NOTICE: Return or renew all Library Materials! The Minimum Fee for each Lost Book is $50.00.

The person charging this material is responsible for its return to the library from which it was withdrawn on or before the Latest Date stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in the removal from the University.

To renew call Telephone Center, 333-8400.

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN
CAC Document Number 252
CCTC-WAD Document Number 7523

Research in Network Data Management and Resource Sharing

INTELLIGENT TERMINAL SOFTWARE FLOWCHARTS

October 31, 1977
The person charging this material is responsible for its return to the library from which it was withdrawn on or before the Latest Date stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN
Intelligent Terminal
Software Flowcharts

Deborah S. Brown
Betty Kasprzycki
John R. Mullen
David A. Willcox

Prepared for the
Command and Control Technical Center
WWMCCS ADP Directorate
Defense Communication Agency
Washington, D.C.

under contract
DCA100-76-C-0088

Center for Advanced Computation
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

October 31, 1977

Approved for release:

James F. Bailey, Principal Investigator
INTRODUCTION

ORGANIZATION

This document contains Nassi-Schneiderman flow charts for the software comprising the operating system and standard support package for the Intelligent Terminal operating system. Readers should refer to the Intelligent Terminal Programmer's Manual (CCTC-WAD document #7616) for a description of how these routines interact, and for descriptions of the proper usage of these routines.

The flow charts on the following pages are arranged alphabetically by routine name. In most cases there is one chart for each routine. However, some charts are too complex to be presented legibly on a single page. In each of these cases, one or more sections of the chart have been broken out and placed on a following page. If a notation such as "See ph_driver: read_type" appears in a chart, then a sub-chart labeled "ph_driver: read_type" will appear on one of the immediately following pages.

There are two implementations of the Intelligent Terminal software. One of these runs on a Digital Equipment Corporation LSI-11 minicomputer, and the other runs on Honeywell Level 6 minicomputers. Most routines are identical in the two implementations. A few routines are implemented differently on the two machines, primarily due to fundamental differences in the structure of the hardware base. Each of these routines has two charts, one for the LSI-11 version and one for the Level 6 implementation.
The diagramming technique developed by I. Nassi and B. Shneiderman (cf. SIGPLAN Notices, August 1973) provides four basic visual structures corresponding to the four basic constructs of a program:

1. process,
2. decision,
3. multi-case decision, and
4. iteration.

The Nassi-Shneiderman visual structures corresponding to these program constructs are described below.

Process

A process (meaning any computation) is represented by a box as follows:

```
compute <a>
```

The box is usually named, or some English phrase or some equation is written in the box to indicate the nature of the process or computation. The box may represent any process or computation, from the whole of an operating system to a single statement of the kind "a = b + c". An empty box represents the null process: "do nothing".

Decision

The two most common decisions are represented by the if statement and the if...else statement. These two decisions are represented as follows:

1. if statement:
2. if ... else statement:

```
< x > ?
<table>
<thead>
<tr>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>compute &lt;a&gt;</td>
<td>compute &lt;b&gt;</td>
</tr>
</tbody>
</table>
```

Multi-case Decision

The representation of a multi-case decision is a simple extension of the previous visual structure for representing simple decisions:

```
case <a> ?
<table>
<thead>
<tr>
<th>Case 1</th>
<th>Case 2</th>
<th>Case...</th>
<th>Case N</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>compute &lt;a&gt;</td>
<td>compute &lt;b&gt;</td>
<td>compute &quot;...&quot;</td>
<td>compute &lt;n&gt;</td>
<td>compute &lt;default&gt;</td>
</tr>
</tbody>
</table>
```

Iteration

The two most common forms of iteration are those with a top test and those with a bottom test. These two forms of iteration are represented as follows:

1. top test:

```
loop <test>

  compute <a>
```

2. bottom test:

```
compute <a>

loop <test>
```
Combination

The visual structures presented above may be combined to any degree to represent a computational structure, e.g.: 
**alloc(size)**

For every used entry in CORETAB.

<table>
<thead>
<tr>
<th><strong>Is the size of this entry greater than or equal to size?</strong></th>
<th><strong>YES</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NO</strong></td>
<td><strong>YES</strong></td>
</tr>
</tbody>
</table>

Remember the address of this entry.

Move the beginning of the entry to after this piece.

Decrement the size of the entry by the size of the piece being allocated.

<table>
<thead>
<tr>
<th><strong>Is size of entry now 0?</strong></th>
<th><strong>YES</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NO</strong></td>
<td><strong>YES</strong></td>
</tr>
</tbody>
</table>

Delete this empty entry by copying rest of CORETAB up one slot.

Return the remembered address.

Return -1.
Is the top of the box on a multiple of 16?

NO

Lite or erase the top, odd-sized row, using put or erase.

Is this a lite or an erase?

LITE

Do while there are at least 16 dots high to lite

Light a row 16 dots high using put

YES

Do while there are at least 16 dots high to erase

Erase a row of dots 16 dots high using erase

Is there still a partial row to do?

YES

Lite or erase the last partial row using put or erase

NO

NOTE: This version of area_lite is specific to the LSI-11 IT.

area_lite

Write an appropriately formatted message to the Z80 panel controller.

NOTE: This version of area_lite is specific to the L6 IT.
blk_alloc(drv)

For every word in the free map

Are any of the bits in this word of the free-map not set?

NO

<table>
<thead>
<tr>
<th>For every bit in this word</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is this bit off?</td>
</tr>
<tr>
<td>NO</td>
</tr>
<tr>
<td>Set this bit to 1.</td>
</tr>
<tr>
<td>blk = number of block corresponding to this bit.</td>
</tr>
</tbody>
</table>

YES

<table>
<thead>
<tr>
<th>blk out of range, or did zeroing block fail?</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
</tr>
<tr>
<td>Return (-1).</td>
</tr>
</tbody>
</table>

YES

Return (blk).

Return (-1).

blk_free(drv, blk)

blk == 0?

NO

Return (0).

YES

Is drv valid?

NO

Return (-1).

YES

Reset the bit in the free map for drive drv that corresponds to block blk.

Return (0).
Save incoming registers.

Update stack ptr and environment linkage registers.

Store environment linkage and return address registers in stack base.

Stack overflow or destroyed guardword?

NO

YES

Call error.

first_block:

while (READY_Q is empty)

Wait for entry on READY_Q.

Dequeue id of next process, and store it in ME.

Has guardword of new process stack gotten munged?

YES

NO

Call error.

Should this process be killed?

YES

NO

Call suicide.

Put return address from stack base in a register.

Restore other registers from environment linkage section.

Return to new process.
buflu(ev, blk, o, userbuf, 1, fcn)

<table>
<thead>
<tr>
<th>NO</th>
<th>YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>fen == FLUSH?</td>
<td></td>
</tr>
<tr>
<td>Is fen either READ or WRITE?</td>
<td>Has sector buffer been updated?</td>
</tr>
<tr>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Return (-1).</td>
<td>Return (0).</td>
</tr>
<tr>
<td>Mark buffer as written out.</td>
<td>Write sector to disk.</td>
</tr>
<tr>
<td>Return results of write</td>
<td></td>
</tr>
<tr>
<td>Are the 1, o, and blk parameters valid?</td>
<td>NO</td>
</tr>
<tr>
<td>Return (-1).</td>
<td></td>
</tr>
</tbody>
</table>

While there is more data to be transferred

- Is this on a sector boundary, with more than one sector to do, and either fen == WRITE or this sector is not in internal buffer?
  - NO
    - Do o and blk indicate sector that is in memory?
      - YES
        - Write out sector.
          - Did write fail?
            - NO
              - Return (-1).
            - YES
              - Set number of in memory sector to impossible value.
          - Set fen to WRITE.
            - NO
              - Return (-1).
            - YES
              - Write sector.
                - Read sector.
                  - Write failed?
                    - NO
                      - Read failed?
                        - NO
                          - Decrease the amount of data to transferred (1) by bytes per sector.
                        - YES
                          - Increase the offset in the block (o) by bytes per sector.
                        - NO
                          - Increment userbuf by bytes per sector.
          - Read in sector user wants.
            - Did read fail?
              - NO
                - Return (-1).
              - YES
                - Set ev values to describe new sector.

  - YES
    - fcn == READ?
      - NO
        - Transfer tran chars from user's buf to sector buf.
          - Mark sector as having been updated.
          - Increase offset (o) by tran.
          - Decrease length (1) by tran.
        - Return (0).
      - YES
        - Transfer tran chars from sector buf to user's buf.
          - Mark buffer as written out.
          - Increase offset (o) by tran.
          - Decrease length (1) by tran.
        - Return (0).
clear_io()

Do for each device.

YES

Is this device in use?

NO

Do n times where n is the number of elements in the requestor queue.

Deq one value from requestor queue.

NO

Is the requestor just deq'd this process?

YES

Re-queue the element.

YES

Is this process the owner of the device?

NO

Close the device.

NOTE: This version of clear_io is specific to the LSI 11 IT.

clear_io()

Do for each device.

YES

Is this device in use?

NO

Do n times where n is the number of elements in the requestor queue.

Deq one value from requestor queue.

NO

Is the requestor just deq'd this process?

YES

Re-queue the element.

YES

Is this process the owner of the device?

NO

Close the device.

YES

Does this process "own" the remote display heads?

NO

Do while rls_pnl() returns zero

NOTE: This version of clear_io is specific to the Level 6 IT.
```plaintext
<table>
<thead>
<tr>
<th>close(device_id, &amp;status)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is device_id out of range?</td>
</tr>
<tr>
<td>NO</td>
</tr>
<tr>
<td>Return error indication.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Does this process own the device?</td>
</tr>
<tr>
<td>YES</td>
</tr>
<tr>
<td>Return error indication.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Write a flush message to the handler process.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Pee the request semaphore.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Write a close message to the handler process.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Pee the request semaphore.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Flag device as being ownerless.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Are there requestors waiting for the device?</td>
</tr>
<tr>
<td>YES</td>
</tr>
<tr>
<td>Vee the requestor semaphore.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Is the closed device a disk file?</td>
</tr>
<tr>
<td>NO</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Do for each disk entry in DEV_TAB</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Does this entry define a file open on the same device as the one just closed?</td>
</tr>
<tr>
<td>YES</td>
</tr>
<tr>
<td>Set a flag.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Are there any files open on that drive (is the flag set)</td>
</tr>
<tr>
<td>NO</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Mark the drive as off_line.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Set status for return from that returned from handler process close request.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Is &quot;catastrophic&quot; bit on in status?</td>
</tr>
<tr>
<td>YES</td>
</tr>
<tr>
<td>Return error indication.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>NO</td>
</tr>
<tr>
<td>Return with no error indication.</td>
</tr>
</tbody>
</table>
```

cmp (a, b, length)

For (i = 0; i less than length; ++i)

Is a(i) != b(i)?

NO

Break.

YES

Did we fall off the end of the strings?

NO

YES

i = i - 1.

Does a(i) == b(i)?

NO

Is a(i) less than b(i)?

NO

RETURN (1).

YES

RETURN (-1).

YES

RETURN (0).
create(name, stat_ptr)

Id = open(name, 0, stat_ptr)

Did open succeed?

NO

Is name a directory?

NO

Truncate file.

Did truncate work?

NO

Close(id, stat_ptr).

Return (-1).

Update userfib for file to indicate 0 length and out-of-date directory information.

Return (id).

Temporarily terminate name at end of name of containing directory.

Open the containing directory and store results in id.

NO

Did open fail?

YES

Restore terminated name.

Return (-1).

