TQ+2 | SKIP IF TASK QUEUE NOT EMPTY |
|     | SAD TQ+1 | TASK QUEUE EMPTY |
|     | JMP EE   | INCREMENT |
|     | JMS T1   | STORE NEW OUTPUT POINTER |
|     | DAC TQ+2 | GET WORD FROM TASK QUEUE |
|     | LAC* TQ+2 | RETURN |

| TII | $DC 0 | SAVE VALUE TO BE STORED |
|     | DAC 24 | GET INPUT POINTER |
|     | LAC TQ+1 | INCREMENT |
|     | JMS T1   | STORE NEW INPUT POINTER |
|     | DAC TQ+1 | TASK QUEUE OVERFLOW |
|     | SAD TQ+2 | GET VALUE TO BE STORED |
|     | JMP EQ   | PUT IN TASK QUEUE |
|     | LAC 24   | RETURN |

<p>| TI  | $DC 0 |</p>
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAD TQ</td>
<td>SKIP IF NO WRAP-AROUND</td>
</tr>
<tr>
<td>LAC = TQ+2</td>
<td>GET ADDRESS BEFORE FIRST DATA WORD</td>
</tr>
<tr>
<td>TAD = 1</td>
<td>INCREMENT POINTER</td>
</tr>
<tr>
<td>JMP* TI</td>
<td>RETURN</td>
</tr>
<tr>
<td>TQ</td>
<td></td>
</tr>
<tr>
<td>$DC *+200</td>
<td></td>
</tr>
<tr>
<td>$DS 200</td>
<td></td>
</tr>
</tbody>
</table>
$TITLE FORMAT CONVERTER

CB6
CLL
LRS 14
ALS 3
LRS 6
ALS 3
LLS 11
DAC T1
LLS 6
ALS 3
LRS 6
ALS 3
AND $77
LRS 11
LAC T1
JMP* C.B6

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

C6A
AND $77
TAD =C6A1
DAC T1
LAC* T1
JMP* C.6A

TRUNCATE HIGH ORDER BITS
ADD ADDRESS OF TABLE
SAVE TEMPORARILY
GET CONVERTED VALUE
RETURN

C6A1
$DC 260
$DC 261
$DC 262
$DC 263
$DC 264
$DC 265
$DC 266
$DC 267
$DC 270
$DC 271
$DC 301
$DC 302
$DC 303
$DC 304
$DC 305
$DC 306
$DC 242
$DC 244
$DC 243
$DC 246
$DC 215
$DC 212
$DC 240
$DC 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  $DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 75
$DC 77
$DC 77
$DC 77
$DC 74
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
```
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77
$DC 77

<table>
<thead>
<tr>
<th>CCB</th>
<th>DAC T1</th>
<th>SAVE VALUE TO BE CONVERTED</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AND =1777</td>
<td>GET MAGNITUDE</td>
</tr>
<tr>
<td></td>
<td>DAC T2</td>
<td>SAVE MAGNITUDE</td>
</tr>
<tr>
<td></td>
<td>LAC T1</td>
<td>GET VALUE TO BE CONVERTED</td>
</tr>
<tr>
<td></td>
<td>AND =2000</td>
<td>GET SIGN BIT</td>
</tr>
<tr>
<td></td>
<td>SNA</td>
<td>SKIP IF NEGATIVE</td>
</tr>
<tr>
<td></td>
<td>JMP CCB1</td>
<td>DO NOT MODIFY MAGNITUDE</td>
</tr>
<tr>
<td></td>
<td>LAC T2</td>
<td>GET MAGNITUDE</td>
</tr>
<tr>
<td></td>
<td>CMA</td>
<td>FORM 1'S COMPLEMENT</td>
</tr>
<tr>
<td></td>
<td>TAD =1</td>
<td>FORM 2'S COMPLEMENT</td>
</tr>
<tr>
<td></td>
<td>JMP* C.CB</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CCB1</th>
<th>LAC T2</th>
<th>GET CONVERTED VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JMP* C.CB</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CBC</th>
<th>SMA</th>
<th>SKIP IF NEGATIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JMP CBC1</td>
<td>DO NOT FORM NEGATIVE</td>
</tr>
</tbody>
</table>
```
CMA FORM 1'S COMPLEMENT
AND =1777 GET MAGNITUDE
TAD =2001 SET SIGN BIT & FORM 2'S COMPLEMENT
AND =3777 CLEAR ESCAPE/INTENSITY BIT
JMP* C*BC RETURN
CBC1 AND =1777 CONVERT TO MODULO 2*10
JMP* C*BC 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
BFEOI $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
    SKIP IF DATA SET CONNECTED
    JMP* B.FI
    DATA SET NOT CONNECTED
    LAC BFS
    GET 201 TASK STATE
    AND  =2
    GET INPUT RECORD AVAILABLE BIT
    SNA
    SKIP IF INPUT RECORD AVAILABLE
    JMP BFI2
    WAIT FOR INPUT RECORD
    LAC BFIB
    GET FIRST RECEIVED CHARACTER
    SZA
    SKIP IF USER RECORD
    JMP BFI2
    WAIT FOR RECORD TO BE TYPED
    LAC* BFIO
    GET CHARACTER FROM INPUT BUFFER
    DAC BFI3
    SAVE INPUT CHARACTER
    ISZ BFIO
    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
BF11  ISZ B.FI  INDICATE SUCCESS
       JMP* B.FI  RETURN
       JMP BFI    GET CHARACTER FROM INPUT BUFFER
BF12  JMS T.P  SCHEDULE PREVIOUS LOC & TERMINATE

BF0   DAC BF03  SAVE CHARACTER TO BE BUFFERED
       IOT 1412  READ 201 STATUS
       AND =1000  GET SET READY BIT
       SNA      SKIP IF DATA SET CONNECTED
       JMP* B.F0  DATA SET NOT CONNECTED
       LAC BFS   GET 201 TASK STATE
       AND =21   GET DATA OUTPUT & ACK EXP BITS
       SZA      SKIP IF OUTPUT BUFFER IS FREE
       JMP BF02  PUT CHARACTER INTO BUFFER LATER
       LAC BF03  GET CHARACTER TO BE BUFFERED
       DAC* BF01  PUT CHARACTER IN OUTPUT BUFFER
       ISZ BF01  INCREMENT INPUT POINTER
       SMA      SKIP IF END-OF-RECORD CHARACTER
       JMP BF01  RETURN
       LAC BFS   GET 201 TASK STATE
       XOR =20   SET DATA OUTPUT PENDING BIT
       DAC BFS   SET NEW 201 TASK STATE
       LAC =BFXMT GET ADDRESS OF TRANSMISSION TASK
       JMS T.S  SCHEDULE TRANSMISSION TASK
BF01  ISZ B.F0  INDICATE SUCCESS
       JMP* B.F0  RETURN
       JMP BF0+1  PUT CHARACTER IN BUFFER
BF02  JMS T.P  SCHEDULE PREVIOUS LOC & TERMINATE

BFXMT IOT 1412  READ 201 STATUS
       AND =60100  GET CAR DET, XMT REQ, CLR SEND BITS
       SZA      SKIP IF ABLE TO TRANSMIT
       JMP BFXMT4 RESCHEDULE BFXMT
       LAC BFS   GET 201 TASK STATE
       RAR      SHIFT ACK EXPECTED BIT INTO LINK
       SZL+RAR  SKIP IF ACK NOT EXPECTED
       JMP BFXMT4 RESCHEDULE BFXMT
       SZL+RAR  SKIP IF INPUT BUFFER EMPTY
       JMP BFXMT4 RESCHEDULE BFXMT
       SNL+RAR  SKIP IF ACK OUTPUT PENDING
JMP BFXMT1  CHECK FOR NAK OUTPUT PENDING
LAC BFS      GET 201 TASK STATE
AND =33      CLEAR ACK OUTPUT PENDING BIT
DAC BFS      SET NEW 201 TASK STATE
LAC =BFIB    GET ADDRESS OF INPUT BUFFER
DAC BFII     RESET INPUT POINTER
DAC BFIO     RESET OUTPUT POINTER
LAC =BFACKR  GET POINTER TO ACK RECORD
JMP BFXMT3   TRANSMIT ACK

BFXMT1 SNL+RAR SKIP IF NAK OUTPUT PENDING
JMP BFXMT2   CHECK FOR DATA OUTPUT PENDING
LAC BFS      GET 201 TASK STATE
AND =27      CLEAR NAK OUTPUT PENDING BIT
DAC BFS      SET NEW 201 TASK STATE
LAC =BFNAKR  GET POINTER TO NAK RECORD
JMP BFXMT3   TRANSMIT NAK

BFXMT2 SNL   SKIP IF DATA OUTPUT PENDING
JMS T.F      NO OUTPUT PENDING
LAC BFS      GET 201 TASK STATE
XOR =21      CLEAR DATA BIT, SET ACK EXP BIT
DAC BFS      SET NEW 201 TASK STATE
LAC =BF0B    GET POINTER TO OUTPUT BUFFER

BFXMT3 DAC BF00 SET OUTPUT POINTER
LAW -10      LOAD AC WITH -8
DAC BFC      SET SYN COUNT
LAW BF SYN   GET SYN CHARACTER
DAC 5        SET TRANSMIT IMAGE
LAC =200005  GET LAC 5 INSTRUCTION
DAC IF0      INITIALIZE XMT INTERRUPT SERVICE
LAC =20000   GET XMT REQ BIT MASK
IOT 1404     SET XMT REQ BIT
JMS T.F      TERMINATE
JMP BFXMT    START TRANSMISSION, IF APPLICABLE

BFXMT4 JMS T.P SCHEDULE PREVIOUS LOCS & TERMINATE

BFACKR LAW BFACK ACK RECORD
BFNAKR LAW BFNAK NAK RECORD

BFTTY LAC BFTTY2 GET CONDITIONAL TELEPRINTER MASK
JMS T.A     ALLOCATE TELEPRINTER, IF NECESSARY
DZM BFTTY2
LAC* BFIO
SPA
JMP BFTTY1
JMS C.A6
XOR =777700
JMS B.T
ISZ BFIO
JMP BFTTY+3
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 ALLOACTION MASK
RELEASE TELEPRINTER
ACKNOWLEDGE RECORD

IFI
LAC 4
LRS 4
AND =377
HLT
SAD =BFSYN
SKP
JMP IFI6
IOT 1412
AND =2000
IOT 1404
LAC =600000+IFI
DAC IFI+3
IOT 1442
JMP IR
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

