

AUCXOD, SPL CODE

```
%  
file auxcod(orgaux)  
%freeze control%  
  (+qkf) +specit,prmspc CALL frzsav +ltspec,prmspc CALL frzst display()  
  GOTO [s]  
  (+qka) CALL relall display() GOTO [s]  
%pointer control%  
  (+qplsl) CALL seeptr RETURN  
  (+qpml) CALL delptn RETURN  
END of auxcod
```

AUCXOD, MOL CODE

%

%pointers%

```
(delptn) PROCEDURE;
  delpt1 ← $1/lit11/;
  delpt2 ← 0;
  WHILE
    delpt2 < ptrtbl
    AND ptrtb/delpt2/ # delpt1
    DO delpt2 ←+ 3;
  IF ptrtb/delpt2/ # delpt1 DO-SINGLE !abort $5;
  delpt2 ←+ 3;
  WHILE delpt2 <= ptrtbl DO BEGIN
    ptrtb[.XR-3/ ← ptrtb/delpt2/;
    BUMP delpt2 END;
  ptrtbl ←+ -3;
  RETURN END.
```

```
(seeptr) PROCEDURE;
  clrdpy(0); BUMP opnfg;
  FOR seept1 FROM 0 INC 3 TO ptrtbl DO BEGIN
    seept2←LSH $8(0,ptrtb/seept1/);
    seept3←LSH $8(0,.BR);
    seept4←LSH $8(0,.BR);
    EXU prmspc; %for calls on aplit%
    CALL aplit(seept2);
    CALL aplit(seept3);
    CALL aplit(seept4);
    CALL aplit($asccr) END;
  opnfg ← 0; RETURN END.
```

%frozen statements%

```
(frzst) PROCEDURE ;
  DECLARE frzst2 = (17,17,"FROZEN TABLE FULL");
  %frzst1 = psid %
  frzst1 ← spskd;
  .XR ← $frzlst;
  WHILE .XR <= frzlsx DO BEGIN
    IF NOT $0[.XR/ OR .AR .A 3777B = frzst1 THEN BEGIN
      .AR ← RSH $1(cdblnk,0);
      $0[.XR/ ← LSH $12(cdtrn) .V frzst1;
      cdtrn ← cdtrnl;
      cdlev ← cdlevl;
      cdblnk ← cdblnkl;
      dblev ← numdpy(cdlev);
      dlbtrn ← numdpy(cdtrn);
      cdrlev ← 0;
      RETURN END;
    EAX $1[.XR/ END;
    CALL dismes( $frzst2,2 );
    RETURN END.
```

```
(relall) PROCEDURE;
  relst1 ← $frzlst;
  WHILE relst1 <= frzlsx DO BEGIN
```

AUCXOD, MOL CODE

```

        /relst1/ ← 0;
        BUMP relst1 END;
    RETURN END.
(frzsav) PROC;
    cdlevl←cdlev;
    cdtrnl←cdtrn;
    cdblkl←cdblkn;
    RETURN END.
%status%
(sttus) PROCEDURE;
    EXTERNAL sttus0;
    (sttus0):
    EXECUTE prmshp; EXECUTE rfbp2;
    IF todas THEN sttus2←1 ELSE BEGIN
        IF NOT .SKIP BRS $1(,,11100000B .V $dpych) DO-SINGLE CALL
            rerror;
        sttus2← .AR; CALL clrall(0) END;
    BUMP opnfg;
    sttus1←BRS $143(,10010B,IF rffn THEN .AR ELSE curfn);
    IF rffn THEN BRS $34("$THIS A A WORKING COPY/",-1,stitus2);
    CIO sttus2(73B); BRS $34($nlsvwd,3,stitus2);
    CIO sttus2(14B); CIO sttus2(0);
    BRS $36(hedsz,10,stitus2); CIO sttus2(75B);
    IF todas THEN CALL crlf ELSE CIO sttus2(155B);
    BRS $34("$FILE SIZE: /",-1,stitus2);
    BRS $36(sttus1,10,stitus2);
    BRS $34(" WORDS, /",-1,stitus2);
    BRS $36(sttus1/256,10,stitus2);
    BRS $34(" DRUM BLOCKS/", -1,stitus2);
    usd3←usd4←$stn2*3+5;
    BRS $34("$FILE OWNER /",-1,stitus2);
    IF NOT .SKIP BRS $105(usd3,usd4,funo) THEN .BR ←.AR;
    BRS $35(.AR,.BR,stitus2);
    BRS $34(", CREATED /",-1,stitus2);
    CALL typdat(fcred);
    BRS $34("$LAST WRITTEN BY /",-1,stitus2);
    BRS $34($finit,3,stitus2);
    CIO sttus2(0); CALL typdat(lwdat);
    BRS $36(b1(lwtim),10,stitus2);
    IF b2(lwtim) ≤ 9 THEN CIO sttus2(20B);
    BRS $36(b2(lwtim),10,stitus2); CIO sttus2(32B);
    BRS $36(b3(lwtim),10,stitus2);
    BRS $34("$NAME DELIMITERS ARE /",-1,stitus2);
    CIO sttus2(7); CIO sttus2(namd11); CIO sttus2(7);
    BRS $34(" AND /",-1,stitus2);
    CIO sttus2(7); CIO sttus2(namd12); CIO sttus2(7);
    IF todas THEN CALL crlf ELSE CIO sttus2(155B);
    usd1←0;
    FOR sttus1 FROM 0 INC 1 TO rsvst1 DO
        IF rsvst[stitus1] = 0 THEN NULL
        ELSE BEGIN

```

AUCXOD, MOL CODE

```

        lodrsv(LSH $8(sttus1,0));
        usd1 ← LRSH $2($1/rfbs[rsvst[sttus1]]/-8) END;
BRS $36(usd1,10,sttus2);
BRS $34(" STATEMENTS IN FILE, APPROX. /",-1,sttus2);
BRS $36((rals-9)*3*usd1/7,10,sttus2);
BRS $34(" WORDS OF TEXT$ /",-1,sttus2);
BRS $36(tnsg,10,sttus2);
BRS $34(" STATEMENTS GENERATED, AVE LENGTH = /",-1,sttus2);
BRS $36((rals-9)*3,10,sttus2);
CALL used("$FILE BLOCKS: /",$rfbs,rfbst1);
CALL used("$TEXT BLOCKS: /",$sdbst,sdbst1);
CALL used("$STRUCTURE BLOCKS: /",$rsvst,rsvst1);
CALL used("$VECTOR BLOCKS: /",$vdbst,vdbst1);
BRS $34("$POINTERS: /",-1,sttus2);
BRS $36((ptrtbl+1)/3,10,sttus2);
CIO sttus2(17B);
BRS $36((ptrtxn+1)/3,10,sttus2);
IF todas THEN CALL crlf ELSE CIO sttus2(155B);
opnfg ← 0; !any sbr END.
(used) PROCEDURE(usd1,usd2,usd3);
  BRS $34(.AR,-1,sttus2);
  usd4 ← 0;
  usd5 ← usd3+usd2;
  FOR usd1 FROM usd2 INC 1 TO usd5 DO
    IF [usd1] # 0 THEN BUMP usd4;
  BRS $36(usd4,10,sttus2);
  CIO sttus2(17B);
  BRS $36(usd3+1,10,sttus2);
  RETURN END.
(b1) PROCEDURE;
  RETURN( LRSH $16(.AR)) END.
(b2) PROCEDURE;
  RETURN( LRSH $8(.AR) .A 377B) END.
(b3) PROCEDURE;
  RETURN( .AR .A 377B) END.
(typdat) PROCEDURE(typdt1);
  BRS $36(b2(typdt1),10,sttus2);
  CIO sttus2(17B);
  BRS $36(b3(typdt1),10,sttus2);
  CIO sttus2(17B);
  BRS $36(b1(typdt1),10,sttus2);
  CIO sttus2(0);
  RETURN END.
%input, open-file, time-date, alarm clock forks%
%*****fork character input routine*****%
  (inptfs) %This routine accepts a character in the A, and puts it
  into the input buffer.
  It checks to see if the input buffer is wrapping around, and
  resets the input buffer pointers if it is
  if the input buffer is full, then it returns if an input fork
  is not running.

```


AUCXOD, MOL CODE

If, however, there is an input fork running, then it waits for NLS to empty the buffer, and then continues. (NLS waits for an interrupt from inptfs when an input processor is running and the input buffer is empty)

It then stores the character into the input buffer, and updates the input buffer pointer.

If inptc is waiting for a character from inptfs, then it interrupts the main fork to re-activate the main fork (interrupt 7)

%

PROCEDURE;

.XR ← .AR; %character to be input%

%get address of position of next slot in input buffer%

inpfsl ← inptbi+1;

IF inpfsl >= inptbe THEN inpfsl ← \$inptbf; %buffer wraps around%

%check to see if buffer is full%

IF inpfsl = inptbo THEN BEGIN %buffer is full%

IF NOT paslfg DO-SINGLE RETURN; %ignore character if input fork is not running (which may not be a very good idea)%

WHILE NOT inptfg DO BRS \$45 END; %wait for main NLS fork to empty input buffer if input fork is running%

%put character into input buffer and update pointer%

[inptbi] ← .XR;

inptbi ← inpfsl;

%check to see if input fork is running, and main fork is waiting for an interrupt%

IF inptfg THEN %create interrupt seven%

BEGIN %interrupt main fork%

inptfg ← 0; %clear flag%

IF NOT .SKIP BRS \$79(7) DO-SINGLE rerror() END; %cause interrupt seven%

RETURN END.

*****input fork*****%

(inptf) % This is the input fork control routine.

It runs whenever NLS is running and there is no input processor

It reads a character from the work station input buffer,

checks to see if it is a rubout

If it is a rubout, then a check is made to see if it is the second consecutive one.

If it is the second consecutive one, the main fork is interrupted so-as to terminate NLS (intr5), and the input fork terminates itself

If it is not the second consecutive one, the following actions are taken:

clear the input buffer

If an output processor is running, then interrupt the main fork (interrupt 10)

If the main fork is busy with a long process which does not require input, (indicated by flag rubabt), then the

AUCKOD, MOL CODE

rubout flag is incremented.
 otherwise the rubout is passed on to the main fork in the
 input buffer
 If the character is a command accept, then it is translated
 to a -1, and the mouse co-ordinates are saved
 The character is put into NLS's input buffer by calling
 inptfs
 %

PROCEDURE;

```
%input fork%
EXTERNAL inptfl, inptf3;
%*****starting location for input fork*****%
(inptfl): LOOP BEGIN
  inptfl ← inptl(); %get next character from w.s. buffer%
  IF .AR = 1000137B THEN BEGIN %rubout%
    inptbo ← inptbi; %clear input buffer%
    IF inptrf THEN BEGIN %this is the second consecutive
      rubout%
        IF NOT .SKIP BRS $79(5) DO=SINGLE error(); %interrupt
          main fork%
        BRS $10 END; %terminate (self-destruct)%
        IF outffg AND NOT .SKIP BRS $79(10)
          DO=SINGLE CALL error; %interrupt main fork if output
            processor running%
        IF rubabt THEN BEGIN %NLS is doing something long and not
          reading chaacters%
          BUMP inptrf; %increment rubout flag%
          GOTO inptfl END; %go read next character%
        END
      ELSE BEGIN
        inptrf ← .BR ← 0; %clear out rubout flag%
        IF .AR = 1000144B OR .AR = 4000001B THEN BEGIN %command
          accept%
            inptmr+inptcm; %save mouse coordinates%
            inptpr←-1 END; %translate into -1%
            CALL inptfs(inptfl) %pass character to input buffer%
          END;
        END. %of input fork control%
      (inptl) %This is the lowest part of the input fork.
        It runs at any time during which NLS is running (not TODAS) and
        an input processor is not running.
        It reads a character from the keyboard when the main fork of
        NLS is busy at some task, and returns with the characte in the
        A register.
        It returns one character each time it is called
        %
```

PROCEDURE;

```
%low priority- lowest level input routine%
LOOP BEGIN
  %check work station input buffer to see if it has any
  characters in it, and read the first one if it does%
```

AUCKOD, MOL CODE

```

WSI ttyno;
%WSI returns the following:
  A register:
    A./0:0/ = 0
    A./1:5/ = device indication
      1 = keyboard
      2 = keyset a
      3 = keyset b
      4 = right mouse button
      5 = center mouse button
      6 = left mouse button
    A./6:3/ = input buffer state
      0 = character read and input buffer now empty
      1 Input buffer empty..no character returned
      4 = more charaters waiting in the input buffer
      2 or 6 = more characters waiting in the same
        sampling interval
    A./9:15/ = character or position of mouse button
      (0=up, 1=down (I think))
  B register:
    Mouse co-ordinates at time character read
  X register:
    value of clock (real) at time character read
%
IF .AR NCB 100000B THEN BEGIN %a character read.. return it%
  inptlc ← .AR .A 77077777B; %character%
  inptcm ← .BR; %mouse co-ordinates%
  inptcl ← .XR;%time%
  RETURN(inptlc) END;
%by here, no character read..wait for wsi buffer not-empty%
  inptfa ← 0; %reset input fork active flag%
  BRS $-10(,,ttyno); %set up b register for dismissal on
  work station buffer not empty%
  BRS $72(,,3); %dismiss on queue 3%
  %input fork reactivated from dismissal..there is a character
  in w.s. buffer%
  BUMP inptfa %set input fork active flag%
  %go back to start and read character%
END;
NULL END.
%*****time and date fork*****%
(TADFK) %This is the time and date fork. It dismisses itself for
ten seconds, and then reads the time and date from the system,
into the proper display buffer. It does not read the date from
the system is OPNFG is set, but waits until it is false. This is
because reading the date and opening a file at the same time
crashes the TSS system. OPNFG is set by all file opening
routines.%
PROCEDURE; %time and date fork%
  EXTERNAL tadfk1;
  (tadfk1):

```

AUCKOD, MOL CODE

```

WHILE opnfg DO BRS $45;
BRS $91($dlbtad*3-1,.AR);
BRS $81(10000);
GOTO tadfkl END.
*****alarm fork*****%
(wtfork) PROCEDURE;
%alarm wait fork%
EXTERNAL wtfork1;
(wtfork1):
IF .AR > 0 THEN BRS $81(.AR);
WHILE NOT inptfg OR outfg OR paslfg DO BRS $45;
IF NOT .SKIP BRS $79(8) DO-SINGLE CALL rerror;
BRS $10 END.
*****pass1 rubout fork*****%
(chkrub) %This routine watches for rubouts coming from the
keyboard when an input processor is running
If it finds a rubout, it clears the input buffer and passes the
rubout on to the main fork%
PROCEDURE;
%check for rubout%
EXTERNAL chkrub1;
(chkrub1):
LOOP BEGIN
WSI ttyno; %read character%
IF .AR CB 100000B THEN BEGIN %no character..wait for one%
BRS $-10(,ttyno); BRS $72 END
ELSE IF .AR .A 77077777B = 1000137B THEN BEGIN %rubout read%
BRS $57; %guarantee 16 ms running time%
inptbo ← inptbi; %clear input buffer%
inptfs(1000137B); %pass rubout along to main fork throu
gh input buffer%
%When input processors other than pass1 are implelented,
a check should be made here to see if rubabt is set.
If it is set, then the main fork should be terminated
along with the input processor%
BRS $10 END END END.
%xerr & reset%
(xerror) PROCEDURE;
DECLARE
errm0=(4,4,"ERROR"),
errm1=(14,14,"FILE COPY FAILS"),
errm2=(17,17,"OPEN SCRATCH FAILS"),
errm3=(18,18,"CANNOT LOAD PROGRAM"),
errm4=(8,8,"I/O ERROR"),
errm6=(16,16,"EXCEEDED CAPACITY"),
errm7=(13,13,"BAD FILE BLOCK");
DECLARE
abmes0=(4,4,"ABORT"),
abmes1=(11,11,"NO SUCH NAME"),
abmes2=(13,13,"NO SUCH NUMBER"),
abmes3=(16,16,"DUPLICATE POINTER"),

```

AUCXOD, MOL CODE

```

    abmes4=(14,14,"NO SUCH POINTER"),
    abmes5=(13,13,"ILLEGAL ENTITY"),
    abmes6=(13,13,"STRUCTURE FULL"),
    abmes7=(13,13,"TEXT AREA FULL"),
    abmes8=(7,7,"BAD FILE"),
    abmes9=(17,17,"STATEMENT TOO LONG"),
    abms10=(8,8,"TOO LARGE"),
    abms11=(11,11,"SYNTAX ERROR"),
    abms12=(11,11,"ILLEGAL MOVE"),
    abms13=(13,13,"ILLEGAL DELETE");
DECLARE fatmes=(10,10,"FATAL ERROR");
DECLARE
    errmes=(errm0,errm1,errm2,errm3,
    errm4,errm5,errm6,errm7);
DECLARE
    abmes=(abmes0,abmes1,abmes2,abmes3,
    abmes4,abmes5,abmes6,
    abmes7,abmes8,abmes9,abms10,abms11,
    abms12,abms13, abmes0);
EXTERNAL xerr,xrerr,xabort;
(xerr):
CALL dismes(errmes/rerrn/,2); CALL resnls;
(xrerr):
BRS $34("$OOOPS ",6,1); SKR stackp; BRS $36(/stackp/,8);
IF NOT exp THEN BEGIN
    CALL dismes($fatmes,1);
    crfnm1 ← $crfnm*3-1;
    crfnm2 ← crfnm1+11;
    IF NOT .SKIP BRS $16(1100000B,$crfnm1,0) DO-SINGLE HLT;
    $1 ← .AR;
    $2 ← $xrerr;
    $3 ← 0;
    $4 ← 37777B;
    $5 ← -1;
    BRS $91(10B*3-1,.AR);
    $20B ← rlr1;
    $21B ← rlr2;
    CALL ovldpg(,,$rfbp2);
    CALL ovldpg(,,$rfbp1);
    IF NOT .SKIP BRS $93(2,$rlr1,$1) DO-SINGLE HLT;
    BRS $20($1) END;
CALL intr5;
(xabort):
CALL dismes(abmes/rerrn/,2);
CALL reset1;
IF NOT todas THEN BEGIN
    EXECUTE inpfbk; CALL uloff END;
CALL ovlgc(abtgt-1) END.
(reset1) PROCEDURE;
FOR smtemp FROM 0 INC 1 TO rficbx DO
    frzcpt/smtemp/ ← -1;

```

AUXKOD, MOL CODE

```
    opnfg ← 0;  
    CALL rlblrf;  
    RETURN END.  
(resnl) PROCEDURE;  
    CALL reset1;  
    IF paslfg DO-SINGLE CALL xintr9;  
    IF outfgr THEN BEGIN CALL rlsall;  
        CALL gtembk END;  
    GOTO gps END.  
.FINISH of AUXKOD
```

:RECINT, 11/06/69 1654:33 WSD ;

1 %Description of the RECINT overlay

1a The RECINT overlay contains MOL code for initialization and for continuation (recovery) of NLS. It is normally relabeled out - it is only in to initialize, restart, or start some special overlay or fork.

