Published: 11/21/68

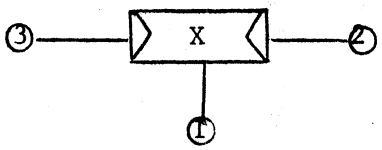
Identification

File System Flowcharts
A. Bensoussan

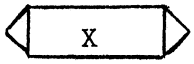
This document contains a series of flowcharts of the file system procedures (as is at May 1968), made following the EPL code. Although not complete and not error proof, it may be used as a guide for having a precise idea about what is the function performed by a procedure and how it is performed.

Data bases are not described but references to them are made using the name which appears in the EPL declaration.

The following notation is used for locks:

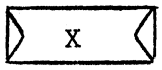


Try to lock X and go to
1 if successful
2 if X was locked by another process.
3 if X was already locked by the process.

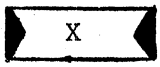


Unlock X

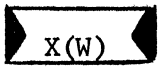
The difference between the various entries to ilock is as follows:



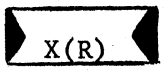
loop lock



lock



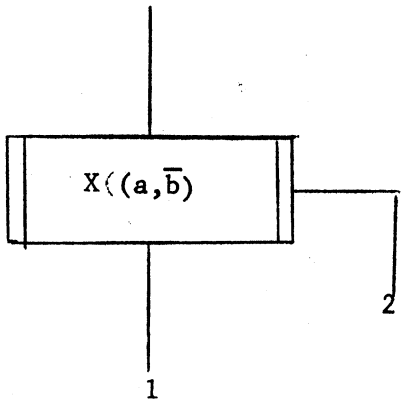
mod lock



read lock

No special notation for try lock. It is explicitly indicated in the flowchart.

calls =

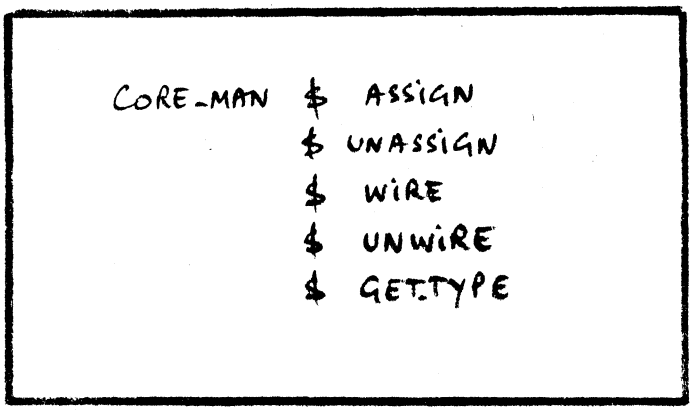


Call X (a,$\overline{b}$) where a is input argument

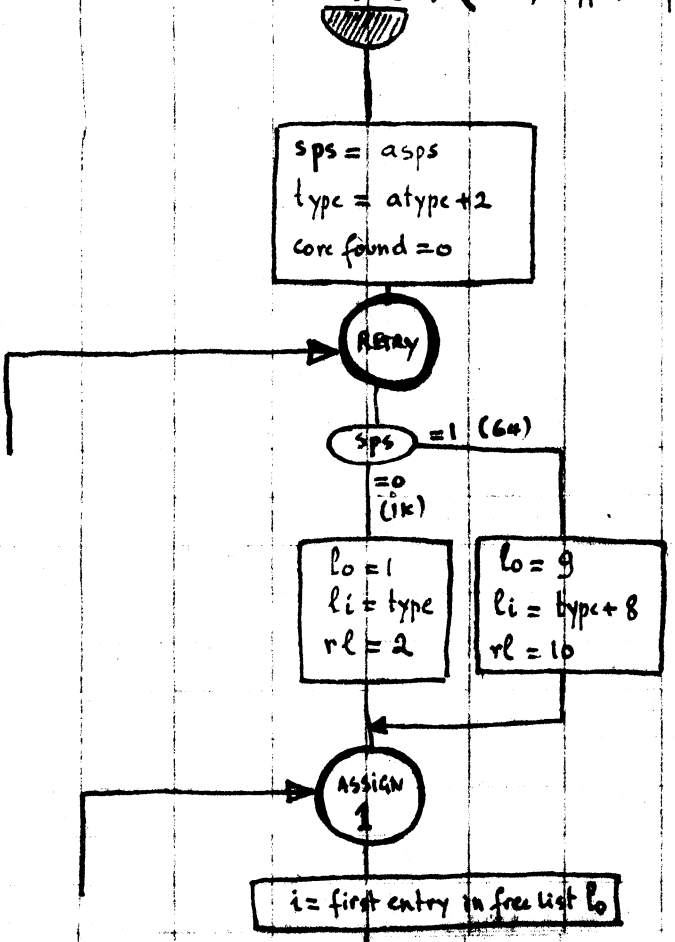
$\overline{b}$ is returned argument

if no error code go to 1
if error code go to 2

CORE CONTROL



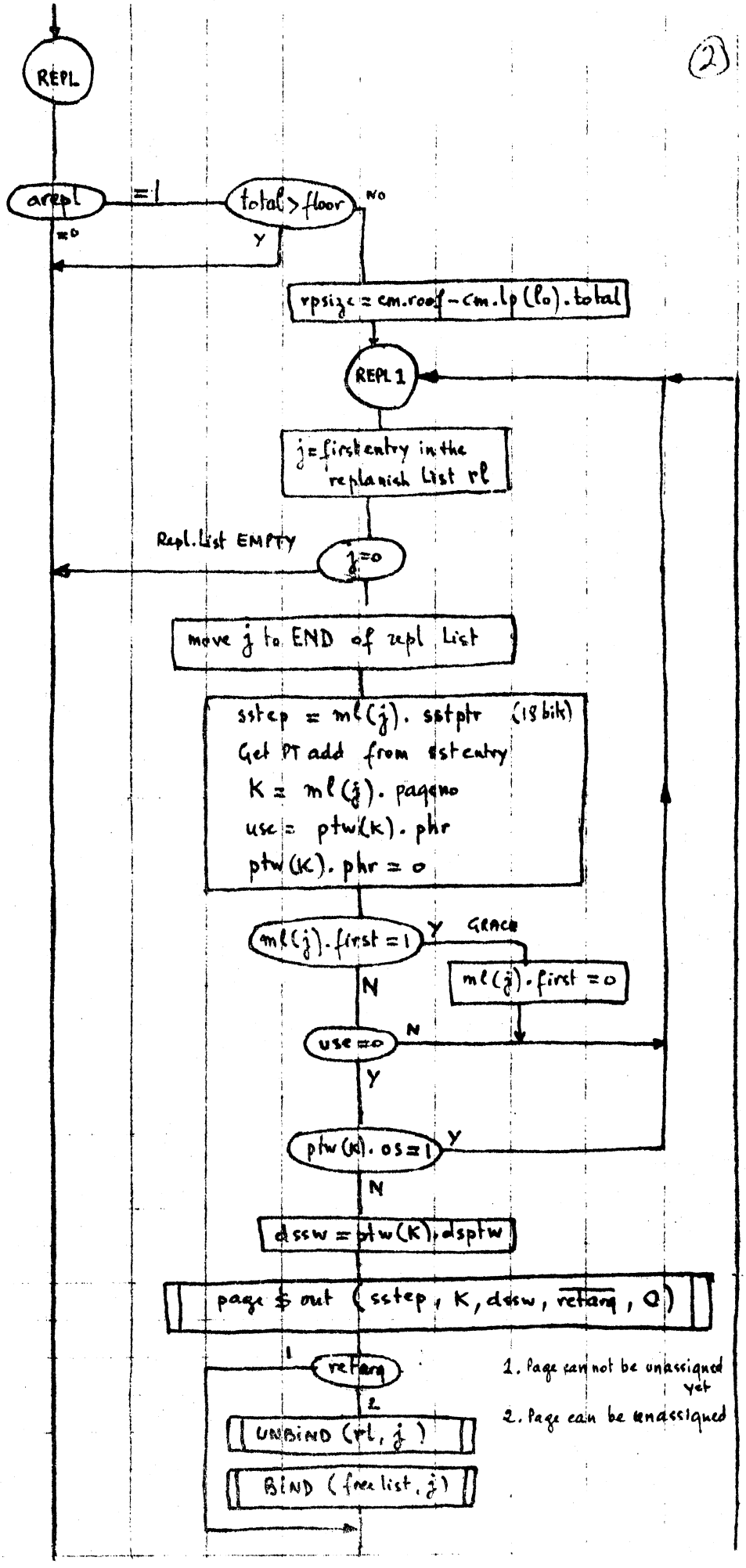
ASSIGN (aloc, atype, asps, apageno, asstp, areplen)



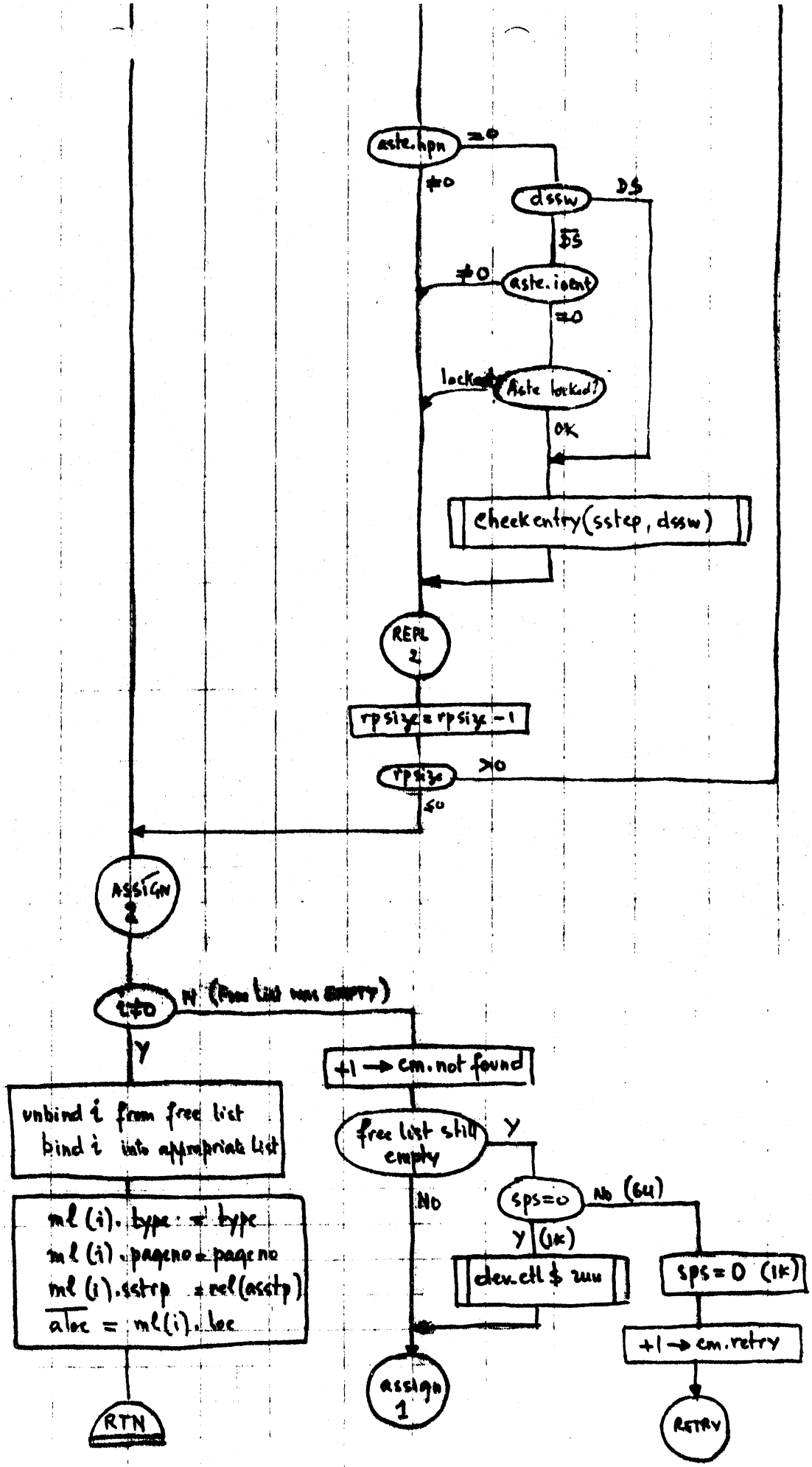
List #	[illegible]	Type
0	unused*	0
1	free	1
2	Normal	2
3	wired down	3
4	PERMANENT	4
5	temporary	5
6	unused type	6
7	split 1024 → 64	7
8	unused	0
9	free	1
10	Normal	2
11	wired down	3
12	Permanent	4
13	temporary	5
14	unused type	6
15	unused	7

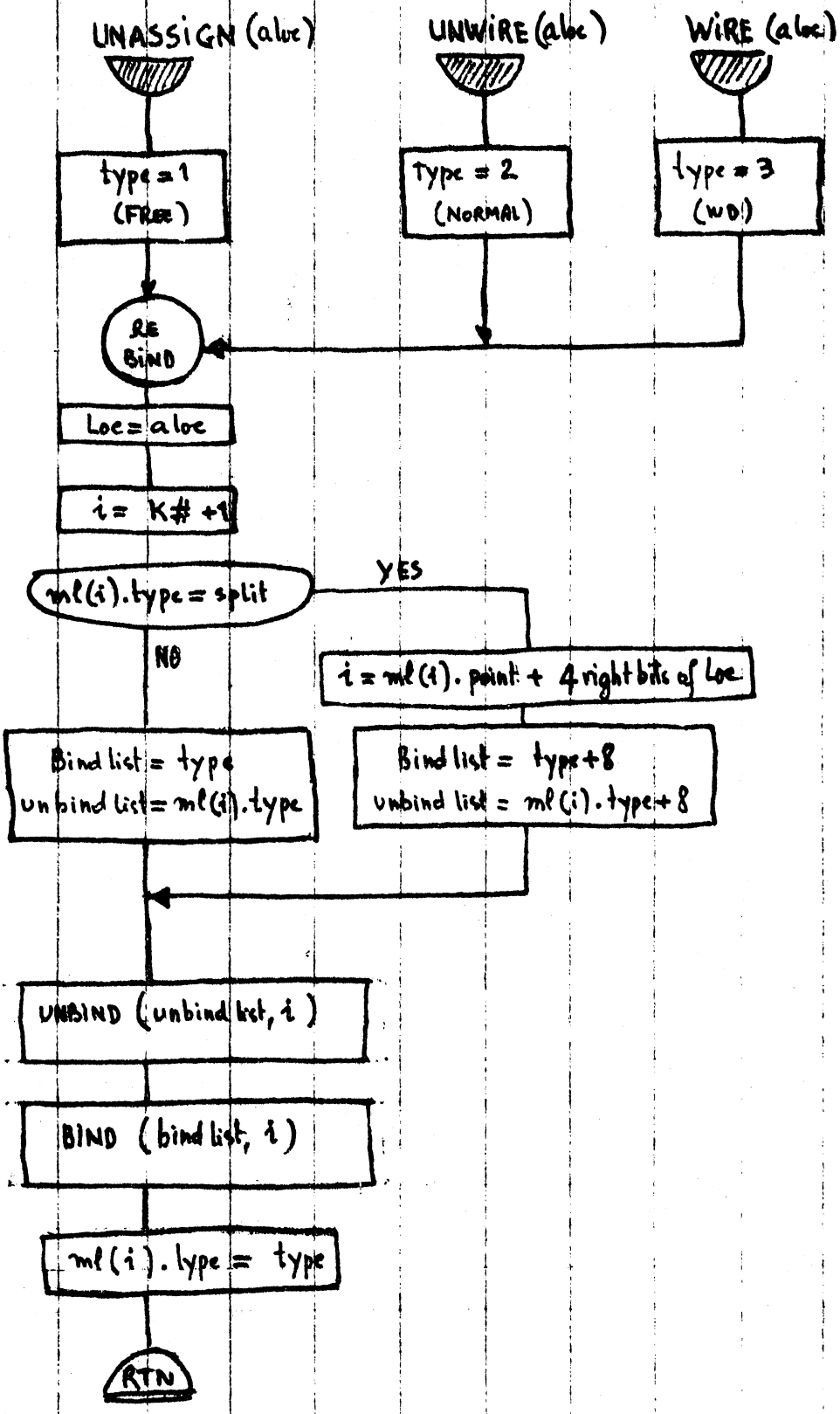
1K

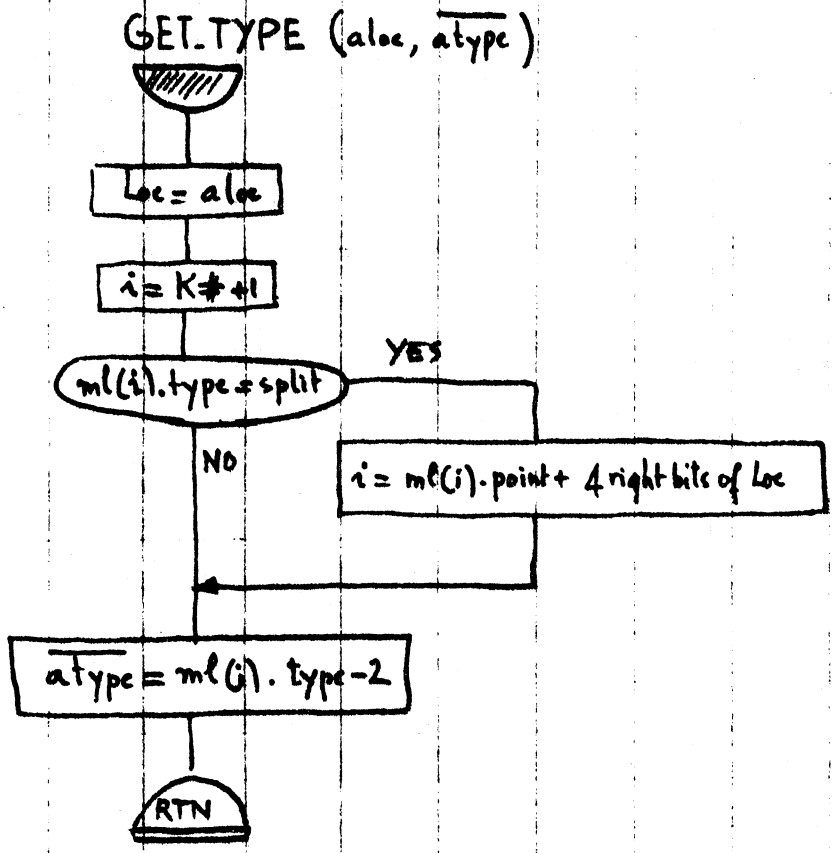
64



1. Page can not be unassigned yet
2. Page can be unassigned


aste.hpn
=0
≠0
dssw
DS
SS
≠0
aste.ioent
=0
lock
Aste locked?
OK
checkentry(sstep, dssw)
REPL 2
rpsiz = rpsiz - 1
rpsiz
>0
≤0
ASSIGN 2
L≠0
N (From list was EMPTY)
Y
+1 → cm.not found
unbind i from free list
bind i into appropriate list
free list still empty
Y
NO
sps=0
No (64)
Y (1K)
clear ctl $ zuu
sps = 0 (1K)
+1 → cm.retry
ml(i).type := type
ml(i).pageno := pageno
ml(i).sstrp := rel(asstp)
aloc = ml(i).loc
assign 1
RTN
RETRY


UNASSIGN (aloc)
UNWIRE (aloc)
WIRE (aloc)
type = 1 (FREE)
Type = 2 (NORMAL)
type = 3 (WD)
RE BIND
Loc = aloc
i = K# + 1
ml(i).type = split
YES
NO
i = ml(i).point + 4 right bits of Loc
Bind list = type
unbind list = ml(i).type
Bind list = type + 8
unbind list = ml(i).type + 8
UNBIND (unbind list, i)
BIND (bind list, i)
ml(i).type = type
RTN


GET_TYPE (aloc, atype)
Loc = aloc
i = K# + 1
ml(i).type ≠ split
YES
NO
i = ml(i).point + 4 right bits of Loc
atype = ml(i).type - 2
RTN

PAGE CONTROL

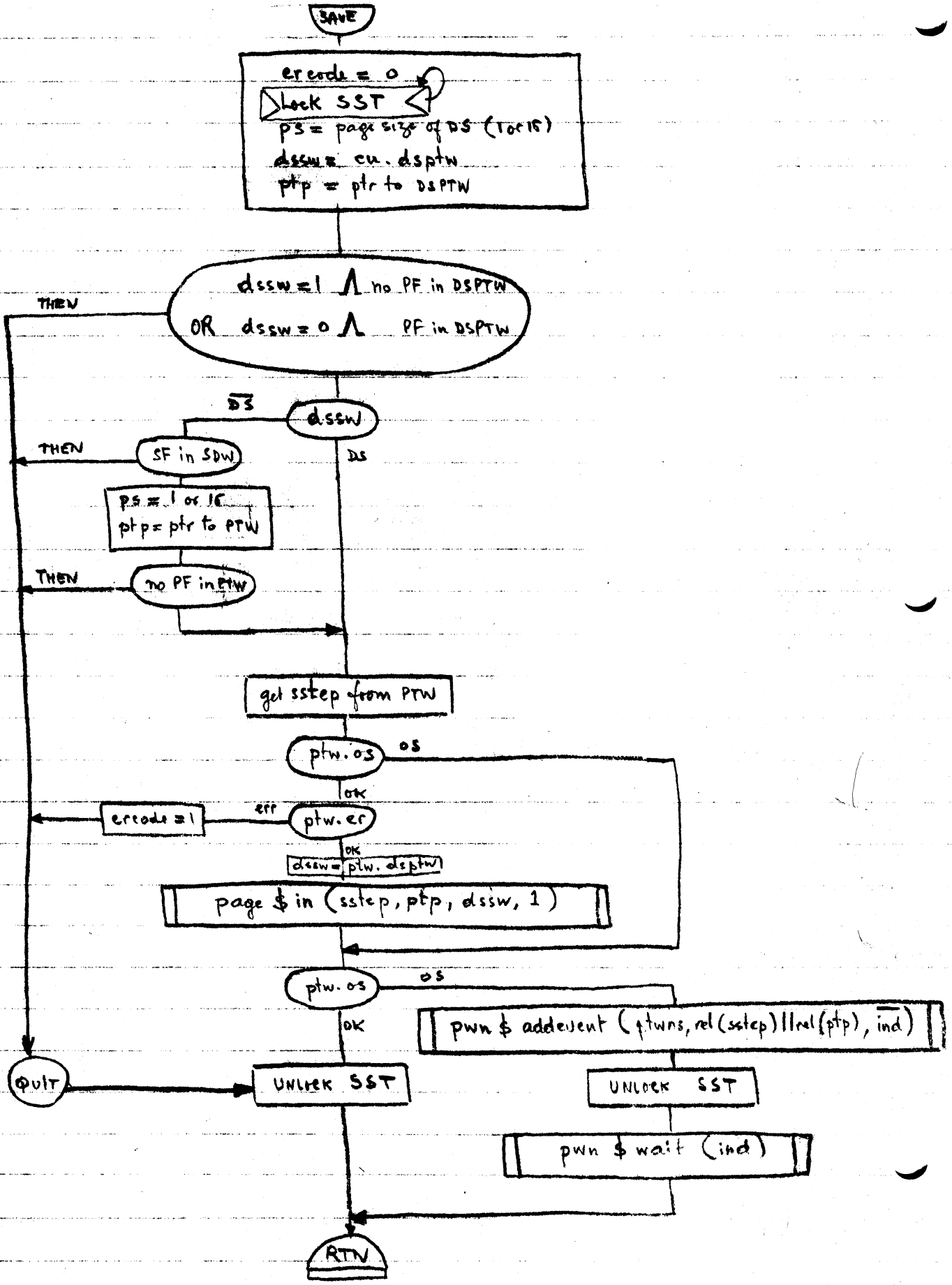
PAGE $ FAULT
$ IN
$ DONE
$ OUT
$ TABLEIN
$ TABLEOUT

PC $ CHECKENTRY
$ CLEAN_UP
$ FREE_CORE
$ READSEG
$ TRUNCATE
$ UNWIRE

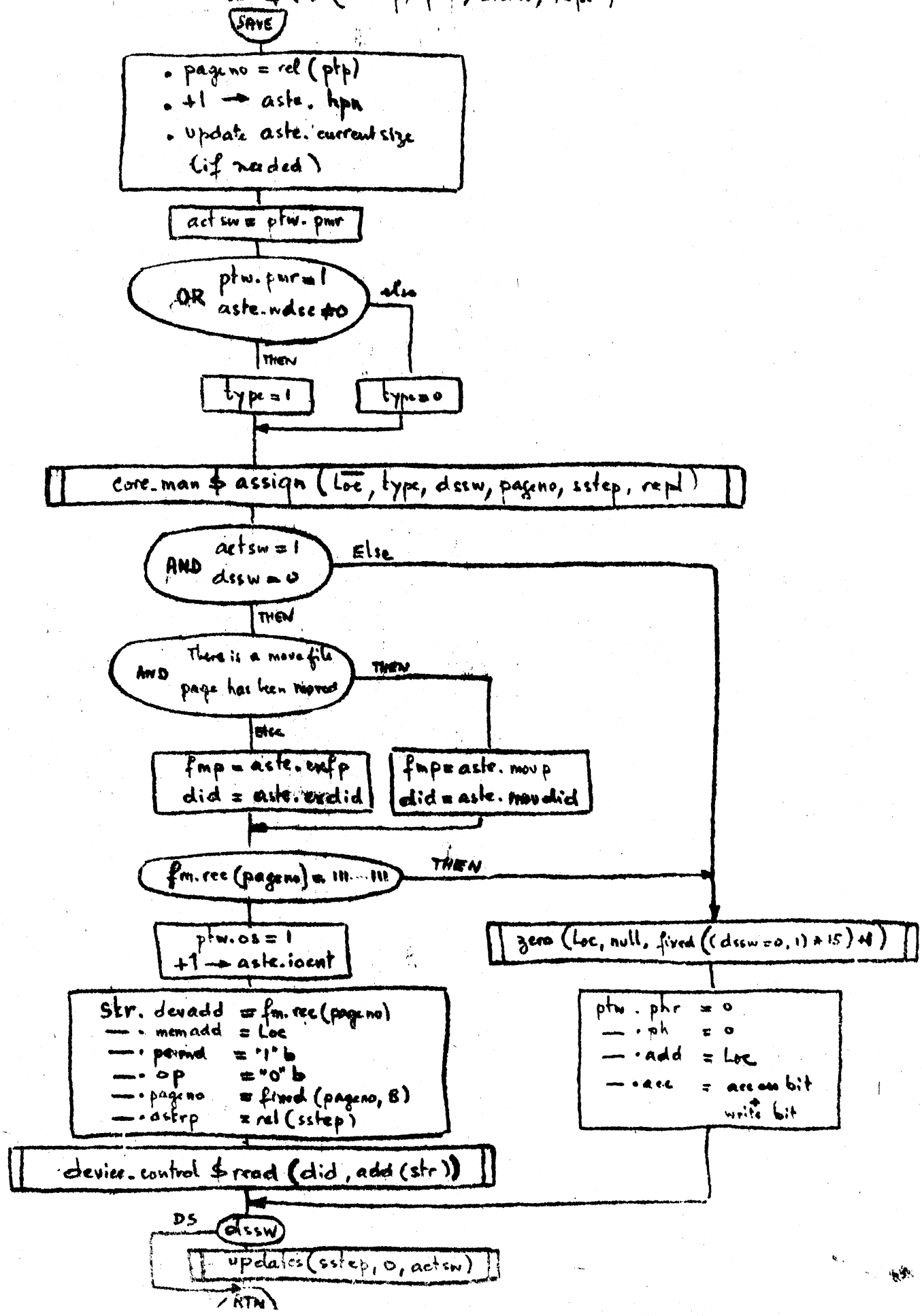
SETFAULTS

UPDATES

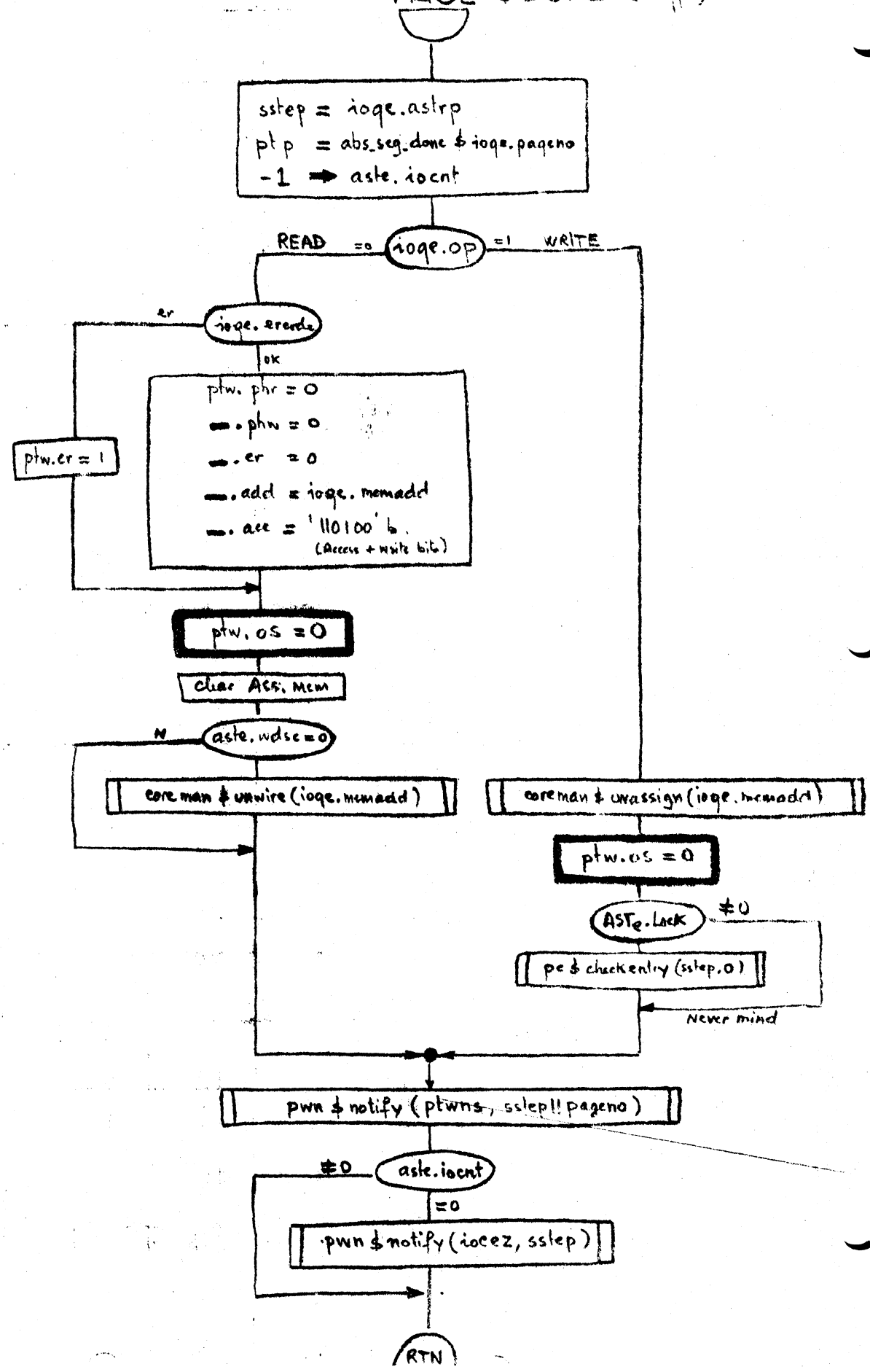
PAGE $ FAULT (scuptr, dbrptr, ercode)



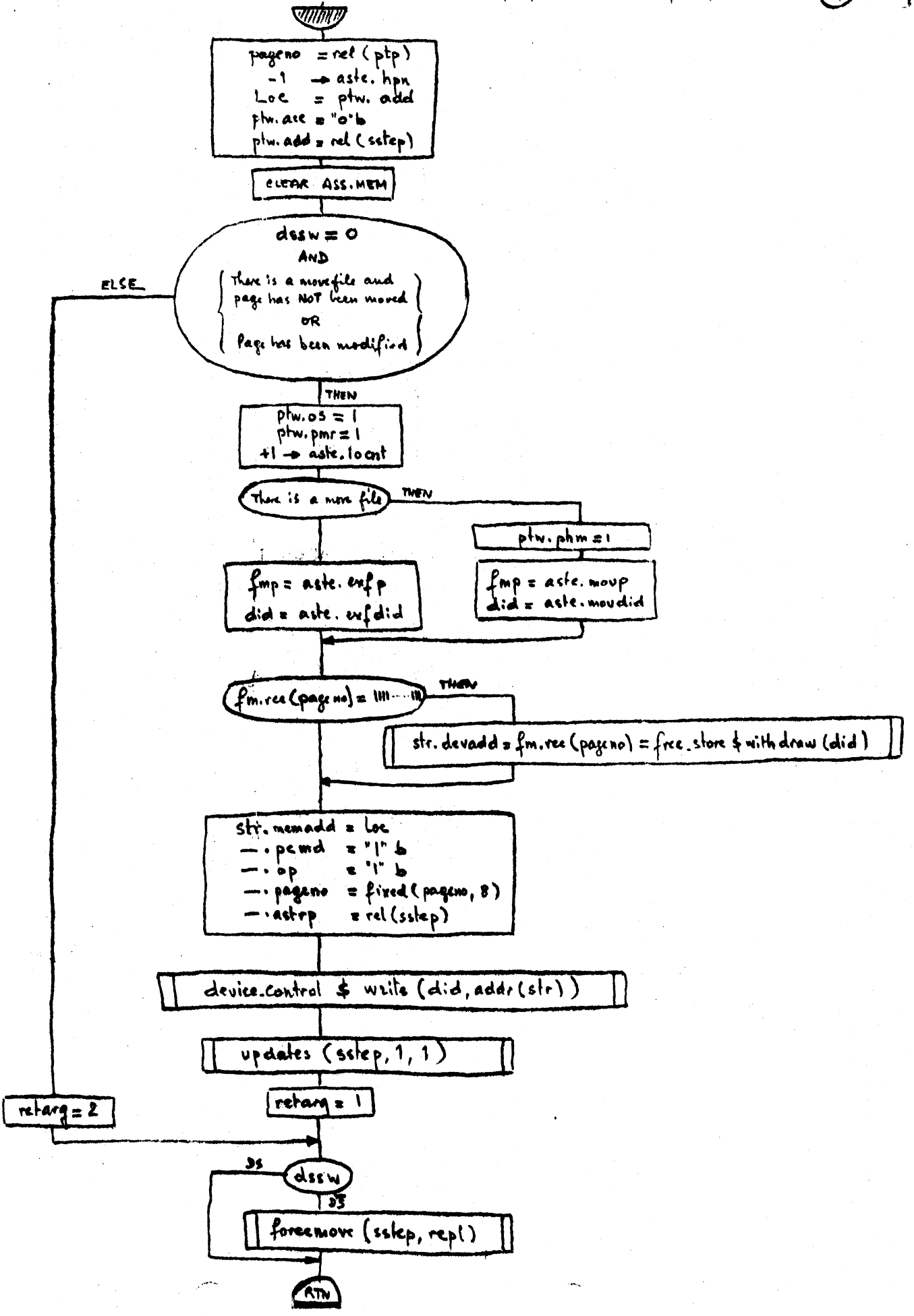
PAGE $in (sstep, ptp, dssw, repl)

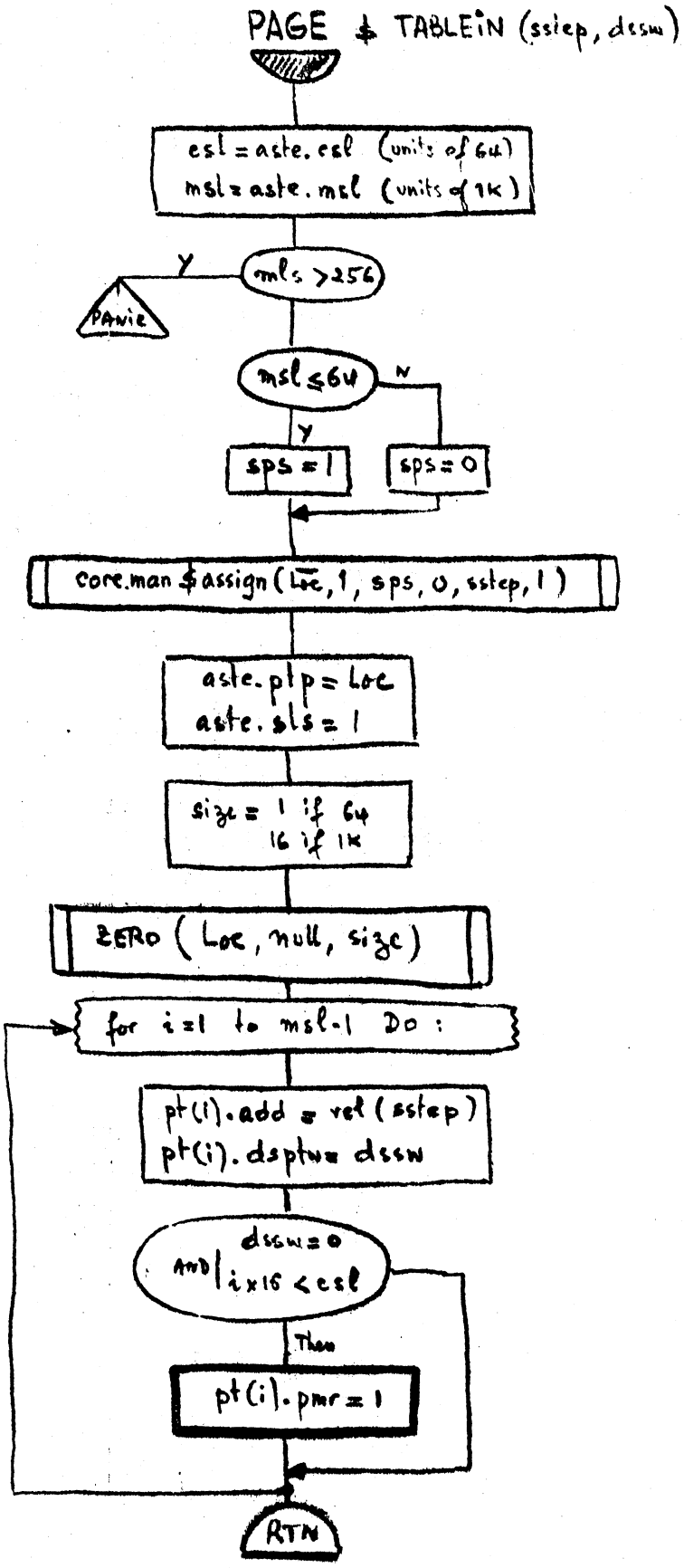


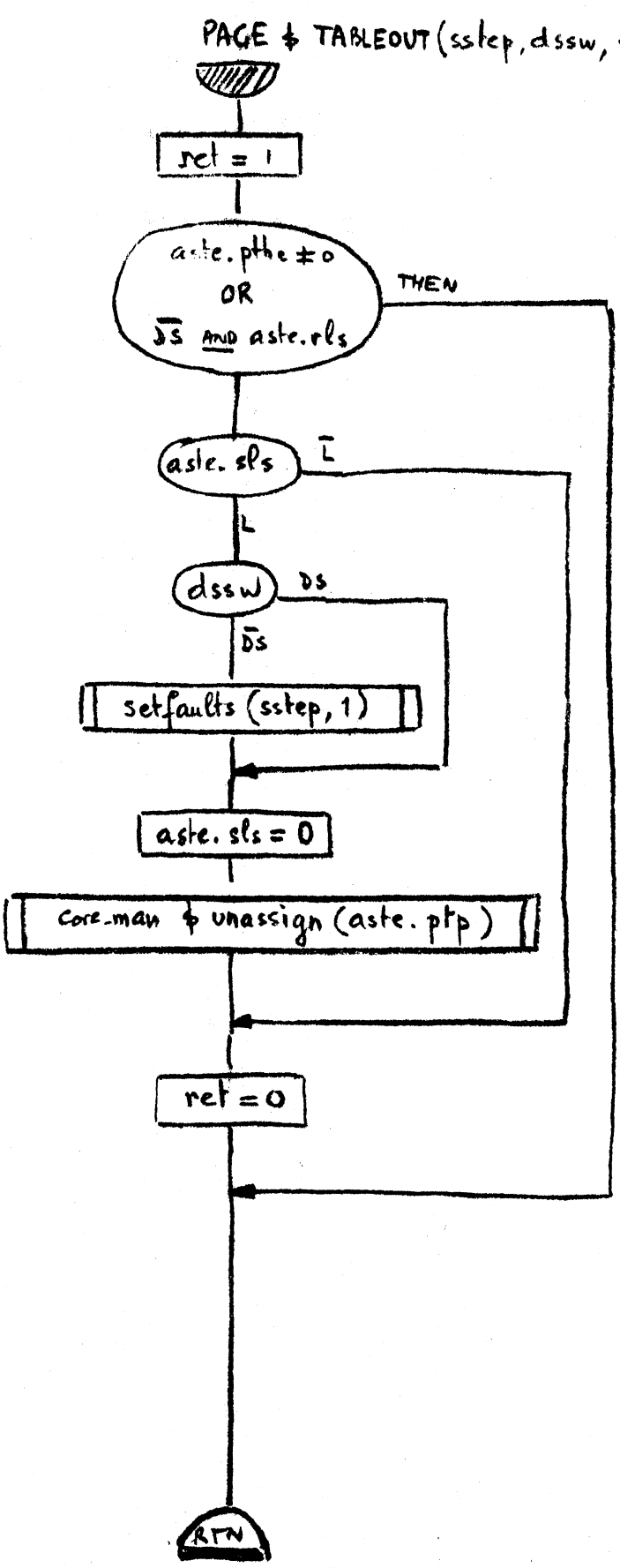
PAGE & DONE (ioqp)