IFI1
SAD =BFSYN
JMP IFI1-2
SAD =BFDE
JMP ++3
LAC =740000
JMP IFI1-3
LAC =600000+IFI2
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 = BFNAK</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 BF0I</td>
<td>RESET INPUT POINTER</td>
</tr>
<tr>
<td>LAC BFS</td>
<td>GET 201 TASK STATE</td>
</tr>
<tr>
<td>AND = 36</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 = BFXMT</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>
IFI4  CLL PREPARE TO SHIFT ZEROS INTO AC
ALS 10 SHIFT HIGH ORDER CHECK INTO POSITION
DAC BFCKR SAVE HIGH ORDER BLOCK CHECK
LAC =600000+IFI5 GET JMP IFI5 INSTRUCTION
JMP IFI1-3 MODIFY INTERRUPT SERVICE
IFI5  XOR BFCKR FORM COMPLETE BLOCK CHECK
SAD BFCK SKIP IF BAD RECORD
JMP IFI51 INDICATE INPUT BUFFER FULL
LAC =BFIB GET ADDRESS OF INPUT BUFFER
DAC BFII RESET INPUT POINTER
DAC BFIO RESET OUTPUT POINTER
LAC BFS GET 201 TASK STATE
XOR =10 INDICATE NAK PENDING
DAC BFS SET NEW 201 TASK STATE
JMP IFI4-3 SCHEDULE TRANSMISSION TASK
IFI51 LAC BFS GET 201 TASK STATE
XOR =2 INDICATE INPUT BUFFER FULL
DAC BFS SET NEW 201 TASK STATE
LAC BFEOF GET END-OF-RECORD CHARACTER
JMS BFENDS PROCESS ENQUIRY, IF PRESENT
LAC BFIB GET FIRST RECEIVED CHARACTER
SNA SKIP IF UNSOLICITED RECORD
JMP IFI32-2 MODIFY INTERRUPT SERVICE
LAC =BFTTY GET ADDRESS OF BYPASS TASK
JMP IFI4-2 SCHEDULE BYPASS TASK
IFI6 IOT 1412 READ 201 STATUS
AND =2000 GET TEXT BIT
IOT 1404 CLEAR TEXT BIT
JMP IFI1-2 CLEAR FLAGS AND RETURN
IFI0 HLT STATE VARIABLE
SAD =760000+BFSN SKIP IF NOT SYN
JMP IF01 SYN SENT LAST TIME
SAD =760000+BFDLE SKIP IF NOT DLE
JMP IF03 DLE SENT LAST TIME
SMA SKIP IF END-OF-RECORD CHARACTER
JMP IF04 TEXT CHARACTER SENT LAST TIME
JMP IF05 ENTER BLOCK CHECK PROCEDURE
IFI0 ISZ BFC SKIP IF LAST SYN SENT
JMP IF02+2 CLEAR FLAGS AND RETURN
LAC =BFdle  GET INITIAL DLE  
SKP  SET TRANSMIT IMAGE  
IFO2 LAW BFdle  GET DLE CHARACTER  
DAC 5  SET TRANSMIT IMAGE  
IOT 1442  CLEAR 201 FLAGS  
JMP IR  RETURN FROM INTERRUPT  
IFO3 LAC* BF00  GET CHARACTER FROM BUFFER  
JMS BFCKS  UPDATE BLOCK CHECK  
LAC BF00  GET OUTPUT POINTER  
SAD =BF0B  SKIP IF NOT FIRST CHARACTER  
DZM BFCK  CLEAR BLOCK CHECK  
LAC* BF00  GET CHARACTER FROM OUTPUT BUFFER  
ISZ BF00  INCREMENT OUTPUT POINTER  
JMP IFO2+1  TRANSMIT CHARACTER  
IFO4 LAC* BF00  GET CHARACTER FROM BUFFER  
SAD =BFdle  SKIP IF NOT DLE (EVEN PARITY)  
JMP IFO2  PRECEDE WITH DLE  
SAD =BFdle-200  SKIP IF NOT DLE (ODD PARITY)  
JMP IFO2  PRECEDE WITH DLE  
SMA  SKIP IF END OF RECORD  
JMP IFO3+1  SEND CHARACTER FROM BUFFER  
AND =377  TRUNCATE HIGH ORDER BITS  
JMS BFENQS  PROCESS ENQUIRY, IF PRESENT  
JMP IFO2  PRECEDE WITH DLE  
IFO5 LAC =600000+IFO6  GET JMP IFO6 INSTRUCTION  
DAC IFO  MODIFY INTERRUPT SERVICE  
LAC BFCK  GET BLOCK CHECK  
LRS 10  SHIFT HIGH ORDER PART INTO POSITI  
JMP IFO2+1  TRANSMIT HIGH ORDER BLOCK CHECK  
IFO6 LAC =600000+IFO7  GET JMP INSTRUCTION  
DAC IFO  MODIFY INTERRUPT SERVICE  
LAC BFCK  GET BLOCK CHECK  
JMP IFO2+1  TRANSMIT LOW ORDER PART  
IFO7 LAC =600000+IFO8  GET JMP IFO8 INSTRUCTION  
DAC IFO  MODIFY INTERRUPT SERVICE  
CLC  GET PAD CHARACTER  
JMP IFO2+1  TRANSMIT PAD  
IFO8 IOT 1412  READ 201 STATUS  
AND =20000  GET XMT REQ BIT  
IOT 1404  CLEAR XMT REQ BIT
JMP IF02+2  CLEAR 201 FLAGS AND RETURN

BFENQS $DC 0
SAD =BFENQ  SKIP IF NOT ENQUIRY
JMP ++4  PROCESS ENQUIRY
SAD =BFEOT  SKIP IF NOT END-OF-TRANSMISSION
SKP  REGARD AS ENQUIRY
JMP* BFENQS  RETURN
LAC =BF0B  GET ADDRESS OF OUTPUT BUFFER
DAC BF0I  RESET INPUT POINTER
LAC =BF1B  GET ADDRESS OF INPUT BUFFER
DAC BF1I  RESET INPUT POINTER
DAC BF10  RESET OUTPUT POINTER
DZM BF1B  DO NOT SCHEDULE BYPASS TASK
DZM BF$  STOP 201 TASK ACTIVITY
LAC =740000  GET NOP INSTRUCTION
DAC IFI+3  MODIFY INTERRUPT SERVICE
JMP* BFENQS  RETURN

BFIS $DC 0
DAC* BFII  PUT CHARACTER IN INPUT BUFFER
JMS BFCKS  UPDATE BLOCK CHECK
LAC BFII  GET INPUT POINTER
SAD =BF1B  SKIP IF INPUT BUFFER NON-EMPTY
DZM BFCK  CLEAR BLOCK CHECK
ISZ BFII  INCREMENT INPUT POINTER
JMP* BFIS  RETURN

BFCKS $DC 0
DAC 23  SAVE CHARACTER
LAW -10  LOAD AC WITH -8
DAC 24  SET COUNTER
LAC BFCK  GET FORMER BLOCK CHECK
BFCKS1 RCR  ROTATE LOW ORDER BIT INTO LINK
DAC BFCK  STORE NEW LOW ORDER 15 BITS
CLQ  PREPARE TO GET LOW ORDER CHAR BIT
LAC 23  GET CHARACTER REMAINS
LRS 1  SHIFT LOW ORDER BIT INTO MO
DAC 23  STORE CHARACTER REMAINS
LACQ  GET LOW ORDER CHARACTER BIT
SZA
CML
LAC BFCK
SZL
XOR #120001
ISZ 24
JMP BFCKS1
DAC BFCK
JMP* BFCKS

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

BFIB $DS 200
BF0B $DS 200
$TITLE READER BUFFER MANAGER

BR  LAC BR0  GET OUTPUT POINTER
    SNA   SKIP IF NOT START OF NEW RECORD
    JMP BR2  CLEAR BUFFER & START READER
    SAD BRI  SKIP IF BUFFER NOT EMPTY
    JMP BR1  WAIT FOR MORE INPUT
    LAC* BR0  GET IMAGE FROM BUFFER
    ISZ BR0  INCREMENT OUTPUT POINTER
    SNA   SKIP IF NOT END OF RECORD
    DZM BR0  INDICATE NEW RECORD NEEDED
    ISZ B.R  INDICATE SUCCESS
    JMP* B.R  RETURN

BR1 SAD =BRQ+200  SKIP IF NOT END OF BUFFER
    JMP BR2  CLEAR BUFFER & START READER
    IOT 314  READ STATUS
    AND =1000  GET READER OUT-OF-TAPE FLAG
    SNA   SKIP IF READER OUT OF TAPE
    JMP BR2-1  SCHEDULE NEW ATTEMPT
    DZM BR0  INDICATE NEW RECORD NEEDED
    JMP* B.R  RETURN
    JMP BR  TRY AGAIN TO GET IMAGE
    JMS T.P  SCHEDULE NEW ATTEMPT

BR2 LAC =BRQ  GET ADDRESS OF READER BUFFER
    DAC BRI  SET INPUT POINTER
    DAC BR0  SET OUTPUT POINTER
    IOT 104  SELECT READER
    JMP BR2-1  SCHEDULE NEW ATTEMPT

IRD IOT 314  READ STATUS
    AND =1000  GET READER OUT-OF-TAPE FLAG
    SZA   SKIP IF TAPE IS IN READER
    JMP IRD1  READER OUT OF TAPE
    IOT 112  READ READER BUFFER
    SZA   SKIP IF BLANK TAPE
    JMP IRD2  PUT IMAGE IN BUFFER
    LAC BRS  GET RECORD SEEK SWITCH
    SZA   SKIP IF END OF RECORD
    JMP IRD3  IGNORE BLANK TAPE
    JMS BRS  SET RECORD SEEK SWITCH
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRS</td>
<td>IDC 0</td>
</tr>
<tr>
<td></td>
<td>RECORD SEEK SWITCH</td>
</tr>
<tr>
<td>DZM*</td>
<td>BRI</td>
</tr>
<tr>
<td></td>
<td>STORE END-OF-RECORD IMAGE</td>
</tr>
<tr>
<td>ISZ</td>
<td>BRI</td>
</tr>
<tr>
<td></td>
<td>INCREMENT INPUT POINTER</td>
</tr>
<tr>
<td>JMP</td>
<td>IR</td>
</tr>
<tr>
<td></td>
<td>RETURN FROM INTERRUPT</td>
</tr>
<tr>
<td>IRD1</td>
<td>IOT 102</td>
</tr>
<tr>
<td></td>
<td>CLEAR READER FLAG</td>
</tr>
<tr>
<td>JMP</td>
<td>IR</td>
</tr>
<tr>
<td></td>
<td>RETURN FROM INTERRUPT</td>
</tr>
<tr>
<td>IRD2</td>
<td>DAC* BRI</td>
</tr>
<tr>
<td></td>
<td>STORE IN READER BUFFER</td>
</tr>
<tr>
<td>ISZ</td>
<td>BRI</td>
</tr>
<tr>
<td></td>
<td>INCREMENT INPUT POINTER</td>
</tr>
<tr>
<td>DZM</td>
<td>BRS</td>
</tr>
<tr>
<td></td>
<td>CLEAR RECORD SEEK SWITCH</td>
</tr>
<tr>
<td>LAC</td>
<td>BRI</td>
</tr>
<tr>
<td></td>
<td>GET INPUT POINTER</td>
</tr>
<tr>
<td>SAD</td>
<td>=BRQ+200</td>
</tr>
<tr>
<td></td>
<td>SKIP IF NOT END OF BUFFER</td>
</tr>
<tr>
<td>JMP</td>
<td>IR</td>
</tr>
<tr>
<td></td>
<td>RETURN FROM INTERRUPT</td>
</tr>
<tr>
<td>IRD3</td>
<td>IOT 104</td>
</tr>
<tr>
<td></td>
<td>SELECT READER</td>
</tr>
<tr>
<td>JMP</td>
<td>IR</td>
</tr>
<tr>
<td></td>
<td>RETURN FROM INTERRUPT</td>
</tr>
<tr>
<td>BRQ</td>
<td>SDS 200</td>
</tr>
</tbody>
</table>
$TITLE

PUNCH BUFFER MANAGER

BP
DAC BP4
JMS BP2
LAC BP4
LMQ
LAC =BPQ
JMS Q.A
JMP BP1
JMS BP2
ISZ B.P
JMP* B.P

BP1
IOT 314
AND =400
SZA
JMP* B.P
SKP
JMP BP+2
JMS T.P

BP2
$DC 0
LAC BP3
SZA
JMP* BP2
LAC =BPQ
JMS Q.F
JMP* BP2
IOT 204
JMS BP3

BP3
$DC 0
JMP* BP2

IPC
IOT 314
AND =400
SZA
JMP IPC1
LAC =BPQ
JMS QF1
JMP IPC1
IOT 204
<table>
<thead>
<tr>
<th>IPC1</th>
<th></th>
<th>BTC 202</th>
<th>CLEAR PUNCH FLAG</th>
</tr>
</thead>
<tbody>
<tr>
<td>DZM</td>
<td></td>
<td>BP3</td>
<td>INDICATE PUNCH IDLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JMP IR</td>
<td>RETURN FROM INTERRUPT</td>
</tr>
<tr>
<td>BP0</td>
<td></td>
<td>$DC ++100</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$DS 100</td>
<td></td>
</tr>
</tbody>
</table>
$TITLE

KEYBOARD BUFFER MANAGER

BK
LAC =BKQ
JMS Q.F
JMP BK1
DAC BKF
JMS C.A6
JMP* B.K
JMP BK
BK1
JMS T.P

GET ADDRESS OF KEYBOARD BUFFER
GET CHARACTER FROM KEYBOARD BUFFER
WAIT FOR MORE INPUT
SAVE ASCII FOR SYSTEM USAGE
CONVERT TO 6-BIT CODE
RETURN
TRY AGAIN TO RETURN CHARACTER
SCHEDULE NEW ATTEMPT