1b This overlay is read-only and shared.%

2 % recint overlay .DSN=1; .SCR=1; .IND=3;%

```
%declarations%
SET rit=100 %this should be the address in data%;
DECLARE origin(7777B);
POP err=175B, any=177B, sbc=107B;
DECLARE EXTERNAL patchn=0, exp=1; %patch number, experimenal
flag%
DECLARE
    icortb=(tprmsp,tsdbmn,trecin,tutilt,
            tinpfb,tioctl,tsege,ttxted),
    icort2=(tlksu,tdiddl,tkeywd,tauxco,
            tstrmn,tcdspl,tclnup),
    icort3=(tvcted,tvctrl,tmnctr),
    icort4=(tcacmp,tcalc),
    icort5=(ttoda1,ttoda2);
LOCATION FOR TEMPORARIES rit; REENTRANT;
%initialisation of data areas%
(setrsv) %This routine sets the ring status block to zeros.%
PROCEDURE;
    FOR setfl1 FROM 0 INC 1 TO rsvst1 DO rsvst/setfl1/ + 0;
    RETURN END.
(setsdb) %This routine sets the SDB status block to zeros.%
PROCEDURE;
    FOR setfl1 FROM 0 INC 1 TO sdbst1 DO sdbst/setfl1/ + 0;
    FOR setfl1 FROM 0 INC 1 TO vdbst1 DO vdbst/setfl1/ + 0;
    RETURN END.
%procedures for playing games with relabelling and PMT%
(rlsall) %All output pages are released by calling this
procedure. First, it terminates th output fork if it is running,
and closes the output file.%
PROCEDURE;
    IF .NEG outfp6 THEN BRS $32($outfpt); %stop the fork if it
is running%
    IF outffg = 1 THEN BEGIN %processor%
        BRS $121(toutv0); BRS $121(toutv1); BRS $121(toutv2);
        BRS $121(toutv3); BRS $121(toutv4); BRS $121(toutv5);
        BRS $121(toutv6); BRS $121(toutov) END
    ELSE BEGIN % pass 4 %
        BRS $121(tp4c3); BRS $121(tp4cod); BRS $121(tp4dat);
        BRS $121(tp4c2); BRS $121(tp4ary) END;
    RETURN END.
(rlblrf) %RECINT fits in the same page position as one of the
```

RF block pages. Hence on returning from any code in REGINT it is necessary to get the RF block pages relabeled back in. This procedure does that. Relabeling does not take place, but the internal relabeling is changed so that when the next overlay call takes place, the RF blocks are in.%

PROCEDURE;

%relabel in random file blcocks%

BUMP ovrelf; ovldpg(,,\$nrfbp1); ovldpg(,,\$nrfbp2);

RETURN END.

(makro) %Given a relabeling byte in the A, this procedure makes that page read only. %

PROCEDURE;

RETURN(BRS \$80(.AR .V stinro) .A 77B) END.

(getpag) %This PROCEDURE is CALLED to obtain a new page of memory. The relabeling byte for the new page is returned in the A, and the new page is left relabeled into page seven.%

PROCEDURE;

%get a new page from system, return rl in a%

BRS \$43();

COPY (AB,BA); .AR ← .AR .A 77777700B;

COPY (AB,BA); BRS \$44();

\$37777B ← 0; BRS \$43();

RETURN(.BR .A 77B) END.

(gtembk) PROC;

%at initialization orafter PROCess, get pages from nlsf2, nlsf3%

%first release any processor pages which may be in%

BRS \$46; %turn rubout off%

setin1 ← \$toutv0-1;

WHILE setin1←setin1+1 ≤ toutv8 DO

IF [setin1] THEN BEGIN

BRS \$121(); [setin1]←0 END;

%now get NLS pages back%

IF todas THEN BEGIN % read in file with todas pages%

IF NOT .SKIP ldsbsy("(NLS)/X4NLS",11,6) DO-SINGLE rerror();

COPY (AB,A);

[/icort5+0/] ← makro(LSH \$6()); % todas main control%

[/icort5+1/] ← makro(LSH \$6(0)) % todas support page one% END

ELSE BEGIN

IF NOT .SKIP ldsbsy("(NLS)/X2NLS",11,6) DO-SINGLE rerror();

setin2 ← .AR; setin3 ← .BR;

[/icort3+0/] ← makro(LSH \$6(0,setin2)); %vctedt%

[/icort3+1/] ← makro(LSH \$6(0)); %2vcted%

[/icort3+2/] ← makro(LSH \$6(0)) %mnctr1% END;

IF NOT .SKIP ldsbsy("(NLS)/X3NLS",11,6) DO-SINGLE CALL rerror;

setin2 ← .AR; setin3 ← .BR;

[/icort4+0/] ← LSH \$6(0,setin2); %content analyzer%

[/icort4+1/] ← LSH \$6(0); %calculator%


```

        outffg ← 0;
        BRS $47; %turn rubout back on%
    RETURN END.
(rlssom) PROC; %release all from nlsf2 and nlsf3%
    IF todas THEN BEGIN
        BRS $46; %turn off rubout%
        BRS $121(ttodal); %todas main control%
        BRS $121(ttoda2) %todas support page 1% END
    ELSE BEGIN
        BRS $121(tvcted); BRS $121(tvcte2);
        BRS $121(tmnctr) END;
        BRS $121(tcacmp); BRS $121(tcalc);
        BUMP outffg;
        BRS $47; %turn rubout back on%
    RETURN END.
(ldsbsy) PROCEDURE(,setin3,setin1); % load
sub-system(name,length) % 3c11a %a contains address of file name%
    %b contains number of characters in file name%
    %x contains number of characters to delete if not
    experimental%
    %skip return with relabeling in AB if successful%
    setin2 ← .AR*3-1; setin3 ←+ .AR;
    IF NOT exp THEN setin2 ←+ setin1;
    IF NOT .SKIP BRS $100(,$setin2,-1) DO-SINGLE RETURN;
    SKIP RETURN END.
%procedures for starting forks%
(stotfk) %This routine simply starts the output fork (without
executivity).%
    PROCEDURE(outfpt);
        RETURN(BRS $9($outfpt .V 02000000B)) END.
(setalm) %The alarm fork is started by calling this
procedure. It assumes that the time is in the spec stack. The
time required to wait (in ms.) is obtained by calling TIMMS with
the current time. The fork is then started with that time in the
A. Control is returned by going to the previous state.%
    PROCEDURE;
        EXTERNAL stalrm,stwfk,sttimr;
        (stalrm):
        wtfpl←alarmwt ← timms(BRS $39);
        (stwfk):
        IF .NEG wtfp6 THEN BRS $32($wtfpt);
        wtfpt ← $wtfprk1; alarmst ← BRS $42;
        BRS $9($wtfpt .V 26000000B);
        CALL ovldpg(,,$rfbpl); BUMP ovrelf; GOTO gps;
        (sttimr):
        wtfpl←alarmwt ← timms(,0); GOTO stwfk END.
(timms) %A time is provided in the B (3 8-bit bytes of hr.,
min., sec.). The difference between the time in the spec stack
(hr., min., sec. from the bottom up) and this given time is
returned in the A, in ms.%
    PROCEDURE;
        WHILE inptm() NOT= '← DO NULL;
        CALL dismes(,0); RETURN END.

```

```
(setnls) %This routine arms the pseudo interrupts, starts the
input fork and the time and date fork, and clears the input
buffer.%
```

```
PROCEDURE;
```

```
CALL rlblrf; BRS $78(intmsk); inpfpt ← $inptf1; inptrf←0;
BRS $9($inpfpt .V 66000000B); tadfpt ← $stadfk1;
BRS $9($stadfpt .V 26000000B); TCO 155B; TCO 152B;
IF NOT ttyinf THEN BRS $11(.,,ttyno);
%clear input buff%
RETURN END.
```

```
%procedures for processors%
```

```
%procedures for ordinary processors (compilers, passh, etc)%
```

```
(waitbs) %waitbs reads characters until a back arrow (+) is
typed, and then returns.%
```

```
PROCEDURE;
```

```
timms1←LSH $8(0); timms2←LSH $8(0); timms3←LSH $8(0);
RETURN(
((($1/spsk)-timms1)*60
+$3/spsk)-timms2)*60
+$5/spsk)-timms3)*1000) END.
```

```
(setpsl) %The insert QED branch command is implemented here.
This procedure puts up the reading text message, and restarts the
input fork with the page containing PASS1 relabeld in, and the
PAS1FG set. It then does an overlay CALL to QIB in STRMNP.%
```

```
PROCEDURE;
```

```
DECLARE plmes =(19,19,"READING TEXT -- WAIT");
EXTERNAL setpl1;
SET tpass1=tcalc;
(setpl1):
```

```
IF NOT .SKIP ldsbsy("(NLS)/NPASS1",12,6) DO-SINGLE !err
$4;
```

```
toutov←.BR←tcalc; %kludge to save rlbl byte%
tpass1←LRSH $18(.AR); CALL dismes($plmes,1);
BRS $32($inpfpt); EXECUTE auxcod;
inpr11←inpfp4; XMA rlr1; setin2←.AR;
inpr12←inpfp5; XMA rlr2; setin3←.AR;
setf11←cortb/pass1 .A 7);
CALL ovldpg(.,,$pass1); CALL ovldpg(.,,$auxcod);
.AR←setin2; XMA rlr1; inpfp4←.AR;
.AR←setin3; XMA rlr2; inpfp5←.AR;
pas1fg←1; CALL ovldn(.,,setf11); BUMP ovrelf;
outfpt←$chkrbl; BRS $9($outfpt .V 44000000B);
inptbo←inptbi; inpfpt←$inptf3; inptfg←inptrf←0;
BRS $9($inpfpt .V 66000000B);
IF todas THEN BEGIN CALL rlblrf; !sov todmnc; !brv inst11
```

```
END;
```

```
!sov prmspc; !brv qib END.
```

```
(inprc) %This is the procedure which is used for starting up
the input processors
```

```
It expects four parameters:
```

```
spsk/1): address of a-string containing name of
processor
```

```
spsk/3): address of cell in overlay table which is to
```

```

be execute upon start of processor
    spsk/5/: starting address for processor
    spsk/7/: parameter to be passed to processor
It performs the following functions:
    read relabelling of processor into PMT and stores into
tinpp0-tinpp8
    stop the input fork
    if NLS is running, start the output fork to check
rubouts
    clear the rubout and input in progress flags(inptfg,
inptrf).
    start up input processor fork
    branch to idle loop in main control
%
PROCEDURE;
    lodprc($1/spsk/); outffg←0; %get relabelling of processor%
    apr($procml,,$1/spsk/);
    dismes($1/spsk/,1); %display message%
    inptfg←inptrf←0; %clear out input in progress and rubout
flags%
    IF NOT todas THEN BEGIN
        outfp4 ← LSH $18(tdata,tauxco) .A 77000000B;
%relabelling for data pae%
        .AR ← LRSB $6;
        outfp5 ← .BR;%relabelling for auxcod%
        outfpt←$chkrb1;
        BRS $9($outfpt .V 64000000B) %start fork and set
relabelling from panic table%
        END;
        inpfpt ← $inprs;
        BRS $9($inpfpt .V 40000000B); %start up fork%
        IF todas THEN BEGIN
            !sov todas1; !brv reset END;
            !sov mnctrl; !brv cmdrst;
            %****start of input processor fork****%
            (inprs):
            END.
        (outprc) % this is the procedure which is used for starting
up a processor.
        It expects four parameters:
            spsk/1/: address of a-string containing name of
processor
            spsk/3/: address of cell in overlay table which is to
be execute upon start of processor
            spsk/5/: starting address for processor
            spsk/7/: parameter to be passed to processor
        This procedure relabels in the processor (after releasing
PMT cells for some parts of NLS), and starts up the processor
fork.
        It waits for the pocessor fork to terminate, and upon
termination, it checks to see if there was an error in the
process, displaying an ppropiate message if there was.
        It finally rstores the relabelling according to NLS needs,

```

```

and then returns to previous state
PROCEDURE;
  DECLARE procml = (11,11," in Progress"),
    procml2 = (10,10,"Error in Process"),
    procml3 = (19,19,"Error in Process---Type ←"),
    rlssom(); %set outfgr = 1 and release NLS pages from PMT%
    lodprc($1/spsk/); %get relabelling for processor%
    stofk($setprf); %start up the processor fork%
    .(utilty,apsr)($procml,, $1/spsk/);
    dismes($1/spsk/,1); %display message and leave it there%
    BRS $31($outfpt); %wait for processor fork to terminate%
    gtembk(); %release processor pages from PMT and get back
NLS pages set outfgr ←0%
  IF outfp6 > 0 THEN %error in process..type message%
  BEGIN
    IF todas THEN dismes(procml2,1)
    ELSE BEGIN
      dismes(procml3,1); waitbs() END
    END;
    GOTO rtngps; %return to previous state%
    (setprf): rlblrf(); %put random file blocks into the
relabelling%
    prmwd ← $7/spsk/; %parameter word%
    lsov [spsk3]; lbrv [spsk5]; %branch to starting code
in processor%
    (xitprc):(extps4):(extout): BRS $10 %terminate fork%
  END.
  (lodprc) % This procedure reads the relabelling of a
processor (address of a-string containing name of subsystem file
is passed in the A register), and stores it into the eight cells
toutv0-toutv6 (one relabelling byte in each cell)%
  PROCEDURE(lodpr1);
    setin2 ← $1/.AR/+1; %length of name+1%
    IF NOT SKIP ldsbsy(lodpr1+2,.BR,0) THEN BEGIN %error on
subsystem%
      BRS $20(outfn); gtembk(); lerr $4 END;
    setin2 ← .AR; setin3 ← .BR;
    setin1 ← $toutv0-1; .XR ← 0;
    WHILE setin1+setin1+1 ≤ toutv8 DO [setin1] ← LCY
$0/.XR+.XR+6/(setin3,setin2) .A 77B
    RETURN END.
  (todpro)PROCEDURE;
    %this is a PROCEDURE which contains code for starting up
pass4 and the processors for todas.
    It should really go into ioctl, but there is no room in
ioctl, and it cannot go into todmnc because todmnc is released
when a processor is brought in%
    (stdpl): %start up pass 1%
    EXECUTE ioctl; smcreg ← inptm();
    outfn ← opnfil(0,33000000B,0); %open the file%
    CALL echo3; GOTO setpl1;
    (stdpr): %set a processor for todas%
    todpr1 ← 0; %a processor is being set%

```

```

!any mks; !sbc setpro; %get the processor into the pmt%
outfig + 1; %for rubout action%
GOTO setpr;
(sttdph): %set up pass h%
todprl + 1; %pass h%
(settpr): EXECUTE ioctl;
smcreg + inptm();
outfn + opnfil(1,3300000B,0); %open the file for the
processor%
CALL rlblrf; %get the file blocks in%
IF todprl DO-SINGLE GOTO setph1;
BRS $3h("$PARM (OCTAL) = /",-1,1); % ask for parameter
% prmwd + BRS $38(,8,0); %get the parameter word%
BRS $3h($tdcrLf,-1,1); %type out a crlf%
GOTO /actpro+1/; % start up the processor..do not
RETURN%
NULL END.
%portal processor procedures%
(stfpr) PROC; %dummy routine%
EXTERNAL stfprc;
DECLARE prtlnm = (33h66163B,"PORTAL");
(stfprc): %PROCESSOR FORK STARTING ADDRESS%
%Prepare new relabeling for processor fork%
%Set up address and length of processor name in AB%
.BR + stnl + 1; %length%
.AR + $stn; %starting address of name string%
.XR + 0;
%Get relabeling for processor into AB and then store in
toutv0-6%
%this needs to be modified to take into account
ldsbsy skip on success (WHP)%
CALL lodrl(ldsbsy());
%Get relabeling for portal and store in toutov%
CALL ldsbsy($prtlnm,9,3);
toutov + LRSH $18(.AR,0);
%Portal will relabel in random file block pages IF it
needs them.%
CALL rlblrf; %FOR NOW, ASSUME THAT IT NEEDS THEM%
%Call portal%
!sov outov1; !brv apgo %go to activate processor
starting address in portal%
END.
(dmLink) PROCEDURE; %Used by portal to link to dismes%
IF clt = 3 THEN BEGIN
CALL dismes($clt+1,1); CALL waitbs;
CALL dismes(,0) END
ELSE CALL dismes($clt+1,clt);
!any sbr END.
(lodrl) PROCEDURE(,lodprl); % relabeling in AB stored in
toutv0-6 %
COPY (AB,A); %A TO B CLEAR A%
toutv0 + LSH $6(); toutv1 + LSH $6(0);
toutv2 + LSH $6(0); toutv3 + LSH $6(0);

```

```

    toutv4 ← LSH $6(0,10dpr1);
    toutv5 ← LSH $6(0); toutv6 ← LSH $6(0);
    RETURN END.
%procedures for returning from processors
(oergps) %The output compiler starting routines CALL this
PROCEDURE after compilation is done. The output fork panic table
is checked for an illegal instruction or memory trap, and the
output error flag (OUTERF) is checked. Then either output fork
trapped, output ok, or syntax error is displayed. WAITBS is
CALLED for confirmation. RETURN is to the previous state.%
PROCEDURE;
    DECLARE
        syner=(20,20,"PROCESS ERROR, TYPE ←"),
        comok=(17,17,"PROCESS OK, TYPE ←"),
        outflm=(26,26,"OUTPUT FORK TRAPPED, TYPE ←");
    EXECUTE recint; EXECUTE inpfbk;
    IF outfp6 > 0 THEN CALL dismes($outflm,1)
    ELSE CALL dismes((IF outerf THEN $syner
    ELSE $comok),1);
    IF todas DO-SINGLE GOTO rtngps;
    CALL waitbs; CALL dismes(,0);
    (rtngps): CALL rlblrf; GOTO gps END.
(bsrtn) PROCEDURE;
    IF todas DO-SINGLE lany gps;
    EXECUTE inpfbk; CALL waitbs;
    lany txt(-1); lany gps END.
%procedures for starting and contining%
(ttyck) %This is called from SETINT. If the executivity flag
is set (Brs 71), it sets TTYINF and writes the input from tty
buffer message.%
PROCEDURE;
    IF .SKIP BRS $71 THEN BEGIN
        BUMP ttyinf;
        BRS $34("$INPUT FROM TTY BUFFER./",-1,1) END;
    GOTO setri END.
(setws) %The function of this routine is to open the work
station via BRS -1.%
PROCEDURE;
    IF NOT .SKIP BRS $-1(,ttyno) DO-SINGLE !err $2;
    BRS $-9(-1,ttyno); RETURN END.
(setint) %This procedure is entered to start up NLS
initially. It does the following things:
    Read-write pages must be obtained for the data page and
display buffer. This is somewhat tricky, since NLS is started
with eight read-only pages (when started as a subsystem). So,
first page zero is relabeled out without doing a store -- the
byte for the page that is relabeled out is saved in the X.
Then another page for the display buffer is obtained, and the
pages are initialized by brs100's.
    The table containing relabeling bytes for the various
overlays is set up, using ICORTB which indicates which overlays
are in the eight initial pages.
    The initials and user name are read. If, before the

```

initials are typed in, a command accept is typed, and the user's executivity is set, then TTYINF is set, and all keyboard input is taken from the teletype input buffer. This is done by calling TTYCK.

