C

Center for Advanced Computation

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

CAC Document Number 252
CCTC-WAD Document Number 7523

Research in
Network Data Management and
Resource Sharing

INTELLIGENT TERMINAL SOFTWARE FLOWCHARTS

October 31, 1977

CAC Document Number 252
CCTC-WAD Document Number 7523

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
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.

NASSI-SHNEIDERMAN DIAGRAMS

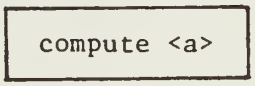
The diagramming technique developed by I. Nassi and B. Shneiderman (cf. SIGPLAN Notices, August 1973) provides four basic visual structures corresponding to the four basic constructs of a program:

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:

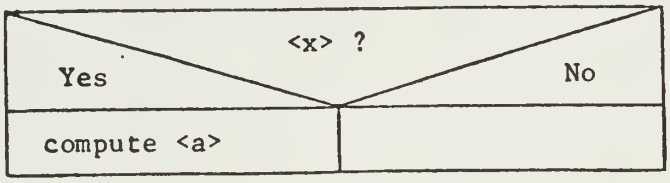


The box is usually named, or some English phrase or some equation is written in the box to indicate the nature of the process or computation. The box may represent any process or computation, from the whole of an operating system to a single statement of the kind "a = b + c". An empty box represents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

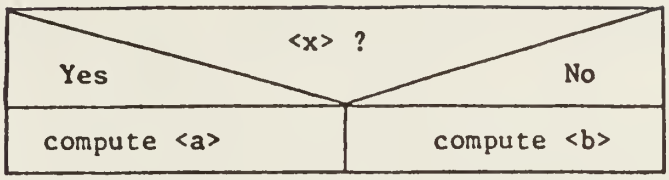
Decision

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:

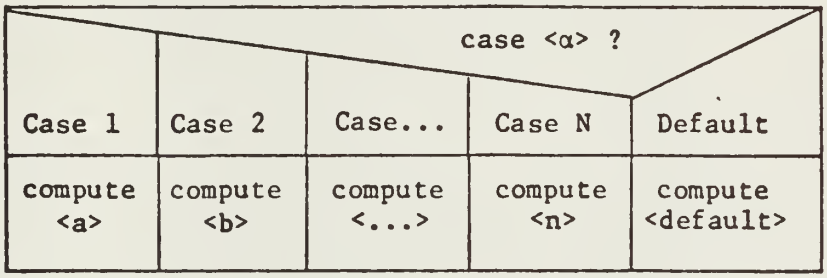


2. if ... else statement:



Multi-case Decision

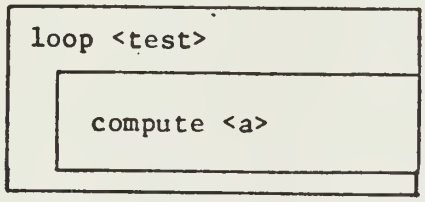
The representation of a multi-case decision is a simple extension of the previous visual structure for representing simple decisions:



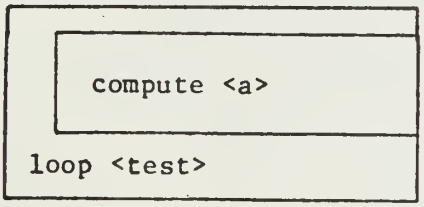
Iteration

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:

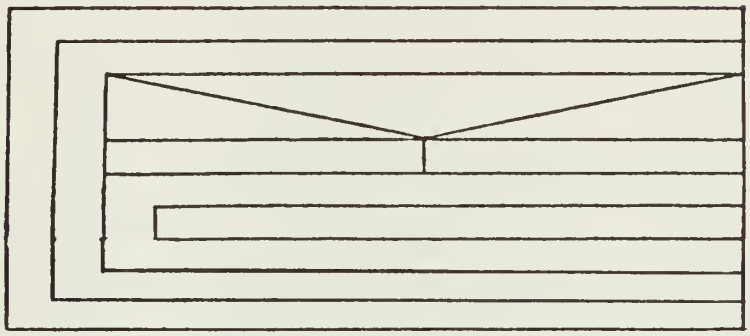


2. bottom test:

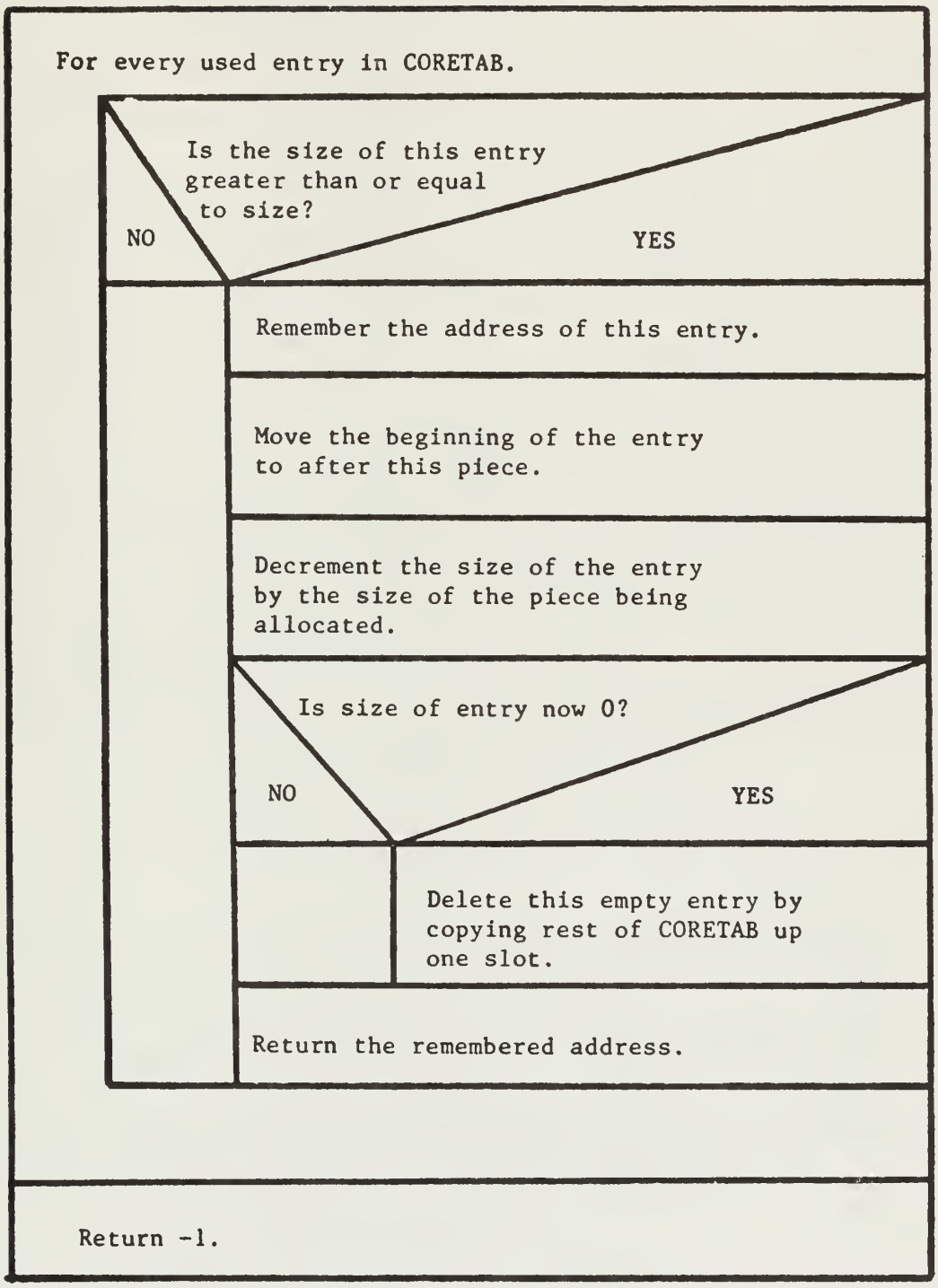


Combination

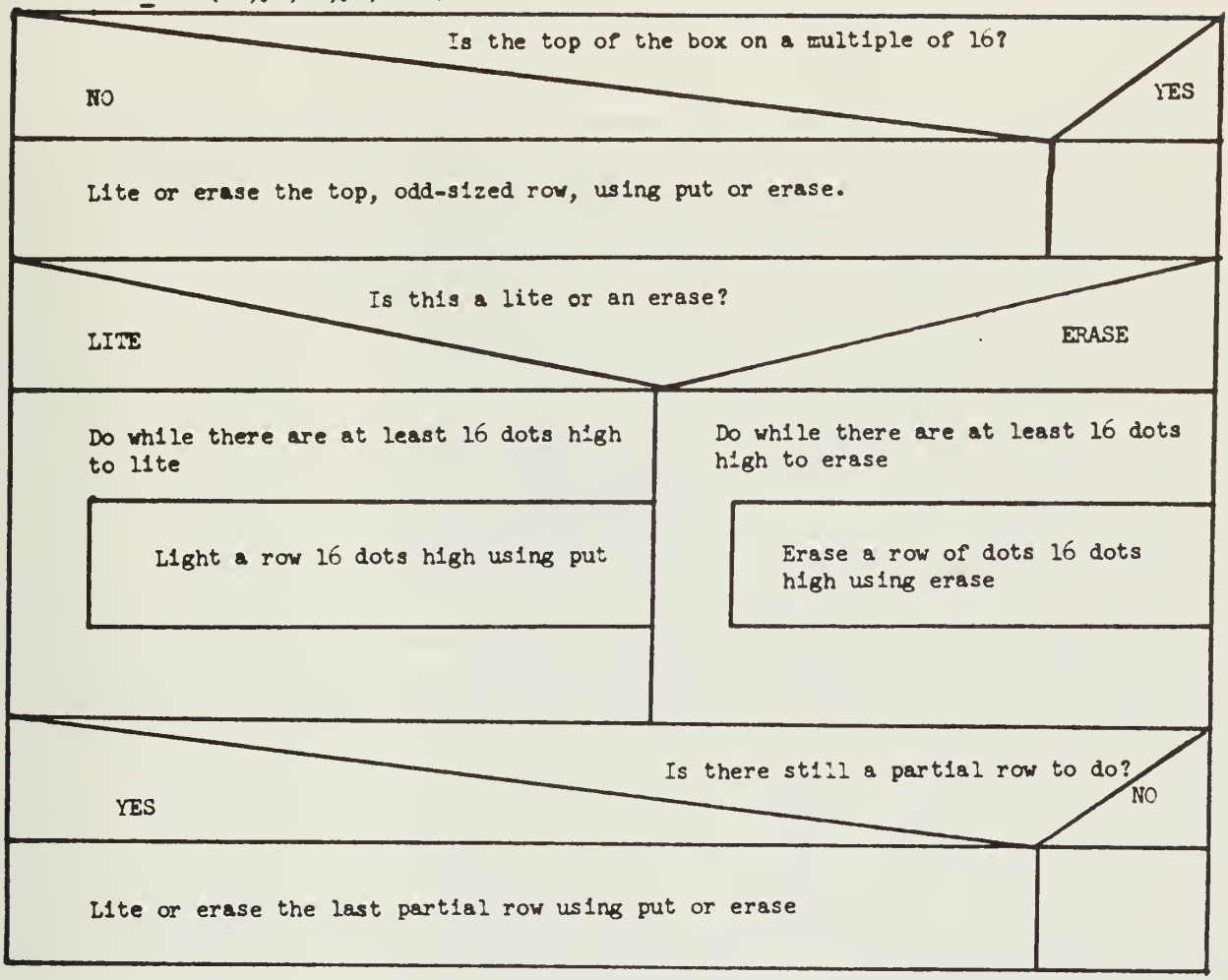
The visual structures presented above may be combined to any degree to represent a computational structure, e.g.:



alloc(size)

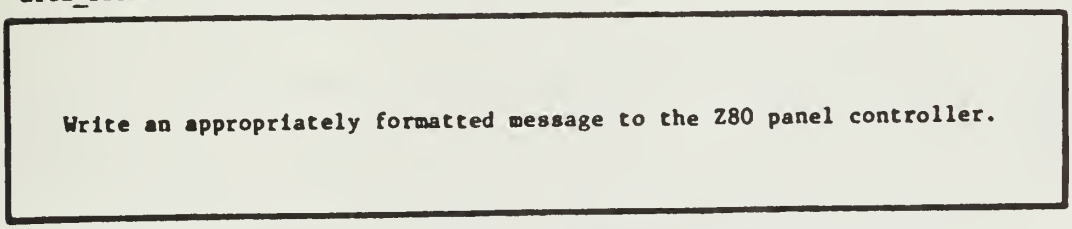

For every used entry in CORETAB.
Is the size of this entry greater than or equal to size?
NO
YES
Remember the address of this entry.
Move the beginning of the entry to after this piece.
Decrement the size of the entry by the size of the piece being allocated.
Is size of entry now 0?
NO
YES
Delete this empty entry by copying rest of CORETAB up one slot.
Return the remembered address.
Return -1.

area_lite(x1,y1,x2,y2,mode)

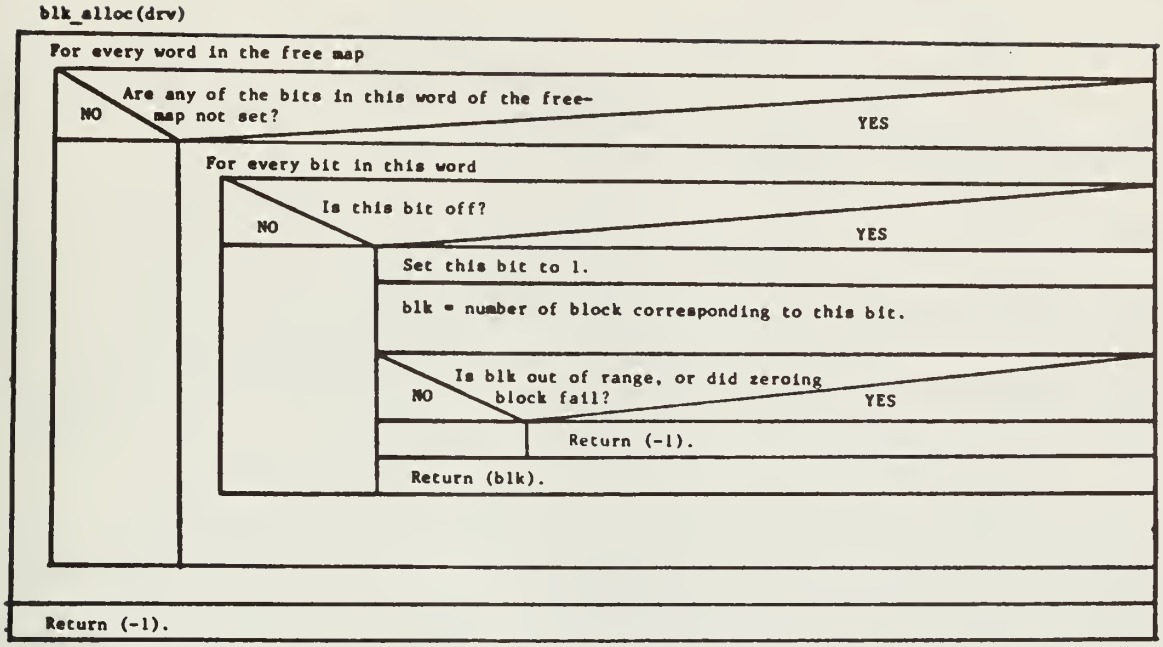


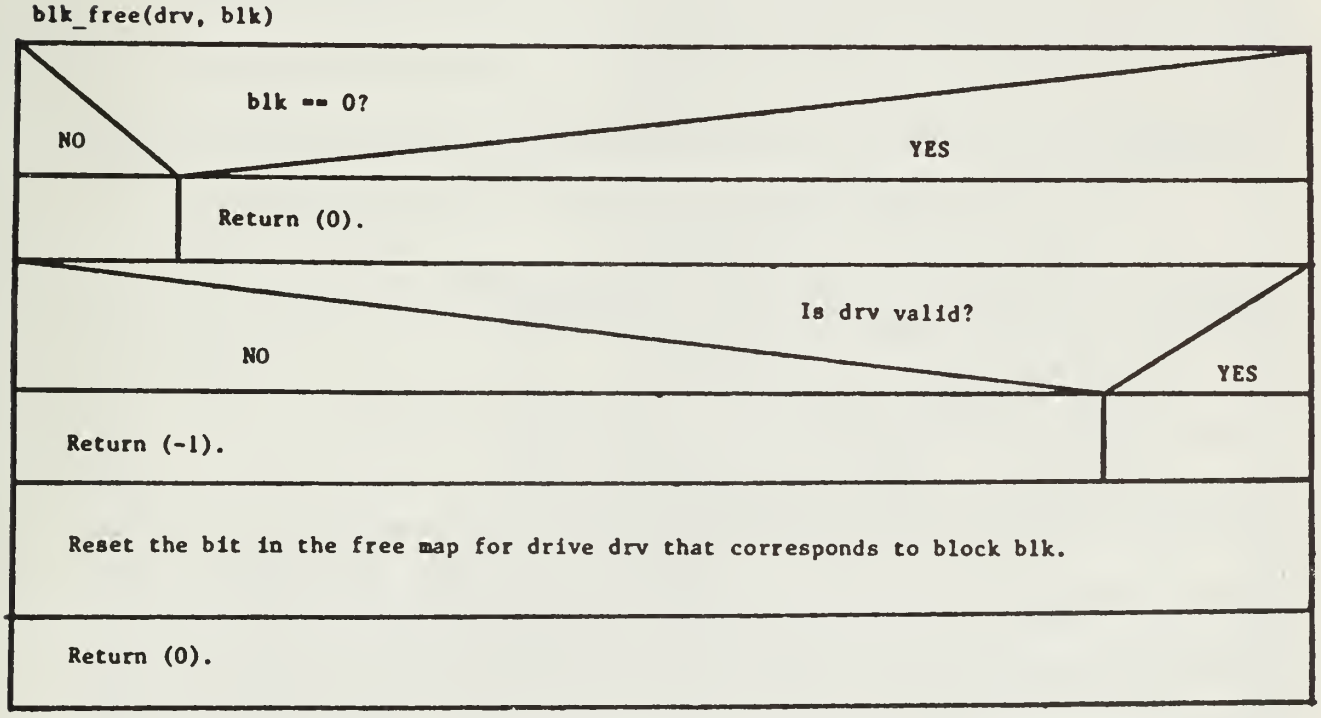
NOTE: This version of area_lite is specific to the LSI-11 IT.

area_lite

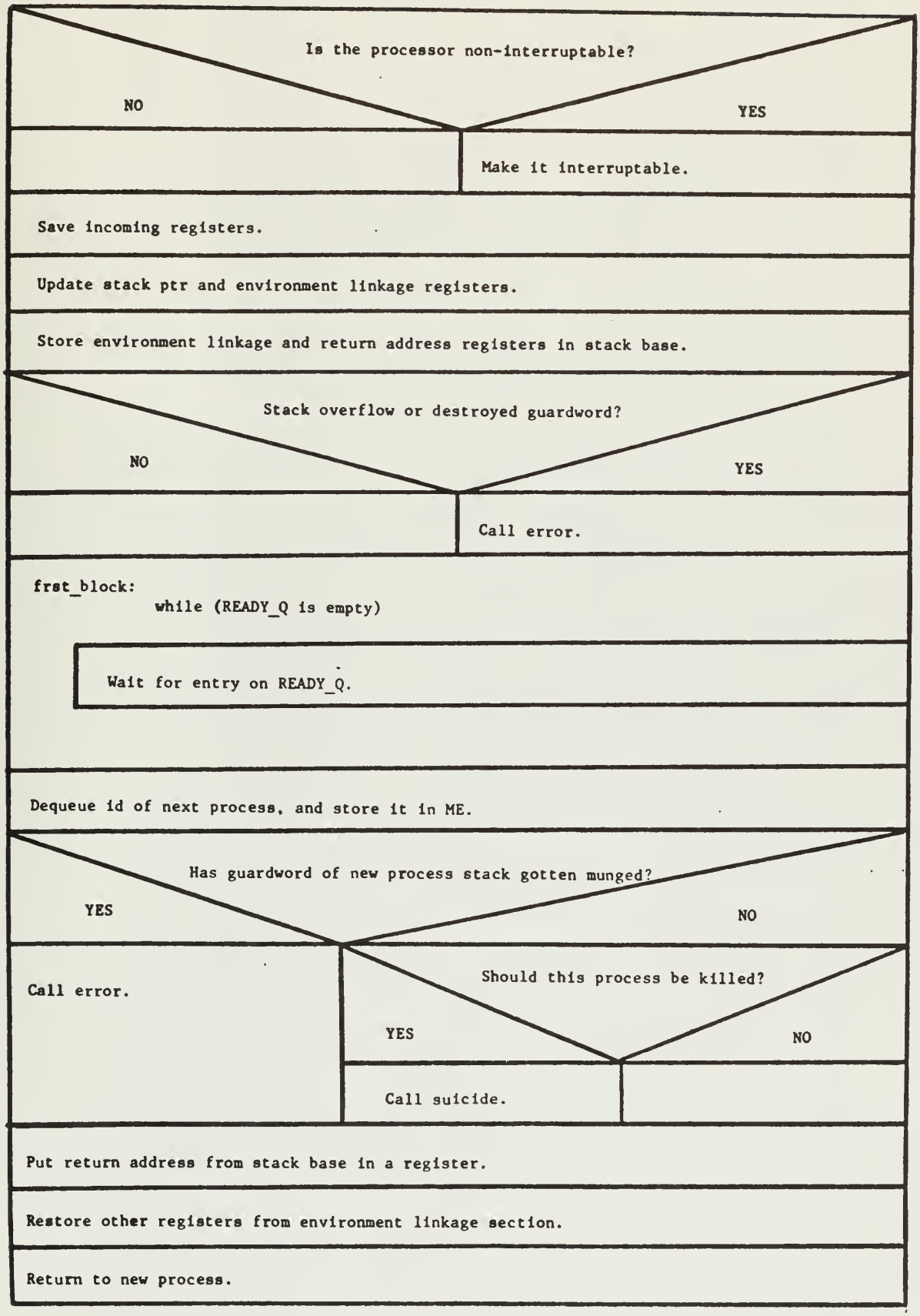


NOTE: This version of area_lite is specific to the L6 IT.

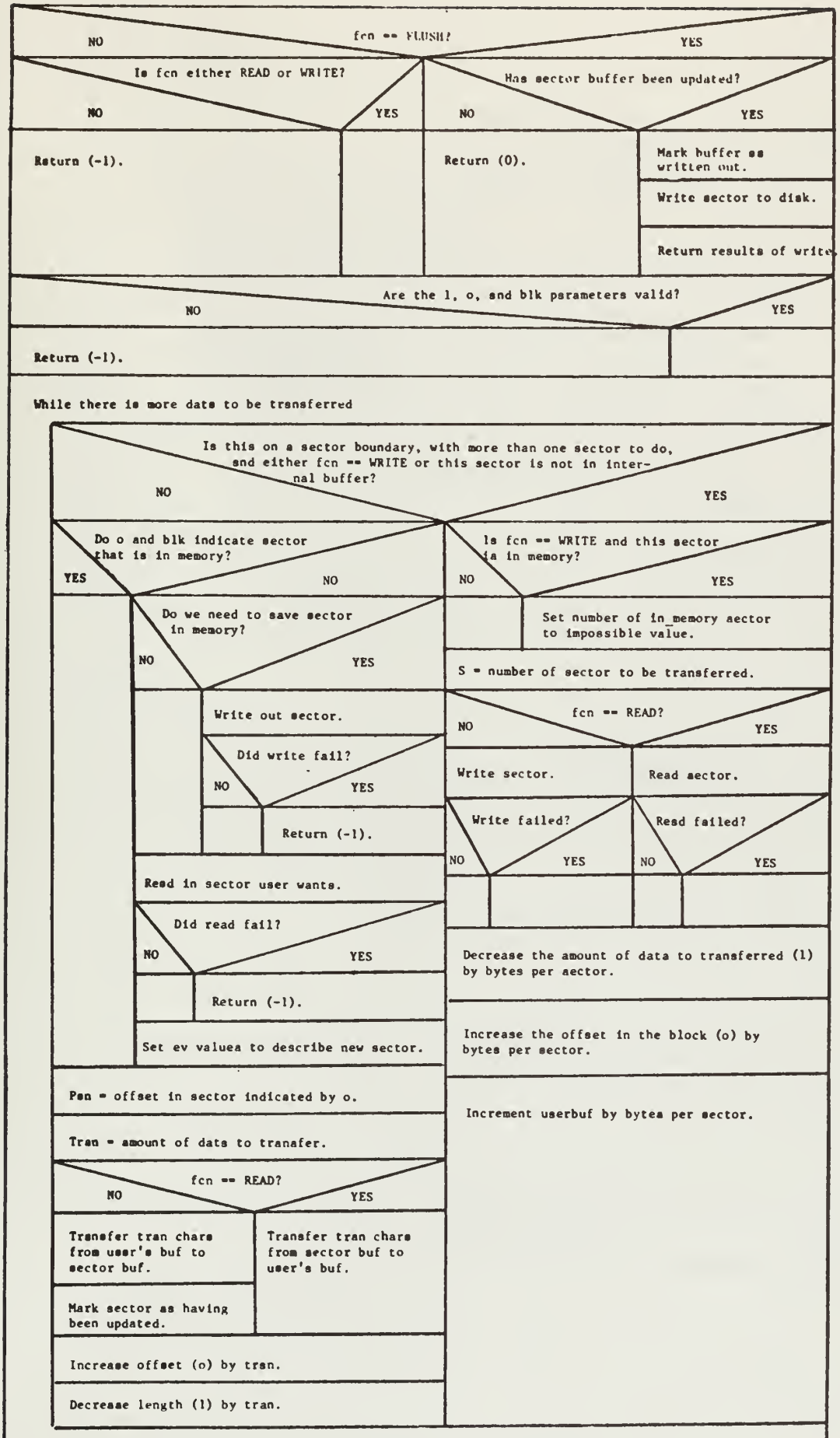

blk_alloc(drv)
For every word in the free map
Are any of the bits in this word of the free-map not set?
NO
YES
For every bit in this word
Is this bit off?
NO
YES
Set this bit to 1.
blk = number of block corresponding to this bit.
Is blk out of range, or did zeroing block fail?
NO
YES
Return (-1).
Return (blk).
Return (-1).


blk_free(drv, blk)
blk == 0?
NO
YES
Return (0).
Is drv valid?
NO
YES
Return (-1).
Reset the bit in the free map for drive drv that corresponds to block blk.
Return (0).

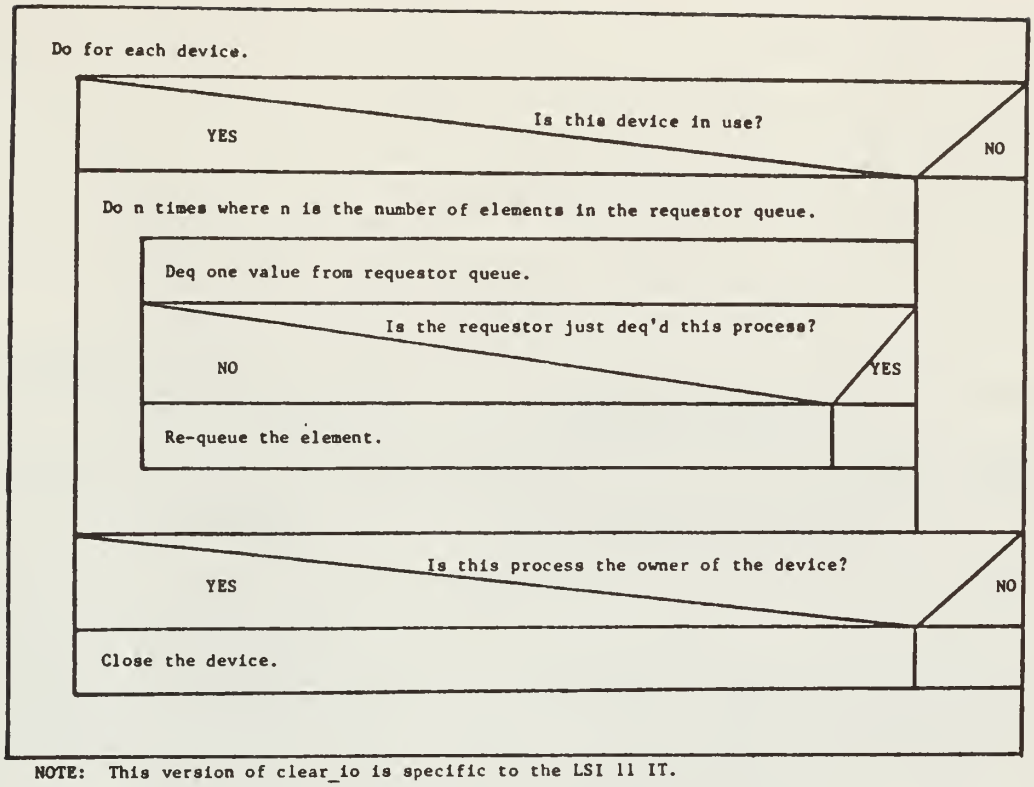
block()


Is the processor non-interruptable?
NO
YES
Make it interruptable.
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.
frst_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.

buflo(ev, blk, o, userbuf, l, fcn)

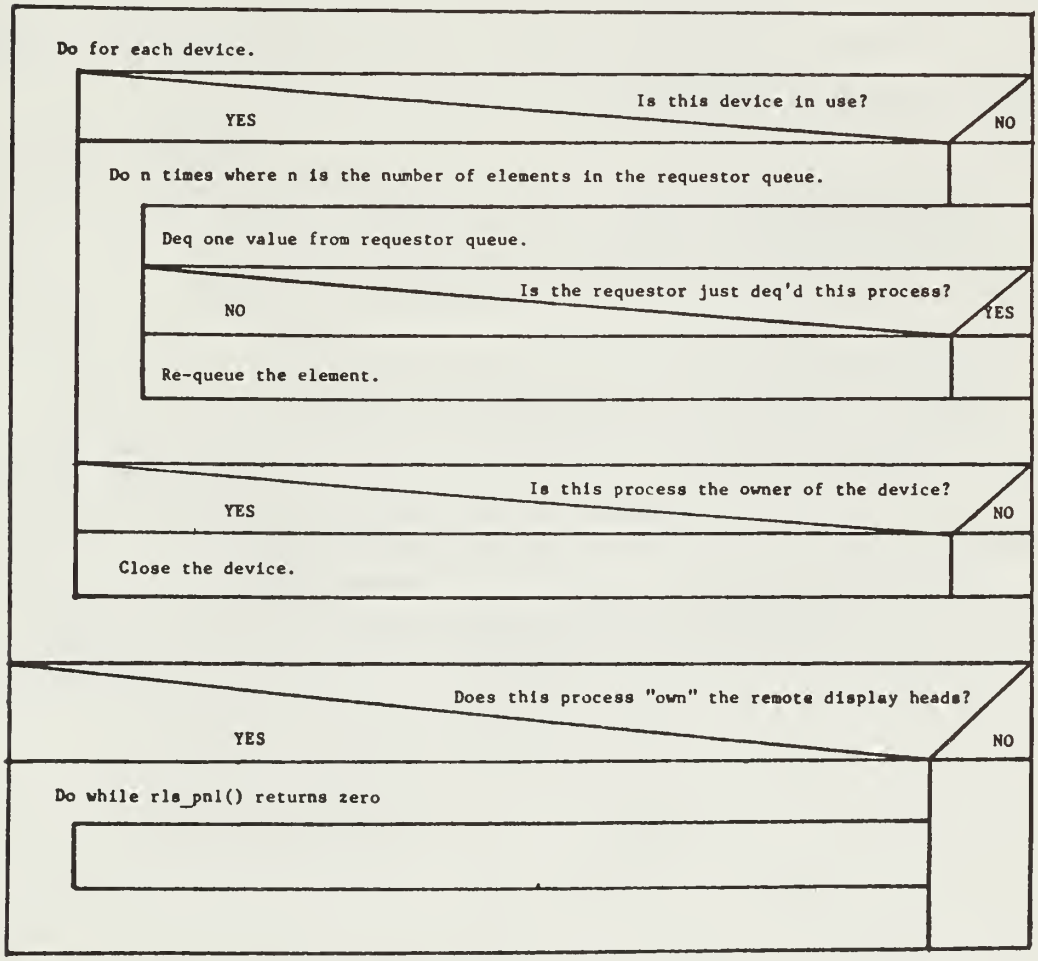


clear_io()



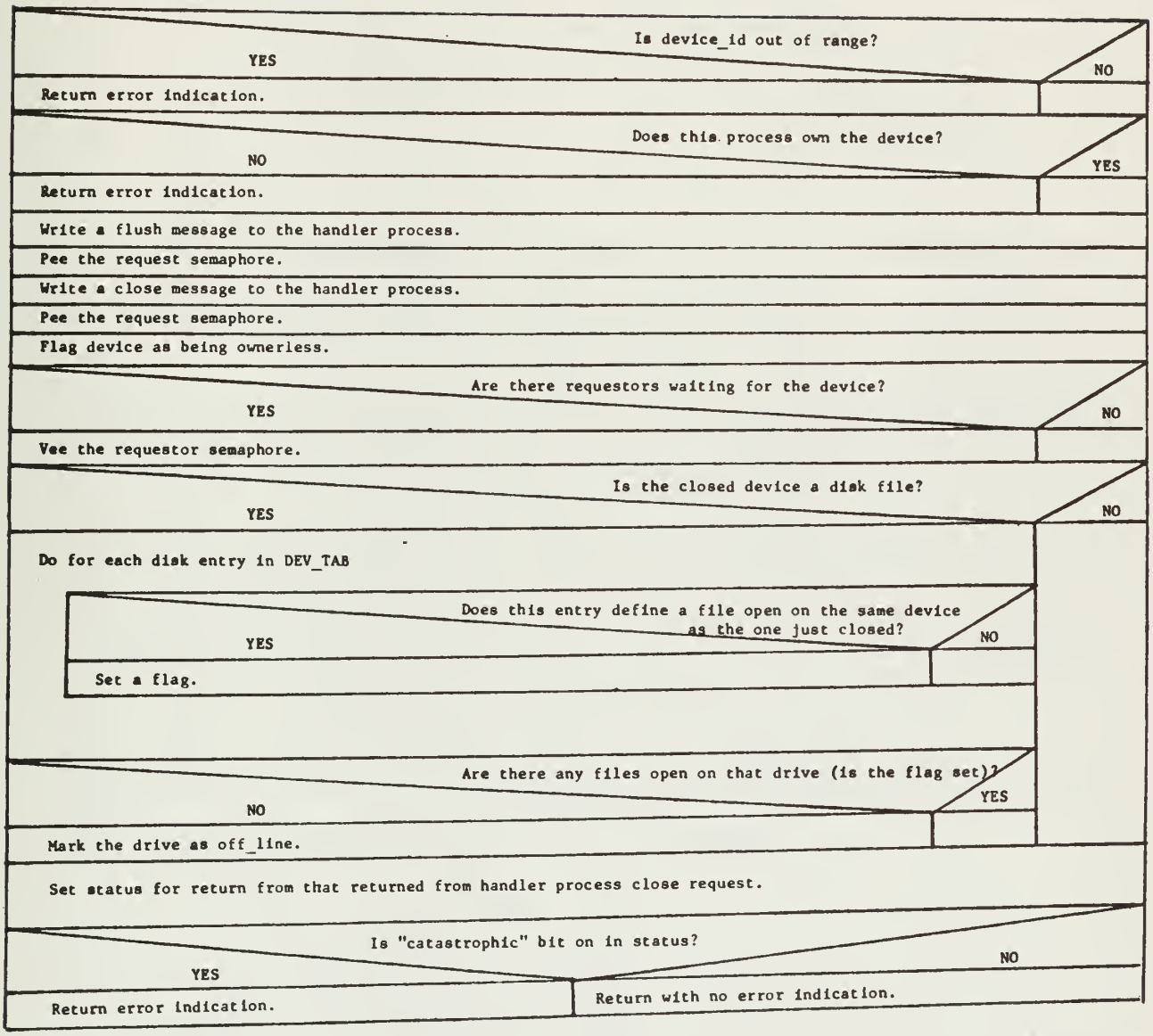
NOTE: This version of clear_io is specific to the LSI 11 IT.

clear_io()

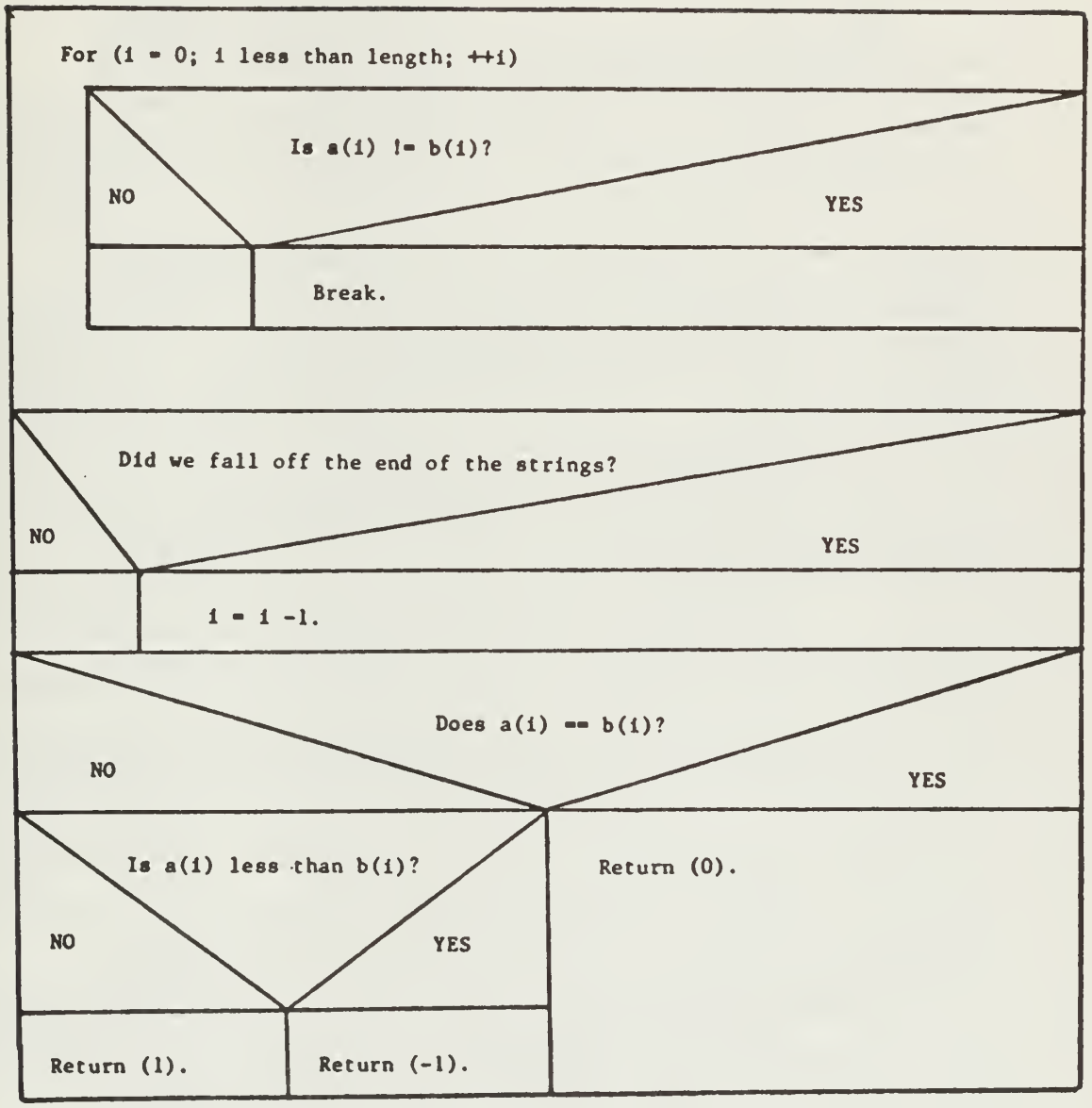


NOTE: This version of clear_io is specific to the Level 6 IT.