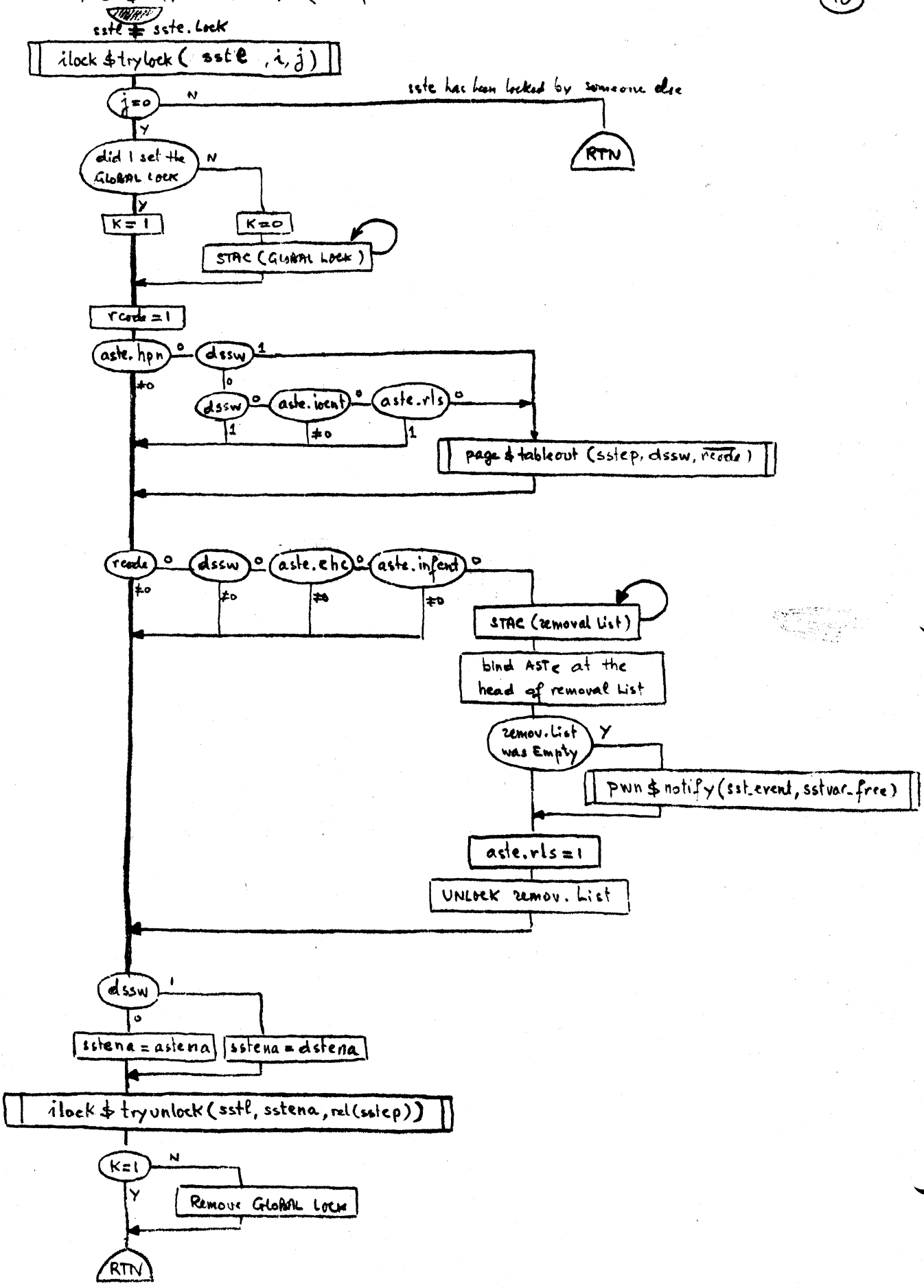
PAGE $ OUT (sstep, ptp, dssw, retarg, repl)



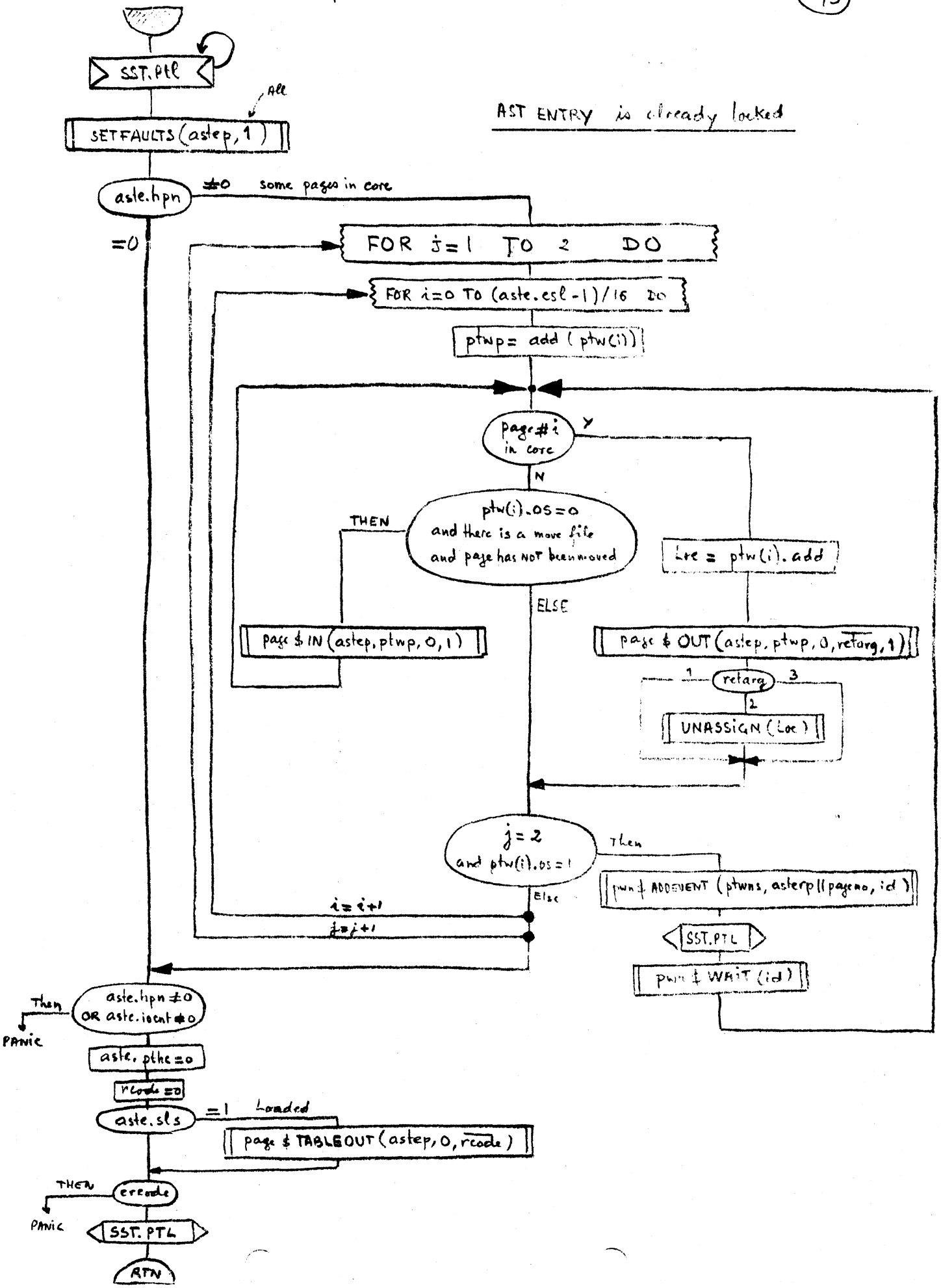

PAGE $ TABLEIN (sstep, dssw)
esl = aste.esl (units of 64)
msl = aste.msl (units of 1K)
msl >256
Y
PANIC
msl ≤64
N
Y
sps = 1
sps = 0
core.man $assign (Loc, 1, sps, 0, sstep, 1)
aste.ptp = Loc
aste.sls = 1
size = 1 if 64
16 if 1K
ZERO (Loc, null, size)
for i=1 to msl-1 Do :
pt(i).add = val (sstep)
pt(i).dsptw = dssw
dssw=0
AND i x16 < esl
Then
pt(i).pmr = 1
RTN


PAGE $ TABLEOUT (sstep, dssw, ret)
ret = 1
aste.pthe ≠ 0
OR
DS AND aste.rls
THEN
aste.sls
L
L
dssw
DS
DS
setfaults (sstep, 1)
aste.sls = 0
core_man $ unassign (aste.ptp)
ret = 0
RTN

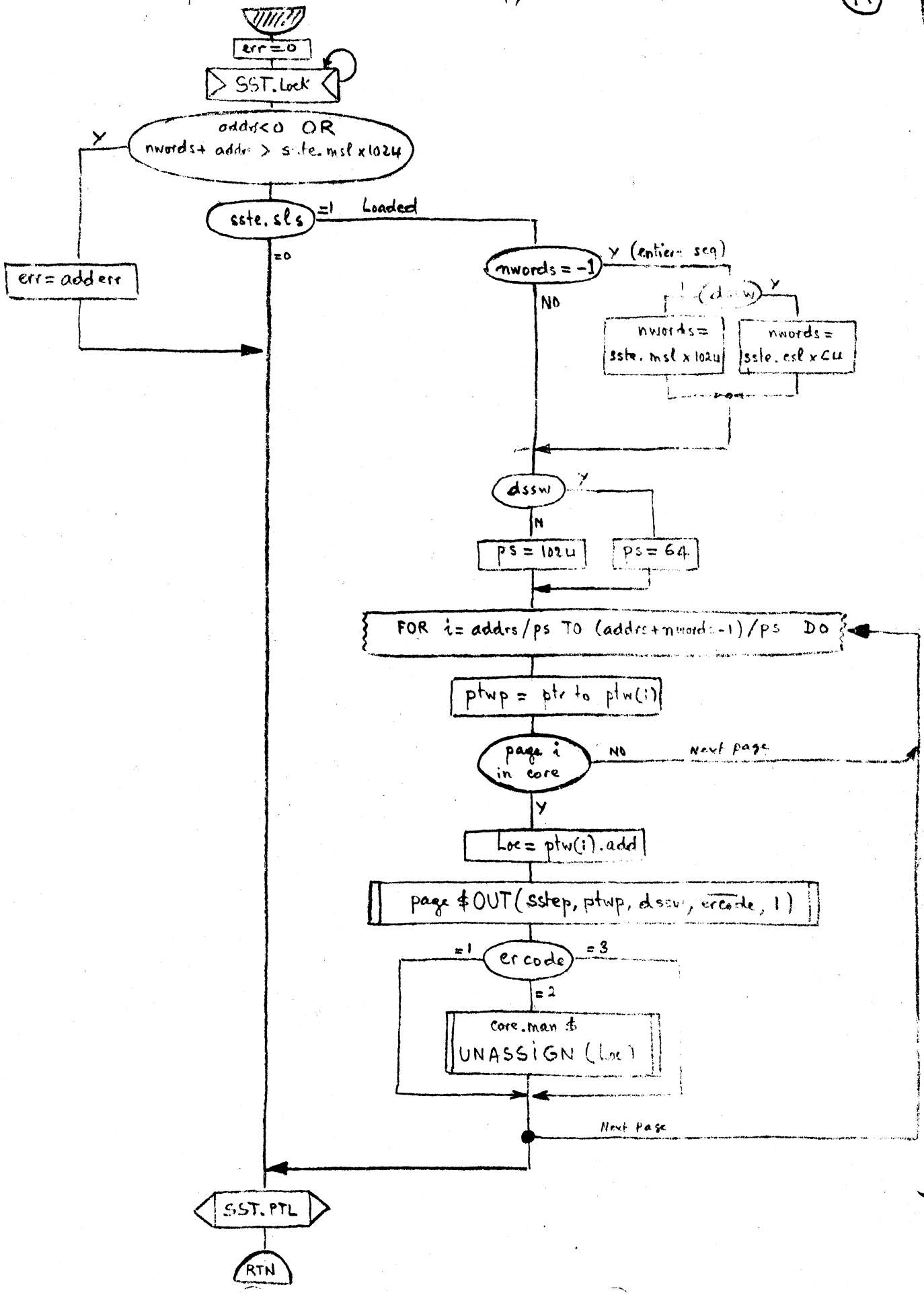
PC $ CHECKENTRY (sstep, dssw)



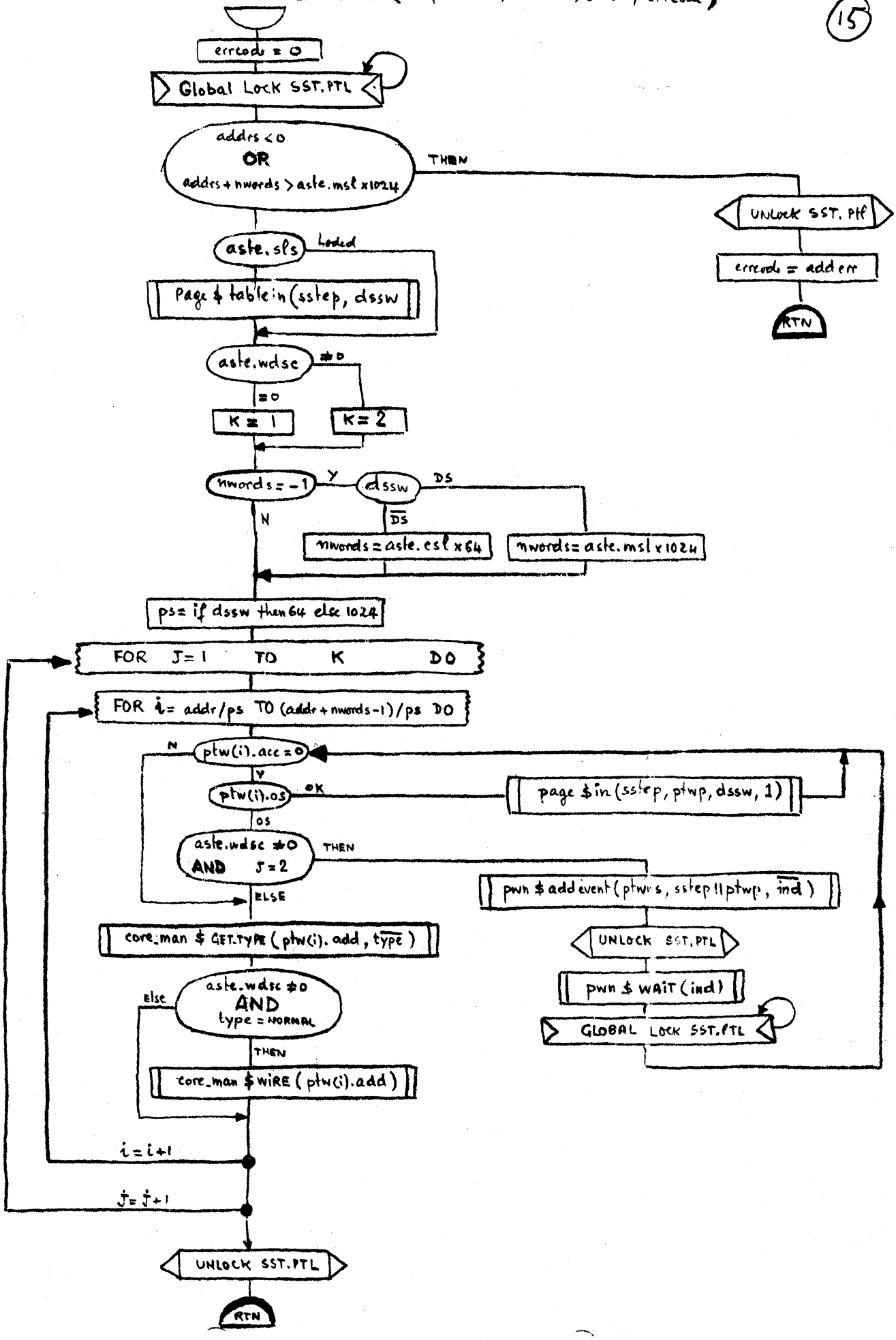
pc $ CLEAN_UP (astep)



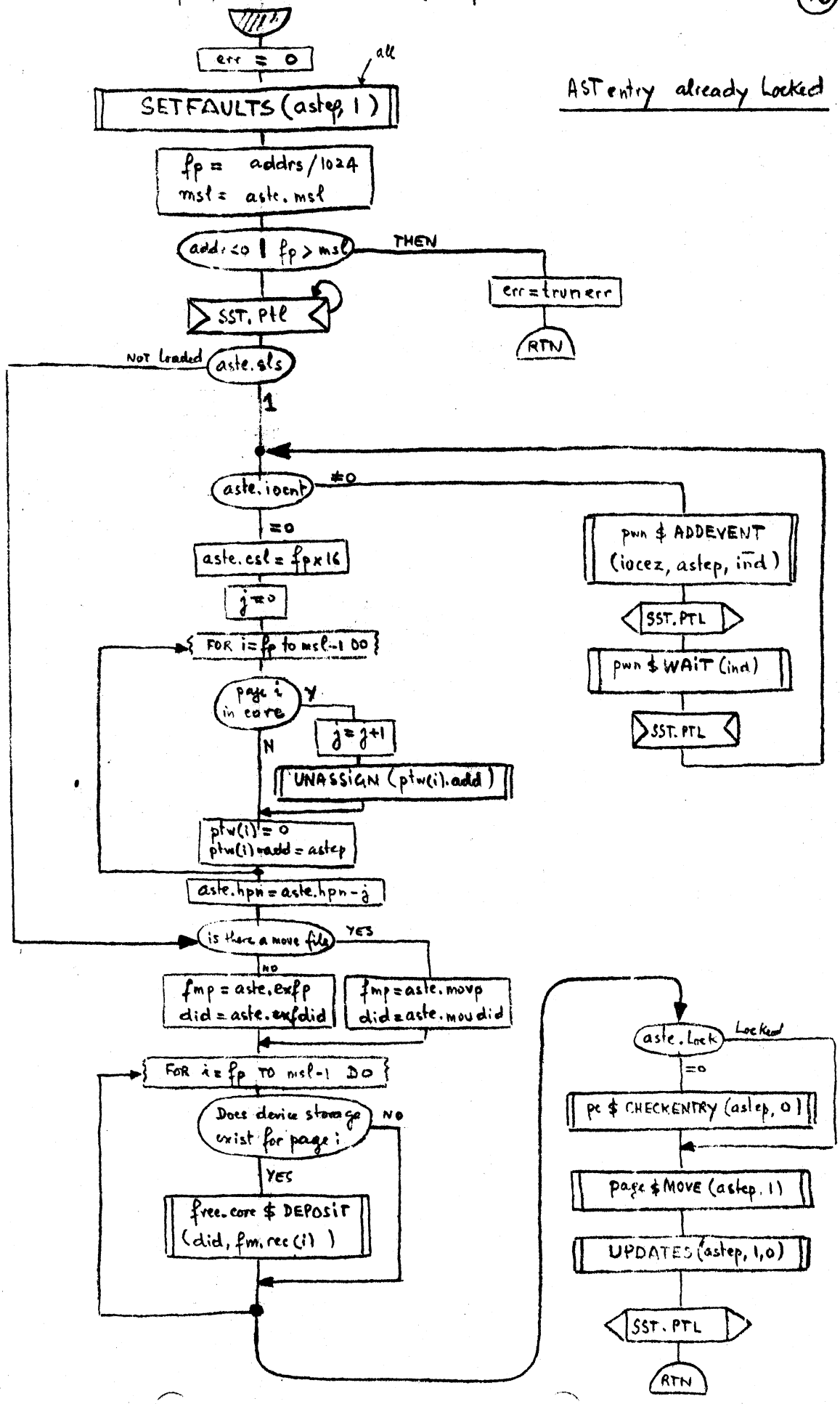
pc $FREE CORE (sstep, addrs, nwords, dssw, err)



PC $ READSEG (sstep, addrs, nwords, dssw, errcode)

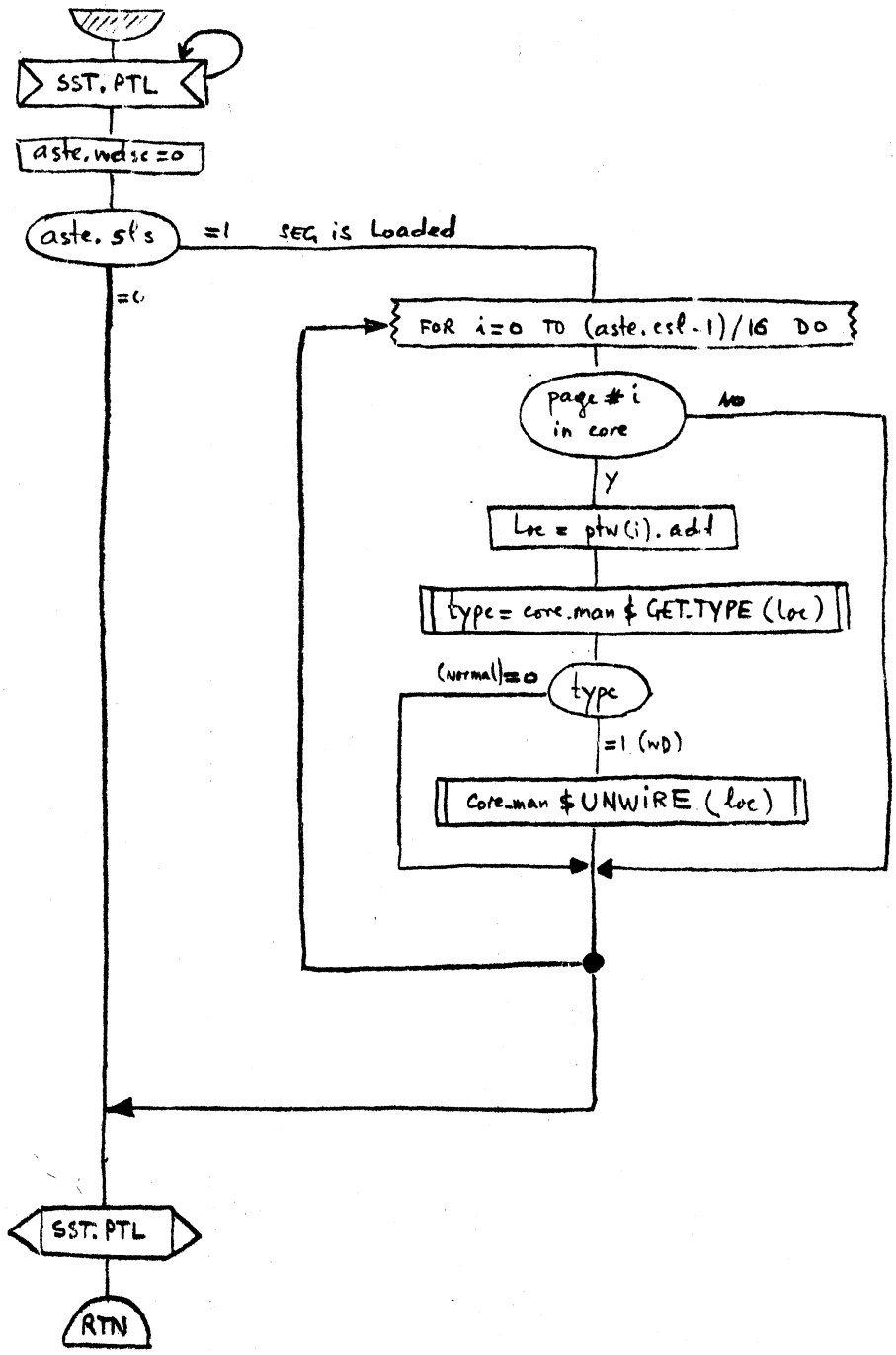


pc$ TRUNCATE (astep, addrs, err)



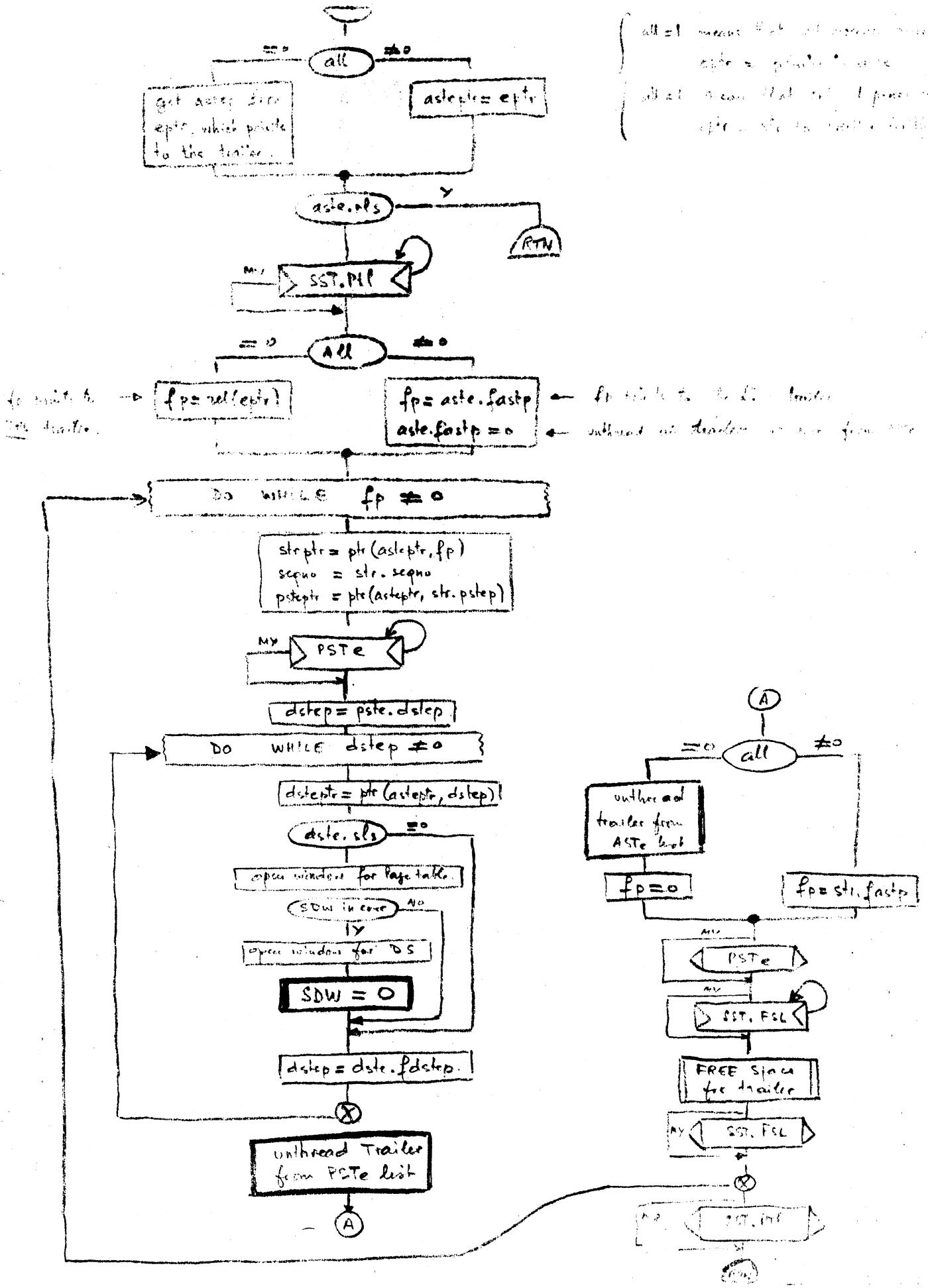
AST entry already Locked

pc $ UNWIRE (astep)



Called by SET UP RING to unwire the first page of the ring n DS when a process leaves ring n.

SETFAULTS (eptr, all)



SEGMENT CONTROL

SEGMENT UTILITY MODULE

ALLOC_SST $ ALLOC_SST
$ FREE_SST
ALLOCATE_SSTVAR $ ALLOCATE_SSTVAR
$ FREE_SSTVAR
CMP_ACC
DST_SEARCH
DST_THREAD
GETASTENTRY $ GETASTENTRY
$ DELASTENTRY
HASH_INDEX $ ID INDEX
$ NAME INDEX
MAKETRAILER
REMOVAL_LIST_UTIL $ SEARCH
$ TEST
$ UNTHREAD
SUM $ ID SRCHKST
$ N SRCH KST
$ SEARCHAST
$ SEARCH HST
THREADTRAILER

SYSTEM INTERFACE MODULE

BOUNDFAULT
GETRING
INITIALIZE_KST
MAKE KNOWN
MAKE UNKNOWN
SEGFAULT
SIM1 $ BRANCHMOD
$ DELETE SEG
$ DIRMOD
$ TRANSUSE
$ UNLOAD SEG
$ UPDATE B
SIM2 $ GETDIRSEG
$ MOVE SEG

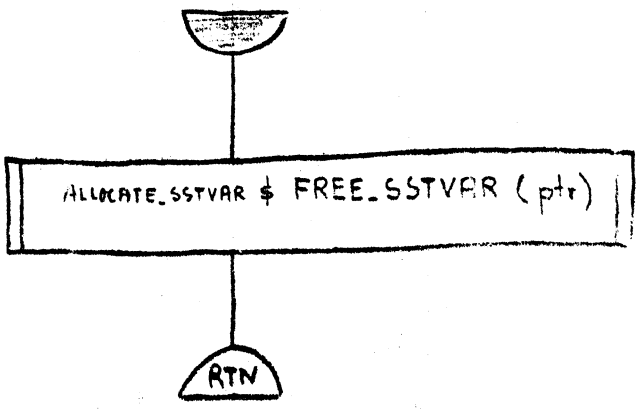
USER INTERFACE MODULE

uim $ CHECK ACCESS
$ CHECK RING
$ CORE TEST
$ FREE CORE
$ READ SEG
$ TRUNCATE SEG
$ WRITE SEG

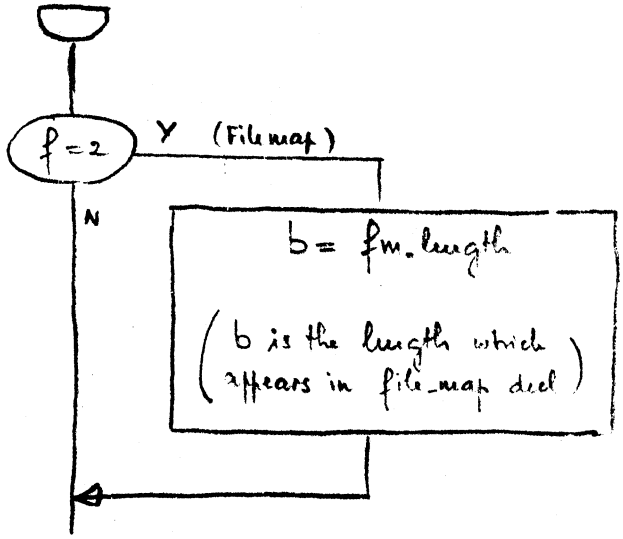
RING REGISTER SIMUL MODULE

SET UP_RING $ LOAD
$ SET UP_RING

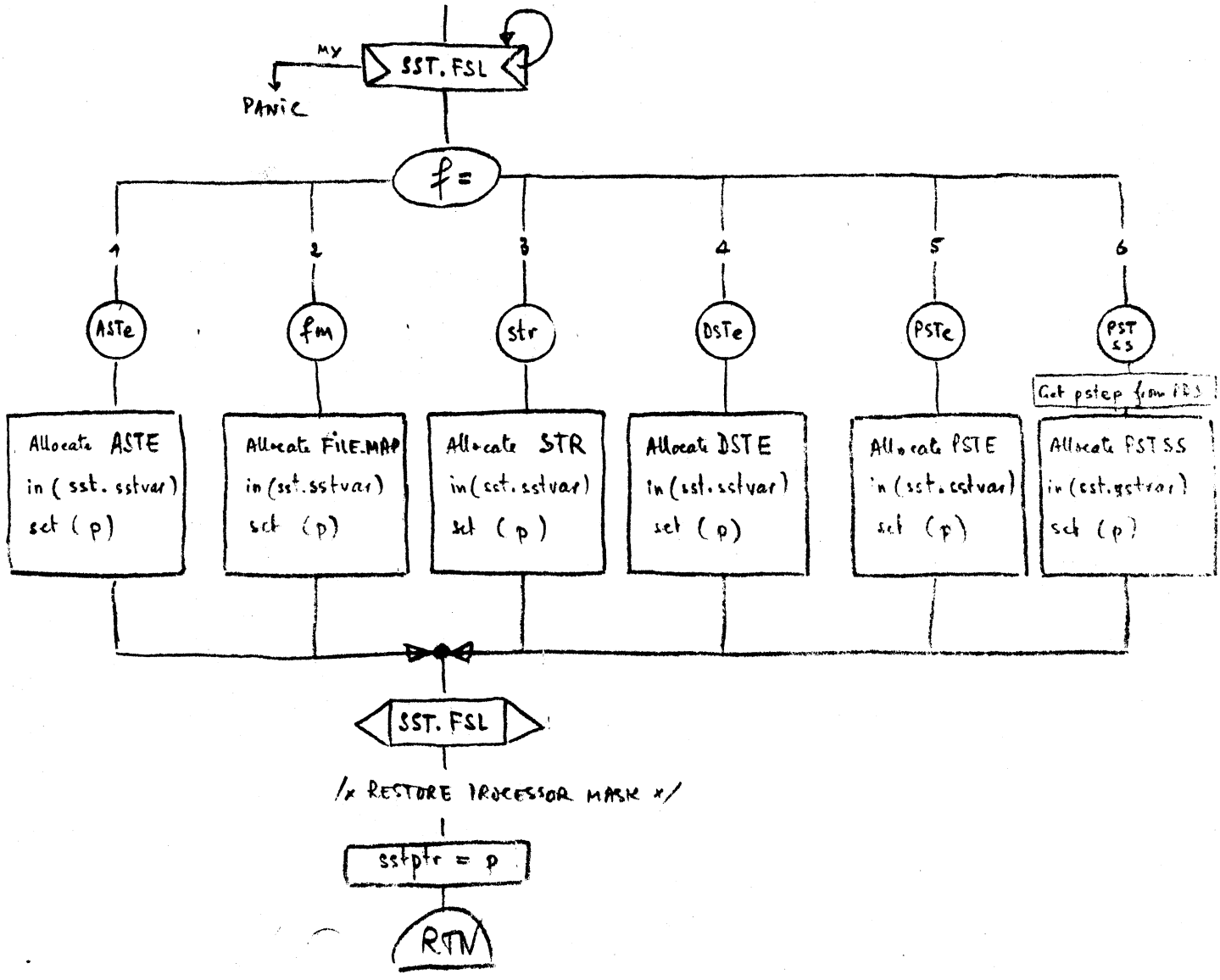
ALLOC_SST $ FREE_SST (ptr)



Allocate sstvar $ ALLOCATE_SSTVAR (sstptr, f, fm_length).