IKB
IOT 312
LMQ
LAC =BKQ
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

BKQ
$DC ++100
$DS 100
**TITLE**

**BT**
- DAC BT5: Save temporarily
- LAC BT5: Get packed word to be buffered
- LMQ: Set up parameter
- LAC = BTQ: Get address of teleprinter buffer
- JMS Q.A: Put packed word into TP buffer
- JMP BT2: Try again later
- LAC BT1: Get teleprinter status switch
- SZA: Skip if teleprinter idle
- JMP* B.T: Return
- IOT 2: Disable interrupts
- LAC B.T: Get return address
- DAC 0: Store interrupt return
- JMS BT1: Set teleprinter status switch

**BT1**
- $DC 0: Teleprinter status switch
- JMP ITP: Fake a teleprinter interrupt
- JMP BT+1: Try again to put char in buffer
- JMS T.P: Schedule new attempt

**BT2**
- $DC 77

**BT3**
- $DC 77

**ITP**
- LAC BT4: Get second character
- SAD = 77: Skip if not null character
- SKP: Look at third character
- JMP ITP3: Type second character
- LAC BT3: Get third character
- SAD = 77: Skip if not null character
- SKP: Type first character
- JMP ITP4: Type third character
- LAC = BTQ: Get address of teleprinter buffer
- JMS QF1: Get packed word from TP buffer
- JMP ITP2: Clear flag & return
- DAC BT3: Set up third character
- LRS 6: Shift second character into place
- DAC BT4: Set up second character
- LRS 6: Shift first character into place

**ITP1**
- AND = 77: Truncate high order bits
- SAD = 77: Skip if not null character
JMP ITP  TYPE NEXT CHARACTER
TAD =C6A1  ADD ADDRESS OF 6-BIT TO ASCII TABLE
DAC 23  SAVE TEMPORARILY
LAC* 23  GET CONVERTED ASCII VALUE
IOT 406  SEND CHARACTER TO TELEPRINTER
JMP IR  RETURN FROM INTERRUPT

ITP2 IOT 402  CLEAR TELEPRINTER FLAG
DZM BT1  INDICATE TELEPRINTER IDLE
JMP IR  RETURN FROM INTERRUPT

ITP3 DAC 23  SAVE TEMPORARILY
LAC #77  GET NULL CHARACTER
DAC BT4  STORE AS SECOND CHARACTER
LAC 23  GET CHARACTER TO BE TYPED
JMP ITP1  TYPE SECOND CHARACTER

ITP4 DAC 23  SAVE TEMPORARILY
LAC #77  GET NULL CHARACTER
DAC BT3  STORE AS THIRD CHARACTER
LAC 23  GET CHARACTER TO BE TYPED
JMP ITP1  TYPE THIRD CHARACTER

BTQ SDC *+100
SDS 100
$TITLE
NONBUFFERED I/O MANAGER

NA
JMS TV PROTECT AGAINST REENTRY
$DC N.A
$DC 0
AND #77 TRUNCATE HIGH ORDER BITS
IOT 1103 SELECT A/D CONVERTER CHANNEL
IOT 1304 SELECT A/D CONVERTER
DZM IAD1 CLEAR CONVERSION SWITCH
NA1 LAC IAD1 GET CONVERSION SWITCH
SNA SKIP IF CONVERSION COMPLETE
JMP NA2 WAIT FOR CONVERSION TO BE COMPLETED
LAC NA3 GET CONVERTED VALUE
JMS T.U UNLOCK N.A
$DC NA
JMP NA1
NA2 JMS T.P CHECK FOR CONVERSION COMPLETE
SCHEDULE CONVERSION CHECK

NC
JMS TV PROTECT AGAINST REENTRY
$DC N.C
$DC 0
DAC 7 SET CLOCK INTERVAL
IOT 44 ENABLE CLOCK
DZM ICK+1 CLEAR CLOCK SWITCH
LAC ICK+1 GET CLOCK SWITCH
SNA SKIP IF TIME INTERVAL HAS ELAPSED
JMP NC1 WAIT A LITTLE LONGER
JMS T.U UNLOCK N.C
$DC NC
JMP *-5
NC1 JMS T.P CHECK ELAPSED TIME
SCHEDULE A LATER CHECK

ND1 IOT 5101 SELECT D/A CONVERTER #1
JMP* N.D1 RETURN

ND2 IOT 5102 SELECT D/A CONVERTER #2
JMP* N.D2 RETURN

ND3 IOT 5104 SELECT D/A CONVERTER #3
JMP* N.D3 RETURN
<table>
<thead>
<tr>
<th>Command</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAD</td>
<td>IOT 1312</td>
<td>READ A/D CONVERTER</td>
</tr>
<tr>
<td></td>
<td>DAC NA3</td>
<td>STORE CONVERTED VALUE</td>
</tr>
<tr>
<td></td>
<td>JMS IAD1</td>
<td>SET CONVERSION SWITCH</td>
</tr>
<tr>
<td>IAD1</td>
<td>SDC 0</td>
<td>CONVERSION SWITCH</td>
</tr>
<tr>
<td></td>
<td>JMP IR</td>
<td>RETURN FROM INTERRUPT</td>
</tr>
<tr>
<td>ICK</td>
<td>JMS ++1</td>
<td>SET CLOCK SWITCH</td>
</tr>
<tr>
<td></td>
<td>SDC 0</td>
<td>CLOCK SWITCH</td>
</tr>
<tr>
<td></td>
<td>IOT 4</td>
<td>CLEAR CLOCK FLAG</td>
</tr>
<tr>
<td></td>
<td>JMP IR</td>
<td>RETURN FROM INTERRUPT</td>
</tr>
</tbody>
</table>
$TITLE

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

PE
JMS ++1
$DC 0
JMP* P.E

PD
DZM PE+1
JMP* P.D

PR
IOT 631
JMP* P.R

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

PN
JMS P.E
JMS T.F

PS1
$DC 0
DAC PRG
LRS 6
AND =77
TAD =200
IOT 705
LLS 6
AND =77
TAD =300
IOT 705
JMP* PS1

IPB
LAC PE+1
SNA
JMP IPB1

PUSH BUTTON PROCESSOR

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

SET PUSH BUTTON ENABLE SWITCH
PUSH BUTTON ENABLE SWITCH
RETURN

CLEAR PUSH BUTTON ENABLE SWITCH
RETURN

READ PUSH BUTTONS
RETURN

DISABLE INTERRUPTS
SET PUSH BUTTONS
ENABLE INTERRUPTS
RETURN

ENABLE MANUAL OPN OF PUSH BUTTONS
TERMINATE TASK

STORE NEW PUSH BUTTON STATUS
SHIFT BITS 0-5 INTO POSITION
TRUNCATE HIGH ORDER BITS
SET BITS 0-5 ENABLE BIT
SET PUSH BUTTONS 0-5
SHIFT BITS 6-11 INTO POSITION
TRUNCATE HIGH ORDER BITS
SET BITS 6-11 ENABLE BITS
SET PUSH BUTTONS 6-11
RETURN

GET PUSH BUTTON ENABLE SWITCH
SKIP IF PUSH BUTTONS ARE ENABLED
RESTORE PUSH BUTTON STATUS
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAC PTT</td>
<td>GET ADDRESS OF PUSH BUTTON SERVICE</td>
</tr>
<tr>
<td>AND =77777</td>
<td>TRUNCATE HIGH ORDER BITS</td>
</tr>
<tr>
<td>JMS TII</td>
<td>SCHEDULE PUSH BUTTON SERVICE</td>
</tr>
<tr>
<td>IOT 631</td>
<td>READ PUSH BUTTONS</td>
</tr>
<tr>
<td>DAC PRG</td>
<td>MODIFY PUSH BUTTON STATUS WORD</td>
</tr>
<tr>
<td>DZM PE+1</td>
<td>DISABLE PUSH BUTTONS</td>
</tr>
<tr>
<td>JMP IR</td>
<td>RETURN FROM INTERRUPT</td>
</tr>
<tr>
<td>IPBI</td>
<td></td>
</tr>
<tr>
<td>LAC PRG</td>
<td>GET FORMER PUSH BUTTON STATUS</td>
</tr>
<tr>
<td>JMS PS1</td>
<td>SET PUSH BUTTONS</td>
</tr>
<tr>
<td>JMP IR</td>
<td>RETURN FROM INTERRUPT</td>
</tr>
</tbody>
</table>
$TITLE DISPLAY COMMUNICATOR

DE JMS ++1 SET DISPLAY INT ENABLE SWITCH
  SDC 0 DISPLAY INT ENABLE SWITCH
  JMP* D.E RETURN

DD DZM DE+1 CLEAR INT ENABLE SWITCH
  JMP* D.D RETURN

DP SNA SKIP IF NOT NULL SERVICE
  LAC =DN GET ADDRESS OF NULL SERVICE
  DAC DPT STORE ADDRESS OF SERVICE TASK
  JMP* D.P RETURN

DA LAC DS1 GET STATUS WORD 1
  LLS 14 SHIFT BREAK FIELD INTO POSITION
  AND =70000 REMOVE ALL BUT BREAK FIELD
  XOR DSA FORM 15-BIT ADDRESS
  JMP* D.A RETURN

DY LAC DS2 GET STATUS WORD 2
  LLS 3 SHIFT HIGH ORDER BIT INTO POSITION
  AND =10000 REMOVE OTHER BITS
  XOR DSY FORM 13-BIT Y COORDINATE
  TAD =-1000 CONVERT RELATIVE TO SCREEN CENTER
  JMP* D.Y RETURN

DX LAC DS2 GET STATUS WORD 2
  LLS 4 SHIFT HIGH ORDER BIT INTO POSITION
  AND =10000 REMOVE OTHER BITS
  XOR DSX FORM 13-BIT X COORDINATE
  TAD =-1000 CONVERT RELATIVE TO SCREEN CENTER
  JMP* D.X RETURN

DO AND =77 TRUNCATE HIGH ORDER BITS
  RCL MULTIPLY PARAMETER BY 2
  CMA FORM 1'S COMPLEMENT
  TAD DSP ADD PUSH DOWN POINTER
  TAD =PDP2-PDP1-1 COMPUTE ADDRESS OF PUSH DOWN ENTRY
  DAC T1 SAVE TEMPORARILY
TAD = -PDP2     FORM VALIDITY CHECK
SPA            SKIP IF PARAMETER VALID
JMP* D.0       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.0        INDICATE SUCCESS
JMP* D.0       RETURN

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

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

DWT $DC 0      DISPLAY READY SWITCH
  LAC DWV      GET TRANSLATION VALUE
  SNA          SKIP IF TRANSLATION PENDING
  JMP XIS1     RESUME DISPLAY & RETURN
  DAC* DWHD    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
TAD = -XP  FORM ADDRESS CHECK
SMA  SKIP IF USER FILE INTERRUPT
JMP XLP  TRACKING INTERRUPT
LAC DE+1  GET DISPLAY INT ENABLE SWITCH
SZA  SKIP IF DISPLAY INTERRUPTS DISABLED
JMP ++3  GET STATUS FOR USER
IOT 724  RESUME DISPLAY
JMP IR  RETURN FROM INTERRUPT
LAC DPT  GET ADDRESS OF SERVICE TASK
JMS DS  SCHEDULE SERVICE & READ STATUS
JMP *-4  RESUME DISPLAY & RETURN

IIS
LAC DSS  GET DISPLAY STATUS WORD 1
AND =7  GET BREAK FIELD
SNA  SKIP IF USER FILE INTERRUPT
JMP XIS  TRACKING INTERRUPT
IOT 611  READ DISPLAY ADDRESS
XOR =10000  INTERPRET WITH BREAK FIELD 1
SAD =D+4  SKIP IF NOT DISPLAY SYNC INTERRUPT
JMS DWT  SET DISPLAY READY SWITCH
LAC DE+1  GET DISPLAY INT ENABLE SWITCH
SZA  SKIP IF DISPLAY INTERRUPTS DISABLED
JMP ++5  GET STATUS FOR USER
IOT 611  READ DISPLAY ADDRESS
TAD =1  FORM RESUME ADDRESS
IOT 1605  RESUME DISPLAY
JMP IR  RETURN FROM INTERRUPT
LAC DSS  GET DISPLAY STATUS WORD 1
LLS 14  SHIFT BREAK FIELD INTO POSITION
AND =70000  REMOVE ALL BUT BREAK FIELD
IOT 601  FORM DISPLAY ADDRESS
DAC 23  SAVE TEMPORARILY
LAC* 23  GET ADDRESS OF SERVICE TASK
JMS DS  SCHEDULE SERVICE & READ STATUS
JMP *-13  RESUME DISPLAY & RETURN