close(device_id, &status)

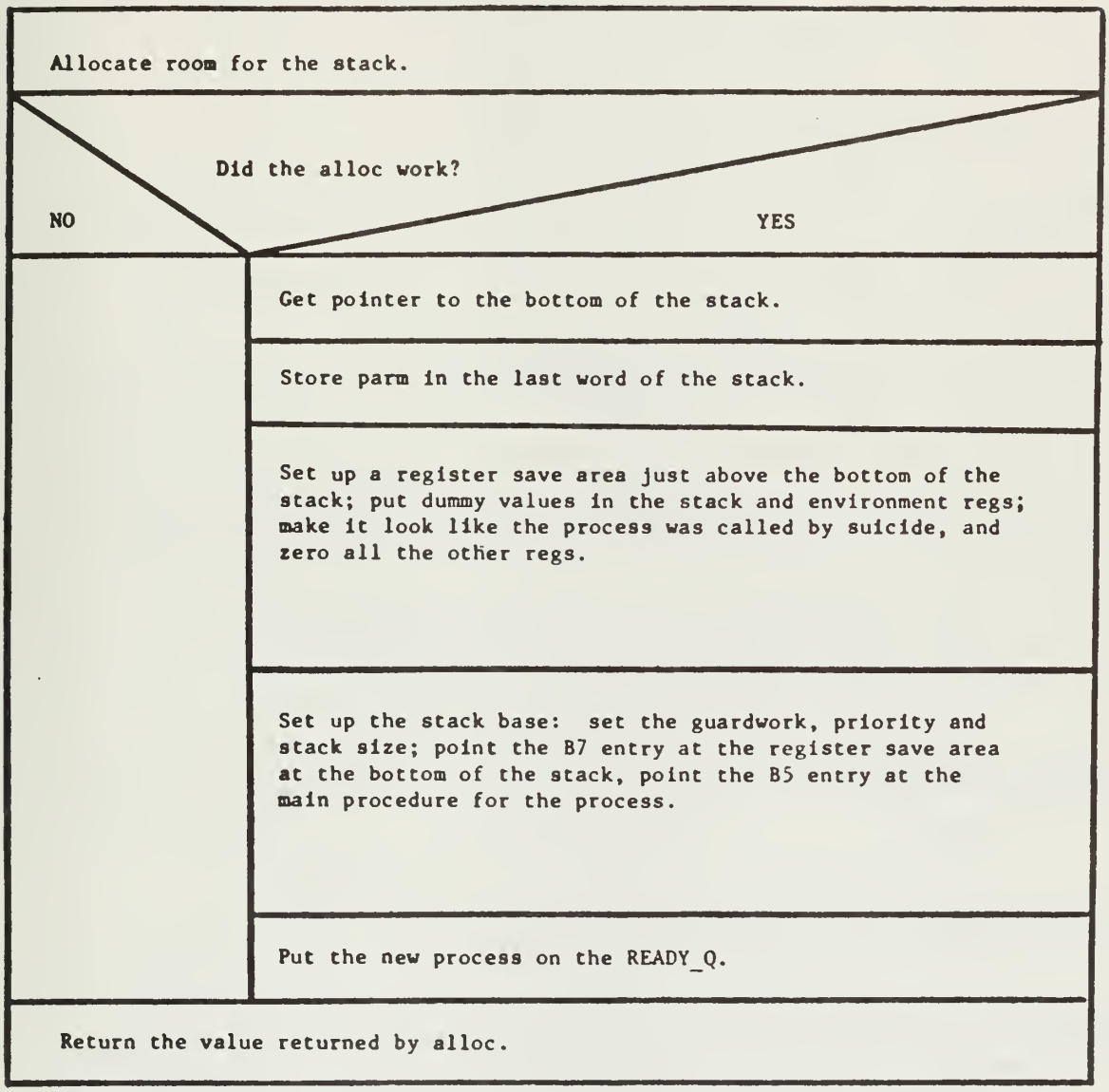


cmp (a, b, length)



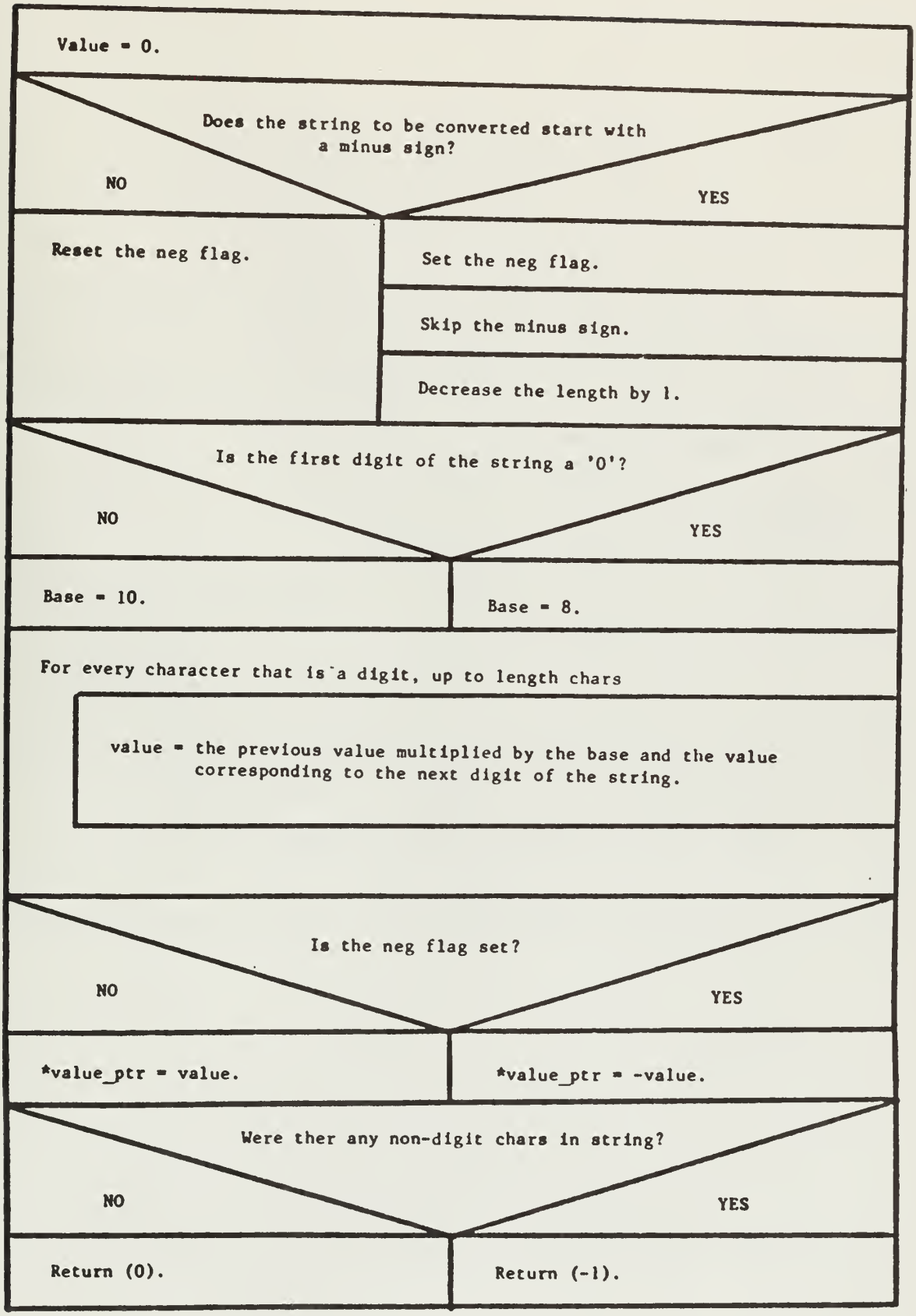
```
create(name, stat_ptr)
Id = open(name, 0, stat_ptr)
Did open succeed?
NO
YES
    Is name a directory?
    NO
    YES
        Close(id, stat_ptr).
        Return (-1).
    Truncate file.
    Did truncate work?
    NO
    YES
    Close(id, stat_ptr).
    Return (-1).
    Update userfib for file to 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.
Did open fail?
NO
YES
    Restore terminated name.
    Return (-1).
Restore terminated name.
a = pointer into name after delimiter after containing directory.
off = 0.
For each entry in the directory.
    Read the directory entry.
    Did the read fail?
    NO
    YES
        Close(id, stat_ptr).
        Return (-1).
    Is directory entry empty?
    NO
    YES
        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
YES
    Close(id, stat_ptr).
    Return (-1).
Write the directory entry structure to the directory.
Did this fail?
NO
YES
    Close(id, stat_ptr).
    Return (-1).
Close the containing directory.
Did this fail?
NO
YES
    Return (-1).
Id = open(name, 0, stat_ptr).
Return (id).
```

creep(stack_size, proc, parm, priority)

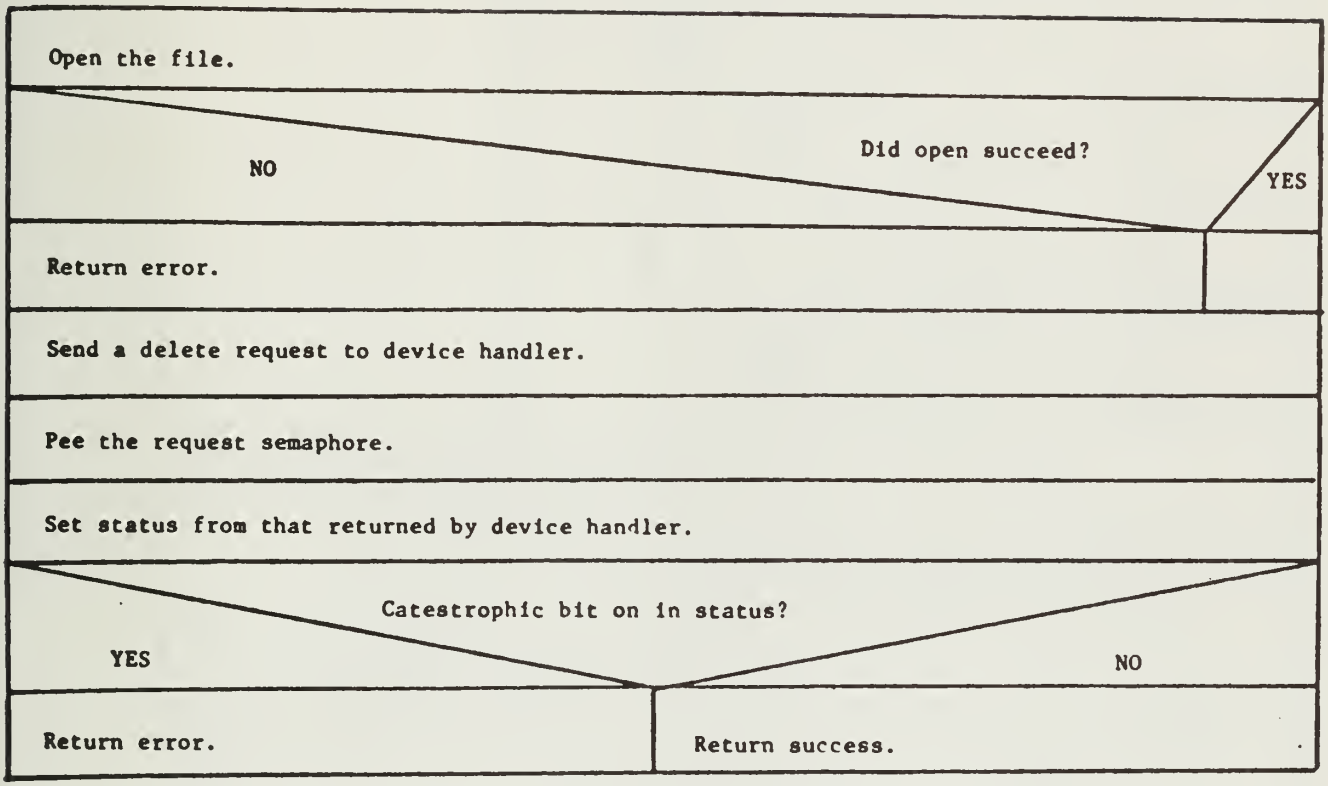


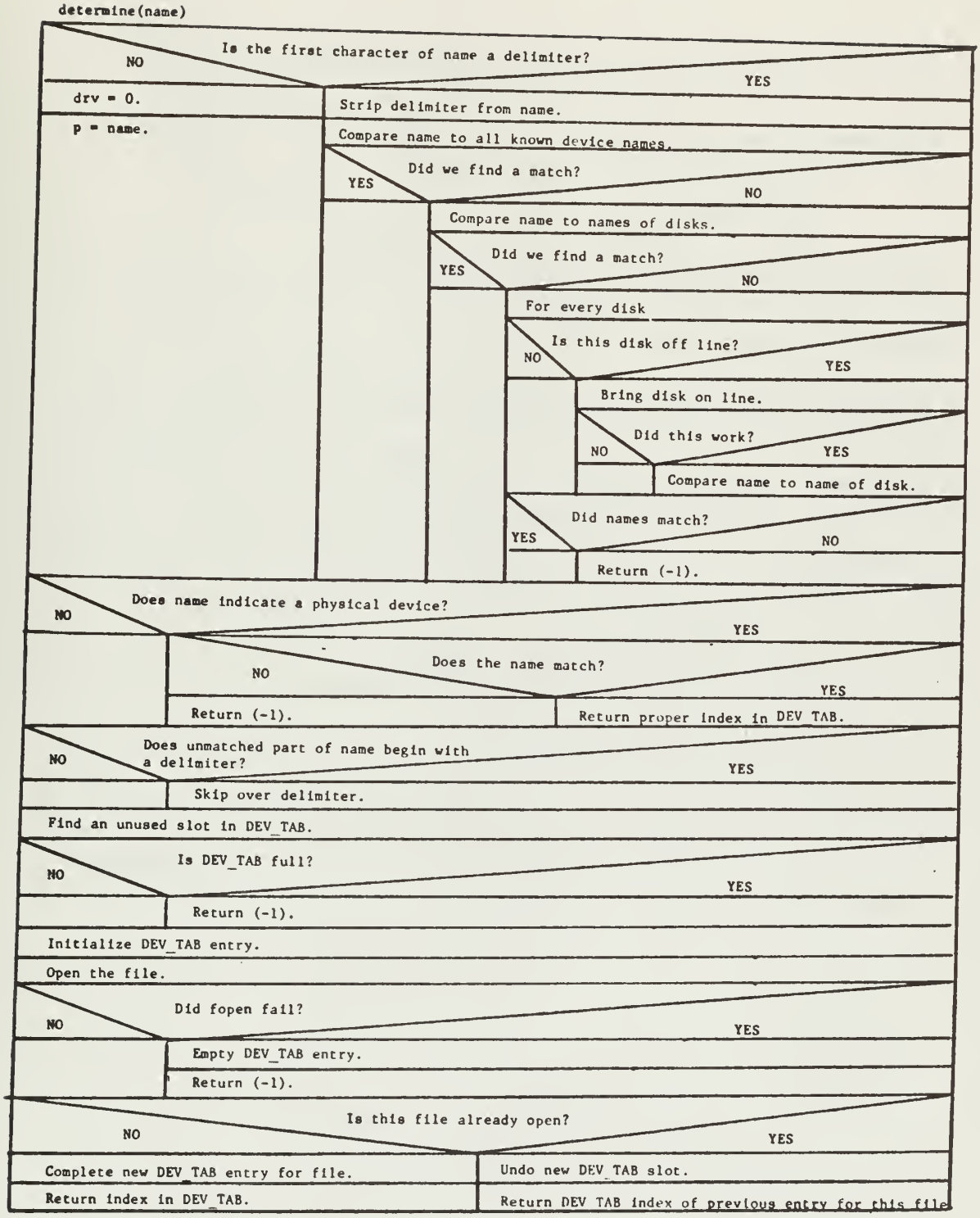
NOTE: This version of creep is specific to the Level 6.

cvb(ptr, length, value_ptr)

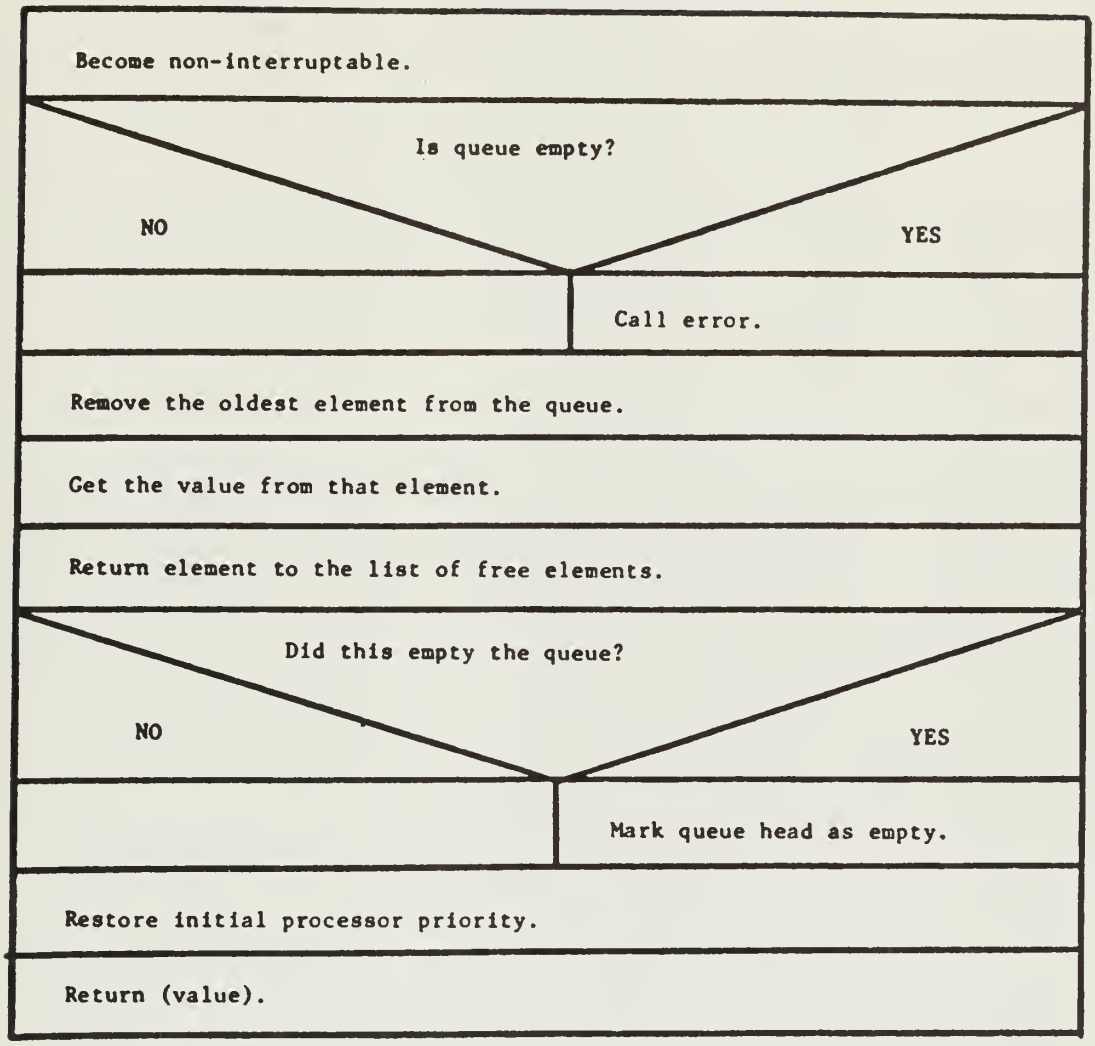


delete (filename, status)

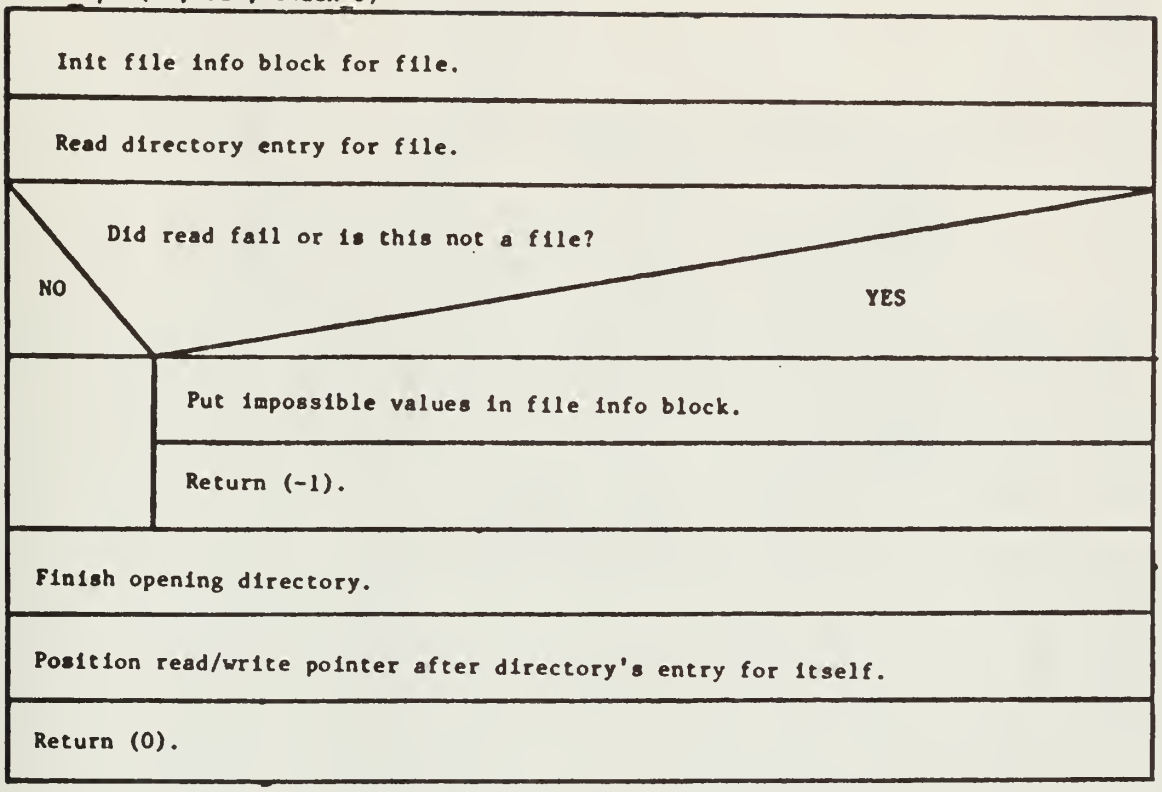



determine(name)
Is the first character of name a delimiter?
NO
YES
drv = 0.
Strip delimiter from name.
p = name.
Compare name to all known device names.
Did we find a match?
YES
NO
Compare name to names of disks.
Did we find a match?
YES
NO
For every disk
Is this disk off line?
NO
YES
Bring disk on line.
Did this work?
NO
YES
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
YES
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

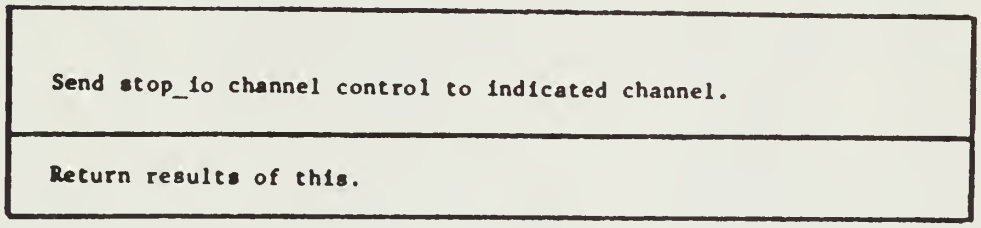
deq(q_ptr)


Become non-interruptable.
Is queue empty?
NO
YES
Call error.
Remove the oldest element from the queue.
Get the value from that element.
Return element to the list of free elements.
Did this empty the queue?
NO
YES
Mark queue head as empty.
Restore initial processor priority.
Return (value).

dir_open(ev, fib, index_b)



disablo_io(chan)



dk_driver()

Do forever

Input next request from DSKQ.

Decode command from request block.

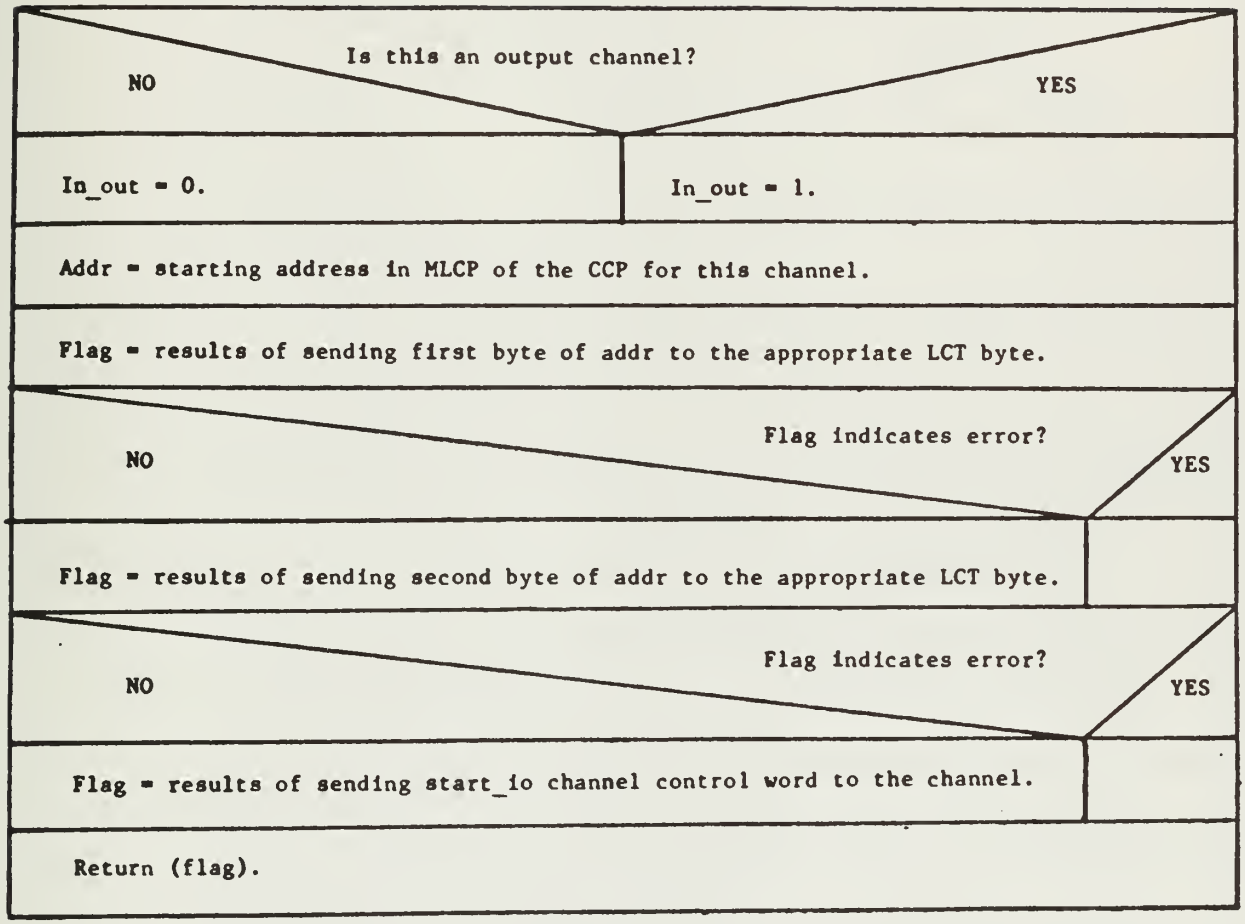
Set user's status word to 0.

Switch (command)

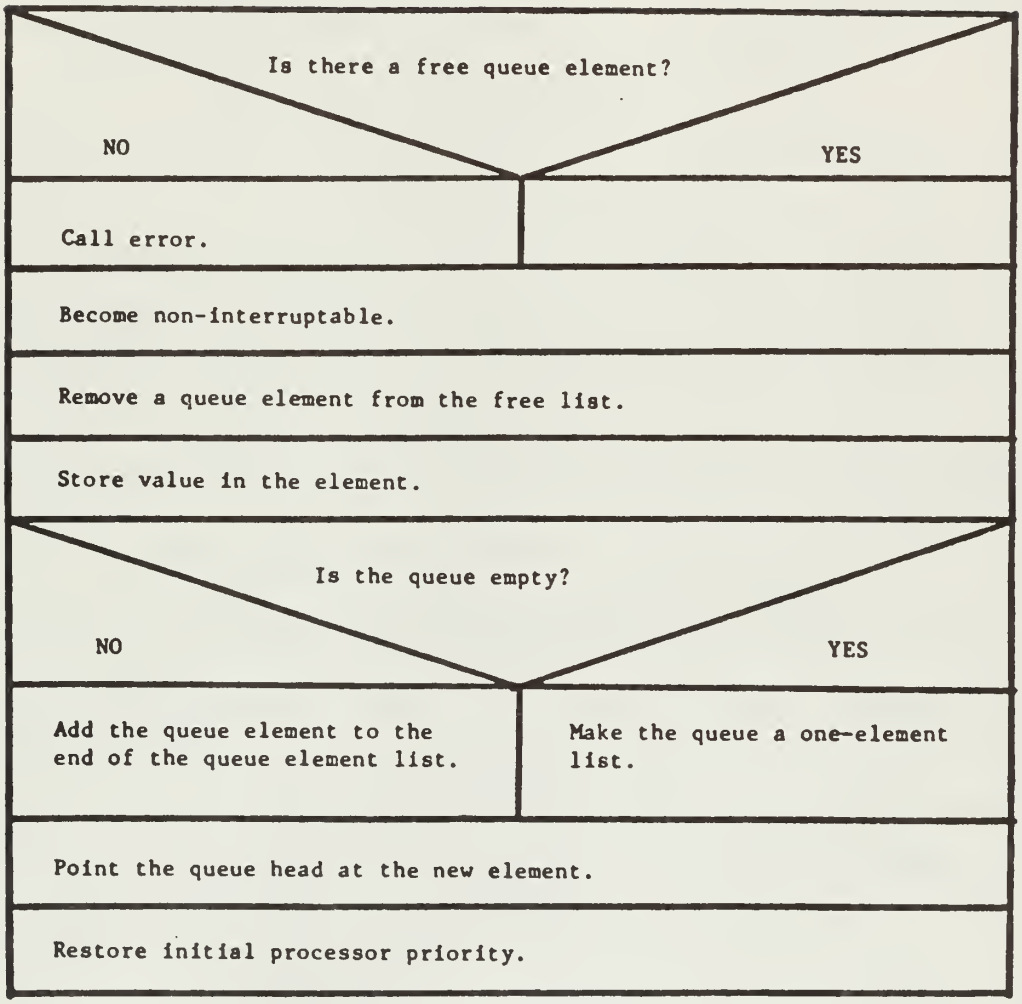
open	close	read	write	peek	flush	delete	default
	Close file.	Read data from file into user's buffer.	Write data from user's buffer to file.		Flush in_memory buffer.	Delete file.	Set reject_reg bit in caller's status
	Did close fail?	Did read fail?	Did write fail?		Did flush fail?	Did delete fail?	
	Set error bits in user's status.	Set error bits in caller's status.	Set error bits in caller's status.		Set error bits in caller's status.	Set error bits in caller's status.	

Vee requesting process.

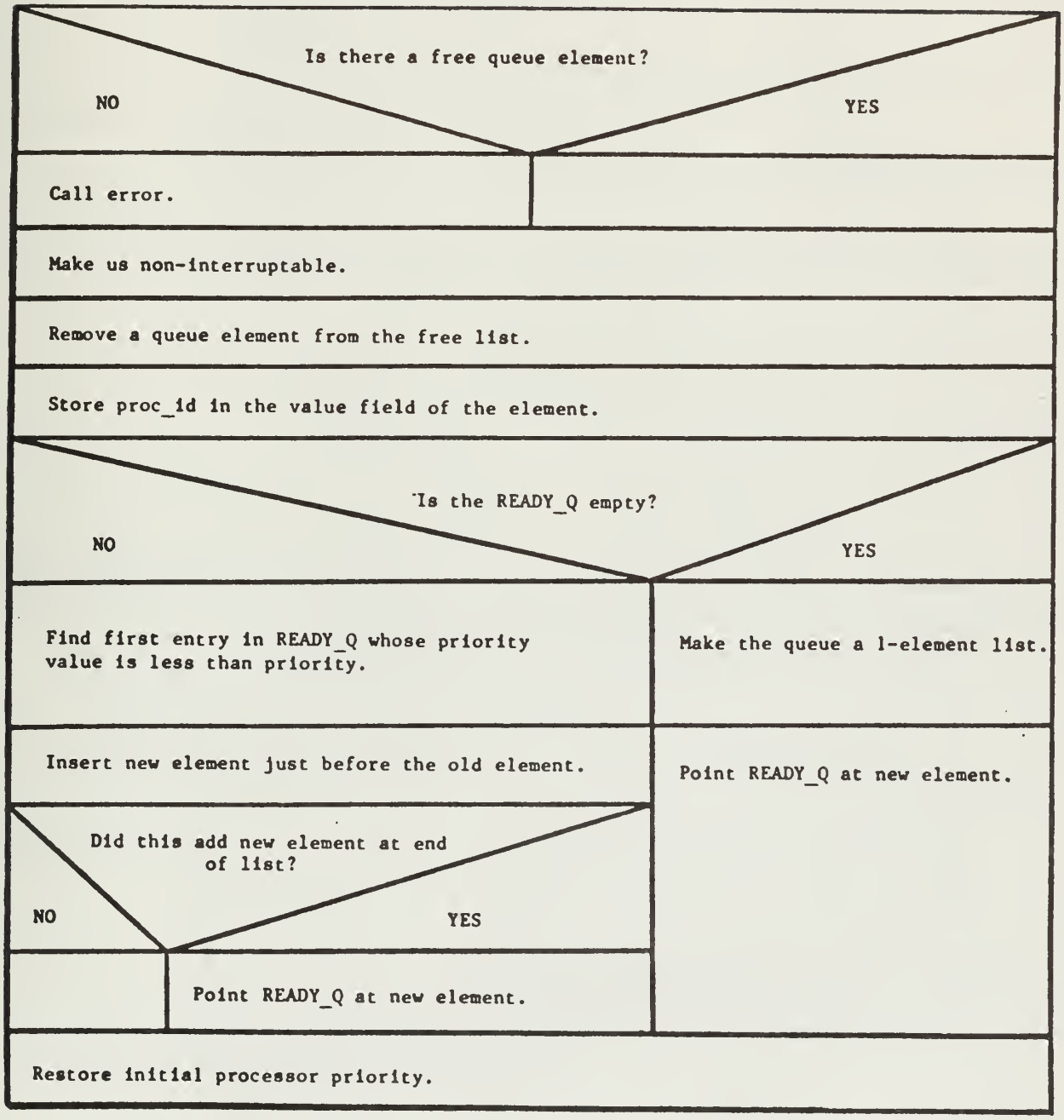
enable_io(chan, dev)



enq(q_ptr, value)



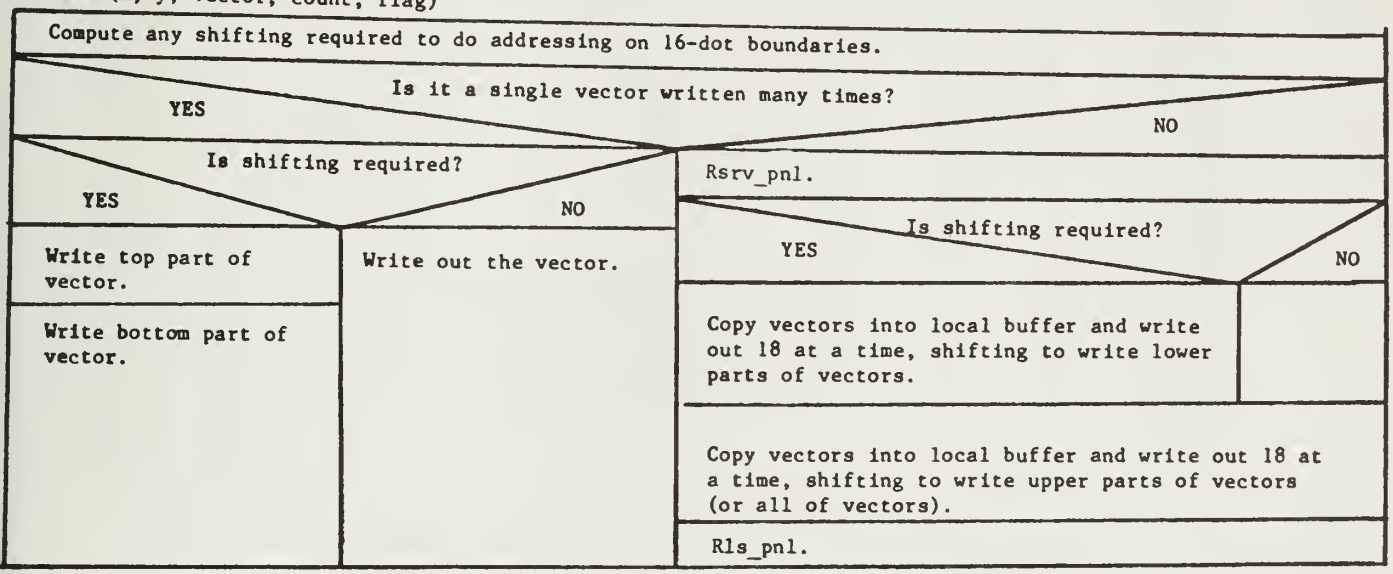
enq_RQ(proc_id, priority)



entry

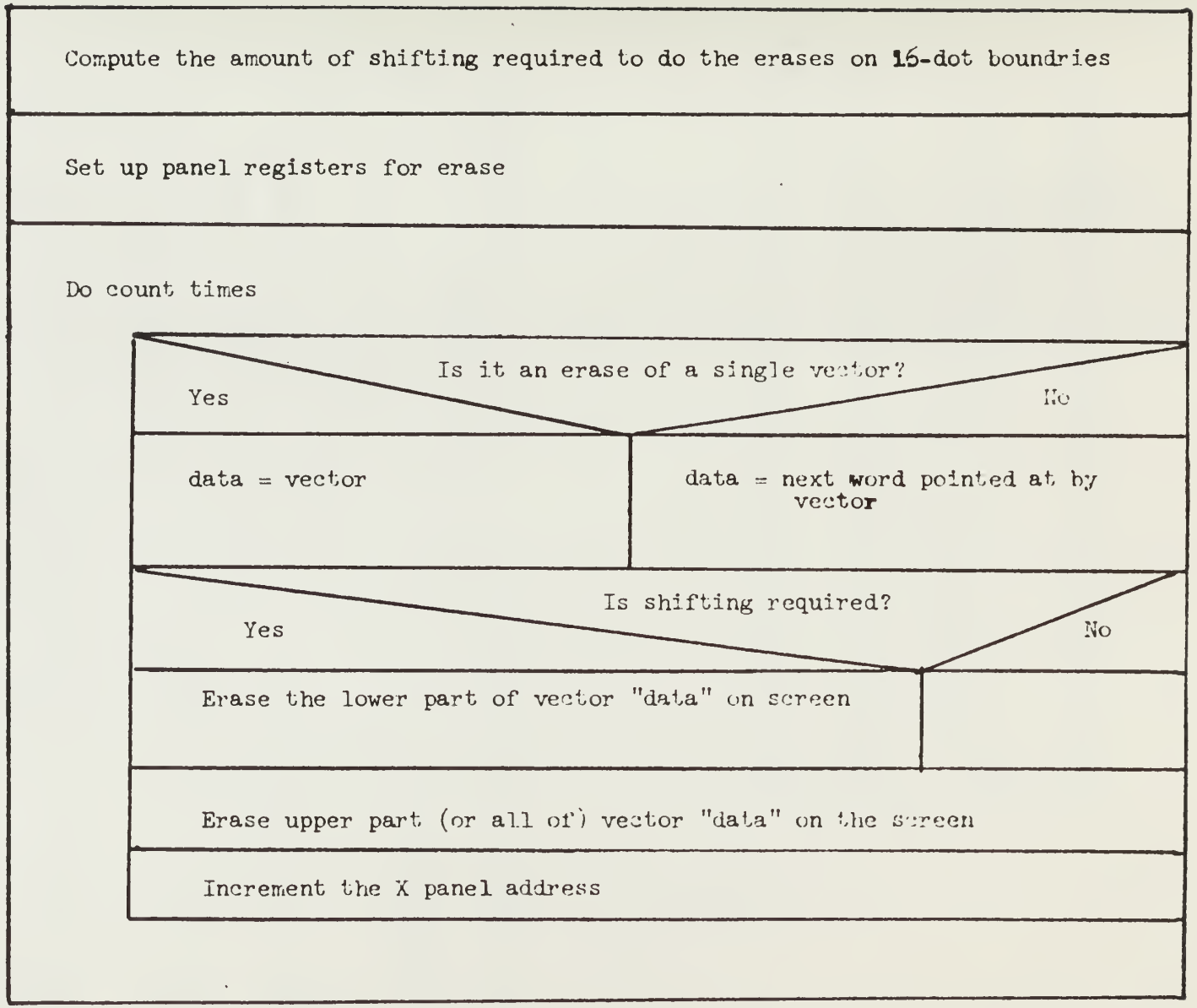
Compute the high end of usable memory.
Set up the stack registers.
Put address of high end of memory onto stack.
Jump to start up.