Restore terminated name.

s = pointer into name after delimiter after containing directory.

off = 0.

For each entry in the directory.

Read the directory entry.

NO

Did the read fail?

YES

Close(id, stat_ptr).

Return (-1).

Is directory entry empty?

NO

Break.

Increment off by size of directory entry.

Seek in directory to beginning of empty directory.

Create a directory entry structure by copying the file name into it and setting the flags and lengths to indicate a zero length new file.

Allocate a disk block to use as index block.

Allocate failed?

NO

Write the directory entry structure to the directory.

Did this fail?

NO

Close(id, stat_ptr).

Return (-1).

NO

Close the containing directory.

Did this fail?

NO

Return (-1).

Id = open(name, 0, stat_ptr).

Return (id).
creep(stack_size, proc, parm, priority)

Allocate room for the stack.

Did the alloc work?

NO

YES

Get pointer to the bottom of the stack.

Store parm in the last word of the stack.

Set up a register save area just above the bottom of the stack; put dummy values in the stack and environment regs; make it look like the process was called by suicide, and zero all the other regs.

Set up the stack base: set the guardwork, priority and stack size; point the B7 entry at the register save area at the bottom of the stack, point the B5 entry at the main procedure for the process.

Put the new process on the READY_Q.

Return the value returned by alloc.

NOTE: This version of creep is specific to the Level 6.
cvb(ptr, length, value_ptr)

Value = 0.

Does the string to be converted start with a minus sign?

NO

Reset the neg flag.

YES

Set the neg flag.

Skip the minus sign.

Decrease the length by 1.

Is the first digit of the string a '0'?

NO

Base = 10.

YES

Base = 8.

For every character that is a digit, up to length chars

value = the previous value multiplied by the base and the value corresponding to the next digit of the string.

Is the neg flag set?

NO

*value_ptr = value.

YES

*value_ptr = -value.

Were there any non-digit chars in string?

NO

Return (0).

YES

Return (-1).
delete (filename, status)

Open the file.

- Did open succeed?
  - NO
    - Return error.
  - YES
    - Send a delete request to device handler.
    - Yee the request semaphore.
    - Set status from that returned by device handler.
      - Catastrophic bit on in status?
        - YES
          - Return error.
        - NO
          - Return success.
determine(name)

Is the first character of name a delimiter?

NO

YES

Strip delimiter from name.

drv = 0.

Compare name to all known device names.

p = name.

Did we find a match?

YES

NO

Compare name to names of disks.

Did we find a match?

YES

NO

For every disk

Is this disk off line?

YES

NO

Bring disk on line.

Did this work?

YES

NO

Compare name to name of disk.

Did names match?

YES

NO

Return (-1).

Does name indicate a physical device?

NO

YES

Does the name match?

NO

Return (-1).

Return proper index in DEV TAB.

Does unmatched part of name begin with a delimiter?

NO

YES

Skip over delimiter.

Find an unused slot in DEV_TAB.

Is DEV_TAB full?

NO

YES

Return (-1).

Initialize DEV_TAB entry.

Open the file.

Did fopen fail?

NO

YES

Empty DEV_TAB entry.

Return (-1).

Is this file already open?

NO

YES

Complete new DEV_TAB entry for file.

Undo new DEV_TAB slot.

Return index in DEV TAB.

Return DEV TAB index of previous entry for this file.
_deq(q_ptr)_

Become non-interruptable.

Is queue empty?

NO

Remove the oldest element from the queue.

Get the value from that element.

Return element to the list of free elements.

YES

Call error.

Did this empty the queue?

NO

Mark queue head as empty.

NO

YES

Restore initial processor priority.

Return (value).
**dir open**(ev, fib, index b)

- Init file info block for file.
- Read directory entry for file.
- Did read fail or is this not a file?
  - **NO**
    - Put impossible values in file info block.
    - Return (-1).
  - **YES**
    - Finish opening directory.
    - Position read/write pointer after directory's entry for itself.
    - Return (0).

**disable_io**(chan)

- Send stop_io channel control to indicated channel.
- Return results of this.
<table>
<thead>
<tr>
<th>open</th>
<th>close</th>
<th>read</th>
<th>write</th>
<th>peek</th>
<th>flush</th>
<th>delete</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Did close fail?</td>
<td>Did read fail?</td>
<td>Did write fail?</td>
<td></td>
<td>Did flush fail?</td>
<td>Did delete fail?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set error bits in user's status.</td>
<td>Set error bits in caller's status.</td>
<td>Set error bits in caller's status.</td>
<td></td>
<td>Set error bits in caller's status.</td>
<td>Set error bits in caller's status.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Vee requesting process.
enable_io(chan, dev)

Is this an output channel?

NO

In_out = 0.

Addr = starting address in MLCP of the CCP for this channel.

Flag = results of sending first byte of addr to the appropriate LCT byte.

Flag indicates error?

NO

Flag = results of sending second byte of addr to the appropriate LCT byte.

Flag indicates error?

NO

Flag = results of sending start_io channel control word to the channel.

Return (flag).
enq(q_ptr, value)

Is there a free queue element?

NO

Call error.

Become non-interruptable.

Remove a queue element from the free list.

Store value in the element.

YES

Is the queue empty?

NO

Add the queue element to the end of the queue element list.

Point the queue head at the new element.

Restore initial processor priority.

YES

Make the queue a one-element list.
enq_RQ(proc_id, priority)

Is there a free queue element?

NO

Call error.

Make us non-interruptable.

Remove a queue element from the free list.

Store proc_id in the value field of the element.

YES

Is the READY_Q empty?

NO

Find first entry in READY_Q whose priority value is less than priority.

Insert new element just before the old element.

Did this add new element at end of list?

NO

YES

Point READY_Q at new element.

Point READY_Q at new element.

NO

YES

Point READY_Q at new element.

Restore initial processor priority.
Compute the high end of usable memory.

Set up the stack registers.

Put address of high end of memory onto stack.

Jump to start up.

---

### erase (x, y, vector, count, flag)

Compute any shifting required to do addressing on 16-dot boundaries.

<table>
<thead>
<tr>
<th>Is it a single vector written many times?</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
</tr>
<tr>
<td>Is shifting required?</td>
</tr>
<tr>
<td>YES</td>
</tr>
<tr>
<td>Write top part of vector.</td>
</tr>
<tr>
<td>Write bottom part of vector.</td>
</tr>
<tr>
<td>NO</td>
</tr>
<tr>
<td>Write out the vector.</td>
</tr>
</tbody>
</table>

| Is shifting required?                    |
| YES                                      |
| Copy vectors into local buffer and write out 18 at a time, shifting to write lower parts of vectors. |
| NO                                       |
| Copy vectors into local buffer and write out 18 at a time, shifting to write upper parts of vectors (or all of vectors). |

Rsrv_pnl.

Rls_pnl.

**NOTE:** This version of erase is specific to the Level 6 IT.
erase (X, Y, vector, count, flag)

Compute the amount of shifting required to do the erases on 16-dot boundaries

Set up panel registers for erase

Do count times

Is it an erase of a single vector?

Yes

data = vector

No

data = next word pointed at by vector

Is shifting required?

Yes

Erase the lower part of vector "data" on screen

No

Erase upper part (or all of) vector "data" on the screen

Increment the X panel address

NOTE: This version of erase is specific to the LSI-11 IT.
**error(err_no)**

- Become non-interruptable.
- Print an error message, depending on the value of `err_no`.
- `halt()`.

**fclose(ev, fib)**

- Flush the in_memory index block for the file.
- Has the file gotten bigger?
  - NO
    - Update directory to indicate new size.
  - YES
    - Flush the in_memory data buffer.
    - Mark `ev` and `fib` as unused.
- Return (0).
fdelete(ev, fib)

Flush in-memory copy of file's index block.

Did flush fail, or is this the root directory?

NO

Return(-1).

YES

Mark all blocks used by the file as free.

Did truncate fail?

NO

YES

Restore in-memory copy of freemap.

Return(-1).

Mark index block for file as free.

Flag = false.

Flag = results of opening directory containing file.

Is flag set?

NO

YES

Flag = results of seeking to directory entry for file.

Is flag set?

NO

YES

Read directory entry for file.

Did read fail?

NO

YES

Flag = true.

Is flag set, or is this a protected file or a directory?

NO

YES

Restore in-memory copy of freemap.

Return(-1).

Mark directory entry on disk for file as empty.

Did write fail?

NO

YES

Restore in-memory copy of freemap.

Return(-1).

Clean up in-memory data buffer.

Did this work?

NO

YES

Return(-1).

Return(0).
first_block

Jump into the middle of block - to the point where it picks up a process from the ready queue.
Set count of bytes transferred (tran) to 0.

\[ fcn = \text{READ} \]  
\[ \text{NO} \]

\[ fcn = \text{WRITE and} \]
\[ fcn = \text{FLUSH!} \]  
\[ \text{NO} \]

\[ \text{Are we at the end of the file?} \]  
\[ \text{NO} \]

\[ \text{Return (0).} \]
\[ \text{YES} \]

\[ \text{Will requested length go past end of file?} \]  
\[ \text{NO} \]

\[ \text{Adjust 1 to be amount of data before end of file.} \]  
\[ \text{YES} \]

While the transfer is not complete, or unconditionally if \( fcn = \text{FLUSH!} \):

\[ \text{Is the portion of the index block that describes this block currently in memory?} \]  
\[ \text{NO} \]

\[ \text{Does the in-memory portion of the index block need to be saved or is this a FLUSH?} \]  
\[ \text{NO} \]

\[ \text{Does saving the freemap fail?} \]  
\[ \text{NO} \]

\[ \text{Return (-1).} \]
\[ \text{YES} \]

\[ \text{Does saving the portion of the index block fail?} \]  
\[ \text{NO} \]

\[ \text{Return (-1).} \]
\[ \text{YES} \]

\[ fcn = \text{FLUSH!} \]  
\[ \text{NO} \]

\[ \text{Return (0).} \]
\[ \text{YES} \]

\[ \text{Are we already at the end of the biggest that the file can be, or did reading the next subset of the index block fail?} \]  
\[ \text{NO} \]

\[ \text{Return (-1).} \]
\[ \text{YES} \]

Mark \( fib \) to indicate new subset of index block.

Assume will want to do a bufio function later.

\( fib = fblk = \) number of disk block containing current portion of file.

\[ \text{Has this block never been allocated?} \]  
\[ \text{NO} \]

\[ fcn = \text{READ?} \]  
\[ \text{YES} \]

\[ \text{Change assumed function to zero sim.} \]
\[ \text{Allocate a block to use.} \]
\[ \text{Did the allocate fail?} \]  
\[ \text{YES} \]

\[ \text{Return (-1). Note that index subset has been modified.} \]
\[ \text{NO} \]

\[ \text{Tranlen = minimum of 1 and bytes to transfer in this block.} \]
\[ \text{Do the appropriate function (either huf in or zero sim).} \]
\[ \text{NO} \]

\[ \text{Did it work?} \]  
\[ \text{YES} \]

\[ \text{Return (-1).} \]
\[ \text{NO} \]

Update user's buffer pointer, l, tran, and fib entries by tranlen.

\[ fcn = \text{WRITE?} \]  
\[ \text{YES} \]

\[ \text{Was the file extended?} \]  
\[ \text{YES} \]

\[ \text{Update file size in fib.} \]
\[ \text{Note that directory entry for file needs to be updated.} \]
\[ \text{NO} \]

\[ \text{Return (tran).} \]
**fixup(reserve_size)**

Save initial values of R5, R6, and memory location 4.

Put the address of trap_catcher in memory location 4.