DS
$DC 0  TRUNCATE HIGH ORDER BITS
AND =77777  SCHEDULE SERVICE TASK
JMS TII
DZM DE+1  DISABLE DISPLAY INTERRUPTS
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
TITLE

XI
TAD = 1000
AND = 1777
DAC XPY
LACQ
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

CONVERT RELATIVE TO ORIGIN
CONVERT MODULO 2*10
SET Y TRACKING COORDINATE
GET X COORDINATE
CONVERT RELATIVE TO ORIGIN
CONVERT MODULO 2*10
SET ESCAPE BIT
SET X TRACKING COORDINATE
ENABLE TRACKING
RETURN
ENABLE TRACKING
RETURN
GET POP INSTRUCTION
TERMINATE TRACKING
RETURN
GET POP INSTRUCTION
SKIP IF TRACKING ENABLED
INDICATE SUCCESS
RETURN
GET Y TRACKING COORDINATE
CONVERT RELATIVE TO SCREEN CENTER
RETURN
GET X TRACKING COORDINATE
TRUNCATE ESCAPE
CONVERT RELATIVE TO SCREEN CENTER
RETURN
STATE VARIABLE
STATE VARIABLE
STATE VARIABLE
DO NOT CHANGE Y TRACKING COORDINATE
TRUNCATE HIGH ORDER BITS
SET Y TRACKING COORDINATE
IOT 512       READ X COORDINATE
TAD =-2000    SUBTRACT 1024
SMA           SKIP IF COORDINATE ON SCREEN
JMP *+4       DO NOT CHANGE X COORDINATE
AND =1777     TRUNCATE HIGH ORDER BITS
XOR =4000     SET ESCAPE BIT
DAC 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     SKIP IF VALID
JMP X31  ENABLE SEARCH PATTERN
TAD XL   ADD LOW COORDINATE
SAD XH   SKIP IF VALID
JMP X31  ENABLE SEARCH PATTERN
RCR     DIVIDE BY 2
TAD = -2000  SUBTRACT 1024
SMA     SKIP IF COORDINATE ON SCREEN
JMP X31+2  DO NOT CHANGE X COORDINATE
AND = 1777  CONVERT MODULO 2\times 10
XOR = 4000  SET ESCAPE BIT
DAC XPX  SET X TRACKING COORDINATE
JMP X31+2  LEAVE SEARCH PATTERN INHIBITED
X31
LAW 777  GET SEARCH ENABLE WORD
DAC XPS  ENABLE SEARCH PATTERN
DZM XL   CLEAR LOW COORDINATE
DZM XH   CLEAR HIGH COORDINATE
LAC = 701612  GET IOT 1612 INSTRUCTION
DAC XLP  MODIFY INTERRUPT SERVICE
LAC = 40000+XL  GET DAC XL INSTRUCTION
DAC XLP+1  MODIFY INTERRUPT SERVICE
JMP XIS1  RESUME DISPLAY & RETURN

X4
LAC XH   GET HIGH COORDINATE
SNA     SKIP IF NOT VALID
JMP X41  ENABLE SEARCH PATTERN
TAD XL   ADD LOW COORDINATE
SAD XH   SKIP IF VALID
JMP X41  ENABLE SEARCH PATTERN
RCR     DIVIDE BY 2
TAD = -2000  SUBTRACT 1024
SMA     SKIP IF COORDINATE ON SCREEN
JMP X41+2  DO NOT CHANGE Y TRACKING COORDINATE
AND = 1777  CONVERT MODULO 2\times 10
DAC XPY  SET Y TRACKING COORDINATE
JMP X41+2  LEAVE SEARCH PATTERN INHIBITED
X41
LAW 777  GET SEARCH ENABLE WORD
DAC XPS  ENABLE SEARCH PATTERN
LAC XLP+7  GET TAD = -2000 INSTRUCTION
DAC XLP+1  MODIFY INTERRUPT SERVICE
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAC = 740100</td>
<td>GET SMA INSTRUCTION</td>
</tr>
<tr>
<td>DAC XLP+2</td>
<td>MODIFY INTERRUPT SERVICE</td>
</tr>
<tr>
<td>JMP XIS1</td>
<td>RESUME DISPLAY &amp; RETURN</td>
</tr>
<tr>
<td>X5 LAW 3000</td>
<td>GET POP INSTRUCTION</td>
</tr>
<tr>
<td>DAC XP</td>
<td>DISABLE TRACKING</td>
</tr>
<tr>
<td>JMP XIS1</td>
<td>RESUME DISPLAY &amp; RETURN</td>
</tr>
</tbody>
</table>
$TITLE STRUCTURE TOPOLOGY OPERATORS

STL

JMS B4
JMP* S.TL
JMP S.TL

GET 8-WORD BLOCK

NOT ENOUGH STORAGE

SAVE ADDRESS FOR RETURN

COMPUTE INITIAL INDEX VALUE

SET AUTOINDEX REGISTER

GET DISPLAY NOP INSTRUCTION

STORE IN FIRST LOCATION IN HEAD

STORE IN SECOND LOCATION IN HEAD

STORE IN THIRD LOCATION IN HEAD

GET VEC INSTRUCTION

STORE IN FOURTH LOCATION IN HEAD

GET DISPLAY NOP INSTRUCTION

STORE IN FIFTH LOCATION IN HEAD

GET ZERO X COORD WITH ESCAPE BIT

STORE IN SIXTH LOCATION IN HEAD

GET JUMP1 INSTRUCTION

STORE IN SEVENTH LOCATION IN HEAD

GET 8-WORD BLOCK

NOT ENOUGH STORAGE

STORE ADDRESS OF TAIL IN HEAD

COMPUTE INITIAL INDEX VALUE

SET AUTOINDEX REGISTER

GET UNCONDITIONAL DISPLAY SKIP

STORE IN FIRST LOCATION IN TAIL

GET INTERNAL STOP INSTRUCTION

STORE IN SECOND LOCATION IN TAIL

ZERO IN THIRD LOCATION IN TAIL

STORE DISPLAY NOP IN FOURTH LOCATION

GET VEC INSTRUCTION

STORE IN FIFTH LOC IN TAIL

STORE IN SIXTH LOC IN TAIL

GET ZERO X COORD WITH ESCAPE BIT

STORE IN SEVENTH LOCATION IN TAIL

GET POP INSTRUCTION

STORE IN EIGHTH LOC IN TAIL

GET ADDRESS OF CREATED LEVEL

INDICATE SUCCESS

RETURN
**STL1**

DZM* T5  FREE FIRST 4-WORD BLOCK IN HEAD
LAC T5  GET ADDRESS OF 8-WORD BLOCK
TAD =4  FORM ADDRESS OF SECOND 4-WORD BLOCK
DAC T5  SAVE TEMPORARILY
DZM* T5  FREE SECOND 4-WORD BLOCK IN HEAD
JMP* S.TL FAILURE RETURN

**STD**

SAD =DHAL SKU IF NOT HIGHEST ACTIVE LEVEL
JMP STD1 HIGHEST ACTIVE LEVEL
DAC T1 SAVE ADDRESS OF FIRST HEAD BLOCK
TAD =4 FORM ADDRESS OF SECOND HEAD BLOCK
DAC T2 SAVE ADDRESS OF SECOND HEAD BLOCK
TAD =3 FORM POINTER TO LAST LOC IN HEAD
DAC T3 SAVE TEMPORARILY
LAC* T3 GET ADDRESS OF TAIL (OR NODE)
DAC T3 SAVE ADDRESS OF TAIL (OR NODE)
TAD =4 FORM ADDRESS OF SECOND TAIL BLOCK
DAC T4 SAVE
LAC* T3 GET FIRST WORD OF TAIL (OR NODE)
AND =777770 TRUNCATE BREAK FIELD
SAD =762010 SKIP IF NOT NODE
JMP* S.TD LEVEL NOT EMPTY
DZM* T1 RELEASE FIRST HEAD BLOCK
DZM* T2 RELEASE SECOND HEAD BLOCK
DZM* T3 RELEASE FIRST TAIL BLOCK
DZM* T4 RELEASE SECOND TAIL BLOCK

**STD1**

ISZ S.TD INDICATE SUCCESS
JMP* S.TD RETURN

**STI**

TAD =7 FORM POINTER TO LAST LOC IN HEAD
DAC T5 SAVE
JMS B3 CREATE 4-WORD BLOCK
JMP* S.TI NOT ENOUGH STORAGE
DAC T1 SET POINTER TO BLOCK
TAD =-1 COMPUTE INITIAL INDEX VALUE
DAC 12 SET AUTOINDEX REGISTER
LLS 6 SHIFT BREAK FIELD INTO AC
AND =7 TRUNCATE HIGH ORDER BITS
XOR =762010 FORM PUSH JUMP INSTRUCTION
DAC* 12 STORE IN FIRST LOC IN BLOCK
AND = 7
LLS 14
DAC* 12
LAW 2001
DAC* 12
DAC* T5
DAC* T1
DAC* T5
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

STR
JMS TV
PROTECT AGAINST REENTRY
SDC S.TR
SDC 0
TAD = 7
GET POINTER TO END OF HEAD
DAC T1
SAVE TEMPORARILY
DAC* T1
GET ADDRESS OF FIRST ELEMENT
DAC T2
SAVE TEMPORARILY
DAC STR2
SAVE ADDRESS FOR REMOVAL
DAC T2
GET FIRST WORD OF FIRST ELEMENT
AND = 777770
SAD = 762010
SKIP IF NOT NODE
SKP NODE
JMP STR1+7
SUBSTRUCTURE NOT IN LEVEL
ISZ T2
FORM POINTER TO ADR OF SUBSTRUCTURE
LACQ
GET ADDRESS OF GIVEN SUBSTRUCTURE
SAD* T2
SKIP IF NO MATCH
JMP STR1
SUBSTRUCTURE FOUND
LAC T2
GET POINTER TO ADR OF SUBSTRUCTURE
TAD =2
FORM POINTER TO END OF NODE
JMP STR+4
TRY NEXT NODE

STR1
ISZ T2
INCREMENT POINTER TO LOC IN NODE
ISZ T2
INCREMENT POINTER TO LOC IN NODE
DAC* T2
GET ADR OF SUCCESSOR TO NODE
DAC* T1
STORE IN PREVIOUS NODE (OR HEAD)
JMS DW
WAIT FOR DISPLAY TO SETTLE DOWN
DZM* STR2
RELEASE NODE TO FREE STORAGE
ISZ STR+2
INDICATE SUCCESS
JMS T.U
UNLOCK S.TRD
$TITLE

SLH
LAC =DHAL
JMP* S.LH

GET ADDRESS OF HIGHEST ACTIVE LEVEL
RETURN

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
LAC0
JMS C.BC
XOR =760000
DAC DWV

PROTECT AGAINST REENTRY

SLY1
LAC DWV
GET TRANSLATION VALUE
SZA
JMP **+4
JMS T.U
$DC SLY
JMP SLY1
JMS T.P

CHECK FOR TRANSLATION COMPLETE
SCHEDULE COMPLETION CHECK

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
LAC0

PROTECT AGAINST REENTRY

SKIP IF NOT HIGHEST ACTIVE LEVEL
RETURN

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 OCCUPIED
GET TRANSLATION VALUE
SKIP IF TRANSLATION COMPLETE
RESCHEDULE COMPLETION CHECK
UNLOCK S.LY
CHECK FOR TRANSLATION COMPLETE
SCHEDULE COMPLETION CHECK

FORM POINTER TO X COORD IN HEAD
SAVE

FORM POINTER TO END OF HEAD
GET ADDRESS OF TAIL
FORM POINTER TO X COORE IN TAIL
SAVE

GET X INCREMENT
JMS C×BC
XOR =4000
DAC DWV
SLX1
LAC DWV
SZA
JMP ++4
JMS T×U
$DC SLX
JMP SLXI
JMS T×P
CHECK FOR TRANSLATION COMPLETE
SCHEDULE COMPLETION CHECK

SLP
SAD =DHAL
JMP* S×LP
TAD =2
DAC T1
LACQ
AND =777
DAC* T1
JMP* S×LP
SKIP IF NOT HIGHEST ACTIVE LEVEL
RETURN
GET ADDRESS OF PARAMETER SLOT
SAVE TEMPORARILY
GET PARAMETERS
TRUNCATE HIGH ORDER BITS
STORE PARAMETERS IN LEVEL
RETURN