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



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

erase (X, Y, vector, count, flag)



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

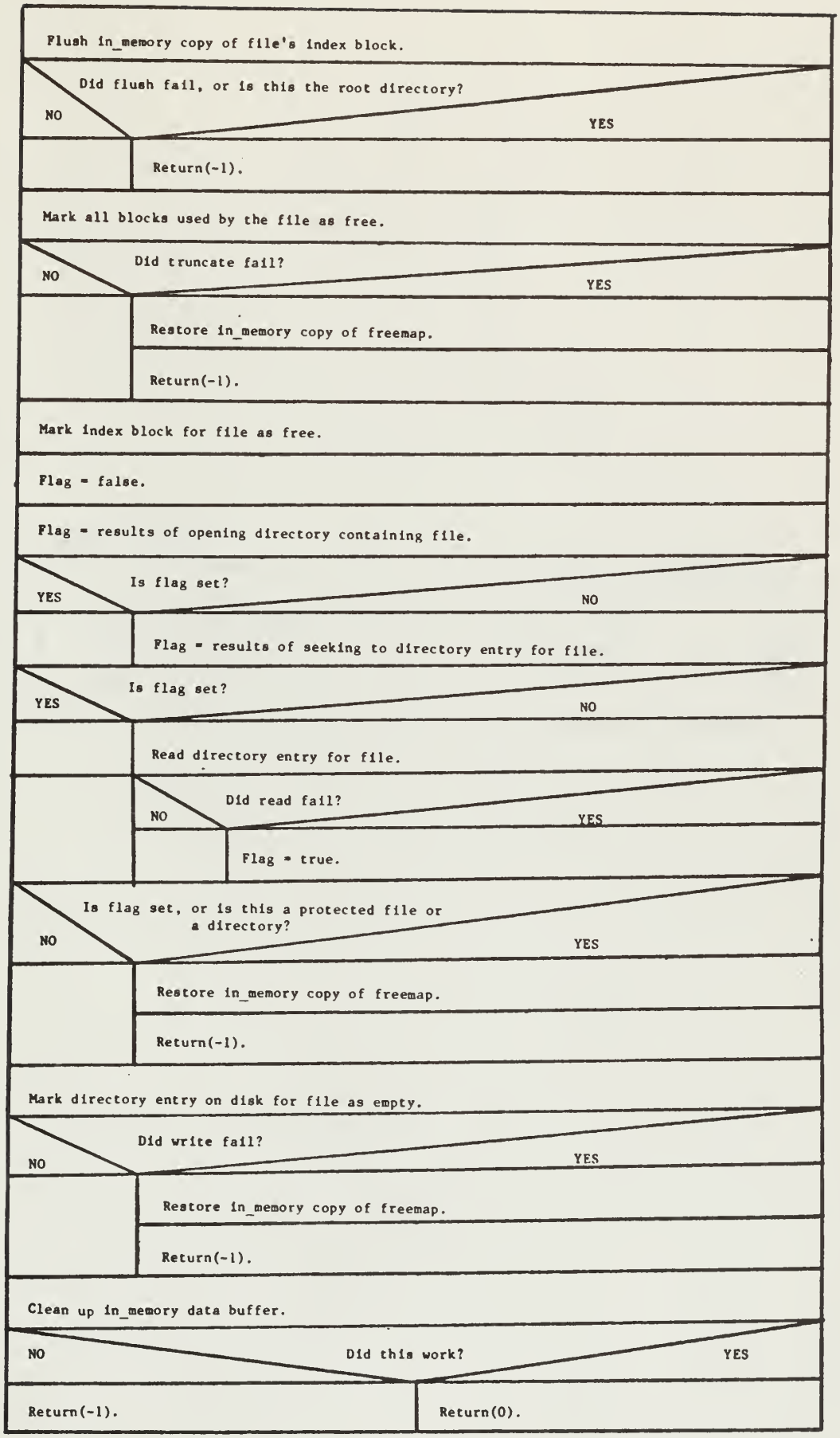
error(err_no)

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

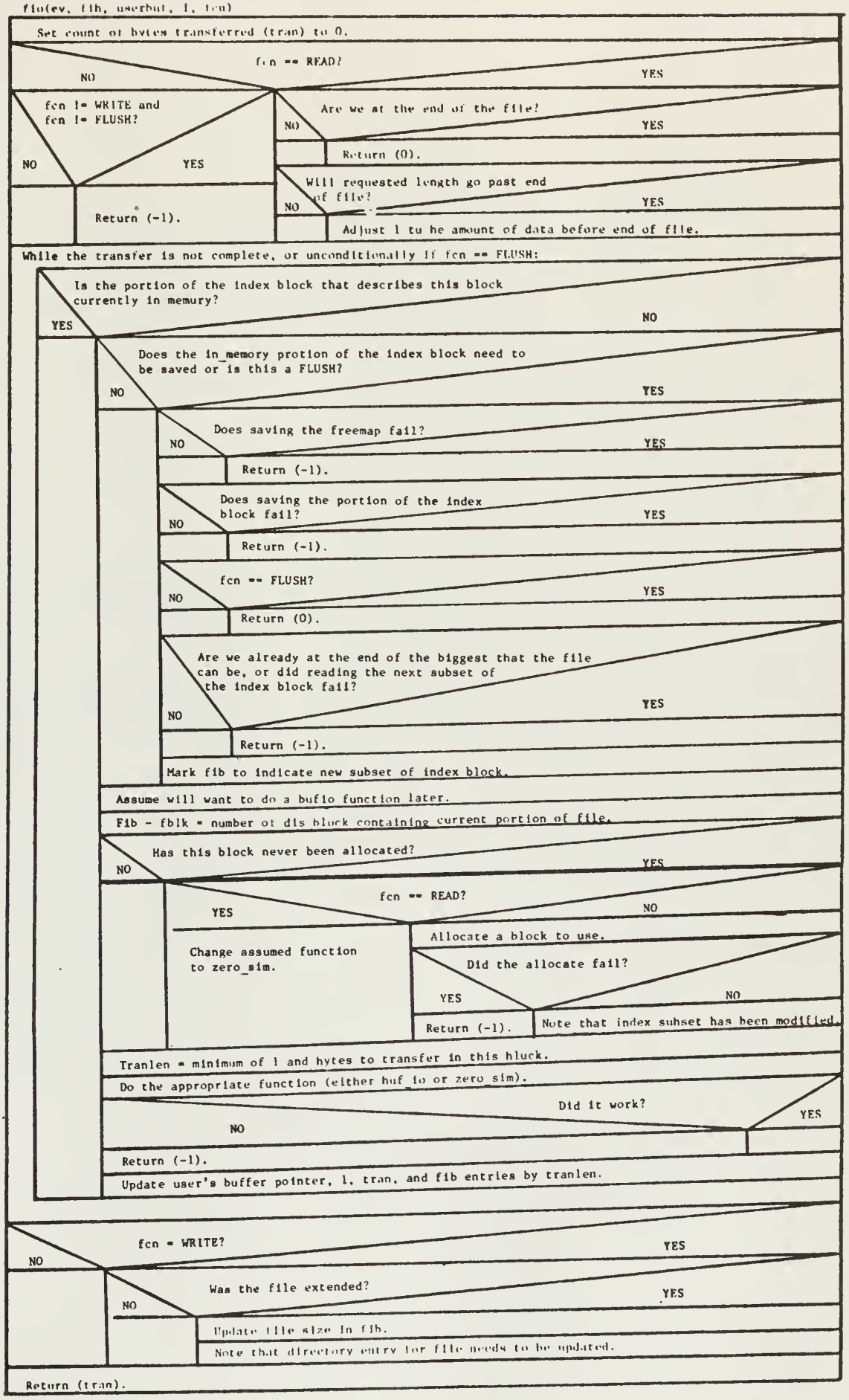
fdelete(ev, fib)



first_block

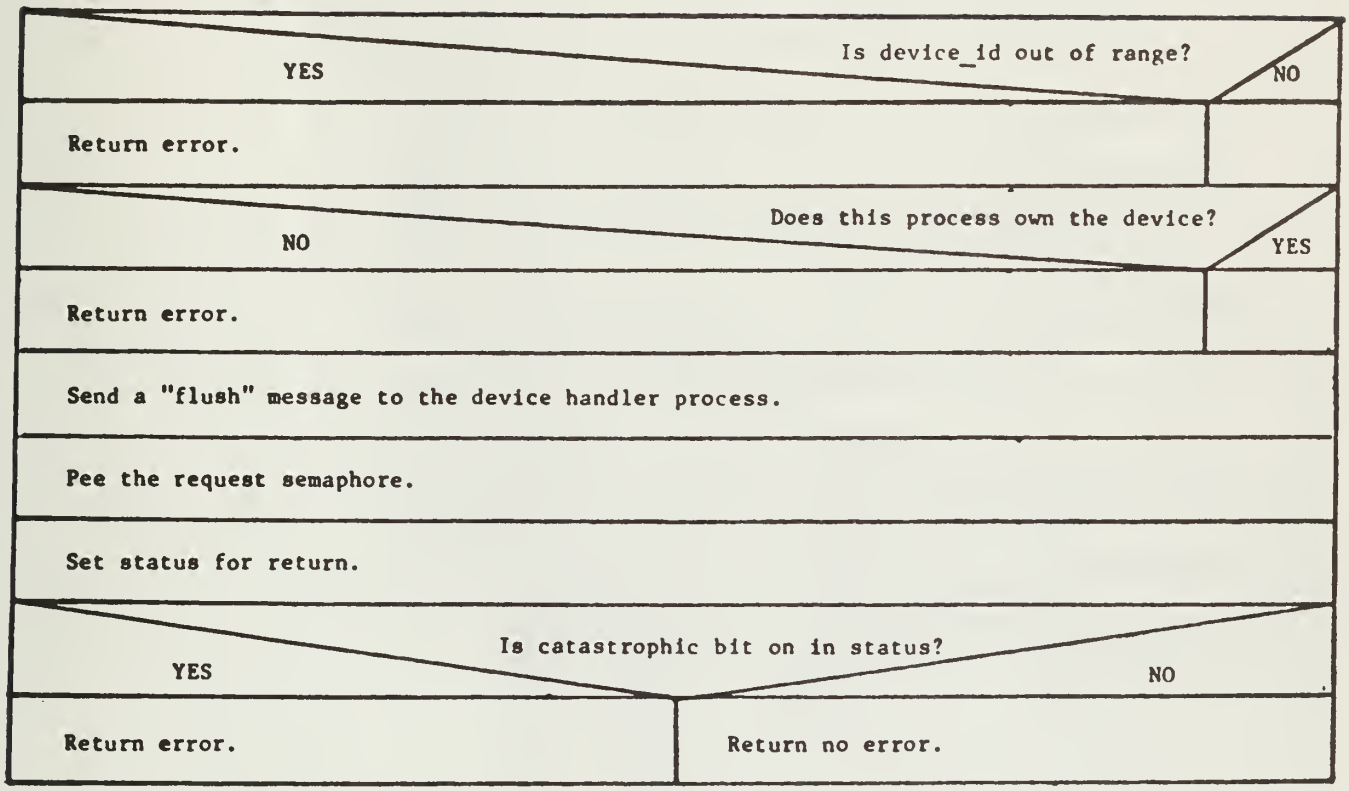
Jump into the middle of block - to the point where it picks up a process from the ready queue.

fio(ev, fib, userbuf, l, fcn)



```
fixup(reserve_size)

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

    Put the address of trap_catcher in memory location 4.

    For (R0 = 0;; R0 =+1024)

        Try accessing memory location whose address is in R0 (will goto trap_catcher
        when try to access a non-existent location).

    Count_down:
            Decrement R0 and try accessing that location.

    Decrement R0 by 1.

    Subtract reserve_size from R0.

    Copy R0 to R5.

    R6 = R0 - 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.
```

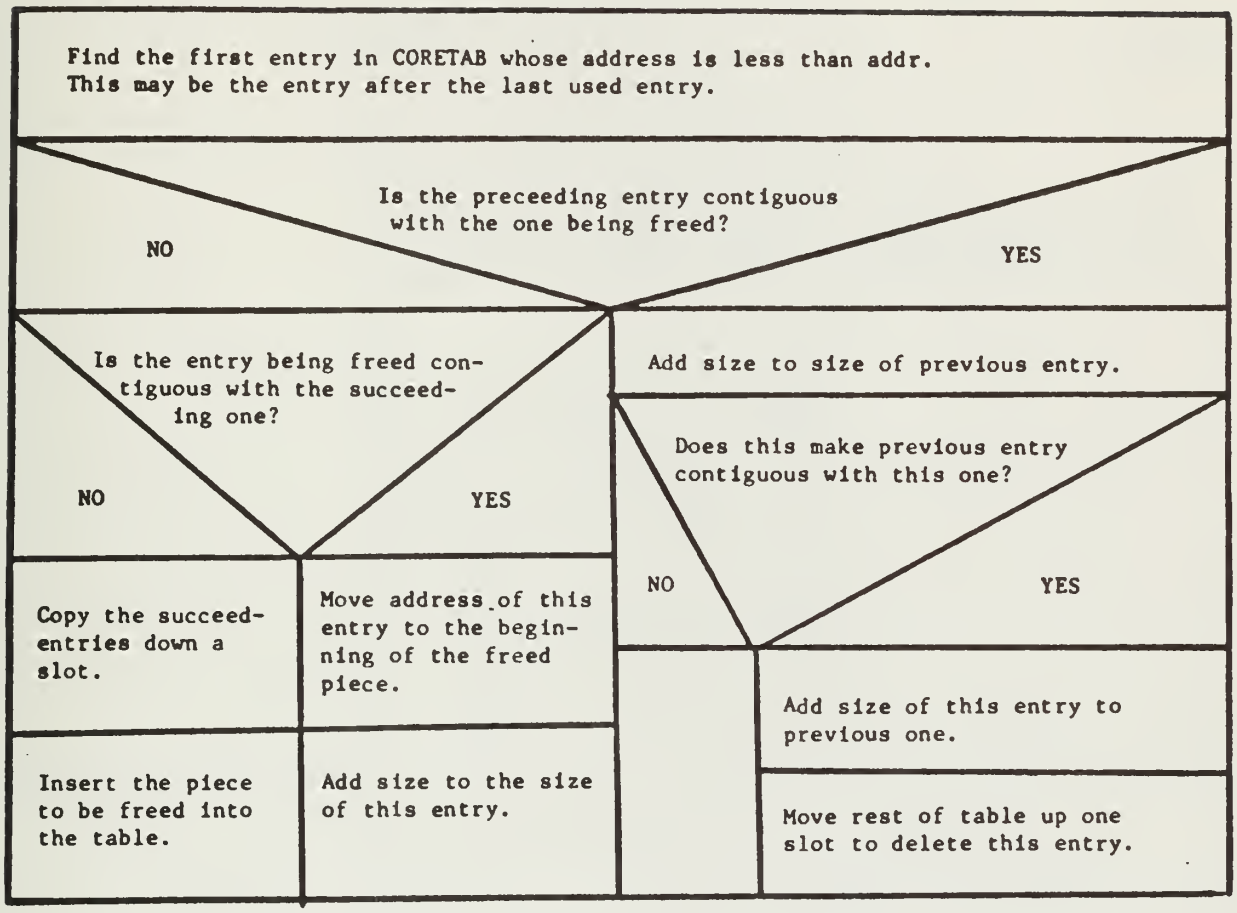
flush(device_id, st_ptr)