/* MASK PROCESSOR */

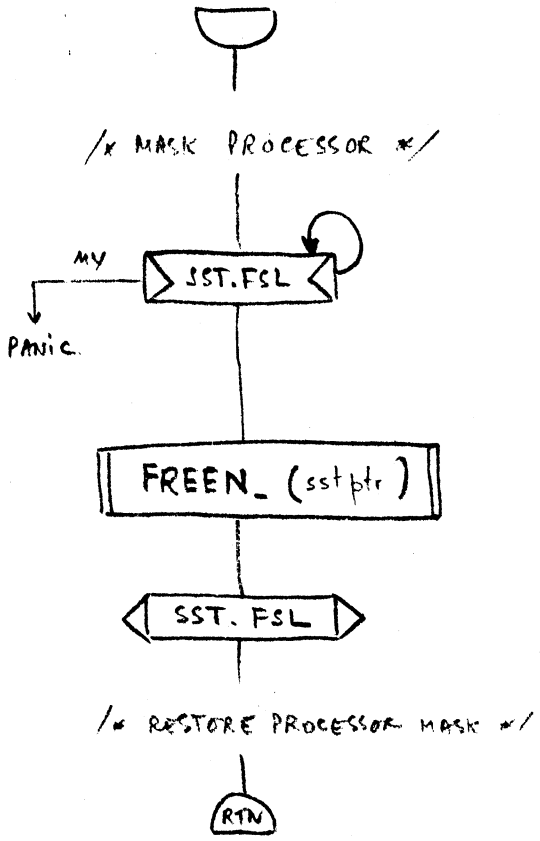


/* RESTORE PROCESSOR MASK */

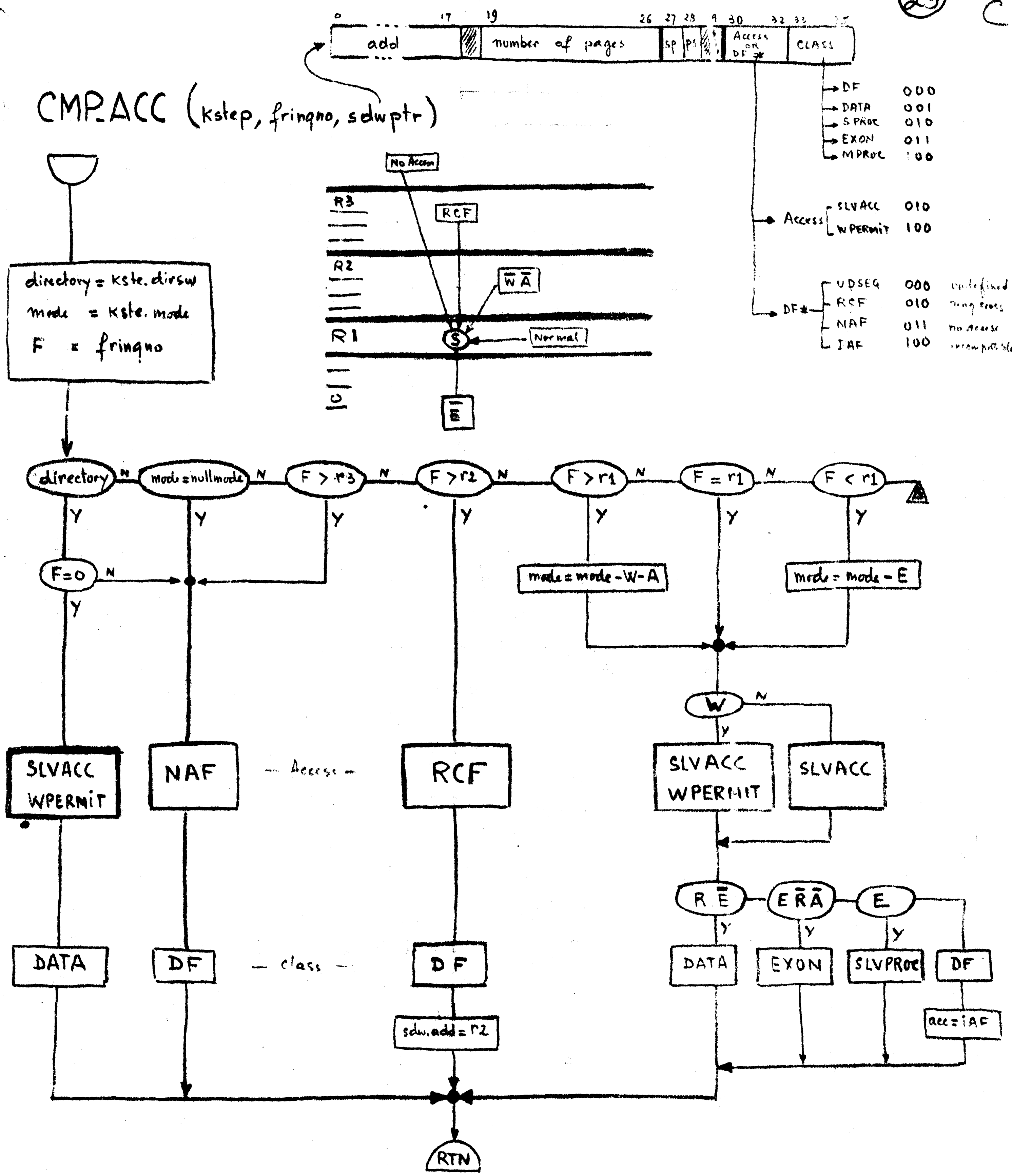
sstptr = p

RTN

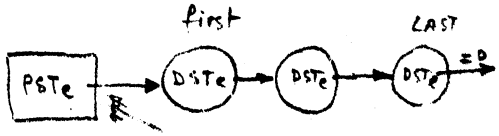
Allocate sstvar & FREE_SSTVAR (sstptr)

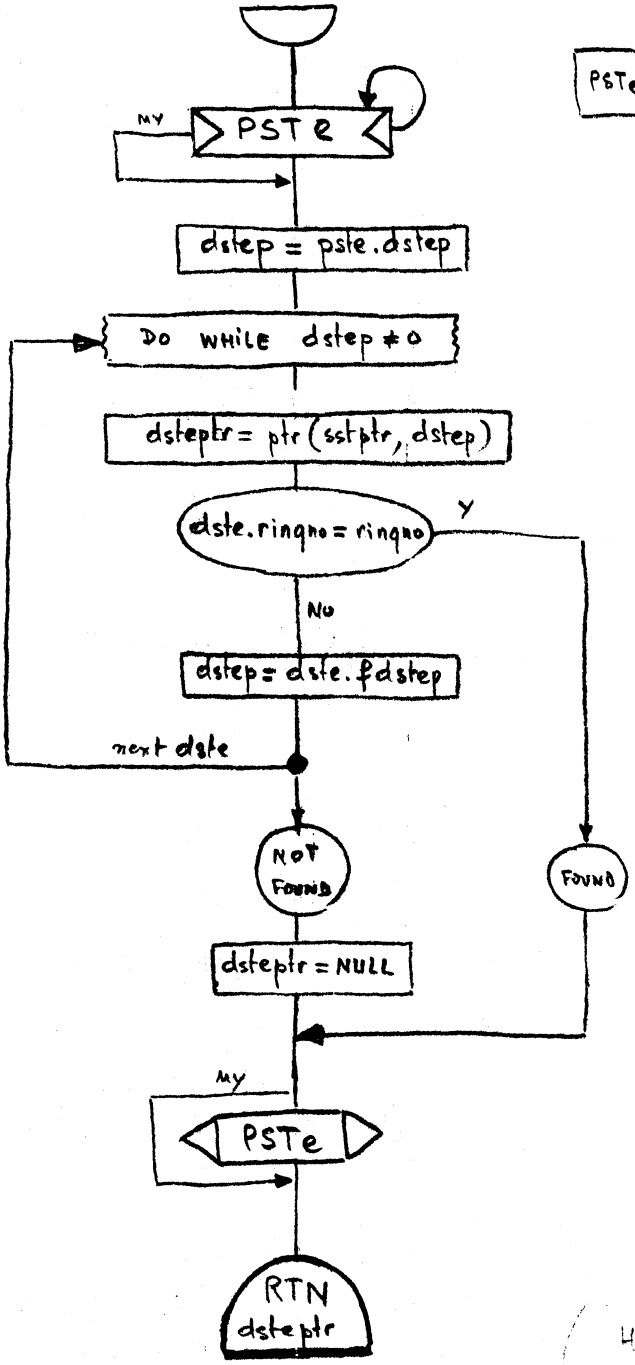


CMP_ACC (kstep, fringno, sdwptr)



DST_SEARCH (ringno, psteptr)

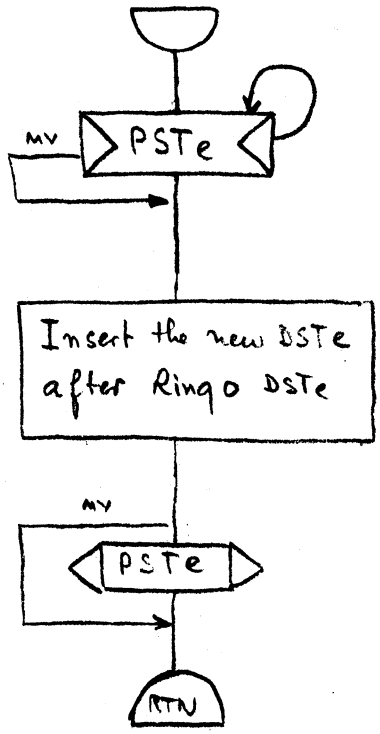


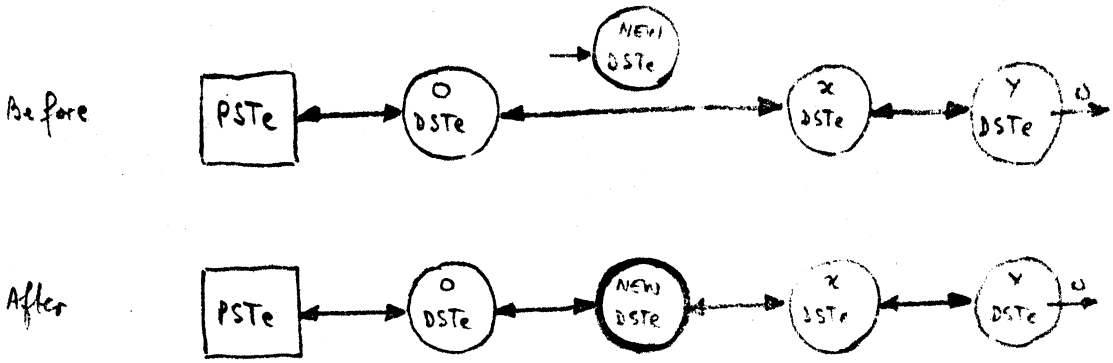


How delete DST e
and

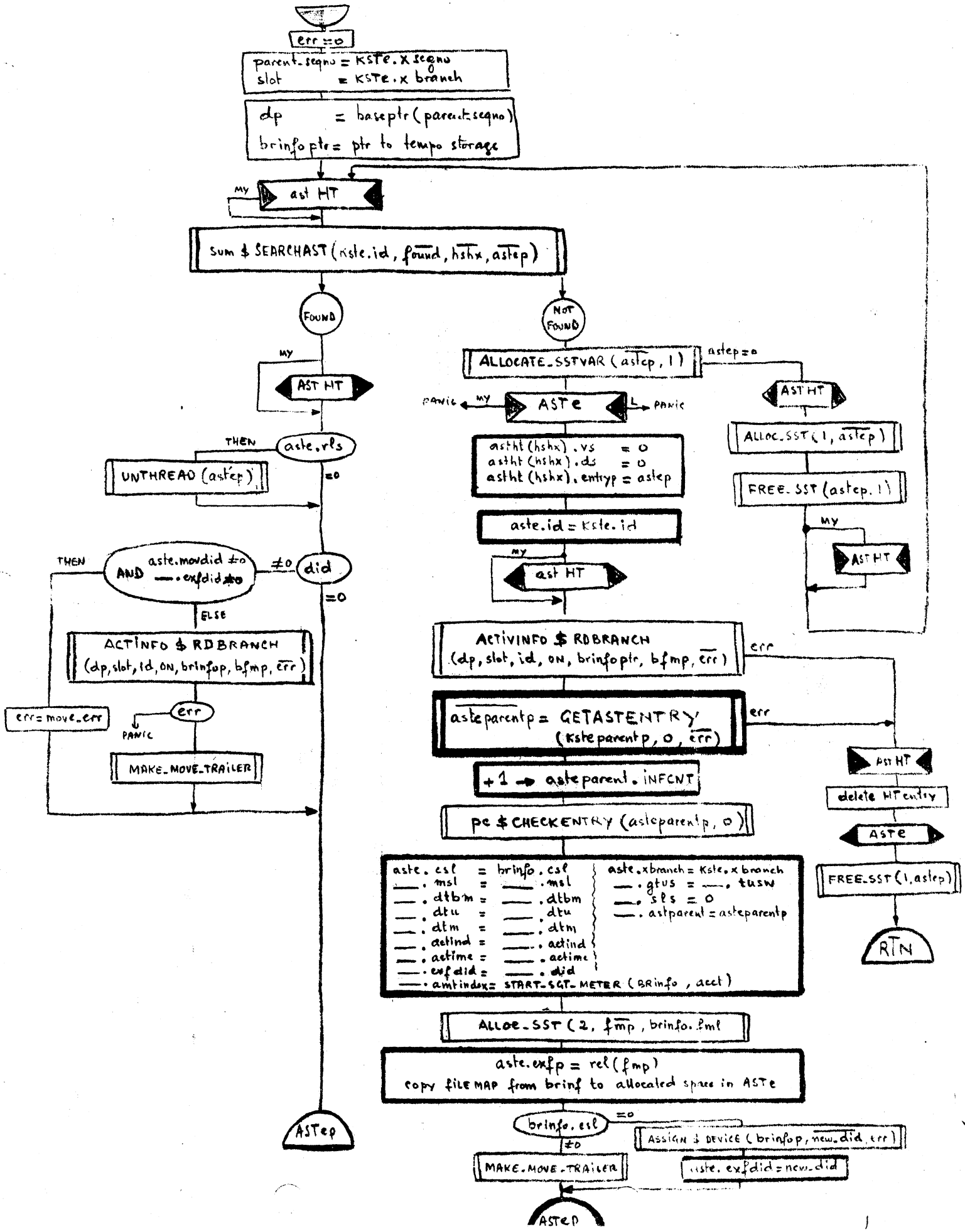
D

DST_THREAD (dsteptr, psteptr)

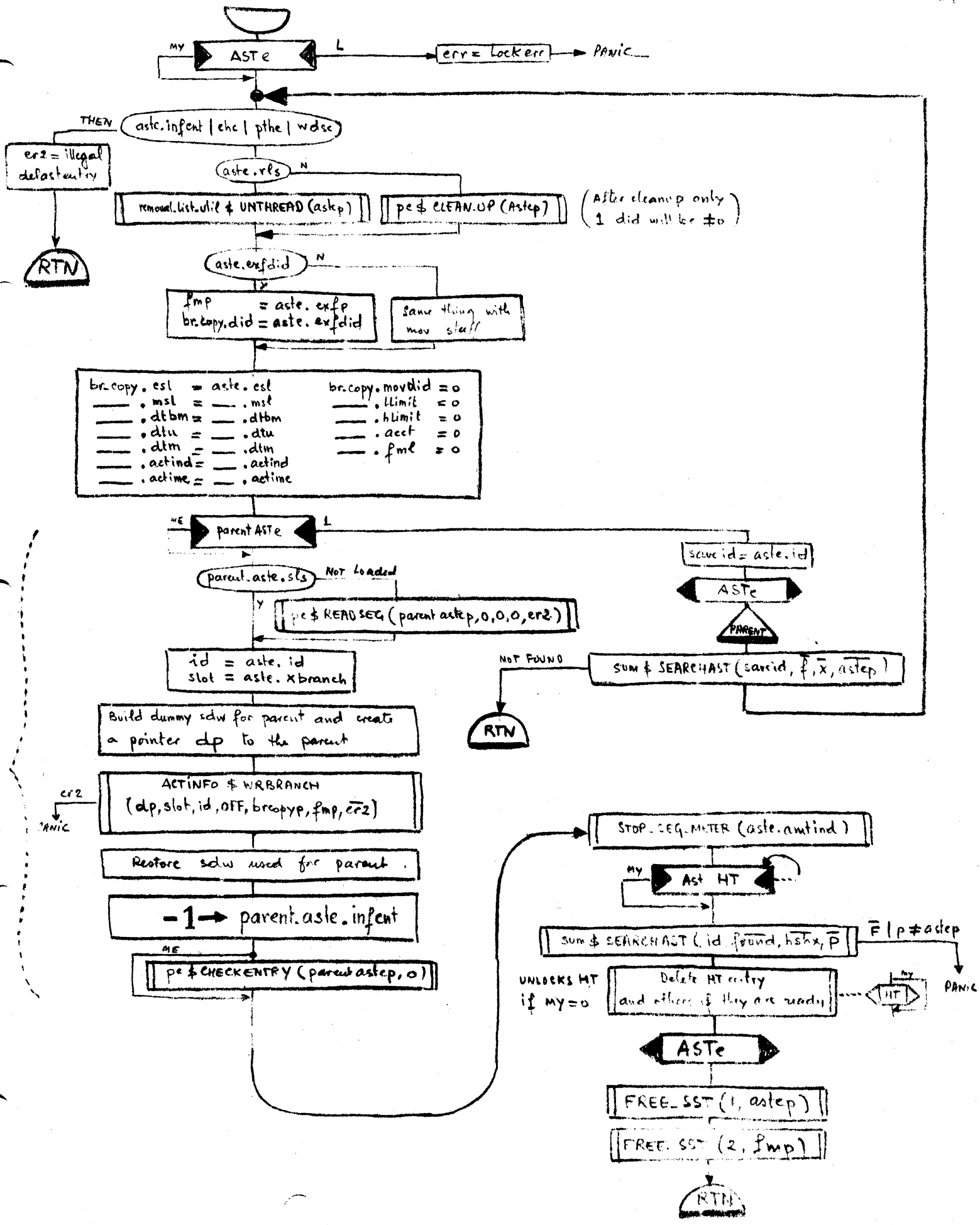




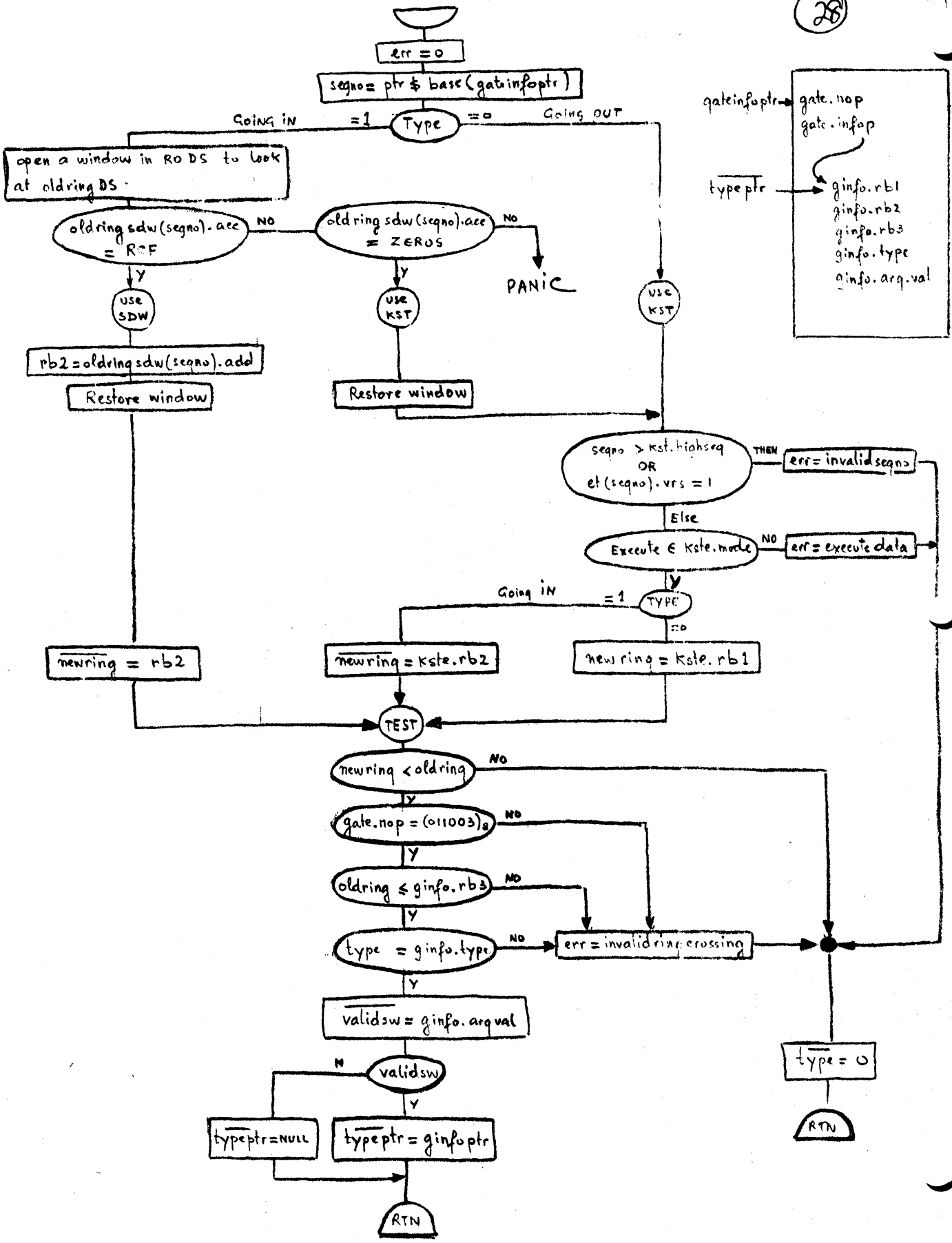
GETASTENTRY (Kstep, did, err)

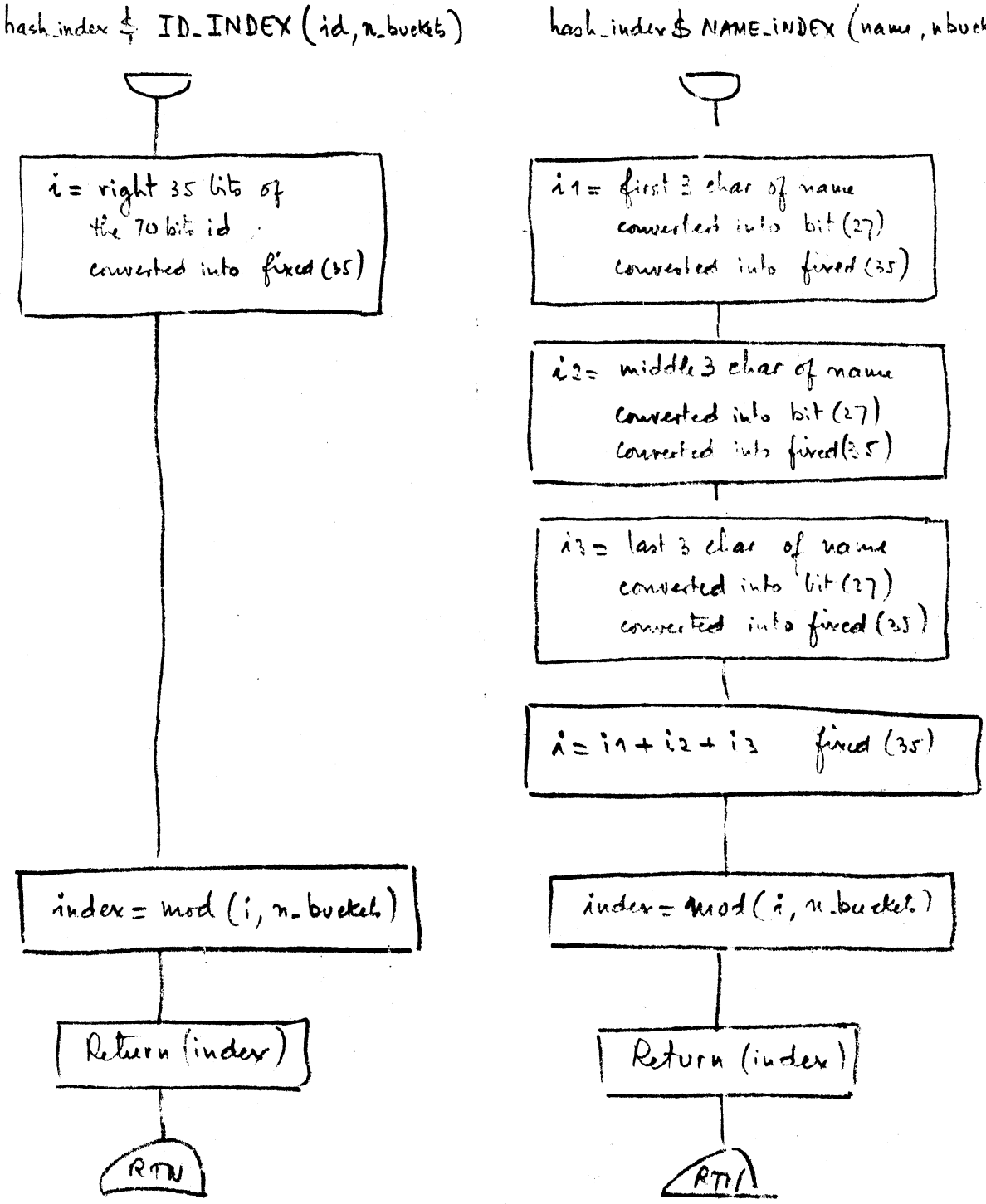


DELASTENTRY (astep, er2)

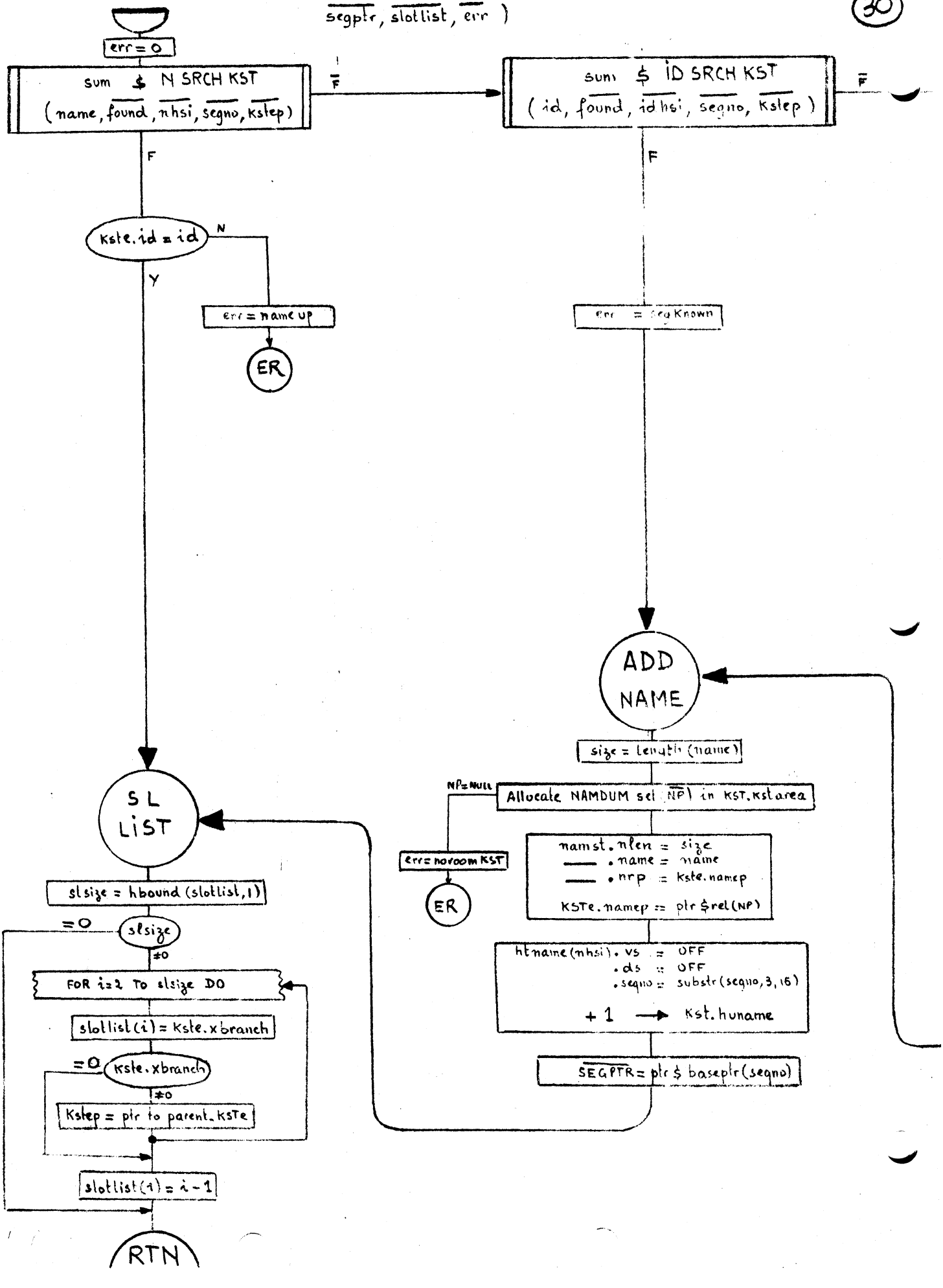


GETRING (gateinfoptr, oldring, newring, type, validsw, typeptr, err)

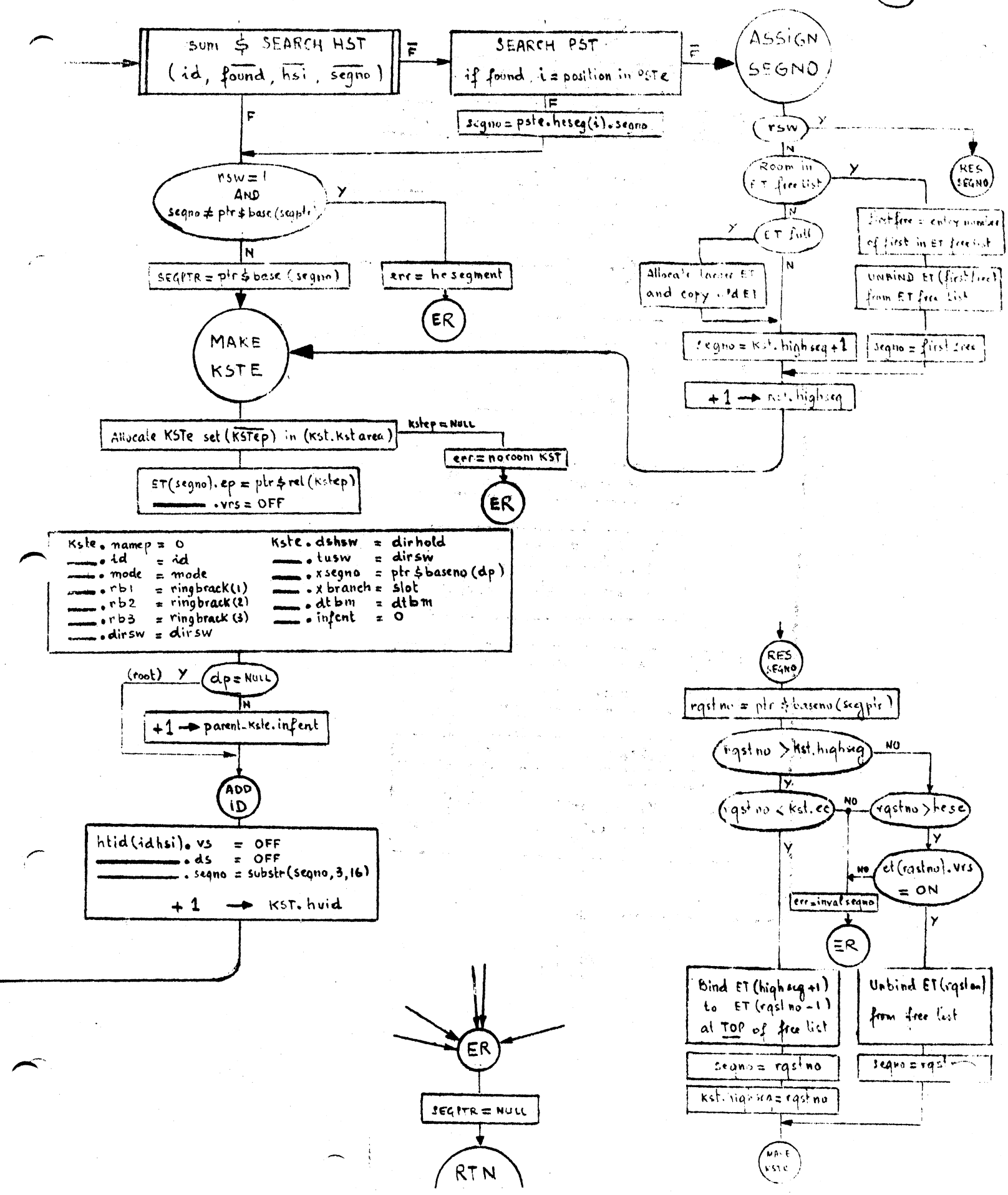



hash_index $ ID_INDEX (id, n_buckets)
i = right 35 bits of the 70 bit id converted into fixed (35)
index = mod (i, n_buckets)
Return (index)
RTN
hash_index $ NAME_INDEX (name, nbuckets)
i1 = first 3 char of name converted into bit (27) converted into fixed (35)
i2 = middle 3 char of name converted into bit (27) converted into fixed (35)
i3 = last 3 char of name converted into bit (27) converted into fixed (35)
i = i1 + i2 + i3 fixed (35)
index = mod (i, n_buckets)
Return (index)
RTN

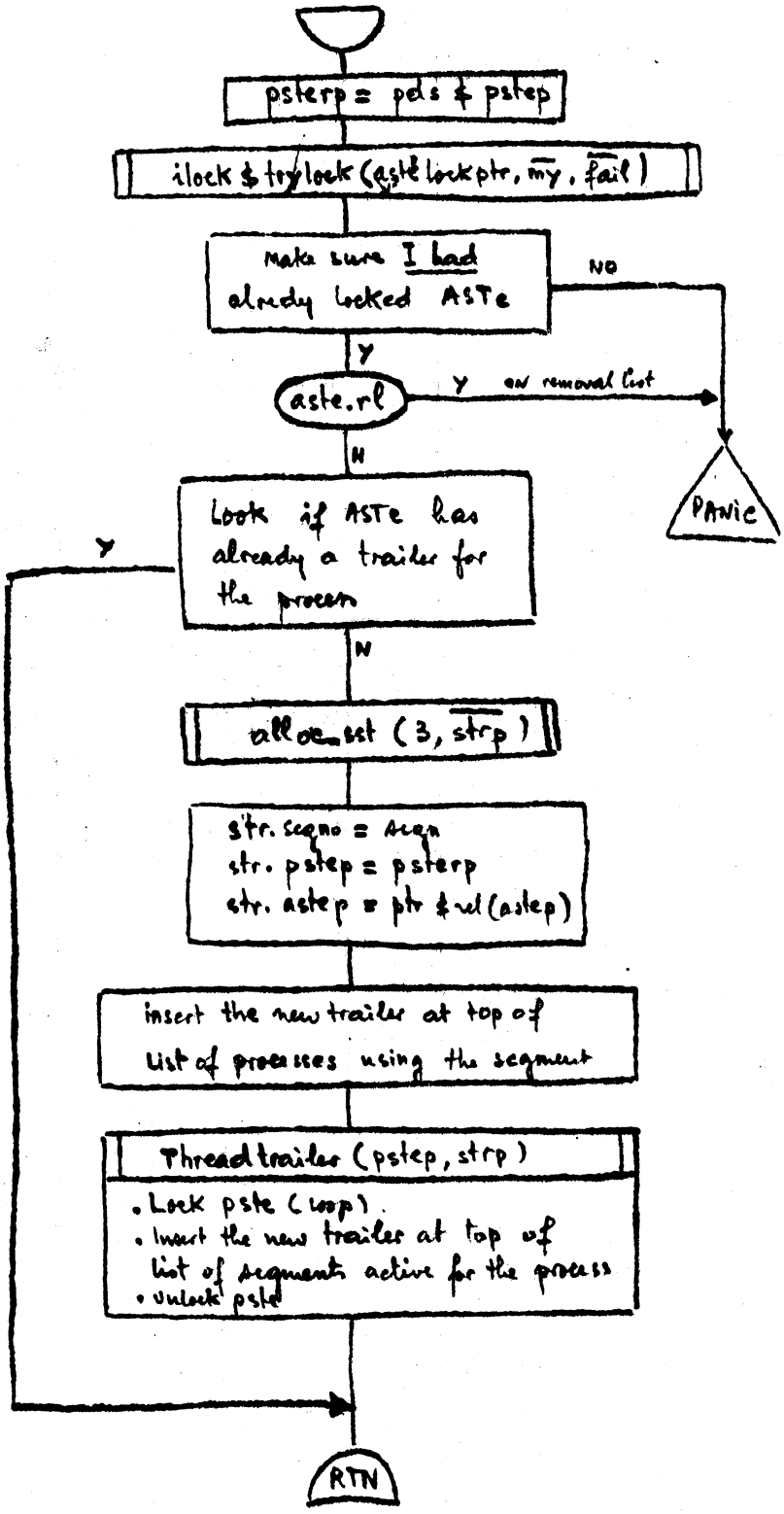
MAKE KNOWN (name, id, mode, ringbrack, dirsw, dtbm, dp, slot, dirhold, rsw, segptr, slotlist, err)



MAKEKNOWN (continued)



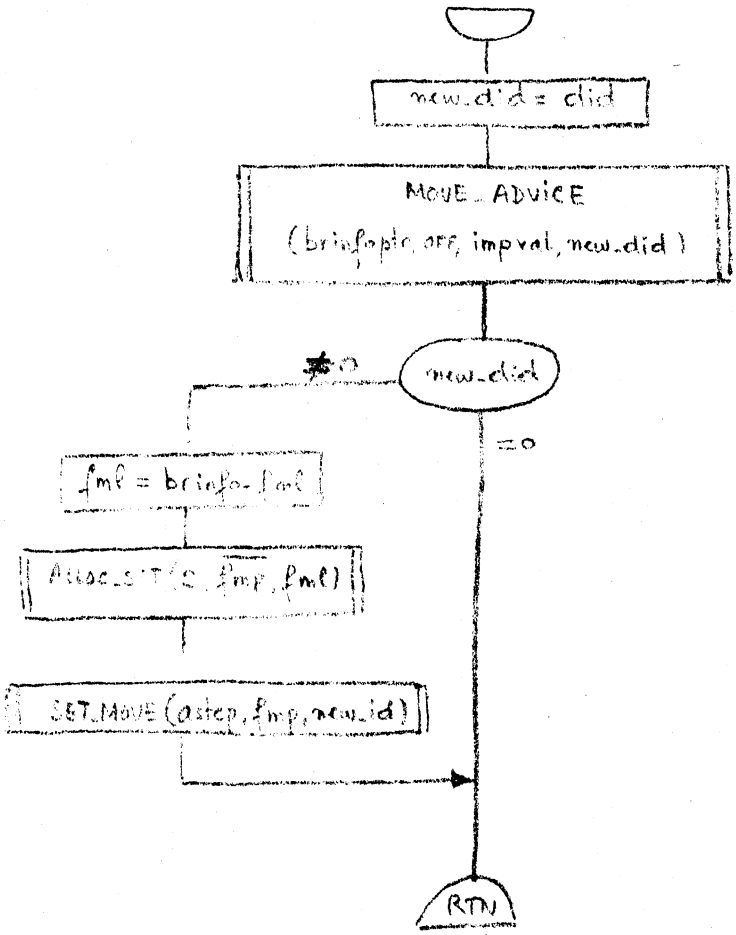
MAKE TRAILER (segno, astep)



MAKE_MOVE_TRAILER

internal to GETASTENTRY

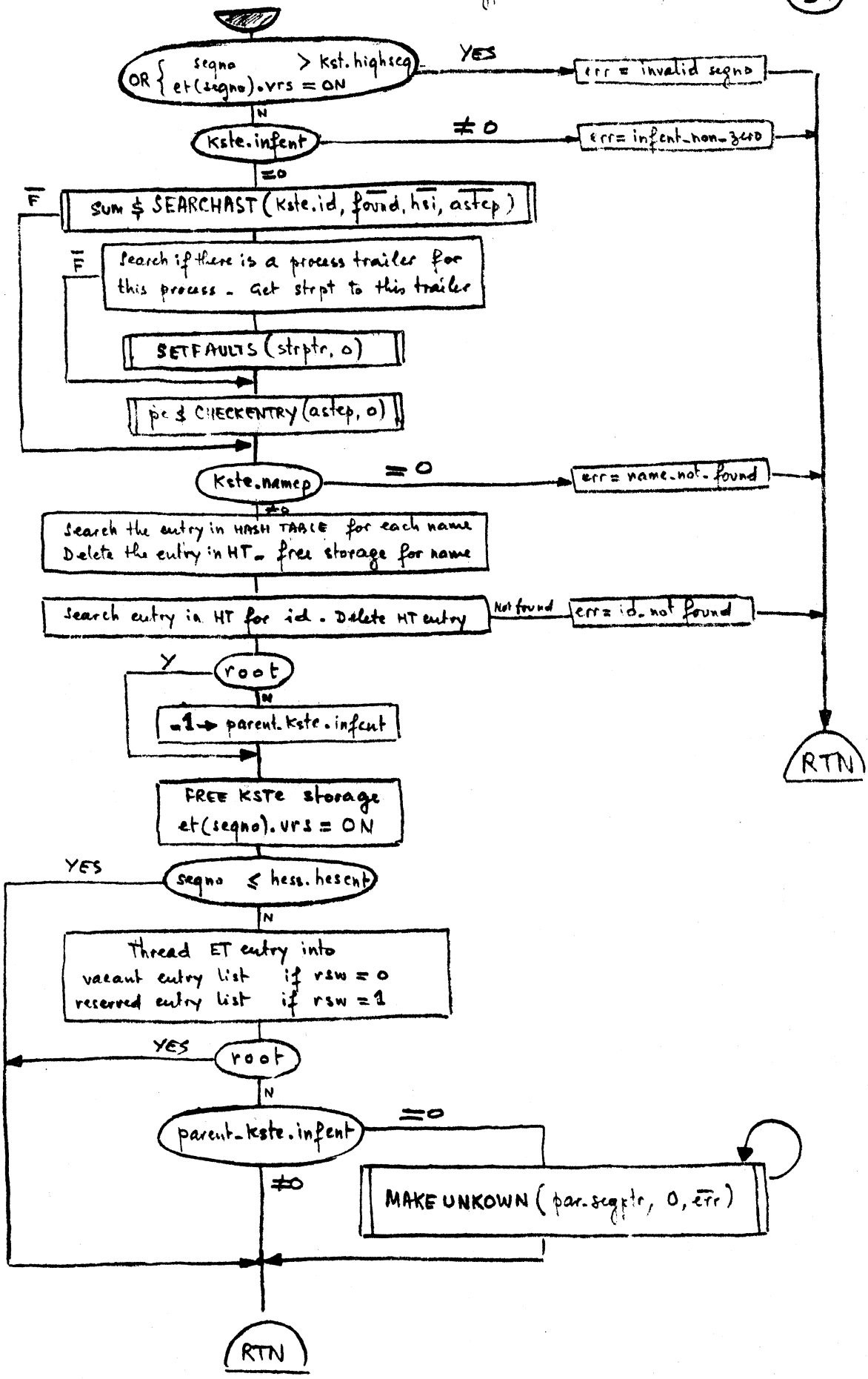
(53)