The initial relabeling is set up. That is, up to this point the overlay relabeling routines have not been called, and the current internal relabeling (RLR1 and RLR2) is zero. So they are called to set the internal relabeling to an initial state.

The relabeling for the forks is set up and put into the panic tables.

The assembly date and time are moved into the command feedback line.

Routines SETFIL and SETNLS are CALLED.

Routine INTRSV in INPFBK is CALLED, and INTRV1 (starting destination) is set to SETDUM in MNCTRL.%

PROCEDURE;

%initial PROCEDURE for firing up nls%

DECLARE setin1=00777777B,stinro=40000000B;

EXTERNAL stint1;

(stint1):

BRS \$43; BRS \$44(.AR .A setin1);

%store initial relabeling temporarily%

setin3 ← .BR; setin2 ← .XR; BRS \$43; tnlsda ← LSH \$6(0);

tdisbu ← getpag();

%getpag must leave a new page in slot seven%

% it will be the display buffer page %

[[icortb+0/] ← makro(LSH \$6(0,setin2));

[[icortb+1/] ← makro(LSH \$6(0));

[[icortb+2/] ← makro(LSH \$6(0));

[[icortb+3/] ← makro(LSH \$6(0));

[[icortb+4/] ← makro(LSH \$6(0,setin3));

[[icortb+5/] ← makro(LSH \$6(0));

[[icortb+6/] ← makro(LSH \$6(0));

[[icortb+7/] ← makro(LSH \$6(0));

CALL setdpg; % initialize data page %

IF NOT .SKIP ldsbsy("(NLS)/X1NLS",11,6) DO-SINGLE BRS \$10;

setin2 ← .AR; setin3 ← .BR;

[[icort2+0/] ← makro(LSH \$6(0,setin2));

[[icort2+1/] ← makro(LSH \$6(0));

[[icort2+2/] ← makro(LSH \$6(0));

[[icort2+3/] ← makro(LSH \$6(0));

[[icort2+4/] ← makro(LSH \$6(0,setin3));

[[icort2+5/] ← makro(LSH \$6(0));

[[icort2+6/] ← makro(LSH \$6(0));

%leave this in to debug didd1,keywd,clnup%

%take out to debug vector package, content analyzer%

%for working system, change to branch over brsl21"s%

BRS \$121(tdidd1); BRS \$121(tkeywd); BRS \$121(tcclnup);

CALL gtembk; %load nlsf2 and nlsf3%

(setri): %read initials%

BRS \$34("\$INITIALS PLEASE: /",-1,1);

cinit←setin2+0;

```

WHILE setin2 <= 2 DO BEGIN
  IF TCI setin3 IN (100B,132B/ THEN
    BEGIN setin3←.AR-40B; TCO setin3 END;
  IF .AR = $asca DO-SINGLE CALL ttyck;
  IF .AR IN (40B,72B/ DO-SINGLE GOTO setri;
  cinit ← LSH $8(cinit,0) .V setin3; BUMP setin2 END;
(setru): %read user%
BRS $34("$USER: /",-1,1);
setin2←setin3+$setinu*3-1;
IF NOT .SKIP BRS $104(0,$setin2,0) DO-SINGLE GOTO setru;
cuno ← .AR;
IF NOT todas DO-SINGLE CALL setws;
trfbp1←getpag(); trfbp2←getpag();
  rfbpgs ← LSH $6(LSH $6(trfbp1,0).V trfbp2);
% now set up rlrl, rlr2, and input fork relabeling %
CALL ovldpg(,,$utilty); CALL ovldpg(,,$nlsdat);
CALL ovldpg(,,$inpfbk); CALL ovldpg(,,$auxcod);
infp4←opnfp4←wtfp4←rlrl;
infp5←opnfp5←wtfp5←rlr2;
CALL ovldpg(,,$disbuf);
IF NOT todas THEN BEGIN
  tadfp4 ← rlrl; tadfp5 ← rlr2 END;
alarmst←litcb←0;
CALL ovldpg(,,$recint); BRS $44(rlrl,rlr2);
IF todas THEN BEGIN
  CALL settod; CALL setfil END
ELSE BEGIN
  CALL setdis(1); CALL setfil; CALL setnls END;
intrvl ← $setdum; lsov strmnp; lbrv intrvl END.
(rstart) %This PROCEDURE is activated when a continue NLS is
done, after leaving NLS. It does approximately the following:
  Calls SETDIS with an argument of zero.
  Restarts the alarm clock with the correct remaining time,
IF it was running.
  Calls SETNLS.
  If the user left gracefully (/RERROR/ is zero and INPTFG
is true), THEN the file he was working with is restore
PROCEDURE;
EXTERNAL rstrtl;
(rstrtl):
CALL gtembk; %get the subsystem files back%
IF NOT todas THEN BEGIN
  CALL setws; EXECUTE disbuf; CALL setdis(0);
  IF alarmst THEN BEGIN
    wtfpl←alarmwt-(BRS $42-alarmst);
    wtfpt←$wtfrkl; BRS $9($wtfpt .V 26000000B) END END;
pl←litcb←flcufg←paslfg←0;
IF todas THEN BEGIN BRS $78(intmsk); CALL rlblrf END
ELSE CALL setnls;
IF /rerror/ = 0 AND inptfg THEN BEGIN
  IF rffn THEN BEGIN
    CALL oprfff;

```



```

        IF todas THEN BEGIN !sov todmnc; !brv reset END;
        !sov mnctrl; !brv dmain END;
        inptfg ← 0; !sov ioctl; !brv rstfil END;
        /rerror/ ← 0; CALL setfil; intrvl ← $setdum;
        !sov strmp; !brv intrvl END.
%file initialisation and recovery procedures%
(junkf) %If a bad file is read, this routine is called to
initialize the file header of the working copy (by calling
SETFIL). Control is returned to SETDUM as in the initialization of
NLS.%
PROCEDURE;
    %read a junky file %
    BRS $!7; %close all files%
    rffn←curfn←0; CALL setfil;
    IF NOT todas THEN BEGIN CALL setdis(1);
        abtgt ← 40600000B .V $setdum END;
    pl ← 0; CALL rlblrf; intrvl ← 0; %for intrvl%
    !sov strmp; !brv intrvl END.
(setfil) %The file header is initialized by this routine.
The working copy is opened and set to null by CALLing BRS66X. All
status blocks are set to empty, the pointer table is set to empty,
and the frozen-core-page table (FRZCPT) is set to indicate that no
pages are frozen. CLRALL is CALLED.%
PROCEDURE;
    %set up random file%
    DECLARE inlsvw = "1.1";
    filhed/1/←20; filhed/2/←2;
    nlsvwd← inlsvw; hedsz←$ihedsz;
    rals←$irals; namdll←$inmdl1; namdl2←$inmdl2;
    funo←cuno; fcredit←BRS $39;
    sdbstl←$sdbstn-1; rsvstl←$rsvstn-1; rfbstl←$rfbsn-1;
    ptrtbl←tnsg←0; ptrtxn←$ptrtbx-1;
    FOR setfil FROM 0 INC 1 TO rficbx DO rfifcb/setfil/←0;
    FOR setfil FROM 0 INC 1 TO rfbstl DO rfbs/setfil/←0;
    BRS $-18; rfsn←.BR;
    %teletype no for file%
    CALL opnrff; rfwfg←1; nacp←0; CALL brs66x(rffn);
    FOR setfil FROM 0 INC 1 TO rficbx DO frzcpt/setfil/←-1;
    rfbs←$filhed; CALL setrsv; CALL setsdb; CALL setptr;
    IF NOT todas THEN %todas% CALL clrall(0);
    RETURN END.
.FINISH of RECINT

```

```

:DATA, 10/16/69 2101:02 WHP ; .HED="DATA PAGE AND SYMBOLS"; .SCR=1;
.PLO=1; .MCH=75; .RTJ=0; .DSN=1; .LSP=0; .MIN=70; .INS=3; .IGD=1;
* .CSW=1; .CMD=1; .DSN=1; .LSP=0; .SCR=1; .DLS=0; .FLN=0;.PSW=0; data
page ["hstckd"];

```

```
nolist ext,nul
```

```
bss 200b interrupt cells
```

```
data rerror,rerror,rerror,rerror,rerror 200-204 illegal
```

```
data intr5 request for termination (brs 10)
```

```
data rerror illegal
```

```
data intr7 requested by inptm - waits for char
```

```
data intr8 alarm clock interupt
```

```
data intr9 pass1 finished interupt
```

```
data intr10 command reset requested
```

```
rpt 25b
```

```
data rerror
```

```
endr
```

```
*this section contains all the equ's used throught the entire system
```

```
debug equ 1 debug flag for conditional assembly
```

```
tty equ -1 run on a teletype???
```

```
squeez equ -1 compact temporary cells
```

```
*origins of the various overlays and files
```

```
orguty equ 14000b utility
```

```
orgifk equ 20000b inpfbk
```

```
orgsmp equ 24000b sdbmnp
```

```
orgcln equ 30000b clnup
```

```
orgseq equ 30000b seqgen
```

```
orgcdy equ 24000b cdsply
```

```
orgtxt equ 34000b txtedt
```

```
orglks equ 20000b txted2
```

```
orgmet equ 24000b mnctrl
```

```
orgtmc equ 24000b todas main control
```

```
orgpsp equ 30000b prmosp
```

```
orgrfb equ 4000b random file blocks start
```

```
orgaux equ 24000b auxcod
```

```
orgrin equ 10000b recint
```

```
orgdbf equ 34000b display buffer
```

```
orgvct equ 24000b vctedt -- first page of graphics
```

```
orgvcl equ 30000b vctrl -- second page of graphics
```

```
orgddl equ 24000b diddle
```

```
orgkwd equ 24000b keyword
```

```
orgcal equ 30000b calculator
```

```
orgstr equ 20000b strmnp
```

```
orgcac equ 34000b content analyzer compiler
```

```
orgioc equ 24000b ioctl
```

```
passlg equ 20001b
```

```
outg equ 34002b outovl entry point
```

```
cacmpg equ 34001b cacmpl entry point
```

```
vm equ orgrfb
```

```
*ascii character codes for special characters
```

```
ascdd equ 136b ascii command delete
```

```
ascca equ 144b ascii command accept
```

```
ascsp equ 0 ascii space
```

```
asctab equ 151b ascii tab
```

DATA PAGE AND SYMBOLS

```

asccr equ 155b  ascii carriage return
asclf equ 152b  ascii line feed
ascbc  equ 141b  ascii backspace character
ascbw  equ 167b  ascii backspace word
asccdt equ 100b  ascii center dot
* register numbers for state machine
esn    equ 5    entity register (string part)
numnn  equ 6    number register
litln  equ 7    literal register
stnn   equ 8    statement name register
stnon  equ 9    statement number register
fnmn   equ 10   current file name register
stn2n  equ 11   second statement name
sarn   equ 13   string area register
* register maximum lengths
numnx  equ 12   max characters in number register
stnx   equ 32   max characters in statement name
stnox  equ 12   max characters in statement number
fnmx   equ 36   max length for file name reg
sarx   equ 32   max characters in string area
* symbols for direction set address (sd)
dirl   equ 0    set direction left (backwards)
dirr   equ 1    set direction right (forward)
* symbols for character position address (cpf)
cpflf  equ 1    set to line front
cpfle  equ 2    set to line end
cpfsf  equ 3    set to statement front
cpfse  equ 4    set to statement end
*symbols for compare character position address (ccp)
rellt  equ 1    less than relation
relle  equ 2    less than or equal relation
releq  equ 3    equal relation
relge  equ 4    greater than or equal relation
relgt  equ 5    greater than relation
relne  equ 6    not equal relation
* symbols for character class variable (ccv)
ccsp   equ 1    a space
cctab  equ 2    a tab
ccld   equ 3    any letter or digit
ccl    equ 4    any letter
ccd    equ 5    any digit
cch    equ 6    any old character
ccnp   equ 7    any non-printing character
ccpt   equ 8    any printing character
cccr   equ 9    a carriage return
* symbols for string class variable instruction (scv)
bggap  equ 1    simple gap
bglg   equ 2    line gap
bgnlg  equ 3    non-line gap
bgpts  equ 4    printing string
bgnps  equ 5    non-printing string

```

DATA PAGE AND SYMBOLS

```

    bgls  equ  6  letter string
    bgds  equ  7  digit string
    bglds equ  8  letter-digit string
* case save block
    ncasv equ  4  number of cells in case save block
    ncasvl equ  ncasv-1
* stack and other parameters
    stackx equ  53 number of cells in general stack
    hstckx equ  30 number of cells in file stack
    jstckx equ  5*2 (even) no. of cells in jump stack
    spskx  equ  10 number of cells in spec stack
    rficbn equ  4  no. of blocks in core at any time
    inptbc equ  10 no. of chars in input buffer
    cacodl equ  75 number of words for c.a. routine
    levelx equ  15 max level allowed in nls
* file parameters
    rsvstn equ  8  number of blocks for rsv in rf
    blkshz equ  1024 random file block size
    blkhdr equ  7  block header size
    rfbsn  equ  64  number of blocks in random file
    ptrtbx equ  30 pointer table length 3 per
    sdbstn equ  55 max num of blocks for sdfs in rf
    vdbstn equ  2  max num of blocks for vdfs
*group names, for use with define state pop (ds)
    grspec equ  1  special group
    gredit equ  2  edit group
    grjump equ  3  jump group
    grvect equ  4  vector package group
    groupx equ  4  maximum group number
* symbols to make up the cc mask and the cc table
    cmfl  equ  1  bit 23 is a letter
    cmfd  equ  2  bit 22 is a digit
    cmfp  equ  4  bti 21 ia a printing char not in
    * another class
    cmfsp equ  8  bit 20 is a space
    cmftb equ  16 bit 19 is a tab
    cmfcr equ  32 bit 18 is a carriage return
vworkx equ  levelx+1 size of work area for stvect calls
*now comes the data area definitions
* state machine*
    smareg data  0  a register
    smbreg data  0  b register
    smec  data  0  character portion of entity register
    smes  data  0  string portion of entity register
    smcreg data  0  c register (input character)
    state  data  0
        *previous state-requires that mnctrl is ovl 7
    abtgt  data  0
        *target for aborts (usually previous state)
    *
    numnxx data  numnx  max length, number register

```

DATA PAGE AND SYMBOLS

```

numn1 data 0 current length
numn bss numnx/3+1 room for string
stn1 data 0 current length
stn bss stnx/3+1 room for string
stn2 data stnx extra st name register
stn21 data -1
bss stnx/3+1
stnoxn data stnox max length, statement number register
stnol data 0 current length
stno bss stnox/3+1 room for string
*
fnmxn data fnmx
fnml data -1
fnm bss fnmx/3+1
*
bugreg bss 2 state machine bug mark area
flag data 1 the general flag
*
*
cflpos data 0 command feedback word counter
*
sarxn data sarx string area max length
sarl data 0 current length
sar bss sarx/3+1 room for string
*
*stuff for case save
*
casv data 0 index to case block cell
casvb bss ncasv case address block
casvn data ncasv number of cells in block
*stack areas, both general stack and spec stack
stackd bss stackx general stack
stack data stackd-1 stack pointer
stackl data stackd-1 initial pointer
stackt data stackx+stackd-1 top of general stack
rtncel data stackd-1 pointer to return cell
spskd bss spskx spec stack
spsk data spskd-2 spec stack pointer (double words here)
spskl data spskd-2 initial pointer
spskt data spskx+spskd-1 top of spec stack
hstack data hstckd+1 file link stack pointer
hstckd data -1,-1; bss hstckx-2
hstckl data hstckd+hstckx-1
hstckn data hstckd+1
jstack data jstckd jump stack pointer
jstckd bss jstckx
jstckl data jstckd+jstckx-1
* TODAS cells used by and for
todas data 0 true if system running as todas
rubabt data 0 true if rubout causes abort for todas

```

DATA PAGE AND SYMBOLS

```

    buffet data 0  index into todas input byffer
    carpos data 0  position of carriage (0=left margin)
    outmsl bss 1  temp for outmes
* input buffer
    inptbf bss  inptbc
    inptbe data  inptbf+inptbc  pointer to end of buffer
    inptbl data  inptbc  number of characters
    inptbi data  inptbf  read in pointer
    inptbo data  inptbf  read out pointer
    inptbb data inptbf  pointer to buffer start
intmsk data 176000b  interupt mask
* text pointers
    p1  bss  2  general pointer array
    p2  bss  2
    p3  bss  2
    p4  bss  2
    p5  bss  2
    p6  bss  2
    p7  bss  2
    p8  bss  2
    p9  bss  2
    p0  equ  p1-2
curgrp data -1  current group number
* work areas
    swork  bss  1  work area for fechl calls
    swork1 bss  1; swork2 bss  1; swork3 bss  1
    swork4 bss  1; swork5 bss  1; swork6 bss  1
    swork7 bss  1; swork8 bss  1; swork9 bss  6
    cwork  bss  5  work area for apachr calls
    cwork1 bss  1; cwork2 bss  1; cwork3 bss  1
    cwork4 bss  1
    vwork  bss  vworkx  work area for fechno calls
    vworkz equ  vwork+vworkx-1  address of last word
* file block IO
    rficb  bss  rficbn  random file index for core blocks
    frzcpt bss  rficbn  frozen core file page table (-1=free)
    nacp  bss  1  next available core page
    cinit  bss  1  current initials of user
    cuno  bss  1  current user number
*
    scndir bss  1  pm scan direction
*
    smtemp bss  1  temp cells for pops
    smtmp1 bss  1  temp cells for pops
    txtloc bss  1  return cell for txt (calls credis)
    rplsid bss  1  psid of statement being replaced in sc
*
* global variables for structure
    adjpos bss  1  contains psid after adjustment
    adjdir bss  1  use sub(lor=0) or suc(lor >0) after adj
    grpl  bss  1  first psid of group

```

DATA PAGE AND SYMBOLS

```
grp2 bss 1 second
svalue bss 1 used in struct routines for temp value
*macros for declaring temps
*
qrcell equ 0
qloc equ 0
qdecic equ 0
qx equ 0
*
step macro
qx equ qloc-qdecic
if qx-qrcell
qrcell equ qx
endif
endm
*
block macro
qrcell equ 0
qloc equ *
qdecic equ *
endm
*
endblk macro
bss qrcell
endm
*
decl macro d
qx narg
rpt (qy=1,qx)
d(qy) equ qloc
qloc equ qloc+1
endr
step
endm
*
samloc macro
if squeez
qloc equ qdecic
endif
endm
*
decb macro d
d(0) equ qloc
qloc equ qloc+d(1)
step
endm
*
decl macro d
rpt (qy=1,d(1))
d(0).($qy) equ qloc
qloc equ qloc+1
```