For (RO = 0; RO =+1024)

Try accessing memory location whose address is in RO (will goto trap_catcher when try to access a non-existent location).

**Count_down:**
Decrement RO and try accessing that location.

Decrement RO by 1.

Subtract reserve_size from RO.

Copy RO to R5.

R6 = RO - the initial difference between R6 and R5.

Restore the initial value in memory location 4.

Make interruptable.

Clean up the stack.

Return to caller.

**Trap_catcher:**
Put the address of count_down on the stack.

Return from trap.
**flush(device_id, st_ptr)**

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Is device_id out of range?</strong></td>
<td><strong>Does this process own the device?</strong></td>
</tr>
<tr>
<td>Return error.</td>
<td>Return error.</td>
</tr>
<tr>
<td></td>
<td><strong>YES</strong></td>
</tr>
<tr>
<td><strong>Send a &quot;flush&quot; message to the device handler process.</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Pee the request semaphore.</strong></td>
<td><strong>Is catastrophic bit on in status?</strong></td>
</tr>
<tr>
<td></td>
<td><strong>NO</strong></td>
</tr>
<tr>
<td></td>
<td>Return error.</td>
</tr>
<tr>
<td></td>
<td>Return no error.</td>
</tr>
</tbody>
</table>
Loop forever

Open file whose index block is root as a directory.

Did dir_open fail?

NO

YES

Return(-1).

NO

Have we reached the end of the name?

YES

Set file offset in fih to 0.

Return(0).

NO

For (off = size of a dir_entry structure; off = size of dir_entry structure)

Read next directory from directory.

Did read fail?

NO

YES

Return(-1).

NO

Is this directory entry empty?

YES

Continue.

NO

Did they match at all?

YES

Did they match up to a delimiter in name?

NO

Is this entry a directory?

YES

NO

Return(-1).

Update name to point to character after delimiter.

Update root to be index block for this entry.

Break.

NO

Did they match up to the end of name?

YES

Is this entry a directory?

NO

Update root to be index block for this entry.

Update name to point to end of name.

Break.

Return results of doing xopen on this entry.
free(size, addr)

Find the first entry in CORETAB whose address is less than addr. This may be the entry after the last used entry.

<table>
<thead>
<tr>
<th>Is the preceding entry contiguous with the one being freed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
</tr>
<tr>
<td>Is the entry being freed contiguous with the succeeding one?</td>
</tr>
<tr>
<td>NO</td>
</tr>
<tr>
<td>Copy the succeeding entries down a slot.</td>
</tr>
<tr>
<td>Insert the piece to be freed into the table.</td>
</tr>
<tr>
<td>Move size of this entry to size of previous entry.</td>
</tr>
<tr>
<td>Does this make previous entry contiguous with this one?</td>
</tr>
<tr>
<td>NO</td>
</tr>
<tr>
<td>Add rest of table up one slot to delete this entry.</td>
</tr>
</tbody>
</table>
fseek(fib, off, type)

<table>
<thead>
<tr>
<th>Blocks</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>b = number of blocks.</td>
<td>NO</td>
</tr>
<tr>
<td>o = 0.</td>
<td>Is position absolute?</td>
</tr>
<tr>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Treating off as a signed integer, convert it to a number of blocks b and an offset o into the last block.</td>
<td>Treating off as an unsigned integer, convert it to a number of blocks b and offset o in the last block.</td>
</tr>
<tr>
<td>o less than 0?</td>
<td></td>
</tr>
<tr>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>b = -1 Add block size to o to make it positive.</td>
<td></td>
</tr>
</tbody>
</table>

Switch(type)

<table>
<thead>
<tr>
<th>0,3</th>
<th>1,4</th>
<th>2,5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set the read/write pointer for this file to block of b, offset of o.</td>
<td>Move read/write pointer to end of file.</td>
<td>Add block offset of b and byte offset of o to read/write pointer.</td>
</tr>
</tbody>
</table>

Is byte offset more than one block big?

<table>
<thead>
<tr>
<th>NO</th>
<th>YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subtract block size from byte offset.</td>
<td>Increment block count by one.</td>
</tr>
</tbody>
</table>
ftrunc(ev, fib)

Write out the index block for this file.

Max = number of blocks in file.

<table>
<thead>
<tr>
<th>Has the last block been allocated?</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
</tr>
<tr>
<td>YES</td>
</tr>
</tbody>
</table>

Decrement max.

Indx = number of disk block that is disk block for this file.

<table>
<thead>
<tr>
<th>Are there any data blocks in this file?</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
</tr>
<tr>
<td>YES</td>
</tr>
</tbody>
</table>

Return(0).

For every block in file:

<table>
<thead>
<tr>
<th>Read in the number of disk block for this block.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Did read work?</td>
</tr>
<tr>
<td>NO</td>
</tr>
<tr>
<td>YES</td>
</tr>
</tbody>
</table>

Return(-1).

Free the disk block with this number.

Zero the index block.

Set size of file to 0.

Set necessary flags in fib.

Return(0).
get_charset()

Return (CS_ID)

get_cursor(x_ptr, y_ptr)

Convert current cursor position from dots to characters, using the size of characters in the current char set.

Assign dimensions into *x_ptr and *y_ptr.

NOTE: This version of get_cursor is specific to the LSI-11 IT.

get_cursor (x_ptr, y_ptr)

Read cursor from the display head.

Convert values returned to character offsets.

NOTE: This version of get_cursor is specific to the Level 6 IT.
get_env (env_ptr)

Call get_charset and save pointer to charset.

Call get_cursor and save cursor values.

Call get_pg and save page descriptor.

Is this the Level 6 version?

YES

Call get_pnl and save panel flags.

NO

get_page_size(w_ptr, h_ptr)

Convert current page size from dots into characters, using the size of characters in the current char set.

Assign values into *w_ptr and *h_ptr.

get_pg(pg_ptr)

Copy values in PAGE to structure pointed to by pg_ptr.

get_size_chars(w_ptr, h_ptr)

Copy character size from CS_ID structure to *w_ptr and *h_ptr.
halt

Do forever

Execute a HLT instruction.

Return to caller.

NOTE: The procedure will never return by itself. The user will have to externally change the PC to get out of the HLT-loop.

i_freemap(drv)

Is drive number (drv) out of range?

NO

YES

Return(-1).

Read the freemap from disk.

Return results of read.

init_ccb(chan, buf, buf_size, cntrl)

Flag = results of doing iold to set up buffer.

Flag indicates error?

NO

YES

Flag = results of sending cntrl to CCB control word.

Return(flag).
get_token(buf_ptr, tok_ptr, delim_ptr)

Get index of first char in buf that is not a delimiter, and store it in count.

Are all the characters delimiters?

NO

Null terminate buffer.

Return (0).

YES

For every char in buffer, starting with first delimiter

For every char in delimiter string

Char in buffer == char in delimiter?

NO

Copy character from buffer to token.

Increment count.

YES

Terminate outermost for_loop.

Null terminate token buffer.

Return count of characters in token and count of delimiter characters scanned at beginning.
init_drive(drv)

Is the drive number (drv) out of range?

NO

Return (-1).

YES

Set up dib entry for drive to indicate that drive is open.

Open root directory of drive.

Did open fail?

NO

Return (-1).

YES

Read entry in root directory describing itself.

Read freemap for disk.

Did either of the reads fail?

NO

Indicate disk is off line.

Return (-1).

YES

Return (0).
init_fib(fib, index_b, off)

Assign appropriate values to elements of file information block.

Return (0).

io_init()

Put the address of each device’s handler’s input queue into DEV_TAB.

Do for each device

Put the address of this device’s request semaphore into the request block for this device.

init_pnl()

Call Z80_LD to get a pointer to the Z80 microcode

Set up so that writes go to all remote display heads

Is there an alternate charset loaded?

YES

Flag the charset as not loaded

NO

Has the microcode already been loaded?

NO

Write the microcode to the Z80 panel controller

Free the space occupied by the microcode in the Level 6 memory.

YES

Initialize the Z80’s internal variables.
index(in_str, of_str)

lth_of = length of "of" string.

Point p1 at in_string and p2 at of_str.

For every character in "in" string

<table>
<thead>
<tr>
<th>Does char in &quot;in&quot; string == char in &quot;of&quot; string?</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
</tr>
<tr>
<td>YES</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Is next char of &quot;of&quot; string ending null?</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>NO</td>
</tr>
<tr>
<td>YES</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Point p1 and p3 into &quot;in&quot; string, where match would start to end here.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Compare the characters pointed at by p2 and p1, ending when they differ, or when hit end of &quot;of&quot; string.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Did all chars match?</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>NO</td>
</tr>
<tr>
<td>YES</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>p1 = p3.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Return index into in_string of beginning of match.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>p2 = beginning of of_str.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Increment p1.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Return (-1).</td>
</tr>
</tbody>
</table>
b_driver()

Enable keyboard input.

Do forever

Read a word from the input queue.

Is the word odd?

YES

Request_code = "data."

NO

The word is a pointer to a request block containing
the request code.

Is high byte of the word a CR?

YES

NO

Input data character is newline.

Input data character is in the
high byte of the word.

Switch on request code.

Error: Reject the
request.

peek | close | open | flush | setmode | data | read | default
---|---|---|---|---|---|---|---
Put count of
characters in
buffer into
request.block.

Set "open"
flag off.

Set "open"
flag on.

Vee the sema-
phore.

Vee the sema-
phore.

(See kb_driver: flush_type)

(See kb_driver: setmode_type)

(See kb_driver: data_type)

(See kb_driver: read_type)

NOTE: This version of kb_driver is specific to the LSI 11 IT.
Enable keyboard input.

Do forever

<table>
<thead>
<tr>
<th>Read a word from the input queue.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is it a known special code?</td>
</tr>
</tbody>
</table>

**YES**

The word read is the request code.

**NO**

The word read is a pointer to a request block containing the request code.

---

**NOTE:** This version of `kb_driver` is specific to the Level 6.
**kb_driver: data type**

1. **Is the keyboard open?**
   - **YES**
     - **Is there a saved up read?**
       - **YES**
         - Put input character into user's buffer.
       - **NO**
         - **Ring bell.**
   - **NO**
     - **Put the character into the buffer.**

2. **Are we echoing?**
   - **YES**
     - **Put the character into the buffer.**
   - **NO**
     - **Print the character using put_ascii.**

*This is not a procedure. It is one case in kb_driver.*

**kb_driver: flush_type**

1. **Empty the input buffer.**
2. **Is a read pending?**
   - **YES**
     - Set data length to zero and reject the saved read request.
   - **NO**
     - **Vee the user's semaphore.**

**NOTE:** This is not a procedure. It is one case in kb_driver.

**kb_driver: read_type**

1. **Is the buffer empty?**
   - **NO**
     - Compute length to give the user.
     - **Save the read request.**
     - Copy characters into user's buffer.
     - **Vee user's semaphore.**
   - **YES**
     - **This is not a procedure. It is one case in kb_driver.**

**kb_driver: setmode_type**

1. **Switch on type of setmode.**

   - **echo_on**
   - **set_xy**
   - **echo_off**
   - **default**

   - **Set "echoing" flag on.**
   - **Set "echoing" flag off.**
   - **Set "reject request" in request block.**
   - **Set x - y coordinates.**
   - **Vee user's semaphore.**

*This is not a procedure. It is one case in kb_driver.*
**kill(proc_id)**

Does id indicate a valid process?

- **NO**
  - Return(-1).
  - Set the guardword to indicate that the process should be killed the next time it is scheduled.
  - Return(0).