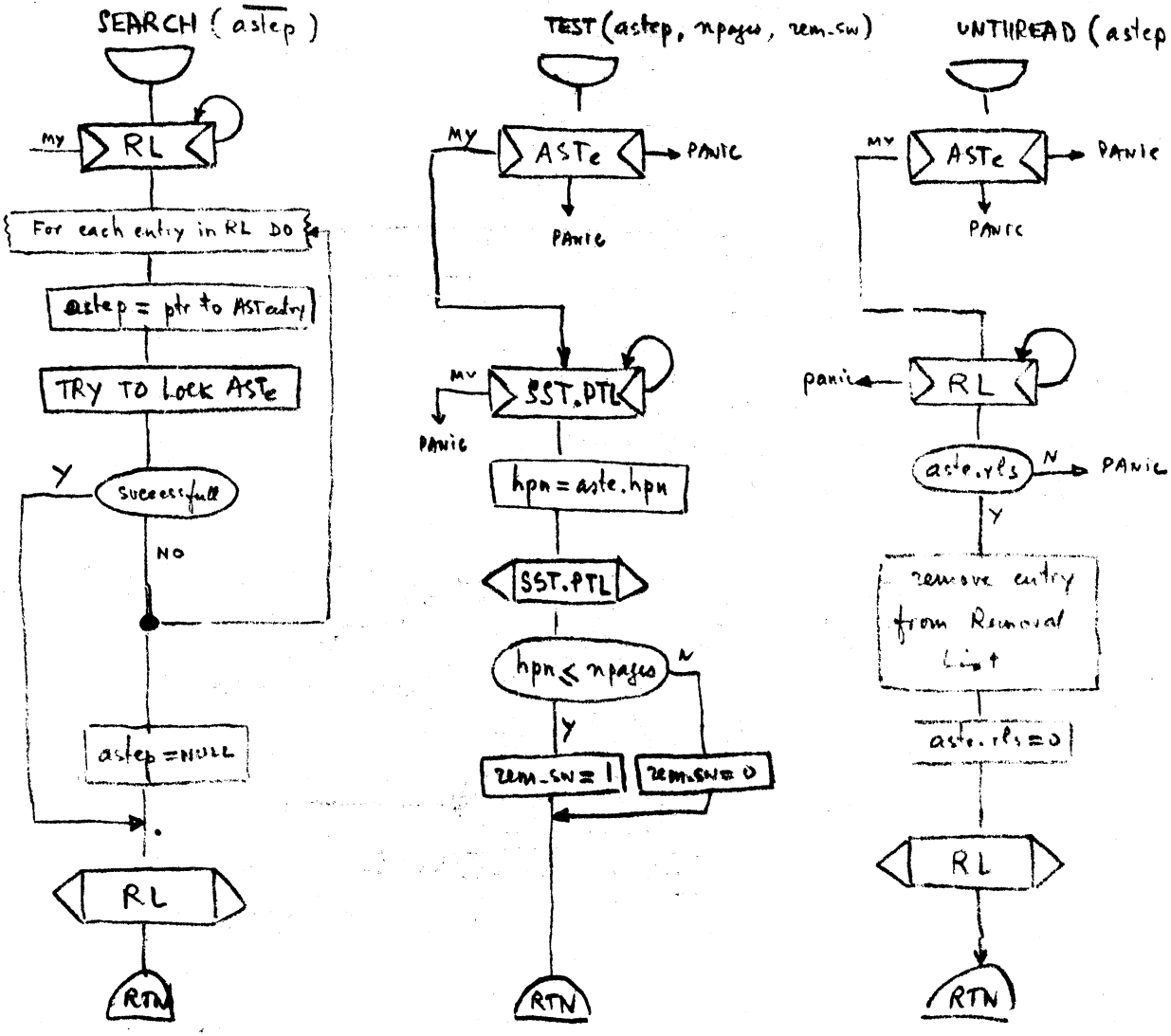
did is argument passed to [illegible]

brinfoptr points to [illegible]

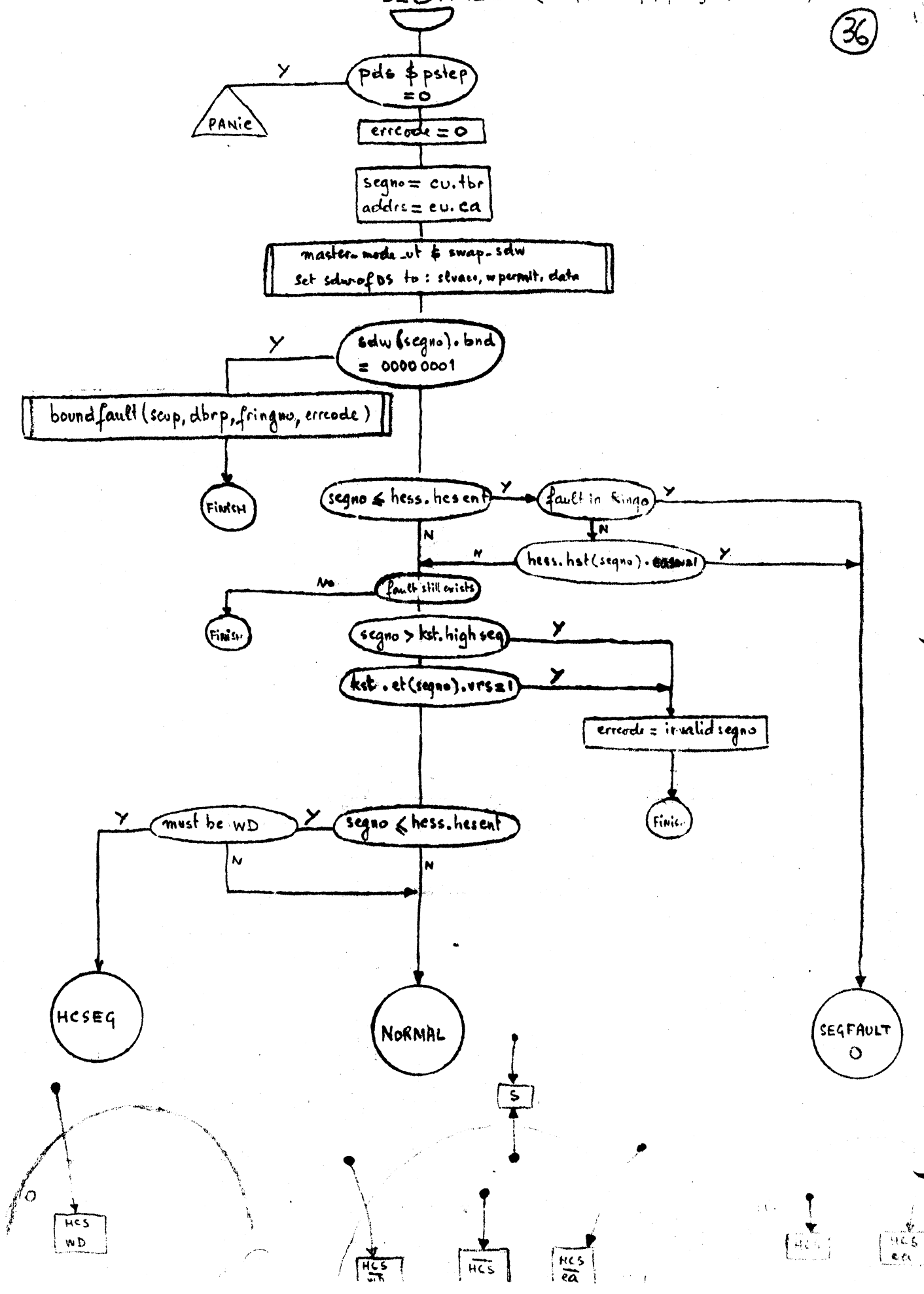
MAKE UNKNOWN (seqptr, rsw, err)

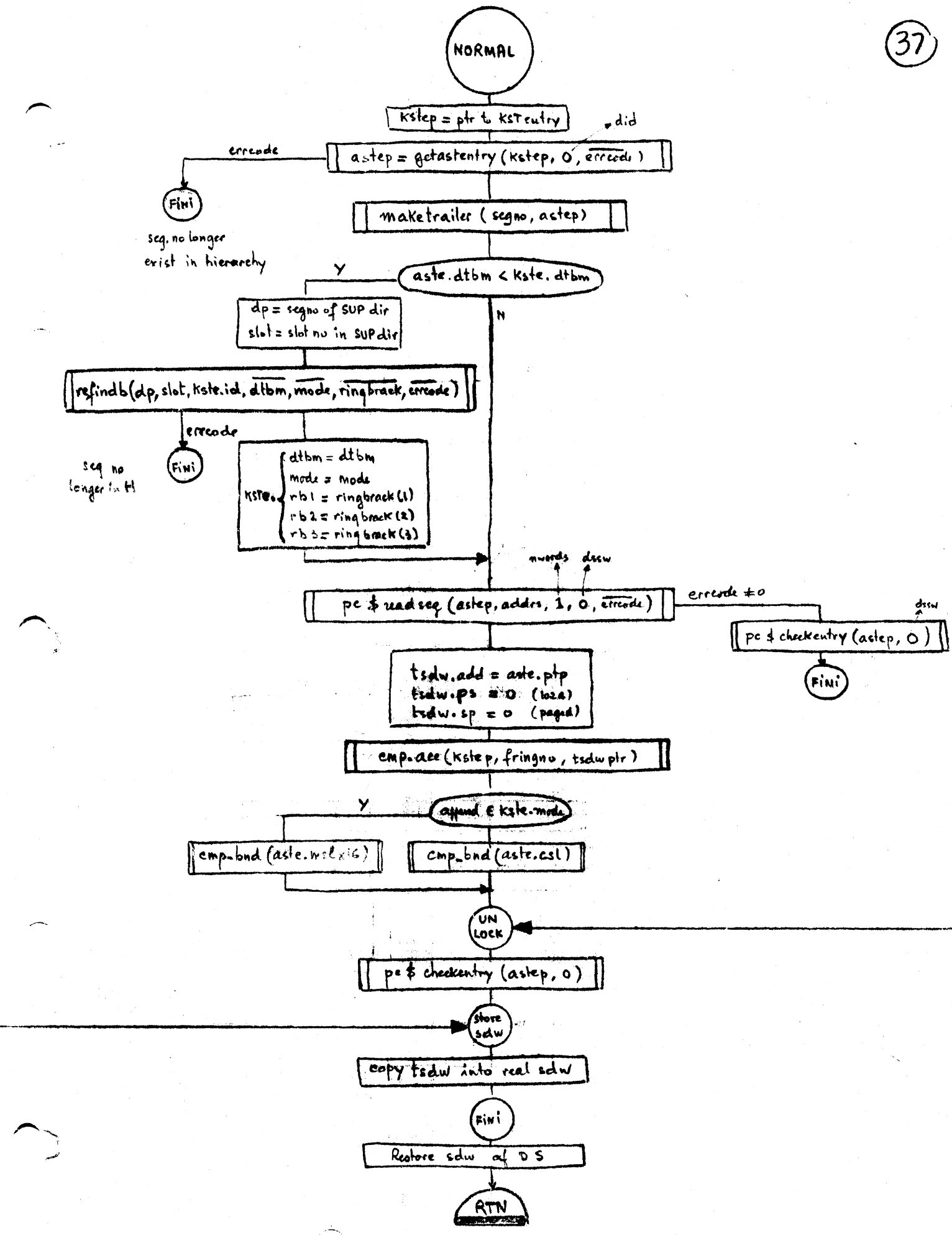


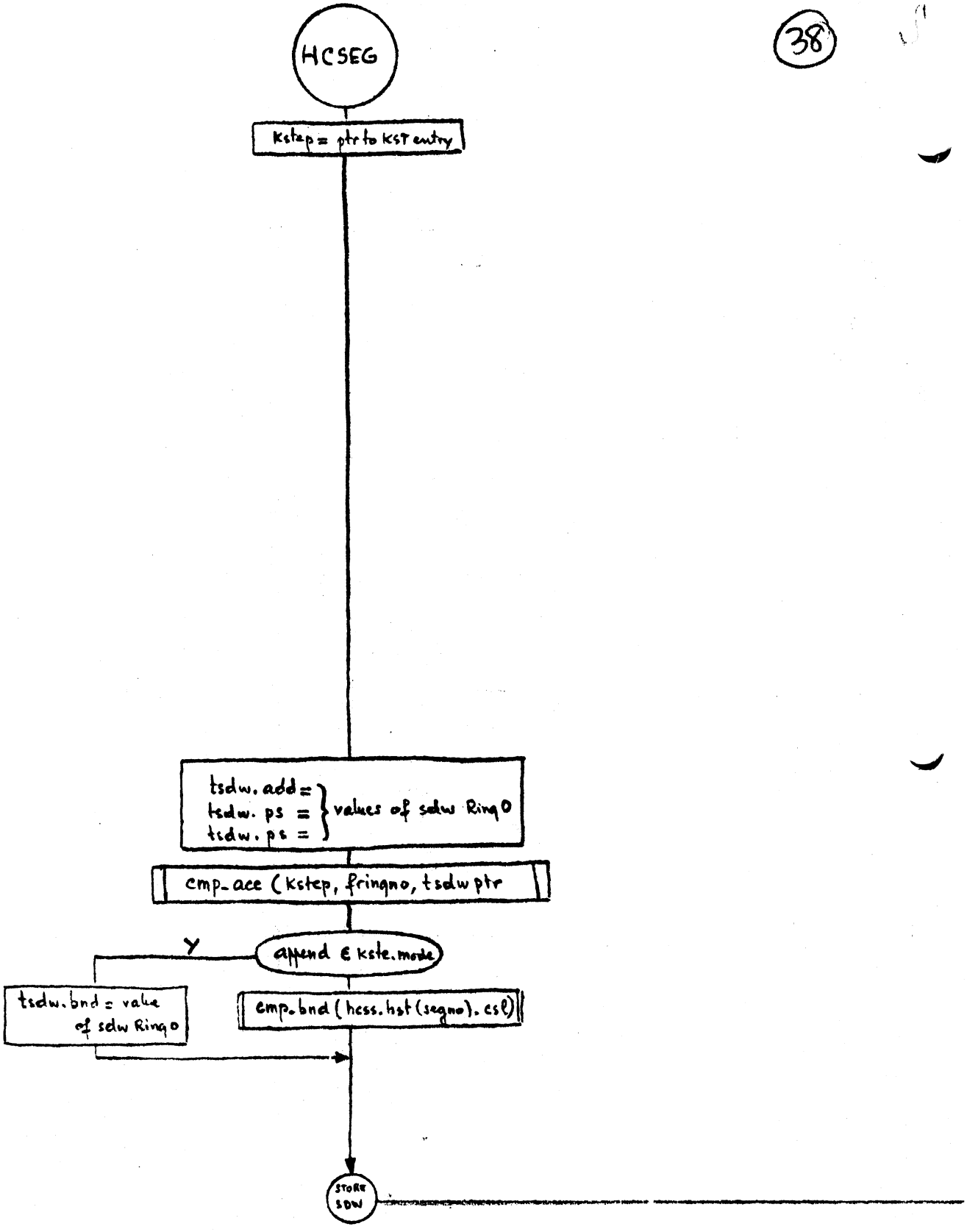
REMOVAL_LIST_UTIL $

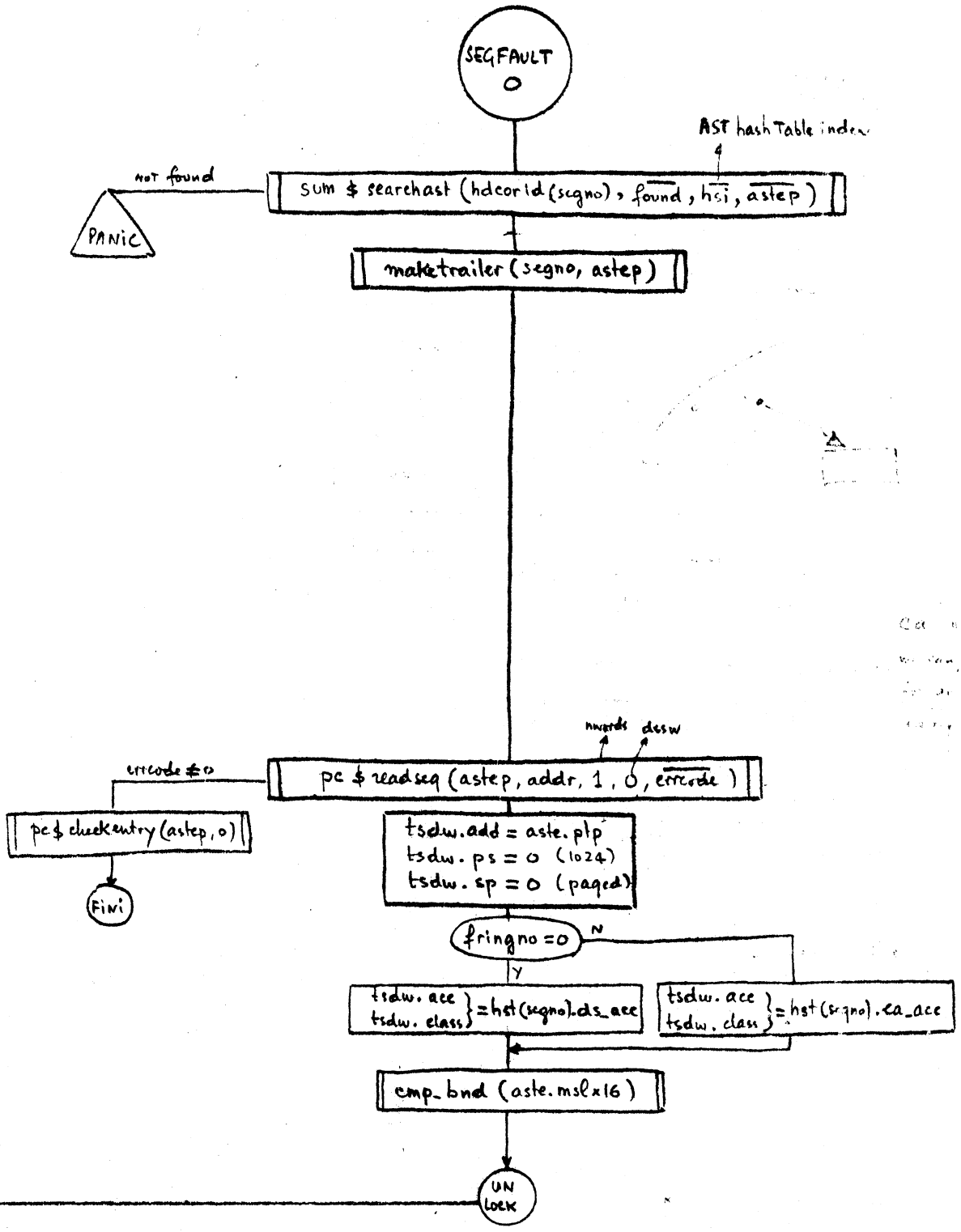


SEGFAULT (scup, dbrp, fringno, errcode)

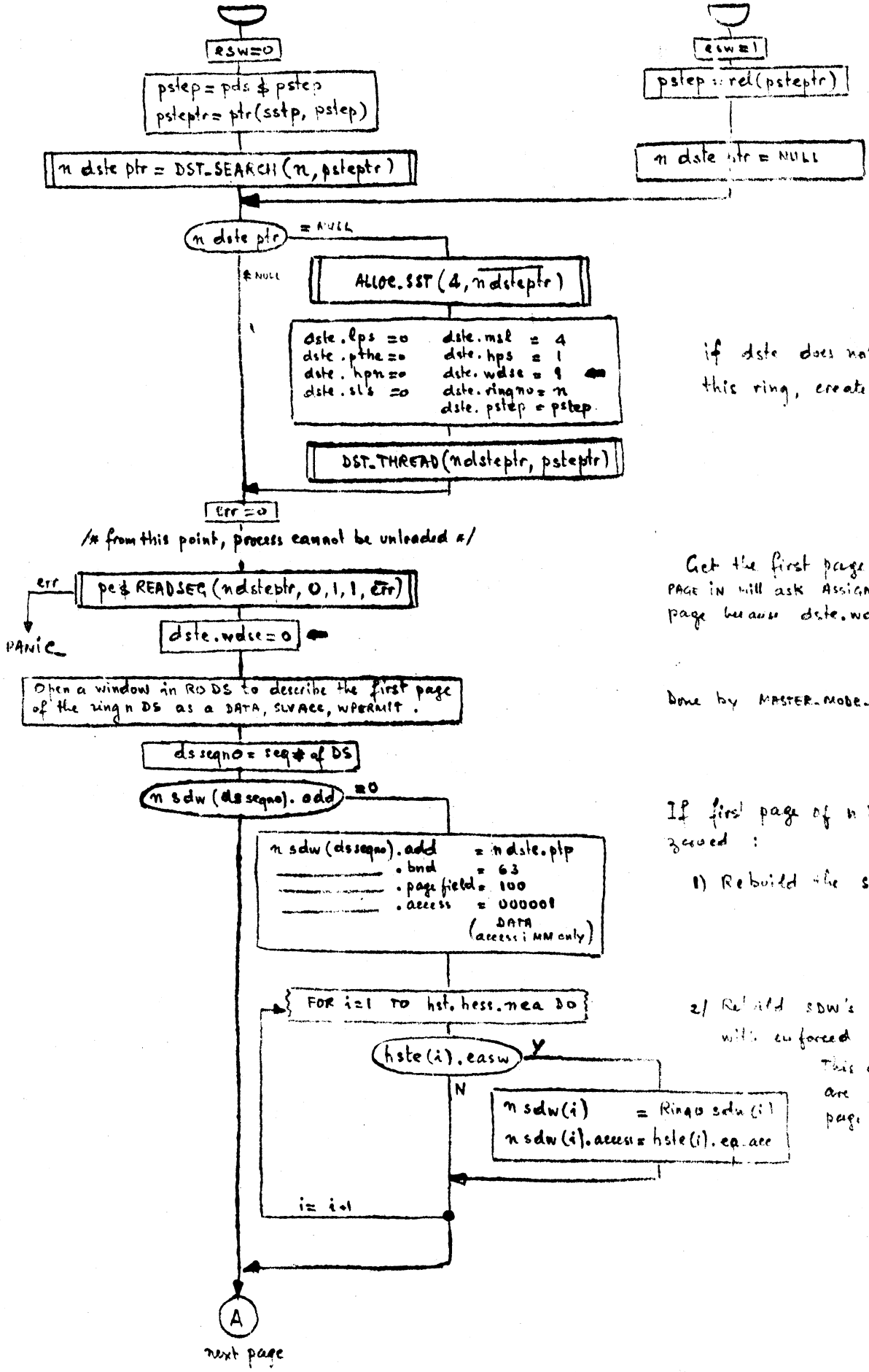



NORMAL
kstep = ptr to KST entry
did
astep = getastentry (kstep, 0, errcode)
errcode
FINI
seg. no longer exist in hierarchy
maketrailer (segno, astep)
aste.dtbm < kste.dtbm
Y
N
dp = segno of SUP dir
slot = slot no in SUP dir
refindb(dp, slot, kste.id, dtbm, mode, ringbrack, errcode)
errcode
FINI
seg no longer in KST
KSTE: dtbm = dtbm
mode = mode
rb1 = ringbrack(1)
rb2 = ringbrack(2)
rb3 = ringbrack(3)
nwords
dseg
pc $ readseg (astep, addrs, 1, 0, errcode)
errcode ≠ 0
dseg
pc $ checkentry (astep, 0)
FINI
tsdw.add = aste.ptp
tsdw.ps = 0 (not a)
tsdw.sp = 0 (paged)
cmp.acc (kstep, fringno, tsdw ptr)
Y
append ∈ kste.mode
cmp_bnd (aste.maxlen)
cmp_bnd (aste.csl)
UN LOCK
pc $ checkentry (astep, 0)
store sdw
copy tsdw into real sdw
FINI
Restore sdw of DS
RTN


HCSEG
kstep = ptr to KST entry
tsdw.add =
tsdw.ps =
tsdw.ps =
values of sdw Ring 0
cmp_ace (kstep, fringno, tsdw ptr
Y
append ∈ kste.mode
tsdw.bnd = value of sdw Ring 0
cmp.bnd (hcss.hst(segno).csl)
STORE SDW


SEGFAULT
0
AST hash Table index
not found
sum $ searchast (hdcorid(segno), found, hsi, astep)
PANIC
maketrailer (segno, astep)
nwords
dcsw
errcode ≠ 0
pc $ readseg (astep, addr, 1, 0, errcode)
pc $ checkentry (astep, 0)
tsdw.add = aste.ptp
tsdw.ps = 0 (1024)
tsdw.sp = 0 (paged)
Fini
fringno = 0
N
Y
tsdw.acc
tsdw.class
= hst(segno).ds_acc
tsdw.acc
tsdw.class
= hst(segno).ca_acc
emp_bnd (aste.msl×16)
UN
LOCK

SETUP_RING & SETUP_RING (n)



SETUP_RING $ LOAD (psteptr, n, dbr)

if dste does not exist for this ring, create one

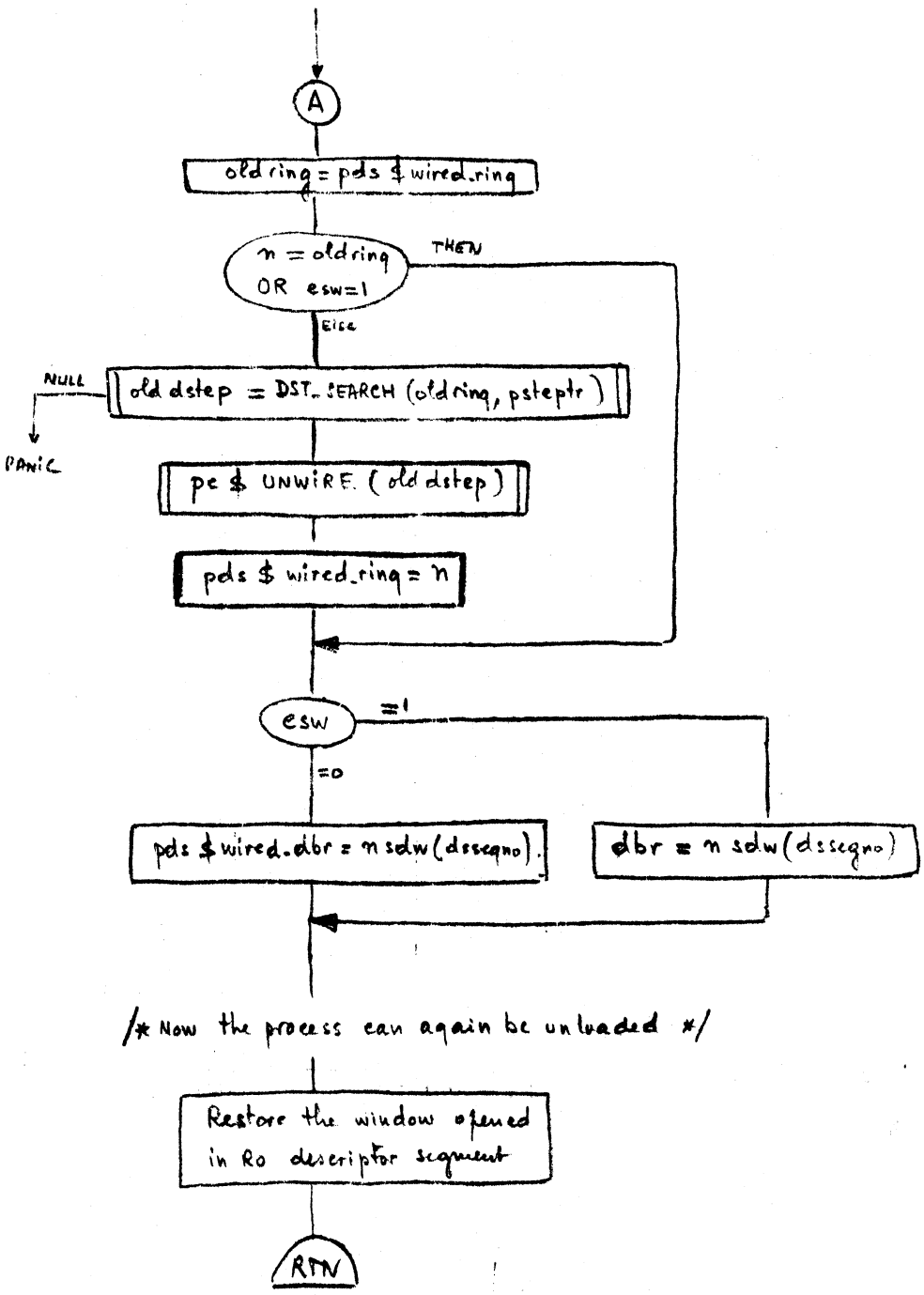
Get the first page of ring n DS - PAGE IN will ask ASSIGN to wire this page because dste.wdse = 1

Done by MASTER_MODE_UT $ SWAP_SDW

If first page of n DS has been zeroed :

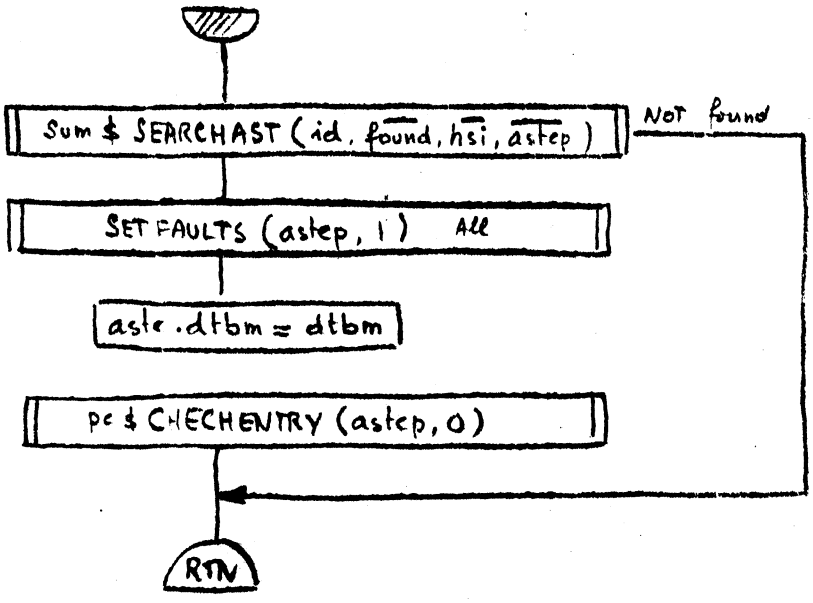
1) Rebuild the SDW of n DS

2) Rebuild SDW's of all HC segments with enforced access.
This assumes that they are all in the first page of the DS.



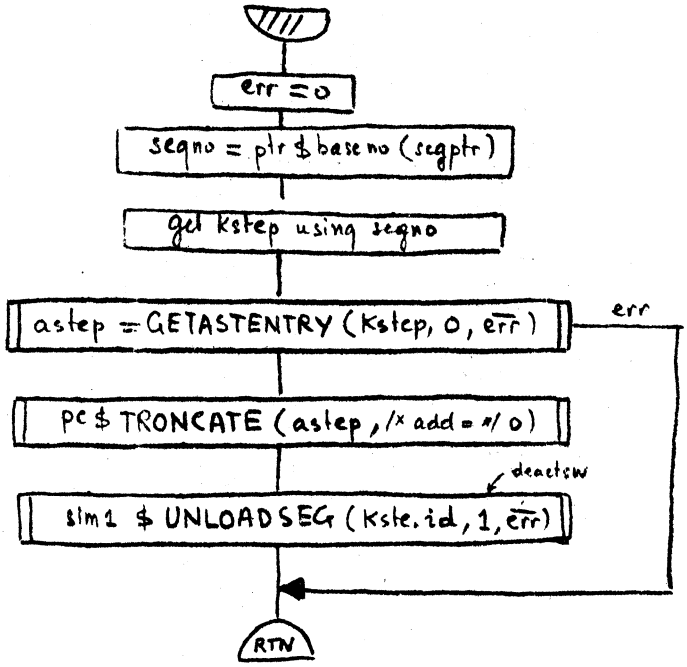
Question : How do we delete a DSTe

sim 1 $ BRANCHMOD (id, dtbm)

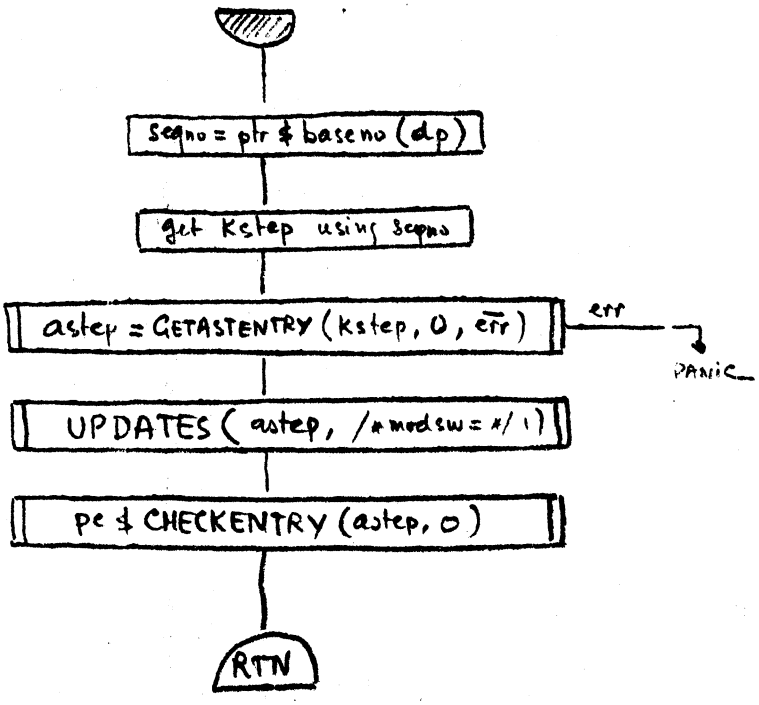


Called by DC to reflect changes in the access control AND/OR protection list

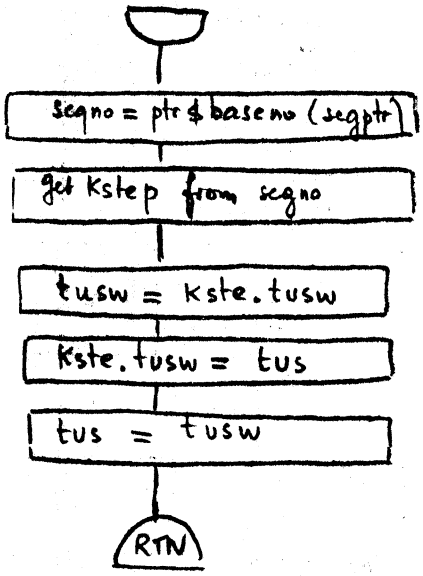
sim1 $ DELETSEG (seqptr, err)



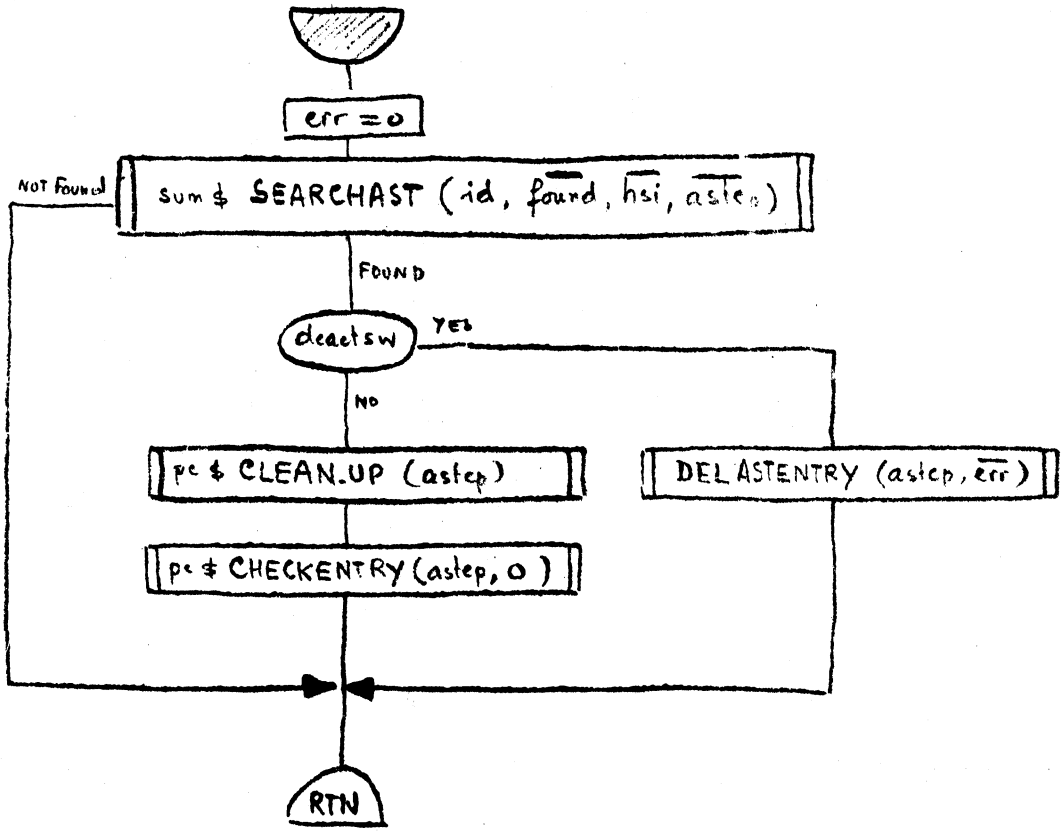
sim1 $ DIRMOD (dp)



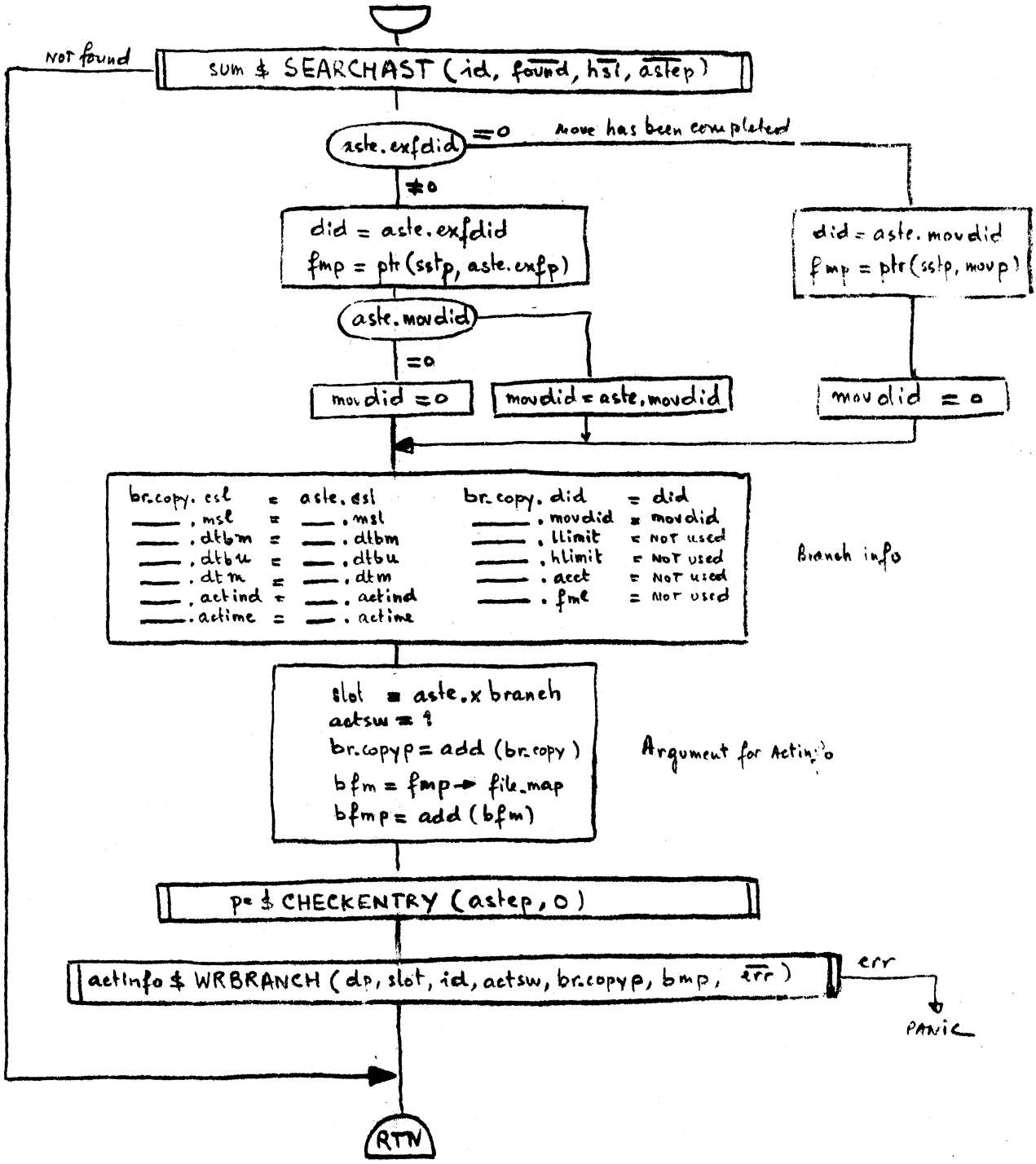
sim1 $ TRANSUSE (segptr, tus)