DATA PAGE AND SYMBOLS

```
    endr
    step
    endm
* temporary declarations
  block always overlaid temps
    apchr decl 2; samloc
    ldchr decl 4; samloc
    newab decl 4; samloc
    cpysr decl 3; samloc
    hash decl 2; samloc
    decl mvfrm,mvtoo,mvnw; samloc
    decl cksum1; samloc
    decl nwrfl; samloc
    decl clrll; samloc
    lass decl 3; samloc
    ovlad decl 2; samloc
    ovlgo decl 2; samloc
    resd decl 2; samloc
    frzrf decl 3; samloc
    compr decl 3; samloc
  endblk
  block non-overlaid temps
    decl cdpnic,swkflg,gtflgl,stsuc1
    decl stsub1,intr81,cptsrc
    decl litlc,litcb,litlcl
    decl ovrelf
    decl prmwd,pass4d,outerf,setfl1
    decl outffg,ttynf,updtfg
    decl litrf
    inpr1 decl 2
    apsr decl 4
    decl stfhd1,stft11,stssf1,stonm1,stsdbl,stvdbl
    stflg decl 2
    strsv decl 2
    ldrsv decl 2
    ovlid decl 3
    lknmh decl 5
    stvct decl 4
    stpos decl 4
    getlv decl 2
    decl stvt1
    ldrfb decl 4
    fecht decl 3
    decl fechr,rerra,rerrb,rerrx
    decl rerrn,rerrz
    ldsdb decl 2
    fechn decl 7
    fechm decl 7
    gtwk decl 4
    cvsno decl 4
    dism decl 5
```


DATA PAGE AND SYMBOLS

```

decl clrd1,d1r11
clra decl 2
decl inpfpt input fork panic table
    infp decl 6
decl opnfpt open file fork panic table
    opnfp decl 6
decl wtfpt alarm fork panic table
    wtfp decl 6
decl tadfpt time and date fork panic table
    tadfp decl 6
decl outfpt output fork panic table
    outfp decl 6
rlr decl 2 current relabeling
decl ovcl,sovl,gcftl,inptfg
decl paslfg,flcufg,flcufer,opnfg
decl inptfa,usrpts,flpts,nmpts,spcpts
sptr1 decb 2 delimit string to be concatenated (cpr)
sptr2 decb 2
decl cdlit,cdlitc,cdclrs,almwt
decl almst,cdlitf
inppr decb 3
klt decb 45
frzlst decb 10
decl frzlsx
relst decl 2
utt decb 113
endblk
lit11 equ lit1c1+40000b indirect pointer to lit reg length
block temps for recint, inpfbk, sdbmnp, linksub, didd1, keywd,
auxcod
    samloc for recint
    rit decb 42 recint temps
    axt decb 34 auxcod temps
    lst decb 10 linksub temps
    ddt decb 45 didd1 temps
    kwt decb 18 keywd temps
    decl setin2,setin3,lodprl
    crfnm decl 2
    init decl 5
    timms decl 3
    setinu decb 6
    samloc for inpfbk
    ift decb 73
    decl pshspl,inptmr,inptcl
    decl inptpr,inptml,intrvl
    decl incse,inpt0c,inptlc
    decl inptrf,inptfl,inptcm
    decl inpfsl,inp,inwait,inbttn
    inptc decl 3
    nwrsv decl 3
    nwrvb decl 3

```

DATA PAGE AND SYMBOLS

```

rdhdr  decl  2
fhux   decl  6
decl  fhuxba,fhuxl,fhuxz,fhuxa
lkptr  decl  2
decl  asrfrm,asrtoo,asrnw,asrrem,asrnc
numd   decl  4
astnm  decl  4
aplit  decl  2
cdlit  decl  9
decl  cdhpol,csel
frsv   decl  2
frevd  decl  5
numreg decb  21
samloc for sdbmnp
mst    decb  26
cpr    decb  2
cptsr  decl  3
apasr  decl  4
apts   decl  10
comsr  decl  4
isrom  decl  3
gcolx  decl  7
gcol   decl  6
cmptr  decl  9
fndrm  decl  3
nwsdb  decl  3
edsdb  decl  3
decl  frsdbl,rtncfl,pdctl,pdct2,delstl
xtrn   decl  2
delpt  decl  4
inspt  decl  5
fxptr  decl  8
samloc for fmtout
sgt    decb  41
fchs   decl  7
kyfch  decl  6
endblk

block  temps for clnup
clt    decb  45  maybe less
clnup  decl  4
crsv   decl  6
chkdb  decl  4
decl  unalbk,cksmer,rfbser,stfler,wrngty
decl  ringer,lsteer,lstser,lstver,flperr
decl  fsper,udcler,flerr,stbner
decl  sdbher,vdbher,bdchr,fcheck,fstore
endblk

block  temps for spclib, fiolib, cdsply
samloc for spclib
plt    decb  32
nnt    decb  34

```

DATA PAGE AND SYMBOLS

```
decl  setlt1,setlt2,taill,subts1,typdt1
decl  setlt3
fedlt  decl  3
pred  decl  2
seept  decl  4
usd  decl  5
sttus  decl  2
samloc  for fiolib
ilt  decb  38
decl  frmfil,opnfs1,popvw,popid,hfuno
decl  clrjst,bmphst,hflnkl
rtnhs  decl  3
pshhs  decl  3
fuser  decl  2
frechr  decl  2
opnfl  decl  5
opnfp  decl  2
copbk  decl  4
wrtbkl  decl  1
copfl  decl  6
samloc  for cdsply
cdt  decb  45
cred  decl  4
samloc  for txtedt
tet  decb  12
tst  decl  5
tstf  decl  5
endblk
* file names
  crfnm  asc  '(sys)/crash'
* pop code definitions
wsi  opd  100b5,1,1,0,1  work station input syspop
any  opd  177b5,1,1  all no-address pops are anypops
err  opd  175b5,1,1
abort  opd  136b5,1,1  abort a command
bt  opd  101b5,1,1
bf  opd  102b5,1,1
lec  opd  103b5,1,1
les  opd  104b5,1,1
lai  opd  105b5,1,1
lbi  opd  106b5,1,1
sbc  opd  107b5,1,1
gset  opd  110b5,1,1
mlfl  opd  111b5,1,1
rep  opd  113b5,1,1
las  opd  114b5,1,1
sap  opd  115b5,1,1
laa  opd  116b5,1,1
saa  opd  117b5,1,1
lap  opd  120b5,1,1
lsa  opd  121b5,1,1
```

DATA PAGE AND SYMBOLS

```

arb  opd  122b5,1,1
bfs  opd  123b5,1,1
brc  opd  124b5,1,1
ccp  opd  125b5,1,1
ccv  opd  126b5,1,1
scv  opd  127b5,1,1
lbp  opd  166b5,1,1,0,1
ln   opd  114b5,1,1
lpp  opd  166b5,1,1,0,2
pps  opd  130b5,1,1
sd   opd  131b5,1,1
cpf  opd  132b5,1,1
cchr opd  133b5,1,1
pdc  opd  134b5,1,1
atc  opd  135b5,1,1
sov  opd  137b5,1,1
ovc  opd  140b5,1,1
mlf  opd  141b5,1,1
mlfr opd  142b5,1,1
brv  opd  143b5,1,1
dfs  opd  144b5,1,1
lass opd  145b5,1,1
tch  opd  146b5,1,1
cta  opd  147b5,1,1
ovl  opd  176b5,1,1
*overlay relabeling table
ovrlt equ *
block relabeling table
  decl tnothg,tutilt,tinpfb,tdisbu,trecin,tauxco,tmnctr
  decl ttodal,ttoda2
  decl tprmsp,tdiddl,tkeywd,tcalc,tioctl,tvcted,tvcte2
  decl tstrmn,tclnup,tseqgen,tcacmp,tcadat,tsdbmn%todas%
  decl txtted,tlnksu,tcdspl,trfbpl,trfbp2,tnrfl,tnrfl2
  decl tnlsda,tphary,tphdat,tphc2,tphcod,tphc3
  decl toutov,toutv0,toutv1,toutv2,toutv3,toutv4
  decl toutv5,toutv6
  endblk
cortb data 0,0,0,0,0,0,0,0 contains overlay num. in that page
* content analyzer room
  cacode bss cacodl c.a. code area
  cacdnd bss 1 end of area
* global display parameters
cdctaf data 0 content flag
cdbrf data 0 branch only flag
cdtrlf data 0 trail flag
cdrlev data 0 relative level
cdlev data 100,100 level
cdabfl data 0 abbreviated feedback line
ltflag data 0
cdptr data 1
cdtrn data -1,-1 truncation

```

DATA PAGE AND SYMBOLS

```

cdblnk data 0,0 blank line flag
cdindf data 1 indenting
cdnamf data 1 names
cdstnf data 0 loc-nums
stnumf data 0 flag for seqgen
cdfzrf data 0 frozen
cdtref data 0 tree
cdidtf data 0 initials, date, and time
cdclpf data 0 clip pictures
cdkeyf data 0 keyword flag
kypsid data 0 for psid of best in keyword
cdpsid data 1 display start
cdwork bss 13
* file i/o parameters
cnlsfn bss 1 c-nls file number
ckptfn bss 1 checkpoint file number
curfn bss 1 current file number
rffn bss 1 random file file number
rfsn data 0 teletype number for scratch file
rfwfg data 0 random file write-ok flag (i/o fork?)
outfn bss 1 output file number
wpsid data 0 display start for working copy
wltwd data 0 view spec for working copy
* initial file parameters
inmdl1 equ 10b left name delimiter
inmdl2 equ 11b right
irals equ 50 average length
* file header block area
filhed equ *
bss 1 checksum (not computed)
data 20
data 2 block type (header)
bss 4
fcredt data 0 file creation date
nlsvwd asc '1.1' nls version word (keyword)
hedsz data ihedsz number of words in this block
finit data 0 initials at last write
funo data 0 user number (file owner)
lwdat data 1 last write date
lwtim data 0 last write time
namdl1 data inmdl1 left name delimiter
namdl2 data inmdl2 right
rals data irals running average length of sts
tnsg bss 1 total number of st. generated
*status blocks
rfbsx data rfbsn-1
rfbst1 equ rfbsx
rfbs bss rfbsn random file block status
sdbst1 data sdbstn-1
sdbst bss sdbstn sdb status block
rsvst1 data rsvstn-1

```

DATA PAGE AND SYMBOLS

```

rsvst  bss  rsvstn  rsv status block
ptrtbl data  0  pointer table length
ptrtxn data  ptrtbx-1
ptrtb  bss  ptrtbx
vdbstl data  vdbstn-1
vdbst  bss  vdbstn  vdb status block
qbst  data  0  query status block (one cell)
*
hedend equ  *
ihedsz equ  hedend-filhed
freeze
frgt  debug,ascsp,dirl,dirr,cpflf,cpfle,cpfsf,cpfse
frgt  rellt,relle,releq,relge,relgt,relne,ccsp,cctab
frgt  cclld,ccl,ccd,cch,ccnp,ccpt,cccr,bggap,bglg,bgnlg
frgt  bgpts,bgnps,bgls,bgds,bglds,ncasv,grspec,grscan
frgt  gredit,grhop,grjump,groupx,cmfl,cmfd,cmfp,cmfsp
frgt  cmftb,cmfcr,numnn,litln,stnn,stnon,dsarn,sarn
frgt  rficbn,esn,blkhdr,spskx,na,stnox,numnx
frgt  tty,disfpn,datfpn,cdskx,inmdl1,inmdl2,inptbc
frgt  qrcell,qloc,qdeclc,qx,qy,squeez,ncasvl,iocfpn,strfpn
end
* the following equs should go into data, but won't until the new mol
  *is ready because of symbol table problems.
asce   equ   45b e
asch   equ   56b n
asco   equ   57b o
ascs   equ   63b s
ascro  equ   146b rub out (for output)
ascbst equ   161b back space statement (control a)
ascpct equ   5b percent
ascbel equ   147b bell
ascrtp equ   11b right paren
ascslh equ   17b slash
ascbks equ   74b backwards slash
ascuar equ   76b up arrow
ascbka equ   77b back arrow
ascdot equ   16b period
asceq  equ   35b equal sign
ascf   equ   46b f
ascg   equ   107b g
ascq   equ   61b q
ascy   equ   71b y
ascctc equ   ascdot will eventually be redefined as ?(maybe ca)
ascast equ   12b asterisk
asclbk equ   34b left broken bracket
ascrbk equ   36b right broken bracket
asccom equ   14b comma
ascamp equ   6b ampersand
ascaps equ   7b apostrophe
ascqus equ   37b question mark

```

:UTILTY, 11/16/69 2023:44 WHP ;

1 .IGD=1;

2 Description of UTILTY overlay

2a The UTILTY overlay contains MOL procedures. which are used by all other overlays. They are used frequently, and should always be available to other routines. Hence they are always relabeled into addressable core, currently in core page 3.

2b Also in the same overlay are several tables. and constants.

2c This page is read-only, and shared.

3 %Procedures in the UTILTY overlay%

3a DECLARE origin(13777B);

3b (contin)PROCEDURE; %for starting NLS%

3b1 %continue point for NLS%

3b2 %This procedure must always start at location 14000%

3b3 EXECUTE recint; GOTO rsttrl END.

3c DECLARE EXTERNAL

3c1 regary=(1,smareg,smbreg,smec,smcreg,smes,

3c1a numn1,lit11,stn1,stnol,fnml,stn21,1,sar1),

3c1b regcv=(0,0,0,0,0,0,-1,-1,-1,-1,-1,-1,-1,-1);

3d DECLARE EXTERNAL

3d1 emdopt=(1,2,4,8), %underline,italics,bold,flicker%

3d2 lcmsk=(0,77600000B,77777400B),

3d3 chrmsk=(77600000B,00177400B,377B),

3d4 endchr=377B, head1=600000B, icksu=0, ttyno=-1,

3d5 igrps=(0,statem,charac,item,vector),

3d6 igrpc=(0,63B,43B,51B,66B),

```

3d7 popcod=17600000B,nopcod=2000000B;
3e DECLARE EXTERNAL
3e1 statem =(8,8,"STATEMENT"),
3e2 item =(3,3,"ITEM"),
3e3 charac =(8,8,"CHARACTER"),
3e4 vector =(5,5,"VECTOR");
3f DECLARE rficbx =3, %rficbn-1%
3f1 blksiz = 2000B, %randomn file block size blkszn %
3f2 rfam = 177777B, % random file block address mask
blkszn*rfbsn-1%
3f3 lfrwdp = 8, %initial free word pointer blkhdr+1 %
3f4 rsvhdr = 2, %missing psids per block due to header%
3f5 rsvfpt = 1775B; %rsv block full free word count%
3g DECLARE EXTERNAL chr1sh = (/LSH 16/, /LSH 8/, /NOP/);
3h DECLARE EXTERNAL chr2sh = (/LRSH 16/, /LRSH 8/, /NOP/);
3i DECLARE EXTERNAL crpgad = (4000B,6000B,10000B,12000B);%
vm,vm+Blkszn,vm+2*Blkszn,vm+3*Blkszn%
3j %overlay table..instructions
3j1 INSTRUCTIONS FOR CHANGING
3j1a output the table to a qed file, and assemble in arpas.
3j1b Then load the table into ddt with the table conversion
program, and insert the resulting table into this file
3j1c TABLE CONVERSION PROGRAM
3j1c1 compile this program with mol and arpas.
3j1c2 After compilation, load it into oddt along with the
binary from the table (as produced below).
3j1c3 Define the symbol TABLE as the starting address of
the binary table produced below.
3j1c4 then typ start;g, followed by a file name, which will
bbe the file on which the final overlay tble will be

```


written.

3jlc5 after running the progrm, do an insert qed branch (with convert case) to get the table into utility.

3jlc6 (cvrt)procedure;

3jlc6a declare t,t1;

3jlc6b declare comma = ' ,';

3jlc6c declare semi = ' ;',

3jlc6c1 filmes = 'file name=/',

3jlc6c2 donmes = '\$done /',

3jlc6c3 decmes = 'declare /';

3jlc6d (start):

3jlc6e .ar + .brs34(\$filmes,-1,1);

3jlc6f .ar+ .brsl6(03000000b,0,0);< data 0>;

3jlc6g t+.ar;

3jlc6h t1+0;

3jlc6i goto first;

3jlc6j while table/t1/ .ne 0 do begin

3jlc6j1 bump t1;

3jlc6j2 .ar + .brs34(\$table/t1/,-1,t);

3jlc6j3 .ar + table/t1-1/;

3jlc6j4 .ar+.brs36(,8,t);

3jlc6j5 .ar + h2b; < cio t>;

3jlc6j6 t1 + +3;

3jlc6j7 if table/t1/ .eq 0 then goto exit;

3jlc6j8 .ar + comma; < cio t>;

3jlc6j9 (first):.ar + 155b; < cio t>;

3jlc6j10 .ar + 155b; < cio t>;

```
3jlc6j11 .ar ← 21b; < cio t>; .ar ← 41b; < cio t>; if
t1 then begin .ar ← 21b; < cio t> end; .ar ← 0; < cio
t>;
```

```
3jlc6j12 if t1 .eq 0 then begin .ar ←
.brs34($decmes,-1,t) end
```

```
3jlc6j13 end;
```

```
3jlc6k (exit): .ar ← semi; < cio t>;
```

```
3jlc6l .ar ← .brs17();
```

```
3jlc6m .ar ← .brs34($donmes,-1,1);
```

```
3jlc6n .ar ← .brs10()
```

```
3jlc6o endp.
```

```
3jlc6p finish
```

```
3jld TABLE
```

```
3jld1 .CSW=1; .CMD=1; .DSN=1; .SCR=1; .LSP=0; .DLS=0;
.FLN=0; .PSW=0;
```

```
3jld1a rad 2
```

```
3jld1b ov1 opd 176b5,1,1
```

```
3jld1c needs macro d
```

```
3jld1c1 d(0) ov1 (r)d(1)-(r)nothg+d(2)
```

```
3jld1c2 numchr nchr d(0)
```

```
3jld1c3 IF numchr=5
```

```
3jld1c3a asc '.d(0). = /'
```

```
3jld1c4 ELSF numchr=4
```

```
3jld1c4a asc '.d(0). = /'
```

```
3jld1c5 ELSE
```

```
3jld1c5a asc '.d(0). = /'
```

```
3jld1c6 ENDF
```

```
3jld1c7 ENDM
```

```
3jld1d nothg needs nothg,0
```