- **YES**

**ld_cs()**

CS ID != CS_LAST (i.e. has the character set been changed)?

- **YES**
  - Is this charset resident in the display head?
    - **YES**
      - Is there an alternate charset already loaded?
        - **YES**
          - Flag the old alternate cs that it is no longer loaded.
        - **NO**
          - Rsv_pnl.
    - **NO**
      - Allocate space for the effector table as "variable" number 3.
      - Write the effector table.
      - Allocate space for the characters as "variable" number 2.
      - Write the characters (the bytes of each word must be swapped before writing them).
      - Flag the current character set as the currently loaded alternate.
      - Rls_pnl.
  - Write out a new charset descriptor (specifying the size of the characters and which "variables" to use for the effector and character tables).

**ld_page()**

Has the page been changed since last time we were here?

- **YES**
  - Format a page descriptor for the Z80.
  - Write it out.

- **NO**
ldiv(hi, lo, d)

Copy the pair (hi, lo) to registers R0 and R1.

Do a double word divide on (R0, R1) by d.

Return the quotient.

NOTE: This version of ldiv is specific to the LSI 11 IT.

ldiv(hi, lo, d)

Copy the pair (hi, lo) to registers R6 and R7.

Do a double word divide on (R6, R7) by d.

Return the quotient.

NOTE: This version of ldiv is specific to the L6 IT.
\texttt{ln\_xpand(out, in, p1, p2, p3, \ldots)}

<table>
<thead>
<tr>
<th>Length = 0.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{Next_parm} = address of first parameter (p1).</td>
</tr>
</tbody>
</table>

While there is still something in the input format specification

<table>
<thead>
<tr>
<th>Is the next char in the input format specifier a parameter replacement indicator?</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
</tr>
<tr>
<td>Copy the char from the input buffer to the output buffer.</td>
</tr>
<tr>
<td>Move the input buffer over one character.</td>
</tr>
<tr>
<td>Increment length by one.</td>
</tr>
</tbody>
</table>

Null terminate the output string.

Increment length to include the trailing null.

Return length.
```plaintext
lrem(hi, lo, d)

Copy the pair (hi, lo) to registers R0 and R1.

Do a double word divide on (R0, R1) by d.

Return the remainder.

NOTE: This version of lrem is specific to the LSI 11 IT.
```

```plaintext
lrem(hi, lo, d)

Copy the pair (hi, lo) to registers R6 and R7.

Do a double word division on (R6, R7) by d.

Return the remainder.

NOTE: This version of lrem is specific to the L6 IT.
```
mfps()

Return the value of the PSW.

NOTE: This version of mfps is specific to the LSI 11 IT.

mfps()

Pick up the system status word.

Decode the current level number from it.

Is this level greater than the non-interruptable level for the L6?

<table>
<thead>
<tr>
<th>NO</th>
<th>YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>lev = the non-interruptable status word for LSI 11.</td>
<td>lev = 0.</td>
</tr>
</tbody>
</table>

Return (lev).

NOTE: This version of mfps is specific to the L6 IT.
**mk_page(p_ptr, left, bottom, width, height)**

Copy parameters to corresponding positions in structure pointed to by p_ptr.

**mtps(prio)**

- **Does prio have the LSI 11 non-interruptable bit set?**
  - **NO**
    - Pick up the hardware status word.
    - Decode the current level.
    - Are we currently interruptable?
      - **NO**
        - Do a LEV that resets the activity bit for the non-interruptable level and schedules the next ready level.
      - **YES**
        - Leave us at our current level.
  - **YES**
    - Do a LEV that will leave the activity bit set for the current level, and switch us to be working at the non-interruptable level.

**NOTE:** This version of mtps is specific to the L6 IT.

**mtps(prio)**

Copy prio into the PSW.

**NOTE:** This version of mtps is specific to the LSI11 IT.
open(name, flag & status)

Determine(name).

Is device name ok?

NO

Return error.

YES

Does some other process own this device?

NO

Did the caller ask to wait for the device?

NO

Is there a requestor queue?

YES

Allocate space for a semaphore.

NO

Allocation failed?

YES

Out-of-memory error.

NO

Initialize the semaphore.

Fro the requestor semaphore.

Are there other requestors waiting for the device?

NO

FREE space for requestor semaphore.

YES

Make this process owner of the device.

Free any outstanding fee's on the delvce semaphore.

Send a flush request to the handler process.

FREE the device semaphore.

Send an open request to the handler process.

FREE the device semaphore.

Set status for caller from status returned from device handler for open request.

Is catastrophic bit on in status?

YES

CLOSE the device.

Return success.

NO

Return error.
The function `parm_xpand(how, what, count)` is described as follows:

Set word pointed to by count to be the number of characters in the format description for this expansion.

<table>
<thead>
<tr>
<th>type 'g'</th>
<th>type 'c'</th>
<th>type 'd'</th>
<th>type 'l'</th>
<th>type 'u'</th>
<th>type 'o'</th>
<th>type 'x'</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pick up char ptr pointed to by what.</td>
<td>Copy char pointed to by what to workspace buffer.</td>
<td>Is parm less than 0?</td>
<td>NO</td>
<td>YES</td>
<td>Put a 'o' in the buffer.</td>
<td>Put an 'x' in the buffer.</td>
<td>Copy the type character into the buffer.</td>
</tr>
<tr>
<td>Increment #what to point to next parameter in caller's list.</td>
<td>Put a ',' in buffer.</td>
<td>Do a str_num on parm, with a base of 10, into buffer.</td>
<td>Copy parm into temp.</td>
<td>Do a str_num on the parm, with a base of 8, into the buffer.</td>
<td>Do a str_num on the parm, with a base of 10, into the buffer.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return char ptr.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Null terminate the string have built in the buffer.

Move #what to point to next parameter in caller's list.

Return a pointer to the beginning of the workspace buffer.
**pause()**

- Enqueue this process on the READY Q.
- Call block().
- Return.

**pathname(your, dir)**

1. Remember where your string begins.
2. Compare characters in your and dir strings up to the end of dir and as long as the two strings match.
3. Did the strings differ before the end of the dir string, or did the comparison end pointing at something other than a NUL or delimiter in your string?
   - **NO**
     - Return pointer to beginning of your string.
   - **YES**
     - Return pointer to your string where comparison ended.
**pes(sem)**

1. Become non-interruptable.
2. Decrement the count for this semaphore.
3. Is count less than 0?
   - NO: Enqueue this process on the queue for this semaphore.
   - YES: Call block().
4. Restore initial priority.

**peek(device_id, &status)**

1. Is device_id out of range?
   - YES: Return error.
   - NO: Does this process own the device?
     - NO: Return error.
     - YES: Send peek request to device handler.
2. Peek the request semaphore.
3. Set status from that returned by handler process for peek request.
4. Is "catastrophic" bit on in status?
   - YES: Return error.
   - NO: Return success.
```
ph_driver()

Ph_available = true.

Saved_read = active_write = 0.

While (true)

Read an element from input queue.

Is this a simple request?

NO

Blk_ptr = what read from queue.

Command = command in request block.

YES

Command = what read from queue.

Switch (command)

open close read write peek flush bar overflow write failed done status_change default

Calculate amount of data in our buffer, including trailing null if buffer is not empty.

Assign the value to caller's data_len.

Set catastrophic bit in writer's status.

Set writer's data_len to amount of data actually transmitted.

Set reject req bit in caller's status.

Set active_write to 0.

Set writing process.

Vee requesting process.

Vee requesting process.

Vee writing process.

Set status_change.

NOTE: This version of ph_driver is specific to the level of El.
```
Flag line as up and input buffer empty

Do "forever"

<table>
<thead>
<tr>
<th>Is it a known special code?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
</tr>
</tbody>
</table>

Request code is decoded from value read from queue

Value read is a pointer to a request block with the request code

---

### Do a switch on the request code

<table>
<thead>
<tr>
<th>open or close</th>
<th>read</th>
<th>write</th>
<th>peek</th>
<th>flush</th>
<th>data</th>
<th>done</th>
<th>up</th>
<th>down</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **read**
  - Is phone available
    - Yes: Set length equal to number of chars available, including trailing null, if buffer not empty
    - No: Start a write
  - Reject user's request

- **write**
  - Is phone available
    - Yes: Assign the value to user's data_len
    - No: No data and phone not available

- **peek**
  - No data and phone not available
    - Yes: Set error flag for user
  - Vee user's semaphore

See Ph_driver:flags_type

- **flush**
  - Turn off output

- **data**
  - Vee saved write's semaphore

- **done**
  - Set phone available flag

- **up**
  - Reset phone available flag

- **down**
  - Reject user's request

- **default**

See Ph_driver:flags_type

- **See Ph_driver:flags_type**
  - No data and phone not available
  - Yes: Set error flag for user

Vee user's semaphore
**ph_driver: data**

While there is a pending read, and there is more data in our buffer.

- Copy the next character from our buffer into the reader's.
- Put a null after char just copied.
- Increment the reader's count of data in the buffer.

Is the buffer full, or was the last char copied a newline?

**NO**

- Vee reading process.
- Set saved_read to 0.

**YES**

- Increment the next character index by 1, module our buffer size.

*This is not a procedure. It is one case in ph_driver.*

**ph_driver: close_type**

Is the phone available?

**NO**

- Flag = results of disabling phone input.

**YES**

Is phone not available, or is flag set?

**NO**

- Vee requesting semaphore.

**YES**

- Set catastrophic and reject_req bits in caller's status word.

*This is not a procedure. It is one case in ph_driver and is specific to the Level 6 IT.*
ph_driver: flush_type

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Stop any current output operation</td>
</tr>
<tr>
<td>2</td>
<td>Empty the input buffer</td>
</tr>
<tr>
<td>3</td>
<td>Is there a saved read?</td>
</tr>
<tr>
<td></td>
<td>YES</td>
</tr>
<tr>
<td>4</td>
<td>Reject the saved read</td>
</tr>
<tr>
<td>5</td>
<td>Is there a saved write?</td>
</tr>
<tr>
<td></td>
<td>YES</td>
</tr>
<tr>
<td>6</td>
<td>Reject the saved write</td>
</tr>
<tr>
<td></td>
<td>Is the phone line down?</td>
</tr>
<tr>
<td></td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>Set error flag in request block.</td>
</tr>
<tr>
<td></td>
<td>Vee user's semaphore</td>
</tr>
</tbody>
</table>

*NOTE: This is not a procedure. It is one case in ph_driver, and is specific to the LSI-11 IT.*
ph_driver: flush_type*

Reset PH_next and PH_free to 0.
Set io error flag to false.

Is there a saved read?

NO
Set aborted bit in reader's status word.
Vee reading process.
Reset saved_read to 0.

YES

Is there an active write?

NO
Flag = results of disabling output.

Flag = results of getting status word for CCB just completed.

Flag = results of getting count of characters not sent.
Set the writer's data count to the number of characters sent.
Set aborted bit in writer's status word.

Flag set?

NO

Yes

Is flag set?

NO

Set catastrophic bit in writer's status.
Vee writing process.
Reset active_write to 0.

YES

Phone unavailable or flag set?

NO

Set catastrophic bit in caller's status.
Vee requesting process.

*This is not a procedure. It is one case in ph_driver and is specific to the Level 6 IT.*
ph_driver: open_type*

Is the phone available?

**NO**
Reset PH_next and PH_free to 0.

**YES**
Is the carrier detect bit on?

**NO**
Reset phone available flag.
Set error flag to -1.

**YES**
Is flag set?

**NO**
Flag = results of resetting CCB list for phone input.

For each buffer used by phrint, as long as flag == 0.