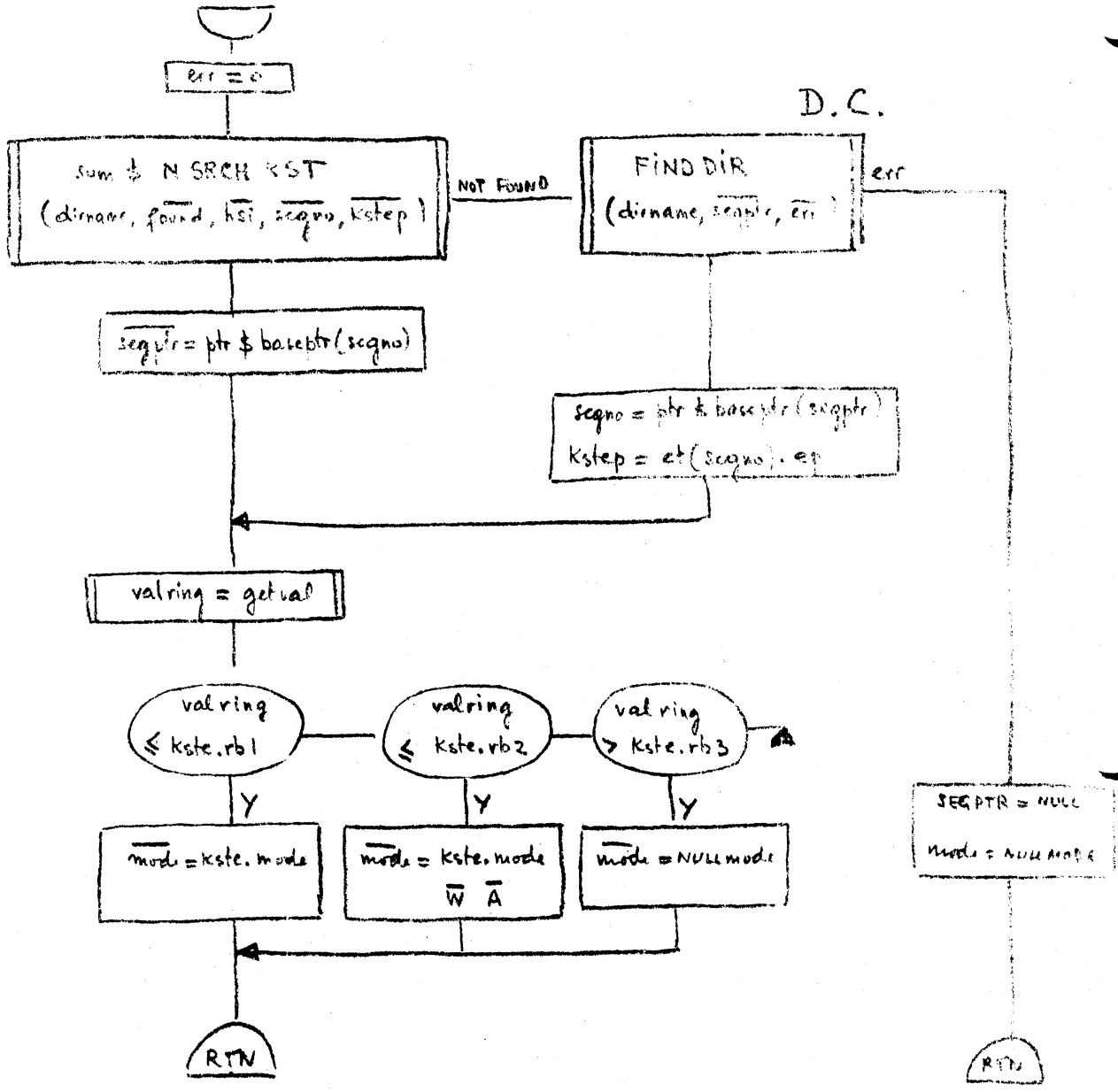
sim1 $ UNLOADSEG (id, deactsw, $\overline{err}$)



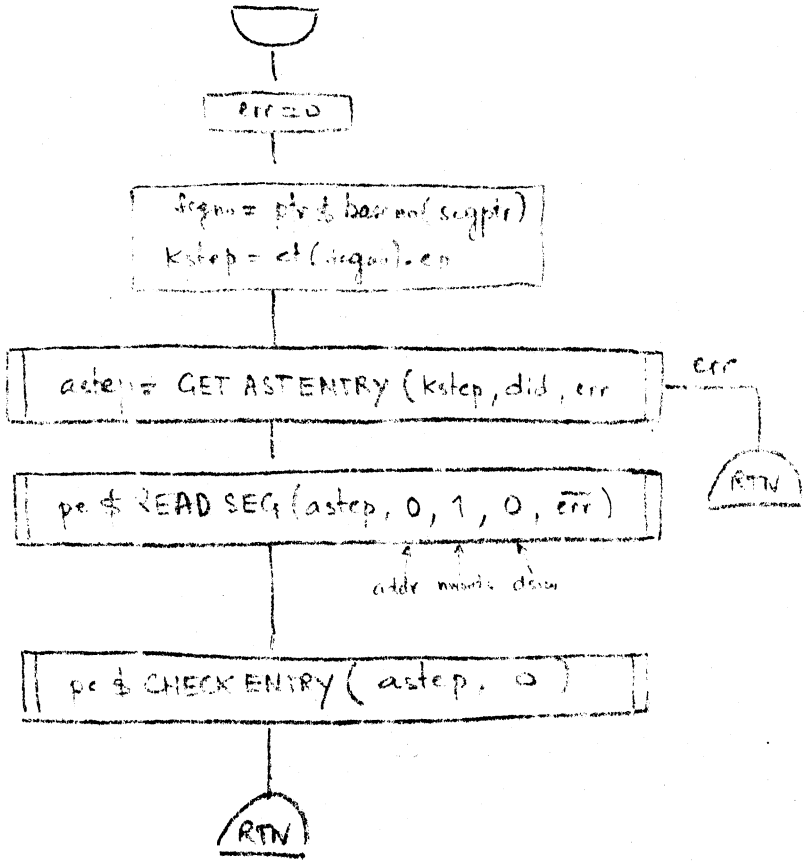
sim1 $ UPDATEB (id, dp)



sim$ GET DIR SEG (dirname, segptr, mode, err)

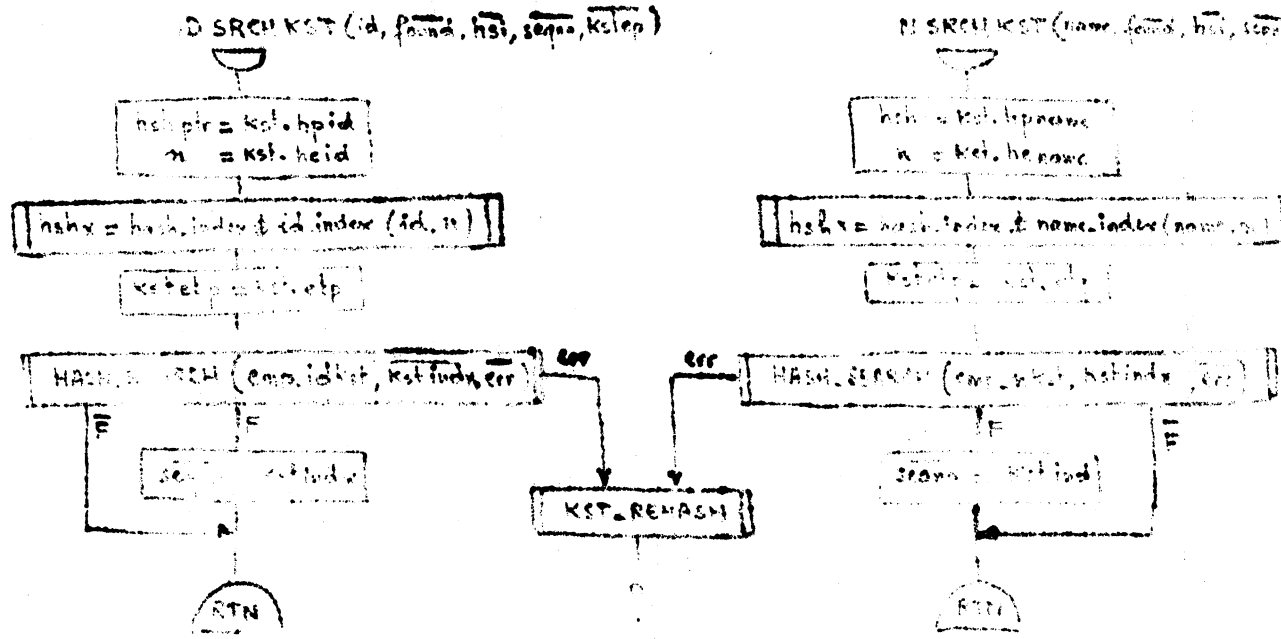


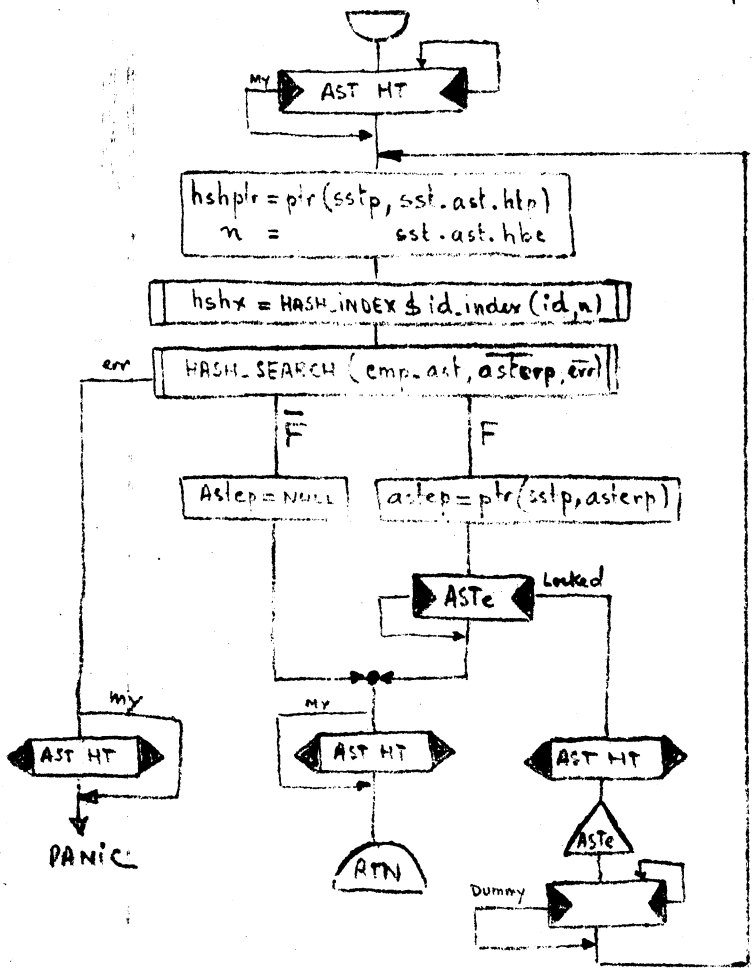
smm $ MOVE SEG (segptr, did, err)

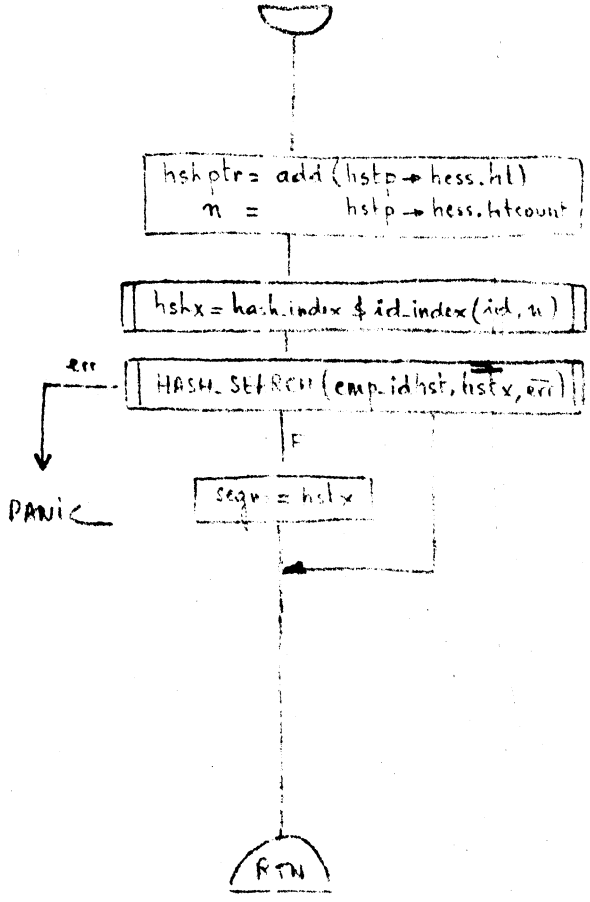


Called by MULTILEVEL to move a file specified by "segptr" to directory "did".
(The segment to be moved is supposed to be known by the process before
the call is issued).

SUM $


D SRCH KST (id, found, hsi, seqno, kstep)
hshptr = kst.hpid
n = kst.heid
hshx = hash_index $ id_index (id, n)
HASH_SEARCH (emp_idkst, kstfndx, err)
err
F
F
KST_REHASH
RTN
N SRCH KST (name, found, hsi, seqno, kstep)
hshptr = kst.hpname
n = kst.henname
hshx = name_index $ name_index (name, n)
HASH_SEARCH (emp_name, hshindex, err)
err
F
F
RTN

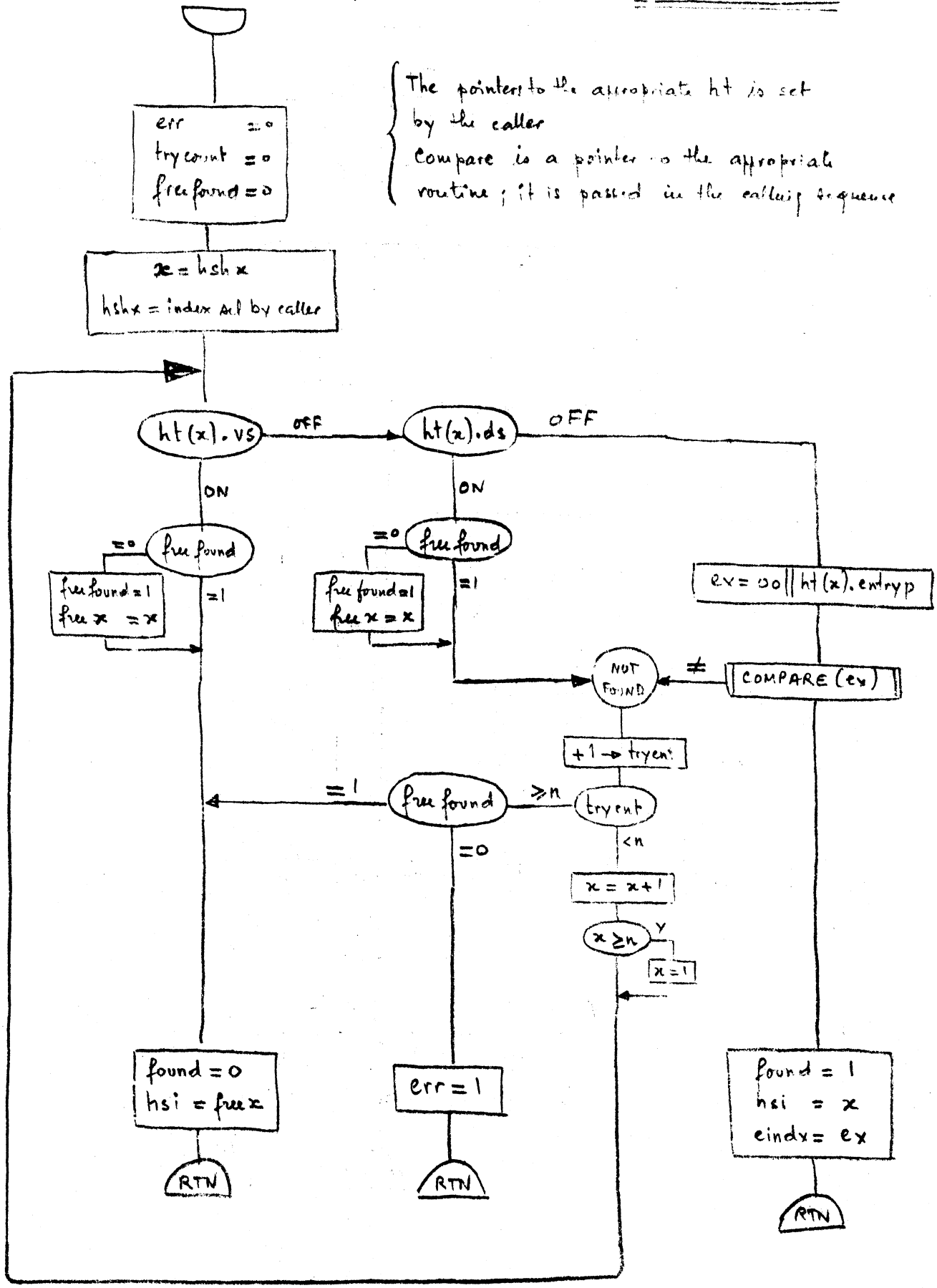

SEARCH AST (id, found, hsi, astep)
my
AST HT
hshptr = ptr (sstp, sst.ast.htp)
n = sst.ast.hbe
hshx = HASH_INDEX $ id_index (id, n)
err
HASH_SEARCH (emp_ast, asterp, err)
F
F
Astep = NULL
astep = ptr (sstp, asterp)
ASTe
Linked
my
AST HT
PANIC
my
AST HT
RTN
AST HT
ASTe
Dummy


SEARCH HST
hshptr = add (hstp → hess.ht)
n = hstp → hess.htcount
hshx = hash_index $ id_index (id, n)
err
HASH_SEARCH (emp_idhst, hstx, err)
PANIC
F
seqn = hslx
RTN

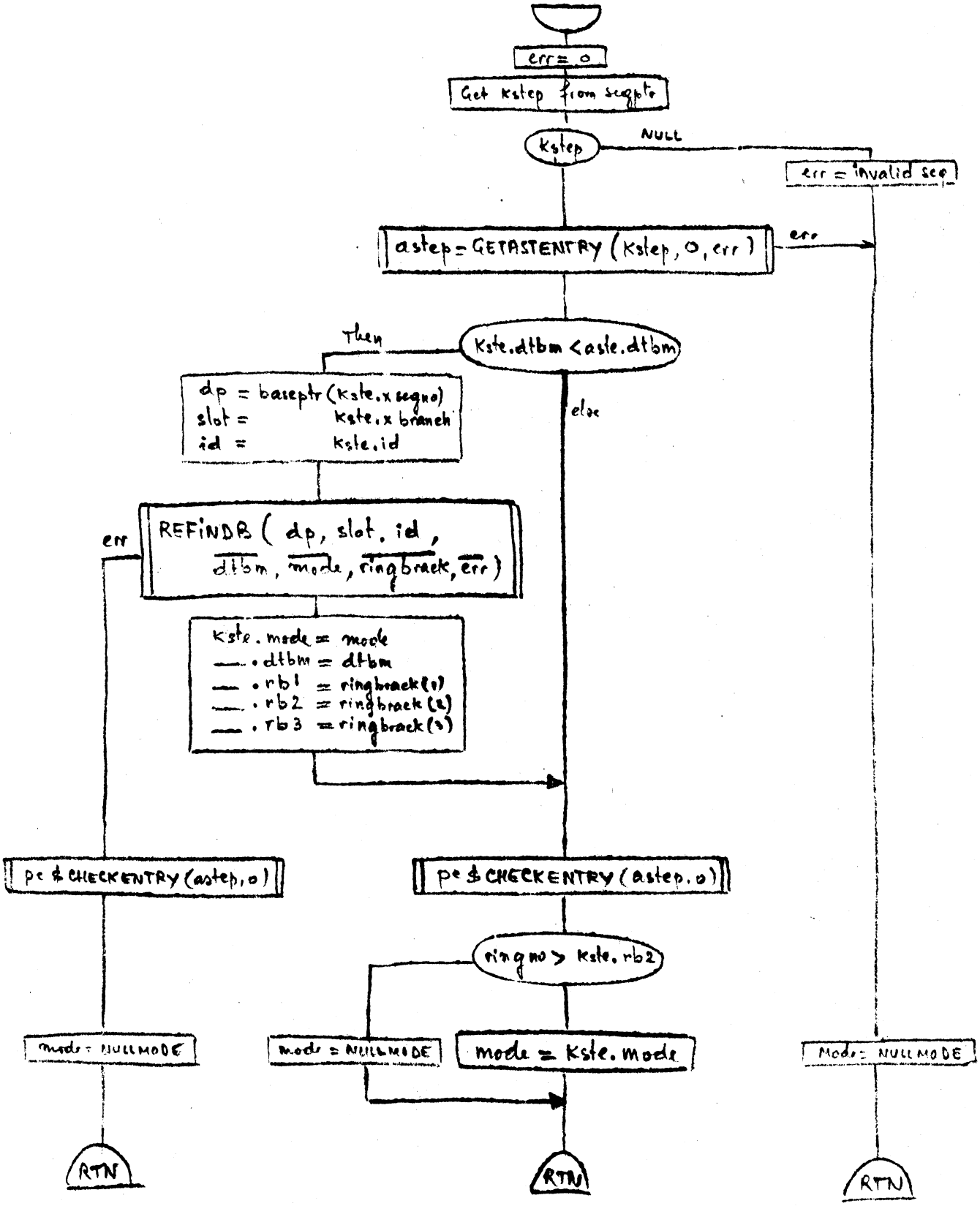
HASH_SEARCH (compare, eindx, err)

INTERNAL TO SUM -

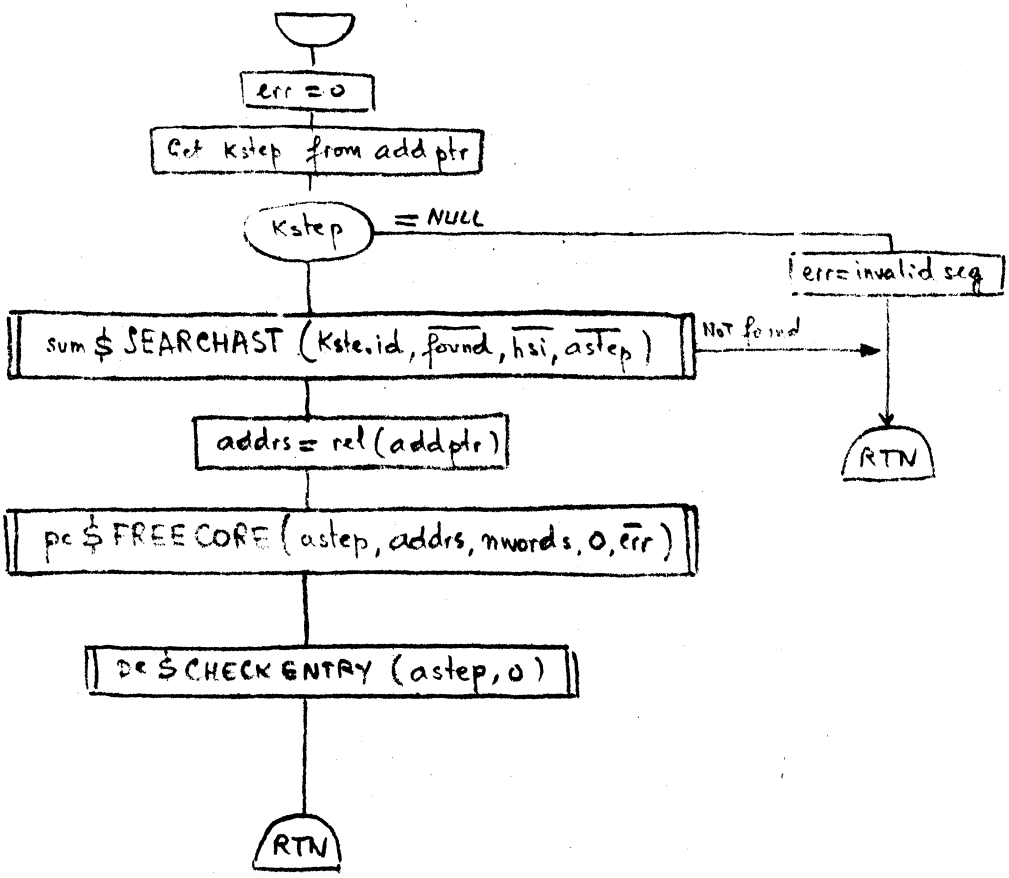
The pointer to the appropriate ht is set by the caller
compare is a pointer to the appropriate routine; it is passed in the calling sequence



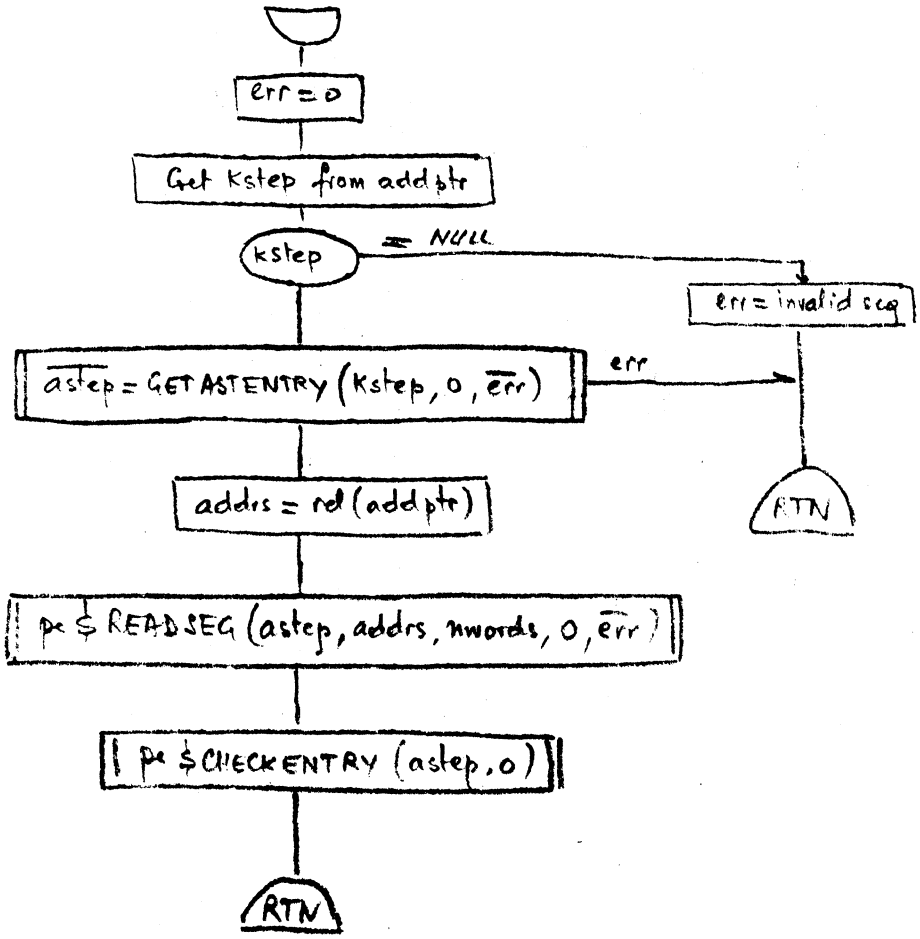
vim $ CHECKACCESS (seaptr, ringno, mode, err)



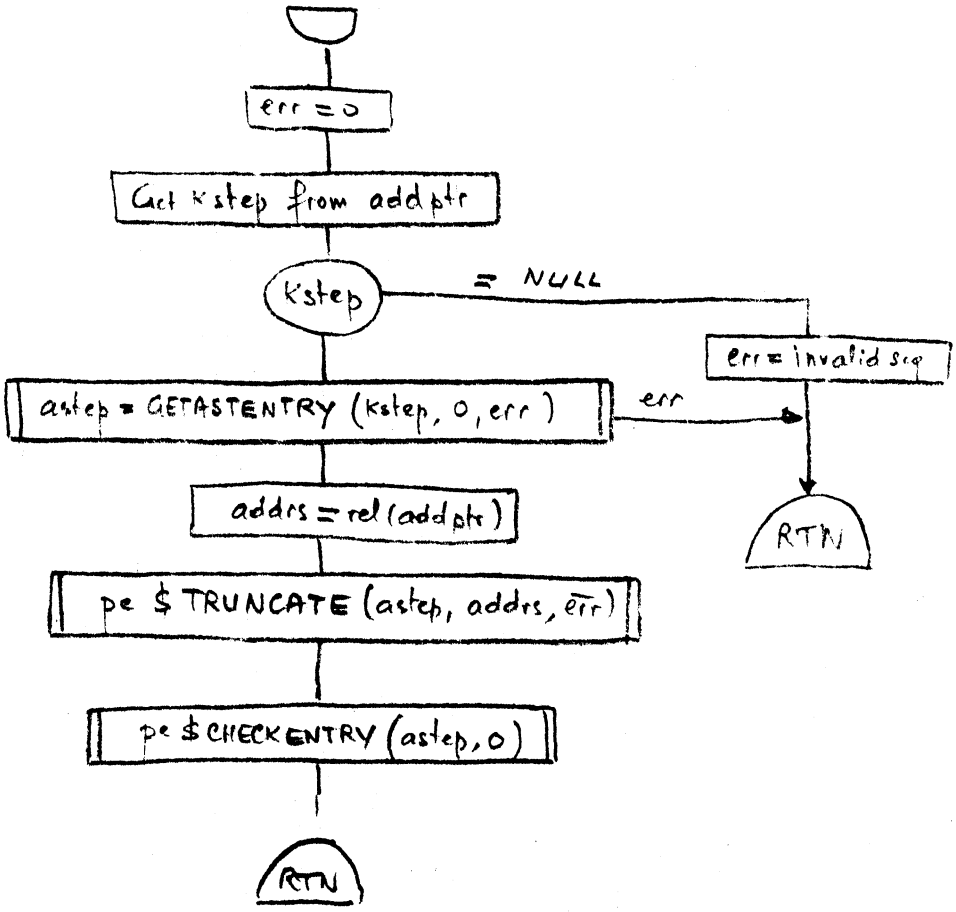
uim $ FREE CORE (addptr, nwords, err)



vim $ READSEG (addptr, nwords, err)



dim $ TRUNCATE SEG (addptr, err)



DIRECTORY CONTROL

DIRECTORY MAINTAINER
FINDBRANCH
FINDENTRY
HASH $IN
$OUT
$SEARCH
PACKER1
PACKER2
REHASH
REMOVEB
REMOVEL

SYSTEM INTERFACE
ACTIVINFO $RDBRANCH
$WRBRANCH
ESTBLSEG
FINDDIR
REFINDB
SETBASEDIR
SETUSAGE

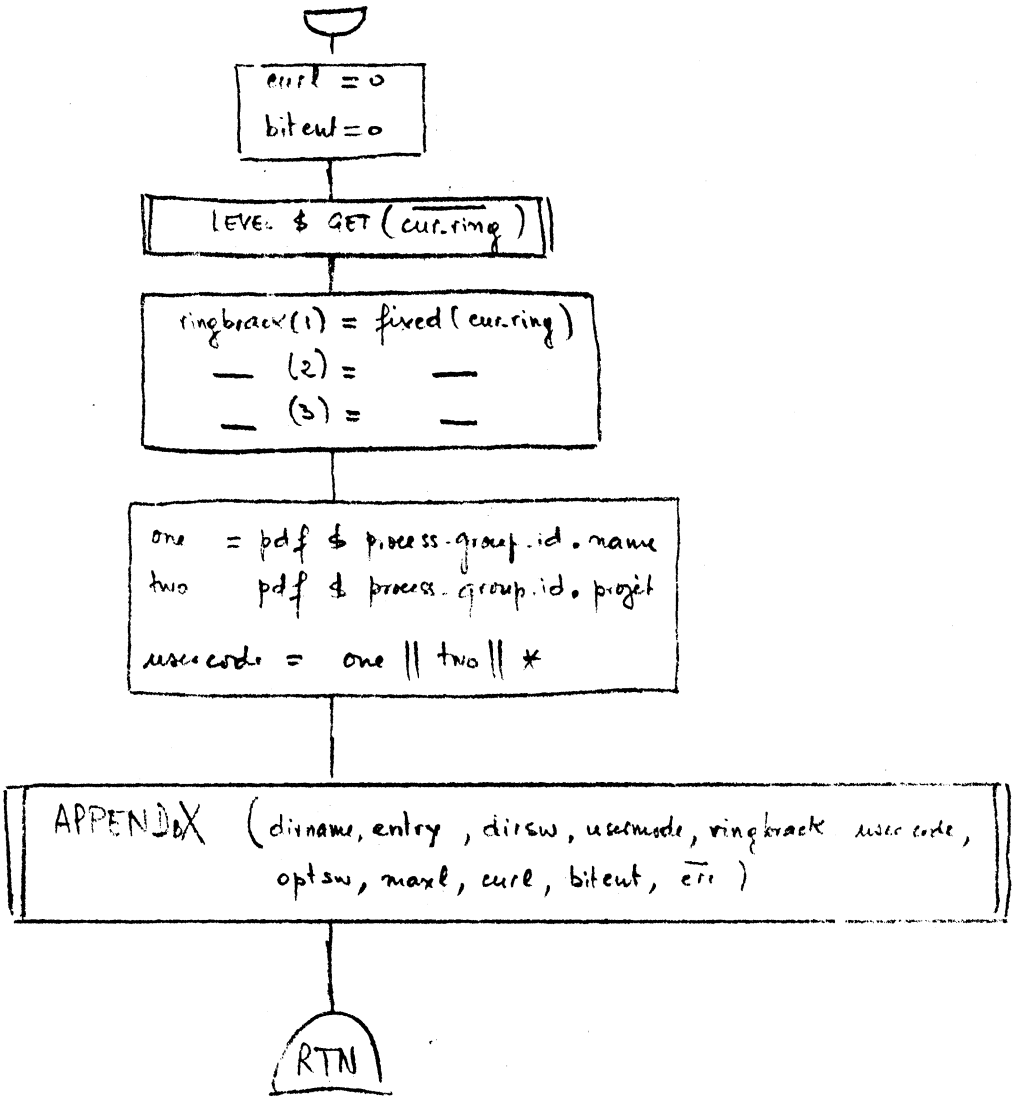
USER INTERFACE
APPENDB
APPENDBX
APPENDL
CHNAME
DELENTRY
LISTDIR
MOVEFILE
READACL
SET$bc
$CONSISTSW
$COPYSW
$RELATEDSW
$RD
SETML
STATUS
WRITEACL

ACCESS CONTROL
APPMODE $APPMODE
$APPMODE.ENTRY
EFFMODE

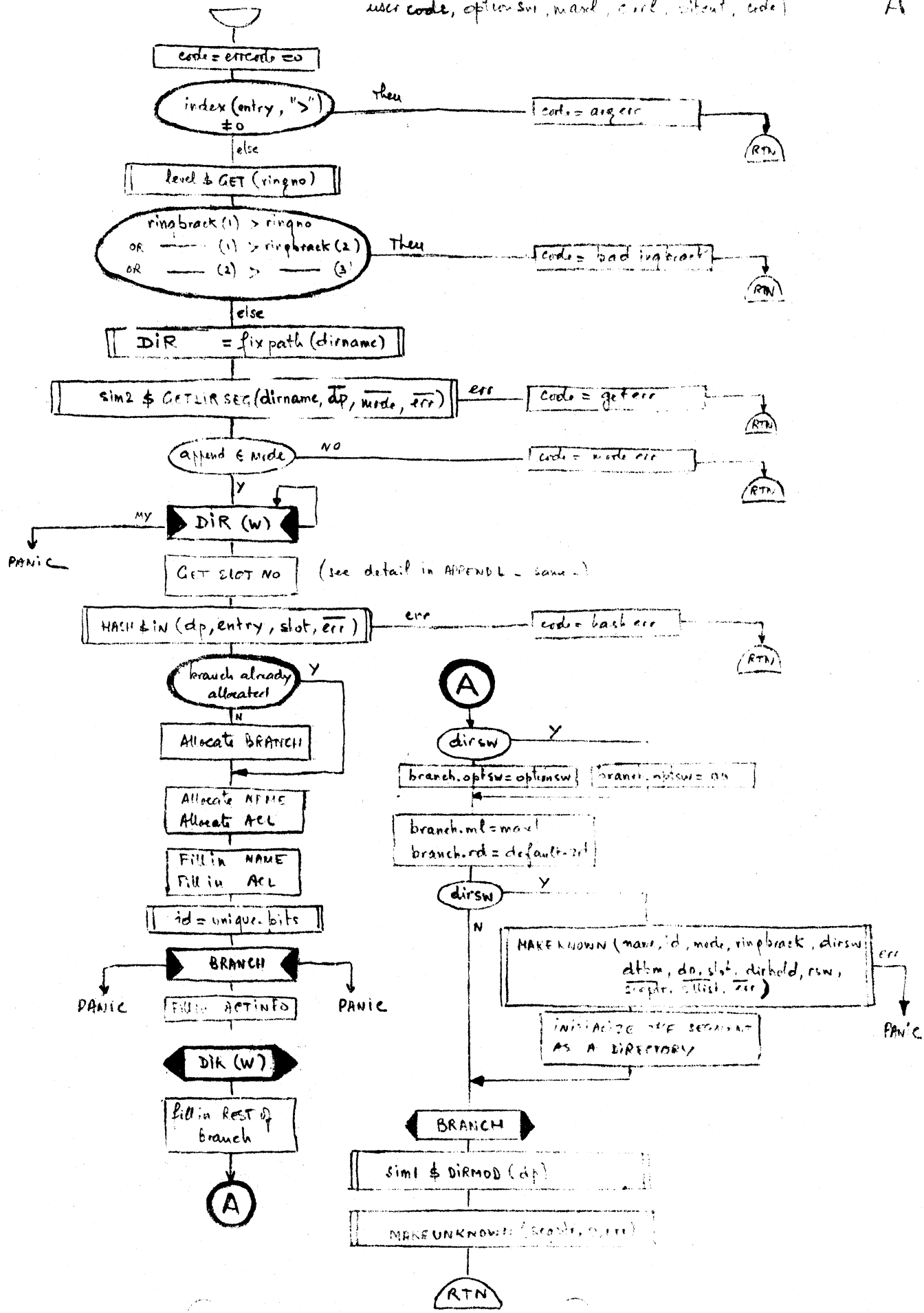
SPECIAL USER INTERFACE
GETENTRY
PUTENTRY
SETUID
SETLIMITS
SETRETRIEVE
SETSYSTRAP

APPEND B (dirname, entry , dirsw, usermode, optsw, maxl, code)