3jldle utilty needs nothg,3
3jldlf inpfbk needs disbuf,4
3jldlg disbuf needs nothg,7
3jldlh recint needs nothg,2
3jldli auxcod needs recint,5
3jldlj mnctr1 needs inpfbk,5
3jldlk todas1 needs inpfbk,5
3jldll todas2 needs inpfbk,5
3jldlm prmspc needs inpfbk,6
3jldln didd1 needs inpfbk,5
3jldlo keywd needs inpfbk,5
3jldlp calc needs inpfbk,6
3jldlq ioctl needs inpfbk,5
3jldlr vctedt needs inpfbk,5
3jldls vctrl needs vctedt,6
3jldlt strmp needs nothg,4
3jldlu stredt equ strmp
3jldlv pass1 equ calc
3jldlw clnup needs inpfbk,6
3jldlx seqgen needs nothg,6
3jldly cacmpl needs seqgen,7
3jldlz cadata needs seqgen,4
3jldla@ sdbmnp needs seqgen,5
3jldlaa txtedt needs sdbmnp,7
3jldlab lnksub needs txtedt,4
3jldlac cdsply needs seqgen,5
3jldlad rfbpl needs nothg,2

```
3jldlae rfbp2 needs rfbp1,1
3jldlaf nrfbp1 needs nothg,2
3jldlag nrfbp2 needs nrfbp1,1
3jldlah nlsdat needs nothg,0
3jldlai phary needs phcod1,7
3jldlaj phdata needs seqgen,5
3jldlak phc2 needs phcod1,7
3jldlal phcod1 needs phdata,4
3jldlam phc3 needs nothg,6
3jldlan outov1 needs seqgen,7
3jldlao outv0 needs nothg,0
3jldlap outv1 needs nothg,1
3jldlaq outv2 needs nothg,2
3jldlar outv3 needs nothg,3
3jldlas outv4 needs nothg,4
3jldlat outv5 needs nothg,5
3jldlau outv6 needs nothg,6
3jldlav end
```

3j2 OVERLAY TABLE ITSELF: %

```
3j2a SET ovltb = *;
```

```
3j2b DECLARE nothg = 17600000B,
```

```
3j2b1 utilty= 17600003B,
```

```
3j2b2 inpfbk= 17600034B,
```

```
3j2b3 disbuf= 17600007B,
```

```
3j2b4 recint= 17600002B,
```

```
3j2b5 auxcod= 17600045B,
```

```
3j2b6 mnctrl= 17600025B,
```

3j2b7 todas1= 17600025B,
3j2b8 todas2= 17600025B,
3j2b9 prmspc= 17600026B,
3j2b10 didd1 = 17600025B,
3j2b11 keywd = 17600025B,
3j2b12 calc = 17600026B,
3j2b13 loctl = 17600025B,
3j2b14 vctedt= 17600025B,
3j2b15 vctrl = 17600166B,
3j2b16 strmnp= 17600004B,
3j2b17 clnup = 17600026B,
3j2b18 seqgen= 17600006B,
3j2b19 cacmpl= 17600227B,
3j2b20 cadata= 17600224B,
3j2b21 sdbmnp= 17600225B,
3j2b22 txtedt= 17600257B,
3j2b23 lnksub= 17600264B,
3j2b24 cdsply= 17600225B,
3j2b25 rfbp1 = /NOP/, %for brs -13 ptnh%
3j2b26 rfbp2 = /NOP/, %for brs -13 ptnh%
3j2b27 nrfbp1= 17600002B,
3j2b28 nrfbp2= 17600331B,
3j2b29 nlsdat= 17600000B,
3j2b30 plary = 17600417B,
3j2b31 pldata= 17600225B,
3j2b32 plc2 = 17600417B,
3j2b33 plcod1= 17600374B,

```

3j2b34 p4c3 = 17600006B,
3j2b35 outov1= 17600227B,
3j2b36 outv0 = 17600000B,
3j2b37 outv1 = 17600001B,
3j2b38 outv2 = 17600002B,
3j2b39 outv3 = 17600003B,
3j2b40 outv4 = 17600004B,
3j2b41 outv5 = 17600005B,
3j2b42 outv6 = 17600006B;

```

3k (push) %All routines that need to push a value onto the general stack DO it by calling this routine. It first checks for stack overflow, THEN pushes the contents of the B register onto the stack. A stack overflow results in an exceed capacity ERROR.%

3l PROCEDURE;

```

3l1 %push the b register onto the general stack%
3l2 IF stackt <= stack DO-SINGLE !err $6;
3l3 BUMP stack;
3l4 [/stack/ ← .BR;
3l5 RETURN END.

```

3m (pop) %Being the reverse of push, POP removes the top element of the stack and RETURNS it in the B register. It checks for stack underflow first, and calls RERROR IF underflow has occurred.%

3n PROCEDURE;

```

3n1 %pop the general stack 1 cell, into the b register%
3n2 IF stack <= stack1 DO-SINGLE rerror();
3n3 .BR ← [/stack/; stack ←+ -1 END.

```

3o (apchr) %The calling arguments are: a character in the A, and the address of an A-string in the X. The character gets appENDED to the

END of the A-string. If the current length of the A-string is the same as the maximum length (i.e. the string is full) when this routine is called, an ERROR 6 (exceed capacity) is generated. Character positions within a word, following the character appended, are set to blanks.%

```

3p PROCEDURE (apchr1,,apchr2);
  3p1 IF $0[.XR] <= $1[.XR] DO-SINGLE !err $6;
  3p2 BUMP $1[.XR];
  3p3 apchr2+ $1[.XR]/3 + 2 +apchr2;
  3p4 .XR + .BR;
  3p5 .BR + apchr1;
  3p6 EXECUTE chr1sh[.XR];
  3p7 .AR + [apchr2] .A lcmsk[.XR];
  3p8 COPY (AX,BX,XA); XXA;
  3p9 [apchr2] + .AR;
  3p10 RETURN END.

```

3q (apstr) %The purpose of this routine is to append one A-string onto another. The routine is called with the addresses of the A-strings in the A and the X. The string specified by the A is appended to the string specified by the X. An ERROR 6 is generated IF the maximum length of the latter string is too short to contain the result. The string is appended character by character using LDCHR and APCHR.%

```

3r PROCEDURE (apsr1,,apsr2);
  3r1 apsr3+.AR+1;
  3r2 IF [apsr3] + $1[.XR] > $0[.XR] DO-SINGLE !err $6;
  3r3 apsr4 + 0;
  3r4 WHILE [apsr3] >= apsr4 DO BEGIN
    3r4a ldchr(apsr1,apsr4);
    3r4b apchr(.AR,,apsr2); BUMP apsr4 END;
  3r5 RETURN END.

```

3s (ldchr) %The function of this routine is to load a specified character from an A-string. Upon entry, the A contains the address of the A-string, and the B contains the number of the character desired. The character is returned in the A, and the X is preserved. Note that the first character in the string is numbered zero. The routine will not check for legality of the character number.%

3t PROCEDURE (ldchr1,ldchr2,ldchr3);

3t1 ldchr4 ← ldchr2/3+ldchr1+2;

3t2 .XR ← .BR;

3t3 .AR ← [ldchr4] .A chrmsk[.XR]; EXU chrrsh[.XR];

3t4 RETURN(.,,ldchr3) END.

3u (cpysr) %To copy one A-string into another, CALL this routine with the address of the source string in the A, and the address of the string to write in, in the X. The move is done by words rather than characters, so this routine is quite fast. An ERROR 6 is generated IF the maximum length of the destination string is shorter than the source string.%

3v PROCEDURE (cpysr1,,cpysr2);

3v1 BUMP cpysr1;

3v2 IF \$0[.XR] ≤ [cpysr1] DO-SINGLE !err \$6;

3v3 BUMP cpysr2;

3v4 cpysr3 ← [cpysr1]/3+2;

3v5 .AR ← - .AR; COPY (AX,N);

3v6 cpysr1 ←+ .AR .V 2B7;

3v7 cpysr2 ←+ .AR;

3v8 (cpysr1): [cpysr2] ← [cpysr1]; BRX cpysr1;

3v9 RETURN END.

3w (err) %ERROR messages are generated using this pop, number 175. This routine displays a message, and calls RESNLS to reset various things and restart at the previous state. The error message is indicated by the address of the pop. The code to DO this is in the RECINT overlay, following label XERR.%

3x POP 175B PROCEDURE(rerra,rerrb,rerrx);

3x1 EAX [extract_itex]{0} (,,0); rerrn + .XR; %error number%

3x2 EXECUTE auxcod; GOTO xerr END.

3y (rerror) %This routine is called when an error of a very serious kind is detected. Currently, RERROR dumps the NLS core image on file (NLS)/CRASH, writes "OOPS" followed by the calling location, and calls INTR5 to terminate NLS. The code to DO this is in the RECINT overlay, following label XRERR.%

3z PROCEDURE(rerra,rerrb,rerrx);

3z1 rerrz +[/extract_itex]{0};

3z2 %for debugging purposes%

3z3 EXECUTE auxcod; GOTO xrerr END.

3a@ (abort) %When the NLS user does something wrong or types a command that cannot be carried out, this pop is used to abort the command. This is pop number 136: it simply displays a message indicated by the address of the pop, and calls RESNLS. The code to DO this is in the RECINT overlay, following label XABORT.%

3aa POP 136B PROCEDURE;

3aa1 rerrz +[/extract_itex]{0};

3aa2 EAX [extract_itex]{0} (,,0); rerrn + .XR;%abort number%

3aa3 EXECUTE auxcod; GOTO xabort END.

3ab (ursnls) %The purpose of this routine is to restart NLS at the previous state. It does this by calling RESNLS in the RECINT overlay, which checks various flags to see if anything should be reset before going to previous state. It checks PASS1FG to see if PASS1 was in use, and calls XINTR9 to reset the input fork's relabeling if the flag is set.%

3ac PROCEDURE;

3ac1 EXECUTE auxcod; resnls() END.

3ad (compr) %This routine is used by several pops to compare two

numbers (binary integers). A relation is given in the X, and the two numbers in the A and B. It RETURNS the result of the compare in the A. The numbers corresponding to the relations are specified by the relations defined in ADATA.%

```
3ae PROCEDURE(compr1,compr2,compr3);
```

```
3ae1 IF compr3 = 1 THEN
```

```
3ae1a .AR ← IF compr1 < compr2 THEN 1 ELSE 0
```

```
3ae2 ELSE IF compr3 = 2 THEN
```

```
3ae2a .AR ← IF compr1 <= compr2 THEN 1 ELSE 0
```

```
3ae3 ELSE IF compr3 = 3 THEN
```

```
3ae3a .AR ← IF compr1 = compr2 THEN 1 ELSE 0
```

```
3ae4 ELSE IF compr3 = 4 THEN
```

```
3ae4a .AR ← IF compr1 >= compr2 THEN 1 ELSE 0
```

```
3ae5 ELSE IF compr3 = 5 THEN
```

```
3ae5a .AR ← IF compr1 > compr2 THEN 1 ELSE 0
```

```
3ae6 ELSE IF compr3 = 6 THEN
```

```
3ae6a .AR ← IF compr1 NOT= compr2 THEN 1 ELSE 0
```

```
3ae7 ELSE rerror(); RETURN END.
```

3af % The following routines are used to read various fields from RING elements. They all require a PSID in the A upon entry. Each of these routines calls LODRSV to get the RING element into core.%

```
3af1 (GETSUC) %The PSID in the successor field is RETURN in the A.%
```

```
3af2 PROCEDURE; lodrsv(.AR); RETURN($1/.XR/ .A 3777B) END.
```

```
3af3 (GETSUB) %The PSID in the sub field is RETURN in the A.
```

```
3af4 PROCEDURE; lodrsv(.AR); RETURN(LRSH $13($1/.XR/)) END.
```

```
3af5 (GETFTL) %The logical value of the tail flag is RETURNED in the A.%
```

3af6 PROCEDURE; lodrsv(.AR); RETURN (\$1/.XR/ .A 4000B) END.

3af7 (GETFHD) %The logical value of the head flag is RETURNed in the A.%

3af8 PROCEDURE; lodrsv(.AR); RETURN (\$1/.XR/ .A) END.

3af9 (GETFLG) %Given a flag number in the X, the logical value of that flag is RETURNed in the A.%

3af10 PROCEDURE;

3af10a gtflgl ← .XR+1;

3af10b lodrsv();

3af10c .BR ← \$0/.XR/; %flag word%

3af10d RETURN(LSH \$0/gtflgl/ .A 1) END.

3af11 (GETSDB) %The PSDB in the specified ring element is RETURNed in the A.%

3af12 PROCEDURE;

3af12a lodrsv(.AR); %load proper ring block%

3af12b RETURN (\$0/.XR/ .A 77777B) %psdb% END.

3af13 (GETNAM) %The 24 bit hash word for the statement name is RETURNed in the A.%

3af14 PROCEDURE; lodrsv(); RETURN(\$2/.XR/) END.

3af15 (GETVDB) %The PVDB in the specified ring element is RETURNed in the A.%

3af16 PROCEDURE; lodrsv(.AR); RETURN(\$3/.XR/ .A 77777B) END.

3ag (fechcl) %This routine is called to initialize a work area for reading characters from a statement. It is called with the address of the 15 word work area in the X. The A indicates the direction in which characters are read from the statement. A zero will cause the routine to read the characters out backwards, and non-zero will result in the reading forward.

3ag1 When calling FECHCL, the first two cells of the work area

must contain the PSID and character count for the first character to be read, respectively. A character count of one indicates the first character of the statement. FECHC1 will initialize the rest of the work area. For a description of the work area, see ...

3ag2 Calling FECHC1 initializes the work area. To read characters from the statement, EXECUTE a BRM 2,2 with the address of the work area in the X. The character is RETURNED in the A register. Subsequent BRM's will RETURN the following characters in the A. After RETURNING the last character of the statement (or first IF the direction of readout is backward), ENDCHARS (code 377b) will be RETURNED from subsequent calls indefinitely.

3ag3 To change position within the statement, change direction, or read from a different statement, the work area must be reinitialized by calling FECHC1 again, as described above.%

3ah PROCEDURE(fechdr,,fecht1);

3ah1 getsdb(\$O/.XR/); lodsdb(.AR);

3ah2 .AR ← .XR+7; .XR ← fecht1;

3ah3 \$12/.XR/ ← .AR; \$7/.XR/←\$12/.XR/+\$1/.XR//3;

3ah4 fecht1←.BR;

3ah5 IF NEG fecht1 DO-SINGLE rerror();

3ah6 \$3/.XR/←20140004B+fecht1; %bru* 4,2%

3ah7 IF fechdr THEN BEGIN

3ah7a \$4/.XR/←\$ffch1; \$5/.XR/←\$ffch2; \$6/.XR/←\$ffch3 END

3ah8 ELSE BEGIN

3ah8a \$4/.XR/←\$bfch1; \$5/.XR/←\$bfch2; \$6/.XR/←\$bfch3 END;

3ah9 \$8/.XR/←LSH \$8(0,[\$7/.XR/]);

3ah10 \$9/.XR/←LSH \$8(0);

3ah11 \$10/.XR/←LSH \$8(0);

3ah12 RETURN;

3ah13 (ffch1):

3ah13a BUMP \$1/.XR/, \$3/.XR/;

3ah13b \$8/.XR/←LSH \$8(0,[\$7/.XR/]);

```

3ah13c $9[.XR]←LSH $8(0);
3ah13d $10[.XR]←LSH $8(0);
3ah13e IF $8[.XR] NOT= endchr DO-SINGLE BRR $2[.XR];
3ah14 (ffche): $3[.XR] ← .BR ← fchenb; BRR $2[.XR];
3ah15 (fechenb): GOTO fchend;
3ah16 (ffch2): BUMP $1[.XR], $3[.XR];
3ah16a IF $9[.XR] NOT= endchr DO-SINGLE BRR $2[.XR]; GOTO
ffche;
3ah17 (ffch3): BUMP $1[.XR], $7[.XR];
3ah17a $3[.XR]←20140004B;
3ah17b IF $10[.XR] NOT= endchr DO-SINGLE BRR $2[.XR]; GOTO
ffche;
3ah18 (fchend): BRR $2[.XR](endchr);
3ah19 (bfch1): $1[.XR], $7[.XR]←+ -1;
3ah19a $3[.XR]←20140006B;
3ah19b IF $8[.XR] NOT= endchr DO-SINGLE BRR $2[.XR]; GOTO
ffche;
3ah20 (bfch2): $3[.XR]←+ -1; $1[.XR] ←+ .AR; BRR
$2[.XR]($9[.XR]);
3ah21 (bfch3): $1[.XR], $3[.XR]←+ -1;
3ah21a $8[.XR]←LSH $8(0,[$7[.XR]]);
3ah21b $9[.XR]←LSH $8(0);
3ah21c BRR $2[.XR](LSH $8(0)) END.

```

3ai (fchsdb) %This routine is called with a PSID in the A, in order to load the text for that statement into core. It RETURNS the address of the SDB in the X, and the SDB block number in the A.%

3aj PROCEDURE; RETURN (lodsdb(, , getsdb(.AR)))END.

3ak (newrfb) %NEWRFB is called to obtain a new random file block. If no more blocks are available, RERROR is called. Otherwise the random

file block status (RFBS) for that block is set to minus one (meaning assigned but not initialized), and the random file block number is RETURNED in the A. LODRFB will initialize the block when it is loaded into core for the first time.%

3a1 PROCEDURE;

3a11 nwrfb1 ← 0;

3a12 WHILE nwrfb1 ≤ rfbsx DO IF rfbs/nwrfb1 = 0 THEN BEGIN

3a12a rfbs/nwrfb1 ← -1; % mark assigned but not initialised%

3a12b RETURN(nwrfb1) END

3a13 ELSE BUMP nwrfb1;

3a14 rerror() END.

3am (lodrfb) %This is the PROCEDURE that everyone calls to have a random file block loaded into core. Also, it is possible to obtain a block of core which is empty and not associated with the file. For example, a long literal register is created in this way. There are several 1K blocks in core, some of which may be "frozen", which means that they may not be moved or used for anything else. LODRFB loads the desired file block into one of these core blocks. On entry, the A contains the random file block number, and the B contains the block type, or -1 IF no file block is to be read into that core block. The algorithm is approximately as follows:

3am1 First, a block is chosen. A quick scan is made to find an unused block. If all are in use, THEN a circular counter (NACP) is used to find the "next" core block that is not frozen. If all are frozen, RERROR is called.

3am2 RERROR is called IF the desired block does not exist.

3am3 If the newly found core block contains a file block, then

3am3a IF the file block is empty, it is released to the system and the corresponding status block is set to indicate that that block is not around, or

3am3b that block is written out on the file IF the check sum has changed, and the random file status block is set to indicate that that block is on the file and not in core.

3am3c When releasing the block to the system, or writing the file block, a working copy is created IF NLS is not already working from a working copy.

3am4 At this point, IF the block type is negative, LODRFB RETURNS with the core block number in the X. Otherwise the desired file block is loaded into the core block. The block is read from the current file IF the random file is not open.

3am5 If the random file block has not been initialized, the initialization is done now. Otherwise the checksum and file type are checked. RFBERR is called IF either of these checks fail.