SLBE
SAD =DHAL
JMP* S×LBE
TAD =1
DAC T2
TAD =6
JMS SLT
TAD =3
DAC T1
LAW 6301
DAC* T1
LAC* T2
AND =74
TAD =6302
DAC* T2
JMP* S×LBE
SKIP IF NOT HIGHEST ACTIVE LEVEL
RETURN
FORM POINTER TO COUNT SLOT
SAVE TEMPORARILY
FORM POINTER TO END OF HEAD
GET ADDRESS OF TAIL
FORM POINTER TO BLINK OFF SLOT
SAVE TEMPORARILY
GET BLINK OFF INSTRUCTION
STORE IN TAIL
GET COUNT INSTRUCTION
GET COUNT BITS
FORM NEW COUNT INSTRUCTION
STORE NEW COUNT INSTRUCTION
RETURN

SLBD
SAD =DHAL
JMP* S×LBD
TAD =1
DAC T2
SKIP IF NOT HIGHEST ACTIVE LEVEL
RETURN
FORM POINTER TO COUNT SLOT
SAVE TEMPORARILY
TAD = 6
JMS SLT
TAD = 3
DAC T1
LAC* T2
AND = 777774
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.LU
DAC* T1
JMP* S.LU

SKIP IF NOT HIGHEST ACTIVE LEVEL
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 SLSP

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

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

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

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

TEXT OPERATORS

LT
DAC LT2
LAC* LT2
CMA
DAC LT3
ISZ LT3
SKP
JMP* L.T
ISZ LT2
LAC* LT2
JMS B.T
ISZ LT3
JMP LT1
JMP* L.T

LT1
RETURN

LD
DAC T5
DAC T1
LAC* T1
CMA
DAC T2
LAC =7
DAC T3
ISZ T2
SKP
JMP LD4
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
DAC T6         SAVE ADDRESS FOR RETURN
LAC* T5        GET WORD COUNT
CMA            FORM 1'S COMPLEMENT
TAD =1         FORM 2'S COMPLEMENT
DAC T2         STORE COMPLEMENTED WORD COUNT
DZM T3         CLEAR HORIZONTAL COUNT
DZM T4         CLEAR VERTICAL COUNT
LD2
ISZ T5         SET POINTER TO NEXT TEXT WORD
LAC* T5        GET NEXT TEXT WORD
LRS 14         SHIFT FIRST CHARACTER INTO POSITION
JMS LD6         PUT FIRST CHARACTER INTO LEAF
JMS LD6         PUT SECOND CHARACTER INTO LEAF
JMS LD6         PUT THIRD CHARACTER INTO LEAF
ISZ T2         INCREMENT COUNT & SKIP IF DONE
JMP LD2         PROCESS NEXT TEXT WORD
LAW 1121       GET VEC INSTRUCTION
DAC* T1        STORE IN TEXT LEAF
ISZ T1         INCREMENT POINTER TO LOC IN LEAF
LAC T4         GET VERTICAL COUNT
CLQ            PREPARE TO SHIFT ZEROS INTO AC
LLS 4          MULTIPLY BY 16
XOR =400000     SET TO NONZERO VALUE
DAC* T1        STORE IN TEXT LEAF
ISZ T1         INCREMENT POINTER TO LOC IN LEAF
LAC T3         GET HORIZONTAL COUNT
LLS 3          MULTIPLY BY 8
AND =1777       CONVERT MODULO 2\times10
XOR =6000       SET ESCAPE BIT & MINUS SIGN
DAC* T1        STORE IN TEXT LEAF
ISZ T1         INCREMENT POINTER TO LOC IN LEAF
LD3
LAW 3000       GET POP INSTRUCTION
DAC* T1        STORE IN TEXT LEAF
LAC T6         GET ADDRESS OF TEXT LEAF
ISZ L*D        INDICATE SUCCESS
JMP* L*D       RETURN
LD4
LAC =LD3       GET ADDRESS OF POP INSTRUCTION
JMP *-3        INDICATE SUCCESS & RETURN
LD5
$DC Ø          TRUNCATE HIGH ORDER BITS
AND =77
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAD =77</td>
<td>Skip if not null character</td>
</tr>
<tr>
<td>JMP *+5</td>
<td>Null character -- return</td>
</tr>
<tr>
<td>SAD =74</td>
<td>Skip if not carriage return</td>
</tr>
<tr>
<td>ISZ T3</td>
<td>Increment leaf size extra time</td>
</tr>
<tr>
<td>ISZ T3</td>
<td>Increment leaf size</td>
</tr>
<tr>
<td>ISZ T3</td>
<td>Increment leaf size</td>
</tr>
<tr>
<td>LLS 6</td>
<td>Shift next character into position</td>
</tr>
<tr>
<td>JMP* LD5</td>
<td>Return</td>
</tr>
<tr>
<td>LD6 $DC 0</td>
<td>Truncate high order bits</td>
</tr>
<tr>
<td>AND =77</td>
<td>Skip if not null character</td>
</tr>
<tr>
<td>SAD =77</td>
<td>Null character -- return</td>
</tr>
<tr>
<td>JMP LD7-2</td>
<td>Skip if not carriage return</td>
</tr>
<tr>
<td>SAD =74</td>
<td>Put carriage return into leaf</td>
</tr>
<tr>
<td>SAD =75</td>
<td>Skip if not line feed</td>
</tr>
<tr>
<td>SKP</td>
<td>Line feed -- increment vert count</td>
</tr>
<tr>
<td>JMP *+3</td>
<td>Normal character</td>
</tr>
<tr>
<td>ISZ T4</td>
<td>Increment vertical count</td>
</tr>
<tr>
<td>SKP</td>
<td>Leave horizontal count alone</td>
</tr>
<tr>
<td>ISZ T3</td>
<td>Increment horizontal count</td>
</tr>
<tr>
<td>TAD =LD8</td>
<td>Add address of conversion table</td>
</tr>
<tr>
<td>DAC T7</td>
<td>Save temporarily</td>
</tr>
<tr>
<td>LAW 2010</td>
<td>Get push jump instruction</td>
</tr>
<tr>
<td>DAC* T1</td>
<td>Store in text leaf</td>
</tr>
<tr>
<td>ISZ T1</td>
<td>Increment pointer to loc in leaf</td>
</tr>
<tr>
<td>DAC* T7</td>
<td>Get address of display for char</td>
</tr>
<tr>
<td>DAC* T1</td>
<td>Store in text leaf</td>
</tr>
<tr>
<td>ISZ T1</td>
<td>Increment pointer to loc in leaf</td>
</tr>
<tr>
<td>LLS 6</td>
<td>Shift next character into position</td>
</tr>
<tr>
<td>JMP* LD6</td>
<td>Return</td>
</tr>
<tr>
<td>LD7 LACQ</td>
<td>Get MQ contents</td>
</tr>
<tr>
<td>DAC T7</td>
<td>Save temporarily</td>
</tr>
<tr>
<td>LAW 1121</td>
<td>Get vector instruction</td>
</tr>
<tr>
<td>DAC* T1</td>
<td>Store in text leaf</td>
</tr>
<tr>
<td>ISZ T1</td>
<td>Increment pointer to loc in leaf</td>
</tr>
<tr>
<td>LAW 0</td>
<td>Get zero y displacement</td>
</tr>
<tr>
<td>DAC* T1</td>
<td>Store zero y displacement in leaf</td>
</tr>
<tr>
<td>ISZ T1</td>
<td>Increment pointer to loc in leaf</td>
</tr>
<tr>
<td>LAC T3</td>
<td>Get horizontal displacement</td>
</tr>
</tbody>
</table>
CLG
LLS 3
AND $1777
XOR $6000
DAC* T1
ISZ T1
DZM T3
LAC T7
LRS T4
JMP* LD6

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

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

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

LD8 $DC D00
$DC D01+10000
$DC D02+20000
$DC D03+30000
$DC D04+40000
$DC D05+50000
$DC D06+60000
$DC D07+70000
$DC D10+100000
$DC D11+110000
$DC D12+120000
$DC D13+130000
$DC D14+140000
$DC D15+150000
$DC D16+160000
$DC D17+170000
$DC D20+200000
$DC D21+210000
$DC D22+220000
$DC D23+230000
$DC D24+240000
$DC D25+250000
$DC D26+260000
$DC D27+270000
$DC D30+300000
$DC D31+310000
$DC D32+320000
$DC D33+330000
$DC D34+340000
$DC D35+350000
$DC D36+360000
$DC D37+370000
$DC D40+400000
$DC D41+410000
$DC D42+420000
$DC D43+430000
$DC D44+440000
$DC D45+450000
$DC D46+460000
$DC D47+470000
$DC D50+500000
$DC D51+510000
$DC D52+520000
$DC D53+530000
$DC D54+540000
$DC D55+550000
$DC D56+560000
$DC D57+570000
$DC D60+600000
$DC D61+610000
$DC D62+620000
$DC D63+630000
$DC D64+640000
$DC D65+650000
$DC D66+660000
$DC D67+670000
$DC D70+700000
$DC D71+710000
$DC D72+720000
$DC D73+730000
NOP
$DC$ D75+750000
$DC$ D76+760000
$TITLE

IDL E

LAW 17475  GET CARRIAGE RETURN, LINE FEED CODE
JMS B.T  TYPE CARRIAGE RETURN, LINE FEED
LAW 10  GET TELEPRINTER ALLOCATION MASK
JMS T.R  RELEASE TELEPRINTER
JMS B.K  GET KEYBOARD CHARACTER
DAC T3  SAVE KEYBOARD CHARACTER
LAW 10  GET TELEPRINTER ALLOCATION MASK
JMS T.A  ALLOCATE TELEPRINTER
LAC BKF  GET ASCII FORM OF CHARACTER
SAD =205  SKIP IF NOT ENQUIRY
JMP TTY4  SEND ENQUIRY RECORD
LAC T3  GET KEYBOARD CHARACTER
SAD =14  SKIP IF NOT C
LAC =IDLEC  GET "CLEAR" RESPONSE POINTER
SAD =33  SKIP IF NOT R
LAC =IDLER  GET "RUN" RESPONSE POINTER
SAD =34  SKIP IF NOT S
LAC =IDLES  GET "SCHEDULE" RESPONSE POINTER
SAD =72  SKIP IF NOT #
LAC =IDLE1  GET TTY/201 RESPONSE POINTER
SAD =17  SKIP IF NOT F
LAC =IDLEF  GET "FROM" RESPONSE POINTER
DAC T3  SAVE SELECTED RESPONSE POINTER
AND =777700  TRUNCATE LOW ORDER BITS
SNA  SKIP IF LEGAL COMMAND
JMP IDLEQ  CANCEL COMMAND
LAC T3  GET RESPONSE POINTER
TAD =1  COMPUTE ADDRESS OF TEXT LIST
JMS L.T  TYPE TEXT LIST
LAC* T3  GET ADDRESS OF RESPONSE
DAC T3  SAVE TEMPORARILY
JMP* T3  EXECUTE RESPONSE

IDLEQ  LAC =657475  GET QUESTION MARK CODE
JMP IDLEQ  TYPE & GET NEW COMMAND

IDLEC  SDC CLEAR
SDC 2
$TEXT "CLEAR"

IDLER $DC RUN
$DC 2
$TEXT "RUN"
$DC 747575

IDLES $DC SCHED
$DC 3
$TEXT "SCHEDULE"

IDLE1 $DC TTY201
$DC 0

IDLEF $DC FROM
$DC 2
$TEXT "FROM"

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

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

CLEARD $DC 5
$TEXT "DISPLAY STORAGE"

CLEART $DC 4
$TEXT "TASK QUEUE"

RUN  LAW 10  GET TELEPRINTER MASK
      JMS T.R  RELEASE TELEPRINTER
      JMS STC  CLEAR DISPLAY STORAGE
RUNI LAC =SCHEDQ GET ADDRESS OF FROZEN TASK QUEUE
JMS Q.F GET TASK FROM FROZEN TASK QUEUE
JMS I.F TERMINATE IDLE-TIME EXECUTION
JMS T.S SCHEDULE TASK FROM FROZEN QUEUE
JMP RUNI ENABLE NEXT TASK