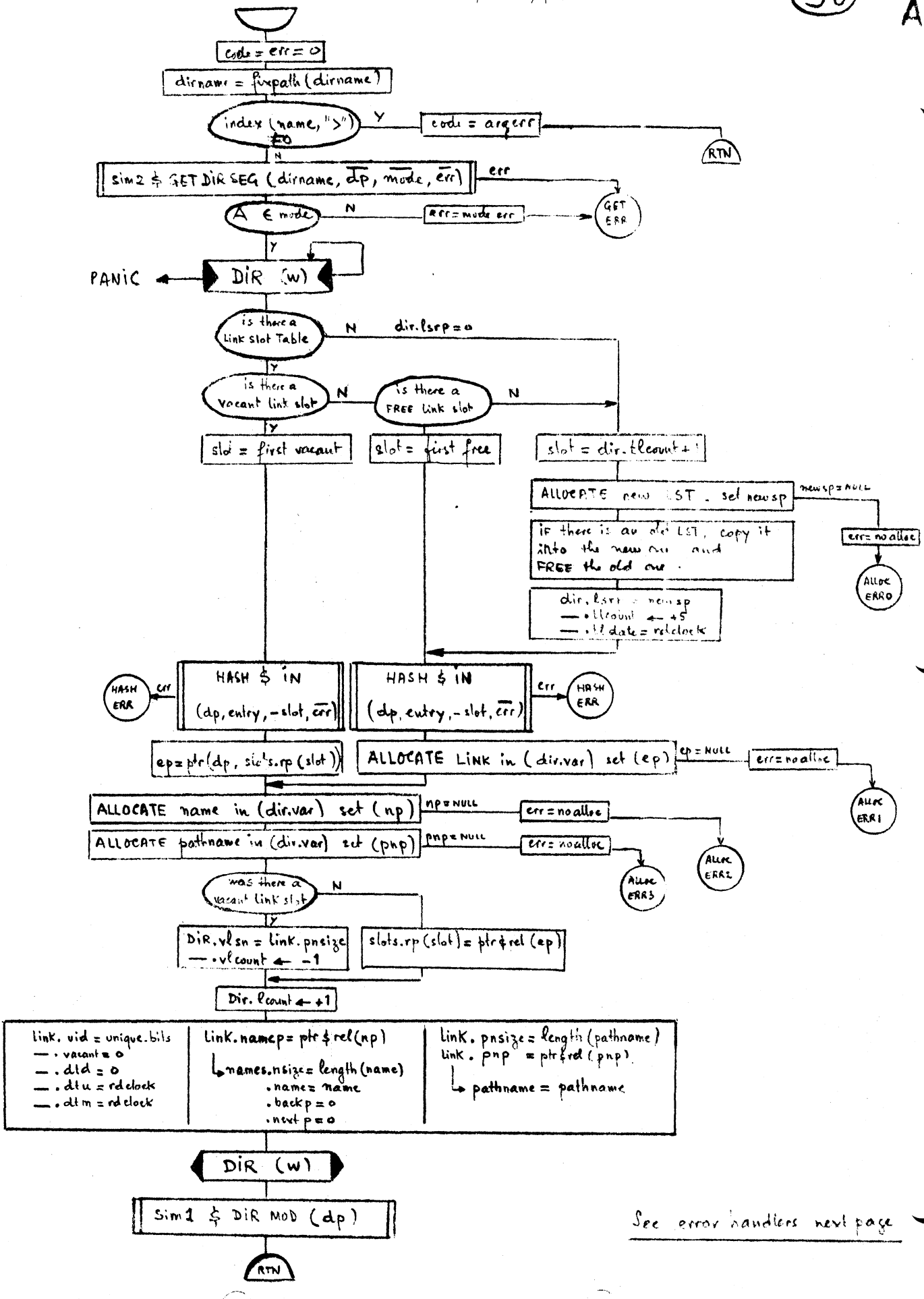
A



APPENDBX (dirname, entry, dirsw, usercurrdir, ringbrack, usercode, optionsw, maxl, curl, silent, code)



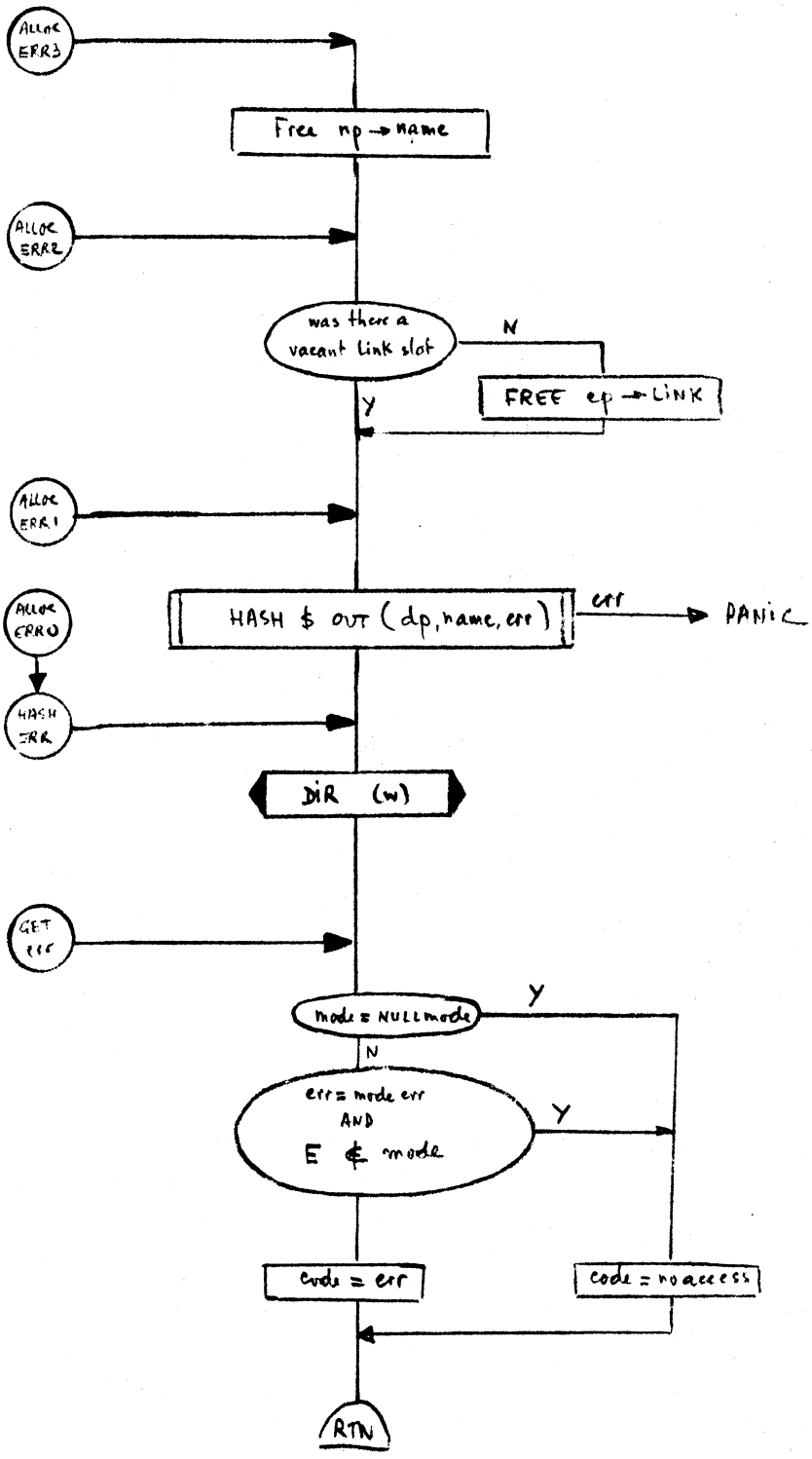
APPENDL (dirname, name, pathname, err)



See error handlers next page

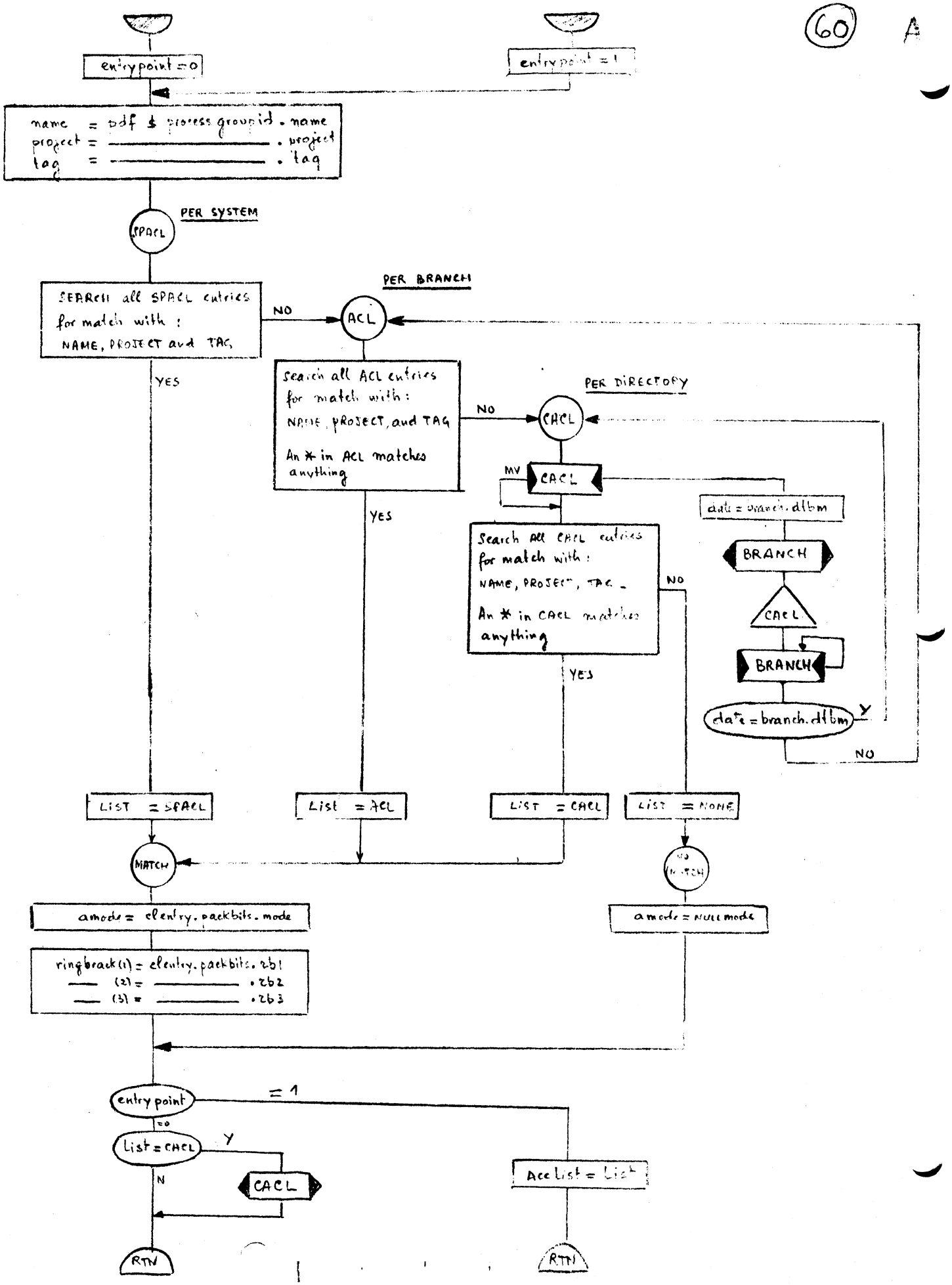
A

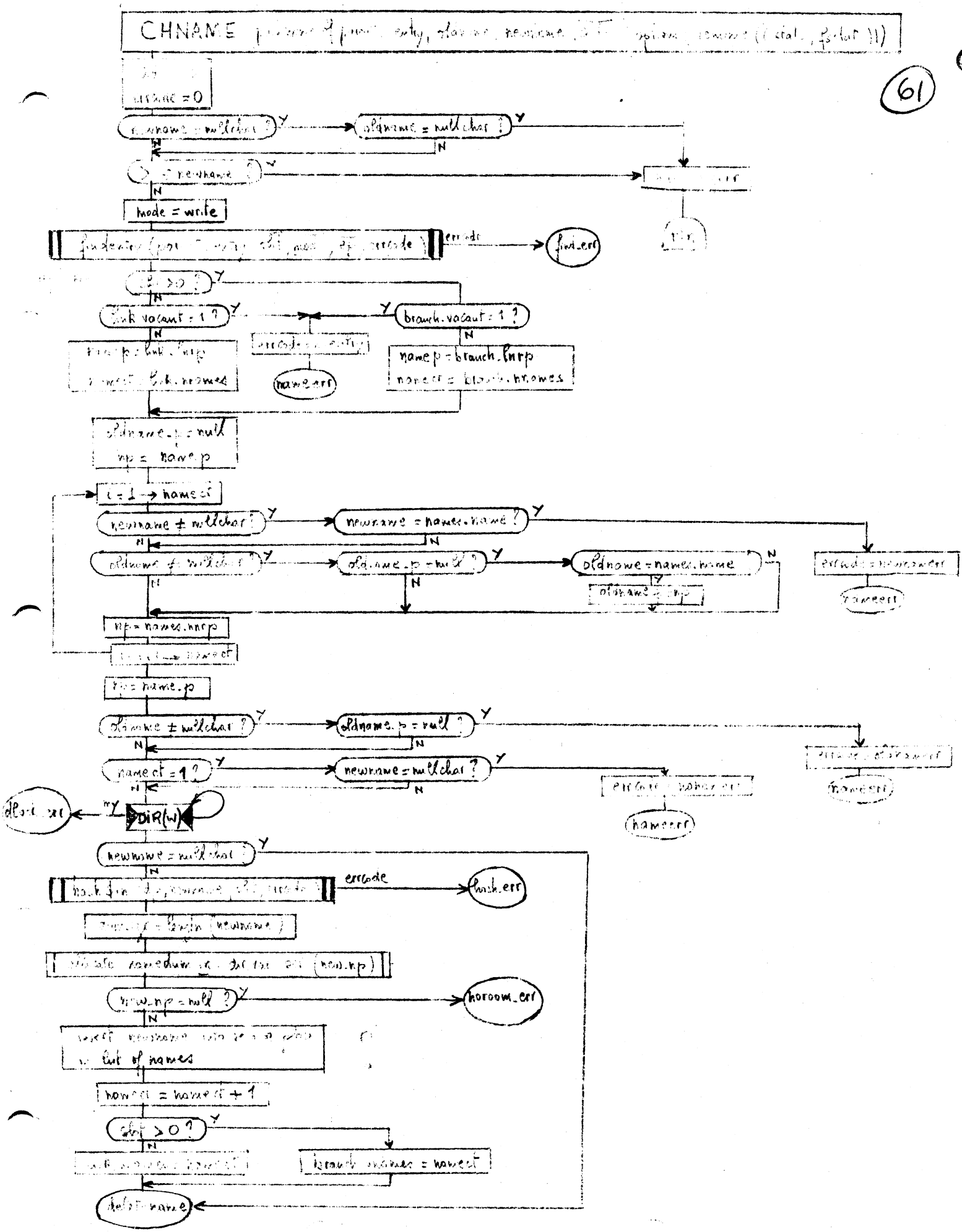
APPEND L error handlers.

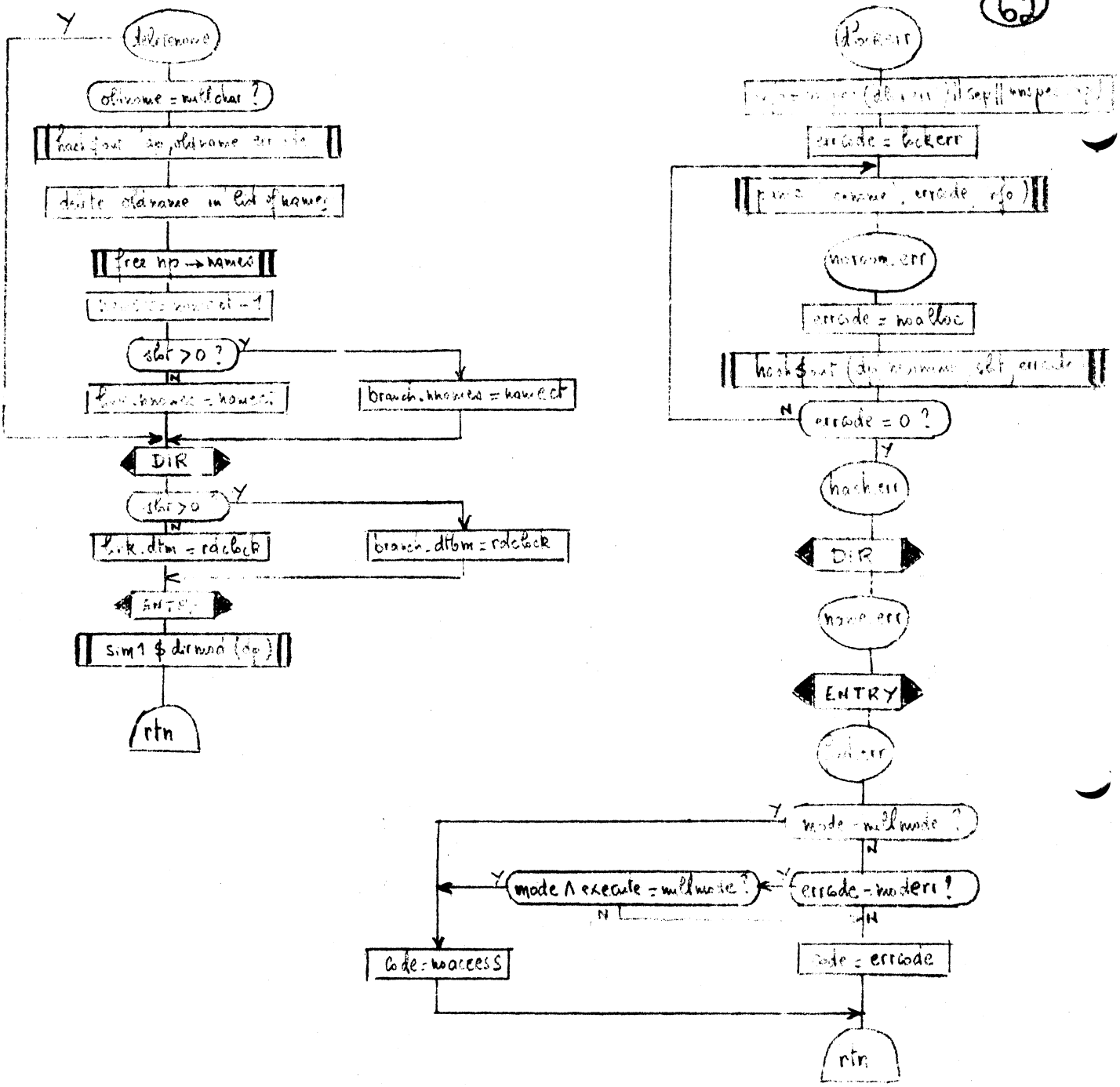


APPMODE (ep, dp, amode, ringbrack)

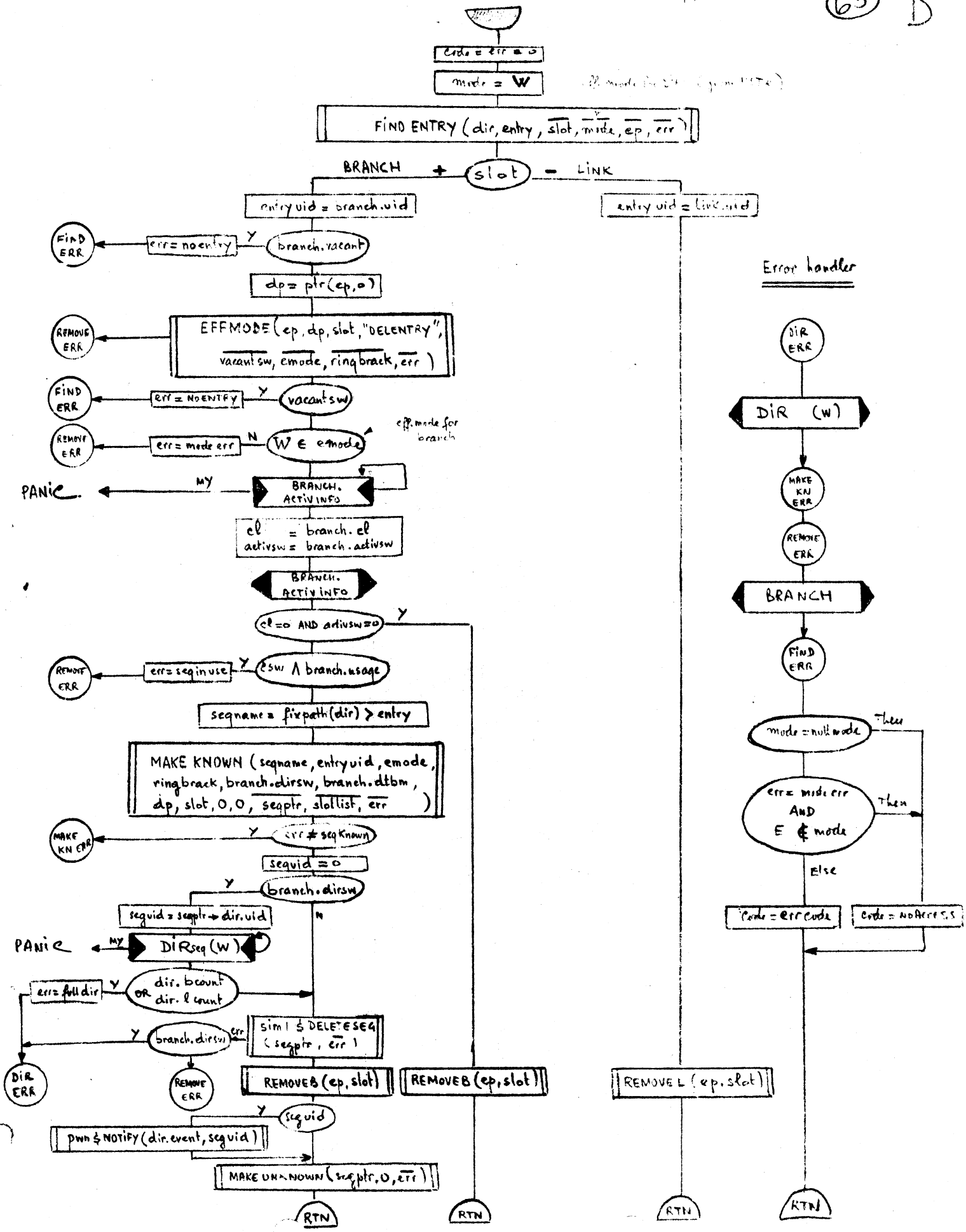
APPMODE & APPMODE_ENTRY (ep, dp, amode, ringbrack, acc list)

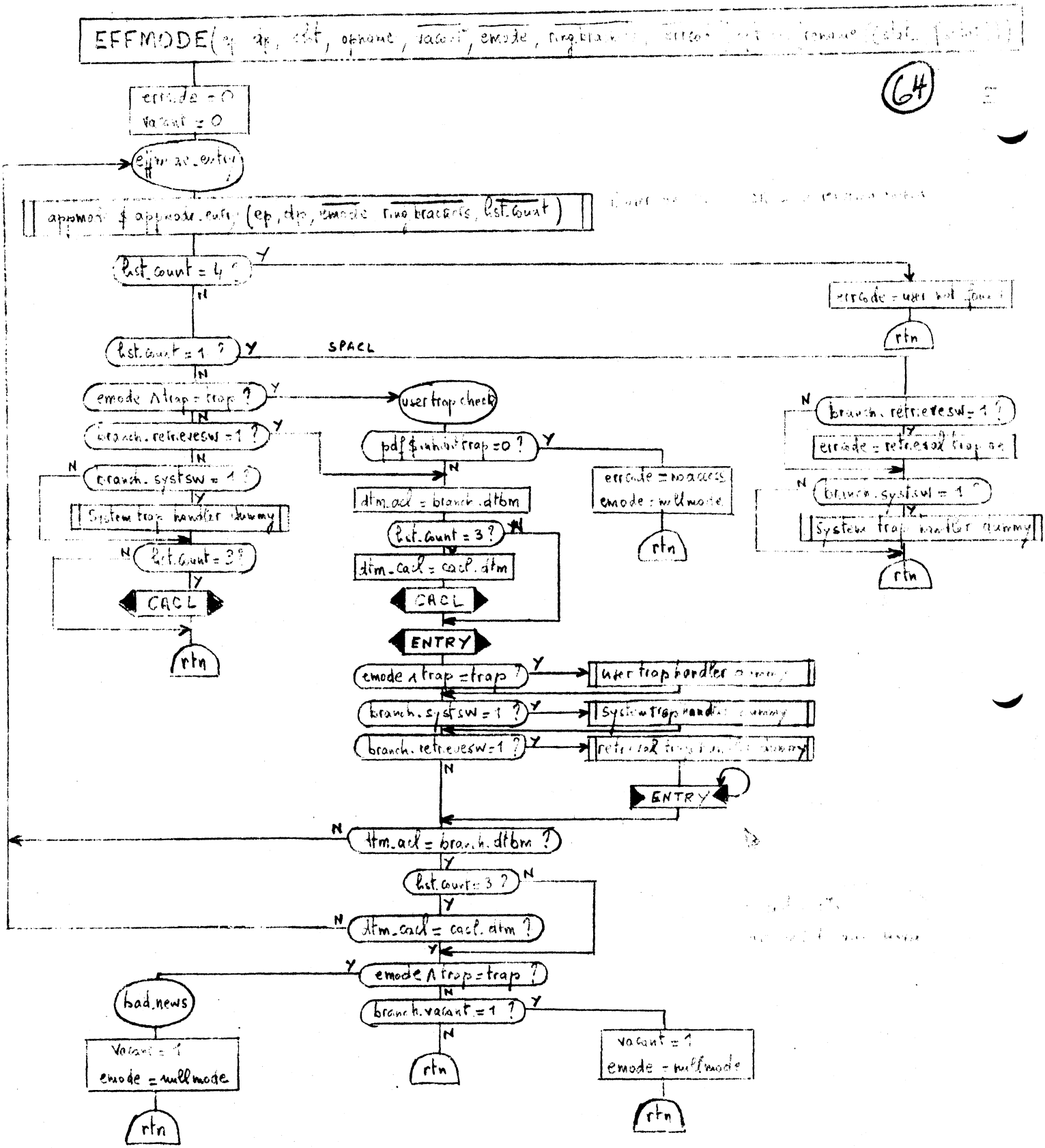



CHNAME
oldname = null ?
newname = null ?
mode = write
err.code
find.err
name.err
oldname_p = null
np = name.p
newname ≠ null.char ?
oldname ≠ null.char ?
oldname = names.name
name.err
DIR(w)
newname = null.char ?
name.err
newname = null.char ?
hash.err
horoom.err
name


oldrename
oldname = nullchar ?
Y
N
DIR
ENTRY
rtn
mode = nullmode ?
mode ∧ execute = nullmode ?
errcode = moderr ?
code = noaccess
code = errcode
rtn

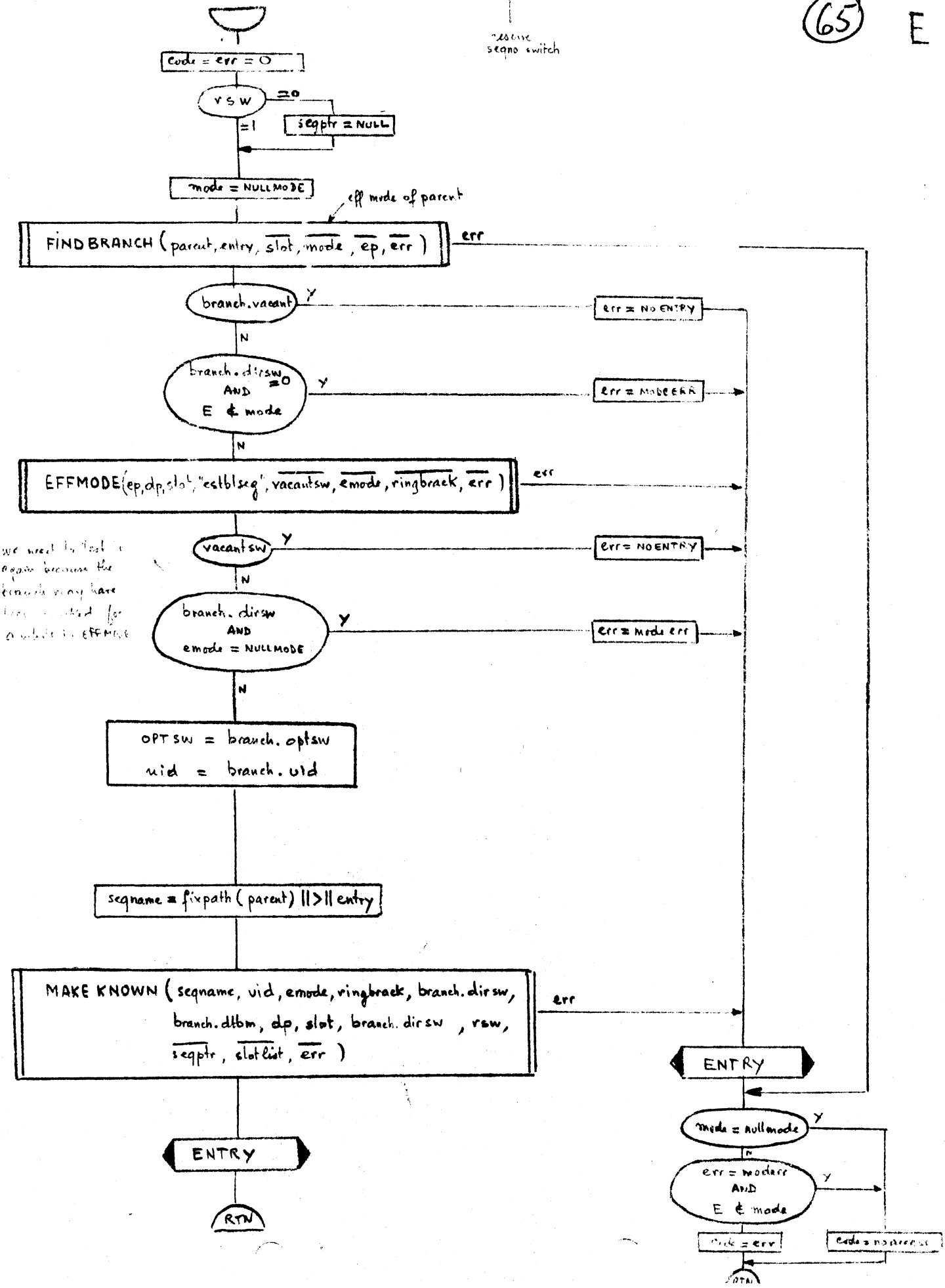
DELENTRY (dir, entry, esw, code)




EFFMODE(
errcode = 0
vacant = 0
effmode_entry
appmode = appmode_entry(ep, dbp, emode, ring brackets, lst.count)
lst.count = 4 ?
Y
N
errcode = user not found
rtn
lst.count = 1 ?
Y
SPACL
N
emode ∧ trap = trap ?
Y
user trap check
N
branch.retrieveesw = 1 ?
Y
N
branch.systsw = 1 ?
Y
System trap handler dummy
N
lst.count = 3 ?
Y
CACL
rtn
pdf $unknown trap = 0 ?
Y
N
dtm_acl = branch.dtbm
lst.count = 3 ?
N
dtm_acl = cacl.dtm
CACL
errcode = no access
emode = nullmode
rtn
N
branch.retrieveesw = 1 ?
Y
errcode = retrieval trap set
N
branch.systsw = 1 ?
Y
System trap handler dummy
rtn
ENTRY
emode ∧ trap = trap ?
Y
user trap handler dummy
branch.systsw = 1 ?
Y
System trap handler dummy
branch.retrieveesw = 1 ?
Y
retrieval trap handler dummy
ENTRY
N
dtm_acl = branch.dtbm ?
N
Y
lst.count = 3 ?
N
Y
dtm_acl = cacl.dtm ?
N
Y
emode ∧ trap = trap ?
Y
N
bad.news
vacant = 1
emode = nullmode
rtn
branch.vacant = 1 ?
Y
N
rtn
vacant = 1
emode = nullmode
rtn

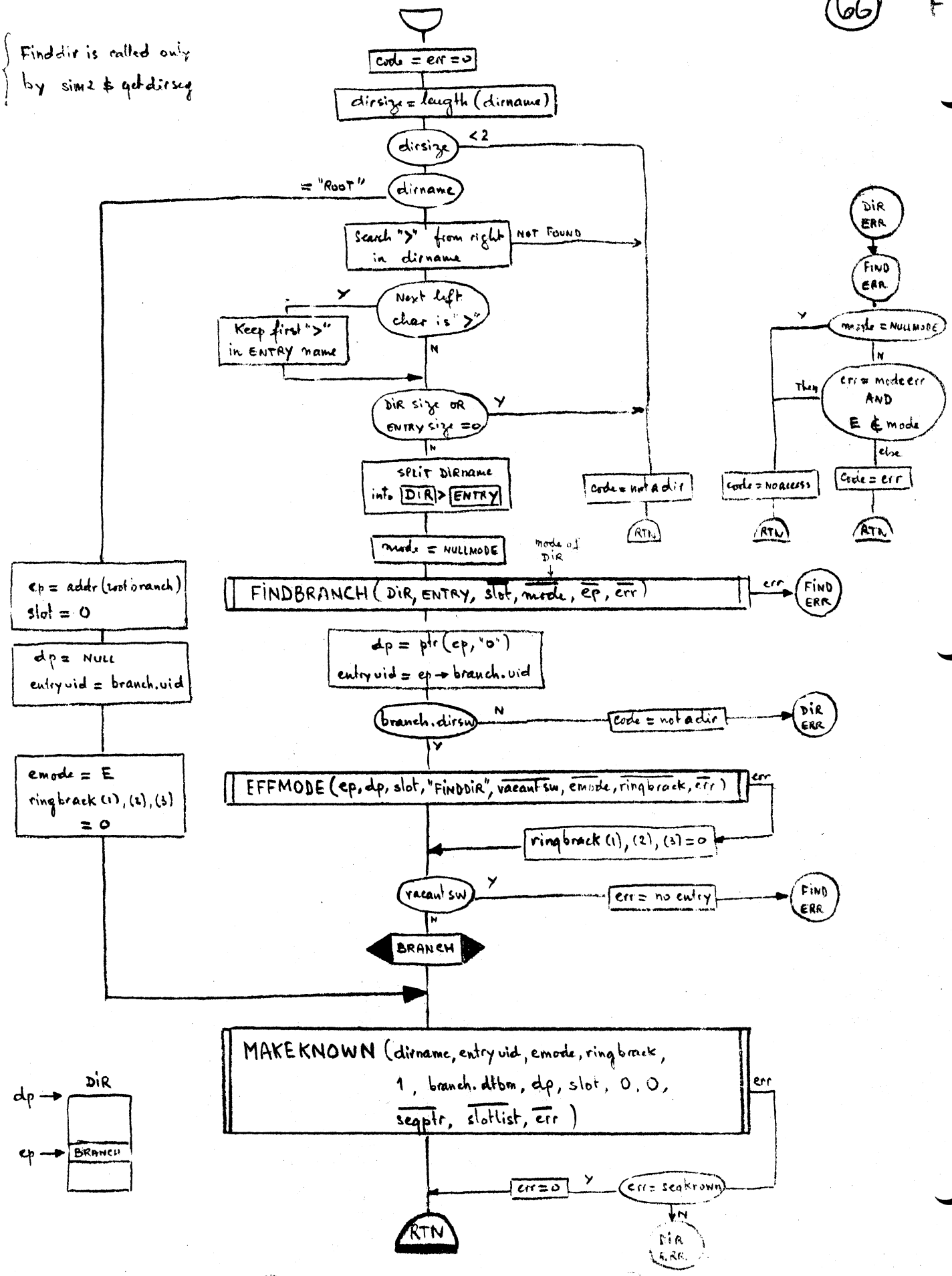
ESTBLSEG (parent, entry, rsw, seqptr, uid, optsw, slotlist, code)

receive seqno switch

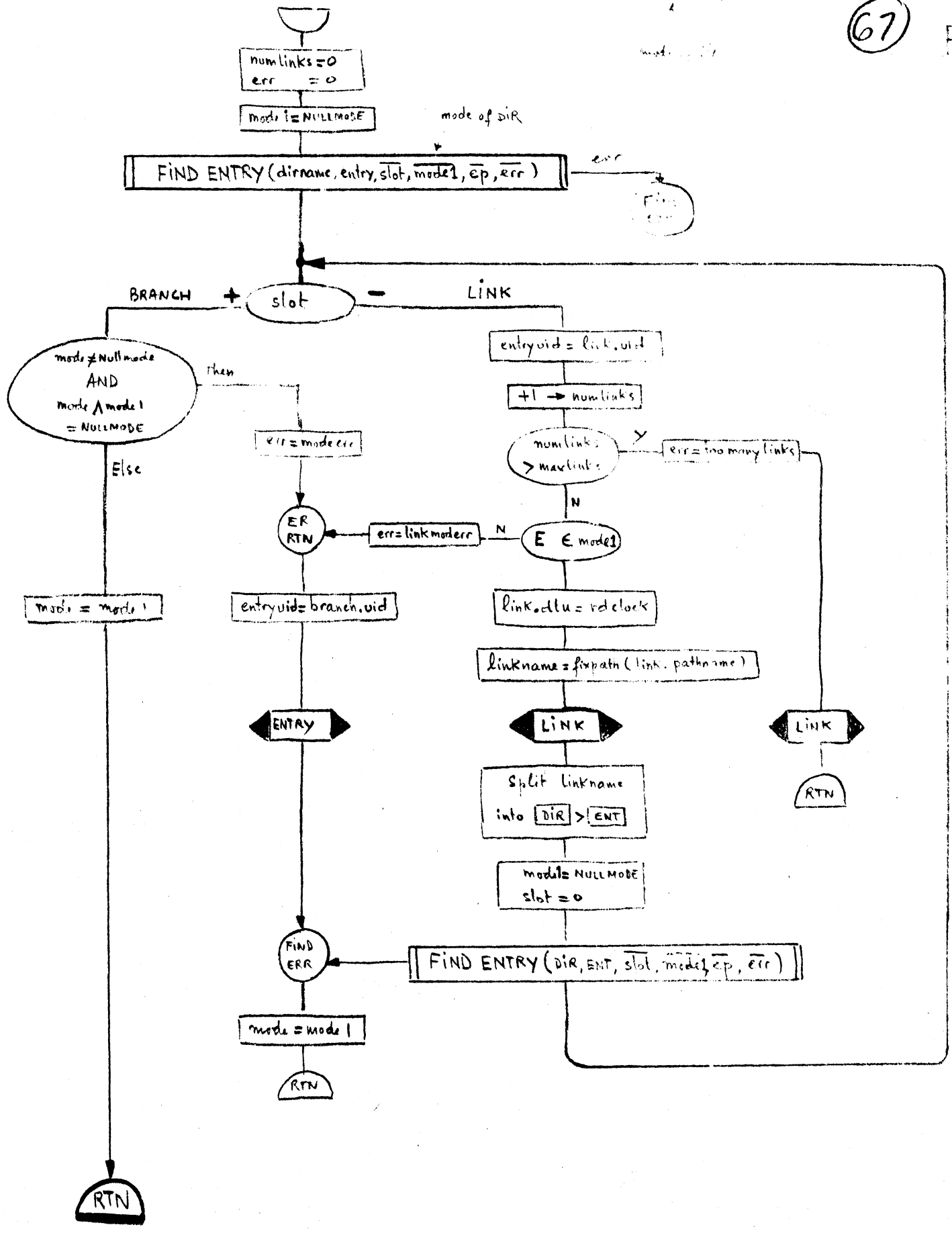


FINDDIR (dirname, seqptr, code)