3am6 Finally, the random file block status is set to show that the block is now in core, and the index for core blocks (RFIFCB) is set to indicate which random file block is in that core block.%

3an PROCEDURE(ldrfb1,ldrfb2);

3an1 %block number, block type%

3an2 %choose the next core page to use%

3an3 %look for an unused one%

3an4 ldrfb3 ← 0;

3an5 LOOP BEGIN

3an5a IF ldrfb3 > rficbx THEN EXIT;

3an5b IF rfifcb/ldrfb3/ = 0 THEN BEGIN

3an5b1 IF NEG frzcpt/.XR/ THEN BEGIN

3an5b1a nacp ← .XR; EXIT END END;

3an5c BUMP ldrfb3 END;

3an6 ldrfb3 ← 0;

3an7 LOOP BEGIN

3an7a nacp ← (nacp+1)↑(rficbx+1);

3an7b IF NEG frzcpt/nacp/ THEN EXIT;

3an7c IF rficbx ≤ ldrfb3 DO-SINGLE rerror();

3an7d BUMP ldrfb3 END;

3an8 (ldrfl1):

3an8a %IF file sectors belong to system allocate them%

3an9 IF rfbs/ldrfb1/ = 0 THEN rerror();

```

3an9a  %watch for attempts to load unknown blocks%
3an10  %IF core block is in use write it out, IF it is all free%
3an11  % release it to the system (brslkhs)%
3an12  IF rfifcb/nacp/ NOT= 0 THEN BEGIN
    3an12a IF $1/crpgad/nacp// = ifrwdp THEN BEGIN
        3an12a1 IF NOT rffn DO-SINGLE getwka();
        3an12a2 BRS $1k4 (rfifcb/nacp/*blkksz,10001B,rffn);
        3an12a3 BRS $1k4(blkksz,10007B,rffn);
        3an12a4 IF $2/crpgad/nacp// = 1 THEN sdbst/$4[.XR//] ← 0
        3an12a5 ELSE IF .AR = 3 THEN rsvst/$4[.XR//] ← 0
        3an12a6 ELSE IF .AR = 4 THEN vdbst/$4[.XR//] ← 0;
        3an12a7 rfbs/rfifcb/nacp//←0 END
    3an12b ELSE BEGIN
        3an12b1 ldrfb4 ← cksum(nacp);
        3an12b2 IF $0/crpgad/nacp// NOT= ldrfb4 OR .AR = 0 THEN
        BEGIN
            3an12b2a $0[.XR/] ← ldrfb4;
            3an12b2b IF NOT rffn DO-SINGLE getwka();
            3an12b2c .BR←(rfifcb/nacp/ .A rfam)*blkksz;
            3an12b2d DBO rffn(blkksz,,crpgad/nacp//) END;
        3an12b3 .AR←IF $2/crpgad/nacp// = 1 AND NEG $3[.XR/] AND
        $1[.XR/] > rals+20 THEN -2
        3an12b4 ELSE -$1[.XR/];
        3an12b5 rfbs/rfifcb/nacp//←.AR END END;
3an13  %load the new block%
3an14  IF NEG ldrfb2 THEN BEGIN
    3an14a rfifcb/nacp/ ← 0; RETURN END;
3an15  IF rffn THEN BEGIN

```



```

3an15a .BR ← ldrfbl*blkosz;
3an15b DBI rffn(blkosz,,crpgad/nacp)) END
3an16 ELSE BEGIN
3an16a .BR ← ldrfbl*blkosz;
3an16b DBI curfn(blkosz,,crpgad/nacp)) END;
3an17 %IF a brand new page set it up, IF an old one check cksum%
3an18 IF rfbs/ldrfbl/ = -1 THEN BEGIN
3an18a $0[crpgad/nacp]] ← icksum;
3an18b $1[.XR/ ← ifrwdp;
3an18c $2[.XR/ ← ldrfb2 END
3an19 ELSE BEGIN
3an19a IF cksum(nacp) NOT= $0[crpgad/nacp]] THEN
RETURN(rfberr());
3an19b IF $2[crpgad/nacp]] NOT= ldrfb2 THEN RETURN(rfberr())
END;
3an20 rfbs/ldrfbl/ ← crpgad/nacp);
3an21 rfifcb/nacp/ ← ldrfbl;
3an22 RETURN END.

```

3ao (rfberr) %This routine is called only by LODRFB as indicated above. If file cleanup is in progress, it sets a flag and simulates a RETURN from LODRFB. Otherwise it generaes an ERROR (bad file block).%

3ap PROCEDURE;

3ap1 IF flcufig THEN BEGIN BUMP flcufer; RETURN END;

3ap2 !err \$7 END.

3aq (cksum) %This PROCEDURE takes a core block number in the A and RETURNS the check sum of that 1K block in the A. The checksum does not include the first word of the 1K block (that's where the cchecksum is kept).%

3ar PROCEDURE;

```

3ar1  .XR ← .AR;
3ar2  cksum1 ← crpgad[.XR]+blkasz .V 20000000B;
3ar3  .XR ← -(blkasz-1);
3ar4  .AR ← 0;
3ar5  .AR ← .AR + [cksum1];
3ar6  BRX .-1;
3ar7  RETURN END.

```

3as (ov1) %This is a pop used only in the overlay tables. The address of the pop indicates the overlay number. A cell preceding the pop contains the address of the OVL pop for the overlay required to run this one. The sign bit of that preceding word is on IF this overlay is in core, and is off otherwise. The purpose of the pop is to get this overly in, as well as all others required to run it. It does this recursively, which is not such a good idea. It calls OVLDPG to get the required page relabeled in. When the top level pop is completed, relabeling takes place (BRS 44) IF the internal relabeling has changed.%

3at POP 176B PROCEDURE;

```

3at1  EAX [$/](,,0); .XR←.XR-$ovltb; % ovl NOT= in x %
3at2  WHILE .XR NOT= 0 DO BEGIN
      3at2a IF NOT .NEG ovrlt[.XR] THEN ovrelf ←+ ovld(ovltb[.XR] .A
      7,,.XR);
      3at2b .XR ← RSH $3(ovltb[.XR]) .A 7777B END;
3at3  IF ovrelf THEN BRS $44(rlr1,rlr2);
3at4  ovrelf ← 0; RETURN END.

```

3au (ovldpg) %Given an overlay number in the X, this routine gets that overlay into the internal relabeling (RLR1 and RLR2). It RETURNS zero in the A IF the relabeling has not changed, otherwise it RETURNS non-zero. the data areas OVPGT, OVLTB, and CORTB must be intact for this routine to work correctly.%

3av PROCEDURE(); % overlay adr in x %

```

3av1  RETURN( ovldn(,,.XR-$ovltb) ) END.

```

3aw (lodsdb) %An SDB block is loaded into core by calling this routine with the PSDB in the A. It RETURNS the SDB block number in the A, and the address of the SDB in the X. If the SDB does not exist, it generates it (which is not such a good idea).%

3ax PROCEDURE;

3ax1 ldsdbl ← RSH \$9(,0); %index into sdbst for the sdb block identified by psdb%

3ax2 ldsdb2 ← LSH \$10(0);%relative adress of sdb in block%

3ax3 IF sdbst1 NOT > ldsdbl DO-SINGLE rerror();%illegal sdb block no. (from psdb)%

3ax4 IF sdbst/ldsdbl/ = 0 THEN BEGIN %block un-allocated%

3ax4a sdbst/ldsdbl/←newrfb(); %get new block%

3ax4b lodrfb(.AR,1); %load and initialise blk%

3ax4c \$4/(rfs/(sdbst/ldsdbl/)) + ldsdbl;%number of sdbst entry%

3ax4d \$3/.XR/ ← 0% garbage collection flag (mark not collected)% END

3ax5 ELSE IF rfs/(sdbst/ldsdbl/) <= 0 THEN %block is allocated, but not in memory..load it%

3ax5a lodrfb(sdbst/ldsdbl/,1);

3ax6 .XR ← rfs/(sdbst/ldsdbl/)+ldsdb2; %core address of sdb%

3ax7 RETURN(ldsdbl) END.

3ay (lodvdb) %Given a PVDB in the a-register, load the VDB into core and RETURN the starting location in the x-register and the block number in the a-register.%

3az PROC;

3az1 ldsdbl←RSH \$10(.AR);

3az2 ldsdb2←LSH \$10(0);

3az3 IF vdbst1 <= ldsdbl DO-SINGLE rerror();

3az4 IF vdbst/ldsdbl/ = 0 THEN rerror();

3az5 IF rfs/.AR/ <= 0 THEN lodrfb(vdbst/ldsdbl/,2);

3az6 RETURN(ldsdbl,,rfbs/(vdbst/ldsdbl))+ldsdb2) END.

3b@ (ovlgo) %This routine is called to RETURN to an overlay. The overlay address is provided in the A.%

3ba PROCEDURE (ovlgo2);

3ba1 ovlgo1 ← RSH \$15(,,) .A 377B + \$ovltb;

3ba2 EXECUTE [ovlgo1];

3ba3 BRR ovlgo2 END.

3bb (ovladr) %This routine generates an overlay address and RETURNS it in the A. A 14 bit address is provided in the A, and, using the CORTB to find which overlay is in, the overlay address is created by or-ing in the overlay number in the top part of the word.%

3bc PROCEDURE;

3bc1 ovlad1 ← .AR .A 77777B;

3bc2 .XR ← RSH \$11(,,) .A 7B;

3bc3 RETURN (LSH \$15(cortb/.XR/,0) .V ovlad1 .V 4000000B) END.

3bd (regadr) %The address of a state-machine register is RETURNED, given the number of the desired register in the A.%

3be PROCEDURE;

3bel CASE .AR OF

3bela = esn: LOAD(,,smes);

3belb = numnn: LOAD(,, \$numn);

3belc = litln: LOAD(,, IF litlc THEN .AR

3beld ELSE rerror());

3bele stnn: LOAD(,, \$stn);

3belf = stnon: LOAD(,, \$stn);

3belg = sarn: LOAD(,, \$sar);

3belh = stn2n: LOAD(,, \$stn2);

3beli = fnmn: LOAD(,, \$fnm);

3be2 ENDCASE rerror()

3be3 END.

3bf (intr2) %this procedure handles memory panics..namely those caused by an attempt to access one of the random fille block pages when it is not in thhe relabbeelling.

3bfl When a panic is sensed, a check is made if the system is experimental to see if it is legitimate, and if it is, the random file block page are relablied in%

3bg PROCEDURE(intr2a,intr2b);

3bg1 IF exp THEN BEGIN

3bg1a % firsst check to see that there is nothing in pages 1 or 2%

3bg1a1 IF LSH \$18(,rlr1) .A 7777B DO-SINGLE rerror();

3bg1b %now make sure that the memory panic took place on an access to page 1 or 2%

3bg1b1 .BR←.XR; %save x register%

3bg1b2 IF \$0([intr2]) .A 37777B NOT IN (4000B,13777B) DO-SINGLE rerror();

3bg1b3 .XR←.BR %restore x register%

3bg2 END;%of checks%

3bg3 rlr1 ← rlr1 .V rfbpgs; %put random file block pages into relabelling%

3bg4 BRS \$44(,rlr2); %set system relabelling%

3bg5 LOAD ([intr2a],intr2b); %restor b register..load return address into a register%

3bg6 XMA intr2a; %restore a, and store return addreto intr2a%

3bg7 GOTO [intr2a] %return%

3bg8 END.

3bh (intr4) PROCEDURE;

3bh1 !err \$4; NULL END.

3bi (intr5) %This is an interrupt routine to handle the 205 pseudo-interrupt. That interrupt is used to terminate NLS. The routine closes the display, the working file, and terminates.%

3bj PROCEDURE;

3bj1 IF NOT todas THEN BEGIN % reactivate display tty simulation%

3bj1a IF not .SKIP BRS \$-1(,,ttyno) DO-SINGLE rerror();

3bj1b .AR ← BRS \$-9(1,,ttyno);

3bj1c .AR ← BRS \$-2(,,ttyno)

3bj2 END;

3bj3 .AR ← BRS \$17(); %close all files%

3bj4 curfn ← 0;

3bj5 EXECUTE recint;

3bj6 rlssom(); %release some memory so kdf, etc can run%

3bj7 .AR ← BRS \$ 10() END.

3bk (intr7) %This interrpt (207) is created by the input fork when the main program has requested to be interrupted when a character has come in. The routine simply restarts the main program in the input routine by branching to INPTCX.%

3bl PROCEDURE;

3bl1 IF NOT .NEG tinpfb DO-SINGLE rerror(); %inpfbk is not in relabelling%

3bl2 .AR ← BRS \$78(intmsk); %re--arm interrupts%

3bl3 GOTO inptcx END.

3bm (intr9) %This interrupt is generated by the input fork when PASS1 is completed. The routine calls XINTR9 in INPFBK, and RETURNS to the input routine.%

3bn PROCEDURE;

3bn1 IF NOT .NEG tinpfb DO-SINGLE rerror(); %inpfbk is not in relabelling%

3bn2 xintr9();

3bn3 intr7() END.

3bo (intr10) %Interrupt 10 is generated by the input fork, when, during outputting, a rubout is typed. The routine causes the main program to go to the previous state. The same function is taken care of for todas in the routine rubout%

3bp PROCEDURE;

3bp1 .AR ← BRS \$78(intmsk); %re-arm interrupts%

3bp2 EXECUTE mnctrl;

3bp3 ursnls() END.

3bq (hash) %An a-string address is required in the A. The hash code for that string is RETURNd in the A. RERROR is called IF the hash is zero. Calling HASH with an empty A-string will result in this CALL to RERROR.%

3br PROCEDURE(hash1);

3br1 %a-string%

3br2 BUMP hash1;

3br3 hash2 ← /hash1/3; %length of string in words (not including any characters in the last word) in the a, number of characters in the last word in the B%%

3br4 .AR ← 0;

3br5 (hash1):

3br6 BUMP hash1;

3br7 .AR ← RCY \$3(,.AR)+/hash1/;

3br8 IF NOT .DECNEG hash2 SINGLE GOTO hash1;

3br9 IF .AR = 0 THEN !err \$6;

3br10 RETURN END.

3bs (getwka) %The purpose of this routine is to get a working copy of the file now being used. It must avoid changing the rebleling, or the core blocks. It works as follows:

3bs1 It opens the random file.

3bs2 If the IOCTL overlay is not in, the relabeling is saved and that overlay is brought in.

3bs3 The message "working copy created" is displayed.

3bs4 The routine COPFIL is called.

3bs5 The message is removed and the relabeling restored. The current file is closed.%

3bt PROCEDURE;

3bt1 %get working random file%

3bt2 declare dmgwka=(19,19,"working copy created");

3bt3 IF NOT rfwfg DO-SINGLE rerror();

3bt4 %write-ok flag not set%

3bt5 IF NOT rffn DO-SINGLE oprfff();

3bt6 IF NOT .NEG tioc1 THEN BEGIN %ioc1 not in relabelling%

3bt6a gtwk1 ← cortb/(inpfbk .A 7);

3bt6b gtwk2 ← cortb/(disbuf .A 7);

3bt6c gtwk4 ← cortb/(ioc1 .A 7);

3bt6d gtwk3 ← \$0;

3bt6e EXECUTE ioc1 END

3bt7 ELSE gtwk3 ← 0;

3bt8 dismes(\$dmgwka,1);

3bt9 copfil(curfn,rffn);

3bt10 dismes(,0);

3bt11 IF gtwk3 THEN BEGIN

3bt11a ovldn(,,gtwk1);

3bt11b ovldn(,,gtwk2);

3bt11c ovldn(,,gtwk4);

3bt11d BUMP ovrelf;

3bt11e EXECUTE nothg;


```

3bt11f $0 ← gtwk3 END;
3bt12 BRS $20(curfn);
3bt13 curfn←0;
3bt14 RETURN END.

```

3bu (opnrff) %This routine opens the working random file (/Oxx). The file number is contained in RFFN.%

3bv PROCEDURE;

```

3bv1 BUMP opnfg;
3bv2 (opnrfl):
3bv3 IF not .skip BRS $19(52000000b,0,rfsn) DO-SINGLE GOTO opnrfl;
3bv4 rffn ← .AR;
3bv5 opnfg ← 0;
3bv6 RETURN;
3bv7 (opnrfl):
3bv8 IF .XR = -3 THEN intr5();
3bv9 % drum space exhausted%
3bv10 rerror(); %call rerror in other cases%
3bv11 GOTO opnrfl END.

```

3bw (brs66x) %This routine is called with a random file number in the A. It deletes the contents of the file, and set the length to a very large number.%

3bx PROCEDURE;

```

3bx1 %delete contents of file%
3bx2 .XR ← BRS $66(.AR);
3bx3 RETURN(BRS $ 144(1000000B,10002B,.XR) ) END.

```

3by (lodrsv) %This routine is similar to LODSDB. It loads a ring block into core. The PSID is provided in the A, and the RSV block number is RETURNED in the A, along wth the address of the PSID

element in the X.

3by1 (ring element) format of ring element:

3by1a Word 1:

3by1a1 [0:1] = statement name flag

3by1a2 [1:1] = content analyser test flag

3by1a3 [2:1] = on if statement has passed CA test

3by1a4 [9:15] = psdb

3by1b word 2:

3by1b1 [0:11] = psid of sub statement

3by1b2 [11:1] = Head flag

3by1b3 [12:1] = Tail flag

3by1b4 [13:11] = psid of successor

3by1c word 3:

3by1c1 Hash code for name (0=no name)

3by1d word 4:

3by1d1 pvdb if there is a vector picture with statemnt,
othrwise 0

3byte %

3bz PROCEDURE;

3bz1 %psid in a, RETURNS block NOT=, core address%

3bz2 ldrsv1 ← RSH \$8(.AR + rsvhdr,0);%ring block number into
ldrsv1%

3bz3 ldrsv2 ← LSH \$10(0); %word address of first word of ring
element for this psid%

3bz4 IF rsvst1 ≤ ldrsv1 DO-SINGLE error(); %illegal ring block%

3bz5 IF rsvst/ldrsv1 = 0 THEN error(); %ring block not
allocated%

3bz6 IF rfb/.AR/ ≤ 0 THEN BEGIN %not in core..load it%

3bz6a lodrfb(.XR,3);

```

3bz6b .AR ← rfs/rsvst(ldrsvl)) end;
3bz7 .XR ← .AR + ldrsv2; %core address of ring element%
3bz8 RETURN(ldrsvl) END.

```

3c@ (lknabh) %This routine will look up a name, given the hash code in the A. It looks it up rather inelegantly, by scanning all ring blocks for a PSID element with that hash code in it. It skips on success, with the PSID in the A, and core address of the PSID element in the X.%

```

3ca PROCEDURE (lknmh1);
  3ca1 %hash code, return psid,,address with skip%
  3ca2 lknmh2 ← 0;
  3ca3 WHILE rsvst1 >= lknmh2 DO
    3ca3a IF rsvst(lknmh2) NOT= 0 THEN BEGIN
      3ca3a1 lknmh5 ← LSH $8(lknmh2,0);
      3ca3a2 %first psid in this block%
      3ca3a3 lodrsv(lknmh5); %get ring block into core%
      3ca3a4 lknmh3 ← .XR + 2; %address of word in this ring
        element containing hash%
      3ca3a5 lknmh4 ← lknmh5 + 254; %psid of last ring element in
        this ring block%
      3ca3a6 LOOP BEGIN
        3ca3a6a IF (lknmh3) = lknmh1 THEN
          3ca3a6a1 %the hashes are equal..return with a skip%
          3ca3a6a2 SKIP RETURN(lknmh5,,lknmh3-2);
        3ca3a6b BUMP lknmh5; %next psid%
        3ca3a6c lknmh3 ++ 4; %address of next ring element (in
          this block)%
        3ca3a6d IF lknmh5 >= lknmh4 DO-SINGLE EXIT; %get next
          ring block%
        3ca3a6e NULL END;
    END;

```

3ca3a7 NULL END;

3ca3b BUMP lknmh2 END;

3ca4 RETURN END.