SCHED JMS OCTAL5 GET ADDRESS FROM KEYBOARD
JMP IDLEQ CANCEL COMMAND
LMQ SET UP PARAMETER
LAC =SCHEDQ GET ADDRESS OF FROZEN TASK QUEUE
JMS Q.A ADD TASK TO FROZEN QUEUE
SKP TYPE DIAGNOSTIC
JMP IDLE GET NEW COMMAND
LAC =SCHED1 GET ADDRESS OF TEXT LIST
JMS L.T TYPE TEXT LIST
JMP IDLE GET NEW COMMAND

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

SCHEDQ $DC **37
$DC **36
$DC **35
$ORG **35

TTY201 LAW 17772 GET # CODE
JMS B.T TYPE #
JMS ECHO ECHO KEYBOARD CHARACTER
LAC BKF GET ASCII FORM OF CHARACTER
SAD =215 SKIP IF NOT CARRIAGE RETURN
JMP TTY1 TERMINATE RECORD WITH ETX
SAD =337 SKIP IF NOT BACK ARROW
JMP TTY2 DELETE CHARACTER
SAD =377 SKIP IF NOT RUBOUT
JMP TTY3 CLEAR 201 OUTPUT BUFFER
JMS B.F0 SEND CHARACTER TO 201 OUTPUT BUFFER
SKP DATA SET NOT CONNECTED
JMP TTY201+2 PROCESS NEXT CHARACTER
LAC =TTY5 GET ADDRESS OF TEXT LIST
JMS L.T TYPE DIAGNOSTIC
TTY1
JMP IDLE   GET NEW COMMAND
LAW BFETX  GET END OF TEXT CHARACTER
JMS B·F0   SEND TO 201 OUTPUT BUFFER
JMP *-5    DATA SET NOT CONNECTED
JMP IDLE   GET NEW COMMAND
TTY2
LAC =BF0B  GET ADDRESS OF OUTPUT BUFFER
SAD BF0I   SKIP IF BUFFER NON-EMPTY
JMP TTY201+2 IGNORE CHARACTER DELETE
JMS B·F0   WAIT FOR ACK TO LAST RECORD
JMP TTY1-3  DATA SET NOT CONNECTED
LAW -2     LOAD AC WITH -2
TAD BF0I   COMPUTE NEW VALUE OF INPUT POINTER
DAC BF0I   BACKSPACE OUTPUT BUFFER
JMP TTY201+2 PROCESS NEXT CHARACTER
TTY3
JMS B·F0   WAIT FOR ACK TO LAST RECORD
NOP        DATA SET NOT CONNECTED
LAC =BF0B  GET ADDRESS OF 201 OUTPUT BUFFER
DAC BF0I   RESET INPUT POINTER
LAC =TTY6  GET ADDRESS OF TEXT LIST
JMS L·T    TYPE DIAGNOSTIC
JMP IDLE   GET NEW COMMAND
TTY4
LAW BFENO  GET ENQUIRY
JMP TTY1+1 SEND TO 201 OUTPUT BUFFER
TTY5
$DC 11     -- DATA SET NOT CONNECTED"
TTY6
$DC 4      -- DELETED"

FROM
JMS B·K   GET KEYBOARD CHARACTER
D1M T3    CLEAR DATA TRANSFER POINTER
SAD =14   SKIP IF NOT C
JMP FROM1  FROM CORE
ISZ T3    INCREMENT DATA TRANSFER POINTER
SAD =31   SKIP IF NOT P
JMP FROM2  FROM PAPER TAPE
ISZ T3    INCREMENT DATA TRANSFER POINTER
SAD =35   SKIP IF NOT T
JMP FROM3  FROM TELETYPewriter
JMP IDLE0  CANCEL COMMAND
FROM1 LAC =FROMC GET ADDRESS OF TEXT LIST
    JMP FROM4 TYPE TEXT LIST
FROM2 LAC =FROMP GET ADDRESS OF TEXT LIST
    SKP TYPE TEXT LIST
FROM3 LAC =FROMT GET ADDRESS OF TEXT LIST
FROM4 JMS L.T TYPE TEXT LIST
    LAC T3 GET DATA TRANSFER POINTER
    CLL+RTL MULTIPLY BY 4
    TAD =FROM11 ADD ADDRESS OF TABLE
    DAC T3 STORE REFINED DATA TRANSFER POINTER
    LAC =FROMTO GET ADDRESS OF TEXT LIST
    JMS L.T TYPE TEXT LIST
    JMS B*K GET KEYBOARD CHARACTER
    SAD =14 SKIP IF NOT C
    JMP FROM5 TO CORE
    ISZ T3 INCREMENT DATA TRANSFER POINTER
    SAD =31 SKIP IF NOT P
    JMP FROM6 TO PAPER TAPE
    ISZ T3 INCREMENT DATA TRANSFER POINTER
    SAD =35 SKIP IF NOT T
    JMP FROM7 TO TELETYPewriter
    ISZ T3 INCREMENT DATA TRANSFER POINTER
    SAD =15 SKIP IF NOT D
    JMP FROM8 TO DISPLAY
    JMP IDLEQ CANCEL COMMAND
FROM5 LAC =FROMC GET ADDRESS OF TEXT LIST
    JMP FROM9 TYPE TEXT LIST
FROM6 LAC =FROMP GET ADDRESS OF TEXT LIST
    JMP FROM9 TYPE TEXT LIST
FROM7 LAC =FROMT GET ADDRESS OF TEXT LIST
FROM8 LAC =FROMMD GET ADDRESS OF TEXT LIST
    SKP TYPE 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
    SDC TRCP
    SDC TRCT
LAW 100  GET ORIGIN CONTROL BIT
DAC T3   SET CONTROL MASK
LAC TRBKL GET ORIGIN OF BLOCK
JMS TRCP2 PUNCH ORIGIN
LAW 200  GET DATA CONTROL BIT
DAC T3   SET CONTROL MASK
TRCP1 LAC* TRBKL GET DATA WORD
JMS TRCP2 PUNCH DATA WORD
ISZ TRBKL INCREMENT POINTER
ISZ TRBKC INCREMENT COUNT & SKIP IF DONE
JMP TRCP1 PUNCH NEXT WORD
JMP IDLE GET NEW COMMAND

TRCP2 $DC 0
DAC T4   SAVE WORD TO BE PUNCHED
LRS 14   SHIFT HIGH ORDER BITS INTO POSITION
AND =77  TRUNCATE BITS FROM LINK
XOR T3   SET CONTROL BIT
JMS PUNCH PUNCH IMAGE
LAC T4   GET WORD TO BE PUNCHED
LRS 6    SHIFT MIDDLE BITS INTO POSITION
AND =77  TRUNCATE HIGH ORDER BITS
XOR T3   SET CONTROL BIT
JMS PUNCH PUNCH IMAGE
LAC T4   GET WORD TO BE PUNCHED
AND =77  TRUNCATE HIGH ORDER BITS
XOR T3   SET CONTROL BIT
JMS PUNCH PUNCH IMAGE
JMP* TRCP2 RETURN

TRCT JMS TRBK GET CORE BLOCK FROM KEYBOARD
LAW 17475 GET CARRIAGE RETURN, LINE FEED CODE
JMS B.T  TYPE CARRIAGE RETURN, LINE FEED
LAC TRBKL GET ADDRESS TO BE TYPED
JMS C.B6 CONVERT TO 6-BIT CODE
TAD =770000 REMOVE HIGH ORDER ZERO
JMS TRKT TYPE ADDRESS
LAW 17676 GET CODE FOR TWO SPACES
JMS B.T  TYPE TWO SPACES
LAW 17770 LOAD AC WITH -8
DAC T3
TRCT1 LAW 17677 GET CODE FOR ONE SPACE
JMS B.T TYPE SPACE
LAC* TRBKL GET WORD TO BE TYPED
JMS C.B6 CONVERT TO 6-BIT CODE
JMS TRKT TYPE WORD
ISZ TRBKL INCREMENT LOCATION POINTER
ISZ TRBKC INCREMENT COUNT & SKIP IF DONE
SKP 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 TRBK 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
TRCD1 LAC TRBK GET ADDRESS TO BE DISPLAYED
JMS C.B6 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.B6 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
ISZ TRBKC
JMP TRCD3
CLC
DAC T3
JMS TRD3
JMP IDLE
TRCD3 ISZ T3
JMP TRCD2
LAC =747575
JMS TRD2
JMP TRCD1
TRPC JMS READ
SNA
JMP IDLE
DAC T3
AND =300
SAD =100
JMP TRPC1
SAD =200
JMP TRPC2
JMS READ
SNA
JMP TRPC
JMP TRPC
TRPC1 JMS TRPC3
DAC T4
JMP TRPC
TRPC2 JMS TRPC3
DAC* T4
ISZ T4
JMP TRPC
TRPC3 SDC 0
JMS READ
LRS 6
DAC T3
LLS 6
DAC T3
JMS READ
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
IGNORE TAPE IMAGE
FINISH READING ORIGIN
SET 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
LRS 6               SHIFT DATA BITS INTO MQ
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               SDC 0        GET END-OF-RECORD MARK
                  CLA           PUNCH END-OF-RECORD MARK
                  JMS PUNCH     PUNCH END-OF-RECORD MARK
                  JMP* TRPP3    RETURN
IDLE-TIME TASK (CONTINUED)

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

**TRPT1**
- LAW 17475
- JMS B*T
- LAC T3

**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
- PRECEDE WITH NULL CHARACTERS
- TYPE CHARACTER FROM TAPE
- GET IMAGE FROM TAPE
- SKIP IF NOT END OF RECORD
- GET NEW COMMAND
- TYPE CHARACTER
- READ IMAGE FROM TAPE
- SKIP IF NOT END-OF-RECORD CHARACTER
- RESTART DATA TRANSFER
- SKIP IF NOT LINE FEED
- RESTART DATA TRANSFER
- SKIP IF NOT END-OF-TAPE GARBAGE
- RESTART DATA TRANSFER
- SAVE TEMPORARILY
- GET CONTROL BITS
- SKIP IF BINARY
- RECORD OK
- READ IMAGE FROM TAPE
- SKIP IF NOT END OF RECORD
- TRY TRANSFER AGAIN
- IGNORE IMAGE
- INITIALIZE TEXT LIST
- LOAD AC WITH -10
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC T4</td>
<td>SET LINE COUNTER</td>
</tr>
<tr>
<td>LAW 17676</td>
<td>LOAD AC WITH -66</td>
</tr>
<tr>
<td>DAC T5</td>
<td>SET CHARACTER COUNTER</td>
</tr>
<tr>
<td>LAC T6</td>
<td>GET FIRST CHARACTER</td>
</tr>
<tr>
<td>JMP TRPD2+3</td>
<td>ADD TO TEXT LIST</td>
</tr>
<tr>
<td>TRPD2</td>
<td>LAW 17676</td>
</tr>
<tr>
<td>DAC T5</td>
<td>SET CHARACTER COUNTER</td>
</tr>
<tr>
<td>JMS READ</td>
<td>READ IMAGE FROM TAPE</td>
</tr>
<tr>
<td>SAD =374</td>
<td>SKIP IF NOT CARRIAGE RETURN</td>
</tr>
<tr>
<td>JMP TRPD3</td>
<td>TERMINATE LINE</td>
</tr>
<tr>
<td>SNA</td>
<td>SKIP IF NOT END OF RECORD</td>
</tr>
<tr>
<td>JMP TRPD4</td>
<td>TERMINATE TRANSFER</td>
</tr>
<tr>
<td>JMS TRD4</td>
<td>ADD CHARACTER TO TEXT LIST</td>
</tr>
<tr>
<td>ISZ T5</td>
<td>INCREMENT CHAR COUNT &amp; SKIP IF DONE</td>
</tr>
<tr>
<td>JMP TRPD2+2</td>
<td>GET NEXT CHARACTER</td>
</tr>
<tr>
<td>TRPD3</td>
<td>ISZ T4</td>
</tr>
<tr>
<td>SKP</td>
<td></td>
</tr>
<tr>
<td>JMP TRPD4</td>
<td>TERMINATE TRANSFER</td>
</tr>
<tr>
<td>LAW 74</td>
<td>GET CARRIAGE RETURN</td>
</tr>
<tr>
<td>JMS TRD4</td>
<td>ADD TO TEXT LIST</td>
</tr>
<tr>
<td>LAW 75</td>
<td>GET LINE FEED</td>
</tr>
<tr>
<td>JMS TRD4</td>
<td>ADD TO TEXT LIST</td>
</tr>
<tr>
<td>JMP TRPD2</td>
<td>BEGIN NEW LINE</td>
</tr>
<tr>
<td>TRPD4</td>
<td>JMS TRD3</td>
</tr>
<tr>
<td>JMP IDLE</td>
<td></td>
</tr>
<tr>
<td>TRTC</td>
<td>LAW 17475</td>
</tr>
<tr>
<td>JMS B.T</td>
<td>TYPE IT</td>
</tr>
<tr>
<td>JMS OCTAL5</td>
<td>GET ADDRESS FROM KEYBOARD</td>
</tr>
<tr>
<td>JMP TRTC4</td>
<td>INTERPRET AS COMMAND</td>
</tr>
<tr>
<td>DPC T5</td>
<td>STORE ADDRESS</td>
</tr>
<tr>
<td>TRTC1</td>
<td>LAW 17677</td>
</tr>
<tr>
<td>JMS B.T</td>
<td>TYPE IT</td>
</tr>
<tr>
<td>LAC* T5</td>
<td>GET CURRENT CONTENT OF WORD</td>
</tr>
<tr>
<td>JMS C.B6</td>
<td>CONVERT TO 6-BIT CODE</td>
</tr>
<tr>
<td>JMS TRKT</td>
<td>TYPE CURRENT CONTENTS</td>
</tr>
<tr>
<td>LAW 17677</td>
<td>GET CODE FOR ONE SPACE</td>
</tr>
<tr>
<td>JMS B.T</td>
<td>TYPE IT</td>
</tr>
<tr>
<td>JMS OCTAL6</td>
<td>GET NEW CONTENTS FROM KEYBOARD</td>
</tr>
<tr>
<td>JMP TRTC3</td>
<td>DETERMINE NATURE OF FAILURE</td>
</tr>
</tbody>
</table>
DAC* T5
STOR E NEW CONTENTS