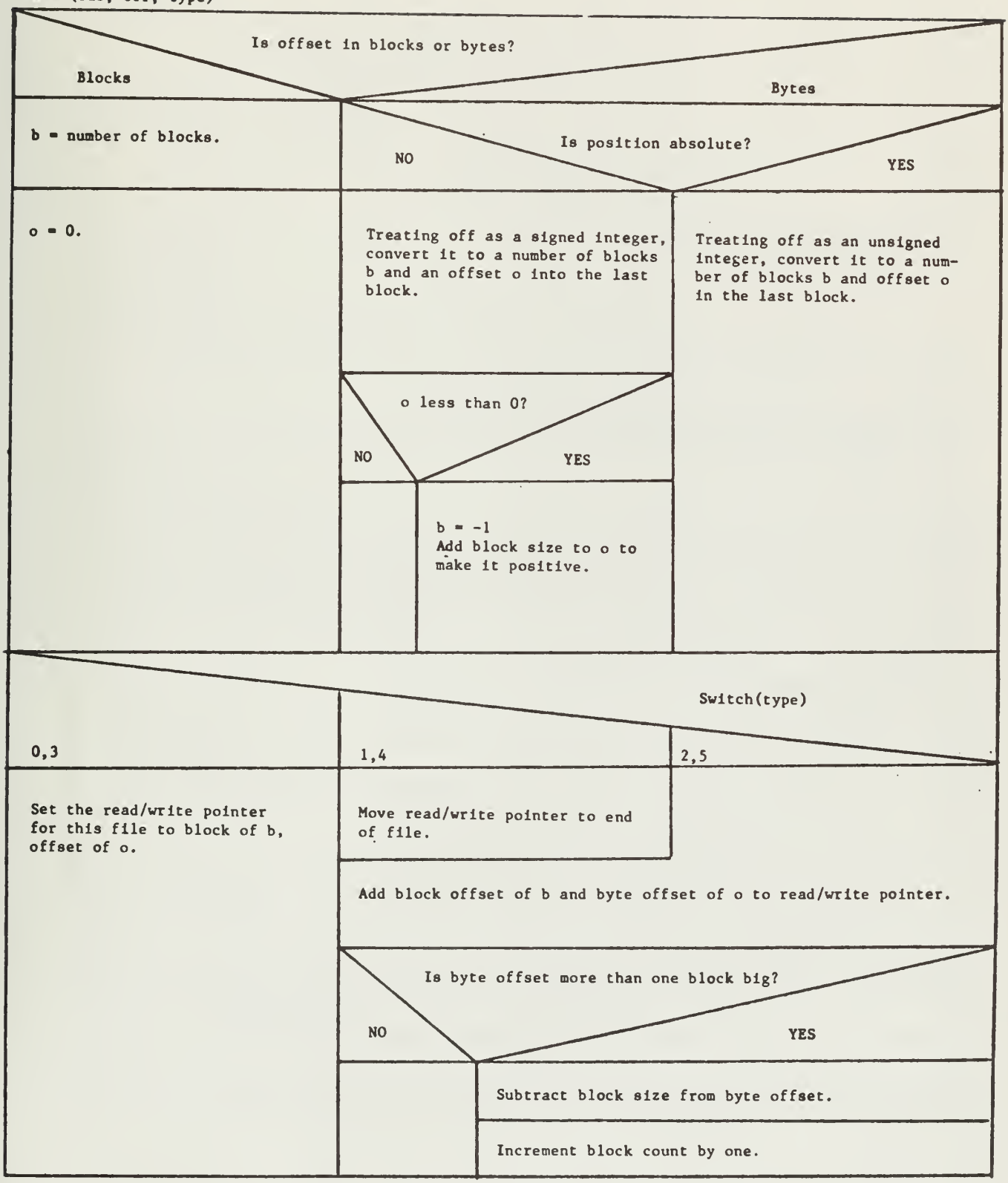
fopen(name, fib, drv, root)

```
Dev = address of buffer structure for this drive.

Loop forever
    Open file whose index block is root as a directory.
    Did dir_open fail?
        NO:
            YES:
                Return(-1).
    Hsve we reached the end of the name?
        NO:
            YES:
                Set file offset in fib to 0.
                Return(0).
    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).
        Is this directory entry empty?
            NO:
            YES:
                Continue.
        New = results of using pathname to compare name to name for this entry.
        Did they match at all?
            NO:
            YES:
                Did they mstch up to a delimiter in name?
                    NO:
                    YES:
                        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.
                        Did they match up to the end of name?
                            NO:
                            YES:
                                Is this entry a directory?
                                    NO:
                                    YES:
                                        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)



fseek(fib, off, type)



ftrunc(ev, fib)

Write out the index block for this file.

Max = number of blocks in file.

Has the last block been allocated?

NO

YES

Decrement max.

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

Are there any data blocks in this file?

NO

YES

Return(0).

For every block in file:

Read in the number of disk block for this block.

Did read work?

NO

YES

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.

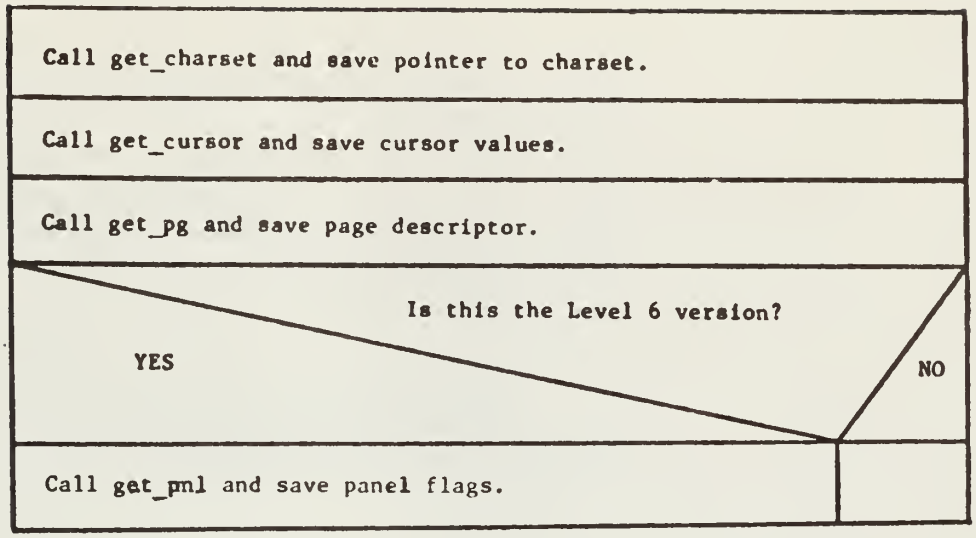
NOTE: This version of get_cursor is specific to the LSI-11 IT.

get_cursor (x_ptr, y_ptr)

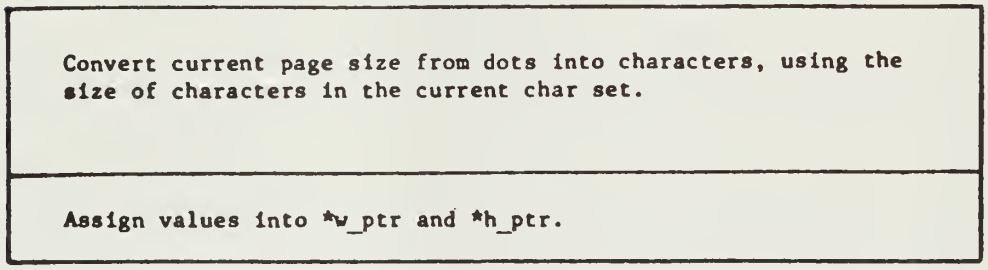
Read cursor from the display head.
Convert values returned to character offsets.

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

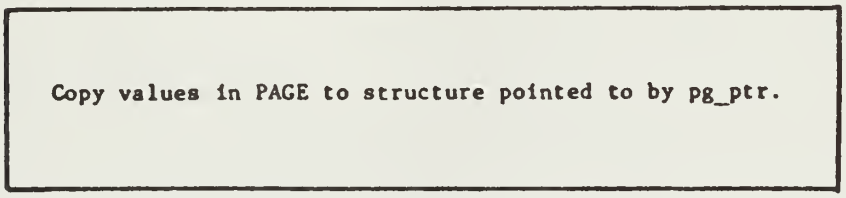
get_env (env_ptr)



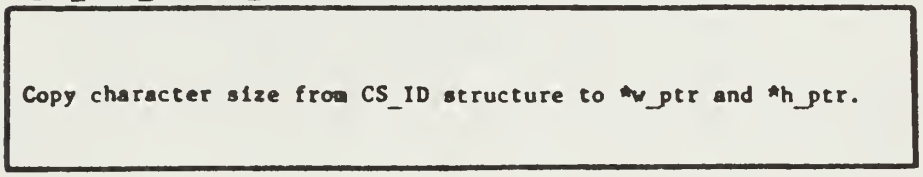
get_page_size(w_ptr, h_ptr)



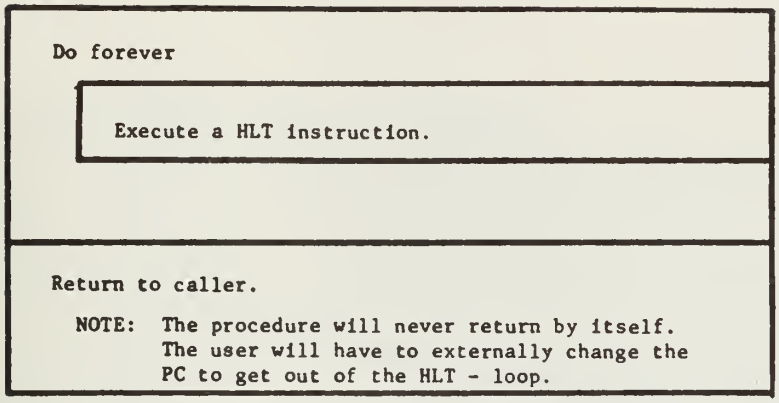
get_pg(pg_ptr)



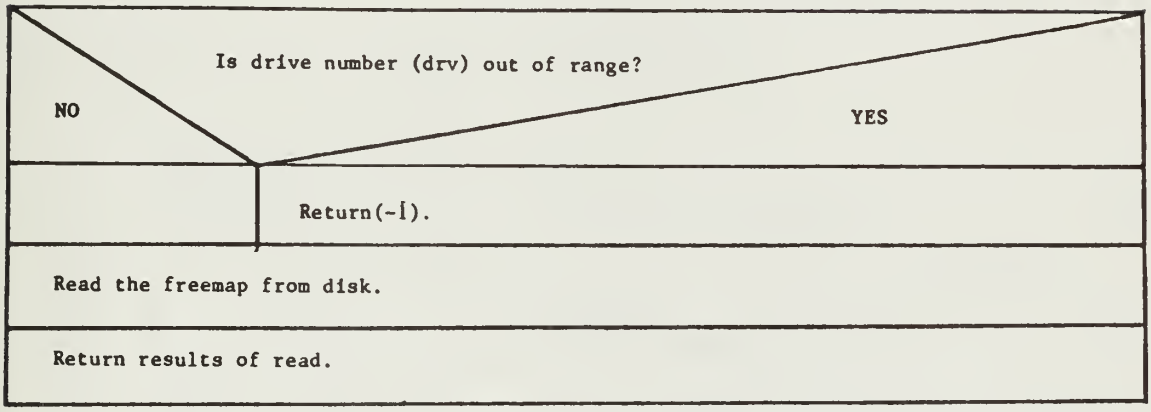
get_size_chars(w_ptr, h_ptr)



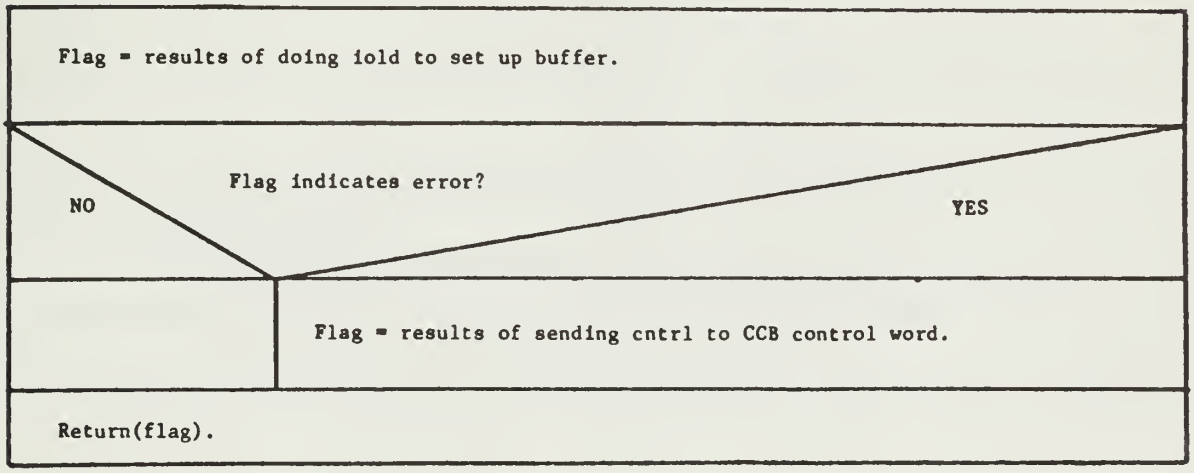
halt



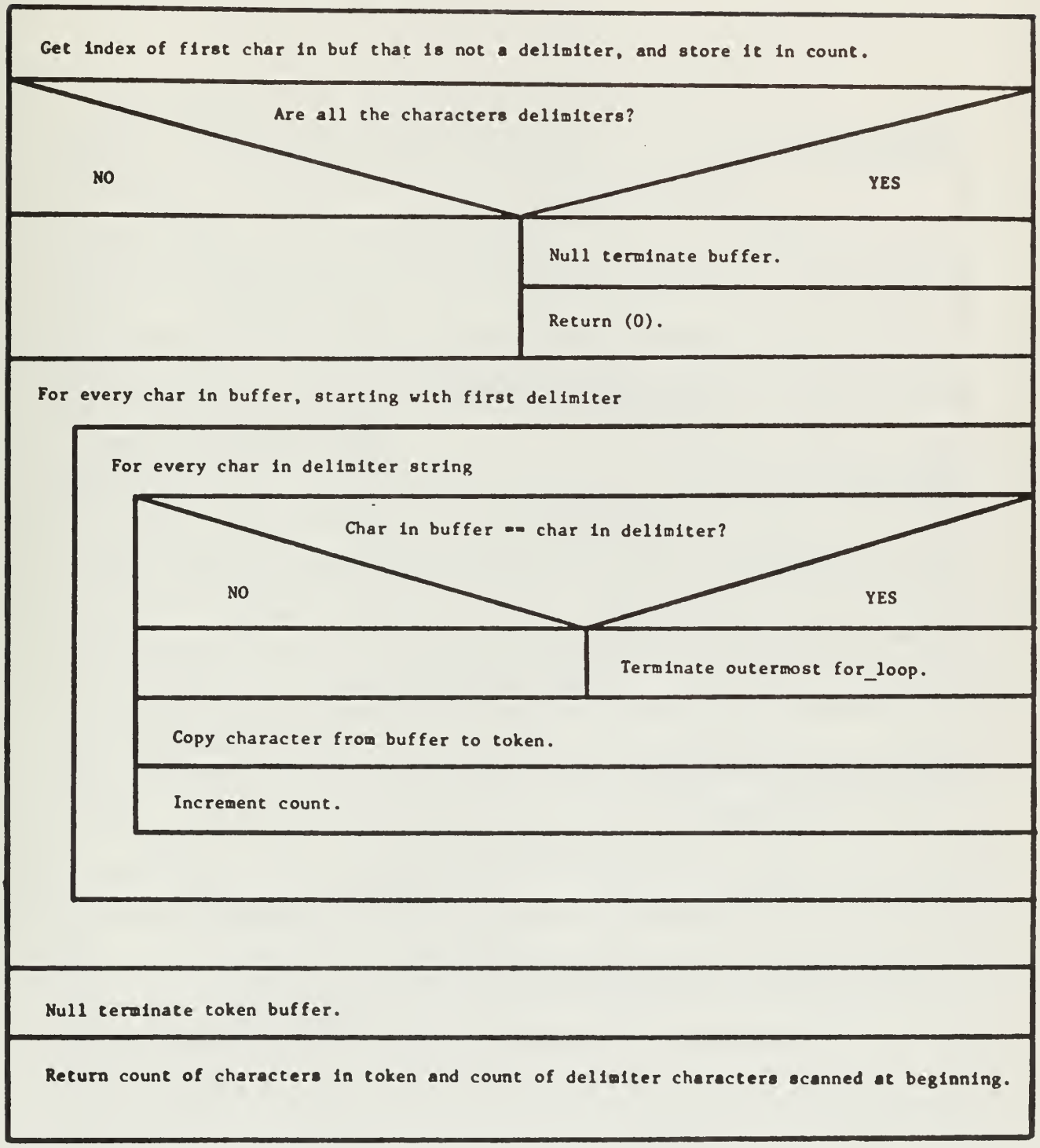
i_freemap(drv)



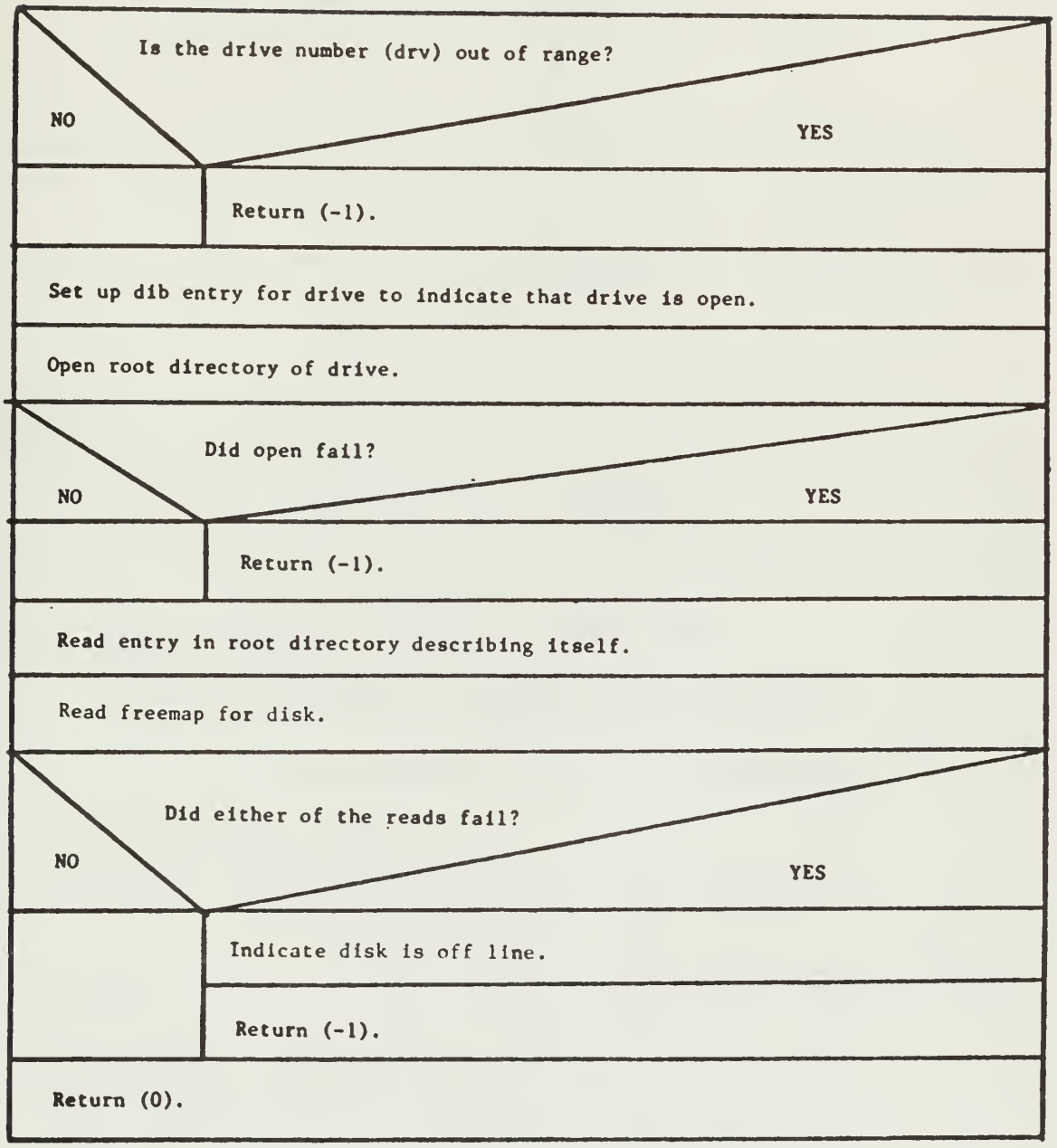
init_ccb(chan, buf, buf_size, cntrl)



get_token(buf_ptr, tok_ptr, delim_ptr)



init_drive(drv)



init_fib(fib, index_b, off)

Assign appropriate values to elements of file information block.
Return (0).

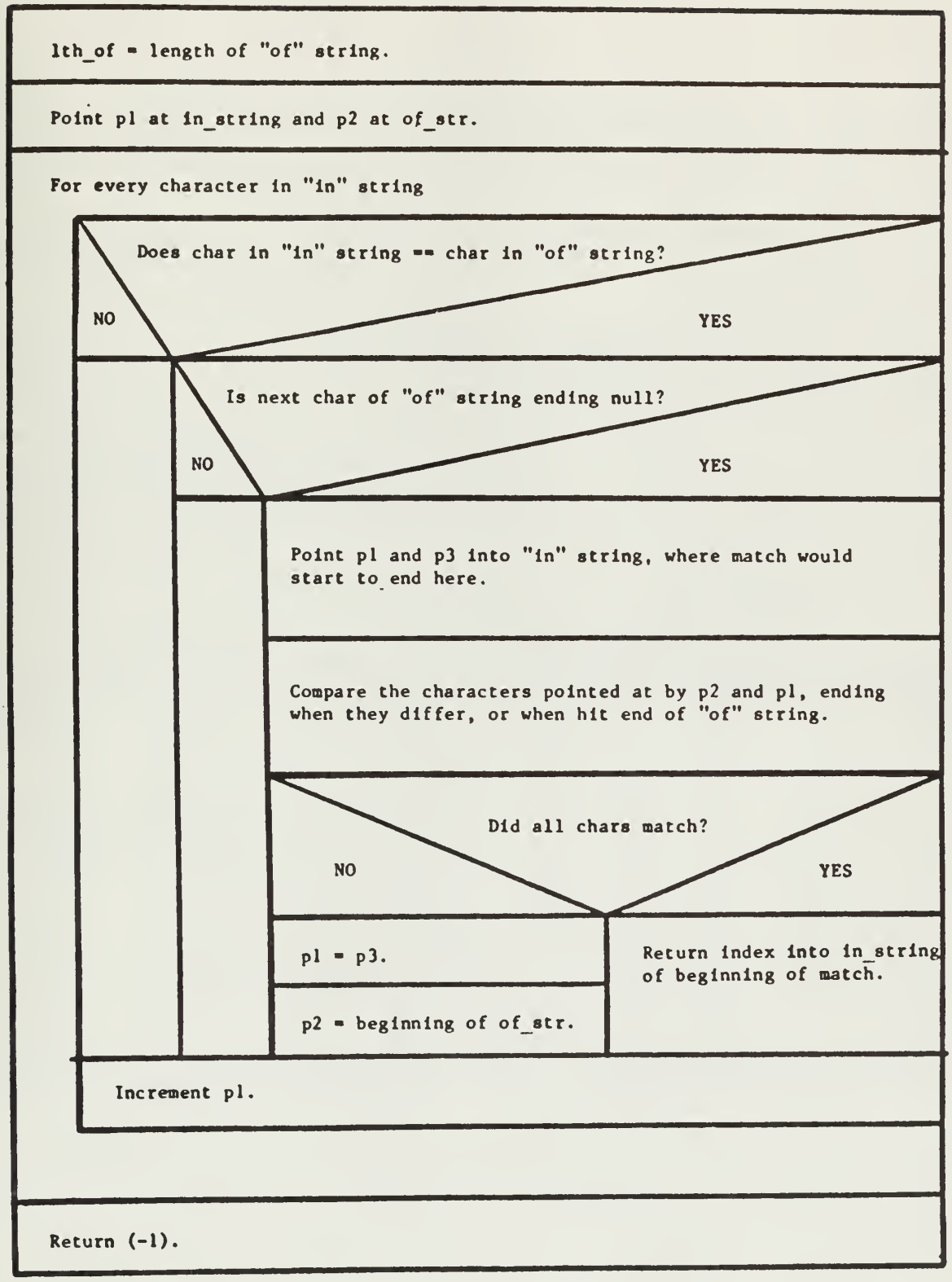
io_init()

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	NO
Flag the charset as not loaded	
Has the microcode already been loaded? NO	YES
Write the microcode to the Z80 panel controller	
Free the space occupied by the microcode in the Level 6 memory.	
Initialize the Z80's internal variables.	

index(in_str, of_str)



b_driver()

Enable keyboard input.

Do forever

Read a word from the input queue.

Is the word odd?

YES

Request_code = "data."

Is high byte of the word a CR?

YES

Input data character is newline.

NO

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

NO

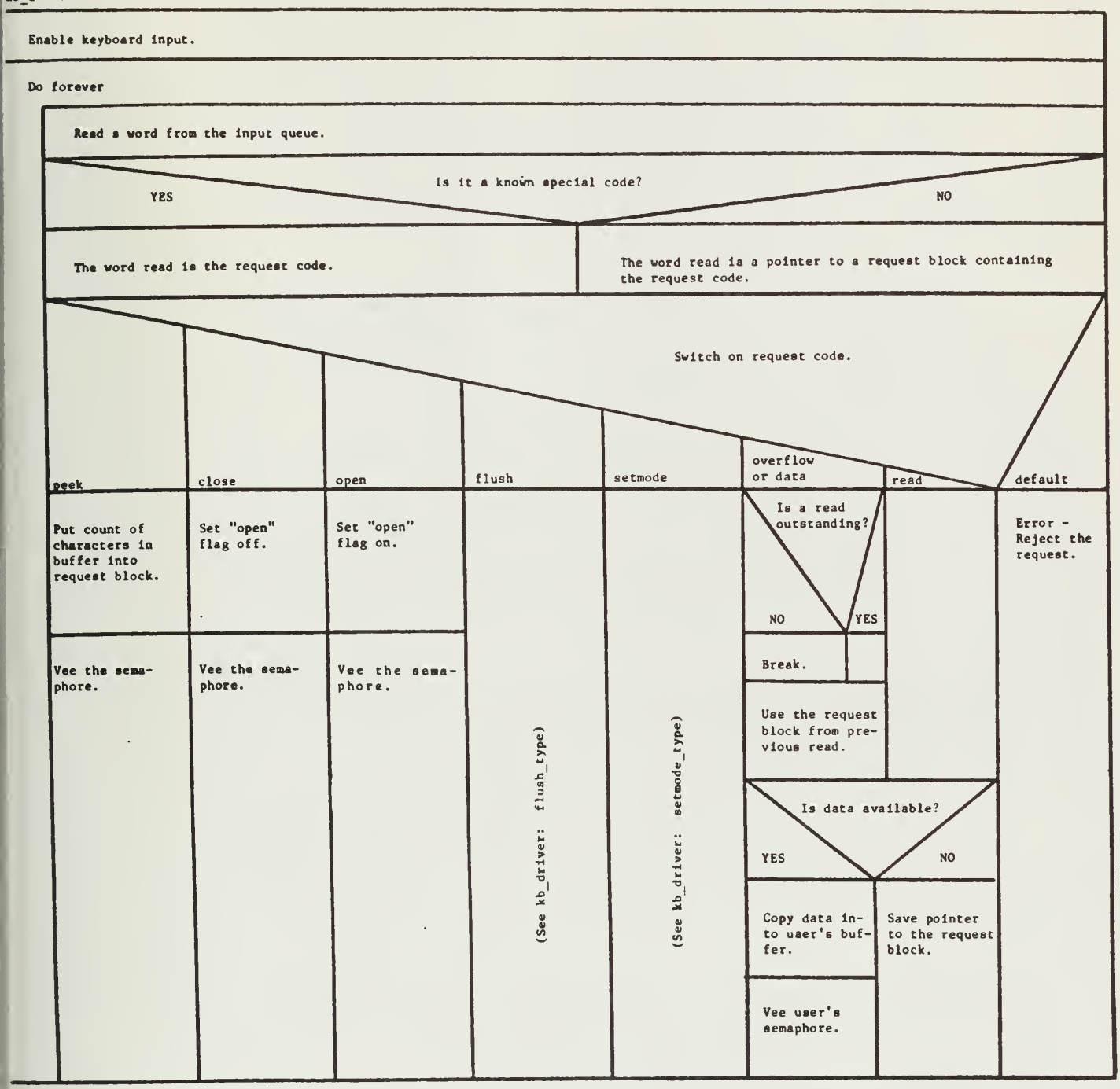
The word is a pointer to a request block containing the request code.

Switch on request code.

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.	(See kb_driver: flush_type)	(See kb_driver: setmode_type)	(See kb_driver: data_type)	(See kb_driver: read_type)	Error: Reject the request.
Vee the semaphore.	Vee the semaphore.	Set x - y address for echoing to default.					
		Vee the semaphore.					

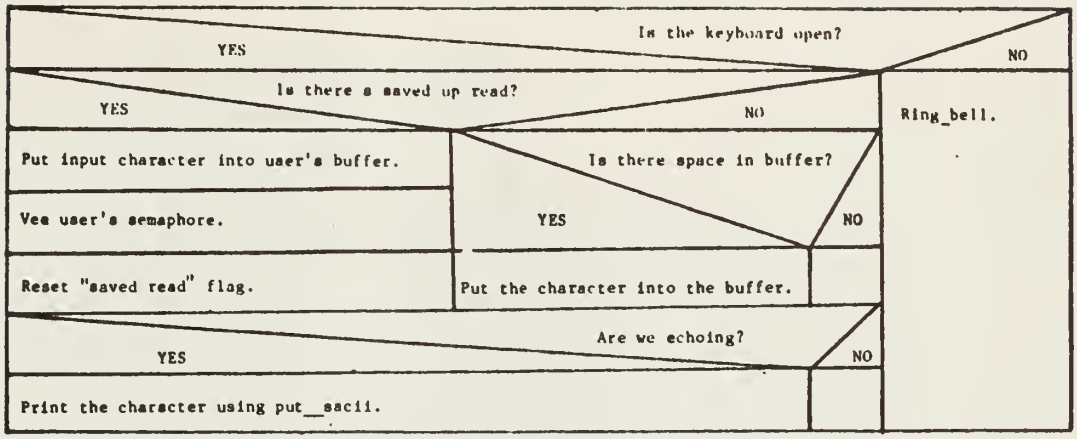
NOTE: This version of kb_driver is specific to the LSI 11 IT.

kb_driver



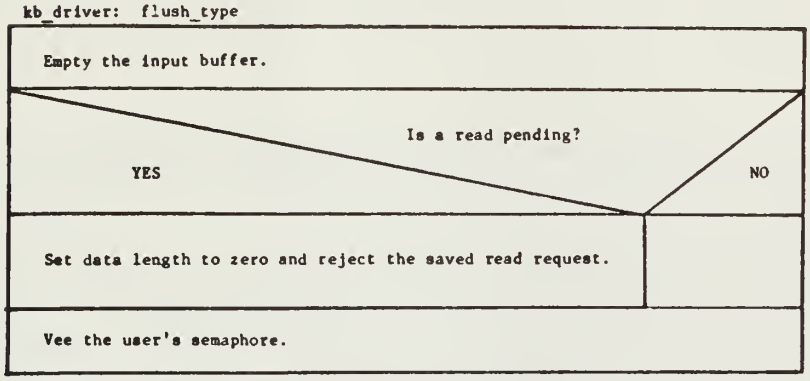
NOTE: This version of kb_driver is specific to the Level 6.

kb_driver: data type*



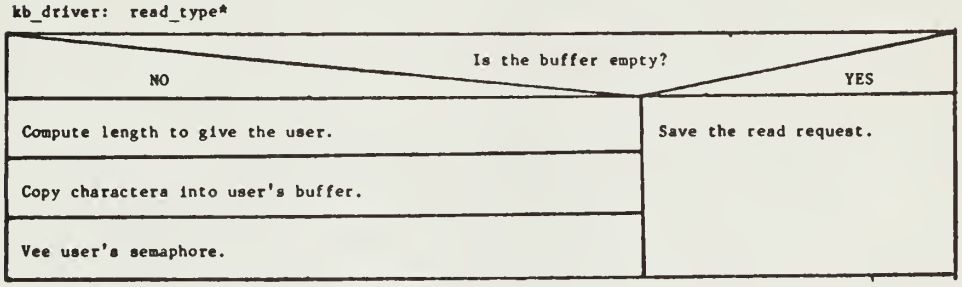
*This is not a procedure. It is one case in kb_driver.

kb_driver: flush_type



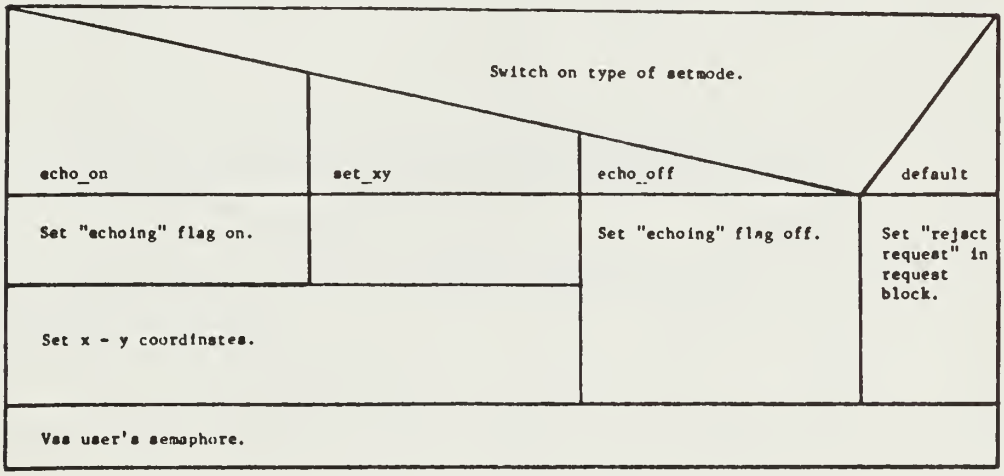
NOTE: This is not a procedure. It is one case in kb_driver.

kb_driver: read_type*

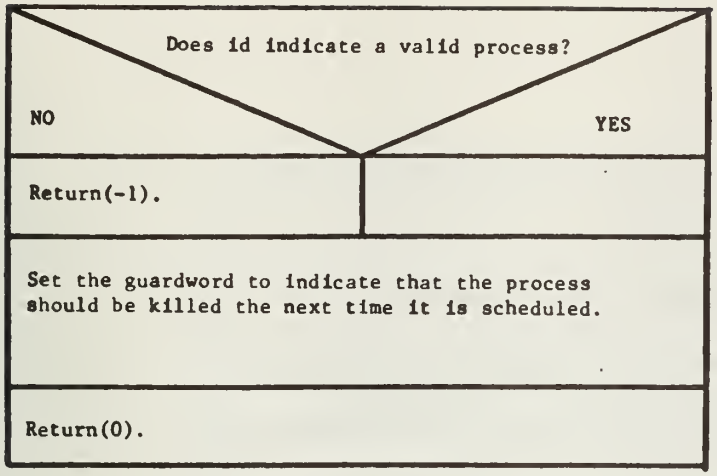


*This is not a procedure. It is one case in kb_driver.

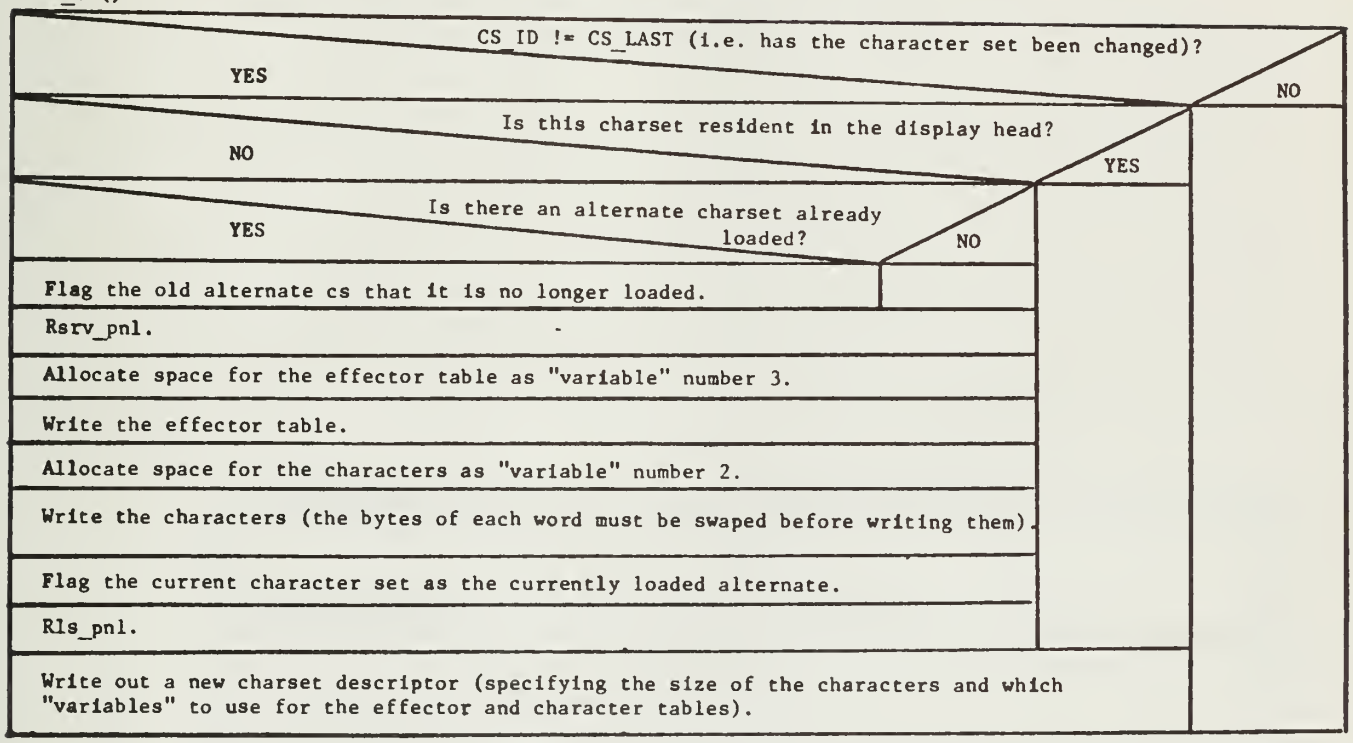
kb_driver: setmode_type



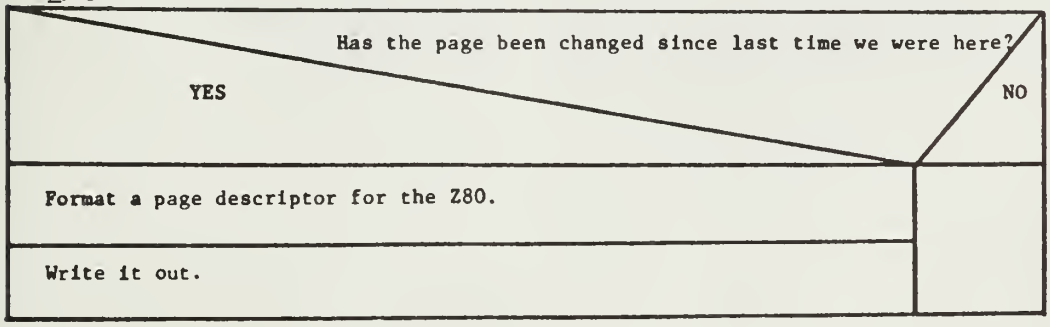
kill(proc_id)



ld_cs()



ld_page()



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.

ln_xpand(out, in, p1, p2, p3,. . .)

Length = 0.

Next_parm = address of first parameter (p1).

While there is still something in the input format specification

Is the next char in the input format specifier a parameter replacement indicator?

NO	YES
Copy the char from the input buffer to the output buffer.	ptr = parm_xpand (pointer to format specifier, and next_parm, and skipped).
Move the input buffer over one character.	Increment in pointer by skipped.
Increment length by one.	Copy the string pointed to by ptr into the output buffer, and increment length by the length of this string.

Null terminate the output string.

Increment length to include the trailing null.

Return length.

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.

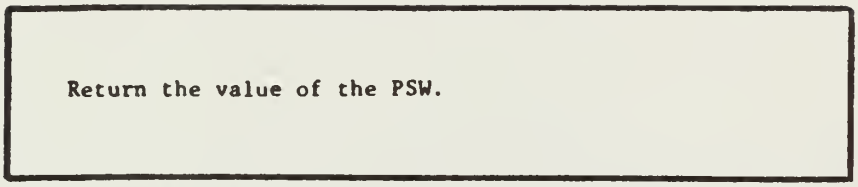
NOTE: This version of lrem is specific to the LSI 11 IT.

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.

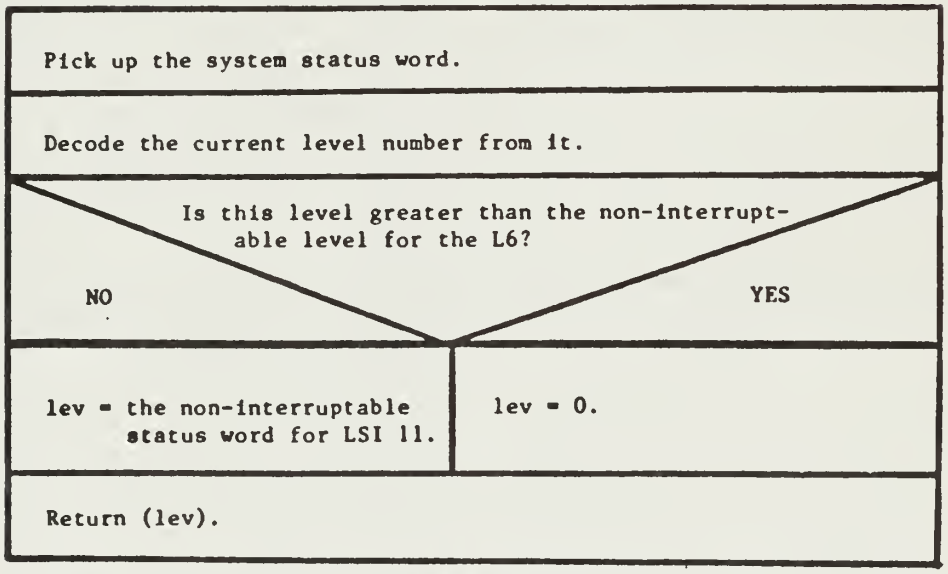
NOTE: This version of lrem is specific to the L6 IT.

mfps()



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

mfps()

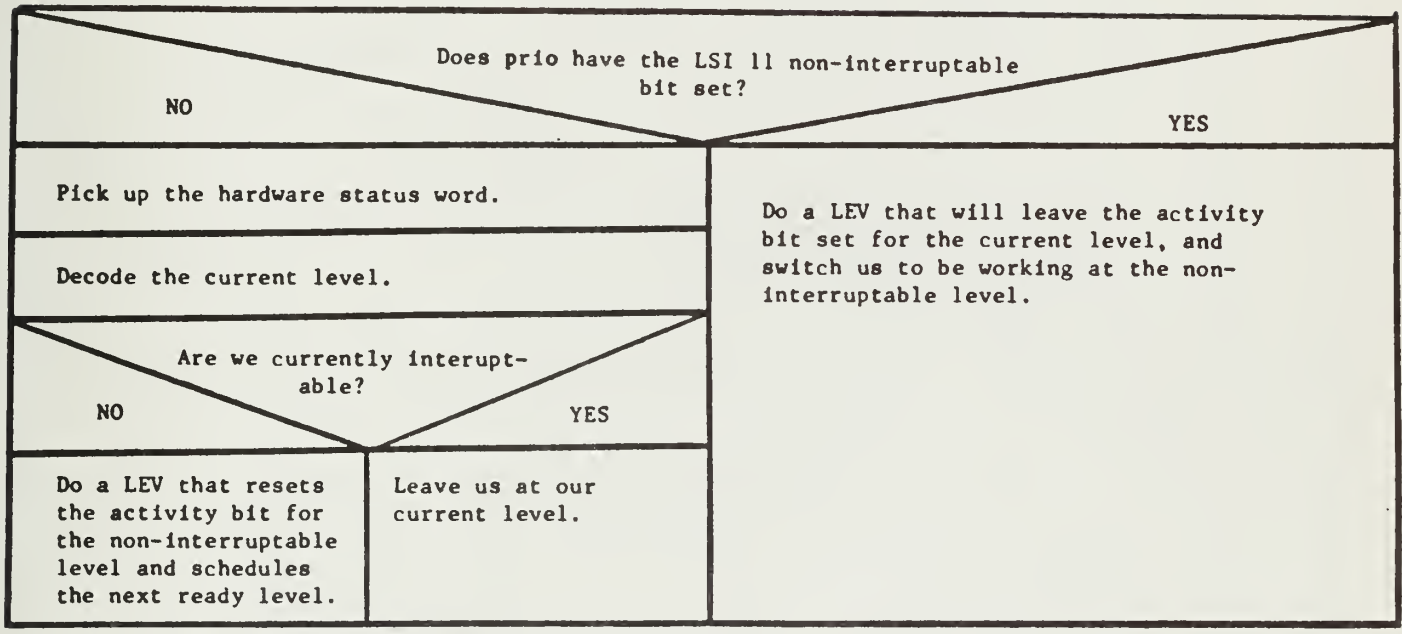


NOTE: This version of mfps is specific to the L6 IT.

mk_page(p_ptr, left, bottom, width, height)

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

mtps(prio)



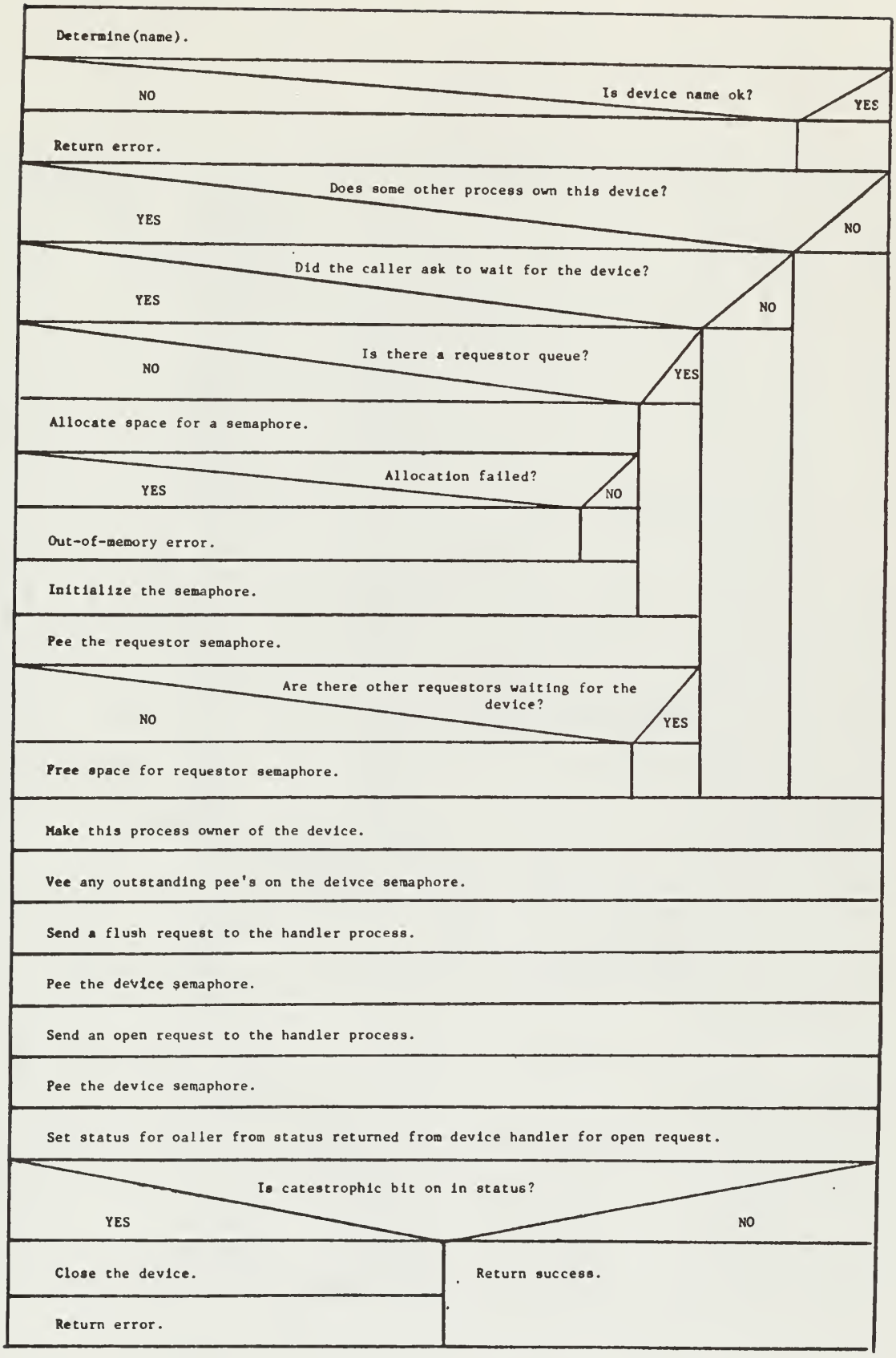
NOTE: This version of mtps is specific to the L6 IT.

mtps(prio)

Copy prio into the PSW.

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

open(name, flag &status)



parm_xpand(how, what, count)

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

Switch on type of expansion.

type 's'	type 'c'	type 'd'	type 'l' type 'u'	type 'o'	type 'x'	default
Pick up char ptr pointed to by what.	Copy char pointed to by what to workspace buffer.	Is parm less than 0? YES / NO		Put a 'o' in the buffer.	Put an 'x' in the buffer.	Copy the type character into the buffer.
Increment *what to point to next parameter in caller's list.		Put a '_' in buffer.	Do a str_num on parm, with a base of 10, into buffer.	Do a str_num on the parm, with a base of 8, into the buffer.	Do a str_num on the parm, with a base of 16, into the buffer.	
		Copy parm into temp.				