3cb (asrbuf) %This routine moves an a-string to a buffer. It requires the a-string address in the A, and the buffer address in the B. It RETURNS the address of the last word used in the A, and the number of print characters transmitted in the B.%

3cc PROCEDURE (asrfrm,asrtoo);

3cc1 %asrnc is number of print char tran%

3cc2 % moves a character a-string to a buffer %

3cc3 %room at buftoo has been checked. final word filled with nulls%

3cc4 % addr of last cell used RETURNed in a%

3cc5 BUMP asrfrm;

3cc6 IF [asrfrm] < 0 THEN RETURN(asrtoo-1);

3cc7 %no transfer of null strings%

3cc8 asrnw ← [asrfrm] / 3 + 1;

3cc9 asrnc ← [asrfrm] +1;

3cc10 asrrem ← .BR;

3cc11 BUMP asrfrm;

3cc12 %asrfrm now points to first word to be moved%

3cc13 asrnw ← mvbfbf(asrfrm,asrtoo,asrnw);

3cc14 %asrnw now contains addr of last cell where data was moved too%

3cc15 .AR ← [asrnw];

3cc16 EXECUTE chrsh[asrrem] ;

3cc17 .BR ← 37677400B ;

3cc18 % nul nul blank %

3cc19 EXECUTE chrsh[.XR] ;

```
3cc20 [asrnw] ← .AR;
```

```
3cc21 RETURN (asrnw,asrnc) END.
```

3cd (mvbfbf) %This routine moves a buffer of words. On entry, the A indicates how many words, the B the address of the buffer to be moved to, and X the address of the buffer to be moved from. It RETURNS the address of the last word moved into in the A.%

```
3ce PROCEDURE (mvfrm,mvtoo,mvnw);
```

```
3ce1 % too RETURNed in the a %
```

```
3ce2 mvfrm,mvtoo ←+ mvnw .V 20000000B ;
```

```
3ce3 .XR ← -mvnw;
```

```
3ce4 [mvtoo] ← [mvfrm] ;
```

```
3ce5 BRX /.-2/;
```

```
3ce6 RETURN ((mvtoo .A 37777B) -1) END.
```

3cf (clrdpy) %This routine clears the display screen starting at the line number given in the A. %

```
3cg PROCEDURE (clrd1);
```

```
3cg1 %clears screen starting at line clrd1. text left intact%
```

```
3cg2 WHILE clrd1 < cdnlin DO BEGIN
```

```
3cg2a clrlin(clrd1);
```

```
3cg2b BUMP clrd1 END;
```

```
3cg3 dllau9+dllaux;
```

```
3cg4 dllaux←0;
```

```
3cg5 RETURN END.
```

3ch (clrlin) %This routine is used to clear a line from the display. The line number is provided in the A.%

```
3ci PROCEDURE (clrl1);
```

```
3ci1 .XR ← LSH $1(,0);
```

```
3ci2 dllbuf[.XR] ← 1;
```

```

3ci3 RETURN END.

3cj (resdpy) %This does a good deal of tricky stuff with the disply
but its hard to tell just what from the code.%

3ck PROCEDURE;

    3ck1 %resets display from the dlrt. %
    3ck2 dllaux←dllau9;
    3ck3 dllbuf/dismh/ ← 1;
    3ck4 dllnam ← $dlpnam;
    3ck5 cdlitf ← 1;
    3ck6 IF cdclin ≤ 0 DO-SINGLE RETURN;
    3ck7 resd2 ← -cdclin -1;
    3ck8 .XR ← $dlrt;
    3ck9 resd1 ← $dllbuf;
    3ck10 WHILE resd2 DO BEGIN
        3ck10a [resd1] ← $3[.XR];
        3ck10b EAX cedlrx[.XR];
        3ck10c BUMP resd2;
        3ck10d resd1 ←+2 END;
    3ck11 RETURN END.

3cl (clrall) %I think this wipes out the whole disply but it would be
nice to know just what it does. It has an argument in the A.%

3cm PROCEDURE (clral);

    3cm1 clra2 ← $dlrt + dlrtx*clral;
    3cm2 WHILE clral < cdnlin DO BEGIN
        3cm2a clrlin(clral);
        3cm2b [clra2] ← 0;
        3cm2c $1[clra2] ← 0;

```

```

3cm2d $3[.XR] ← .AR;
3cm2e BUMP clra1;
3cm2f clra2 ←+ dlrtx END;
3cm3 RETURN END.

3cn (gd1set) % ???? %
3co PROCEDURE;
3co1 %line number in x register -- gives correct dlrt entry%
3co2 LSH $2(,.XR);
3co3 .XR ← .BR;
3co4 .XR ← $dlrt[.XR];
3co5 RETURN END.

3cp (gd1lc1) % ???? %
3cq PROCEDURE;
3cq1 LRSB $17($0[.XR]);
3cq2 RETURN END.

3cr (gd1psi) % ???? %
3cs PROCEDURE;
3cs1 LRSB $13($1[.XR]);
3cs2 RETURN END.

3ct (gd1pwd) % ???? %
3cu PROCEDURE;
3cu1 RETURN($2[.XR]) END.

3cv (gd1adr) % ???? %
3cw PROCEDURE;

```

```

3cw1 RETURN($3[.XR] .A 37777B) END.

3cx (gd1wds) % ???? %
3cy PROCEDURE;
    3cy1 LRSB $14($3[.XR]);
    3cy2 RETURN END.

3cz (gd1dot) % ???? %
3d@ PROCEDURE;
    3d@1 RSB $9($0[.XR]) .A 1;
    3d@2 RETURN END.

3da (gd1nul) % ???? %
3db PROCEDURE;
    3db1 RSB $7($0[.XR]) .A 1;
    3db2 RETURN END.

3dc (gd1spc) % ?????%
3dd PROCEDURE;
    3dd1 RSB $10($0[.XR]) .A 1;
    3dd2 RETURN END.

3de (gd111n) % ??? %
3df PROCEDURE;
    3df1 RSB $11($0[.XR]) .A 1;
    3df2 RETURN END.

3dg (puttab) % ??? %
3dh PROCEDURE (putta1,putta2,putta3);
    3dh1 %1 - address where the 3 words are to go%

```



```

3dh2 %2 - col NOT= where tab goes%
3dh3 %line NOT= where tab goes%
3dh4 .XR ← putta3*dirtx + $dirt;
3dh5 [/puttal] ← ( $2[/XR] .A 77776000B ) .V (putta2*cdhinc);
3dh6 BUMP puttal;
3dh7 [/puttal] ← 32277577B;
3dh8 %tab nul nul%
3dh9 RETURN(puttal) %RETURNS last addr used%
3dh10 END.

```

3di (fndtab) % This procedure returns the column position of the next tab, given a column position in the A %

```
3dj PROCEDURE;
```

```

3dj1 %col number of tab ch is in the a reg%
3dj2 .XR ← 0;
3dj3 WHILE .AR > tablst[/XR] DO .XR←.XR+1 ;
3dj4 RETURN(tablst[/XR]) END.

```

3dk (mvdwn) %This moves a buffer around somehow.%

```
3dl PROCEDURE(mvdsn1,mvdsn2,mvdsn3);
```

```

3dl1 %moves mvdsn2 words ENDing at address mvdsn1 down (in core)
mvdsn3 cells%
3dl2 mvdsn4 ← mvdsn1 .V 20000000B;
3dl3 mvdsn5 ← 1;
3dl4 WHILE mvdsn5 <= mvdsn2 DO BEGIN
    3dl4a [/mvdsn4] ← [/mvdsn1];
    3dl4b BUMP mvdsn5;
    3dl4c mvdsn4,mvdsn1 ←+ -1 END;
3dl5 RETURN END.

```

3dm (frzrfb) %This routine is called for all freezing and thawing of random file blocks. The random file block number is provided in the A. The B contains a 1 to freeze the block, and a -1 to thaw it. RERROR is called if that block is not in core. Anyone who freezes a block must be sure it is thawed - and only once. At each state definition, IF any blocks are frozen (or over thawed) RERROR is called.%

3dn PROCEDURE(frzrf2,frzrf3);

3dn1 %freeze random file block %

3dn2 %rfb index, increment (1=freeze,-1=release%

3dn3 frzrfl ← 0;

3dn4 WHILE frzrfl ≤ rficbx DO IF rfifcb/frzrfl = frzrf2 THEN BEGIN

3dn4a frzcpt/frzrfl/ ←+ frzrf3;

3dn4b RETURN END

3dn5 ELSE BUMP frzrfl;

3dn6 rerror() END.

3do (rellit) %This routine is called to release the literal register. It does nothing IF the literal register is not contained in a core block, otherwise it releases (thaws) that block.%

3dp PROCEDURE;

3dp1 IF NOT .NEG frzcpt/litcb/ AND litrf THEN

3dpla frzcpt/.XR/ ← -1;

3dp2 litlc,litrf ← 0;

3dp3 RETURN END.

3dq (newabs) % This routine is used by CDSPLY and VCTEDT to calculate a new absolute position given an absolute position in the a-register and a displacement (vector) in the b-register.%

3dr PROC(,newabl);

3dr1 newab2+RSH \$12(.AR .A 17771777B);

3dr2 newab3+LSH \$12(0,.BR);

```

3dr3 newab2←←RSH $12(newab1 .A 37773777B);
3dr4 newab3←←LSH $12(0,.BR);
3dr5 .AR←LSH $12(newab2,0) .V newab3;
3dr6 RETURN(.AR .A 37773777B) END.

```

4 %pops in the UTILTY overlay %

4a (bt) %If FLAG is true, a branch to the address of the pop takes place. The A and X are unchanged.%

4b POP 101B PROC; %branch true%

4b1 .BR←-1;

4b2 IF .BR .CB flag DO-SINGLE GOTO [\$0];

4b3 RETURN END.

4c (bf) %If FLAG is false, a branch to the address of the pop takes place.%

4d POP 102B PROC; %branch false%

4d1 .BR←-1;

4d2 IF .BR .CB flag DO-SINGLE RETURN;

4d3 GOTO [\$0] END.

4e (lai) %Load A immediate. SMAREG is set to the contents of the address of the pop.%

4f POP 105B PROC;

4f1 % load a immediate %

4f2 .XR←0;

4f3 EAX [\$0];

4f4 smareg←.XR;

4f5 RETURN END.

4g (lbi) %Load B Immediate. SMBREG is set to the address portion of the pop.%

4h POP 106B PROC;

4h1 % load b immediate %

4h2 .XR←0;

4h3 EAX [extract_itex]0];

4h4 smbreg←.XR;

4h5 RETURN END.

4i %The following POPs are argument pops. When used, the addresses are argument numbers, and the tag bit is generally on. The pops load RTNCEL in the X, so that the argument is referenced with respect to that stack mark.%

4i1 (sap) %Store Argument Pointer. The two word current text pointer (in SWORK) is stored indirectly through the pop address.%

4i2 POP 115B PROC; %store argument pointer %

4i2a .XR←rtncel;

4i2b lldp swork; lstp [extract_itex]0] END.

4i3 (laa) %Load Argument (A only). The A is loaded indirectly through the address of the pop.%

4i4 POP 116B PROC; %load argument, a only %

4i4a .XR ← rtncel;

4i4b RETURN([extract_itex]0]) END.

4i5 (saa) %Store Argument (A only). The contents of the A are stored indirectly through the address.%

4i6 POP 117B PROC; %store argument, a only%

4i6a .XR←rtncel;

4i6b [extract_itex]0]←.AR;

4i6c RETURN END.

4i7 (lap) % Load Argument Pointer. The A-B register is loaded indirectly through the address using the LDP syspop.%

4i8 POP 120B PROC; % load argument pointer %

4i8a .XR←rtncel;

4i8b lldp [extract_itex]0];

4i8c RETURN END.
.

4i9 (lsa) % Load Subroutine Argument. The subroutine argument addressed by the pop is pushed onto the stack as an argument for another subroutine being called. This is a little tricky, since the mark has been changed by the MKS pop.%

4i10 POP 121B PROC; %load subroutine argument %

4i10a .XR ← rtncel;

4i10b .XR ←[/extract_itex]37777B[.XR];

4i10c RETURN (push(,[extract_itex]0)) END.
.

4j (sov) % Select an Overlay. This pop is used before using a SBC, OVC, or BRV pop, to select which overlay to load before branching. The overlay name is given in the address of the pop. (If no overlay is selected before calling one of the branch pops, no change in the overlay configuration will take place).%

4k POP 137B PROC; % select overlay %

4k1 EAX [extract_itex]0](,,0);

4k2 sov1←.XR;

4k3 RETURN END.

4l (sbc) % SuBroutine Call. The RETURN location (pointed to by RTNCEL) is set to the overlay address of the SBC instruction. Then the selected overlay (IF any) is called in, and a branch to the subroutine address is executed.%

4m POP 107B PROC; % subroutine CALL %

4m1 ovladr([extract_itex]0);

4m2 GOTO ovcsta END.

4n (ovc) % Overlay Call. This works the same as SBC, except that the RETURN location that is stored, is the previous state. This implements the minus calls in the input-feedback language.%

4o POP 140B PROC; %overlay call- no RETURN (gps) %

4o1 .AR ← state-1;

4o2 (ovcsta):

4o3 [rtncel] ← .AR END.

4p (brv) % Branch to overlay. This pop calls in the selected overlay (IF any), and branches to the address given in the address of the pop.%

4q POP 143B PROC; % branch to overlay %

4q1 (brv1):

4q2 EAX [\$0](,,0);

4q3 ovcl←.XR;

4q4 EXECUTE [sov1];

4q5 sov1←\$nopcod;

4q6 GOTO [ovcl] END.

4r (lass) % Load argument String on Stack. The address of this pop is taken as two 4 bit numbers, say n1 (bits 16-19) and n2 (bits 20-23). Then the addresses of pointers P(n1) through P(n2) are loaded on the stack in that order. This implements the [P1-8] construct, and saves a good deal of core in the TXTEDT.overlay.%

4s POP 145B PROCEDURE; %load argument string on stack%

4s1 EAX [\$0](,,0);

4s2 lass1 ← RSH \$4(.XR,0);

4s3 lass2 ← LSH \$5(0,.BR) + \$p0;

4s4 lass1 ← LSH \$1(lass1,0);

4s5 FOR lass3 FROM \$p0 + lass1 INC 2 TO lass2 DO

4s5a push(,lass3);

4s6 RETURN END.

4t (las) %Load Argument Stack. The word addressed by the pop is pushed onto the general stack. Subroutine arguments are loaded with

this pop.%

4u POP 114B PROC; %load argument on stack %

4u1 RETURN (push(,[\$0])) END.

4v (anypop) %Zero address POP's%

4v1 POP 177B PROC;

4v2 EXTERNAL sbr,txt,cec,cic,gc,gps, kset,mks,atg,clr % %

4v3 GOTO [\$0];

4v4 (sbr): %Subroutine Return. Control RETURNS to the location and overlay in which the last CALL (SBC) took place.%

4v5 %subroutine RETURN %

4v5a rtncel←\$37777B[rtncel];

4v5b .XR←.XR-2;

4v5c stack←.XR;

4v5d IF NOT .NEG \$2[.XR] DO-SINGLE BRR \$2[.XR];

4v5e ovlgo(\$2[.XR]);

4v6 (txt): %The contents of the A and B registers is passed on to the create display routine. This may involve an overlay change to get CDSPLY (now in page 5) and DISBUF (now in page 7) in, but control will RETURN to the overlay in which the ANYPOP TXT took place. There is a danger here: note that ONLY the overlay in which the ANYPOP TXT was located (and overlays above it in the overlay structure) will be relabelled back in. Thus IF an overlay in page 5 or 7 is brought in, THEN a TXT is done from a page other than 5 or 7, the overlay may not still be in page 5 or 7. %

4v7 % create a new display %

4v7a !stp smtemp;

4v7b IF todas DO-SINGLE RETURN; %todas%

4v7c IF paslfg DO-SINGLE RETURN;

4v7d txtloc←ovladr(\$0);

4v7e EXECUTE rfbp2;

```

4v7f EXECUTE cdsply;
4v7g IF NOT cdtref THEN BEGIN
    4v7g1 lldp smtemp;
    4v7g2 credis() END
4v7h ELSE BEGIN
    4v7h1 lldp smtemp;
    4v7h2 cdtree() END;
4v7i ovlgo(txtloc);
4v8 (cec): %Copy Entity Character. The entity character (in
SMEC) is copied into SMBREG.%
4v9 % copy entity character *%
    4v9a smbreg+smec;
    4v9b RETURN;
4v10 (cic): %Copy Input Character. The input character in SMCREG
in copied into SMBREG.%
4v11 % copy input character *%
    4v11a smbreg+smcreg;
    4v11b RETURN;
4v12 (GC): %Get Character. The AB register contains a T-pointer.
The character pointed to by the T-pointer is copied into SMBREG.%
4v13 % get character that ab points to (into b) %
    4v13a lstp swork;
    4v13b % store pointer in work area %
    4v13c smbreg+fechcl(,, $swork);
    4v13d RETURN;
4v14 (kset): %This pop sets FLAG to true.%
4v15 % set flag to true (54) %
    4v15a flag+1;

```


lv15b RETURN;

lv16 (gps): %Go to Previous State. A branch to the previous state definition takes place, after the overlay in which the state was defined is brought into core. The define state pop at that location is executed again.%

lv17 % go to previous state %

lv17a .XR←state;

lv17b IF NOT .NEG state DO-SINGLE GOTO \$0[.XR];

lv17c .XR←.XR-1;

lv17d ovlgo(.XR);

lv18 (mks): %Mark Stack. The mark for the general stack (RTNCEL) is pushed onto the stack, followed by a zero (for the RETURN location). RTNCEL is THEN set to point to the current top of the stack (the future RETURN location). This pop is executed before a subroutine call, and before loading arguments.%

lv19 % mark stack for sub CALL *%

lv19a push(,rtncel);

lv19b push(,0);

lv19c rtncel←stack;

lv19d RETURN END.

lv20 (clr): %Clear a state machine register%

lv20a [regary[.XR]]←regcv[smareg];

lv20b RETURN;

lv21 (atg): %Set abort target. This does not change the current state, but does set a "pseudo-state" as an abort location.%

lv21a abtgt ← ovladr(\$0);

lv21b GOTO zap; %set abort target%

lv22 %END of anypop%

5 %stacks, registers, strings, buffers%

5a %literal register%

5a1 (getlit) PROCEDURE;

5a1a litrf ← 0;

5a1b litlc ← \$stnxcn;

5a1c litlcl ← \$stnl;

5a1d stnl ← -1;

5a1e RETURN END.