TRTC2 ISZ T5
INCREMENT STORED ADDRESS
LAW 17475
GET CARRIAGE RETURN, LINE FEED CODE
JMS B.T
TYPE CARRIAGE RETURN, LINE FEED
LAC T5
GET CURRENT ADDRESS
JMS C.B6
CONVERT TO 6-BIT CODE
TAD =770000
REMOVE HIGH ORDER ZERO
JMS TRKT
TYPE CURRENT ADDRESS
JMP TRTC1
TYPE CONTENTS OF CURRENT LOCATION

TRTC3 SAD =74
SKIP IF NOT CARRIAGE RETURN
JMP TRTC2
LEAVE WORD UNCHANGED
JMP TRTC
BEGIN INTERPRETATION OF NEW BLOCK

TRTC4 DAC T3
SAVE KEYBOARD CHARACTER
LAW 10
GET TELEPRINTER MASK
JMS T.R
RELEASE TELEPRINTER
JMP IDLE+6
INTERPRET CHARACTER AS COMMAND

TRTP CLA
GET END-OF-RECORD MARK
JMS PUNCH
PUNCH IT
LAW 17475
GET CARRIAGE RETURN, LINE FEED CODE
JMS B.T
TYPE CARRIAGE RETURN, LINE FEED

TRTP1 JMS ECHO
ECHO KEYBOARD CHARACTER
SAD =77
SKIP IT NOT NULL CHARACTER
JMP TRTP2
TERMINATE TRANSFER
XOR =300
SET ALPHANUMERIC CONTROL BITS
JMS PUNCH
PUNCH CHARACTER
JMP TRTP1
GET NEXT CHARACTER

TRTP2 CLA
GET END-OF-RECORD MARK
JMS PUNCH
PUNCH IT
JMP IDLE
GET NEW COMMAND

TRTT LAW 17475
GET CARRIAGE RETURN, LINE FEED CODE
JMS B.T
TYPE IT
JMS ECHO
ECHO KEYBOARD CHARACTER
SAD =77
SKIP IF NOT NULL CHARACTER
JMP IDLE
GET NEW COMMAND
JMP *-3
GET NEXT CHARACTER

TRTD LAW 17766
LOAD AC WITH -10
DAC T4
INITIALIZE LINE COUNTER
JMS TRD1

TRTD1 LAW 17475
JMS B T
LAW 17700
DAC T5

TRTD2 JMS ECHO
SAD #77
JMP TRTD4
SAD #74
JMP TRTD3
JMS TRD4
ISZ T5
JMP TRTD2

TRTD3 ISZ T4
SKP
JMP TRTD4
LAW 74
LAW 75
JMS TRD4
LAW 75
JMS TRD4
LAW 75
JMS TRD4
JMP TRTD1

TRTD4 JMS TRD3
JMP IDLE

ECHO SDC 0
JMS B K
DAC T6
XOR =777700
JMS B T
LAC T6
JMP* ECHO

PUNCH SDC 0
JMS B P
SKP
JMP* PUNCH
LAC =PUNCH1
JMS L T

INITIALIZE TEXT LIST
GET CARRIAGE RETURN, LINE FEED CODE
TYPE CARRIAGE RETURN, LINE FEED
LOAD AC WITH -64
INITIALIZE CHARACTER COUNTER
ECHO KEYBOARD CHARACTER
SKIP IF NOT NULL CHARACTER
DISPLAY TEXT LIST
SKIP IF NOT CARRIAGE RETURN
TERMINATE LINE
ADD CHARACTER TO TEXT LIST
SKIP IF END OF LINE
GET NEXT CHARACTER
INCREMENT LINE COUNT & SKIP IF DONE
TERMINATE LINE
TERMINATE TRANSFER
GET CARRIAGE RETURN CODE
ADD TO TEXT LIST
GET LINE FEED CODE
ADD TO TEXT LIST
GET LINE FEED CODE
ADD TO TEXT LIST
BEGIN NEW LINE
DISPLAY TEXT LIST
GET NEW COMMAND
GET CHARACTER FROM KEYBOARD
SAVE TEMPORARILY
PRECED WITH NULL CHARACTERS
ECHO CHARACTER ON TELEPRINTER
GET CHARACTER FOR RETURN
RETURN
SEND IMAGE TO PUNCH
PUNCH OUT OF TAPE
RETURN
GET ADDRESS OF TEXT LIST
TYPE TEXT LIST
JMP IDLE GET NEW COMMAND

PUNCH1 $DC 7
$DC 747531
$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 =READ1 GET ADDRESS OF TEXT LIST
JMS L•T TYPE TEXT LIST
JMP IDLE GET NEW COMMAND

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

TRD1 $DC 0
LAC 27 GET POINTER TO LEAF
LM• GET 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
$DC 0
LAC T3   GET REMAINING CHARACTERS
JMS TRD2 PUT IN TEXT LIST
LAC 26   GET ADDRESS OF LEVEL
SNA      SKIP IF LEVEL EXISTS
JMP TRD31 DISPLAY STORAGE EXCEEDED
LAC =TRDT GET ADDRESS OF TEXT LIST
JMS L*D  CREATE TEXT LEAF
JMP TRD31 STORAGE EXCEEDED
DAC 27   SAVE ADDRESS OF LEAF
LMQ      SET UP PARAMETER
LAC 26   GET ADDRESS OF LEVEL
JMS S.TI INSERT LEAF
JMP TRD31 STORAGE EXCEEDED
JMP* TRD3 RETURN

TRD31 LAC =**3 GET ADDRESS OF TEXT LIST
JMS L*T  TYPE DIAGNOSTIC
JMP IDLE GET NEW COMMAND
$DC 12
$DC 747577
$TEXT "NOT ENOUGH DISPLAY STORAGE"

TRD4
$DC 0
LRS 6    SHIFT CHARACTER INTO MQ
LAC T3   GET PREVIOUS CHARACTERS
LLS 6    SHIFT ALL CHARACTERS INTO AC
DAC T3   SAVE CHARACTERS
AND =770000 GET HIGH ORDER CHARACTER
SAD =770000 SKIP IF NOT NULL
JMP* TRD4 RETURN
LAC T3   GET WORD OF 3 CHARACTERS
JMS TRD2 ADD TO TEXT LIST
CLC      GET 3 NULL CHARACTERS
DAC T3   STORE NULL CHARACTERS
JMP* TRD4 RETURN

TRDT $DS 351
STC $DC 0
LAC =D
DAC DHAL+7
JMS DW
LAW STORE
DAC T1
IOT 7704
DZM* T1
ISZ T1
JMP *=2
IOT 7702
JMP* STC
GET ADDRESS OF HIGHEST ACTIVE LEVEL
REMOVE ALL NODES FROM HAL
WAIT FOR DISPLAY TO RECOVER
GET INITIAL COUNTER VALUE
SET POINTER & COUNTER
LEAVE EXTEND MODE
CLEAR STORAGE LOCATION
INCREMENT POINTER & COUNTER
CLEAR NEXT STORAGE LOCATION
ENTER EXTEND MODE
RETURN

OCTAL1 $DC 0
JMS B*T
TAD =-10
SPA
JMP *=3
TAD =10
JMP* OCTAL1
DAC T3
XOR =70
JMS B*T
LAC T3
LRS 3
LAC T4
LLS 3
DAC T4
ISZ OCTAL1
JMP* OCTAL1
GET KEYBOARD CHARACTER
MAKE NEGATIVE IF OCTAL
SKIP IF NOT OCTAL DIGIT
OCTAL DIGIT TYPED
RESTORE CHARACTER
INDICATE FAILURE
SAVE OCTAL INFORMATION
CONVERT TO 6-BIT CODE
TYPE OCTAL DIGIT
GET OCTAL INFORMATION
SHIFT DIGIT INTO MQ
GET RECORDED DIGITS
CONCATENATE NEW DIGIT
RECORD NEW WORD
INDICATE SUCCESS
RETURN

OCTAL5 $DC 0
DZM T4
JMS OCTAL1
JMP OCTAL5
JMS OCTAL1
JMP OCTAL5
JMS OCTAL1
JMS OCTAL1
CLEAR OCTAL RECORDING WORD
GET OCTAL DIGIT FROM KEYBOARD
NON-OCTAL CHARACTER TYPED
GET OCTAL DIGIT FROM KEYBOARD
NON-OCTAL CHARACTER TYPED
GET OCTAL DIGIT FROM KEYBOARD
NON-OCTAL CHARACTER TYPED
GET OCTAL DIGIT FROM KEYBOARD
JMP* OCTAL5
JMS OCTAL1
JMP* OCTAL5
ISZ OCTAL5
JMP* OCTAL5

NON-OCTAL CHARACTER TYPED
GET OCTAL CHARACTER FROM KEYBOARD
NON-OCTAL CHARACTER TYPED
INDICATE SUCCESS
RETURN

OCTAL6 $DC 0
JMS OCTAL5
JMP* OCTAL6
JMS OCTAL1
JMP* OCTAL6
ISZ OCTAL6
JMP* OCTAL6

GET 5 OCTAL DIGITS FROM KEYBOARD
NON-OCTAL CHARACTER TYPED
GET OCTAL DIGIT FROM KEYBOARD
NON-OCTAL CHARACTER TYPED
INDICATE SUCCESS
RETURN

TRKT $DC 0
DAC T1
DAC T6
LAC T1
LAC T6
JMS B*T
JMP* TRKT

SAVE HIGH ORDER DIGITS
GET LOW ORDER DIGITS
SAVE LOW ORDER DIGITS
GET HIGH ORDER DIGITS
GET LOW ORDER DIGITS
TYPE HIGH ORDER DIGITS
TYPE LOW ORDER DIGITS
RETURN

TRBK $DC 0
LAC =TRBK1
JMS OCTAL5
JMP IDLEQ
DAC TRBKL
LAW 16277
JMS B*T
JMP OCTAL5
JMP IDLEQ
CMA
TAD TRBKL
SMA
JMP IDLEQ
DAC TRBK1
LAW 15177

GET ADDRESS OF TEXT LIST
TYPE TEXT LIST
GET LOW ADDRESS FROM KEYBOARD
CANCEL COMMAND
STORE LOW ADDRESS
GET COMMA CODE
TYPE COMMA
GET HIGH ADDRESS FROM KEYBOARD
CANCEL COMMAND
FORM ONE'S COMPLEMENT
ADD LOW ADDRESS
SKIP IF PROPERLY ORDERED ADDRESSES
CANCEL COMMAND
STORE LOCATION COUNT
GET RIGHT PARENTHESIS CODE
JMS B.T
JMP* TRBK