Return char ptr.		Do a str_num on temp, with a base of 10, into buffer.				

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.

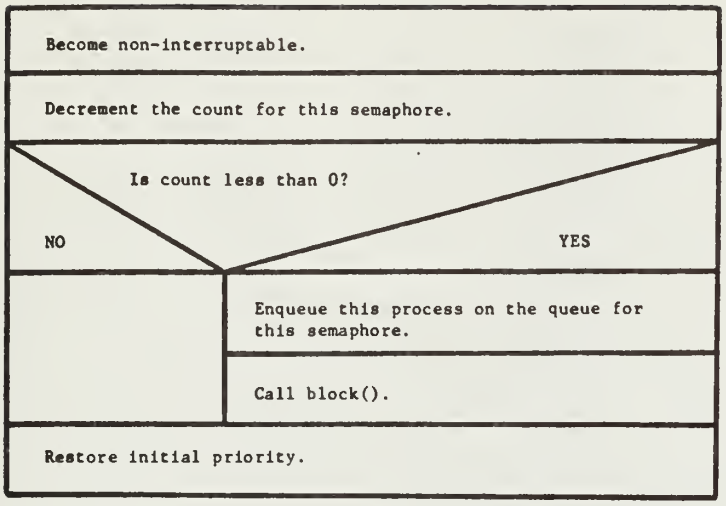
pause()

Enqueue this process on the READY_Q.
Call block().
Return.

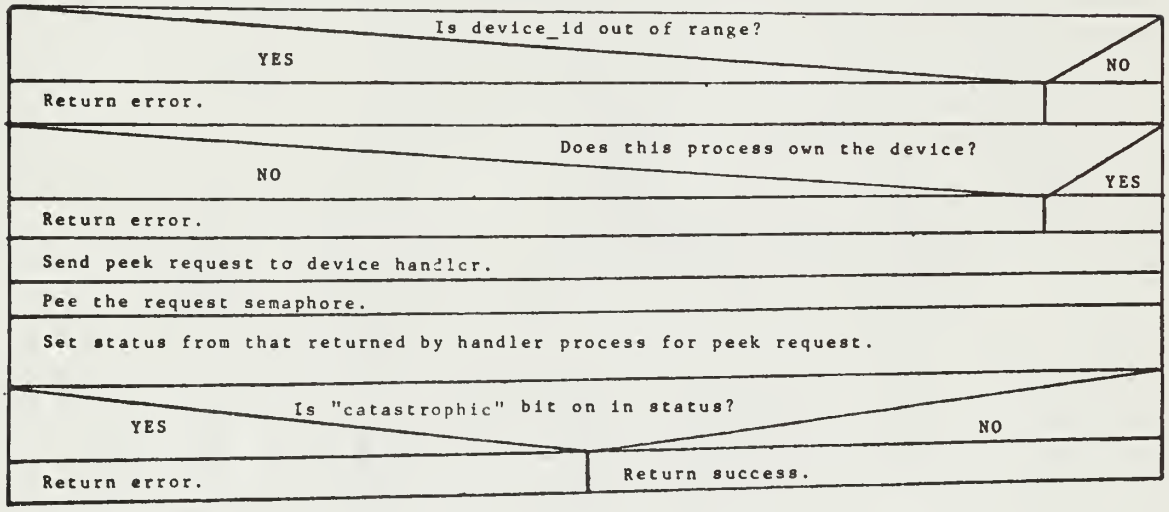
pathname(your, dir)

Remember where your string begins.	
Compare characters in your and dir strings up to the end of dir and as long as the two strings match.	
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	YES
	Return pointer to beginning of your string.
Return pointer to your stirng where comparison ended.	

pee(sem)



peek(device_id, &status)



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

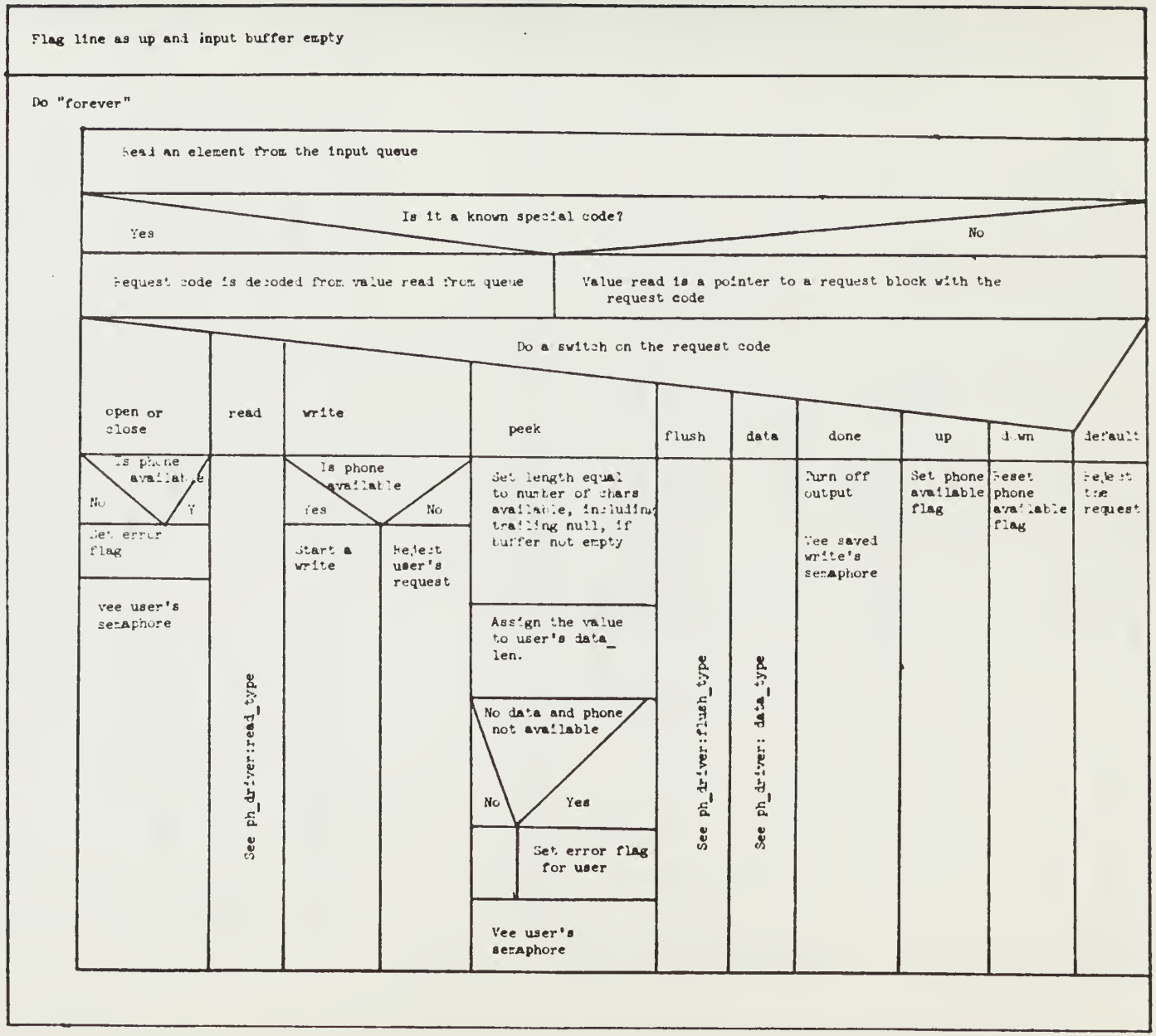
Command = what read from queue.

Switch (command)

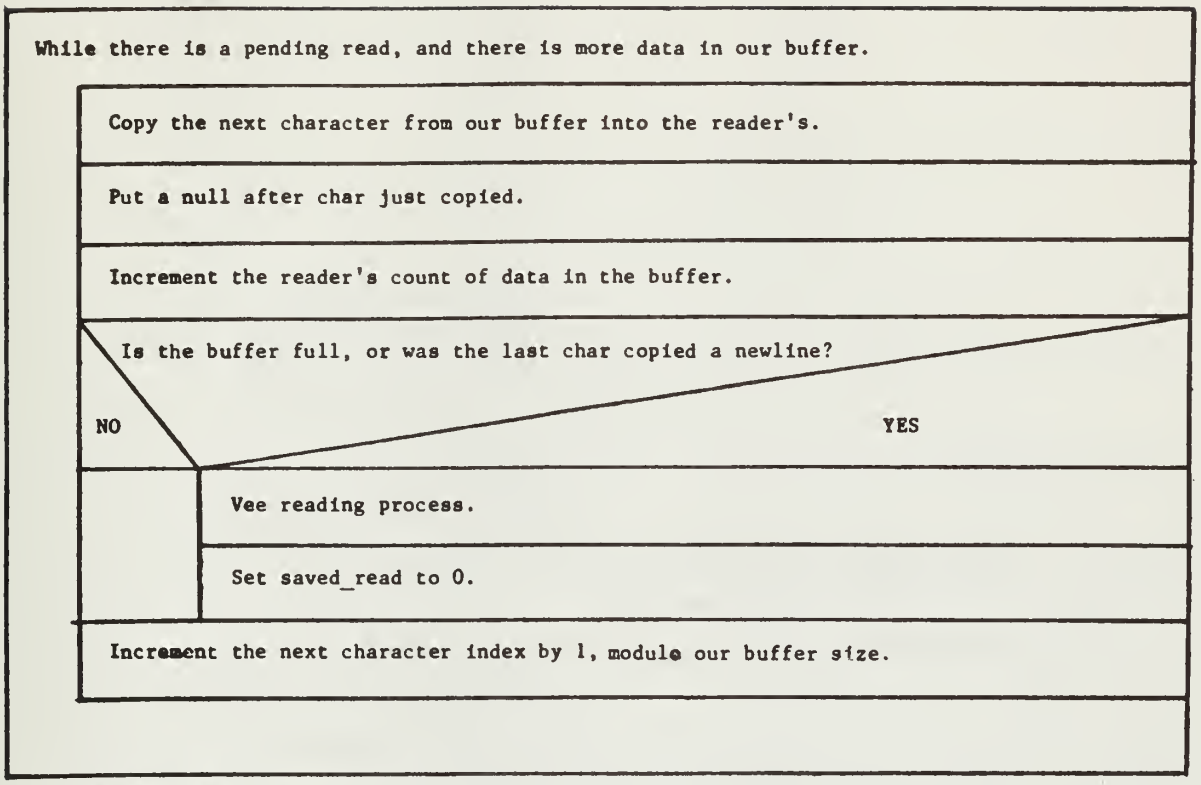
open	close	read	write	peek	flush	data	overflow	write failed	done	status change	default
(See ph_driver: open_type)	(See ph_driver: close_type)	(See ph_driver: read_type)	(See ph_driver: write_type)	Calculate amount of data in our buffer, including trailing null if buffer is not empty. Assign the value to caller's data_len. If no data and phone not available? NO: Vee requesting process. YES: Set catastrophic bit in caller's status.	(See ph_driver: flush_type)	(See ph_driver: data)		Set catastrophic bit in writer's status. Set writer's data_len to amount of data actually transmitted. Vee writing process. Set active_write to 0.		(See ph_driver: status_change)	Set reject_req bit in caller's status. Vee requesting process.

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

ph_driver ()

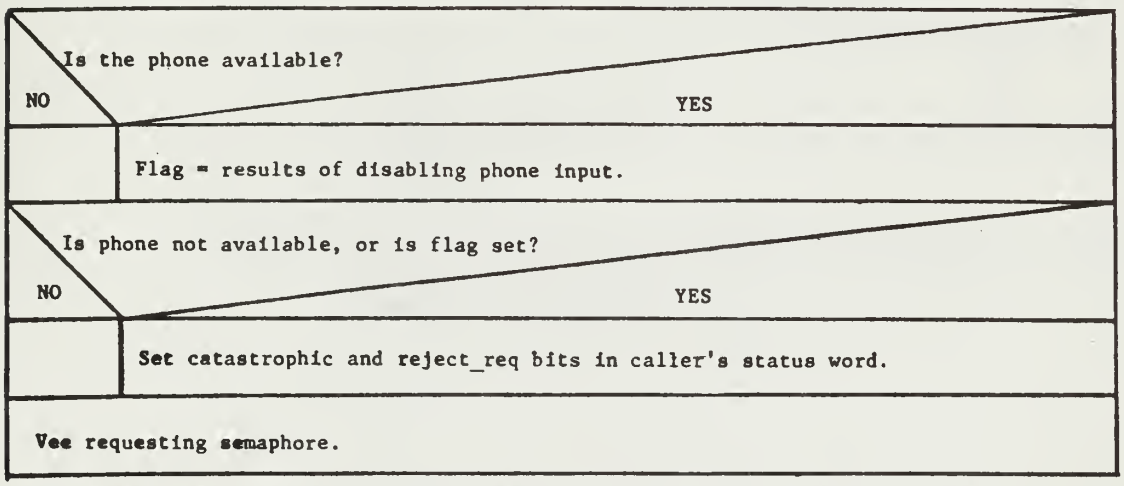

Flag line as up and input buffer empty
Do "forever"
Read an element from the input queue
Is it a known special code?
Yes
No
Request code is decoded from value read from queue
Value read is a pointer to a request block with the request code
Do a switch on the request code
open or close
read
write
peek
flush
data
done
up
down
default
Is phone available
No
Yes
Set error flag
Vee user's semaphore
See ph_driver:read_type
Is phone available
Yes
No
Start a write
Reject user's request
Set length equal to number of chars available, including trailing null, if buffer not empty
Assign the value to user's data_ len.
No data and phone not available
No
Yes
Set error flag for user
Vee user's semaphore
See ph_driver:flush_type
See ph_driver: data_type
Turn off output
Vee saved write's semaphore
Set phone available flag
Reset phone available flag
Reject the request

ph_driver: data*



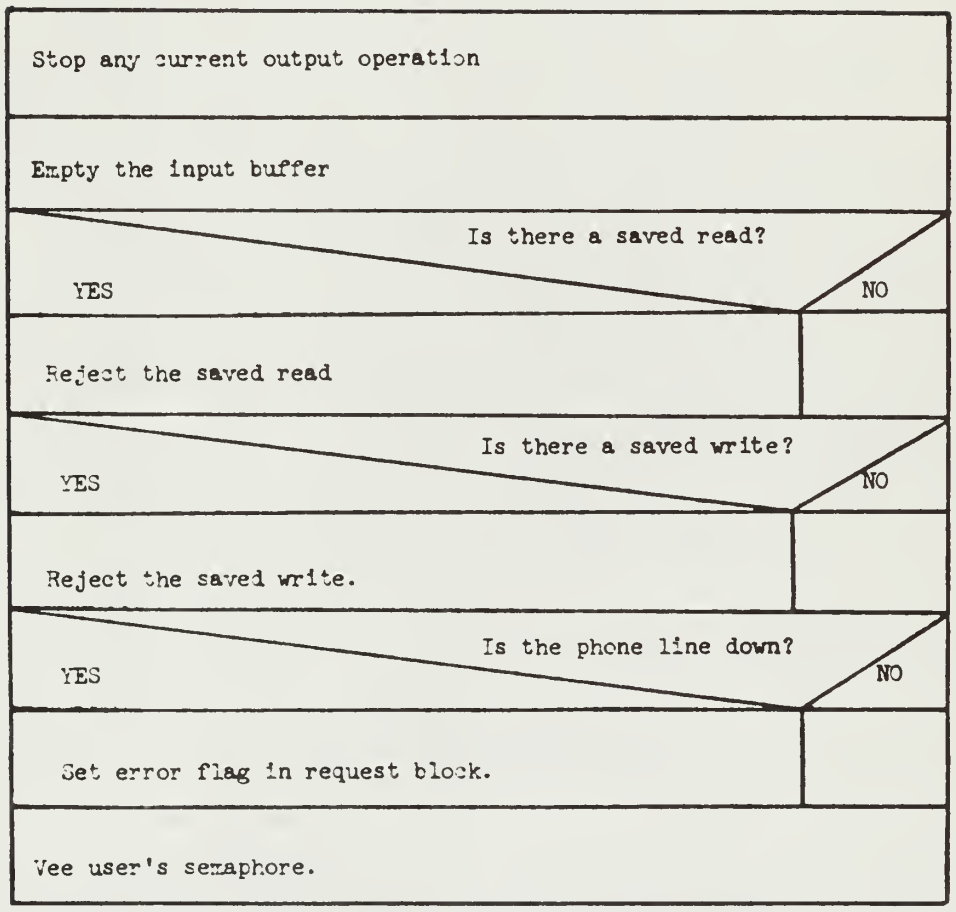
*This is not a procedure. It is one case in ph_driver.

ph_driver: close_type*



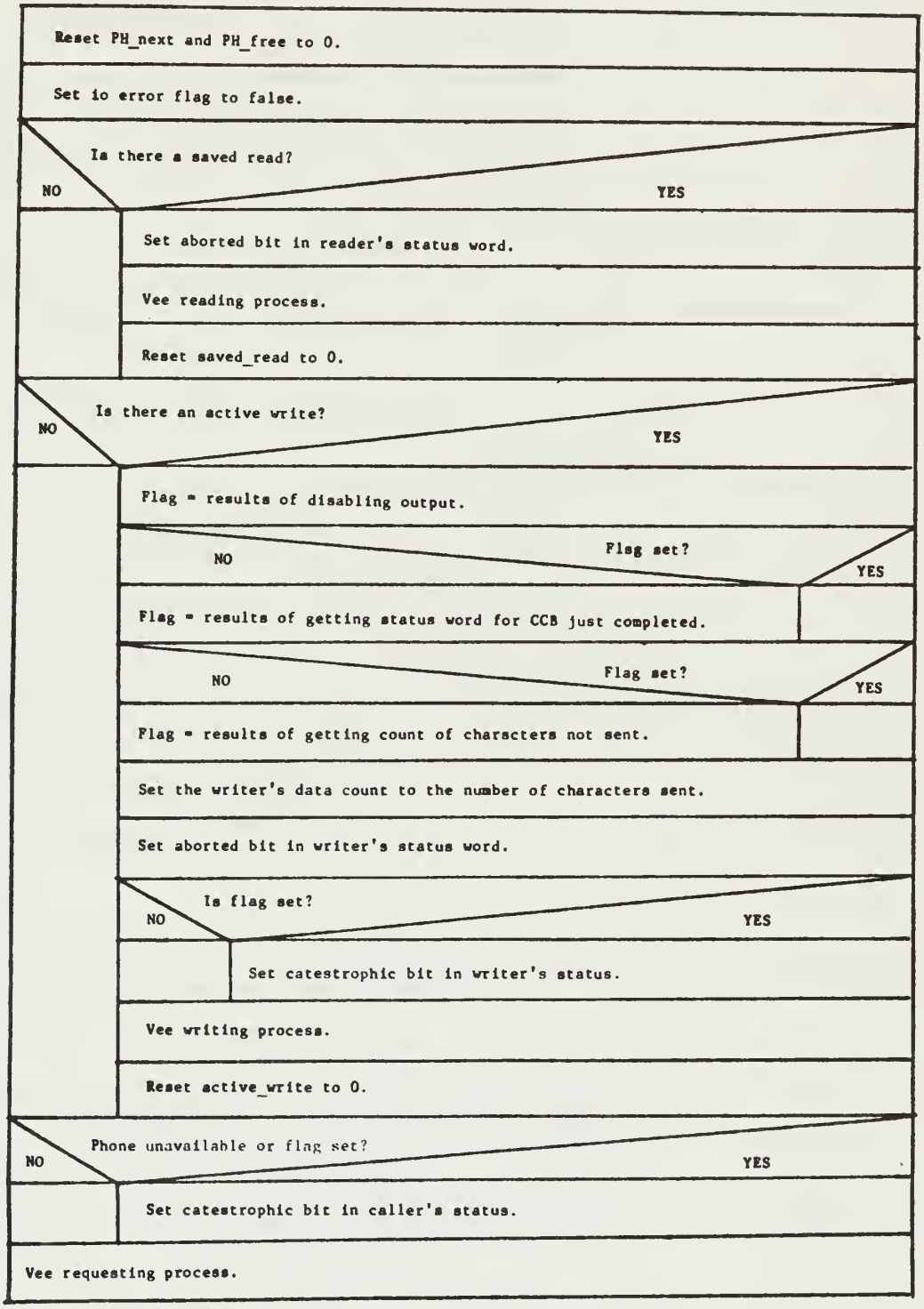
*This is not a procedure. It is one case in ph_driver and is specific to the Level 6 IT.

ph_driver: flush_type*



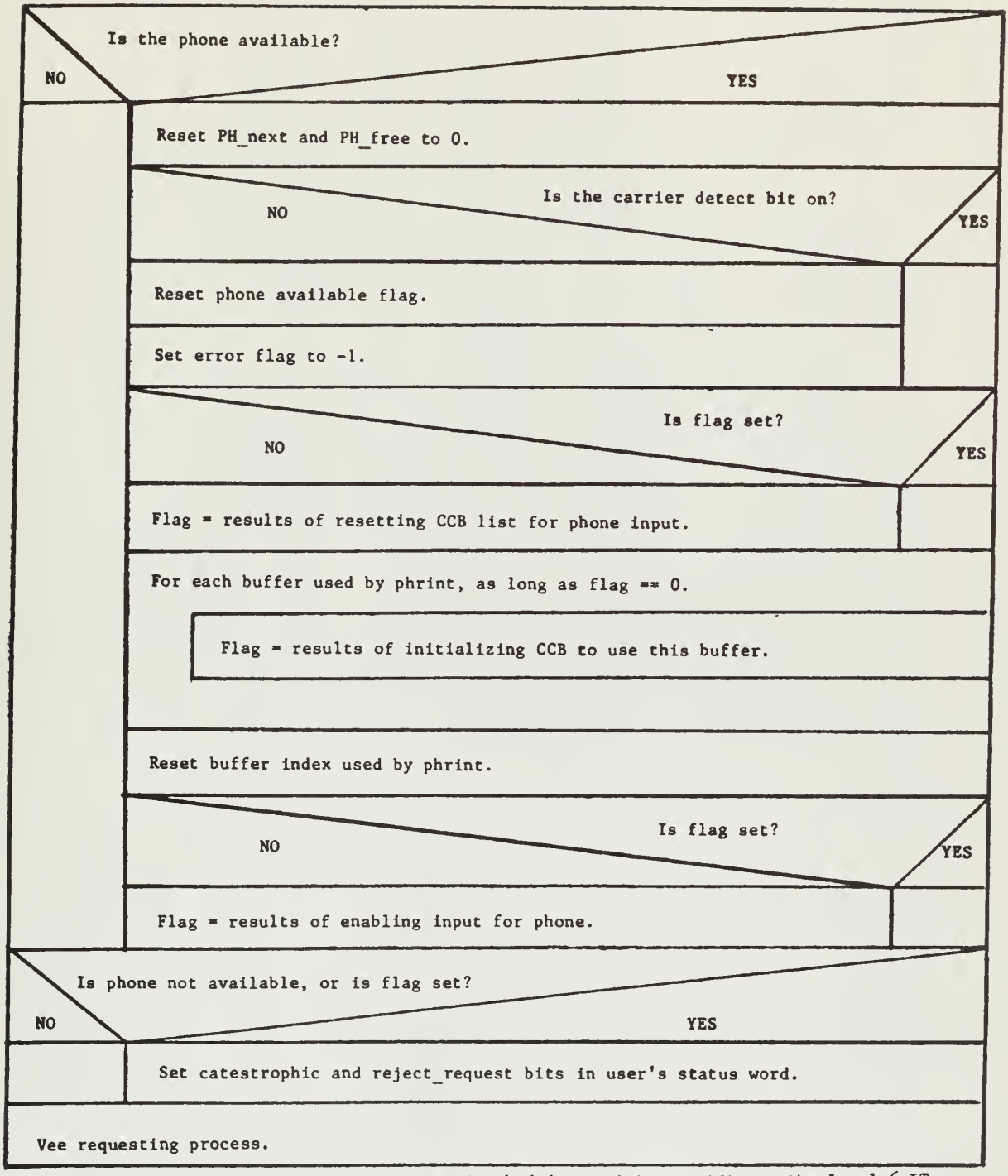
*NOTE: This is not a procedure. It is one case in ph_driver, and is specific to the LSI-11 IT.

ph_driver: flush_type*



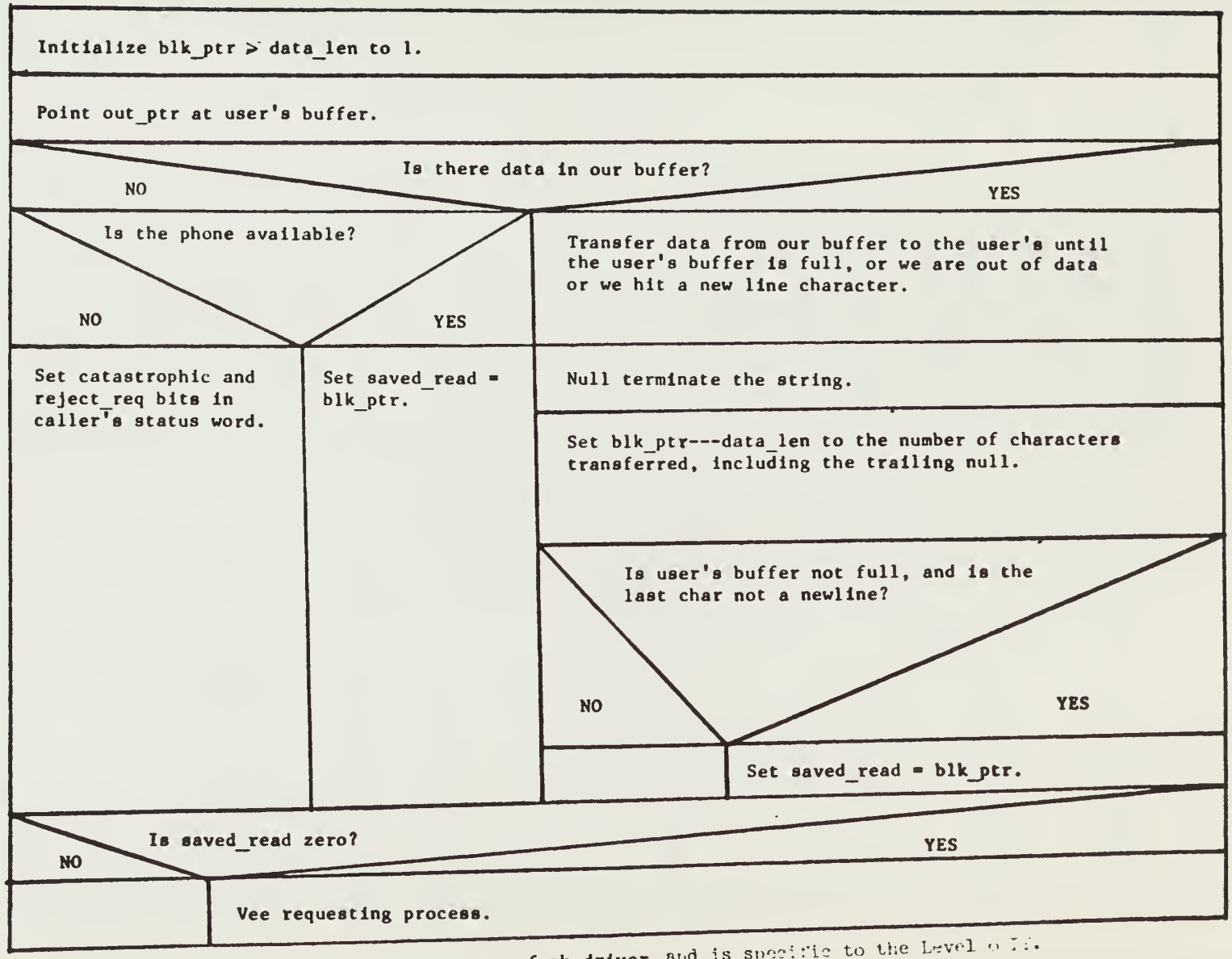
*This is not a procedure. It is one case in ph_driver and is specific to the Level 6 IT.

ph_driver: open_type*



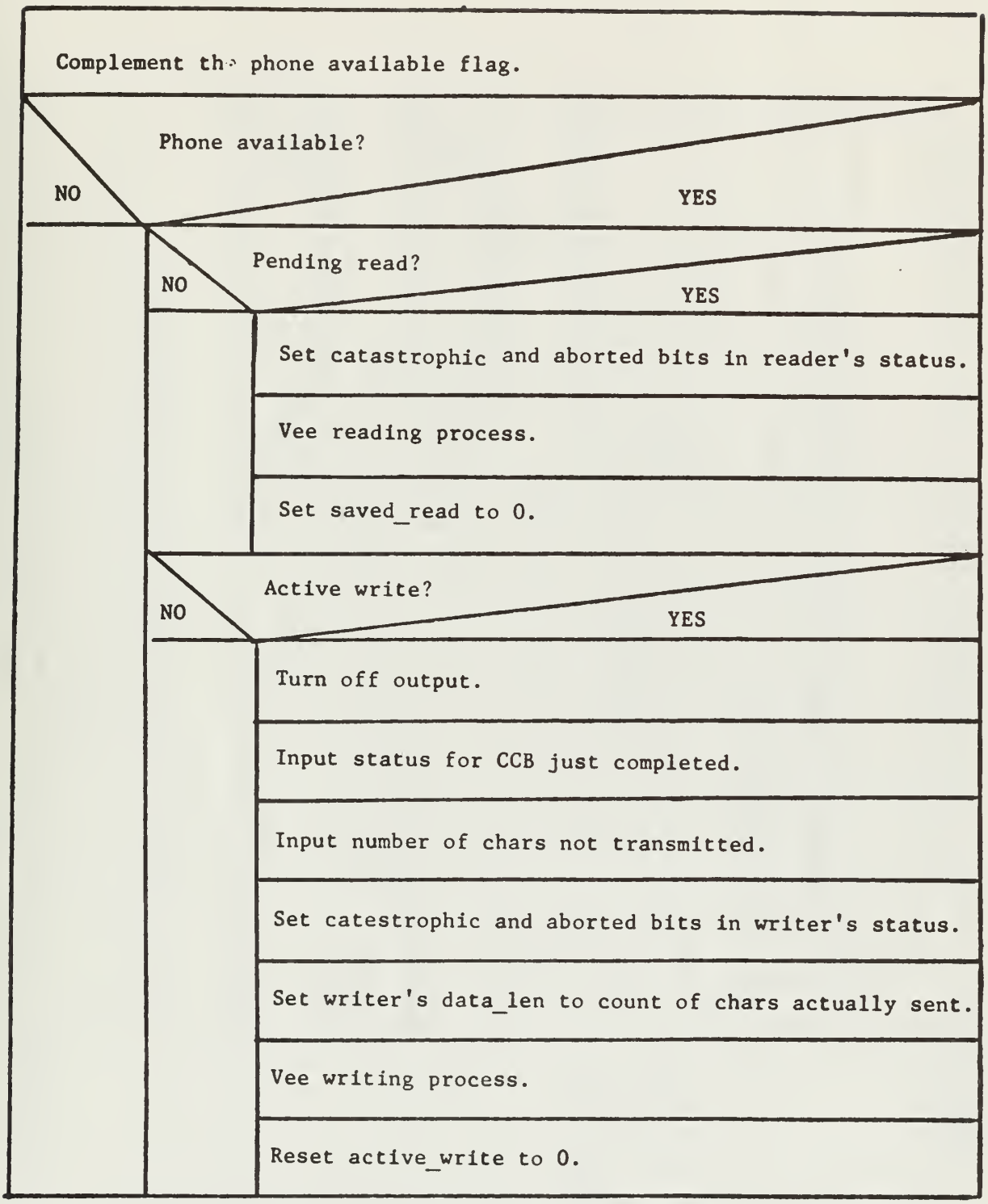
*This is not a procedure. It is one case in ph_driver and is specific to the Level 6 IT.

ph_driver: read_type*



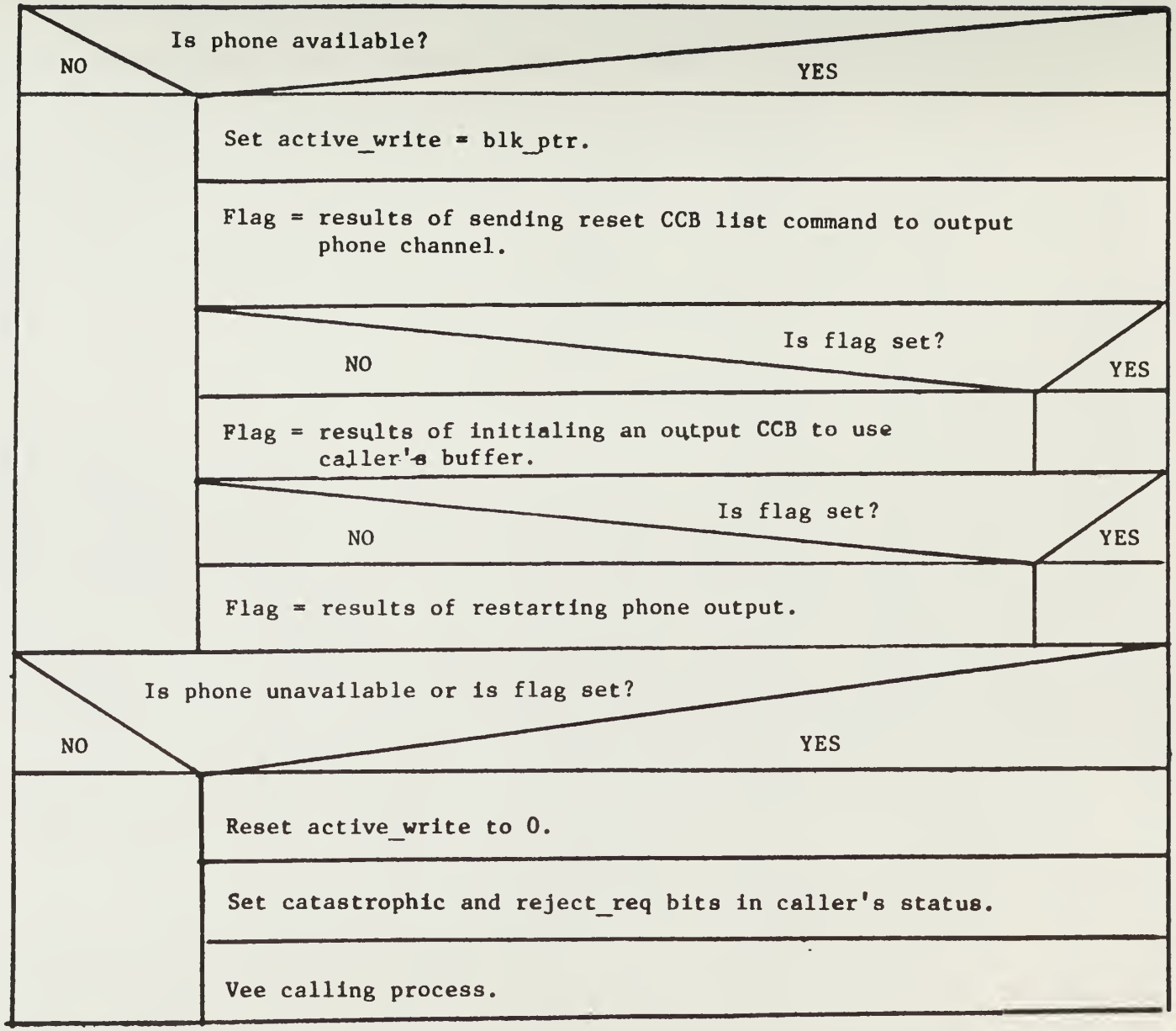
*This is not a procedure. It is one case of ph_driver, and is specific to the Level 0 [illegible]

ph_driver: status_change*



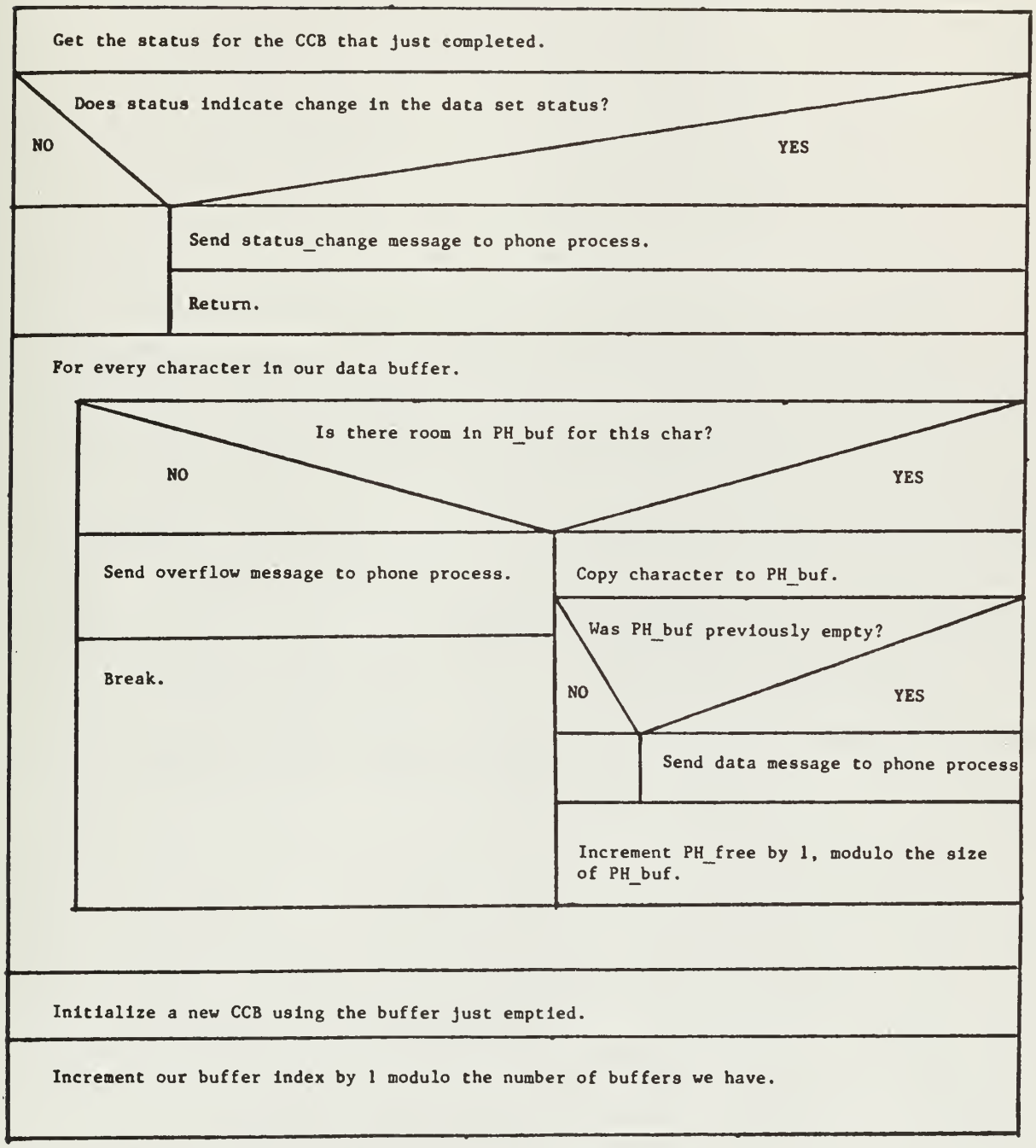
*This is not a procedure. It is one case in ph_driver and is specific to the Level 6 IT.

ph_driver: write_type*



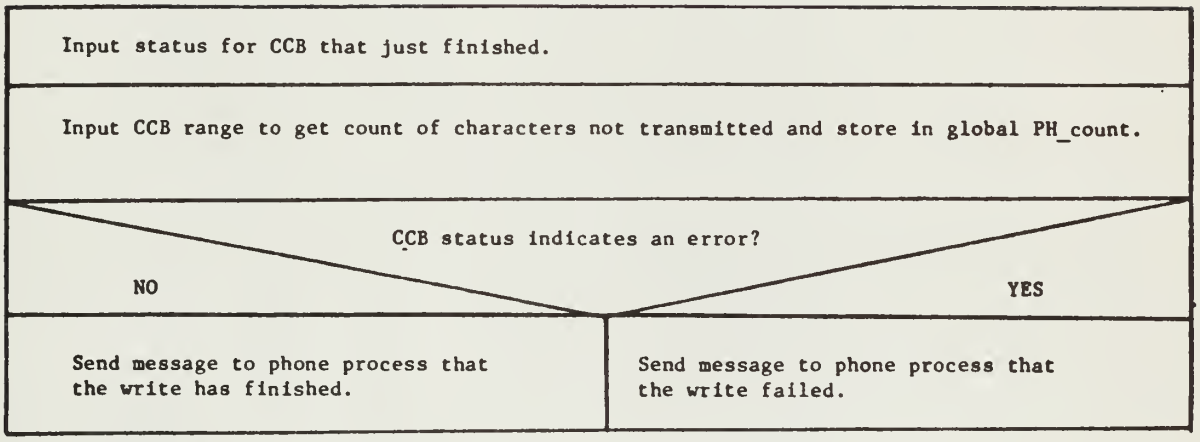
*This is not a procedure. It is one case in ph_driver and is specific to the Level 6 IT.

phrint()

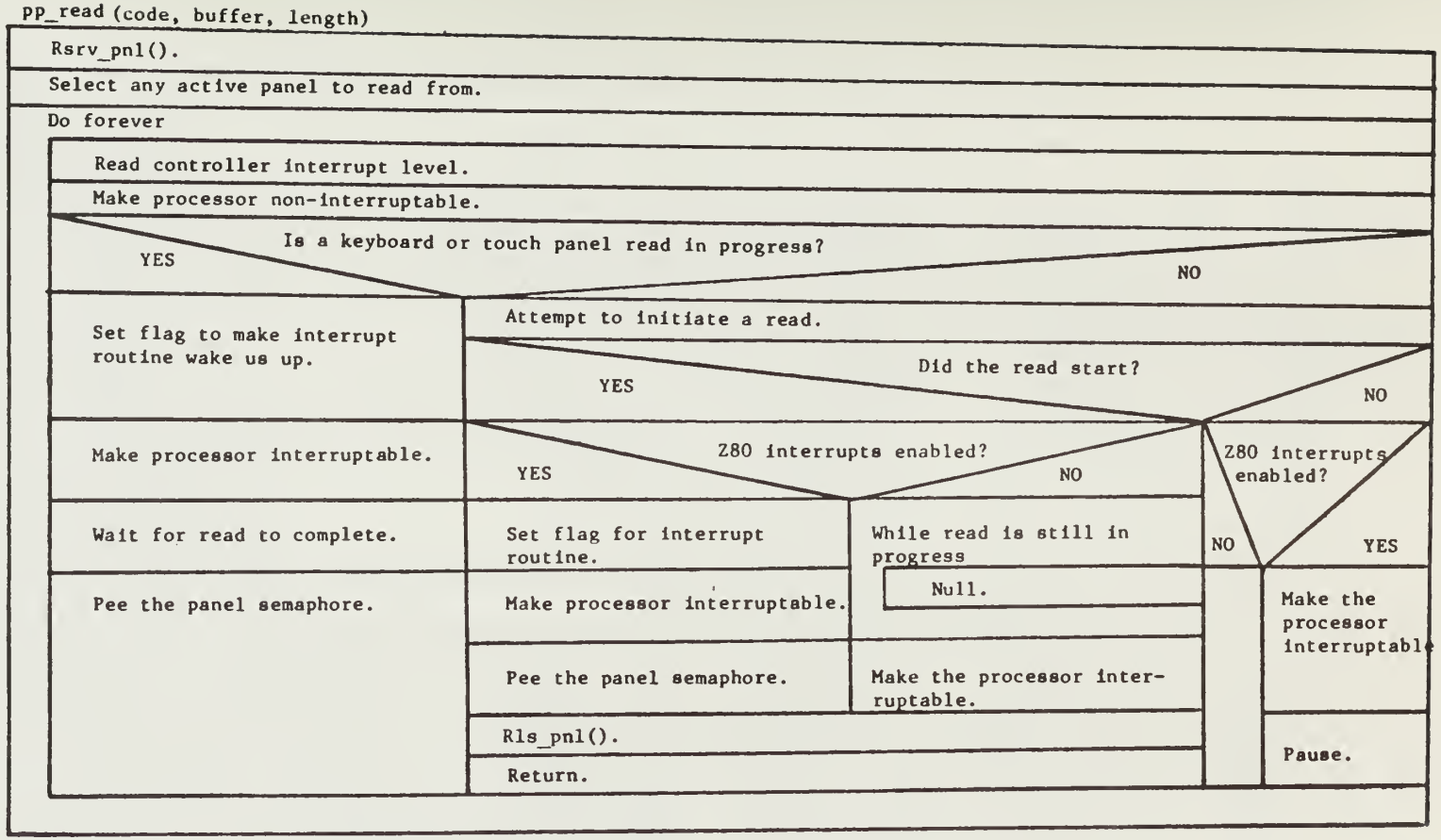


NOTE: This version of phrint is specific to the Level 6 IT.

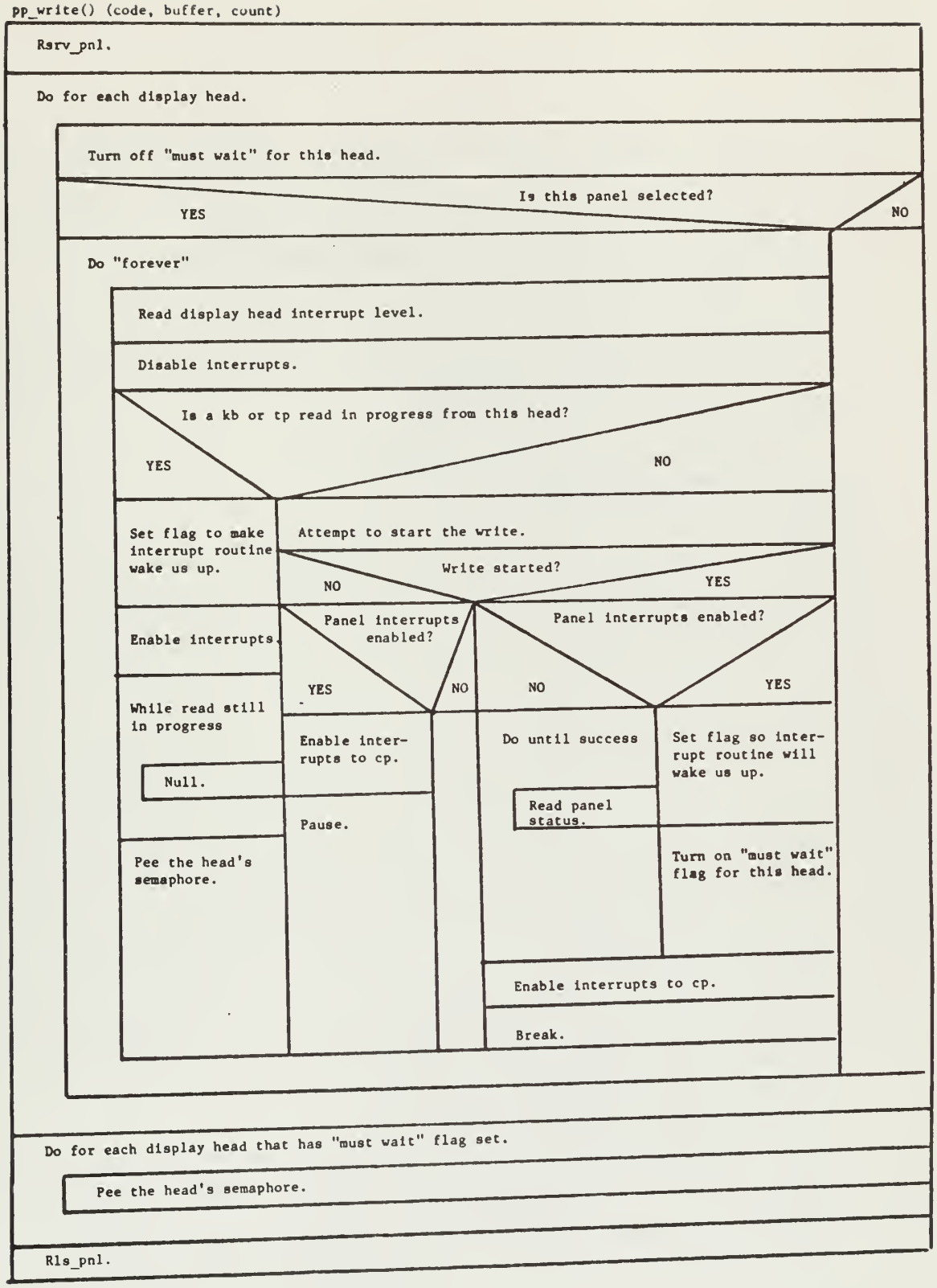
phxint()



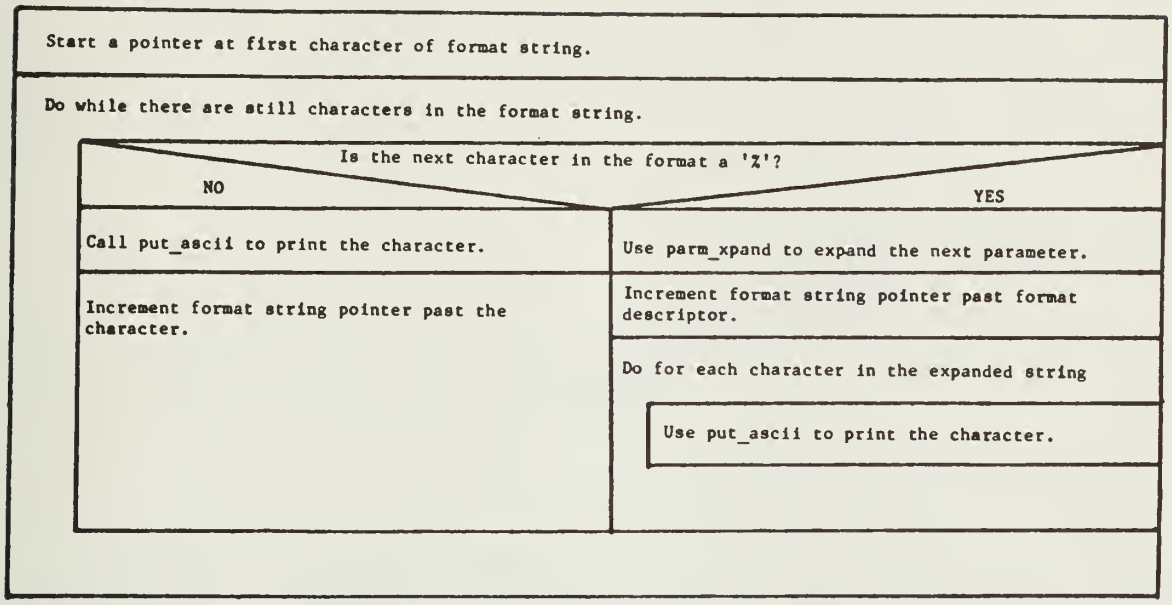
NOTE: This version of phxint is specific to the Level 6 IT.


pp_read (code, buffer, length)
Rsrv_pnl().
Select any active panel to read from.
Do forever
Read controller interrupt level.
Make processor non-interruptable.
Is a keyboard or touch panel read in progress?
YES
NO
Set flag to make interrupt routine wake us up.
Attempt to initiate a read.
Did the read start?
YES
NO
Make processor interruptable.
Z80 interrupts enabled?
YES
NO
Z80 interrupts enabled?
NO
YES
Wait for read to complete.
Set flag for interrupt routine.
While read is still in progress
Null.
Pee the panel semaphore.
Make processor interruptable.
Make the processor interruptable
Pee the panel semaphore.
Make the processor interruptable.
Rls_pnl().
Return.
Pause.

pp_write() (code, buffer, count)

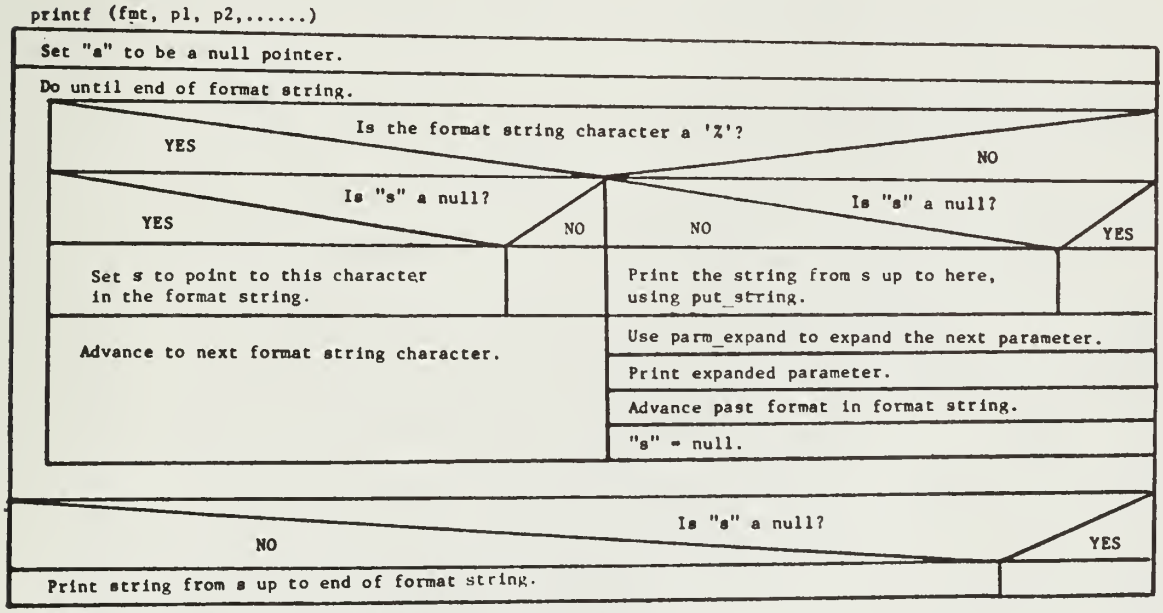


printf(format, p1, p2,.....)



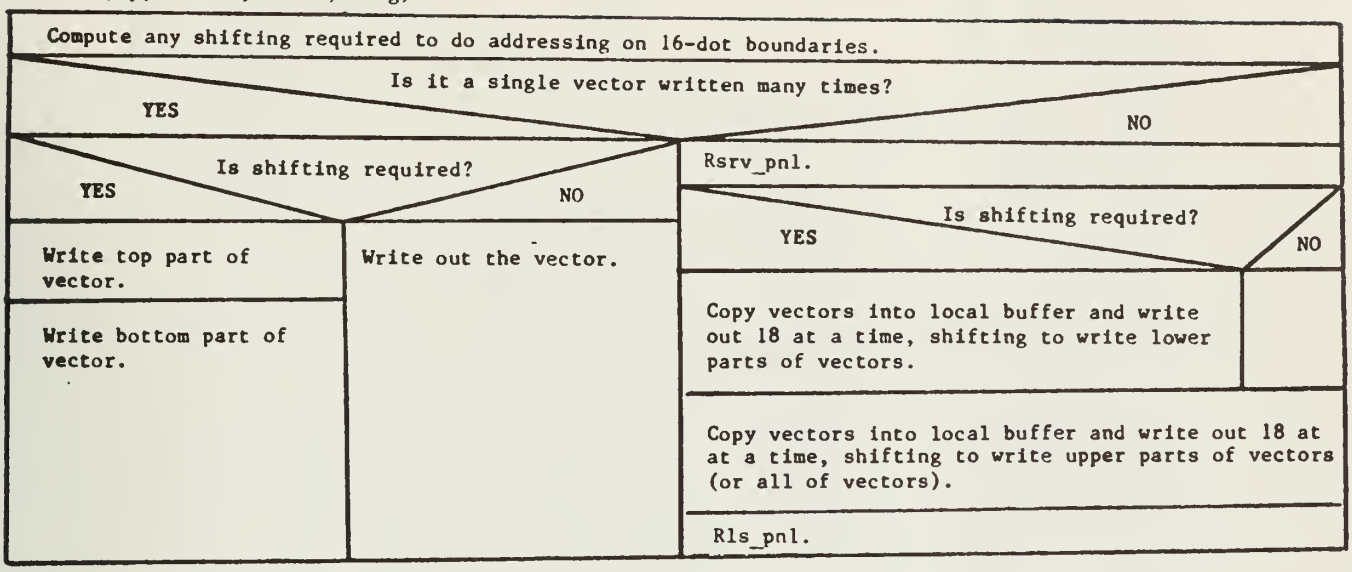
NOTE: This version of printf is specific to the LSI 11 IT.

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



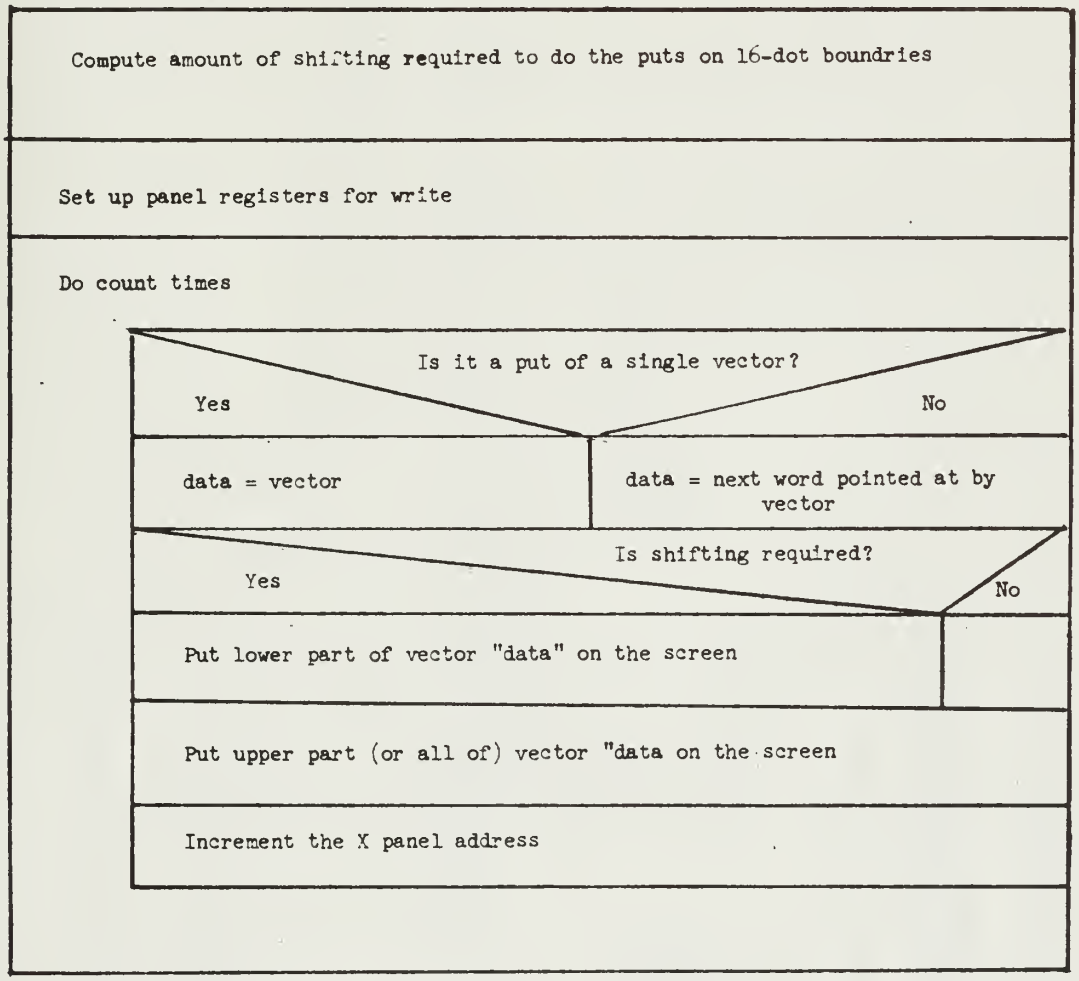
NOTE: This version of printf is specific to the Level 6 IT.

put (x, y, vector, count, flag)



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

put (X, Y, vector, count, flag)



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?

YES: Put cursor at start of next lower line.

NO:

Is the y-cursor outside of the page?

YES: Move y-cursor to top of page.

NO:

Do switch on effector table entry for the character.

ordinary	bell	back_sp	tab	new_line	line feed	car_return	vert_tab	form_feed	char_delete	line delete	default
Putchar to print the character.	Ring_bell.	Decrement x-cursor by one character.	Increment x-cursor to next multiple of 8.	Move x-cursor to start of line.	Null.	Move x-cursor to start of line.	Move y-cursor up one line.	See put_ascii: put_ascii_form.	Move x-cursor back one character.	Use erase to erase entire line.	Ring bell.
Increment x-cursor by one character.				Move y-cursor down one line.					Use "erase" to erase one character.	Move x-cursor to start of line.	

Is the x-cursor outside of the page?

YES: Put cursor at start of next lower line.

NO:

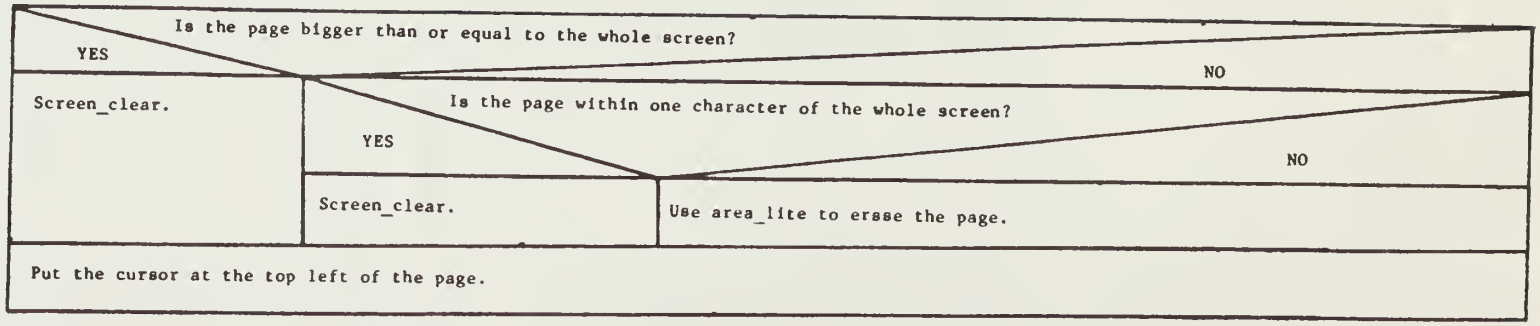
Is y-cursor outside of page?

YES: Move cursor to top of page.

NO:

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

put_ascii: put_ascii_form*



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

put_string (buffer, count)

Ld_page.
Ld_cs.
Write the string.

putchar(x, y, 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.

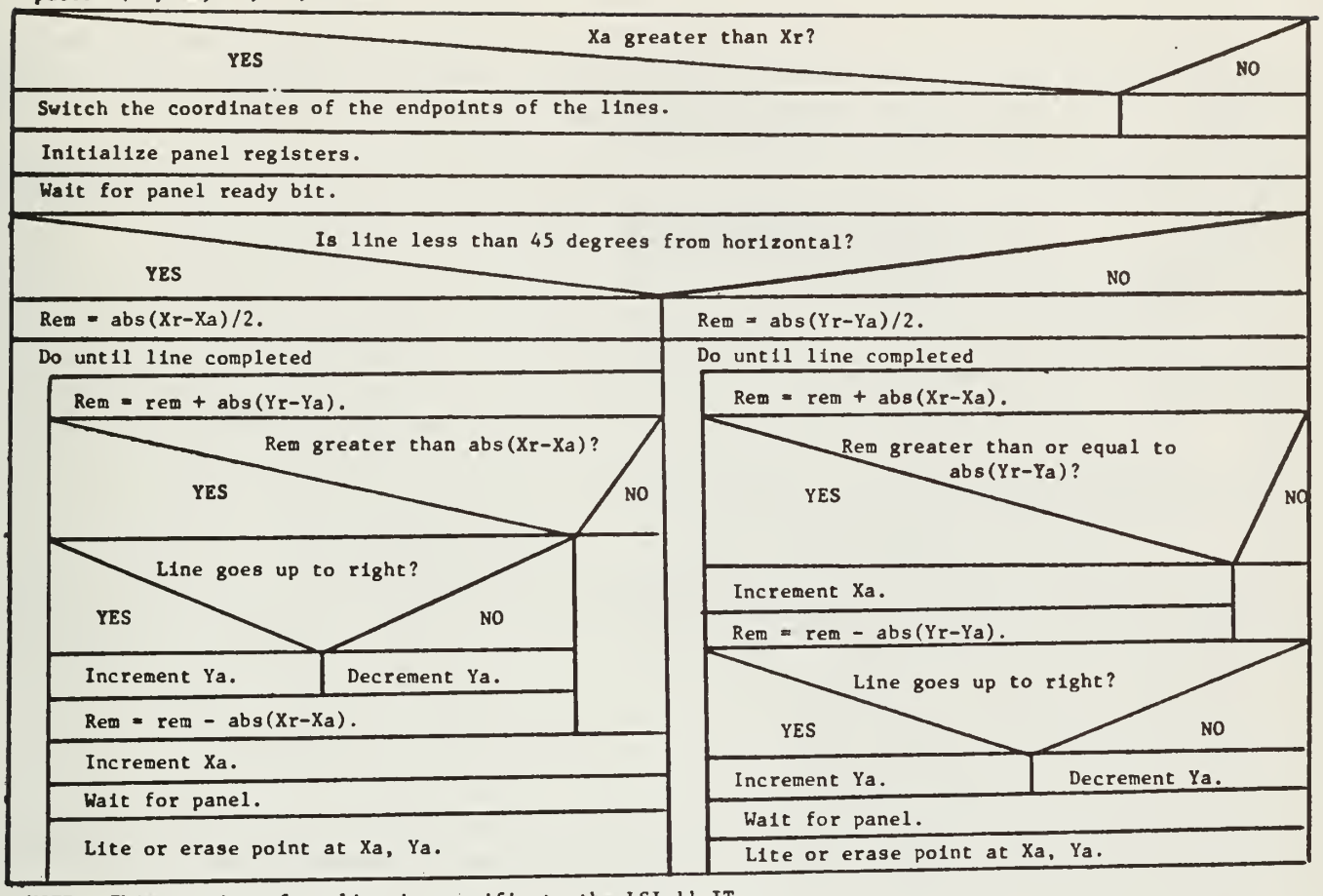
NOTE: This version of putdot is specific to the L6 IT.

putdot (X,Y,mode)

Set panel registers to write or erase the dot at X,Y.

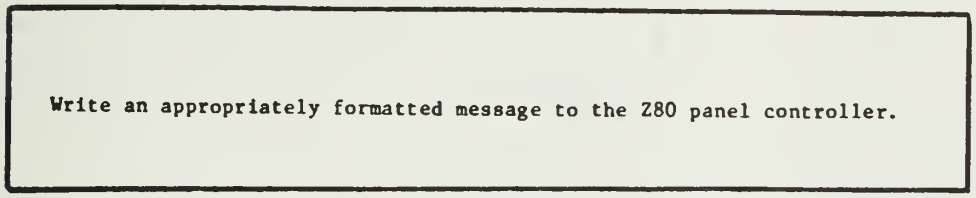
NOTE: This version of putdot is specific to the LSI-11 IT.