Flag = results of initializing CCB to use this buffer.

**YES**
Reset buffer index used by phrint.

Is flag set?

**NO**
Flag = results of enabling input for phone.

**YES**
Is phone not available, or is flag set?

**NO**

**YES**
Set catastrophic and reject_request bits in user's status word.

Vee requesting process.

*This is not a procedure. It is one case in ph_driver and is specific to the Level 6 IT.
### ph_driver: read_type*

<table>
<thead>
<tr>
<th>Block</th>
<th>Condition</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize blk_ptr &gt; data_len to 1.</td>
<td>NO</td>
<td>Is there data in our buffer?</td>
</tr>
<tr>
<td></td>
<td>YES</td>
<td>Transfer data from our buffer to the user's until the user's buffer is full, or we are out of data or we hit a new line character.</td>
</tr>
<tr>
<td>Point out_ptr at user's buffer.</td>
<td>NO</td>
<td>Is the phone available?</td>
</tr>
<tr>
<td></td>
<td>YES</td>
<td>Set catastrophic and reject_req bits in caller's status word.</td>
</tr>
<tr>
<td></td>
<td>NO</td>
<td>Null terminate the string.</td>
</tr>
<tr>
<td></td>
<td>YES</td>
<td>Set saved_read = blk_ptr.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set blk_ptr-data_len to the number of characters transferred, including the trailing null.</td>
</tr>
<tr>
<td>Is saved_read zero?</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>YES</td>
<td>Set saved_read = blk_ptr.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Is user's buffer not full, and is the last char not a newline?</td>
</tr>
<tr>
<td></td>
<td>NO</td>
<td>Vee requesting process.</td>
</tr>
<tr>
<td></td>
<td>YES</td>
<td></td>
</tr>
</tbody>
</table>

*This is not a procedure. It is one case of ph_driver, and is specific to the level 0 I/O.*
**ph_driver: status_change***

Complement the phone available flag.

<table>
<thead>
<tr>
<th>Phone available?</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
</tr>
<tr>
<td>YES</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pending read?</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
</tr>
<tr>
<td>YES</td>
</tr>
</tbody>
</table>

Set catastrophic and aborted bits in reader's status.

Vee reading process.

Set saved_read to 0.

<table>
<thead>
<tr>
<th>Active write?</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
</tr>
<tr>
<td>YES</td>
</tr>
</tbody>
</table>

Turn off output.

Input status for CCB just completed.

Input number of chars not transmitted.

Set catastrophic and aborted bits in writer's status.

Set writer's data_len to count of chars actually sent.

Vee writing process.

Reset active_write to 0.

*This is not a procedure. It is one case in ph_driver and is specific to the Level 6 IT.*
**ph_driver: write_type**

<table>
<thead>
<tr>
<th>NO</th>
<th>YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is phone available?</td>
<td>Set active_write = blk_ptr.</td>
</tr>
<tr>
<td>Flag = results of sending reset CCB list command to output phone channel.</td>
<td></td>
</tr>
<tr>
<td>Is flag set?</td>
<td>NO</td>
</tr>
<tr>
<td>Flag = results of initializing an output CCB to use caller's buffer.</td>
<td></td>
</tr>
<tr>
<td>Is flag set?</td>
<td>NO</td>
</tr>
<tr>
<td>Flag = results of restarting phone output.</td>
<td></td>
</tr>
<tr>
<td>Is phone unavailable or is flag set?</td>
<td>YES</td>
</tr>
<tr>
<td>Reset active_write to 0.</td>
<td></td>
</tr>
<tr>
<td>Set catastrophic and reject_req bits in caller's status.</td>
<td></td>
</tr>
<tr>
<td>Vee calling process.</td>
<td></td>
</tr>
</tbody>
</table>

*This is not a procedure. It is one case in ph_driver and is specific to the Level 6 IT.*
phprint()

Get the status for the CCB that just completed.

Does status indicate change in the data set status?

NO

Send status_change message to phone process.

Return.

YES

For every character in our data buffer.

Is there room in PH_buf for this char?

NO

Send overflow message to phone process.

Break.

YES

Copy character to PH_buf.

Was PH_buf previously empty?

NO

Send data message to phone process

YES

Increment PH_free by 1, modulo the size of PH_buf.

Initialize a new CCB using the buffer just emptied.

Increment our buffer index by 1 modulo the number of buffers we have.

NOTE: This version of phprint is specific to the Level 6 IT.
phxint()

<table>
<thead>
<tr>
<th>Input status for CCB that just finished.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input CCB range to get count of characters not transmitted and store in global PH_count.</td>
</tr>
</tbody>
</table>

**CCB status indicates an error?**

<table>
<thead>
<tr>
<th>NO</th>
<th>YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send message to phone process that the write has finished.</td>
<td>Send message to phone process that the write failed.</td>
</tr>
</tbody>
</table>

**NOTE:** This version of phxint is specific to the Level 6 IT.
pp_read (code, buffer, length)

Rsrv_pnl().

Select any active panel to read from.

Do forever

<table>
<thead>
<tr>
<th>Read controller interrupt level.</th>
<th>Make processor non-interruptable.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is a keyboard or touch panel read in progress?</td>
<td></td>
</tr>
<tr>
<td>YES</td>
<td>Set flag to make interrupt routine wake us up.</td>
</tr>
<tr>
<td>NO</td>
<td>Make processor interruptable.</td>
</tr>
<tr>
<td>YES</td>
<td>280 interrupts enabled?</td>
</tr>
<tr>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>NO</td>
<td>Set flag for interrupt routine.</td>
</tr>
<tr>
<td>YES</td>
<td>While read is still in progress</td>
</tr>
<tr>
<td>NO</td>
<td>Make the processor interruptable.</td>
</tr>
<tr>
<td>YES</td>
<td>Null.</td>
</tr>
<tr>
<td>NO</td>
<td>Make the processor interruptable.</td>
</tr>
<tr>
<td>YES</td>
<td>Pause.</td>
</tr>
</tbody>
</table>

Return.
pp_write() (code, buffer, count)
Rsrv_pnl.

Do for each display head.

Turn off "must wait" for this head.

Is this panel selected?

YES

NO

Do "forever"

Read display head interrupt level.

Disable interrupts.

Is a kb or tp read in progress from this head?

YES

NO

Set flag to make interrupt routine wake us up.

Attempt to start the write.

Write started?

YES

NO

Panel interrupts enabled?

Enable interrupts.

Panel interrupts enabled?

While read still in progress

NO

NO

YES

Enable interrupts.

Set flag so interrupt routine will wake us up.

Read panel status.

Turn on "must wait" flag for this head.

Do until success

Enable interrupts to cp.

Break.

NO

Enable interrupts to cp.

Pause.

Null.

Pee the head's semaphore.

Pee the head's semaphore.

Do for each display head that has "must wait" flag set.

Rls_pnl.
### printf(format, p1, p2,......)

Start a pointer at first character of format string.

<table>
<thead>
<tr>
<th>Do while there are still characters in the format string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is the next character in the format a 'Z'?</td>
</tr>
<tr>
<td>NO</td>
</tr>
<tr>
<td>Call put_ascii to print the character.</td>
</tr>
<tr>
<td>Increment format string pointer past the character.</td>
</tr>
<tr>
<td>Do for each character in the expanded string</td>
</tr>
<tr>
<td>Use put_ascii to print the character.</td>
</tr>
</tbody>
</table>

**NOTE:** This version of printf is specific to the LSI 11 IT.

### printf (fmt, p1, p2,......)

Set "a" to be a null pointer.

<table>
<thead>
<tr>
<th>Do until end of format string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is the format string character a 'Z'?</td>
</tr>
<tr>
<td>YES</td>
</tr>
<tr>
<td>Is &quot;s&quot; a null?</td>
</tr>
<tr>
<td>YES</td>
</tr>
<tr>
<td>Set s to point to this character in the format string.</td>
</tr>
<tr>
<td>Advance to next format string character.</td>
</tr>
<tr>
<td>Advance past format in format string.</td>
</tr>
<tr>
<td>&quot;s&quot; = null.</td>
</tr>
</tbody>
</table>

**NOTE:** This version of printf is specific to the Level 6 IT.
put (x, y, vector, count, flag)

Compute any shifting required to do addressing on 16-dot boundaries.

Is it a single vector written many times?

<table>
<thead>
<tr>
<th>YES</th>
<th>Is shifting required?</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>Write top part of vector.</td>
<td>Write bottom part of vector.</td>
</tr>
</tbody>
</table>

Rsrv_pnl.

<table>
<thead>
<tr>
<th>YES</th>
<th>Is shifting required?</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>Copy vectors into local buffer and write out 18 at a time, shifting to write lower parts of vectors.</td>
<td>Copy vectors into local buffer and write out 18 at a time, shifting to write upper parts of vectors (or all of vectors).</td>
</tr>
</tbody>
</table>

Rls_pnl.

NOTE: This version of put is specific to the Level 6 IT.

NOTE: This version of put is specific to the LSI-11 IT.
put_ascii(char, x, y)

Is the x-cursor outside of the page? NO

Put cursor at start of next lower line.

Is the y-cursor outside of the page? NO

Move y-cursor to top of page.

Do switch on effector table entry for the character.

<table>
<thead>
<tr>
<th>ordinary</th>
<th>bell</th>
<th>back_tab</th>
<th>tab</th>
<th>new_line</th>
<th>line_feed</th>
<th>car_return</th>
<th>vert_tab</th>
<th>form_feed</th>
<th>char_delete</th>
<th>line_delete</th>
<th>default</th>
</tr>
</thead>
</table>

Increment x-cursor by one character.

Is the x-cursor outside of the page? NO

Put cursor at start of next lower line.

Is y-cursor outside of page? NO

Move cursor to top of page.

NOTE: This version of put_ascii is specific to the LSI 11/1T.
### put_ascii: put_ascii_form

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is the page bigger than or equal to the whole screen?</td>
<td></td>
</tr>
<tr>
<td>Screen_clear.</td>
<td>Is the page within one character of the whole screen?</td>
</tr>
<tr>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Screen_clear.</td>
<td>Use area_lite to erase the page.</td>
</tr>
</tbody>
</table>

Put the cursor at the top left of the page.

*This is not a procedure. It is one case in put_ascii.
**put_string** \( (buffer, \text{count}) \)

- `Ld_page`
- `Ld_cs`
- Write the string.

**putchar**\( (x, y, \text{ch}) \)

Get a pointer to the vectors for this character in the current character set.

Use `put` to write the vectors.

**NOTE:** This version of `putchar` is specific to the LSI 11 IT.

**putdot**

Write an appropriately formatted message to the Z80 panel controller.

**NOTE:** This version of `putdot` is specific to the L6 IT.

**putdot** \( (x, y, \text{mode}) \)

Set panel registers to write or erase the dot at \( x, y \).

**NOTE:** This version of `putdot` is specific to the LSI-11 IT.
PUTLINE(XA, YA, XR, YR, MODE)

**NOTE:** This version of putline is specific to the LSI 11 IT.

Write an appropriately formatted message to the 280 panel controller.

**NOTE:** This version of putline is specific to the L6 IT.
read(device_id, buf_ptr, length, &status)

- **Is device_id out of range?**
  - **YES**
    - Return error.
  - **NO**
    - **Does this process own the device?**
      - **NO**
        - Return error.
      - **YES**
        - Send read request to device handler.
        - Peek the request semaphore.
        - Set status from that returned by handler process for read request.
        - **Is "catastrophic" bit on in status?**
          - **YES**
            - Return error.
          - **NO**
            - Return success.

read_q(q_ptr)

- Peek the semaphore for this queue.
- Deq an element from this queue.
- Return the value returned by deq.
restart_io(chan, it_id)

Is this an output channel?