TRBK F
SDC 3
SDC 747513
$TEXT "LOCK("

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

$END
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:

Name -- Symbolic name of the system subroutine
Entry point -- Address at which the subroutine starts
Section -- Section of the report in which the subroutine is described
Failure return -- Whether or not a failure return exists
Delay possible -- Whether or not other tasks may be executed before the subroutine returns

<table>
<thead>
<tr>
<th>Name</th>
<th>Entry point</th>
<th>Section</th>
<th>Failure return</th>
<th>Delay possible</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.FI</td>
<td>140</td>
<td>3.4.1</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>B.FO</td>
<td>142</td>
<td>3.4.1</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>B.K</td>
<td>150</td>
<td>3.4.3</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>B.P</td>
<td>146</td>
<td>3.4.2</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>B.R</td>
<td>144</td>
<td>3.4.2</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>B.T</td>
<td>152</td>
<td>3.4.3</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>C.6A</td>
<td>130</td>
<td>3.3</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>C.A6</td>
<td>132</td>
<td>3.3</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>C.B6</td>
<td>126</td>
<td>3.3</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>C.BC</td>
<td>136</td>
<td>3.3</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>C.GB</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.O</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·R</td>
<td>174</td>
<td>3·6</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>P·S</td>
<td>176</td>
<td>3·6</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>P·T</td>
<td>166</td>
<td>3·6</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Q·A</td>
<td>102</td>
<td>3·1</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Q·C</td>
<td>100</td>
<td>3·1</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Q·F</td>
<td>106</td>
<td>3·1</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Q·I</td>
<td>104</td>
<td>3·1</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>S·LBD</td>
<td>254</td>
<td>3·10</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S·LBE</td>
<td>252</td>
<td>3·10</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S·LC</td>
<td>256</td>
<td>3·10</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S·LH</td>
<td>242</td>
<td>3·10</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S·LL</td>
<td>264</td>
<td>3·10</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S·LN</td>
<td>266</td>
<td>3·10</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S·LP</td>
<td>250</td>
<td>3·10</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S·LS</td>
<td>262</td>
<td>3·10</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S·LU</td>
<td>260</td>
<td>3·10</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>S·LX</td>
<td>246</td>
<td>3·10</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>S·LY</td>
<td>244</td>
<td>3·10</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>S·TD</td>
<td>234</td>
<td>3·9</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>S·TI</td>
<td>236</td>
<td>3·9</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>S·TL</td>
<td>232</td>
<td>3·9</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>S·TR</td>
<td>240</td>
<td>3·9</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>T·A</td>
<td>116</td>
<td>3·2</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>T·F</td>
<td>114</td>
<td>3·2</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>T·L</td>
<td>122</td>
<td>3·2</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>T·P</td>
<td>112</td>
<td>3·2</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>T·R</td>
<td>120</td>
<td>3·2</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>NAME</td>
<td>ENTRY POINT</td>
<td>SECTION</td>
<td>FAILURE RETURN</td>
<td>DELAY POSSIBLE</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>---------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>T.S</td>
<td>110</td>
<td>3.2</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>T.U</td>
<td>124</td>
<td>3.2</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>X.I</td>
<td>216</td>
<td>3.8</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>X.R</td>
<td>220</td>
<td>3.8</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>X.S</td>
<td>224</td>
<td>3.8</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>X.T</td>
<td>222</td>
<td>3.8</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>X.X</td>
<td>230</td>
<td>3.8</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>X.Y</td>
<td>226</td>
<td>3.8</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>
APPENDIX C -- SUMMARY OF IOT INSTRUCTIONS

STATUS WORDS
All bits whose interpretations are not specified below are not used.

PDP-9 I/O STATUS

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

201 DATAPHONE STATUS

<table>
<thead>
<tr>
<th>BIT</th>
<th>INTERPRETATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>INTERRUPT PENDING</td>
</tr>
<tr>
<td>1</td>
<td>DATA LOST</td>
</tr>
<tr>
<td>2</td>
<td>PARITY ERROR</td>
</tr>
<tr>
<td>3</td>
<td>REQUEST TO SEND</td>
</tr>
<tr>
<td>4</td>
<td>TRANSMIT REQUEST</td>
</tr>
<tr>
<td>5</td>
<td>CLEAR TO SEND</td>
</tr>
<tr>
<td>6</td>
<td>CHECK PARITY</td>
</tr>
<tr>
<td>7</td>
<td>TEXT MODE</td>
</tr>
<tr>
<td>8</td>
<td>SET READY</td>
</tr>
<tr>
<td>9</td>
<td>TERMINAL READY</td>
</tr>
<tr>
<td>10</td>
<td>RING</td>
</tr>
<tr>
<td>11</td>
<td>CARRIER DETECTED</td>
</tr>
<tr>
<td>12</td>
<td>FRAME SIZE REGISTER BIT 0</td>
</tr>
<tr>
<td>13</td>
<td>FRAME SIZE REGISTER BIT 1</td>
</tr>
<tr>
<td>14</td>
<td>FRAME SIZE REGISTER BIT 2</td>
</tr>
<tr>
<td>15</td>
<td>FRAME SIZE REGISTER BIT 3</td>
</tr>
</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 |
| 7  | 1 -- RIGHT HAND INCREMENT BEING EXECUTED |
| 8  | LIGHT PEN ENABLED |
| 9  | BIT 0 OF Y POSITION REGISTER |
| 10 | BIT 0 OF X POSITION REGISTER |
| 11 | SCALE BIT 0 |
| 12 | SCALE BIT 1 |
| 13 | MODE BIT 0 |
| 14 | MODE BIT 1 |
| 15 | MODE BIT 2 |
| 16 | INTENSITY BIT 0 |
| 17 | INTENSITY BIT 2 |
DISPLAY INITIAL CONDITIONS

BIT INTERPRETATION

---

6 ENABLE EDGE FLAG INTERRUPT
7 ENABLE LIGHT PEN FLAG INTERRUPT
8 0 -- DO NOT DISABLE LIGHT PEN AFTER RESUMING DISPLAY
    1 -- ENABLE LIGHT PEN ACCORDING TO BIT 9
9 0 -- ENABLE LIGHT PEN AFTER FIRST DATA REQUEST AFTER RESUMING DISPLAY
    1 -- DO NOT ENABLE LIGHT PEN AFTER RESUMING DISPLAY
10 BIT 0 OF Y DIMENSION
11 BIT 1 OF Y DIMENSION
12 BIT 0 OF X DIMENSION
13 BIT 1 OF X DIMENSION
14 INTENSIFY ALL POINTS
15 INHIBIT EDGE FLAGS
16 ENABLE PUSH BUTTON INTERRUPT
17 ENABLE INTERNAL STOP INTERRUPT

BREAK FIELD LOAD PARAMETER

BIT INTERPRETATION

---

6 LOAD BREAK FIELD ACCORDING TO BITS 7-9
7 BREAK FIELD BIT 0
8 BREAK FIELD BIT 1
9 BREAK FIELD BIT 2
10 LOAD PUSH BUTTONS ACCORDING TO BITS 11-17
11 0 -- LOAD PUSH BUTTONS 0-5
    1 -- LOAD PUSH BUTTONS 6-11
12 PUSH BUTTON 0 OR 6
13 PUSH BUTTON 1 OR 7
14 PUSH BUTTON 2 OR 8
15 PUSH BUTTON 3 OR 9
16 PUSH BUTTON 4 OR 10
17 PUSH BUTTON 5 OR 11
IOT INSTRUCTIONS

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

<table>
<thead>
<tr>
<th>CODE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0002</td>
<td>ENABLE INTERRUPTS</td>
</tr>
<tr>
<td>0042</td>
<td>DISABLE INTERRUPTS</td>
</tr>
<tr>
<td>0001</td>
<td>SKIP IF CLOCK FLAG IS SET</td>
</tr>
<tr>
<td>0004</td>
<td>CLEAR CLOCK FLAG AND DISABLE CLOCK</td>
</tr>
<tr>
<td>0044</td>
<td>CLEAR CLOCK FLAG AND ENABLE CLOCK</td>
</tr>
<tr>
<td>0101</td>
<td>SKIP IF READER FLAG IS SET</td>
</tr>
<tr>
<td>0102</td>
<td>CLEAR READER FLAG, INCLUSIVE OR CONTENT OF READER BUFFER INTO AC</td>
</tr>
<tr>
<td>0104</td>
<td>SELECT READER IN ALPHANUMERIC MODE</td>
</tr>
<tr>
<td>0144</td>
<td>SELECT READER IN BINARY MODE</td>
</tr>
<tr>
<td>0201</td>
<td>SKIP IF PUNCH FLAG IS SET</td>
</tr>
<tr>
<td>0202</td>
<td>CLEAR PUNCH FLAG</td>
</tr>
<tr>
<td>0206</td>
<td>PUNCH TAPE IMAGE FROM BITS 10-17 OF AC</td>
</tr>
<tr>
<td>0244</td>
<td>PUNCH TAPE IMAGE IN BINARY MODE FROM BITS 12-17 OF AC</td>
</tr>
<tr>
<td>0301</td>
<td>SKIP IF KEYBOARD FLAG IS SET</td>
</tr>
<tr>
<td>0302</td>
<td>OR CONTENT OF KEYBOARD BUFFER INTO BITS 10-17 OF AC</td>
</tr>
<tr>
<td>0304</td>
<td>OR I/O STATUS WORD INTO AC</td>
</tr>
</tbody>
</table>
CODE    FUNCTION

0401    SKIP IF TELEPRINTER FLAG IS SET
0402    CLEAR TELEPRINTER FLAG
0406    LOAD TELEPRINTER BUFFER FROM BITS 10-17 OF THE AC
0501    OR DISPLAY PUSH-DOWN POINTER INTO BITS 6-17 OF THE AC
0502    OR BITS 1-12 OF THE DISPLAY CONTROL X POSITION REGISTER INTO BITS 6-17 OF THE AC
0601    OR BITS 3-14 OF THE DISPLAY ADDRESS COUNTER INTO BITS 6-17 OF THE AC
0602    OR DISPLAY STATUS WORD 1 INTO BITS 6-17 OF THE AC
0621    OR PUSH BUTTONS 0-11 INTO BITS 6-17 OF THE AC
0642    SKIP IF THE LIGHT PEN FLAG IS SET
0645    SET DISPLAY PUSH DOWN POINTER FROM BITS 6-17 OF THE AC
0665    SET DISPLAY INITIAL CONDITIONS FROM BITS 6-17 OF THE AC
0701    SKIP IF DISPLAY EXTERNAL STOP FLAG IS SET
0702    SKIP IF EITHER THE VERTICAL OR HORIZONTAL EDGE FLAG IS SET
0704    STOP DISPLAY (EXTERNAL)
0705    LOAD BREAK FIELD AND/OR PUSH BUTTONS FROM THE BREAK FIELD PARAMETER IN BITS 6-17 OF THE AC
0721    SKIP IF DISPLAY INTERNAL STOP FLAG IS SET
0722    SKIP IF MANUAL INTERRUPT FLAG IS SET
<table>
<thead>
<tr>
<th>CODE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>
CODE  FUNCTION
----  --------
1602  OR BITS 1-12 OF THE DISPLAY Y POSITION REGISTER INTO BITS 6-17 OF THE AC
1604  RESUME DISPLAY AFTER INTERNAL STOP
1605  INITIALIZE DISPLAY AT ADDRESS GIVEN IN BITS 6-17 OF THE AC
1622  OR DISPLAY STATUS WORD 2 INTO BITS 6-17 OF THE AC
3301  SKIP IF THE TELETYPING IS CONNECTED
3302  CLEAR ALL FLAGS
3344  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
5101  LOAD D/A CONVERTER CHANNEL #1 FROM BITS 0-11 OF THE AC
5102  LOAD D/A CONVERTER CHANNEL #2 FROM BITS 0-11 OF THE AC
5104  LOAD D/A CONVERTER CHANNEL #3 FROM BITS 0-11 OF THE AC
7701  SKIP IF IN EXTEND MODE
7702  ENTER EXTEND MODE
7704  LEAVE EXTEND MODE
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.

$EQU  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.