putline(Xa, Ya, Xr, Yr, mode)



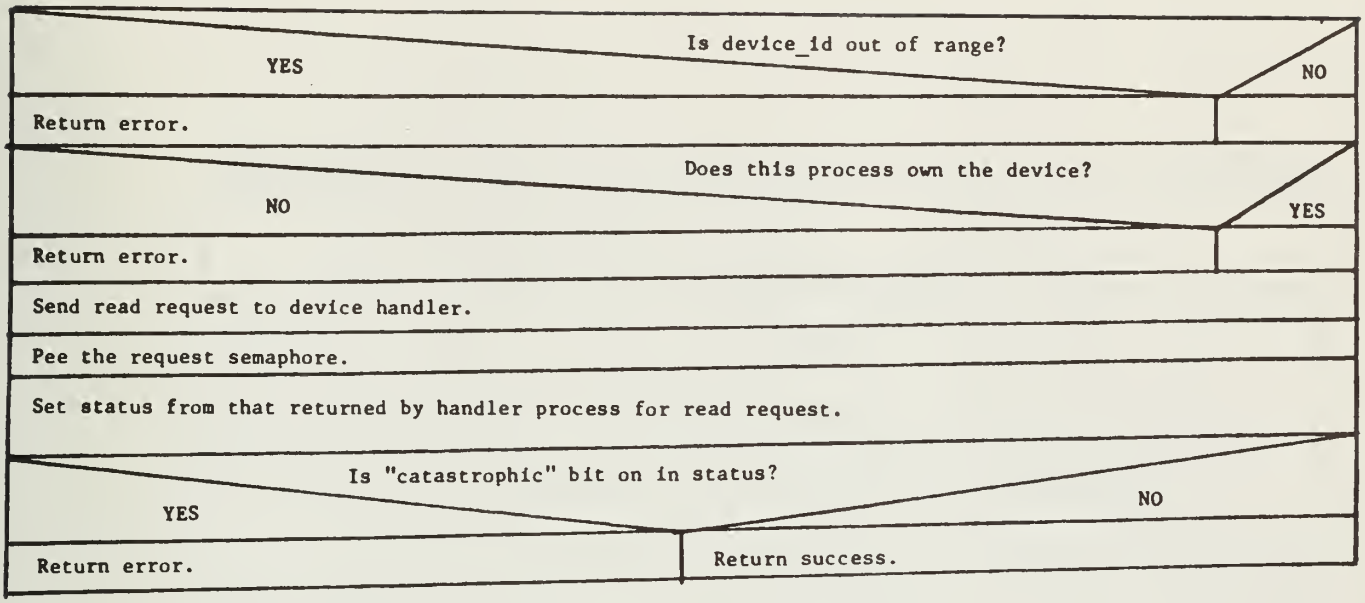
NOTE: This version of putline is specific to the LSI 11 IT.

putline

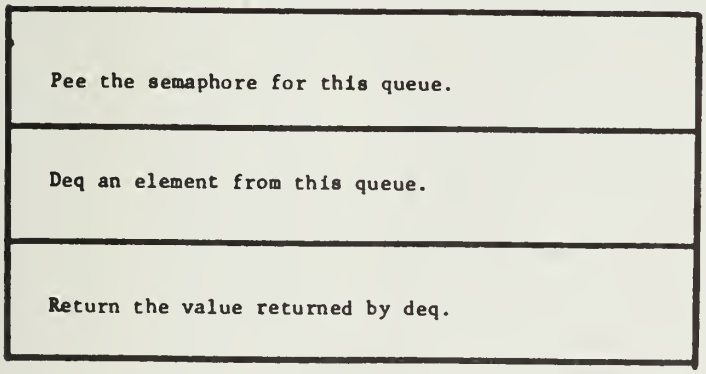


NOTE: This version of putline is specific to the L6 IT.

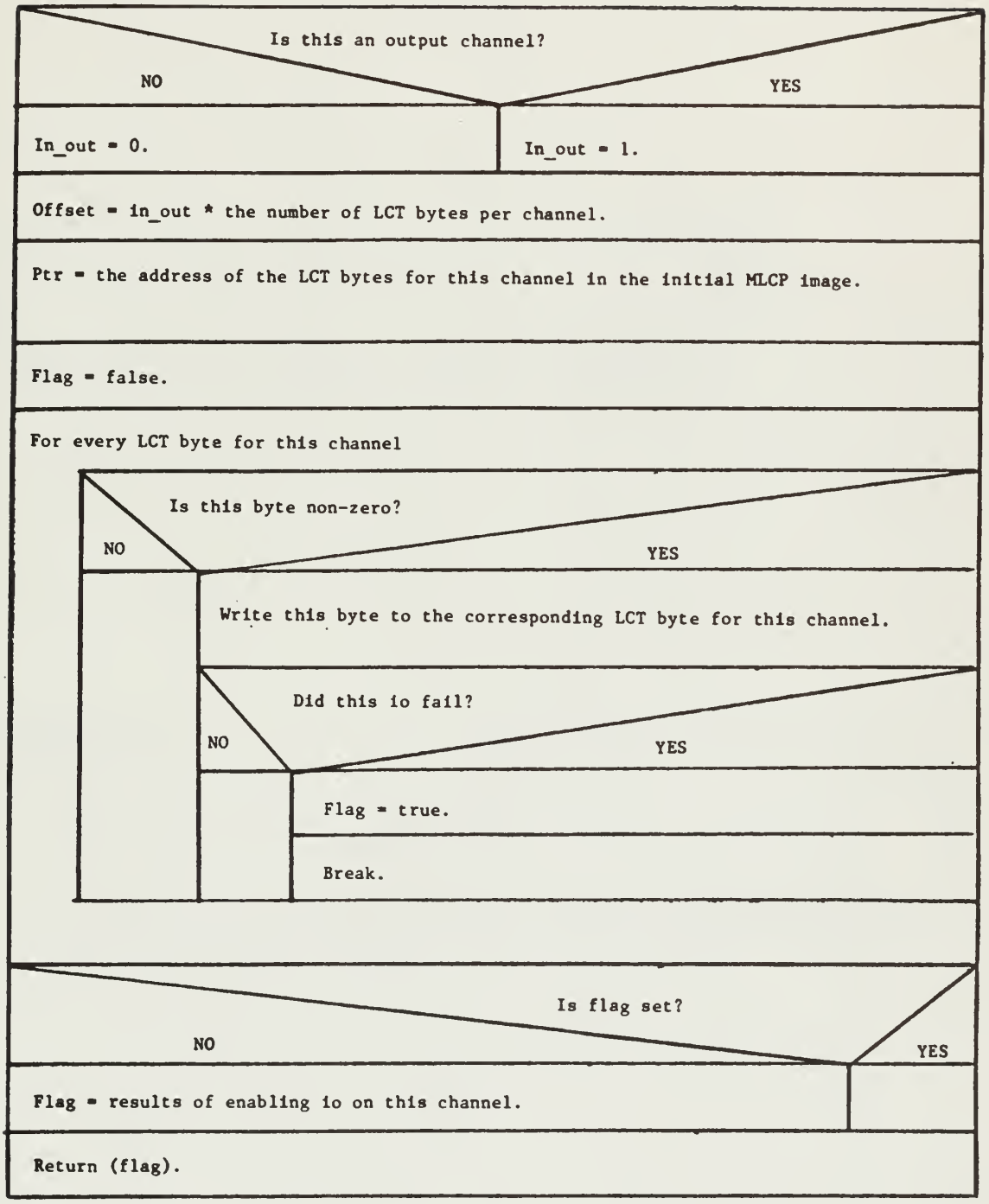
read(device_id, buf_ptr, length, &status)



read_q(q_ptr)



restart_io(chan, it_id)



ring_bell()

Set the panel registers to ring the bell.

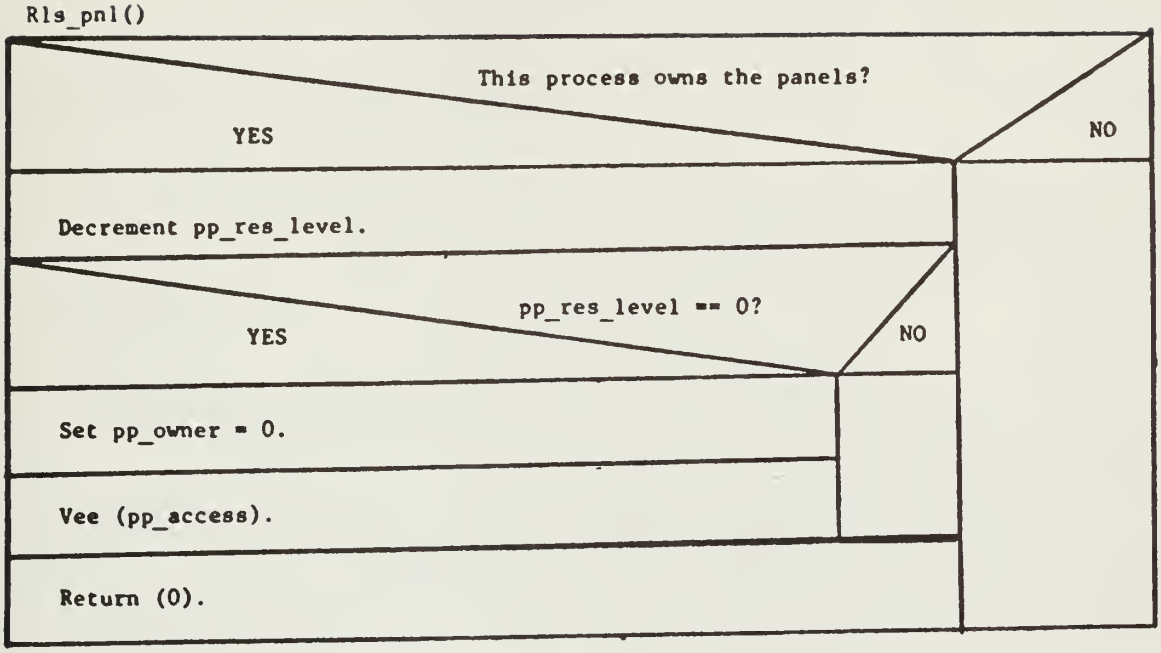
NOTE: This version of ring_bell is specific to the LSI11 IT.

ring_bell()

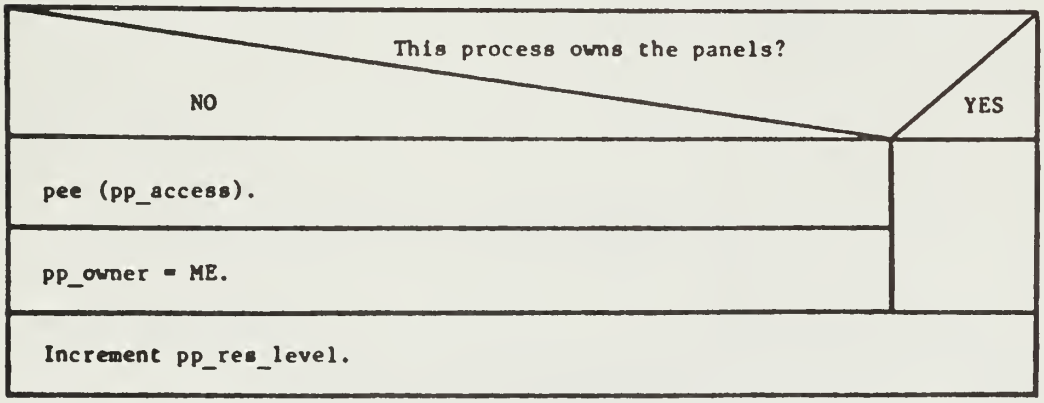
Write an appropriately formatted message to the Z80 panel controller.

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

Rls_pnl()



rsrv_pnl()



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 trsck and sector number for the sector.

Do up the maximum number of retries

Is this first time thru for this invocstion?
- 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?
- NO: Return success.
- YES:

Return error.

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

read(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 read command to disk.

Wait for the disk to be ready.

Was there an error on the disk?

NO

YES

Continue.

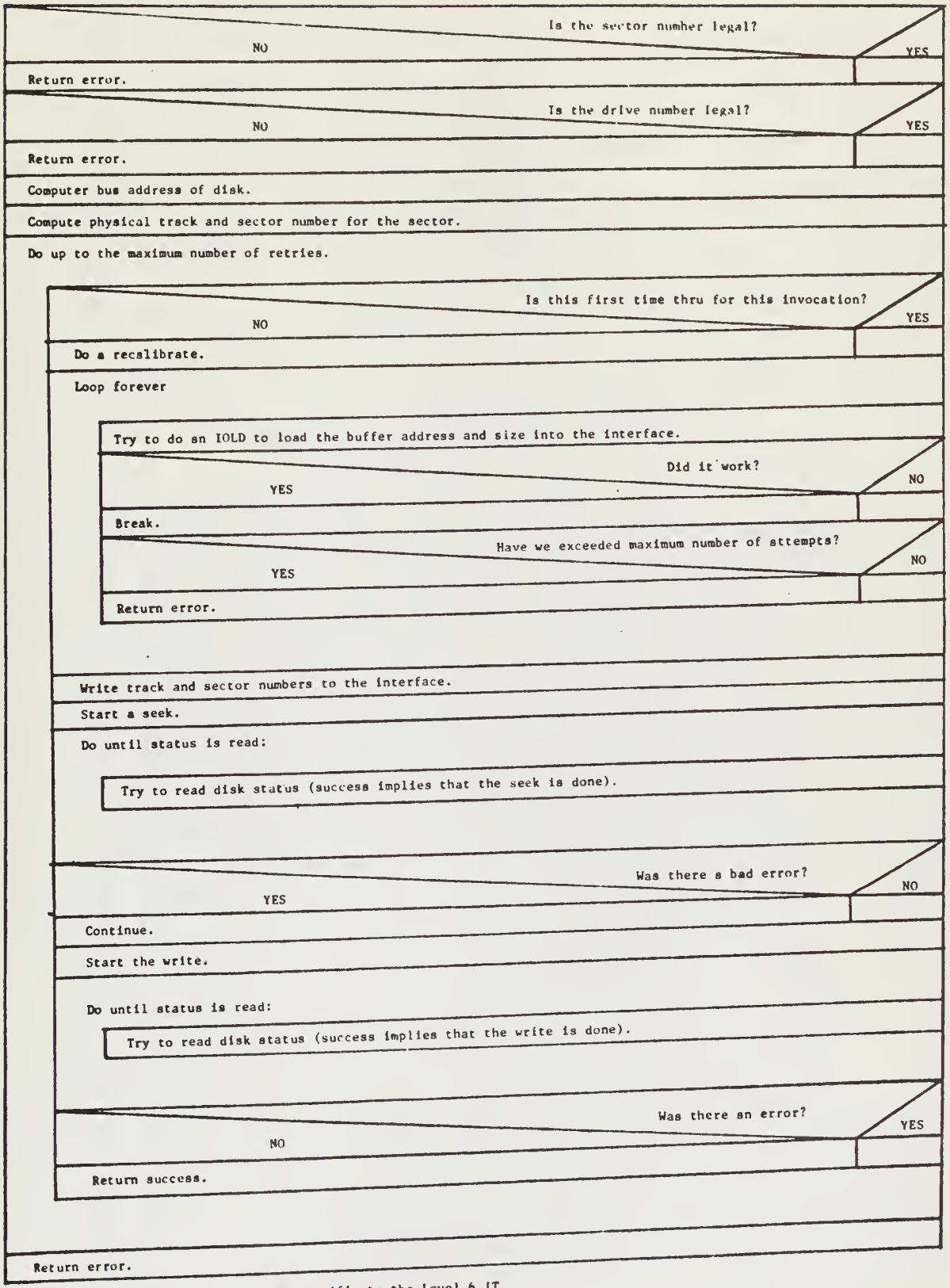
Swab (the data read).

Return (0).

Return (-1).

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

s_write(drive, sector, buffer)



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

a 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.

Swab(the data read).

Return (0).

Return (-1).

NOTE: This version of a write is specific to the LSI 11 11.

save_free(drv)

Is drive number (drv) out of range?	
NO	YES
	Return(-1).
Write the in_memory copy of freemap to disk.	
Return results of write.	

screen_clear

Write an appropriately formatted message to the Z80 panel controller.

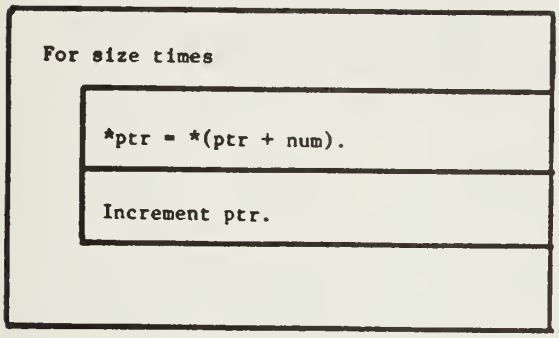
NOTE: This version of screen_clear is specific to the L6 IT.

screen_clear()

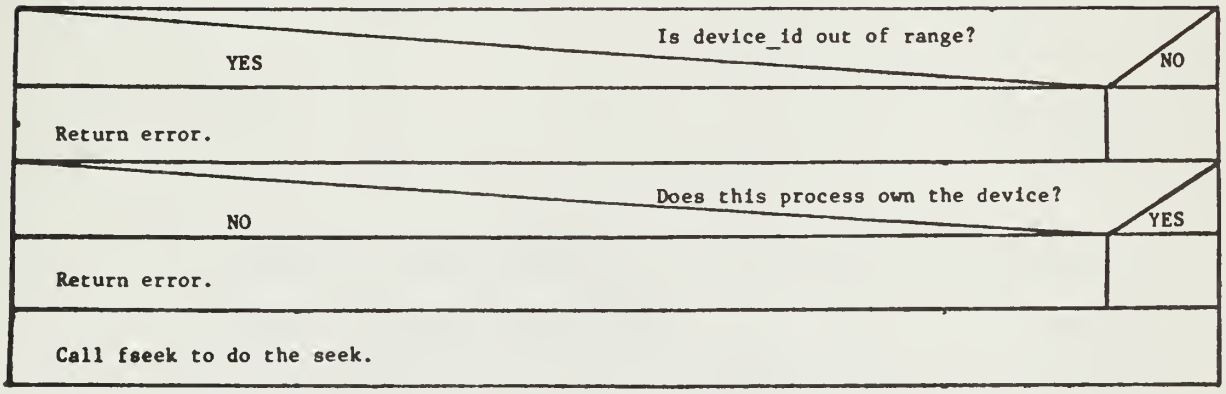
Set panel control register to clear the screen.

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

scrunch(ptr, size, num)



seek(device_id, length, type)



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.

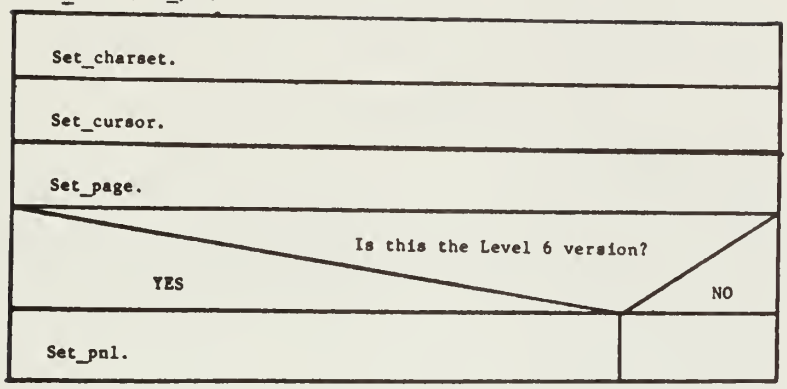
NOTE: This version of set_cursor is specific to the LSI-11 IT.

set_cursor (x_addr, y_addr)

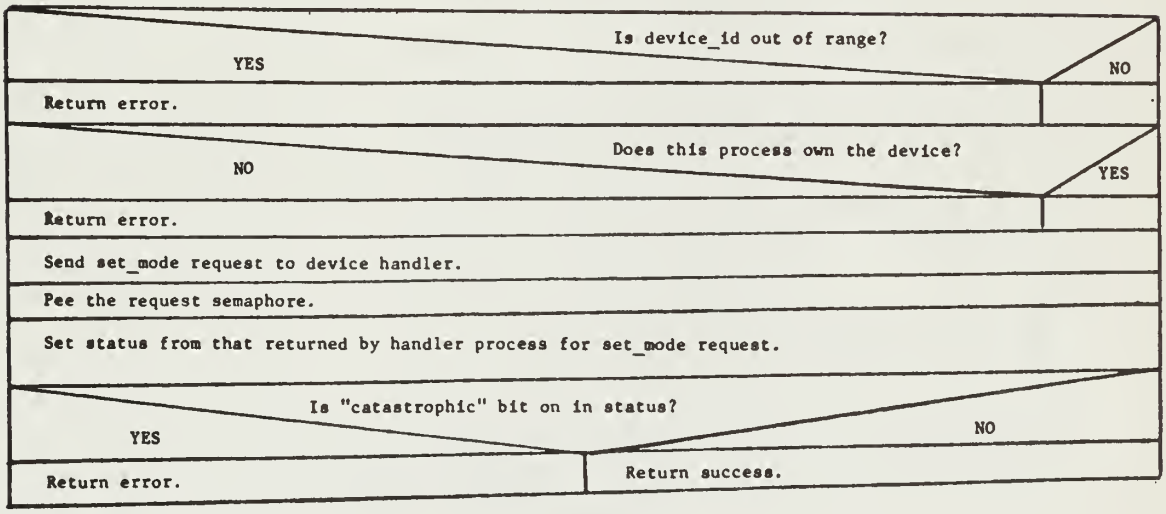
Convert parameters to dot offsets.
Write offsets to display head controller.

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

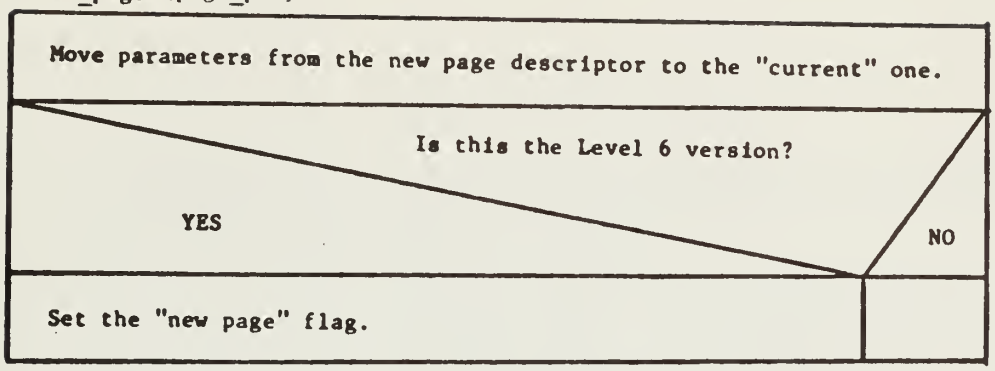
set_s::v (env_ptr)



set_mode(device_id, buf_ptr, length, &status)



set_page (page_ptr)



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.

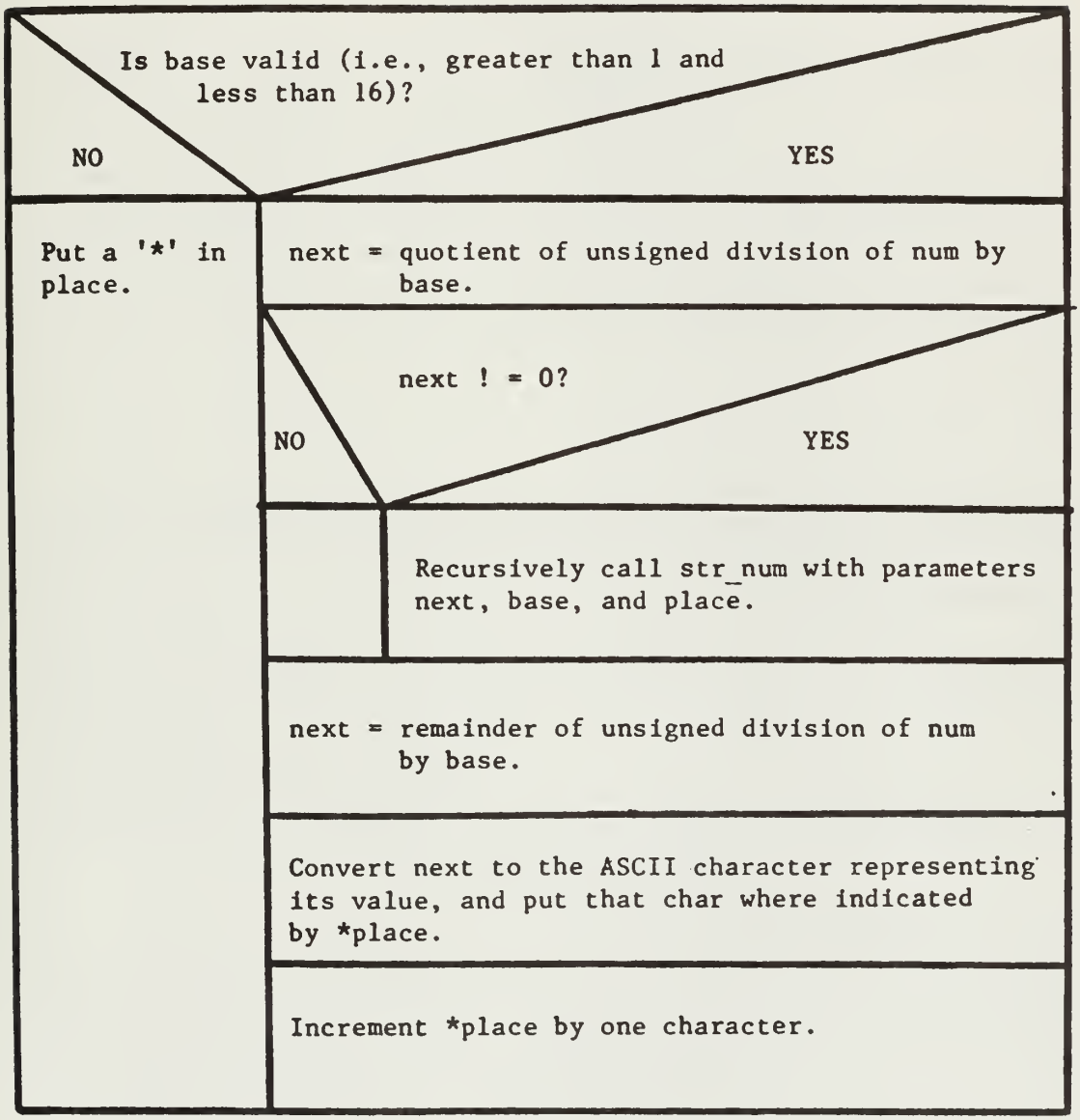
NOTE: This version of startup is specific to the LSI11 IT.

startup (high)

Set up CORETAB to indicate all of memory.
Allocate space for the free queue.
Initialize the list of free queue elements.
Disable interrupts from all of the remote display heads.
Set up READY_Q as empty.
Initialize the remote display heads.
Set default page, character set, and sursor position for plasma panel printing.
Create all processes specified in PROCTAB.
Load the MLCP.
Initialize the I/O system.
Enable interrupts from all remote display heads.
Call first_block.

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

str_num(num, base, place)



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	NO
Print a newline using printf.	
Print the token and separator strings using printf.	

tok_print(delims, separator, text, parms)

Use index to find first parameter replacement character in "text."

Was one found?

YES

Change the parameter replacement character to a null.

Set "move" flag to true.

NO

Set "move" flag to false.

Do while there are still tokens in text string:

Use get_token to get one token from text.

Was one found?

YES

Increment "text" pointer to past the token.

Does "text" now point to a null?

YES

Use tiod to print the token.

NO

Use tiod to print the token and a separator.

NO

Is "move" flag on, indicating a parm to expand?

YES

Move "text" pointer up to point at the parameter replacement character that was nulled out.

Change the null back to what it was.

Use parm_xpand to expand the parameter.

Increment "text" pointer past the parameter replacement stuff.

Do while there are still tokens:

Use get_token to get one token from the expanded parameter.

Did we get a token?

YES

Point past the token.

Are we at the end of the expanded parameter and is the character following the parameter a delimiter?

YES

Use tiod to print the token.

NO

Use tiod to print the token followed by a separator.

NO

Use index to find the next parameter replacement character.

Was one found?

YES

Change the character to null.

Set the "move" flag on.

NO

Set the "move" flag off.

Set flag so outer loop continues.

tp_driver()

Do forever

Read a word from input queue.

Is the word odd?

YES

Request code = "data."

The touch coordinates are in bits 2 - 14 of the word.

NO

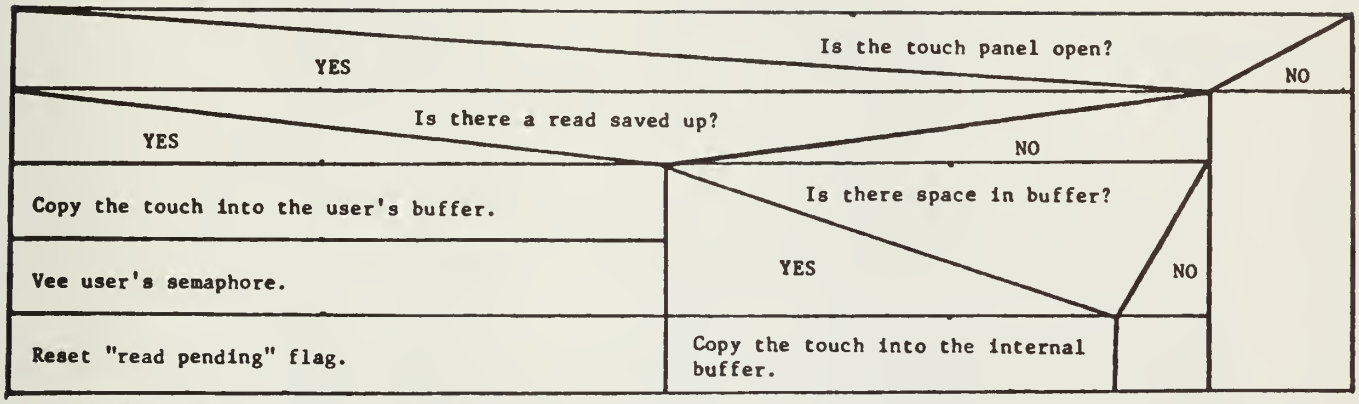
The word is a pointer to a request block containing the request code.

Switch on request code.

open	close	flush	peek	data	read	default
Set "open" flag on.	Set "open" flag off.	Empty the buffer.	Put the count of characters in buffer into request block.	(See tp_driver: data_type)	(See tp_driver: read_type)	Error: Reject the request.
Empty the input buffer.	Disable touch panel input.	Is a read outstanding? YES / NO	Put count of characters in buffer into request block.			
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.		Vee user's semaphore.				

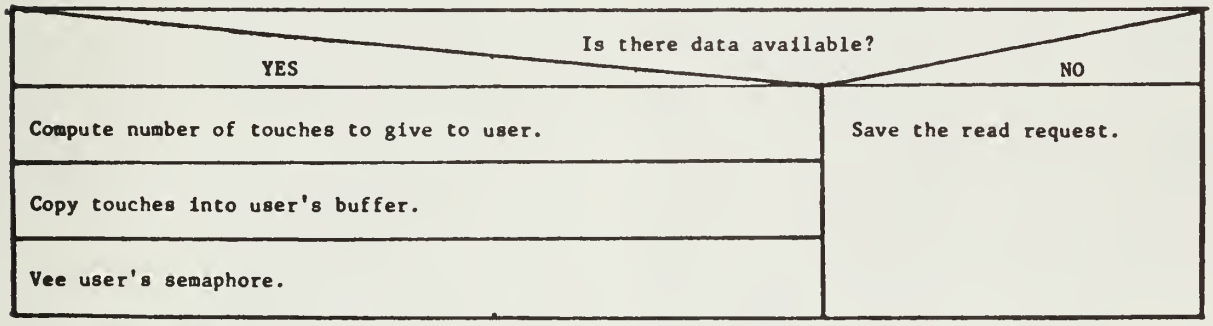
NOTE: This version of tp_driver is specific to the LSI 11 IT.

tp_driver: data_type*

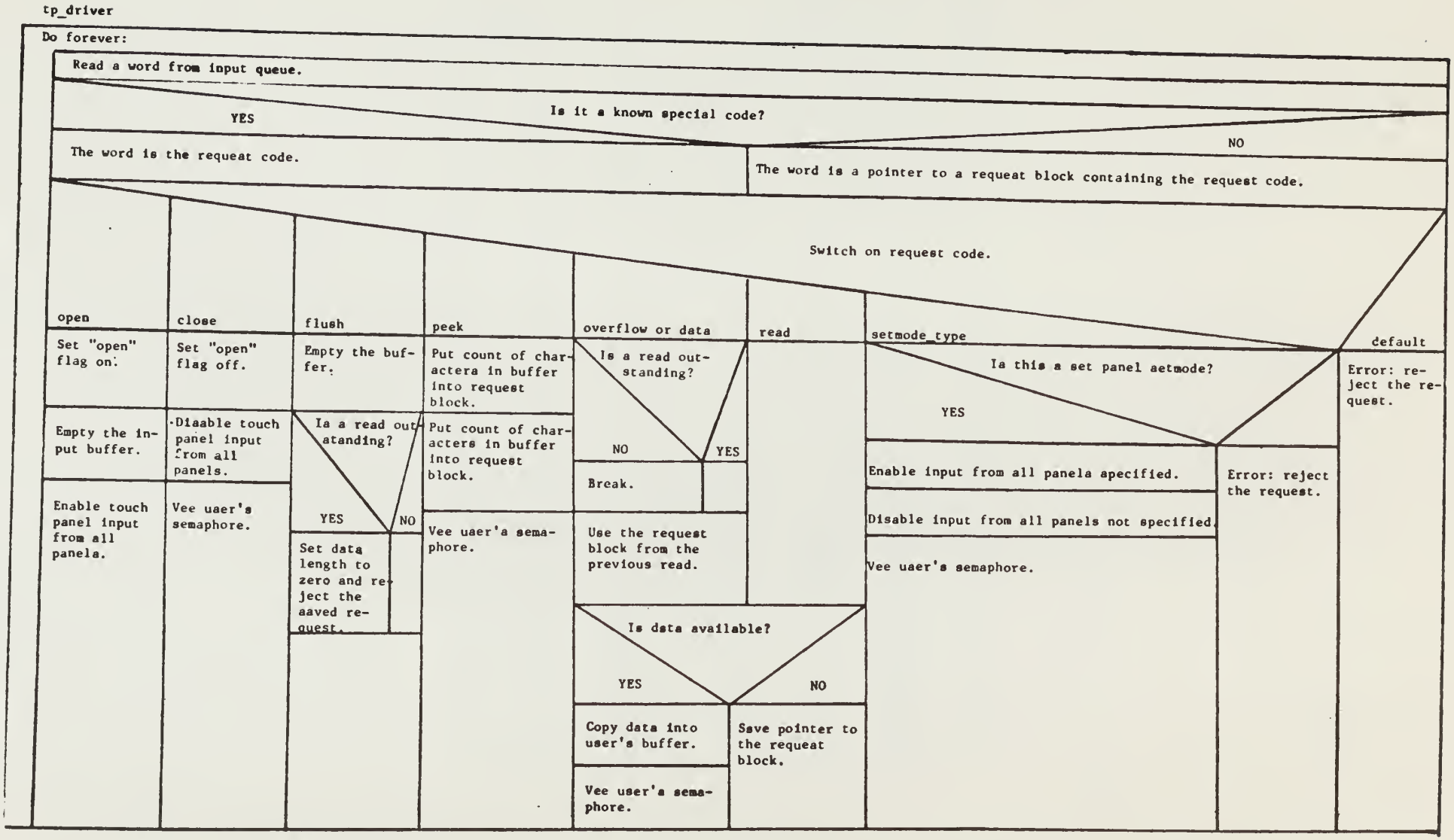


*This is not a procedure. It is one case in tp_driver.

tp_driver: read_type*



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



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                                                   YES │
│   ├──────────────────────────────────────────────────────────┤
│   │   tt_current(slot) = t.                                  │
│   │   Display the target.                                    │
│   │   Return(slot).                                          │
│   └──────────────────────────────────────────────────────────┘
│                                                              │
├──────────────────────────────────────────────────────────────┤
│ Return (-1).                                                 │
└──────────────────────────────────────────────────────────────┘
```