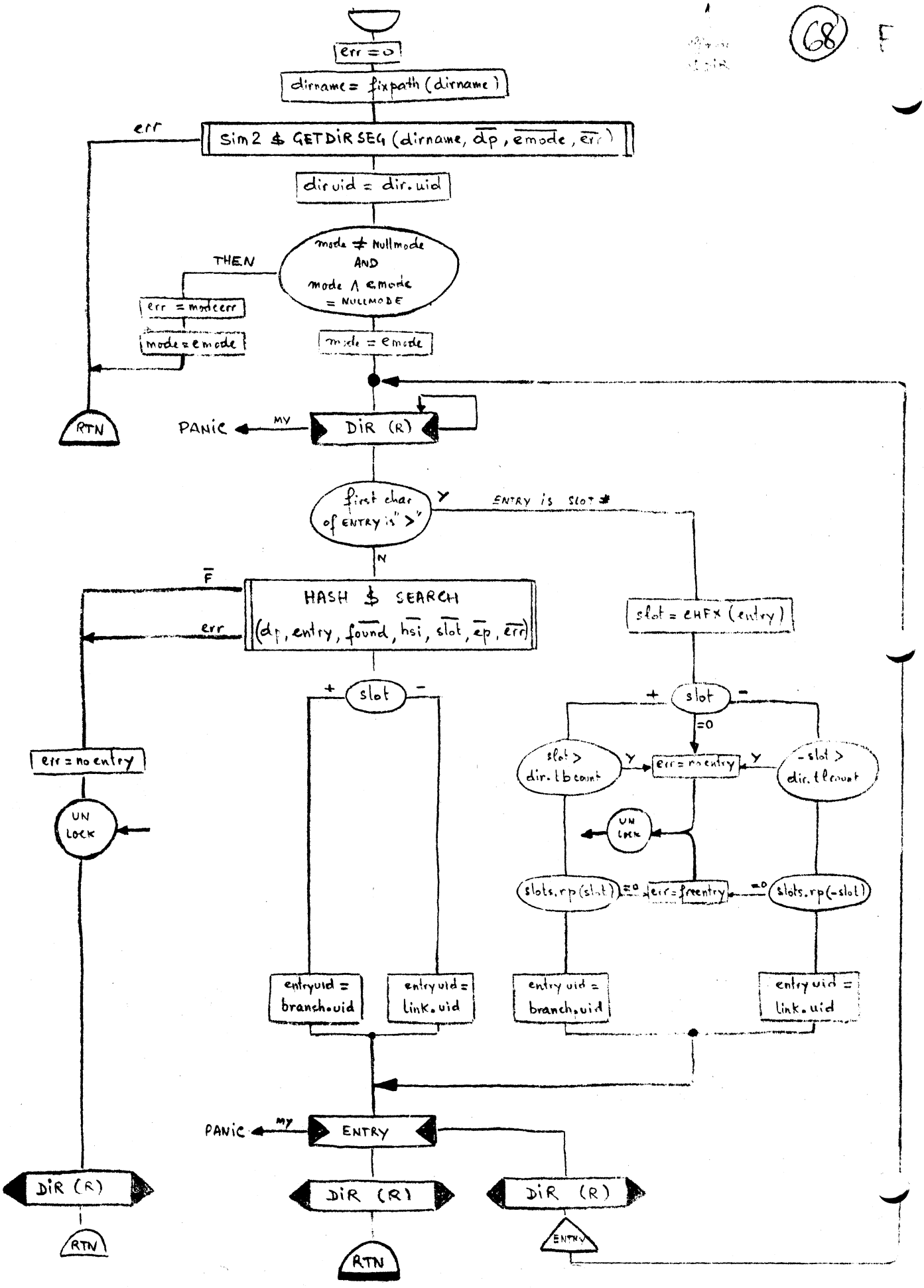
Finddir is called only by sim2 & getdirseg



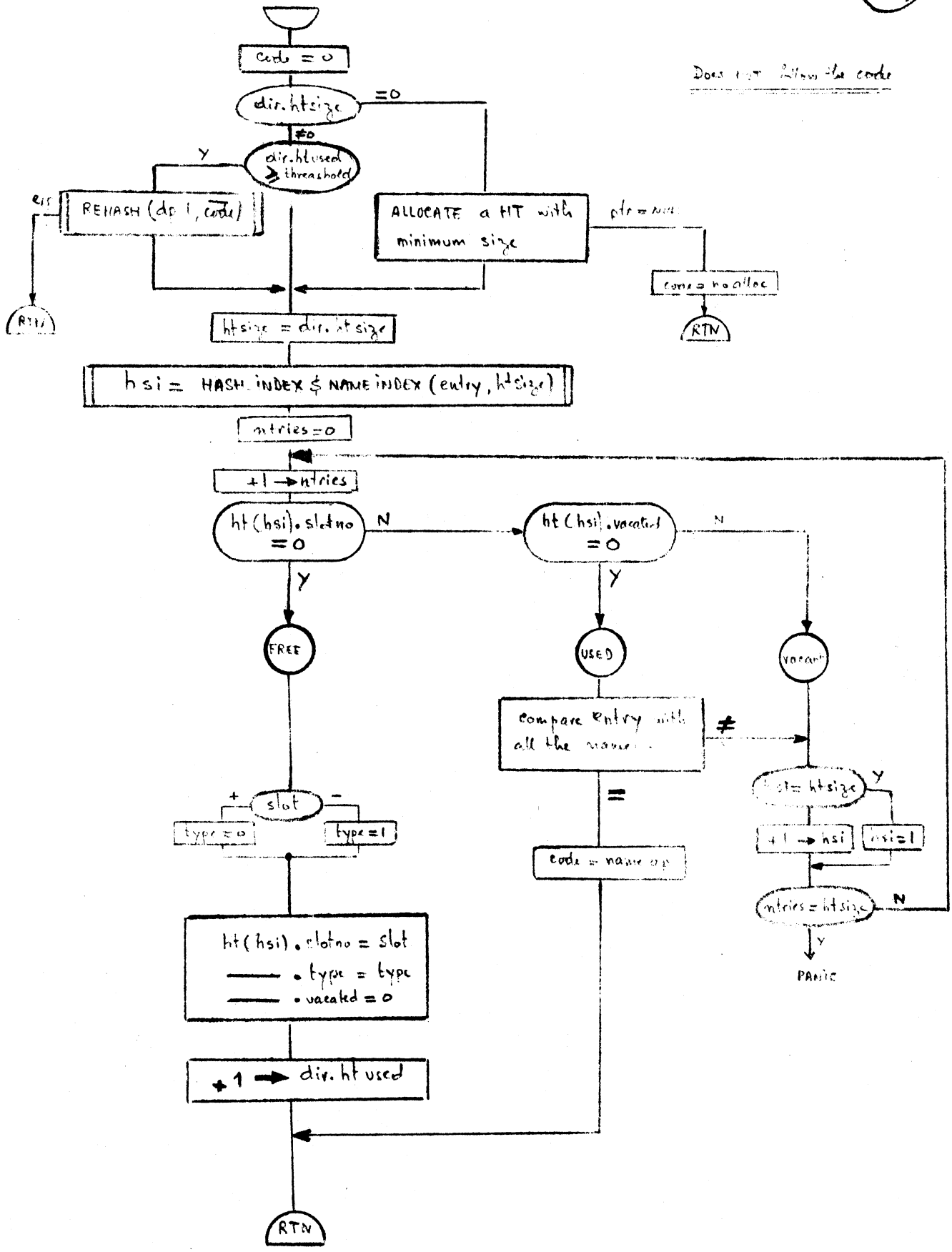
FIND BRANCH (dirname, entry, slot, mode1, ep, err)



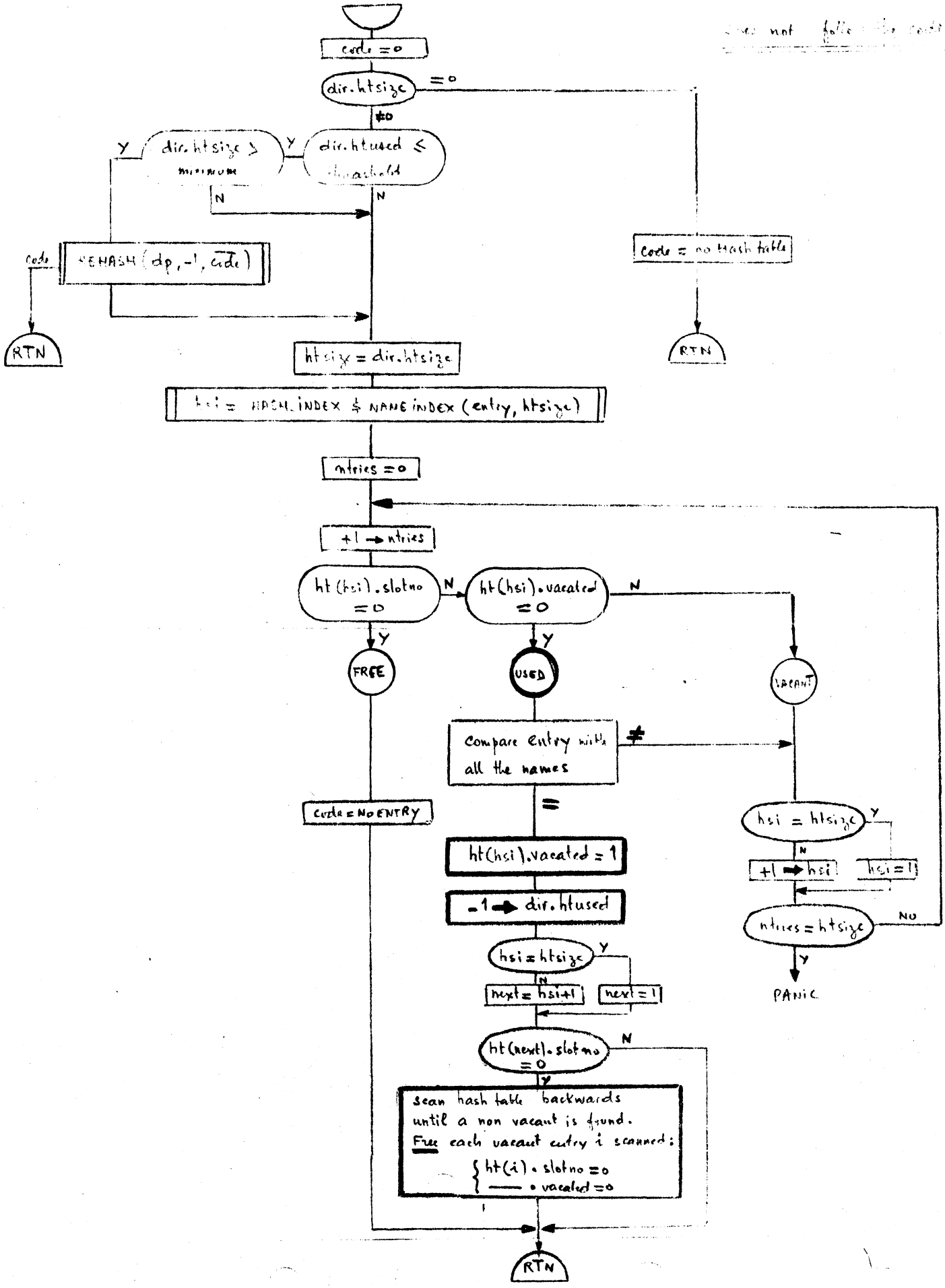
FIND ENTRY (dirname, entry, slot, mode, ep, err)



HASH$IN (dp, entry, slot, code)

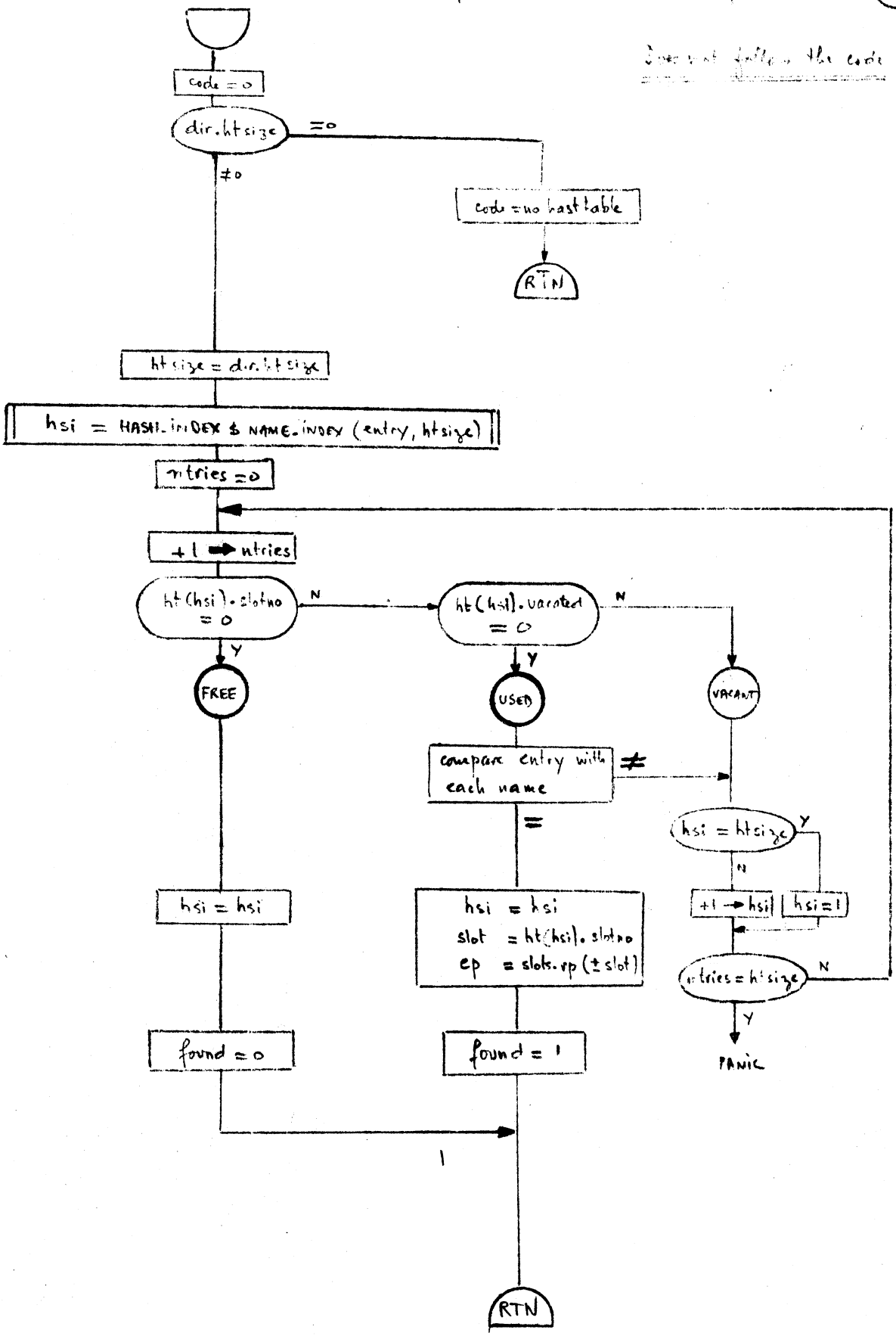


HASH $ OUT (dp, entry, code)

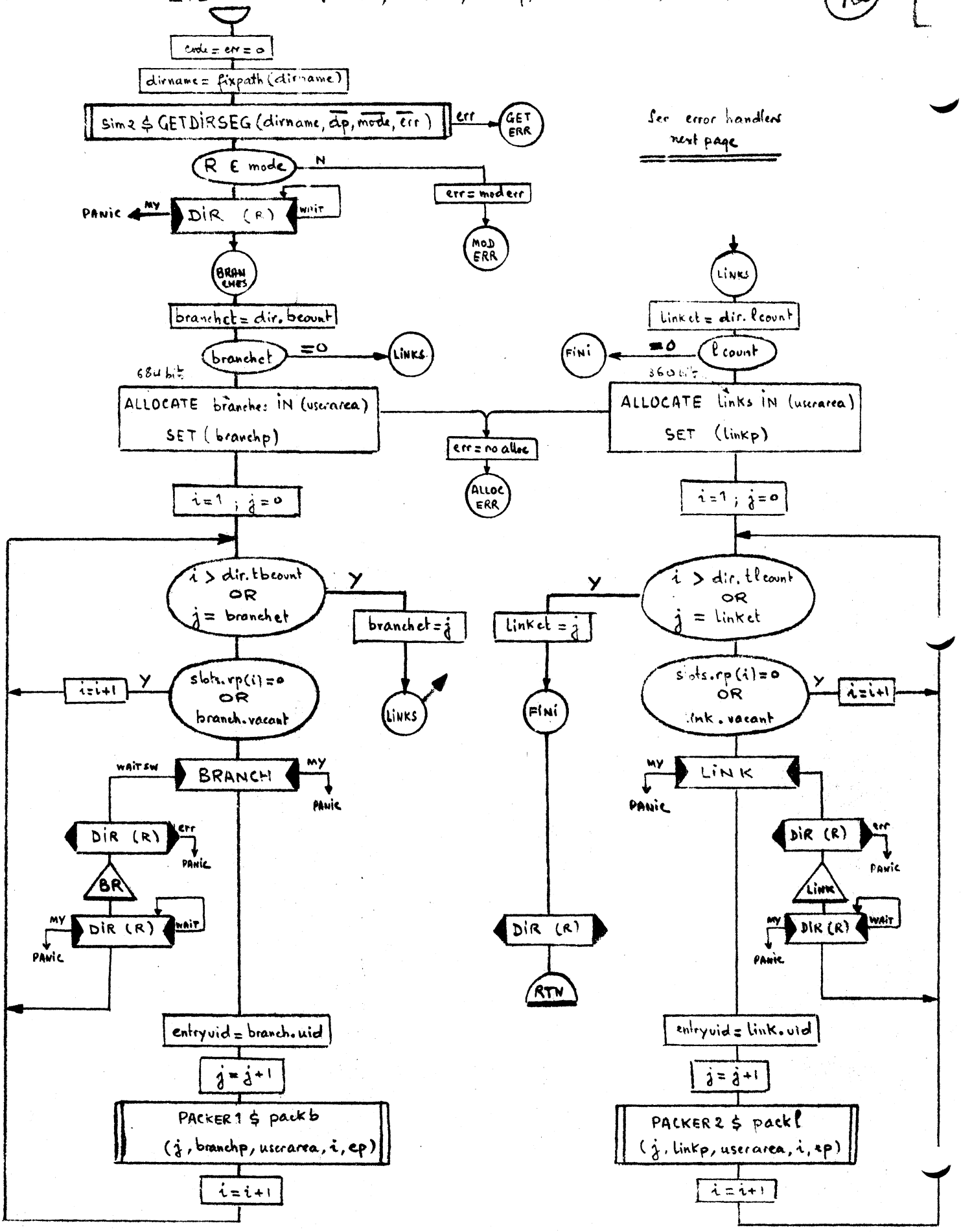


HASH & SEARCH (dp, entry, found, hsi, slot, ep, code)

Does not follow the code

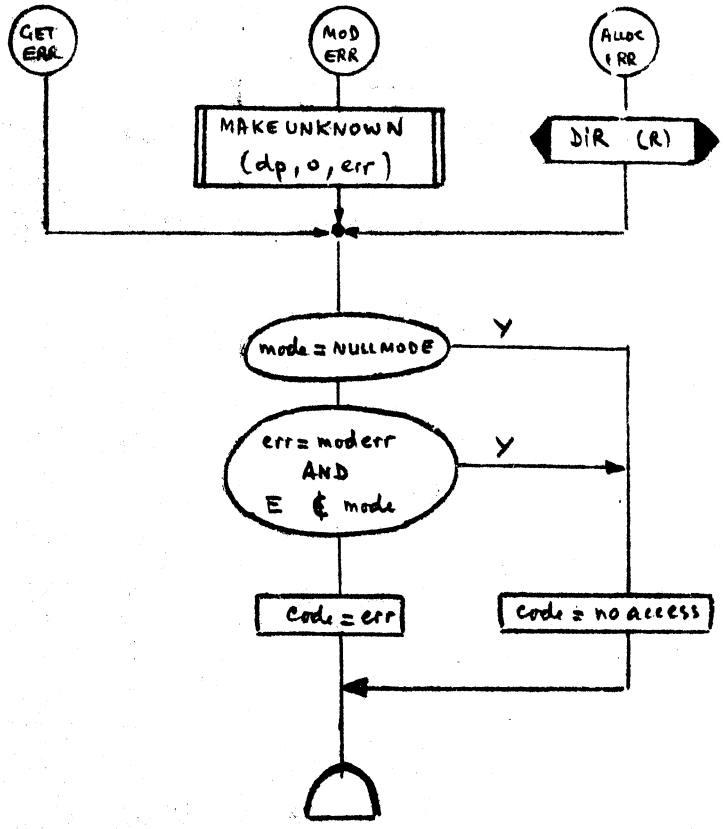


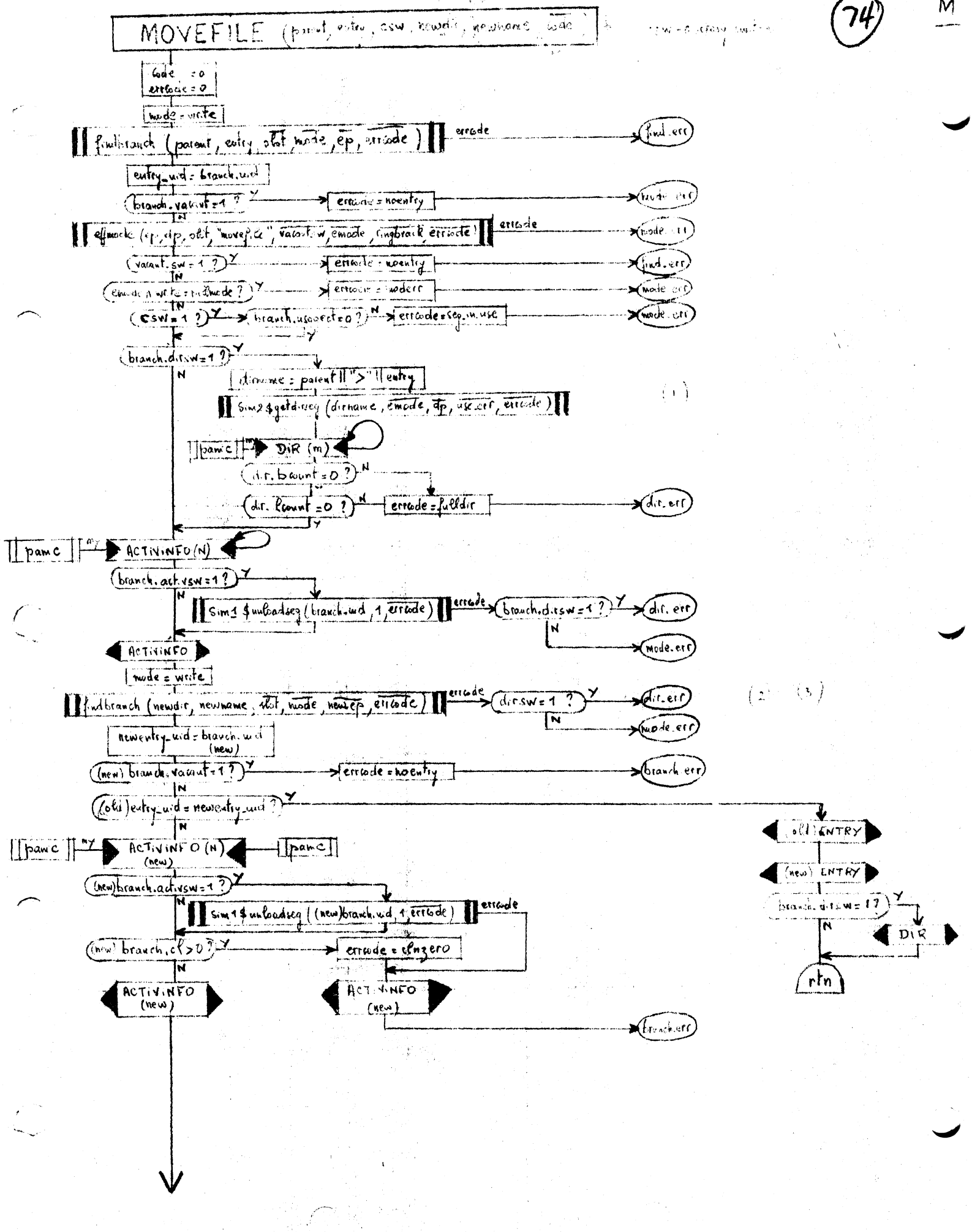
LIST DIR (dirname, userarea, branchp, branchct, linkp, linkct, code)

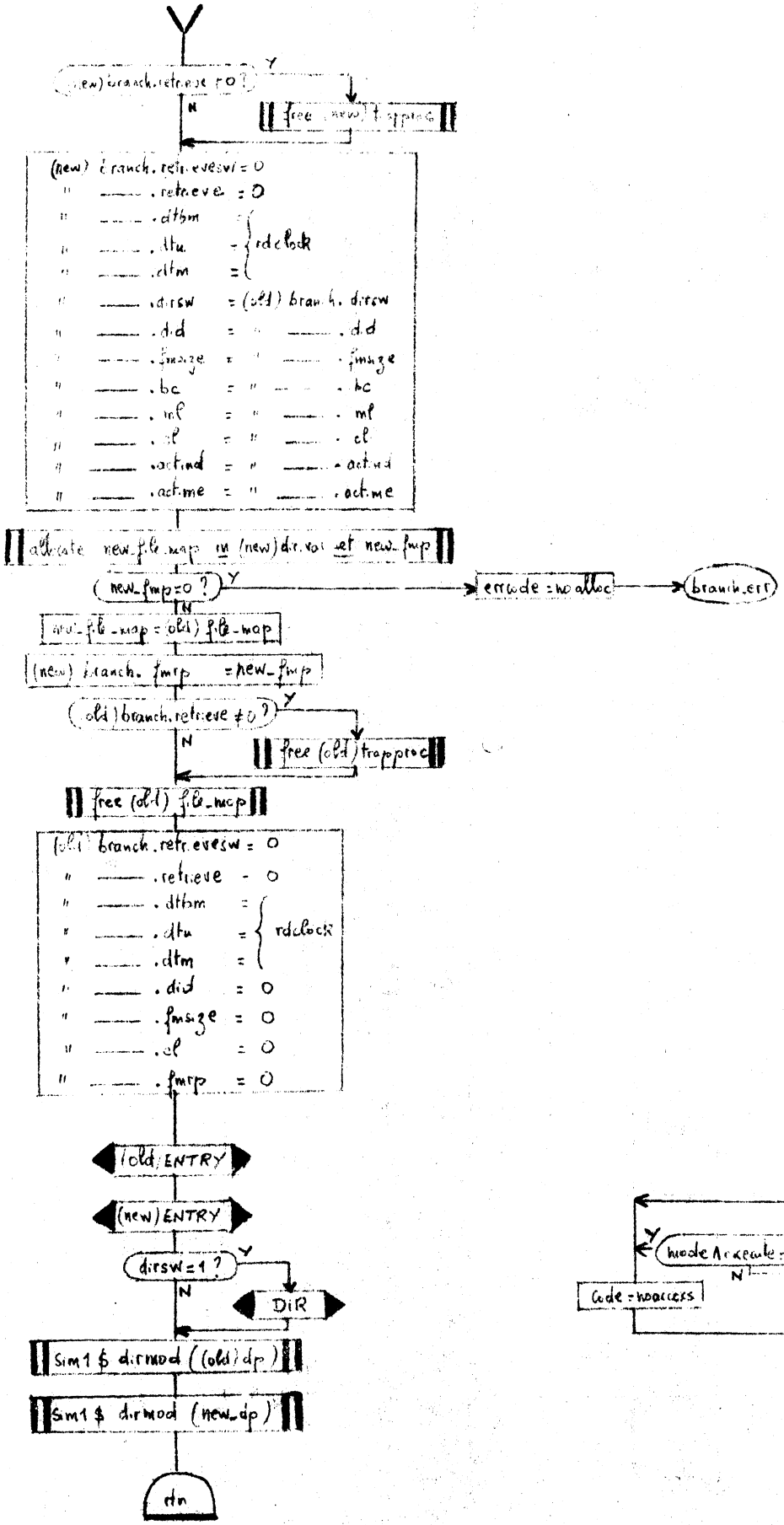


See error handlers next page

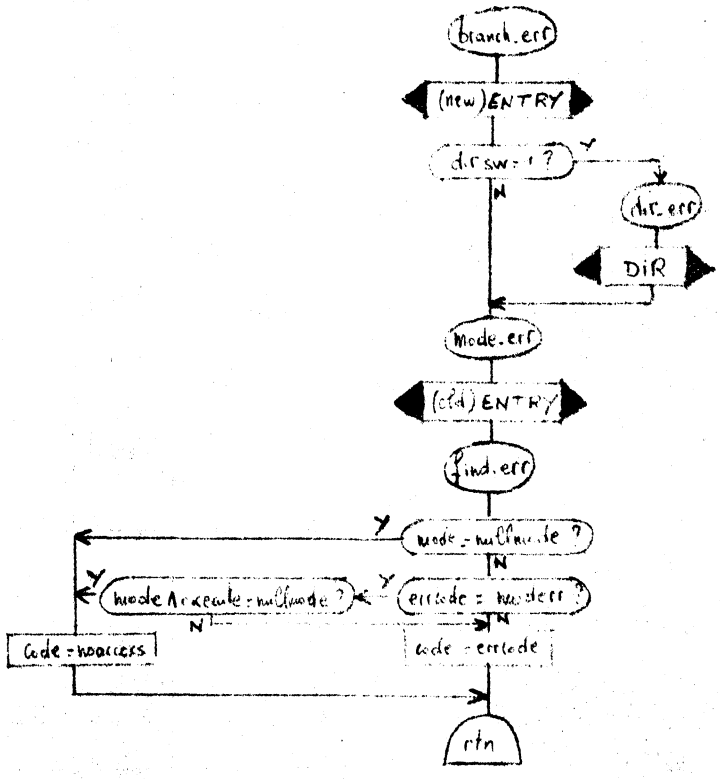
LISTDIR – ERROR HANDLERS

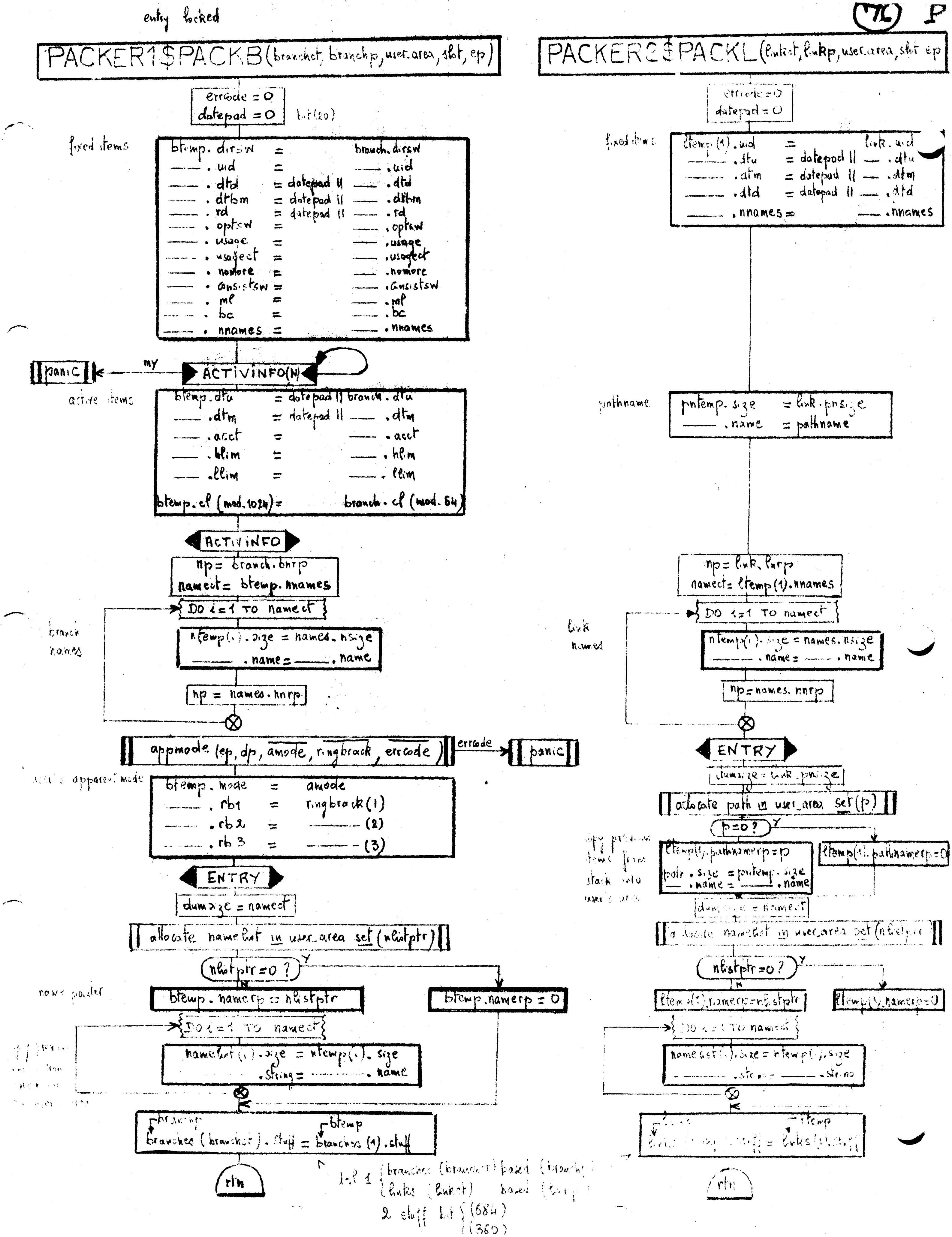


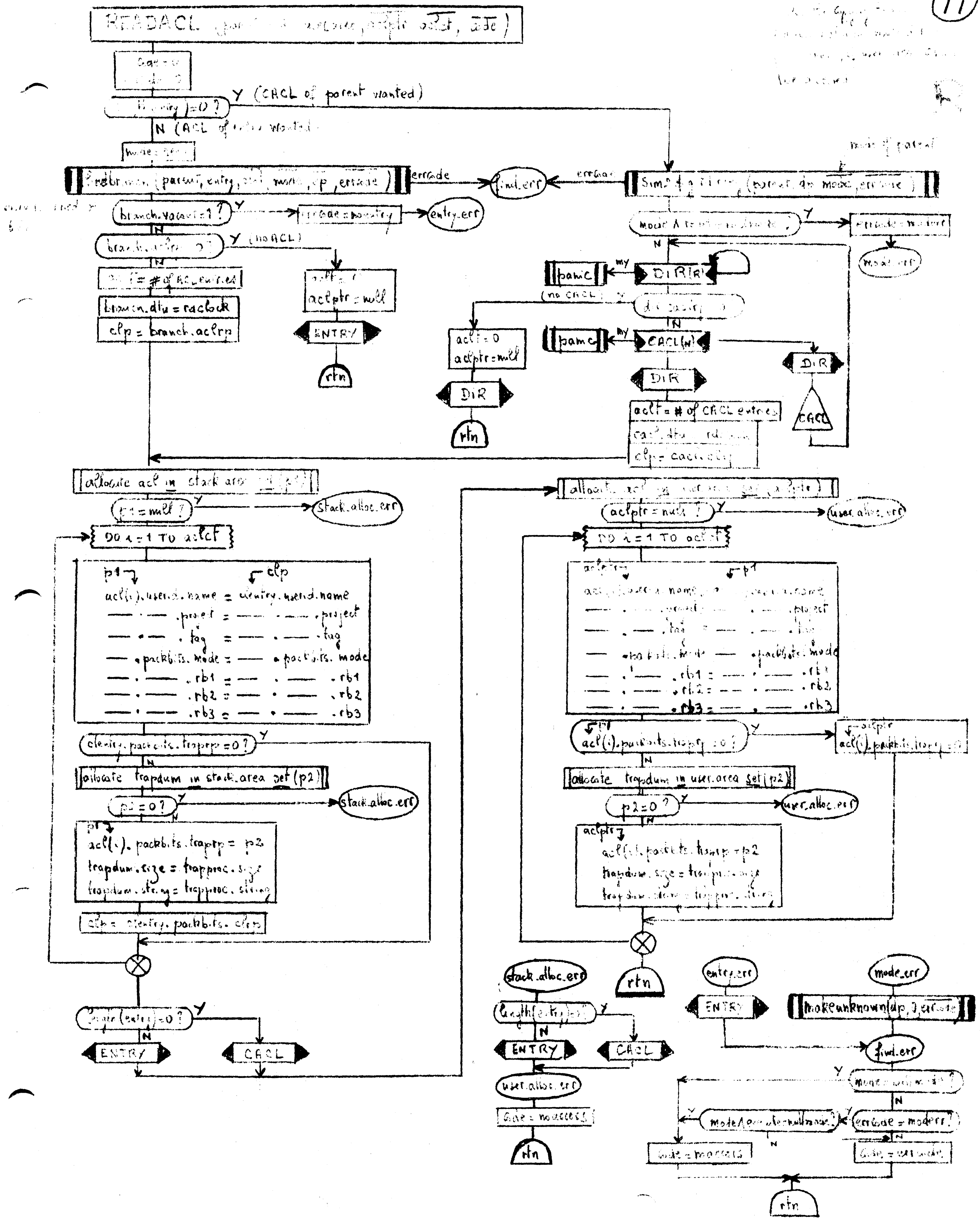

MOVEFILE (parent, entry, csw, newdir, newname, code)
csw = 0 crash switch
code = 0
errcode = 0
mode = write
findbranch (parent, entry, stat, mode, ep, errcode)
errcode
find.err
entry_uid = branch.uid
branch.variant = 1 ?
Y
errcode = noentry
branch.err
N
effmode (ep, dp, stat, "movefile", variant.w, emode, cngbranch, errcode)
errcode
mode.err
variant.sw = 1 ?
Y
errcode = noentry
find.err
N
errcode = write ? mdmode ?
Y
errcode = wderr
mode.err
N
csw = 1 ?
Y
branch.usercnt = 0 ?
N
errcode = seg_in_use
mode.err
Y
branch.dir.sw = 1 ?
Y
dirname = parent || ">" || entry
Sim2$getdirseg (dirname, emode, dp, user.err, errcode)
(1)
panic
m
DIR (m)
dir.bcount = 0 ?
N
dir.lcount = 0 ?
N
errcode = fuldir
dir.err
Y
panic
m
ACTIVINFO (N)
branch.activsw = 1 ?
Y
N
Sim1$unloadseg (branch.ud, 1, errcode)
errcode
branch.d.rsw = 1 ?
Y
dir.err
N
mode.err
ACTIVINFO
mode = write
findbranch (newdir, newname, stat, mode, newep, errcode)
errcode
dir.sw = 1 ?
Y
dir.err
(2)
(3)
N
mode.err
newentry_uid = branch.uid (new)
(new) branch.variant = 1 ?
Y
errcode = noentry
branch.err
N
(old) entry_uid = newentry_uid ?
Y
old ENTRY
N
panic
m
ACTIVINFO (N) (new)
panic
(new) ENTRY
(new) branch.activsw = 1 ?
Y
branch.d.rsw = 1 ?
Y
N
Sim1$unloadseg ((new)branch.ud, 1, errcode)
errcode
DIR
(new) branch.cl > 0 ?
Y
errcode = clnzero
N
rtn
ACTIVINFO (new)
ACTIVINFO (new)
branch.err