NO

In_out = 0.

YES

In_out = 1.

Offset = in_out * the number of LCT bytes per channel.

Ptr = the address of the LCT bytes for this channel in the initial MLCP image.

Flag = false.

For every LCT byte for this channel

Is this byte non-zero?

NO

Write this byte to the corresponding LCT byte for this channel.

YES

Did this io fail?

NO

Flag = true.

Break.

YES

Is flag set?

NO

Flag = results of enabling io on this channel.

YES

Return (flag).
ring_bell()

Set the panel registers to ring the bell.

NOTE: This version of ring_bell is specific to the LSI11 IT.

ring_bell()

Write an appropriately formatted message to the 280 panel controller.

NOTE: This version of ring_bell is specific to the L6 IT.
Rls_pnl()

This process owns the panels?

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Decrement pp_res_level.

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Set pp_owner = 0.

Vee (pp_access).

Return (0).

rsrv_pnl()

This process owns the panels?

<table>
<thead>
<tr>
<th>NO</th>
<th>YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

pee (pp_access).

pp_owner = ME.

Increment pp_res_level.
s_read(drive, sector, buffer)

Is the sector number legal?
NO
Return error.
YES

Is the drive number legal?
NO
Return error.
YES

Compute bus address of disk.
Compute physical track and sector number for the sector.

Do up the maximum number of retries

Is this first time thru for this invocation?
NO
Do a recalibrate.
YES

Loop forever

Try to do an IOLD to load the buffer address and size into the interface.

Did it work?
YES
Break.
NO

Have we exceeded the maximum number of attempts?
YES
Return error.
NO

Write track and sector numbers to the interface.
Start a seek.

Do until status is read

Try to read disk status (success implies that the seek is done).

Was there a bad error?
YES
Continue.
NO

Start the read.

Do until status is read

Try to read disk status (success implies that read is done).

Was there an error?
YES

Return success.
NO

Return error.

**NOTE:** This version of s_read is specific to the level 6 IT.
Is sector or drive out of range?

NO

Return (-1).

YES

Drive = drive number less than 11.

Track = number of track containing sector.

Sect = number of sector on track.

Are we mapping sectors to improve access?

NO

Sect = mapped value of sect.

YES

For the maximum number of attempts:

Was there an error on the disk?

NO

YES

Reset the disk.

Wait for the disk to be ready.

Was there an error on the disk?

NO

YES

Continue.

Send drive and sector numbers to disk.

Wait for the disk to be ready.

Was there an error on the disk?

NO

YES

Continue.

Is the disk operational?

NO

YES

Return (-1).

Set disk's count.

Set disk's address.

Send track number and read command to disk.

Wait for the disk to be ready.

Was there an error on the disk?

NO

YES

Continue.

Such (the data read).

Return (0).

Return (-1).

Note: This version of read is specific to the ESI II II.
s_write(drive, sector, buffer)

Is the sector number legal?

NO

Return error.

YES

Is the drive number legal?

NO

Return error.

Computer bus address of disk.

Compute physical track and sector number for the sector.

Do up to the maximum number of retries.

Is this first time thru for this invocation?

NO

Do a recalibrate.

YES

Loop forever

Try to do an IOLD to load the buffer address and size into the interface.

Did it work?

YES

Break.

NO

Have we exceeded maximum number of attempts?

YES

Return error.

NO

Write track and sector numbers to the interface.

Start a seek.

Do until status is read:

Try to read disk status (success implies that the seek is done).

Was there a bad error?

YES

Continue.

NO

Start the write.

Do until status is read:

Try to read disk status (success implies that the write is done).

Was there an error?

YES

Return success.

NO

Return error.

NOTE: This version of s_write is specific to the Level 6 IT.
write(drive, sector, buffer)

Is sector or drive out of range?

NO

YES

Return (-1).

Drv = drive number less than 11.

Trk = number of track containing sector.

Sect = number of sector on track.

Are we mapping sectors to improve access?

NO

YES

Sect = mapped value of sect.

For the maximum number of attempts:

Was there an error on the disk?

NO

YES

Reset the disk.

Wait for the disk to be ready.

Was there an error on the disk?

NO

YES

Continue.

Send drv and sect numbers to disk.

Wait for the disk to be ready.

Was there an error on the disk?

NO

YES

Continue.

Is the disk operational?

NO

YES

Return (-1).

Set disk's count.

Set disk's address.

Send track number and write command to disk.

Wait for the disk to be ready.

Was there an error on the disk?

NO

YES

Continue.

Swit(data read).

Return (0).

Return (-1).
save_free(drv)

<table>
<thead>
<tr>
<th>NO</th>
<th>YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is drive number (drv) out of range?</td>
<td>Return(-1).</td>
</tr>
<tr>
<td>Write the in_memory copy of freemap to disk.</td>
<td>Return results of write.</td>
</tr>
</tbody>
</table>

screen_clear()

Set panel control register to clear the screen.

NOTE: This version of screen_clear is specific to the LSI-11 IT.
scrunch(ptr, size, num)

For size times

*ptr = *(ptr + num).

Increment ptr.

seek(device_id, length, type)

Is device_id out of range?

YES

Return error.

NO

Does this process own the device?

YES

Call fseek to do the seek.

NO

Return error.
**set_charset(cs)**

CS_ID = CS

**set_cursor(x,y)**

Convert x and y from character to dot coordinates.

Assign values to curs_x and curs_y.

**NOTE:** This version of set_cursor is specific to the LSI-11 IT.

**set_cursor (x_addr, y_addr)**

Convert parameters to dot offsets.

Write offsets to display head controller.

**NOTE:** This version of set_cursor is specific to the Level 6 IT.
set_s:v (env_ptr)

Set_charset.

Set_cursor.

Set_page.

Is this the Level 6 version?

YES

NO

Set_pnl.

set_mode(device_id, buf_ptr, length, status)

Is device_id out of range?

YES

NO

Return error.

Does this process own the device?

NO

YES

Return error.

Send set_mode request to device handler.

Fee the request semaphore.

Set status from that returned by handler process for set_mode request.

Is "catastrophic" bit on in status?

YES

NO

Return error.

Return success.
set_page(page_ptr)

Move parameters from the new page descriptor to the "current" one.

Is this the Level 6 version?

YES

Set the "new page" flag.

NO

startup()

Size = maximum address of available memory.

Zero memory from the end of the program to the end of memory.

Set up CORETAB to indicate all of free memory.

Clear the panel.

Allocate space for the free queue elements.

Initialize the list of free queue elements.

Set the default page, character set, and cursor position to use for plasma panel printing.

Set up the READY_Q as empty.

Create all the processes specified in PROCTAB.

Initialize the I/O system.

Call first_block.

NOTE: This version of startup is specific to the LSI11 IT.
<table>
<thead>
<tr>
<th>startup (high)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set up CORETA5 to indicate all of memory.</td>
</tr>
<tr>
<td>Allocate space for the free queue.</td>
</tr>
<tr>
<td>Initialize the list of free queue elements.</td>
</tr>
<tr>
<td>Disable interrupts from all of the remote display heads.</td>
</tr>
<tr>
<td>Set up READY_Q as empty.</td>
</tr>
<tr>
<td>Initialize the remote display heads.</td>
</tr>
<tr>
<td>Set default page, character set, and cursor position for plasma panel printing.</td>
</tr>
<tr>
<td>Create all processes specified in PROCTAB.</td>
</tr>
<tr>
<td>Load the MLCP.</td>
</tr>
<tr>
<td>Initialize the I/O system.</td>
</tr>
<tr>
<td>Enable interrupts from all remote display heads.</td>
</tr>
<tr>
<td>Call first_block.</td>
</tr>
</tbody>
</table>

NOTE: This version of startup is specific to the Level 6 IT.
**str_num(num, base, place)**

- **Is base valid (i.e., greater than 1 and less than 16)?**
  - **NO**
    - Put a '★' in place.
  - **YES**
    - next = quotient of unsigned division of num by base.
      - **NO**
        - next ≠ 0?
          - **NO**
            - next = remainder of unsigned division of num by base.
              - next ≠ 0?
                - **NO**
                  - Recursively call str_num with parameters next, base, and place.
                - **YES**
                  - Convert next to the ASCII character representing its value, and put that char where indicated by *place.
              - **YES**
                - Increment *place by one character.
suicide()

Clean up any pending I/O associated with this process.

Return the process' stack to the pool of free memory.

Do a process switch.

tiod(token, separator)

Get_cursor.

Get_page_size.

Compute length of token string.

Compute length of separator string.

Is (length of token string) + (length of separator string) more than the space left on this line?

YES

Print a newline using printf.

Print the token and separator strings using printf.

NO
```
tok_print(delim, separator, text, parm)
Use index to find first parameter replacement character in "text."

<table>
<thead>
<tr>
<th>YES</th>
<th>WAS ONE FOUND?</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change the parameter replacement character to a null.</td>
<td>Set &quot;move&quot; flag to true.</td>
<td></td>
</tr>
</tbody>
</table>

Set "move" flag to false.

Do while there are still tokens in text string:

<table>
<thead>
<tr>
<th>YES</th>
<th>WAS ONE FOUND?</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use get_token to get one token from text.</td>
<td>Is &quot;move&quot; flag on, indicating a parm to expand?</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
</table>
| Increment "text" pointer to past the token. | Move "text" pointer up to point at the parameter replacement character that was nulled out.
Use parm_expand to expand the parameter.
Increment "text" pointer past the parameter replacement stuff.
Do while there are still tokens:

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
</table>
| Use tiol to print the token. | Use tiol to print the token and a separator.

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use get_token to get one token from the expanded parameter.</td>
<td>Did we get a token?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point past the token.</td>
<td>Are we at the end of the expanded parameter and is the character following the parameter a delimiter?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
</table>
| Use tiol to print the token. | Use tiol to print the token followed by a separator.
Use index to find the next parameter replacement character.

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Was one found?</td>
<td>Change the character to null.</td>
</tr>
</tbody>
</table>

Set the "move" flag on.
Set flag so outer loop continues.
```
**tp_driver()**

Do forever

1. Read a word from input queue.

   - **Is the word odd?**
     - **YES**
     - Request code = "data."
     - The word is a pointer to a request block containing the request code.
     - The touch coordinates are in bits 2 - 14 of the word.
     - Switch on request code.
     - **open**
       - Set "open" flag on.
     - **close**
       - Set "open" flag off.
     - **flush**
       - Empty the buffer.
     - **peek**
       - Put the count of characters in buffer into request block.
     - **data**
       - (See tp_driver: data_type)
     - **read**
       - (See tp_driver: read_type)
     - **default**
       - Error: Reject the request.
     - **Is a read outstanding?**
       - **YES**
         - Vee user's semaphore.
       - **NO**
         - Enable touch panel input.
         - Vee user's semaphore.
         - Set data length to zero and reject the saved request.
         - Vee user's semaphore.
         - Vee user's semaphore.