tt_arranger(list, num, values, labels, cs, 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:
  - Max_per_row greater than total number?
    - NO:
    - YES: Max_per_row = num.
- YES: Max_per_row = 1.

Are we doing vertical centering?

- NO:
- YES:
  - i = height of final block of targets.
  - Move bottom up by half of difference between i and available area.
  - height = i.

Are we doing horizontal centering?

- NO:
- YES:
  - i = width of row of targets.
  - Move left over by half the difference between i and available area.
  - width = i.

y = height of area.

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.

tt_create(t, x, y, width, height, value, label, cs, flag)

Copy the parameters into the appropriate fields of the target structure.

Reset the in_use flag.

Is tflags set so that the target would appear on no remote display heads?

YES

Set tflags so the target will appear on head 0, only.

NO

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

tt_create(t, x, y, width, height, value, label, cs, flag)

Copy the parameters into the appropriate fields of the target structure.

Reset the in_use flag.

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

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	YES
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	YES
Return (-1).	
Is this target flashable?	
NO	YES
	Lite all the dots in the target.
	Erase them all.
	Display the target.
Return (0).	

tt_label(slot)

Does slot indicate an active target?

NO: Return (-1).

YES:

t = pointer to target structure for target.

Is this a target that should not be labeled?

NO:

YES: Return (0).

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.

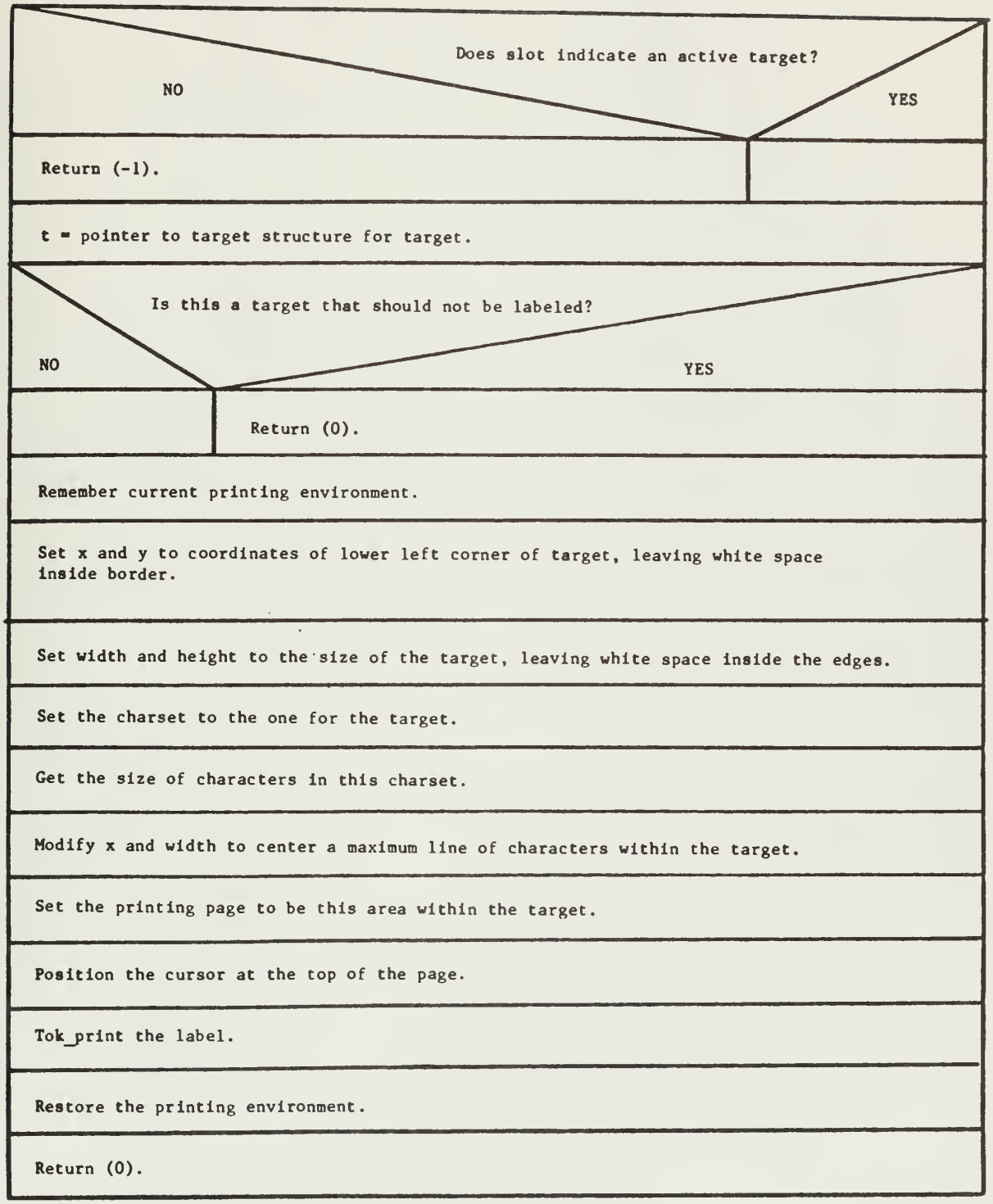
Restore the printing environment.

Rls_pnl.

Return (0).

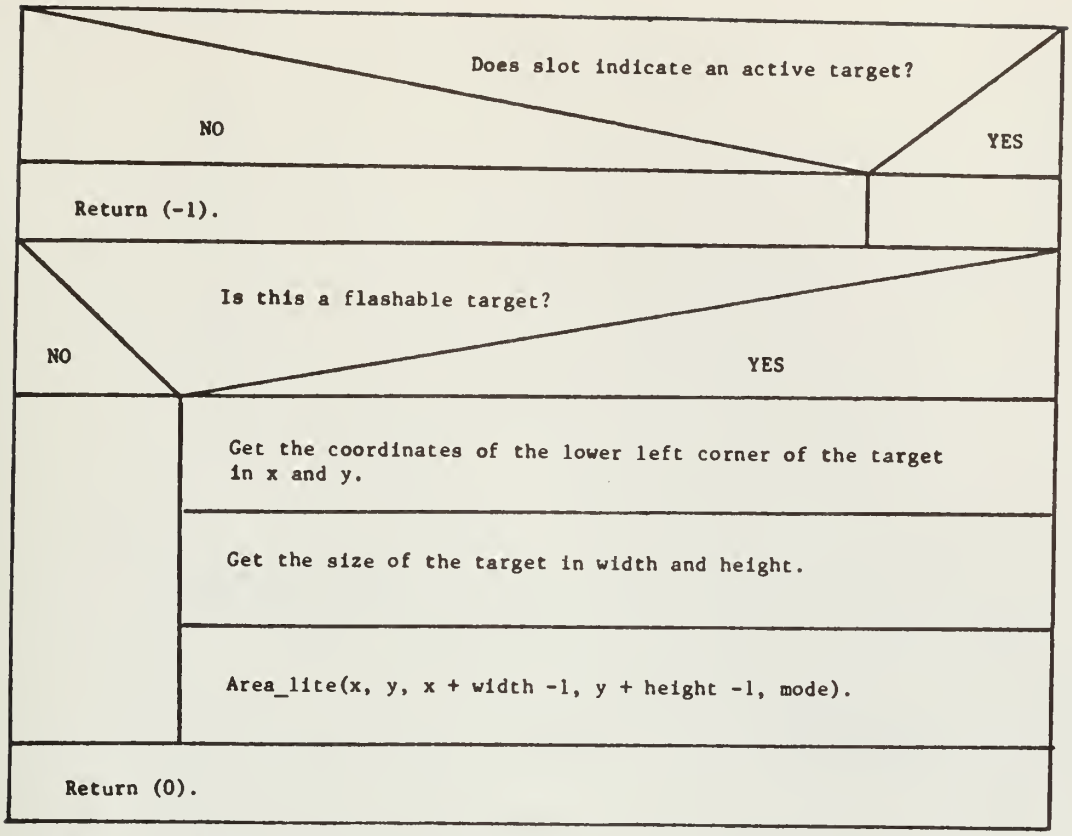
NOTE: This version of tt_label is specific to the Level 6 IT.

tt_label(slot)



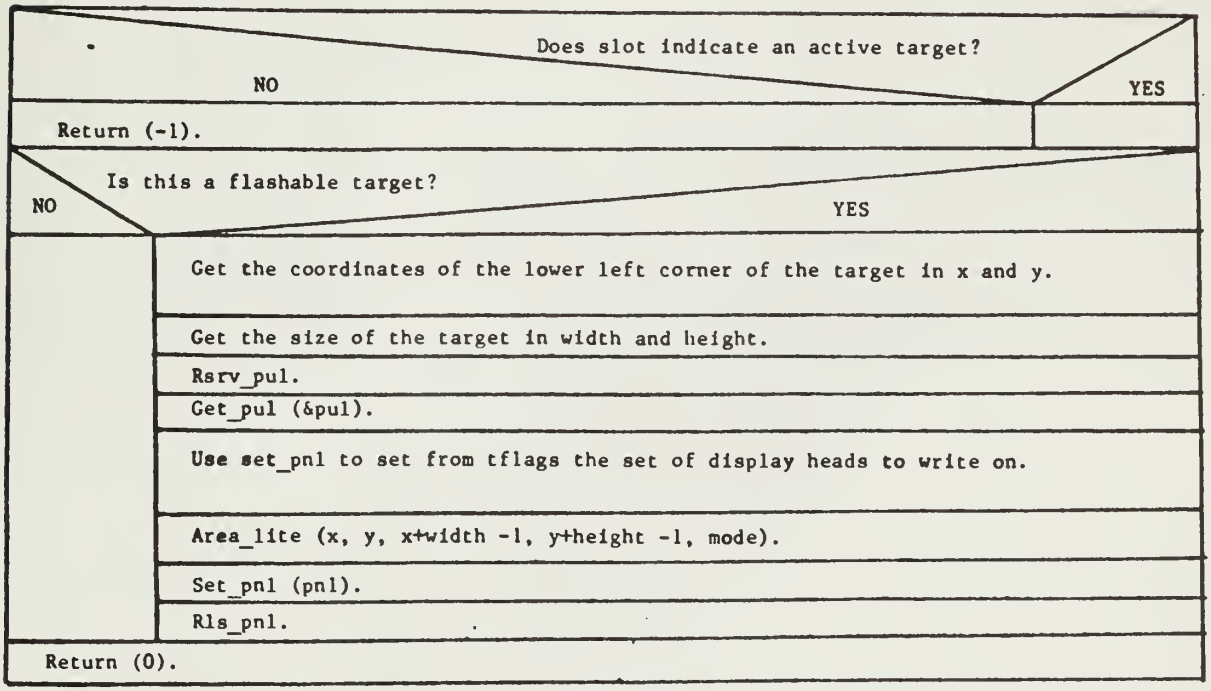
NOTE: This version of tt_label is specific to the LSI-11 IT.

tt_lite(slot, mode)



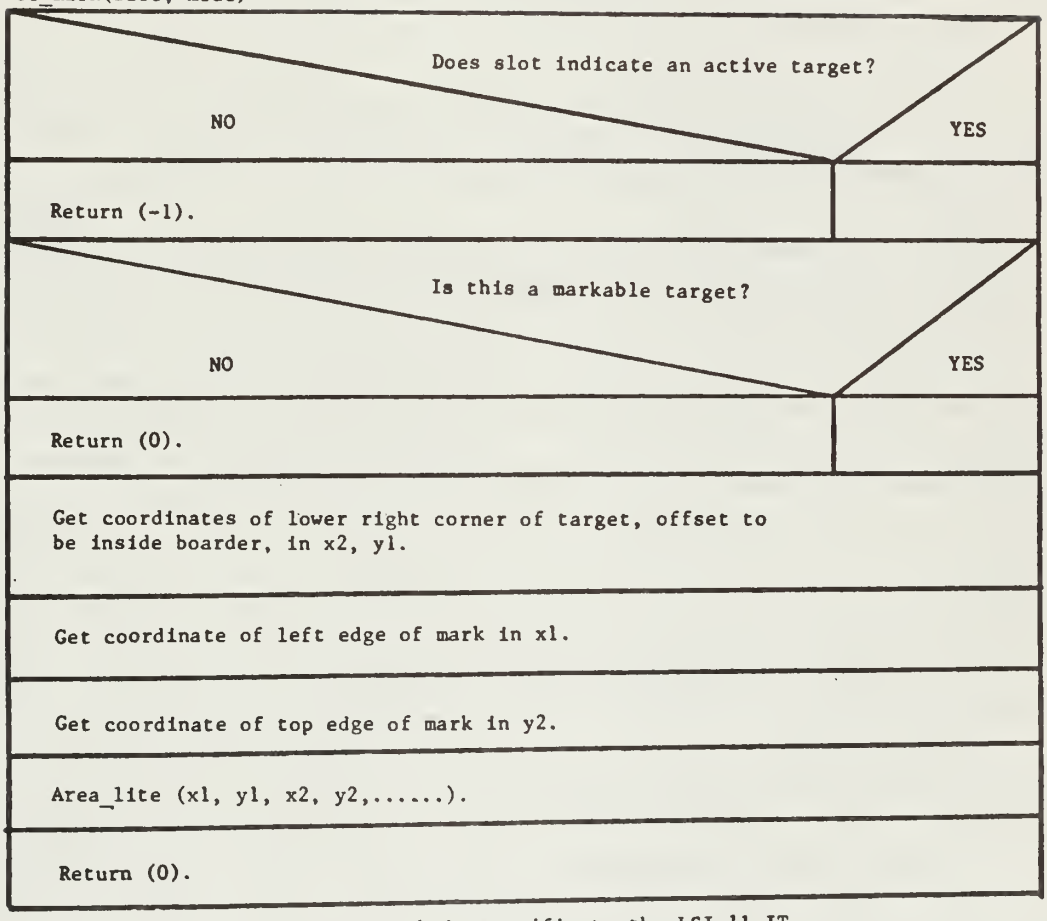
NOTE: This version of tt_lite is specific to the LSI-11 IT.

tt_lite(slot, mode)



This version of tt_lite is specific to the Level 6.

tt_mark(slot, mode)



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

tt_mark(slot, mode)

Does slot indicate an active target?	
NO	YES
Return (-1).	
Is this a markable target?	
NO	YES
Return (0).	
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.	
Rsrv_pnl.	
Get_pnl(&pnl).	
Use set_pnl to set from tflags the set of remote display heads to be written on.	
Area_lite(x1, 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	YES
Return (-1).	

t = tt_current(slot).

Is *t an outlineable target?

NO	YES
Return (0).	

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, y1, x2, and y2 to the inside of the target to leave white space around it.

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

Return (0).

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

tt_outline(slot, mode)