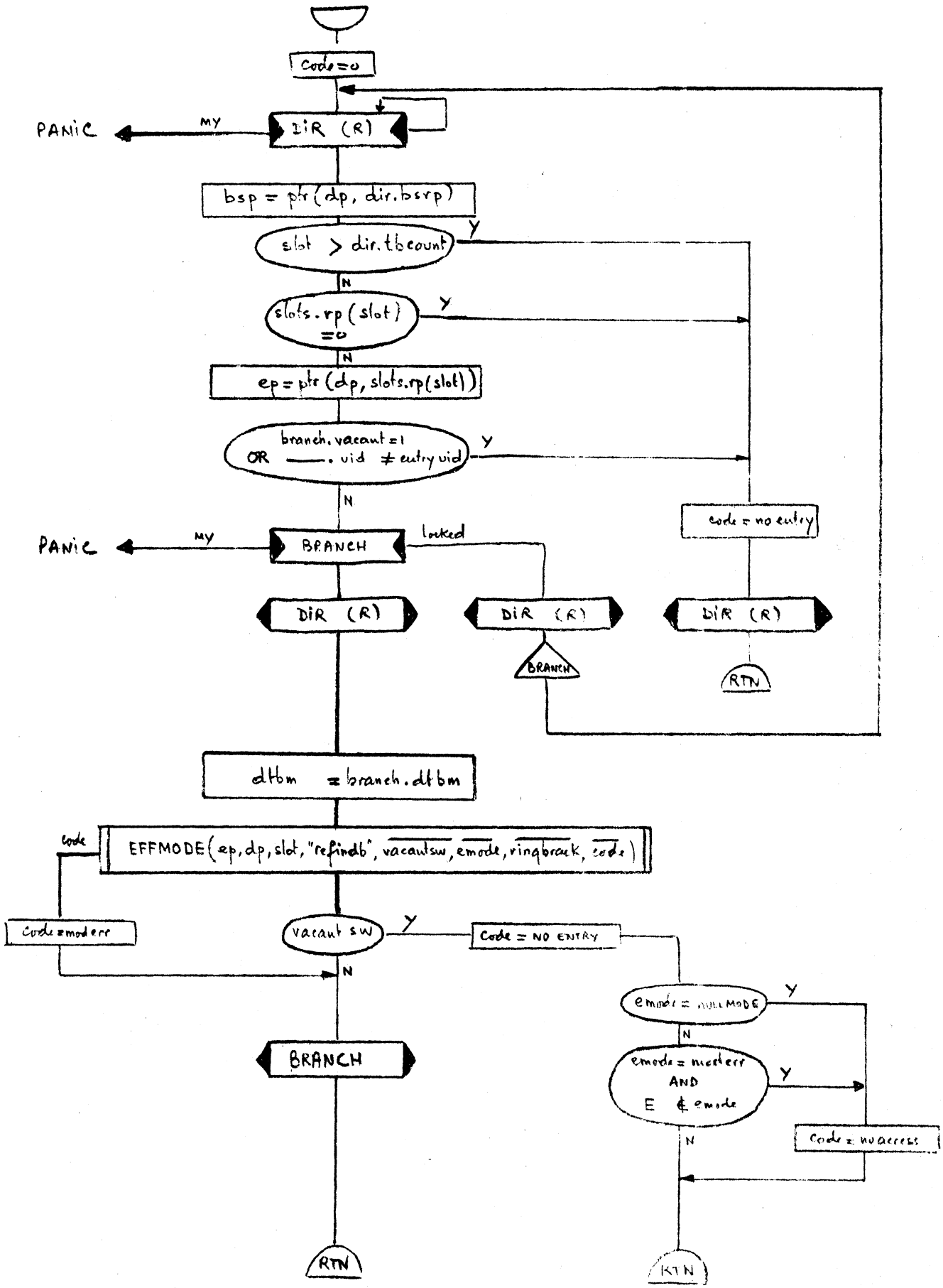
errors handlers

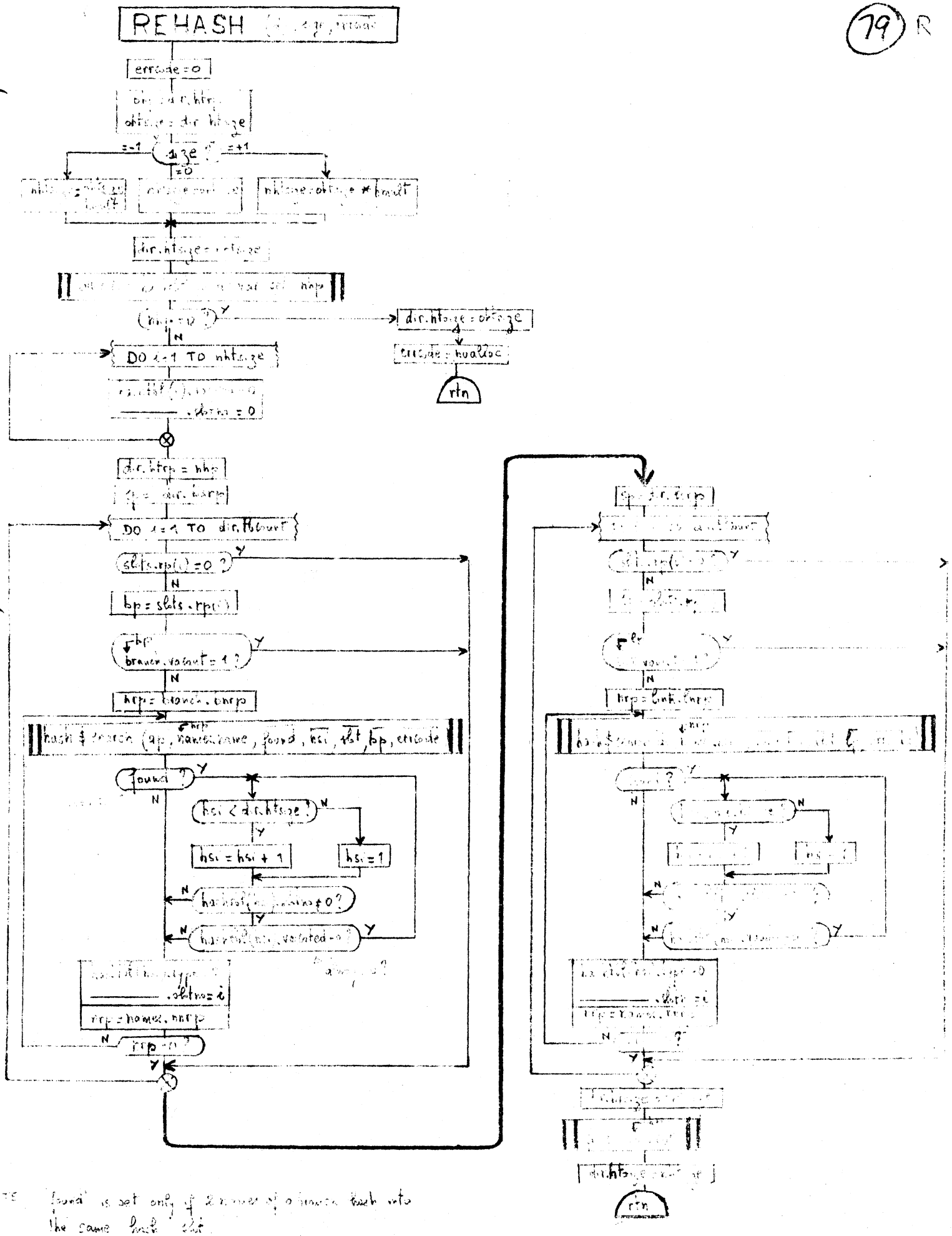



entry locked
PACKER1$PACKB(branchet, branchp, user_area, slot, ep)
errcode = 0
datepad = 0
bit(20)
fixed items
btemp.dirsw = branch.dirsw
.uid = .uid
.dtd = datepad || .dtd
.dtbm = datepad || .dtbm
.rd = datepad || .rd
.optsw = .optsw
.usage = .usage
.usagect = .usagect
.nomore = .nomore
.consistsw = .consistsw
.ml = .ml
.bc = .bc
.nnames = .nnames
panic
my
ACTIVINFO(N)
active items
btemp.dtu = datepad || branch.dtu
.dtm = datepad || .dtm
.acct = .acct
.hlim = .hlim
.llim = .llim
btemp.cl (mod.1024) = branch.cl (mod.64)
ACTIVINFO
np = branch.bnrp
namect = btemp.nnames
branch names
DO i=1 TO namect
ntemp(i).size = names.nsize
.name = .name
np = names.nnrp
appmode (ep, dp, amode, ringbrack, errcode)
errcode
panic
user's apparent mode
btemp.mode = amode
.rb1 = ringbrack(1)
.rb2 = (2)
.rb3 = (3)
ENTRY
dumsize = namect
allocate namelist in user_area set (nlistptr)
nlistptr = 0 ?
Y
N
name pointer
btemp.namerp = nlistptr
btemp.namerp = 0
DO i=1 TO namect
namelist(i).size = ntemp(i).size
.string = .name
branches (branchct).stuff = branches (1).stuff
branchp
btemp
rtn
PACKER2$PACKL(linkct, linkp, user_area, slot ep)
errcode = 0
datepad = 0
fixed items
ltemp(1).uid = link.uid
.dtu = datepad || .dtu
.dtm = datepad || .dtm
.dtd = datepad || .dtd
.nnames = .nnames
pathname
pntemp.size = link.pnsize
.name = pathname
np = link.lnrp
namect = ltemp(1).nnames
link names
DO i=1 TO namect
ntemp(i).size = names.nsize
.name = .name
np = names.nnrp
ENTRY
dumsize = link.pnsize
allocate path in user_area set (p)
p=0 ?
Y
copy pathname items from stack into user's area
ltemp(1).pathnamerp = p
ltemp(1).pathnamerp = 0
path.size = pntemp.size
.name = .name
dumsize = namect
allocate namelist in user_area set (nlistptr)
nlistptr = 0 ?
Y
N
ltemp(1).namerp = nlistptr
ltemp(1).namerp = 0
DO i=1 TO namect
namelist(i).size = ntemp(i).size
.string = .string
ltemp
rtn
1 { branches (branchct) based (branchp)
{ links (linkct) based (linkp)
2 stuff bit { (684)
{ (360)


READACL
Y (CACL of parent wanted)
N (ACL
errcode
find.err
branch.vacant=1 ?
entry.err
(no ACL)
of ACL entries
branch.dtu = raclock
clp = branch.aclrp
aclptr = null
ENTRY
rtn
mode_err
DIR(R)
panic
(no CACL)
CACL(W)
DIR
acl = 0
aclptr = null
acl = # of CACL entries
CACL
allocate acl in stack area
stack.alloc.err
DO i = 1 TO aclcf
p1
clp
acl(i).userid.name = clentry.userid.name
.project
.tag
.packbits.mode
.rb1
.rb2
.rb3
allocate trapdum in stack.area set (p2)
acl(i).packbits.traprp = p2
trapdum.size = trapproc.size
trapdum.string = trapproc.string
clp = clentry.packbits.clrp
aclptr = null ?
user.alloc.err
DO i = 1 TO aclcf
aclptr
allocate trapdum in user.area set (p2)
p2 = 0 ?
acl(i).packbits.traprp = p2
entry.err
makeunknown(dp, 0, errcode)
errcode = moderr ?
ENTRY
CACL
Y
N

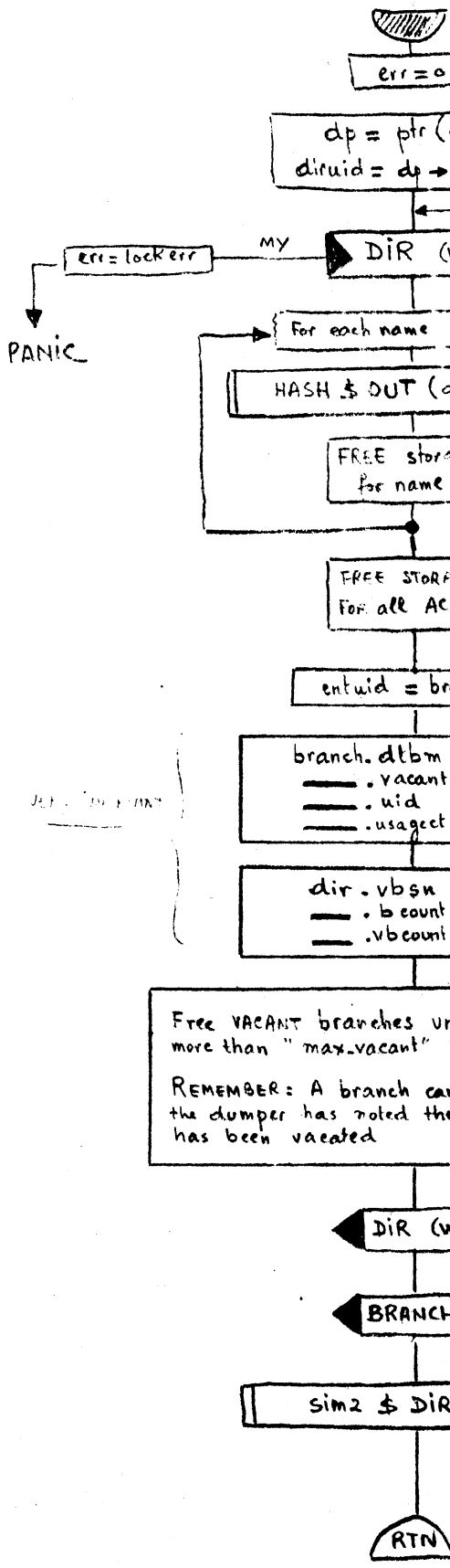
REFINDB (dp, slot, entryuid, dtbm, emode, ringbrack, code)



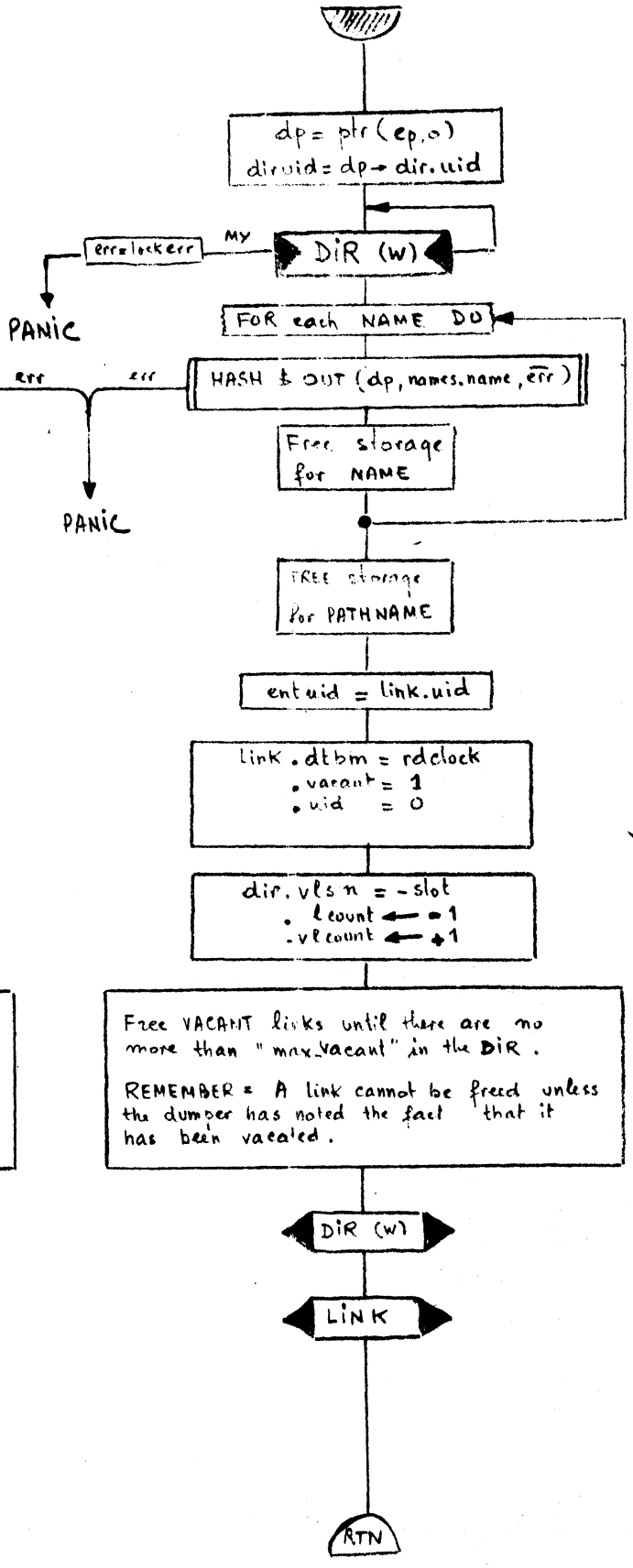


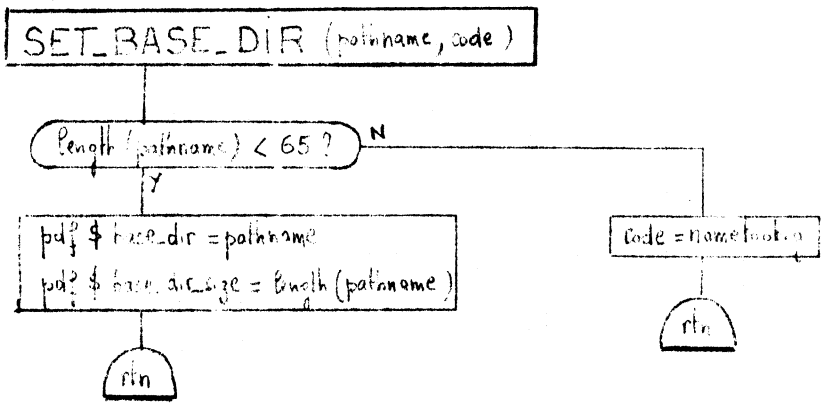
NOTE 'found' is set only if a browser finds an entry in the same hash slot

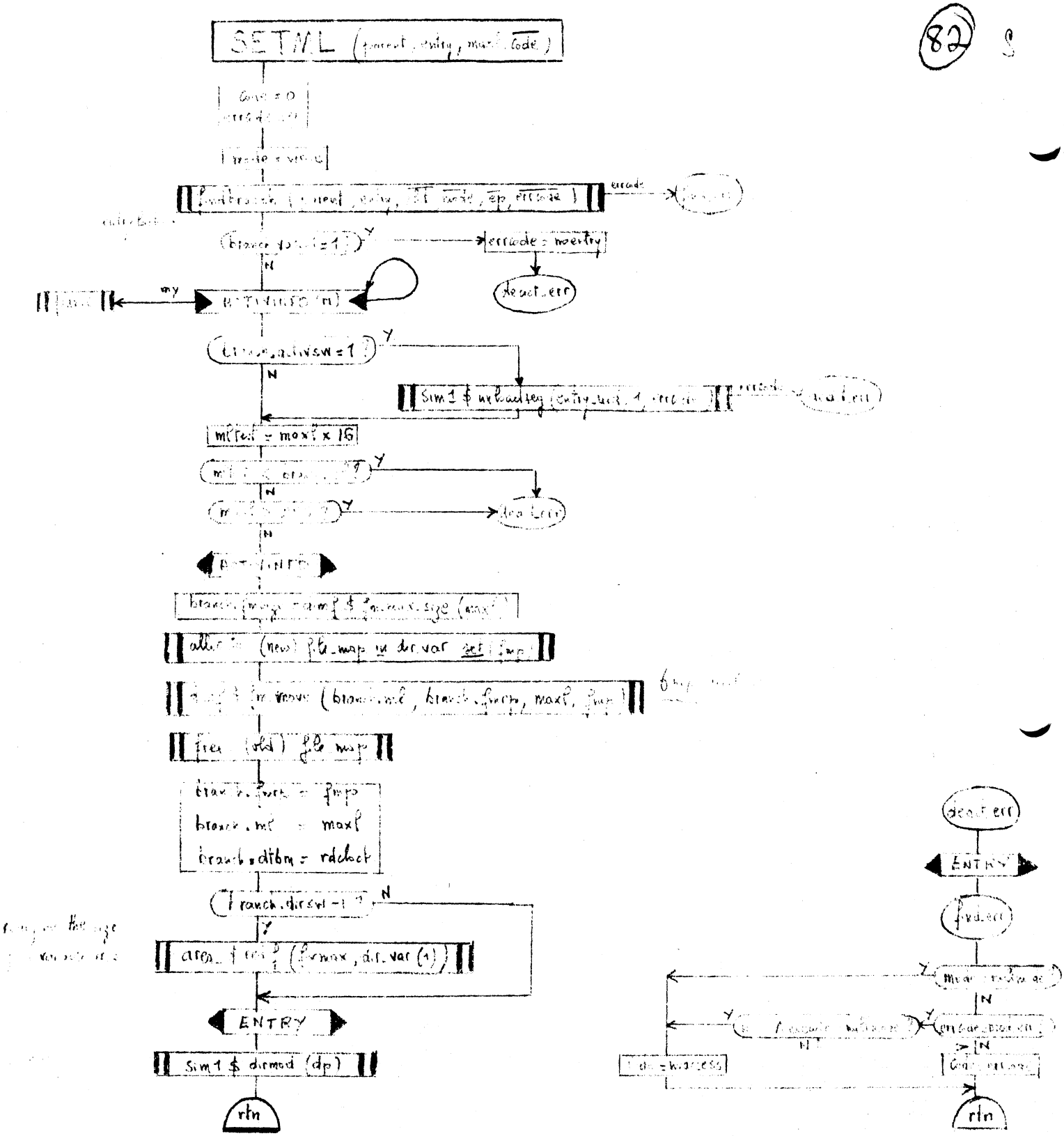
REMOVE B (ep, slot)



REMOVE L (ep, slot)

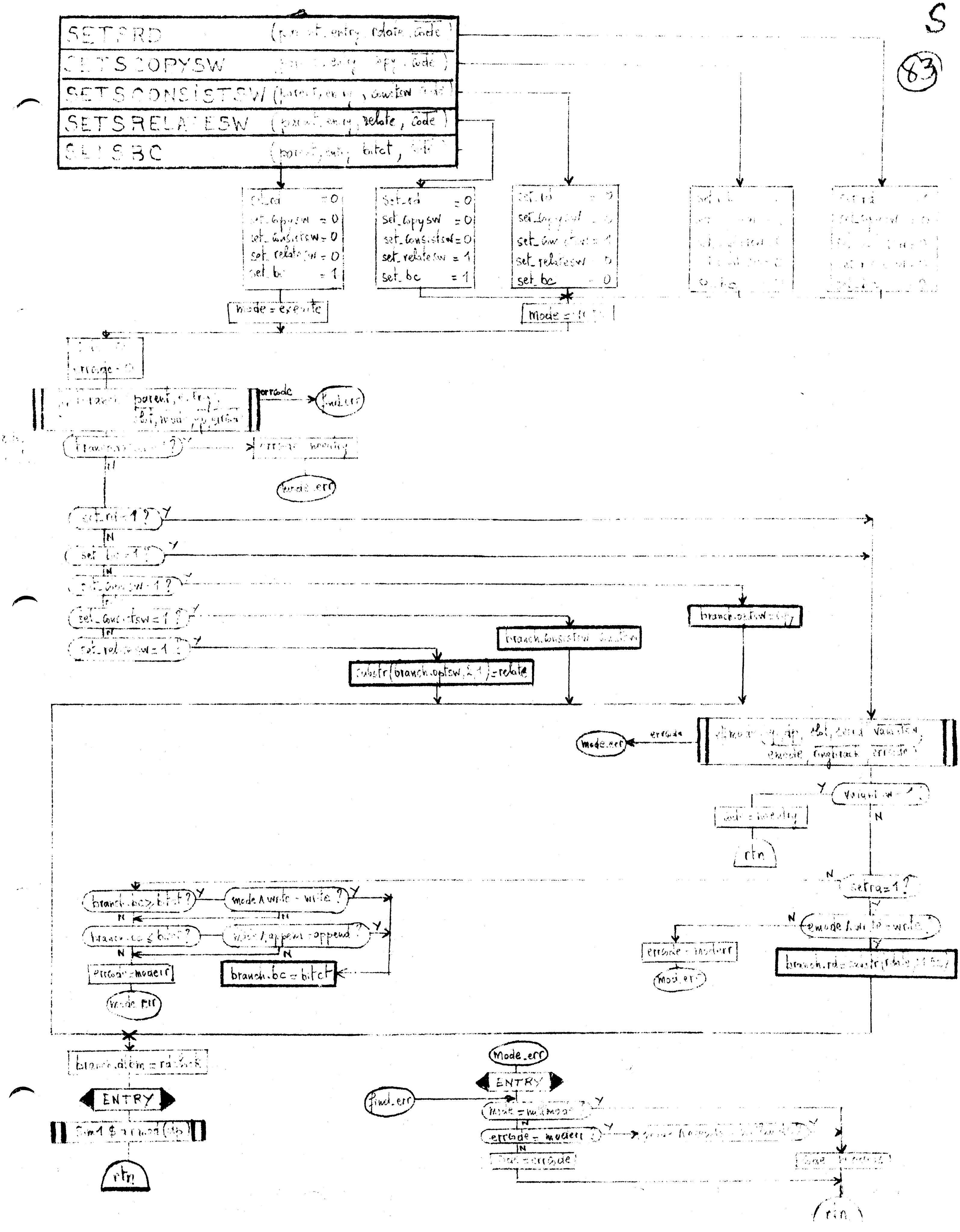


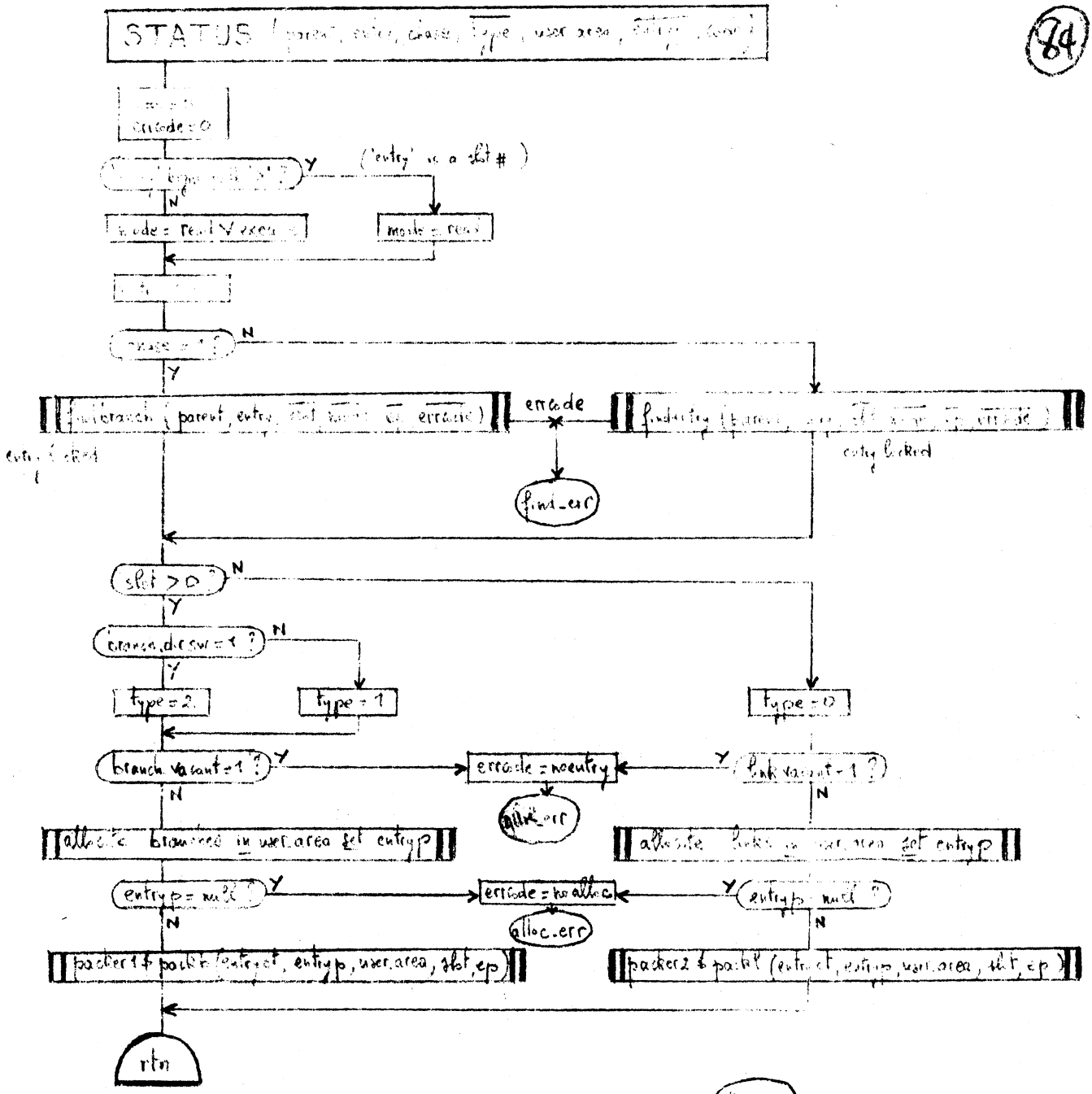

SET_BASE_DIR (pathname, code)
length (pathname) < 65 ?
N
Y
pd? $ base_dir = pathname
pd? $ base_dir_size = length (pathname)
code = name too long
rtn
rtn


SETML
ENTRY
rtn
ENTRY
rtn

S

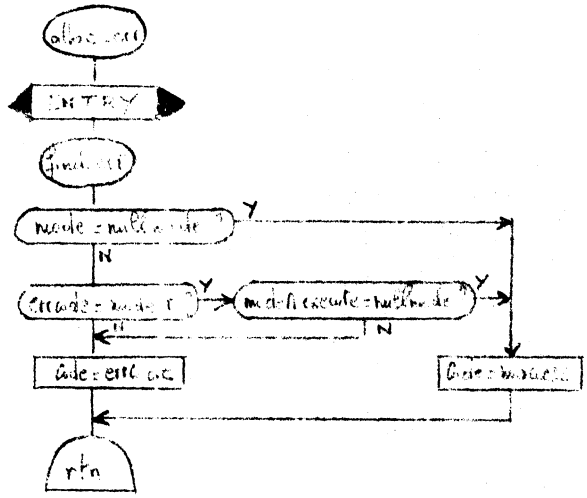
83

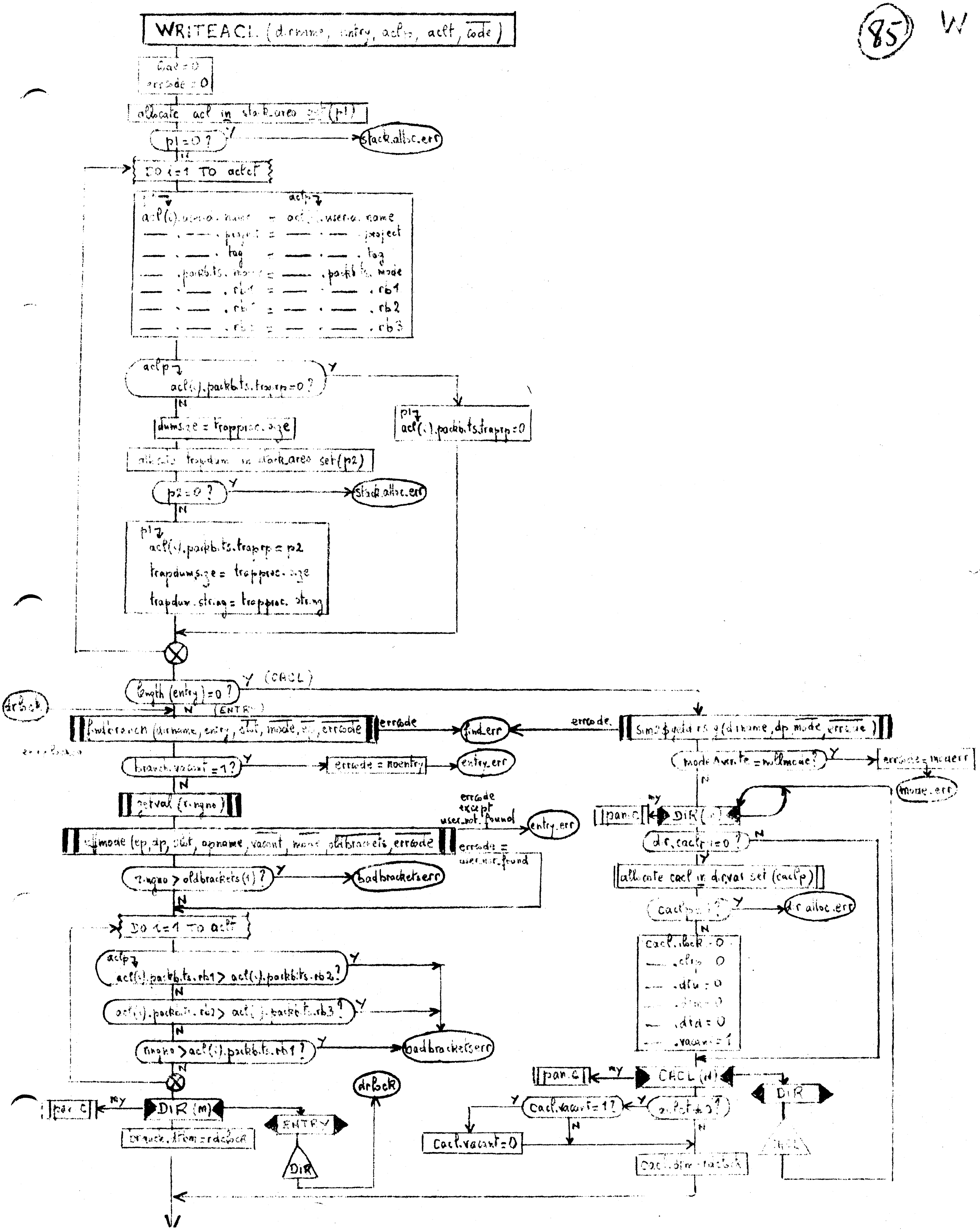

SETSRD
SETSCOPYSW
SETSCONSISTSW
SETSRELATESW
SETSBC
set_copysw = 0
set_consistsw = 0
set_relatesw = 0
set_bc = 1
set_relatesw = 1
set_bc = 0
mode = execute
errcode
find.err
mode.err
N
Y
substr(branch.optsw, 2, 1) = relate
branch.bc ≥ bitct ?
mode ∧ write = write ?
branch.bc = bitct
errcode = moderr
errmode
setra = 1 ?
ENTRY
rtn
Mode_err
errcode = moderr

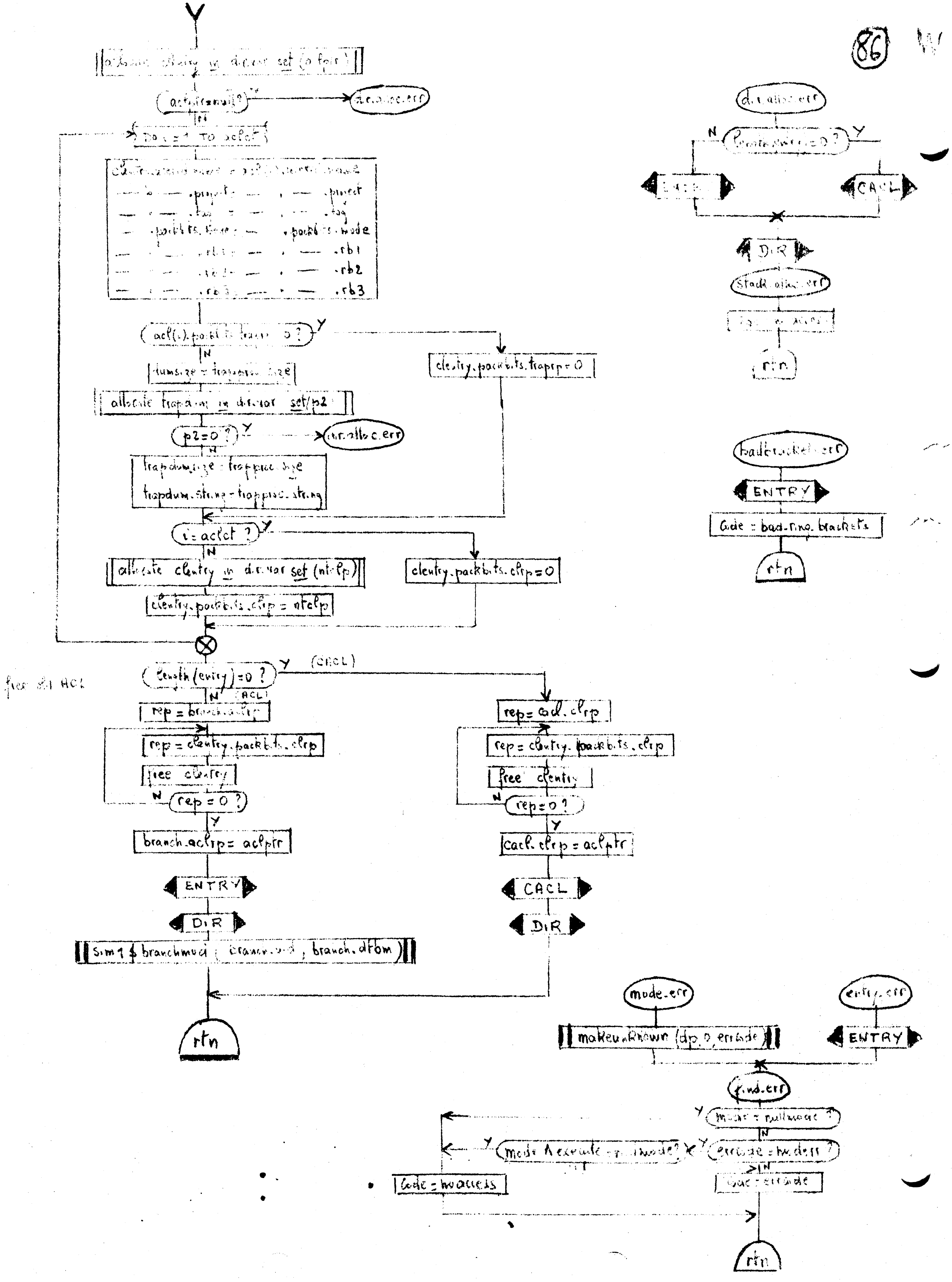


chase { =0 return contents of 'entry' whatever it is (branch or link)
=1 return contents of branch pointed to if 'entry' is a link

type { =0 link
=1 non-directory branch
=2 directory branch




WRITEACL (dirname, entry, aclp, aclt, code)
code = 0
errcode = 0
allocate acl in stack_area set (p1)
p1 = 0 ?
stack.alloc.err
DO i=1 TO aclt
acl(i).userid.name = acl(i).userid.name
.project
.tag
.packbits.mode
.rb1
.rb2
.rb3
acl(i).packbits.trapp=0 ?
acl(i).packbits.trapp=0
dumsize = trapproc.size
allocate trapdum in stack_area set (p2)
p2 = 0 ?
stack.alloc.err
acl(i).packbits.trapp = p2
trapdumsize = trapproc.size
trapdum.string = trapproc.string
length (entry) = 0 ?
(CACL)
(ENTRY)
drlock
findbranch (dirname, entry, slot, mode, ep, errcode)
errcode
find.err
sim$getdirseg (dirname, dp, mode, errcode)
branch.vacant = 1 ?
errcode = noentry
entry.err
mode
mode.err
getval (ringno)
errcode except user.not.found
entry.err
acl(i).mode (ep, dp, slot, opname, vacant, mode, oldbrackets, errcode)
errcode = user.not.found
ringno > oldbrackets(1) ?
badbracketserr
DIR
dir.caclp = 0 ?
allocate cacl in dirval set (caclp)
caclp = 0 ?
dir.alloc.err
cacl.lock = 0
.dtu = 0
.dtm = 0
.dtd = 0
.vacant = 1
DO i=1 TO aclt
acl(i).packbits.rb1 > acl(i).packbits.rb2 ?
acl(i).packbits.rb2 > acl(i).packbits.rb3 ?
ringno > acl(i).packbits.rb1 ?
badbracketserr
pan.c
CACL
DIR
cacl.vacant = 1 ?
cacl.vacant = 0
pan.c
DIR (m)
ENTRY
DIR
drlock



Y
DO i = 1 TO aclct
.project
.tag
.packbits.mode
.rb1
.rb2
.rb3
N
Y
dumsize = trapproc size
allocate trapdum in dir.var set/p2
p2=0 ?
trapdum.size = trapproc.size
trapdum.string = trapproc.string
clentry.packbits.traprp = 0
i = aclct ?
allocate clentry in dir.var set (ntclp)
clentry.packbits.clrp = ntclp
clentry.packbits.clrp = 0
length (entry) = 0 ?
(CACL)
(ACL)
rep = branch.aclrp
rep = clentry.packbits.clrp
free clentry
rep = 0 ?
branch.aclrp = aclptr
ENTRY
DIR
rep = cacl.clrp
cacl.clrp = aclptr
CACL
rtn
CACL
stack.alloc.err
badbracket.err
code = bad_ring_brackets
mode.err
entry.err
makeunknown (dp, 0, errcode)
find.err
mode = nullmode ?
mode ∧ execute = nullmode ?
code = noaccess
code = errcode