**NOTE:** This version of tp_driver is specific to the LSI 11 IT.
### tp_driver: data_type*

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is the touch panel open?</td>
<td></td>
</tr>
<tr>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Is there a read saved up?</td>
<td></td>
</tr>
<tr>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Copy the touch into the user's buffer.</td>
<td></td>
</tr>
<tr>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Is there space in buffer?</td>
<td></td>
</tr>
<tr>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Vee user's semaphore.</td>
<td></td>
</tr>
<tr>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Reset &quot;read pending&quot; flag.</td>
<td></td>
</tr>
<tr>
<td>Copy the touch into the internal buffer.</td>
<td></td>
</tr>
</tbody>
</table>

*This is not a procedure. It is one case in tp_driver.*

### tp_driver: read_type*

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is there data available?</td>
<td></td>
</tr>
<tr>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Compute number of touches to give to user.</td>
<td></td>
</tr>
<tr>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Save the read request.</td>
<td></td>
</tr>
<tr>
<td>Copy touches into user's buffer.</td>
<td></td>
</tr>
<tr>
<td>Vee user's semaphore.</td>
<td></td>
</tr>
</tbody>
</table>

*This is not a procedure. It is one case in tp_driver.*
tp_driver

Do forever:

Read a word from input queue.

Is it a known special code?

YES

The word is the request code.

NO

The word is a pointer to a request block containing the request code.

Switch on request code.

open

Set "open" flag on.

Set "open" flag off.

close

Empty the buffer.

flush

Put count of characters in buffer into request block.

peek

Is a read outstanding?

NO

YES

Is this a set panel setmode?

YES

Enable input from all panels specified.

NO

Disable input from all panels not specified.

read

Use the request block from the previous read.

setmode type

Use the request block from the previous read.

overflow or data

Is a read outstanding?

NO

Break.

Is this a set panel setmode?

YES

Enable input from all panels specified.

NO

Disable input from all panels not specified.

NOTE: This version of tp_driver is specific to the Level 6 IT.
tt_activate(t)

For every slot in tt_current

Is this slot empty?

NO

tt_current(slot) = t.

Display the target.

Return(slot).

YES

Return (-1).
tt_arranger(list, num, values, labels, vs, flags, area, t_width, t_height, mode)

Copy values in area to left, bottom, width, and height.

Max_per_row = max number of targets t_width dots wide that will fit in one row.

Max_per_row less than or equal to 0?

NO

YES

Max_per_row greater than total number?

NO

YES

Max_per_row = num.

Are we doing vertical centering?

NO

YES

Are we doing horizontal centering?

NO

YES

For num descriptions

Are we at the beginning of a row?

NO

YES

Move y down the height of one target.

Move x back to 0.

Is this the last row and are we centering horizontally?

NO

YES

Get width of this row.

Move left over by half the difference between this width and width of area.

Create a target at x=left, y=bottom having the attributes specified in parameters.

Move x over one target width.
tt_cleanup()

For every slot in tt_current

Delete the target pointed to by the entry in that slot.

\[
\begin{align*}
\text{tt_create}(t, x, y, width, height, value, label, cs, flag) \\
\text{Copy the parameters into the appropriate fields of the target structure.} \\
\text{Reset the in_use flag.} \\
\text{Is tflags set so that the target would appear on no remote display heads?} \\
\text{YES} \\
\text{Set tflags so the target will appear on head 0, only.} \\
\text{NOTE: This version of tt_create is specific to the Level 6 IT.}
\end{align*}
\]

\[
\begin{align*}
\text{tt_create}(t, x, y, width, height, value, label, cs, flag) \\
\text{Copy the parameters into the appropriate fields of the target structure.} \\
\text{Reset the in_use flag.} \\
\text{NOTE: This version of tt_create is specific to the LSI-11 IT.}
\end{align*}
\]
**tt_deactivate(slot)**

Erase the target identified by slot.

Delete the target.

Return the results of erasing and deleting.

**tt_delete(slot)**

Does slot indicate an active target?

**NO**

Return (-1).

Turn off in_use flag in target indicated by slot.

Zero tt_current(slot).

Return (0).

**tt_flash(slot)**

Does slot indicate an active target?

**NO**

Return (-1).

Is this target flashable?

**NO**

Lite all the dots in the target.

Erase them all.

Display the target.

Return (0).

**YES**

Return (0).
tt_label(slot)

<table>
<thead>
<tr>
<th>NO</th>
<th>Does slot indicate an active target?</th>
<th>YES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Return (-1).</td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>pointer to target structure for target.</td>
<td></td>
</tr>
</tbody>
</table>

Is this a target that should not be labeled?

<table>
<thead>
<tr>
<th>NO</th>
<th>Return (0).</th>
<th>YES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RSrv_pnl.

Remember current printing environment.

Call set_pnl with bits from tflags so that the target will be displayed on appropriate panel(s).

Set x and y to coordinates of lower left corner of target, leaving white space inside border.

Set width and height to the size of the target, leaving white space inside the edges.

Set the charset to the one for the target.

Get the size of characters in this charset.

Modify x and width to center a maximum line of characters within the target.

Set the printing page to be this area within the target.

Position the cursor at the top of the page.

Tok_print the label.

Restore the printing environment.

Ris_pnl.

Return (0).

NOTE: This version of tt_label is specific to the Level 6 IT.
tt_label(slot)

Does slot indicate an active target?

NO

Return (-1).

t = pointer to target structure for target.

Is this a target that should not be labeled?

NO

YES

Return (0).

Remember current printing environment.

Set x and y to coordinates of lower left corner of target, leaving white space inside border.

Set width and height to the size of the target, leaving white space inside the edges.

Set the charset to the one for the target.

Get the size of characters in this charset.

Modify x and width to center a maximum line of characters within the target.

Set the printing page to be this area within the target.

Position the cursor at the top of the page.

Tok_print the label.

Restore the printing environment.

Return (0).

NOTE: This version of tt_label is specific to the LSI-11 IT.
tt_lite(slot, mode)

Does slot indicate an active target?

NO

Return (-1).

YES

Is this a flashable target?

NO

Get the coordinates of the lower left corner of the target in x and y.

Get the size of the target in width and height.

Area_lite(x, y, x + width -1, y + height -1, mode).

Return (0).

NOTE: This version of tt_lite is specific to the LSI-11 IT.

This version of tt_lite is specific to the Level 6.
tt_mark(slot, mode)

<table>
<thead>
<tr>
<th>Does slot indicate an active target?</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
</tr>
<tr>
<td>Return (-1).</td>
</tr>
<tr>
<td>YES</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Is this a markable target?</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
</tr>
<tr>
<td>Return (0).</td>
</tr>
<tr>
<td>YES</td>
</tr>
</tbody>
</table>

Get coordinates of lower right corner of target, offset to be inside border, in x2, y1.

Get coordinate of left edge of mark in x1.

Get coordinate of top edge of mark in y2.

Area_lite (x1, y1, x2, y2,......).

Return (0).

NOTE: This version of tt_mark is specific to the LSI 11 IT.
tt_mark(slot, mode)

Does slot indicate an active target?

NO

Return (-1).

YES

Is this a markable target?

NO

Return (0).

YES

Get coordinates of lower right corner of target, offset to be inside border, in x2, y1.

Get coordinate of left edge of mark in xl.

Get coordinate of top edge of mark in y2.

Rsrv_pnl.

Get_pnl(&pnl).

Use set_pnl to set from tflags the set of remote display heads to be written on.

Area_light(xl, y1, x2, y2, mode).

Set_pnl(pnl).

Return (0).

NOTE: This version of tt_mark is specific to the Level 6 IT.
tt_move(t, new_x, new_y)

Set tx and ty entries in target structure t to new_x and new_y.

tt_outline(slot, mode)

Does slot indicate an active target?

NO

Return (-1).

YES

\[ t = \text{tt_current(slot)}. \]

Is \( t \) an outlineable target?

NO

Return (0).

YES

Get coordinates of lower left corner of target in \( x_1 \) and \( y_1 \).

Get coordinates of upper right corner of target in \( x_2 \) and \( y_2 \).

Move \( x_1, y_1, x_2, \) and \( y_2 \) to the inside of the target to leave white space around it.

Use putline to display or erase the four lines connecting \((x_1, y_1)\) to \((x_1, y_2)\) to \((x_2, y_2)\) to \((x_2, y_1)\) and back.

Return (0).

NOTE: This version of tt_outline is specific to the LSI-11 IT.
tt_outline(slot, mode)

Does slot indicate an active target?

NO

Return (-1).

YES

t = tt_current(slot).

Is *t an outlineable target?

NO

Return (0).

YES

Get coordinates of lower left corner of target in x1 and y1.

Get coordinates of upper right corner of target in x2 and y2.

Move x1, x2, and y2 to the inside of the target to leave white space around it.

Rsrv_pnl.

Use get_pnl to save the set of currently selected remote display heads.

Use set_pnl to set the set of selected heads to those specified in tflags.

Use putline to display or erase the four lines connecting (x1, y1) to (x1, y2) to (x2, y2) to (x2, y1) and back.

Use set_pnl to put the set of selected heads back.

Rls_pnl.

Return (0).

NOTE: This version of ttoutline is specific to the Level 6 IT.
tt_read(touch)

Flush input from the touch panel.

While (true)

Read the coordinates of a touch.

tx = the x coordinate, in dots.

ty = the y coordinate, in dots.

For every slot in tt_current

Does this slot describe an active target located on the display head from which this touch came?

NO

Is the point (tx, ty) within the area of this target?

NO

YES

Flash this target.

Return (slot).

NOTE: This version of tt_read is specific to the Level 6 IT.
**tt_read(touch)**

Flush input from the touch panel.

While (true)

- Read the coordinates of a touch.
  - \(tx\) = the x coordinate, in dots.
  - \(ty\) = the y coordinate, in dots.

For every slot in `tt_current`.

- Does this slot describe an active target?
  - **NO**
  - **YES**

- Is the point \((tx, ty)\) within the area of this target?
  - **NO**
  - **YES**

- Flash this target.
- Return (slot).

**NOTE:** This version of `tt_read` is specific to the LSI 11 IT.

**tt_relabel(t, str, value, mode)**

Is \(t\) an index into `tt_current`?

<table>
<thead>
<tr>
<th>NO</th>
<th>YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use (t) as target pointer.</td>
<td>Get pointer to target structure to be relabeled.</td>
</tr>
</tbody>
</table>

Change `tlabel` and `tvalue` fields of target structure to be `str` and `value`. 
tt_selections(touch, max num tchu, num, num overfl, values, slots)

While (true)

Slot = results of reading one touch via tt_read.

t = tt_current(slot).

add_to_list = true.

Do duplicate touches cancel each other?

NO

YES

For every target chosen so far

Does value of chosen target == t_value?

NO

YES

Decrease count of chosen targets.

Scrub the arrays of target slots and target values.

Unmark the target.

add_to_list = false.

Break.

Is add_to_list true?

NO

YES

Is the number chosen greater than or equal to the max number allowed?

NO

YES

Should we return in this case?

NO

YES

Return (num chosen).

Decrement number chosen.

Delete oldest choice from lists.

Add target to list of choices.

Mark target.

Increment count of chosen targets.

Down this target have the forcy-return bit set?

NO

YES

Return (number chosen).
vee(sem)

Become non-interruptable.

Is the count for this semaphore less than 0?

NO

YES

Dequeue the process waiting on this semaphore.

Add that process to the READY_Q.

Increment the count for this semaphore.

Restore the initial interruptable status.

verify(s1, s2)

Count = 0.

For every character in s1 string

For every character in s2.

Does this char in s2 match the current char in s1?

NO

YES

GO TO OK.

Return(count).

Return (count)

OK: Increment count

Return (-1).
INFO: This version of vip.proc is specific to the LSI II IT.
vip_proc: vip_read*

Is this a read request and another read is already pending?

YES
Reject the new read request.
Break.

NO

Is the buffer empty?

YES
Save a pointer to this request block as a pending read.
Break.

NO

While there are still input characters and the user's buffer is not exhausted.