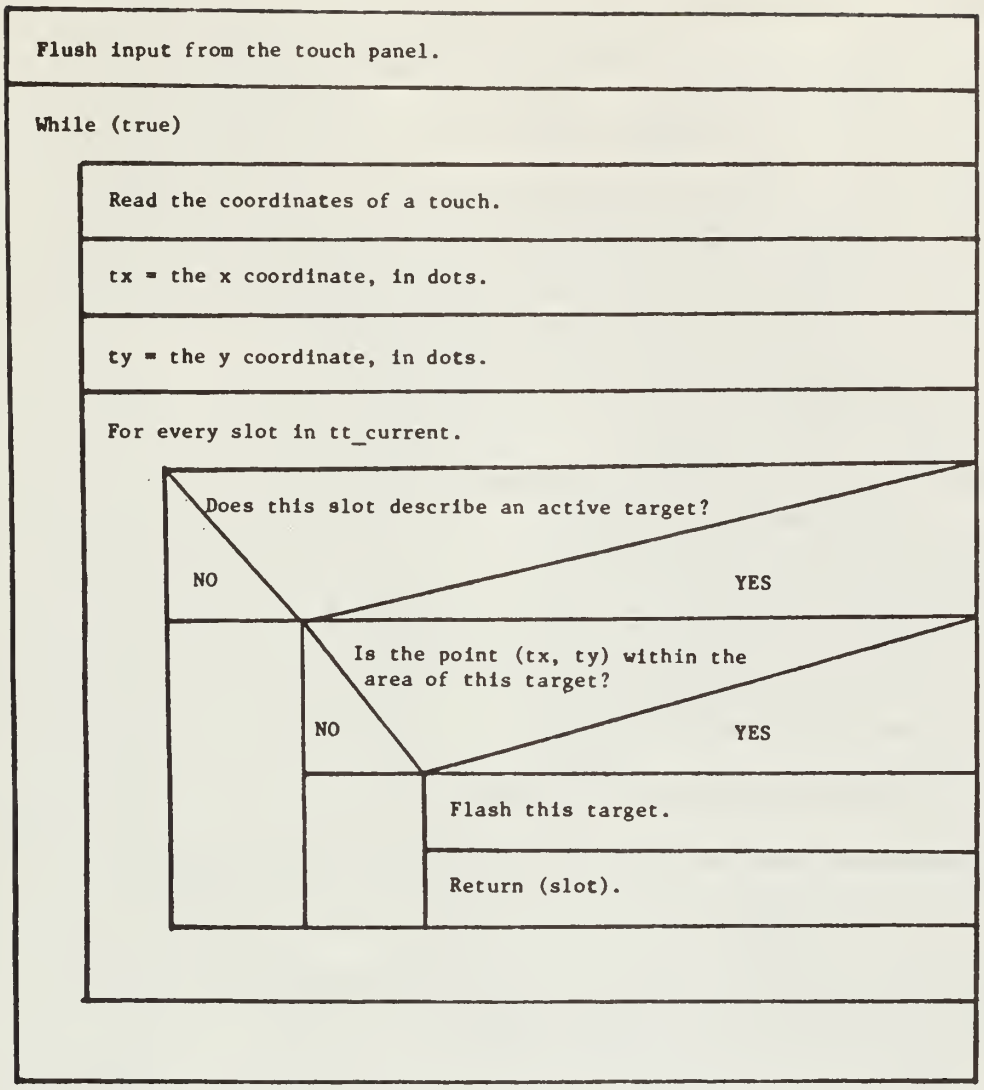
Does slot indicate an active target?	
NO	YES
Return (-1).	
t = tt_current(slot).	
Is *t an outlineable target?	
NO	YES
Return (0).	
Get coordinates of lower left corner of target in x1 and y1.	
Get coordinates of upper right corner of target in x2 and y2.	
Move x1, x2, and y2 to the inside of the target 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 tt_outline 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                                   YES         |     |   |
|   |   |                 +-------------------------+     |     |   |
|   |   |                 | Is the point (tx, ty)   |     |     |   |
|   |   |                 | within the area         |     |     |   |
|   |   |                 | of this target?         |     |     |   |
|   |   |                 | NO                  YES |     |     |   |
|   |   |                 |         +-------------+ |     |     |   |
|   |   |                 |         | Flash this  | |     |     |   |
|   |   |                 |         | target.     | |     |     |   |
|   |   |                 |         | Return      | |     |     |   |
|   |   |                 |         | (slot).     | |     |     |   |
|   |   |                 |         +-------------+ |     |     |   |
|   |   |                 +-------------------------+     |     |   |
|   |   +-------------------------------------------------+     |   |
|   +-----------------------------------------------------------+   |
+-------------------------------------------------------------------+
```

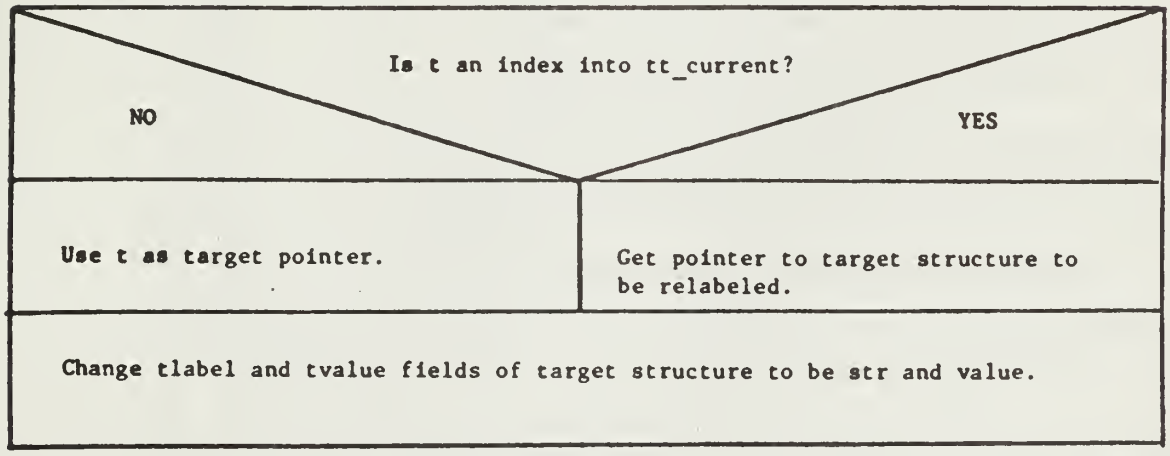
NOTE: This version of tt_read is specific to the Level 6 IT.

tt_read(touch)

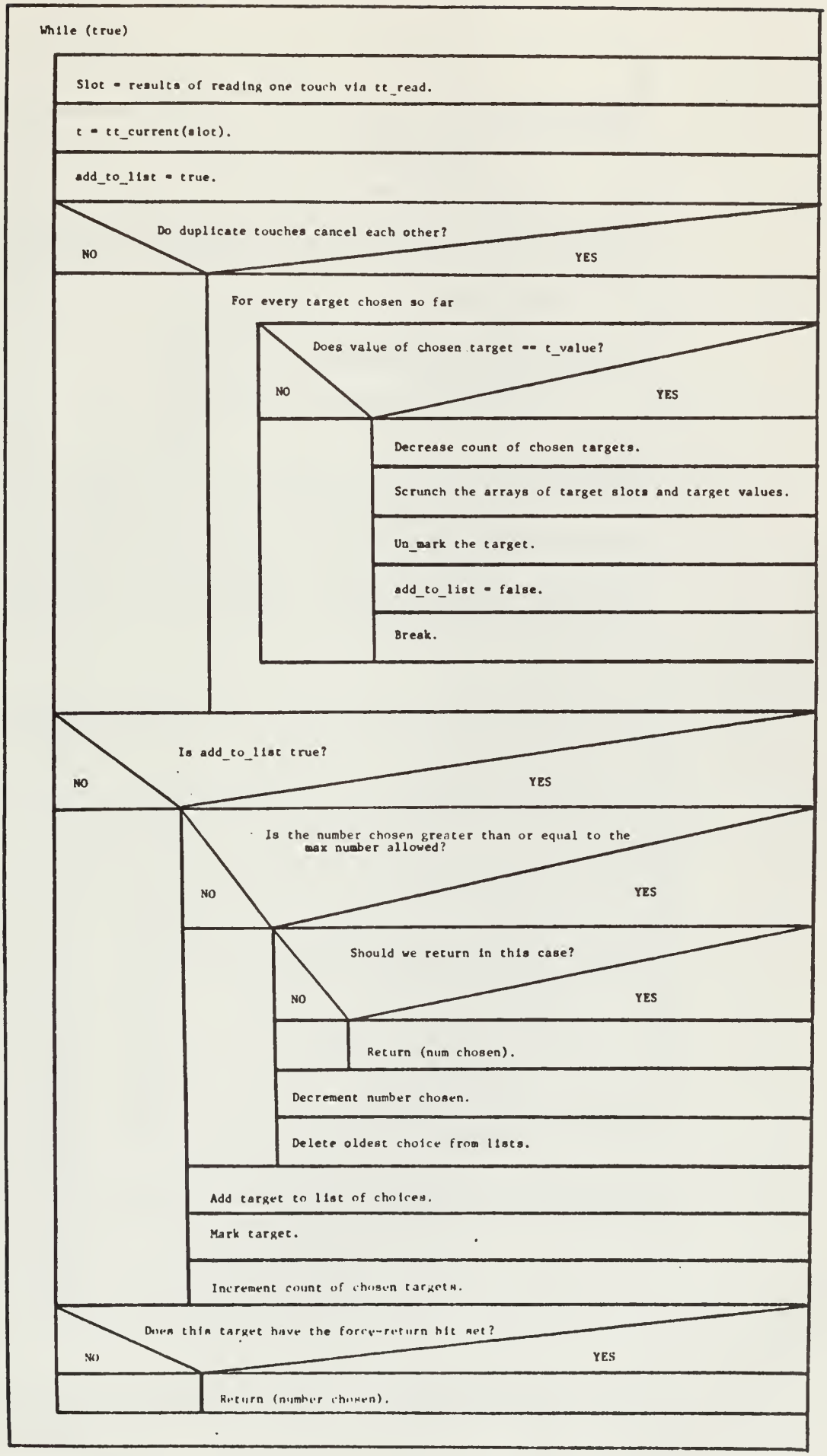


NOTE: This version of tt_read is specific to the LSI 11 IT.

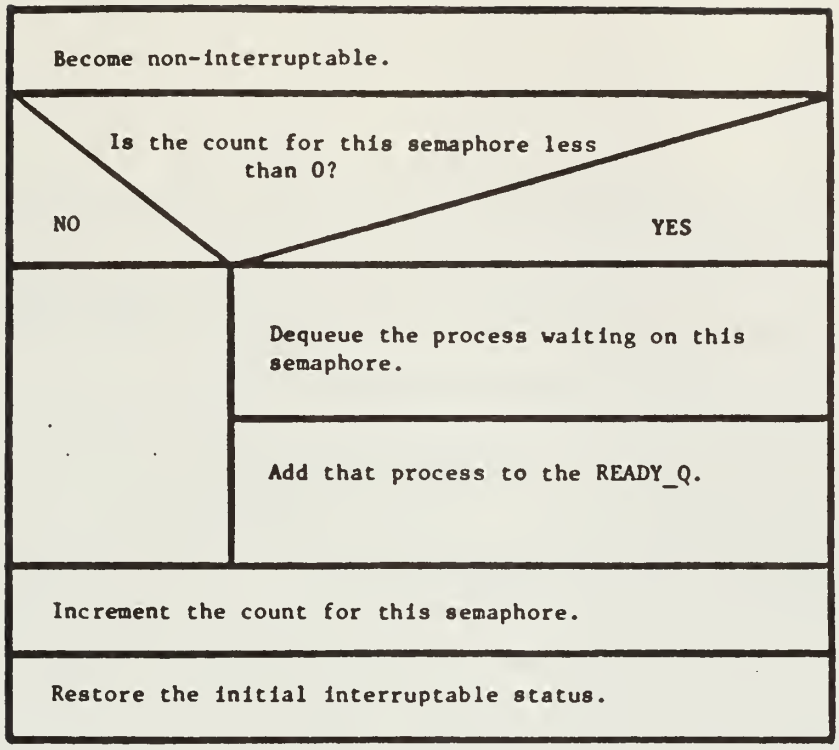
tt_relabel(t, str, value, mode)



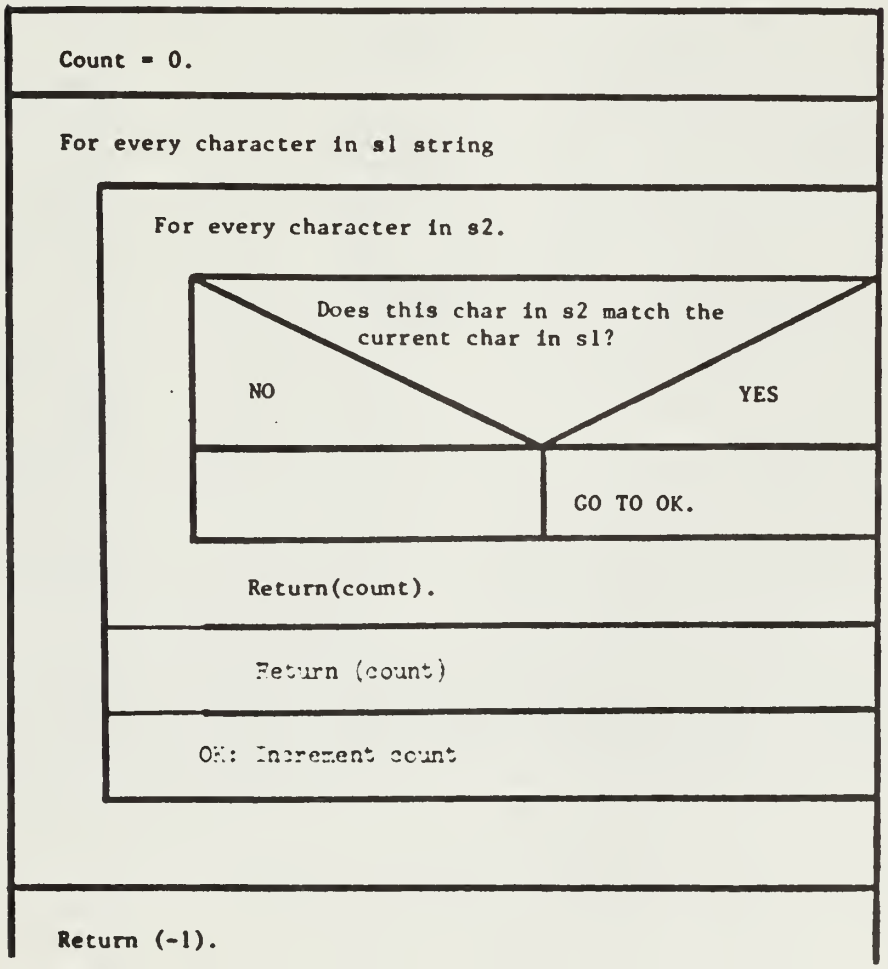
tt_selections(touch, max num tchs, num, ovrfl, values, slots)



vee(sem)



verify(s1, s2)



vip_proc ()

Initialize DUV11 Device.

Start transmission of a quiescent frame.

Do forever

Read word from input queue.

Is word a small odd integer?

YES: The word is the request code.

NO: The word is a pointer to a request block with the request code.

Switch on the request code.

open

Zero saved read and write indicators.

Vip line operating?

NO: Reject request.

YES: Set "open" flag on.

Vee user's semaphore.

close

Set "open" flag off.

Empty input buffer.

Outstanding read?

YES: Reject outstanding request.

NO:

Outstanding write?

YES: Reject outstanding request.

NO:

Vee user's semaphore.

flush

Null.

vip up

Set "vip_ready" flag on.

vip down

Set "vip_ready" flag off.

data

Is a read pending?

Break.

Get pointer to request block for the pending read.

See vip_proc: vip_read

read

Null.

write

Is a write pending?

YES: Reject the new write request.

Break.

NO: Set this up as the current write request.

See vip_proc:vip_write

ack

Null.

peek

Put count of characters in input buffer into the request block.

Vee user's semaphore.

default

Reject the request.

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

Vee 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?

YES

Vee user's request.

Set flag indicating no write pending.

Break.

NO

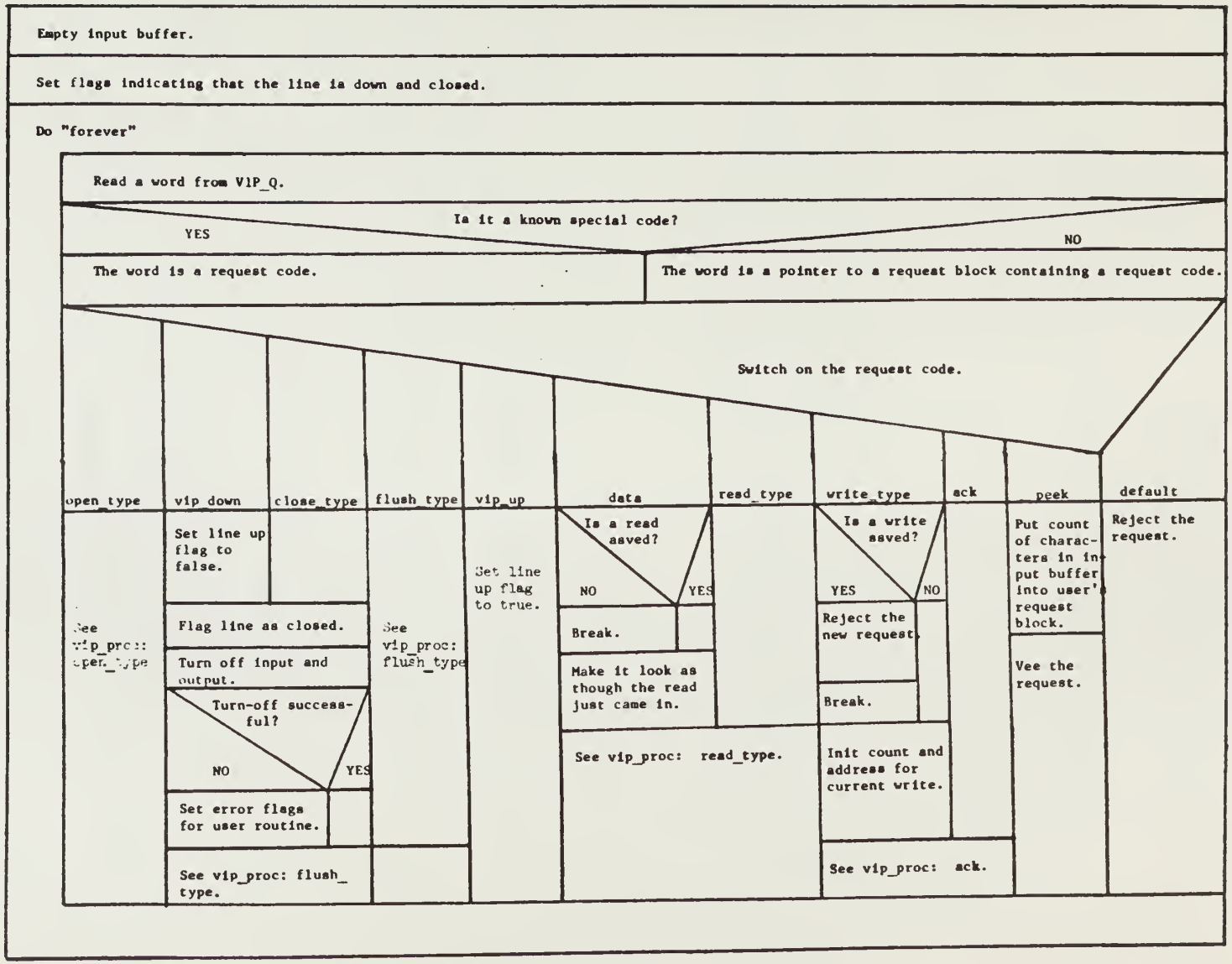
Construct an output message, using as much data from the current write request as will fit in the output buffer.

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()



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

vip_proc: ack

Is there s current write?	
NO	YES
Break.	
Is the count zero (i.e. is the write now completed)?	
YES	NO
Set return to user to show successful write.	
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 incresse current sddress by the number of user's characters put in messsge.	
Set a flag so that the interrupt routine will initiate transmission of this message when the time comes.	

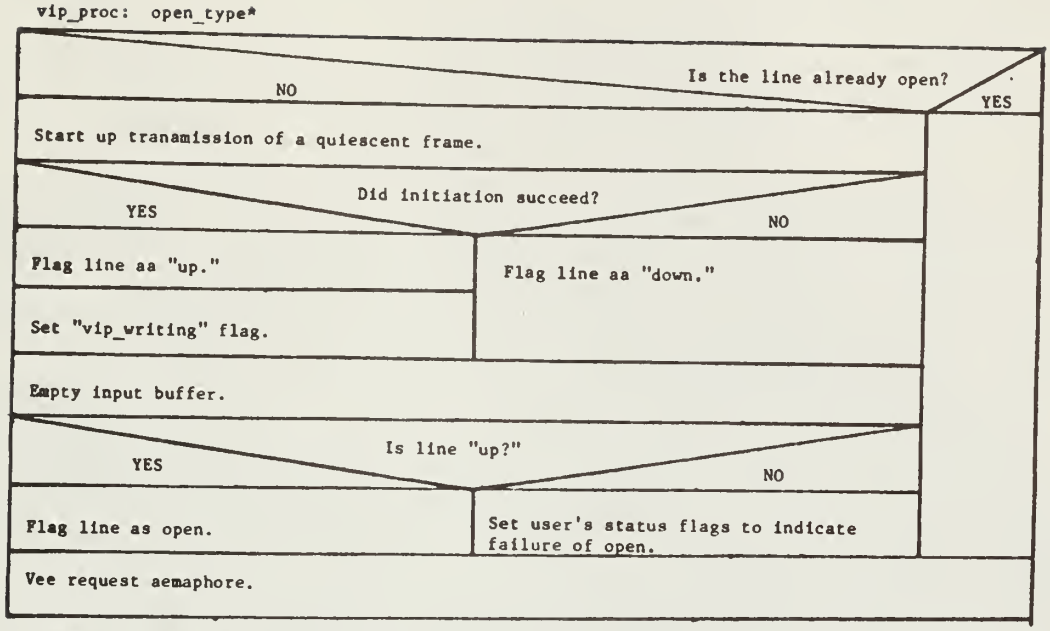
*This is not a procedure. It is one case in vip_proc snd is specific to the Level 6 IT.

vip_proc: flush_type*

Is this s close or a flush?	
YES	NO
Empty the input buffer.	
Is s read saved up?	
YES	NO
Reject the saved read.	
Is a write saved up?	
YES	NO
Reject the write.	
Is this a close or s flush?	
YES	NO
Vee the request semaphore.	

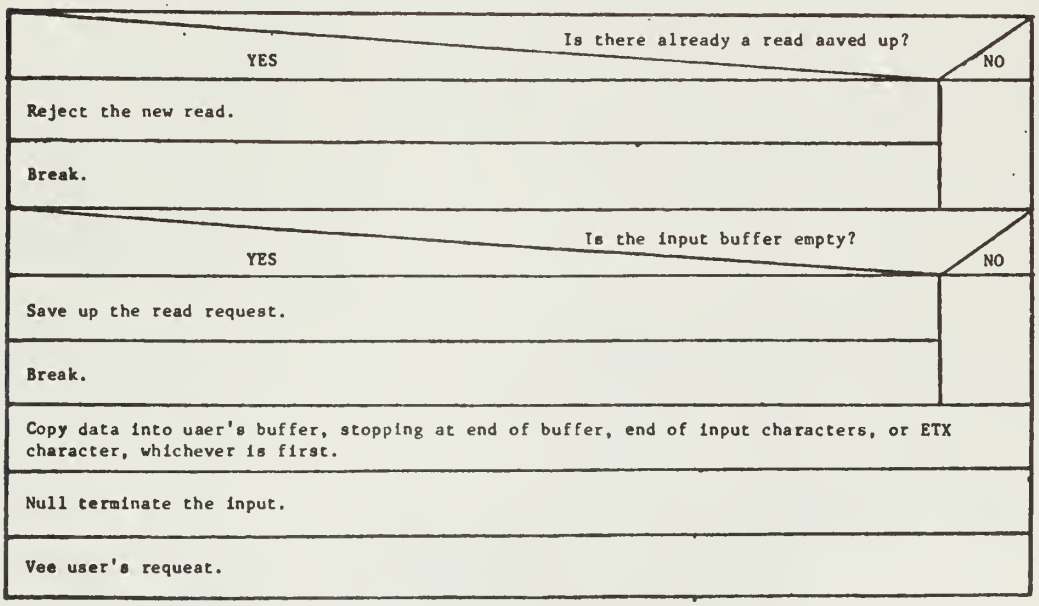
*This is not a procedure. It is one csse in vip_proc snd is specific to the Level 6 IT.

vip_proc: open_type*



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

vip_proc: read_type*



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

vipint()

Is flag set indicating that this was a write interrupt?

YES

Is the vip line still open?

YES

Start up a read.

Was there an error reading the status or did starting the read fail?

YES

Send "vip_down" message to vip_proc.

Set "writing" flag to false.

Return.

NO

NO

Head line status and number of characters read.

Was there error reading status?

YES

Write vip_down message to vip_proc.

Return.

NO

Do for each character just read in:

The following switch statement is a finite state machine which parses input messages.

Switch on inp state.

SOH_EOT_wait	adr_wait	stat_wait	null_STX_wait	null_ETX_wait	STX_wait	text_rev	bcc_wait	got_trans or error_flush
SOH_EOT_wait.	adr_wait.	stat_wait.	Update longitudinal parity check.	Update longitudinal parity check.	Update longitudinal parity check.	text_rev.	bcc_wait.	Null.
			Input=STX? YES: Inp_state= null_ETX_ wait. NO:	Input=ETX? YES: Inp_ state= bcc wait. NO: Inp_ state= error flush.	Input=STX? YES: Inp_state= text_rev. NO:			

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

YES

Did the FSM find a valid message?

NO

Was it a bad message?

YES

Set up to send a NAK.

NO

Set up to send a quiescent frame.

YES

Switch on type of message received.

quiescent frame	ACK	NAK	data
See vipint: quiescent	See vipint: ACK.	See vipint: NAK.	See vipint: data.

Is vip line still open?

YES

Start transmission of message selected above.

Did it start ok?

YES

Set "writing" flag.

NO

Send vip_down message to vip_proc.

NO

Is vip line still open?

YES

Start up a new read.

Did it start ok?

NO

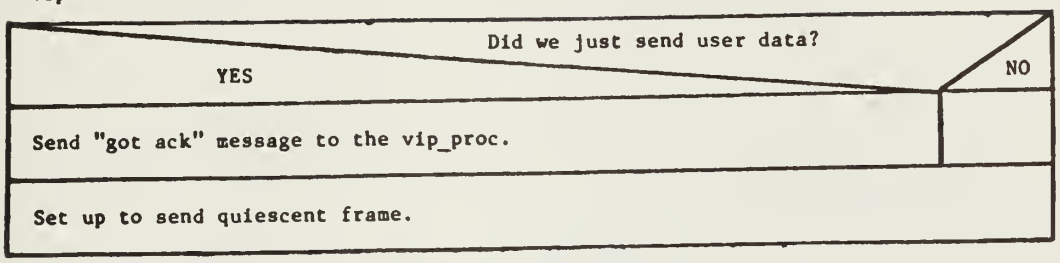
Send vip_down message to vip_proc.

YES

NO

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

vipint: ACK*

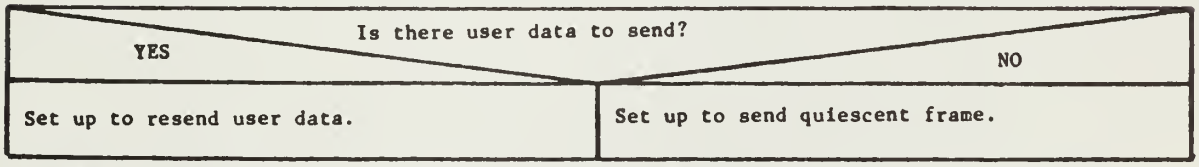


*This is not a procedure. It is one case in vipint and specific to the Level 6 IT.

vipint: data*

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

vipint: NAK*



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

viprint ()

Have any of the status register bits changed?

YES

The DUV11 become ready?

YES

Send a message to the vip process.

NO

The DUV11 become not ready?

YES

Send a message to the vip process.

NO

The carrier just went away?

YES

NO

We now must make some response to the DN355 which is at the other end of the line.

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

YES

Switch on "input_type."

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

NO

Did what we got look like a quiescent frame?

YES: Set up to send a quiescent frame.

NO: Set up to send a NAK.

Set status registers to start a transmission.

Set inp_state = 0.

Return.

Is a character available?

YES

Read the character into variable "input."

NO

return.

Was there a transmission error and we are in the middle of a transmission?

YES

Set inp_state = error_state.

Return.

NO

Inp_state = 0?

YES

Inp_state = SOH_EOT wait.

Initialize temporary input buffer pointer.

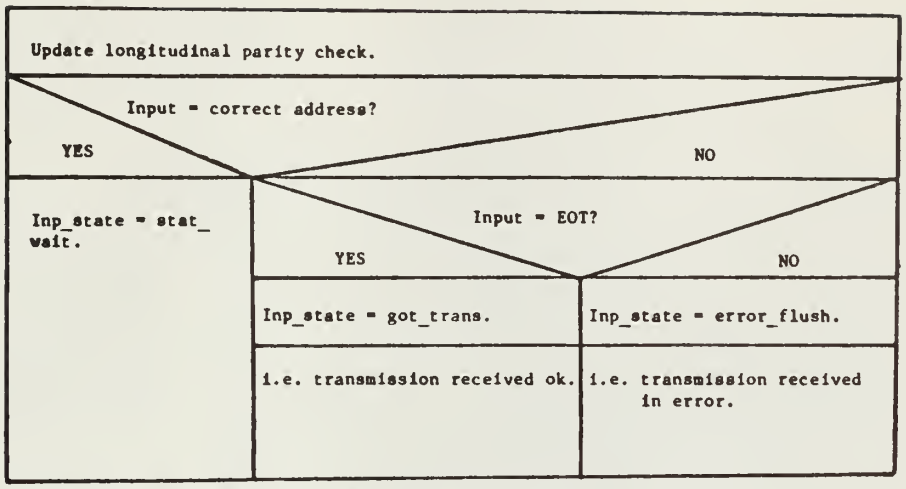
Input_type = 0.

The following switch statement is a finite state machine which parses input messages.

Switch on inp_state.

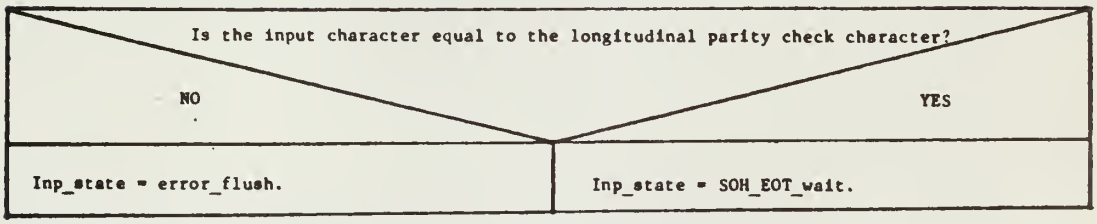
SOH_EOT_wait	adr_wait	stat_wait	nul' STX_wait	null_ETX_wait	STX_wait	text_rev	bcc_wait	got_trans or error_flush
SOH_EOT_wait.	adr_wait.	stat_wait	Update longitudinal parity check.	Update longitudinal parity check.	Update longitudinal parity check.	text_rev.	bcc_wait.	Null.
			Input=STX? YES: Inp state= null ETX wait. NO:	Input=ETX? YES: Inp state= bcc wait. NO: Inp state= error flush.	Input=STX? YES: Inp state= text rev. NO:			

adr_wait*



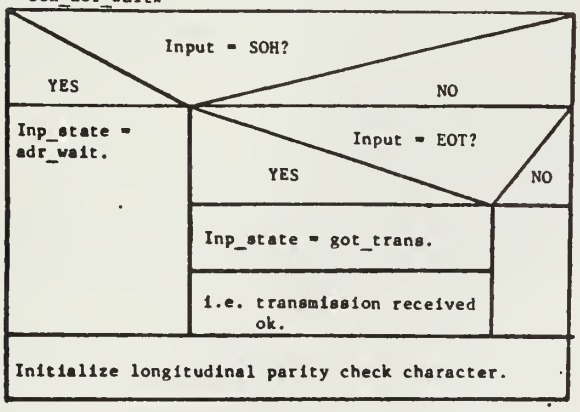
*This is not a procedure. It is one case in viprint.

bcc_wait*



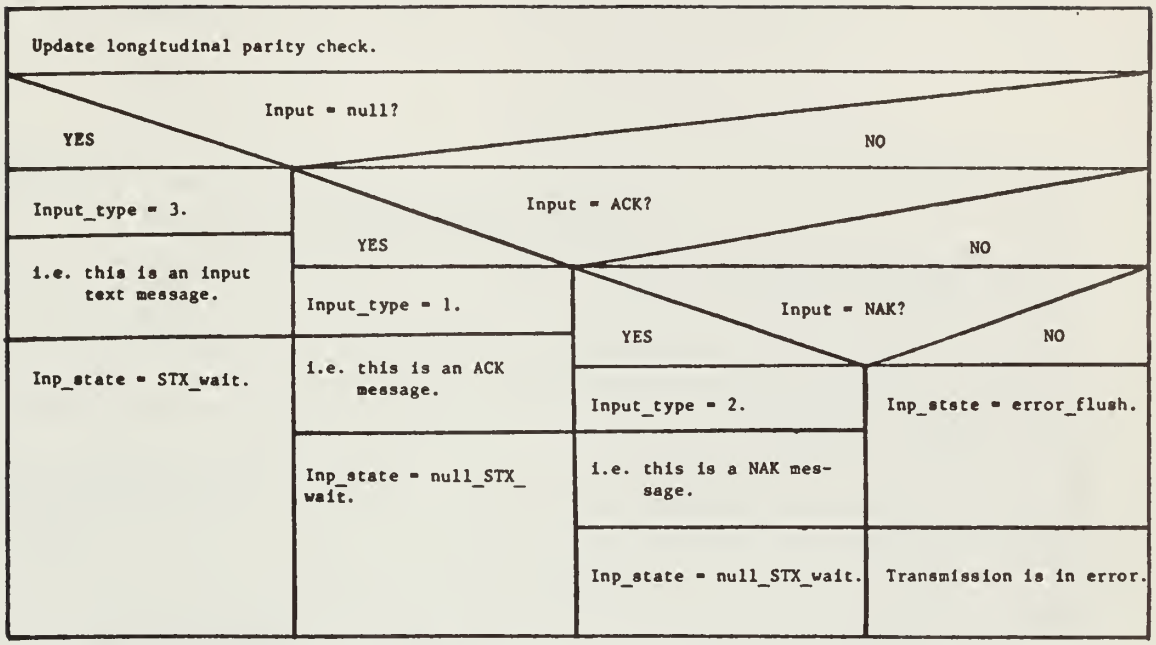
*This is not a procedure. It is one case in viprint.

SOH_EOT_wait*



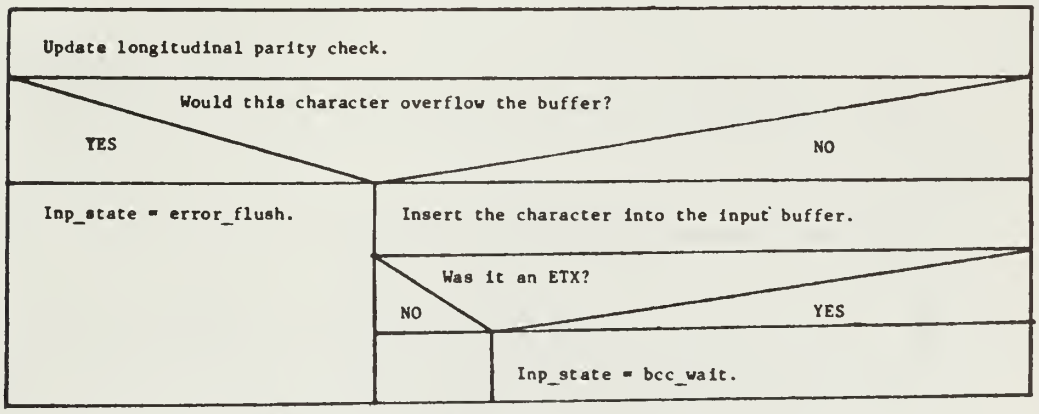
*This is not a procedure. It is one case in viprint.

stat_wait*



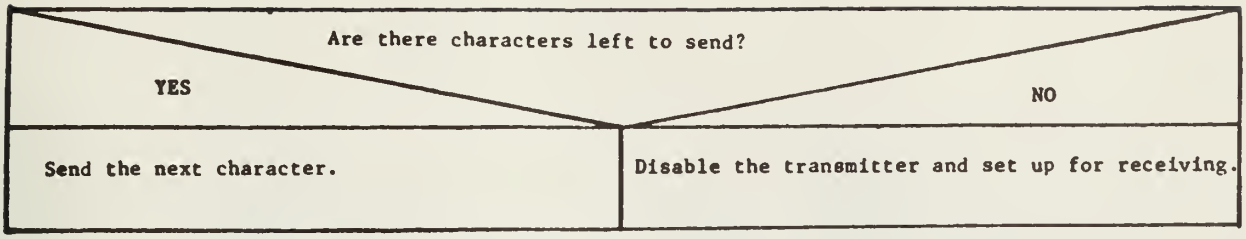
*This is not a procedure. It is one case in viprint.

text_rev *

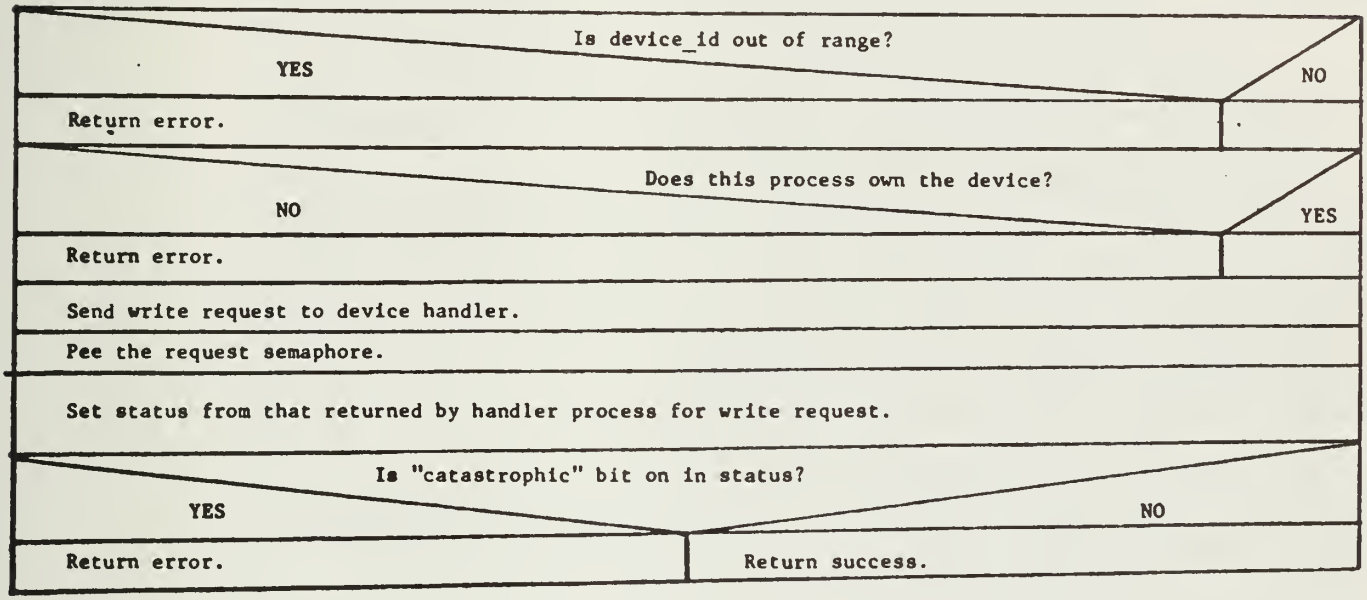


*This is not a procedure. It is one case in viprint.

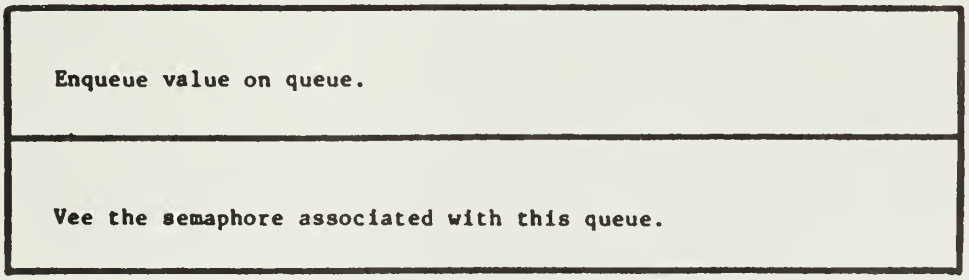
vipxint()



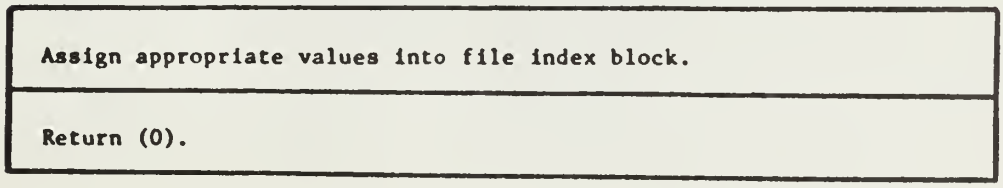
write(device_id, &status)



write_q(q_ptr, value)



xopen(fib, index_b, o, slot)



Z80_int

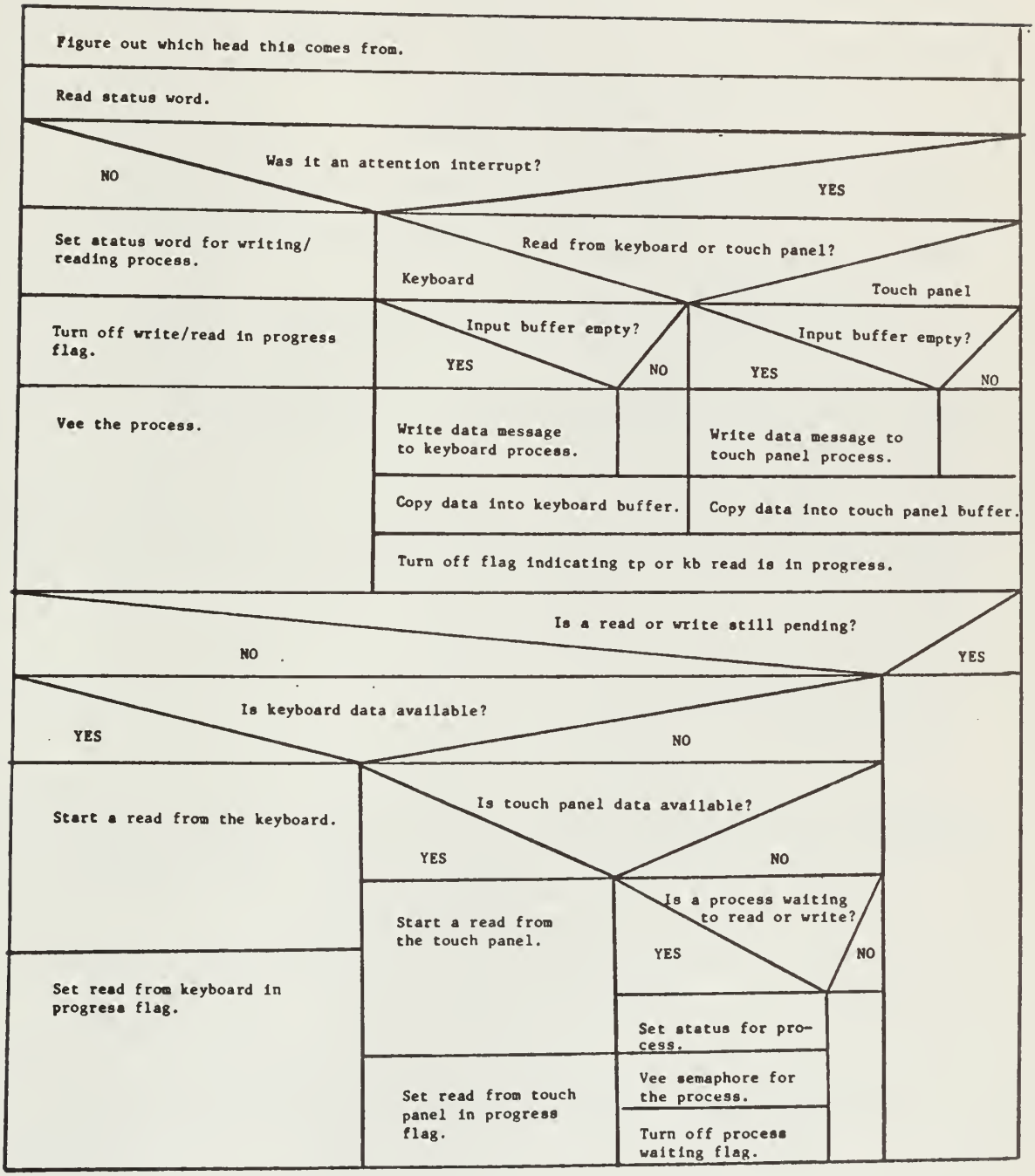
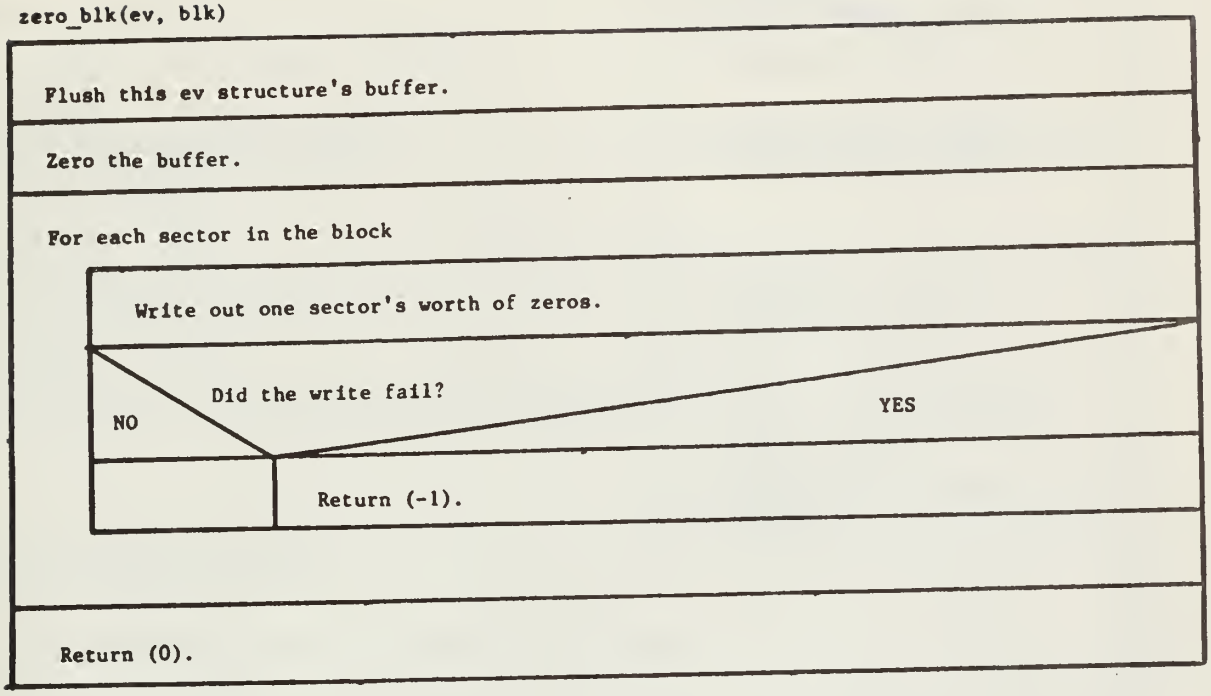
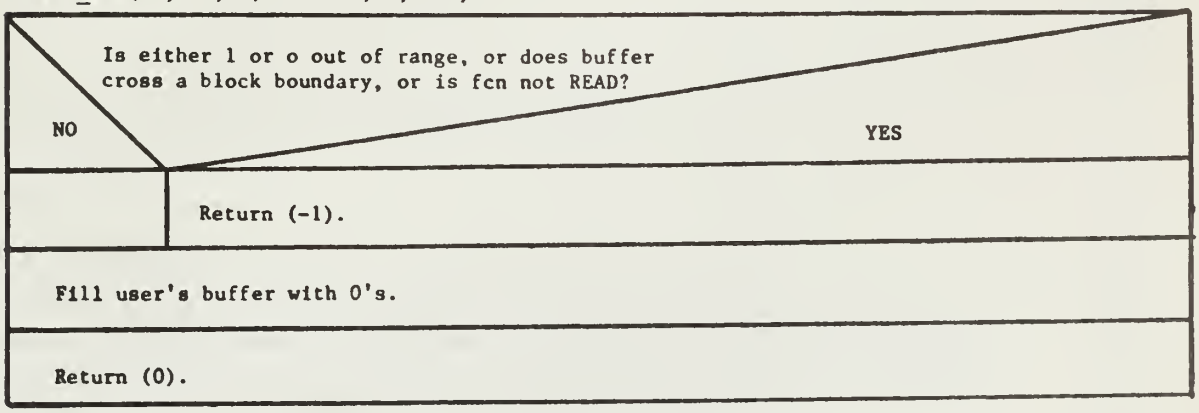


Figure out which head this comes from.
Read status word.
Was it an attention interrupt?
NO
YES
Set status word for writing/ reading process.
Read from keyboard or touch panel?
Keyboard
Touch panel
Turn off write/read in progress flag.
Input buffer empty?
YES
NO
Input buffer empty?
YES
NO
Vee the process.
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
YES
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 read from keyboard in progress flag.
Set status for process.
Vee semaphore for the process.
Set read from touch panel in progress flag.
Turn off process waiting flag.

zero_blk(ev, blk)



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



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
CAC Document Number 252
CCTC-WAD Document Number 7523

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

Intelligent Terminal Software Flowcharts

5. TYPE OF REPORT & PERIOD COVERED

Research

6. PERFORMING ORG. REPORT NUMBER
CAC #252

7. AUTHOR(s)

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

8. CONTRACT OR GRANT NUMBER(s)

DCA100-76-C-0088

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Center for Advanced Computation
University of Illinois
Urbana, Illinois 61801

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Command and Control Technical Center
11440 Isaac Newton Square, North
Reston, Virginia 22090

12. REPORT DATE
October 31, 1977

13. NUMBER OF PAGES

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

15. SECURITY CLASS. (of this report)

UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Copies may be obtained from the address in 11 above.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

No restriction on distribution.

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Intelligent Terminal

Man-Machine Interface

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

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."

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)