Copy a character into the user's buffer.

Was the character an ETX?

YES
Break.

NO

Null terminate the user's buffer.

Vea the user's semaphore.

*This is not a procedure. It is one case in vip_proc.

vip_proc: vip_write*

Is a write active?

NO
Break.

YES

Has all data for this write been written?

NO

Vea user's request.

Set flag indicating no write pending.
Break.

Construct an output message, using as much data from the current write request as will fit in the output buffer.

Set buffer pointer and count for the output interrupt handler.

Set flag for input interrupt processor indicating that an output message is available.

*This is not a procedure. It is one case in vip_proc.
vip_proc()

Empty input buffer.

Set flags indicating that the line is down and closed.

Do "forever"

Read a word from VIP_Q.

Is it a known special code?

YES

The word is a request code.

NO

The word is a pointer to a request block containing a request code.

Switch on the request code.

open_type vip down close_type flush_type vip_up data read_type write_type ack peek default

See vip_proc: open_type

Set line up flag to false.

Flag line as closed.

Turn off input and output.

Turn-off successful?

NO

YES

Set error flags for user routine.

Set line up flag to true.

Is a read saved?

NO

Break.

Make it look as though the read just came in.

Is a write saved?

YES

NO

Put count of characters in input buffer into user's request block.

Reject the request.

Reject the new request.

Vee the request.

Init count and address for current write.

See vip_proc: read_type.

See vip_proc: ack.

See vip_proc: flush_type.
vip_proc: ack

Is there a current write?

NO

Break.

YES

Is the count zero (i.e. is the write now completed)?

NO

Set return to user to show successful write.

YES

Vee user's request.

Break.

Format an output message, using as much of the user's data as will fit in the buffer.

Decrease current count and increase current address by the number of user's characters put in message.

Set a flag so that the interrupt routine will initiate transmission of this message when the time comes.

*This is not a procedure. It is one case in vip_proc and is specific to the Level 6 IT.

vip_proc: flush_type*

Is this a close or a flush?

YES

Empty the input buffer.

NO

Is a read saved up?

YES

Reject the saved read.

NO

Is a write saved up?

YES

Reject the write.

NO

Is this a close or a flush?

YES

Vee the request semaphore.

*This is not a procedure. It is one case in vip_proc and is specific to the Level 6 IT.
vip_proc: open_type

- Is the line already open?
  - NO
  - Start up transmission of a quiescent frame.
    - YES
      - Did initiation succeed?
        - YES
          - Flag line as "up."
        - NO
          - Flag line as "down."
    - NO
      - Set "vip_writing" flag.
  - Empty input buffer.
    - YES
      - Is line "up?"
        - YES
          - Flag line as open.
        - NO
          - Set user's status flags to indicate failure of open.
  - Vee request semaphore.

*This is not a procedure. It is one case in vip_proc and is specific to the Level 6 IT.

vip_proc: read_type

- Is there already a read saved up?
  - YES
    - Reject the new read.
    - Break.
  - NO

- Is the input buffer empty?
  - YES
    - Save up the read request.
    - Break.
  - NO

- Copy data into user's buffer, stopping at end of buffer, end of input characters, or ETX character, whichever is first.
- Null terminate the input.
- Vee user's request.

*This is not a procedure. It is one case in vip_proc and is specific to the Level 6 IT.
The following switch statement is a finite state machine which parses input messages.

Switch on inp state:

<table>
<thead>
<tr>
<th>SOM_EOT_wait</th>
<th>adr_wait</th>
<th>stat_wait</th>
<th>null_STX_wait</th>
<th>null_ETX_wait</th>
<th>STX_wait</th>
<th>text_rev</th>
<th>bcc_wait</th>
<th>got_trans or error_flush</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOM_EOT_wait</td>
<td>adr_wait</td>
<td>stat_wait</td>
<td>Update longi-</td>
<td>Update longi-</td>
<td>Update longi-</td>
<td>text_rev</td>
<td>bcc_wait</td>
<td>Null</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>tudinal pari-</td>
<td>tudinal pari-</td>
<td>tudinal pari-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ty check.</td>
<td>ty check.</td>
<td>ty check.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Input=STX?</td>
<td>Input=ETX?</td>
<td>Input=ETX?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Did the carrier go down at the completion of the above text?

YES

Did the FSM find a valid message?

YES

Switch on type of message received.

NO

Was it a bad message?

YES

Is vip line still open?

NO

Start up a new read.

YES

Did it start ok?

NO

Send vip_down message to vip_proc.

YES

Send vip_down message to vip_proc.

NOTE: This version of vipint is specific to the Level 6 IT.
vipint: ACK*

Did we just send user data?

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send &quot;got ack&quot; message to the vip_proc.</td>
<td></td>
</tr>
<tr>
<td>Set up to send quiescent frame.</td>
<td></td>
</tr>
</tbody>
</table>

*This is not a procedure. It is one case in vipint and specific to the Level 6 IT.

vipint: data*

Set up to send an ACK.

*This is not a procedure. It is one case in vipint and is specific to the Level 6 IT.

vipint: NAK*

<table>
<thead>
<tr>
<th>YES</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is there user data to send?</td>
<td>Set up to send quiescent frame.</td>
</tr>
<tr>
<td>Set up to resend user data.</td>
<td></td>
</tr>
</tbody>
</table>

*This is not a procedure. It is one case in vipint and is specific to the Level 6 IT.
```plaintext
/*viprint ()
Have any of the status register bits changed?

   YES
   Send a message to the vic process.
   The EV11 became ready?
   YES
   Send a message to the vic process.
   The EV11 became not ready?
   NO
   The carrier just went away?
   NO

   We now must make some response to the VICSS which is at the other end of the line.

   Was a message received intact? (i.e. Imp_state == got_trans)

   YES
   Switch on "input_type."
   0
   We got a G (quiescent frame).
   1
   We got an ACK.
   Is there output waiting?
   YES
   Set up to send output.
   NO
   Set up to send quiescent frame.
   2
   We got a NAK.
   Did we really send data?
   YES
   Send ACK message to the vic proc.
   NO
   Set up to send quiescent frame.
   3
   We got a data message correctly.
   Did we really send data?
   YES
   Send a "data" message to the vic proc.
   NO
   Set up to send a quiescent frame.

   Set status registers to start a transmission.
   Set Imp_state = 0.
   Return.

   Is a character available?

   YES
   Read the character into variable "input."
   Was there a transmission error and we are in the middle of a transmission?
   NO

   Set Imp_state = error state.
   Return.

   YES
   Imp_state = SOH EOT wait.
   Initialize temporary input buffer pointer.
   Input_type = 0.
   The following switch statement is a finite state machine which parses input messages.
   Switch on Imp_state:

   SOH_EOT_wait
   addr_wait
   stat_wait
   null_STX_wait
   null_FIX_wait
   STX_wait
   text_rev
   bre_wait
   got_trans
   error_flush
   Null.
```
**adr_wait**

Update longitudinal parity check.

- Input = correct address?
  - YES
  - Input = EOT?
    - YES
      - Inp_state = got_trans.
    - NO
      - Inp_state = error_flush.
  - i.e. transmission received ok.
- NO
  - i.e. transmission received in error.

*This is not a procedure. It is one case in viprint.

**bcc_wait**

Is the input character equal to the longitudinal parity check character?

- NO
  - Inp_state = error_flush.
- YES
  - Inp_state = SOH_EOT_wait.

*This is not a procedure. It is one case in viprint.

**SOH EOT wait**

- Input = SOH?
  - YES
  - Input = EOT?
    - YES
      - Inp_state = got_trans.
    - NO
  - NO

Initialize longitudinal parity check character.

*This is not a procedure. It is one case in viprint.
**stat_wait**

Update longitudinal parity check.

<table>
<thead>
<tr>
<th>Input = null?</th>
<th>Input = ACK?</th>
<th>Input = NAK?</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input_type = 3.</th>
<th>Input = ACK?</th>
<th>Input = NAK?</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i.e. this is an input text message.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input_type = 1.</td>
</tr>
<tr>
<td>YES</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inp_state = STX_wait.</th>
<th>i.e. this is an ACK message.</th>
<th>Inp_state = null_STX_wait.</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.e. this is a NAK message.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input_type = 2.</td>
<td>NO</td>
<td>Transmission is in error.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inp_state = null_STX_wait.</th>
</tr>
</thead>
<tbody>
<tr>
<td>If transmission is in error.</td>
</tr>
</tbody>
</table>

*This is not a procedure. It is one case in viprint.*

**text_rev**

Update longitudinal parity check.

<table>
<thead>
<tr>
<th>Would this character overflow the buffer?</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inp_state = error_flush.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert the character into the input buffer.</td>
</tr>
<tr>
<td>Was it an ETX?</td>
</tr>
<tr>
<td>NO</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inp_state = bcc_wait.</th>
</tr>
</thead>
</table>

*This is not a procedure. It is one case in viprint.*
viprint()

Are there characters left to send?

YES
Send the next character.

NO
Disable the transmitter and set up for receiving.

write(device_id, &status)

Is device_id out of range?

YES
Return error.

NO
Does this process own the device?

YES
Send write request to device handler.

NO
Set status from that returned by handler process for write request.

Is "catastrophic" bit on in status?

YES
Return error.

NO
Return success.

write_q(q_ptr, value)

Enqueue value on queue.

Vee the semaphore associated with this queue.

xopen(fib, index_b, o, slot)

Assign appropriate values into file index block.

Return (0).
Figure out which head this comes from.

Read status word.

Was it an attention interrupt?

NO

Set status word for writing/reading process.

YES

Read from keyboard or touch panel?

Keyboard

Touch panel

Input buffer empty?

YES

NO

Turn off write/read in progress flag.

Input buffer empty?

YES

NO

Vee the process.

Write data message to keyboard process.

Write data message to touch panel process.

Copy data into keyboard buffer.

Copy data into touch panel buffer.

Turn off flag indicating tp or kb read is in progress.

Is a read or write still pending?

NO

Is keyboard data available?

YES

NO

Start a read from the keyboard.

Is touch panel data available?

YES

NO

Start a read from the touch panel.

Is a process waiting to read or write?

YES

NO

Set status for process.

Set read from touch panel in progress flag.

Vee semaphore for the process.

Turn off process waiting flag.
zero_blk(ev, blk)

Flush this ev structure's buffer.

Zero the buffer.

For each sector in the block

Write out one sector's worth of zeros.

Did the write fail?

NO

YES

Return (-1).

Return (0).

zero_sim(d1, d2, o, userbuf, l, fcn)

Is either l or o out of range, or does buffer cross a block boundary, or is fcn not READ?

NO

YES

Return (-1).

Fill user's buffer with 0's.

Return (0).
### Intelligent Terminal Software Flowcharts

**Authors:** Deborah S. Brown, David A. Willcox, Betty Kasprzycki, John R. Mullen

**Performing Organization:**
Center for Advanced Computation, University of Illinois, Urbana, Illinois 61801

**Controlling Office:**
Command and Control Technical Center, 11440 Isaac Newton Square, North Reston, Virginia 22090

**Contract or Grant Number(s):**
DCA100-76-C-0088

**Report Date:**
October 31, 1977

**Number of Pages:**
1

**Distribution Statement:**
Copies may be obtained from the address in Block 11 above.

**No restriction on distribution.**

**Keywords:** Intelligent Terminal, Man-Machine Interface

**Abstract:**
This document contains Nassi-Shneiderman flowcharts for the Intelligent Terminal system described in CAC document number 236 (CCTC-WAD Document Number 7516), "Intelligent Terminal Programmer's Manual."