5b %stacks%

5c %work with a-strings%

5d %move buffers%

5e %messages, errors, and aborts%

5e1 (dismes) %This routine is called to display a message on the screen. It blanks out the text portion of the screen and puts the message across the center. The address of an A-string is usually provided in the A. The B contains an integer which determines the action taken:

5e1a A 2 in the B causes the message to be displayed for 1 second, THEN the display is restored and the routine RETURNS.

5e1b If the B contains a 1, the message will be displayed, and the routine will RETURN (with the message still on the screen).

5e1c A zero in the B will remove any message on the screen and restore the display. An A-string need not be given in the A in this case. This is done by simply calling RESDPY.%

5e2 PROCEDURE(dism1,dism2);

5e2a IF todas THEN BEGIN

5e2a1 crlf(); IF dism2 THEN !typeas dism1; RETURN END;

5e2b IF NOT .NEG tdisbu THEN BEGIN

5e2b1 dism5 ← cortb(7); EXECUTE disbuf END

5e2c ELSE dism5 ← 0;

5e2d IF dism2 THEN BEGIN

5e2d1 clrdpy(0);

```

5e2d2 dism3 ← asrbuf(dism1,cdlit) - cdlit;
5e2d3 d11buf/dism4/ ← cdlit .V LSH $14(IF > .GT 23 THEN 24
5e2d3a ELSE dism3+1,0);
5e2d4 IF dism2 = 2 THEN BEGIN
5e2d4a BRS $81(1000); resdpy()END END
5e2e ELSE resdpy();
5e2f IF dism5 THEN BEGIN
5e2f1 ovldn(,,dism5); EXECUTE nothg END;
5e2g RETURN END.

5e3 (typeas) POP 174B PROCEDURE;
5e3a %Type a-string on tty%
5e3b %address of word containing address of a-string in address
field of instruction%
5e3c .BR+$(1/($0//)+1; %a-string address in x, length in B%
5e3d BRS $34(.XR+2,,1);
5e3e RETURN END.

5e4 (crlf) PROCEDURE; %todas%
5e4a TODCO 155B; TODCO 152B; TODCO 146B;
5e4b carpos←0;
5e4c RETURN END.

5f %gets from ring--fehc from sdb%
5f1 (getssf) % return the successor and sub-statement psids of
the ring element of the psid passed in tthe A%
5f2 PROCEDURE;
5f2a lodrsv(.AR); %load ring block and return address of ring
element in the X%
5f2b RETURN( $1/.XR/) %second word in ring element% END.

```

5g %random files%

5g1 %lod-sto-fch sdb, rsv, vdb, and qb%

5g1a (storsv) PROCEDURE; %move back to strmp when cac uses bit
table%

5g1a1 lodrsv(.AR);%load ring block and return address of
ring element in the X%

5g1a2 strsv1 ← .XR;

5g1a3 strsv2 ← .AR;

5g1a4 [rfbs/(rsvst[strsv2])]] ← 0;

5g1a5 RETURN(strsv2,,strsv1) END.

5g1b (stovdb) PROC(,stvdb1); % psid in a, pvdb in b %

5g1b1 lodrsv(.AR); %load ring block%

5g1b2 \$3[.XR]+(\$3[.XR] .A 77700000B) .V stvdb1;

5g1b3 RETURN END.

5g1c (lodqb) PROC;

5g1c1 declare quhmax=50;

5g1c2 IF qbst = 0 THEN BEGIN

5g1c2a qbst←newrfb();

5g1c2b lodrfb(.AR,4);

5g1c2c .XR←rfbs[qbst];

5g1c2d \$11b[.XR],\$1[.XR]+quhmax+8 END

5g1c3 ELSE IF rfbs[qbst] < 0 THEN lodrfb(qbst,4);

5g1c4 RETURN (,,rfbs[qbst] + 10B) END.

5h %overlays%

5h1 %overlay pops--arguments and calls%

5h2 %ovealay PROCEDURES%

```

5h2a (ovld) PROCEDURE(ovld2,,ovld1); % pos,,new ovl NOT=%
    5h2a1 XXA; XMA cortb[.XR]; % old NOT= in A, new in core %
    5h2a2 .XR ← .AR;
    5h2a3 ovrlt[.XR] ← ovrlt[.XR] .A 37777777B; % mark old
    overlay out %
    5h2a4 .AR ← ovrlt/ovld1/ .V 40000000B; % mark new overlay
    in %
    5h2a5 XMA ovrlt[.XR]; ovld3 ← .AR; %PMT number%
    5h2a6 .XR ← ovld2*6; %number of bits to shift% ovld2 ← 0;
    5h2a7 .AR ← LCY $6[.XR](rlr2,rlr1) .A 77777700B .V ovld3;
    %shift and or new overlay (PMT) number into relabelling%
    5h2a8 .AR ← RCY $6[.XR](); %shift relabelling registers back
    to normal position%
    5h2a9 IF .AR NOT= rlr2 DO-SINGLE BUMP ovld2; %relabelling
    changed% XAB;
    5h2a10 IF .AR NOT= rlr1 DO-SINGLE BUMP ovld2; %relabelling
    changed%
    5h2a11 rlr1 ← .AR; rlr2 ← .BR; %set up new
    relabelling%RETURN(ovld2,,ovld1) END.

5h2b (ovldn) PROCEDURE; % overlay number in X %
    5h2b1 RETURN(ovld(ovltb[.XR] .A 7,,.XR)) END.

```

5i %intr routines%

```

5i1 (rubout) %This is the routine for handling interrupt 1, rubout
It saves all of the registers, checks to see IF this is the second
consecutive rubout, and terminates IF it is, clears all of the
input and output buffers, and THEN dispatches control to the
proper place, depending on what todas is doing at the particular
time. Be careful, insofar as all action on rubouts is not taken
here, butt rather in odd places throughout the system, eg
inptc
%

```

```

5i2 PROCEDURE;

```

```

5i2a rubou1 ← .AR; rubou2 ← .BR; rubou3 ← .XR; %save the
registers%

```

```

5i2b clrbuf(1);%clear input and output buffers%
5i2c IF inptrf THEN intr5();%terminate IF second consecutive
rub out%
5i2d .AR ← BRS $78(intmsk); %re-arm interrupts%
5i2e IF outffg DO-SINGLE intr10(); %re-initialise IF file being
output%
5i2f IF opnfg THEN BEGIN %file open in progress%
    5i2f1 opnfg ← 0; GOTO gps END;
5i2g IF rubabt DO-SINGLE GOTO gps;
5i2h [inptbi]←1000137B; BUMP inptbi;%pass rubout to input
routine%
5i2i IF inptfg THEN %input in progress.. CALL interrupt 7 to
simulate% intr7();
5i2j LOAD(/rubout/,rubou2,rubou3);
5i2k XMA ruboul;
5i2l GOTO [ruboul]
5i2m END.

```

5i3 (clrbuf) %clear all of the todas input and output buffers
the value in the a causes the output buffer to be cleared if is
non-zero . nothing is RETURNed%

5i4 PROCEDURE;

```

5i4a IF .AR THEN BRS $29(,-1); % clear output buffer%
5i4b BRS $11(,-1); %clear tty buffer%
5i4c buffct ← 0; %clear the todas buffer%
5i4d inptbi,inptbo ← $inptbf;%clear nls buffer%
5i4e RETURN END.

```

5j %names and pointers%

5k %basic pops and anypop%

5l %PROCEDURES for use by CDSPLY and VCTEDT%

511 %clear and restore the display%

512 %get and put bits in the dlrt%

512a (gd11c1) PROCEDURE;

512a1 .AR ← \$0[.XR] .A 177B;

512a2 RETURN END.

512b (gd1sch) PROCEDURE;

512b1 .AR ← \$1[.XR] .A 3777B;

512b2 RETURN END.

513 %tabs%

514 (cdf1p) PROC;

514a .AR←cdclin;

514b IF .AR > 0 THEN .AR←.AR-1;

514c .XR←\$dlrt+LSH \$2(,0);

514d .BR←(\$3[.XR] .A 37777B)+ RSH \$14(\$3[.XR]);

514e RETURN(\$2[.XR],.BR) END.

6 finish

:INPFBK, 12/05/69 0952:11 WSD ; INPUT FEEDBACK .SCR=1; .PLO=1; .MCH=75;
.RTJ=0; .DSN=1; .LSP=0; .MIN=70; .INS=3; ["FOR"];

Description of the INPFBK overlay

The INPFBK overlay is one core page of MOL PROCEDURES that are used by other overlays that DO user terminal input and display output. The display buffer page is always relabeled in when INPFBK is in, since many display feedback routines are in INPFBK. Also, tables in the display buffer page are used to PROCESS bug marks. The tables for decoding the binary handset strokes are also in INPFBK. The INPFBK page is read-only and shared.

*display output
view specs, bug etc*


```
%HED="INPFBK, inpfbk overlay %
STACK stackp, rbr, rskip, 161B;
POP
```

```
sbc = 107B,
las = 114B,
abort = 136B,
sov = 137B,
typeas = 172B,
echo = 174B,
err = 175B,
any = 177B;
```

```
DECLARE origin(17777B);
```

```
DECLARE inpkcn = (
35660536B,220501B,421102B,
621503B,1022104B,1222505B,
1423106B,1623507B,2024110B,
2224511B,10025112B,2625513B,
3226114B,2426515B,3627116B,
17427517B,4030120B,4230521B,
4431122B,4631523B,5032124B,
5232525B,5433126B,5633527B,
6034130B,6234531B,7235132B,
16616014B,17217016B,17615033B,
27236037B,33264400B);
```

```
DECLARE inpkat = (0,
1000101B,1000102B,1000103B,
1000104B,1000105B,4000000B,
1000107B,1000110B,1000111B,
1000112B,1000113B,1000114B,
1000115B,1000116B,1000117B,
1000120B,1000121B,1000122B,
1000123B,1000124B,1000125B,
1000126B,1000127B,1000130B,
1000131B,1000132B,1000000B,
1000000B,1000000B,100B,
1000000B);
```

```
%input characters%
```

```
(readm) %This PROCEDURE RETURNS in the A and B the mouse
coordinates from the last command accept character or pointer. In
the case of a bug mark or CA key, the 24 bit coordinates are RETURNed
in the B, with a -1 in the A. In the case of a pointer, the
```

```
Ⓟ T-pointer is RETURNed in the A and B.%
PROCEDURE;
```

```
RETURN(inptr,inptmr) END.
```

```
(inpt0) %This routine RETURNS a character from the keyboard,
handset, or mouse. If no characters are in the system's input
buffer, the PROCEDURE dismisses (the main fork) on the highest
priority queue until there is a character waiting. This is done with
BRS's -10 and 72 which are EXEC only. The format of the character
RETURNed is the same as from the system. The clock reading is saved
in INPTCL, the mouse coordinates are saved in INPTCM, and the
character is RETURNed in the A.%
PROCEDURE;
```

why?

%high priority- lowest level input routine%

```
(inpt01):  
WSI ttyno;  
IF .AR CB 100000B DO-SINGLE GOTO inpt0W; %no character read%  
inpt0c ← .AR .A 77077777B; VS THEN-ELSE const. (?)  
inpt0l ← .ZK;  
inpt0f ← .Rb;  
EXTERNAL(inpt0c);  
(inpt0W):  
BRS @-10(.,ttyno);  
BRS $72(); %wait for workstation input%  
GOTO inpt01 END.
```

(inptc) %INPTC is the single character input routine. It RETURNS characters from the input devices in the A, in the following format: A command accept button going down is a -1, coming up is a -2, and all other characters are in ASCII. This routine takes care of the binary handset. Button strokes or special handset chords are PROCessed, but INPTC does not RETURN until a real character for the main part of NLS is read. INPTC functions approximately in this way:

is the
button
involved
transmitted
?

First a character is obtained.

If there is a character in the input buffer, it is taken.

⑦ Otherwise, IF Pass1 is running, THEN the routine dismisses itself (using BRS's -10 and 72), after setting a flag for the input fork to create pseudo-interrupt 207.

Or IF the input fork is active, INPTC waits (BRS 45) and goes back to its BEGINning.

And finally, it calls INPTO to read the character IF all the above fail.

Having obtained a character, it checks to see IF it is a rubout and PROCesses it IF it is. Or IF it is a command accept code of some kind, a flag is set to indicate that and the coordinates are saved.

Then the character is PROCessed through a switch, depending on the source of the character (mouse, keyboard, handset).%

PROCEDURE;

%RETURNS a character from input devices%

```
DECLARE inptyp=(inptb,inptc,inptk,inptcx,  
inptog,inptp,inptx,inptcx);
```

```
DECLARE inpkst=(  
inptb,inplts,inptcx,inptcx,inpcd);
```

```
EXTERNAL inptcx;
```

```
(inptcx):
```

```
IF inptoi ≠ inptbo THEN BEGIN
```

```
inptcl ← /inptbo/;
```

```
BUMP inptbo;
```

```
IF inptbo ≥ inptbe THEN inptbo ← $inptbf END
```

```
ELSE IF paslfg THEN BEGIN
```

```
BUMP inptfg;
```

```
BRS $72(,100200B,0) %suspend until interrupt%
```

```
END
```

```
ELSE IF inptfa THEN BEGIN %input fork is running, wait for it to  
stop%
```

```

      BRS $45(1);
      GOTO inptcx END
ELSE BEGIN
  BUMP inptfg;
  IF todas THEN TOI inptcl ELSE
  inptcl ← inpto();
  inptfg ← 0 END;
IF inptcl = 1000137B THEN BEGIN
  IF inptrf THEN BEGIN
    BUMP inptfg;
    intr5() END
  ELSE BEGIN
    BUMP inptrf;
    IF todas THEN EXECUTE todmnc ELSE EXECUTE mnctrl;
    IF paslfg DO-SINGLE ursnls();
    IF todas THEN GOTO reset ELSE GOTO cmdrst
    END END
ELSE BEGIN
  IF (inptcl = 1000144B) OR (.AR = 4000001B) THEN BEGIN
    inptmr←inptcm;
    inptpr←-1 END;
    inptrf ← 0; IF todas THEN RETURN(inptcl) END;
GOTO /inptyo/(RSH $18(inptcl,0) .A 7));
(inptb):
  BUMP inp;
  IF incse = 3 THEN GOTO inplts
  ELSE RETURN(LSH $18(0) .A 177B);
(inptk):
  BUMP inp;
  IF incse = 3 THEN GOTO [inpkst4/(LRSH $18(inpkst/(LSH
  $18(0))),0))]
  ELSE RETURN((LRSH $0[.XR]/(inpkch/(LSH $18(0) .A 37B)),, LSH
  $3(incse,0)) .A 177B);
(inptbg):
  RETURN(IF .BR CB -1 THEN -1
ELSE -2);
(inptx):
  inbttn←2;
  GOTO inptck;
(inptp):
  inbttn←1;
  % GOTO inptck; %
(inptck):
  IF .BR CB -1 THEN BEGIN
    inp←inwait←0;
    incse←incse .V inbttn END
  ELSE BEGIN
    incse←incse .A (inbttn .X 3);
    IF NOT inp THEN BEGIN
      IF incse = 0 THEN BEGIN
        IF inwait THEN RETURN(LRSH $16(inpkch))
        ELSE IF inbttn = 1 THEN RETURN(inpkch .A 177B)
        ELSE RETURN(LRSH $8(inpkch) .A 177B) END
      END
    END
  END

```

```

        ELSE BUMP inwait END END;
    GOTO inptcx;
(inpcd):
    inptc1 ← cortb5;
    inptc2 ← cortb6;
    inptc3 ← %0;
    .AR ← -1;
    !any txt;
    ovldn(,,inptc1);
    ovldn(,,inptc2);
    BUMP ovrelf;
    EXECUTE inpfbk;
    %0 ← inptc3;
    GOTO inptcx;
(inplts):
    inptc2 ← LSH $18(0);
    inptc3 ← %0;
    inptc1 ← cortb6;
    EXECUTE prmspc;
    setlt(inptc2);
    ovldn(,,inptc1);
    BUMP ovrelf;
    EXECUTE inpfbk;
    %0 ← inptc3;
    GOTO inptcx END.
(inptm) %This is the main program's input routine. It is one
logical level higher than INPTC in that it PROCesses pointers and
turns command accept buttons up-down into command accept characters
(144b). The pointer names are read into an A-string named INPPR,
and LKPTR is called to obtain a T-pointer. %
PROCEDURE;
    %main programs input routine%
    (inptml):
        IF todas THEN RETURN(tinptc());
        inptml ← inptc();
    (inptmx):
    IF NEG inptml THEN BEGIN
        IF .AR = -1 THEN BEGIN
            inppr ← 2;
            inppr1 ← -1;
            inppr2 ← 0;
            inptml ← inptc();
            WHILE inptml # -2 DO BEGIN
                IF NEG inptml THEN !abort %5;
                apchr(inptml,,sinppr);
                inptml ← inptc() END;
            IF NEG inppr1 THEN RETURN($ascca);
            lkptr(inppr2);
            !abort %5;
            inptmr ← .BR;
            inptpr ← .AR;
            inptml ← $ascca END
        ELSE IF .AR = -2 THEN GOTO inptml

```

```

        ELSE rerror() END;
RETURN(inptml) END.
(lkptr) %LKPTR takes a three character pointer name in the A and
RETURNS the T-pointer for that pointer in the A and B. IT skips IF
that pointer exists, does not skip IF it fails to find the pointer in
the table.%
PROCEDURE(lkptr1); %3 character name in .AR%
lkptr2 ← -3; WHILE (lkptr2←lkptr2+3) <= ptrtbl DO IF ptrtb[lkptr2/
= lkptr1 THEN BEGIN
    %skip RETURN%
    SKIP RETURN (ptrtb[lkptr2+1/ , ptrtb[.XR+1/]) END;
RETURN %no skip%
END.
%todas kludge for opening and loading files%
(tdiokd)PROCEDURE; %todas io kludge%
%the following is a kludge for todas...very very temporary%
EXTERNAL outckp,outfil,loadckp,loadfil;
(outckp):
    EXECUTE rfbp2;
    EXECUTE ioctl;
    frecor();
    opnckp();
    copfil(rffn,ckptfn);
    BRS #20(ckptfn);%close file%
    ckptfn ← 0;
    GOTO gps;
(outfil):
    EXECUTE rfbp2;
    EXECUTE ioctl;
    frecor();
    smcreg ← inputc();
    IF fnml > -1 THEN BEGIN % type file name and look for ca%
        !typeas $fnmxn;
        IF smcreg # $ascca DO-SINGLE GOTO outfil;
        curfn ← opnfil(1,72000000B,$fnmxn)
    END ELSE BEGIN
        (outfil):
            curfn ← opnfil(1,72000000B,0);
            cpysr($stnzn,,$fnmxn) END;
    ofdate();
    !any mks; !las $pl; !sov txtedt; !sbc setrot;
    EXECUTE ioctl;
    frecor();
    copfil(rffn,curfn);
    BRS #20(curfn);
    curfn ← 0;
    GOTO gps;
(loadckp):
    EXECUTE rfbp2;
    EXECUTE ioctl;
    frcrff();
    opnckp();
    rdhdr(ckptfn);

```

```

    copfil(ckptfn,rffn);
    BRS $20(ckptfn);
    ckptfn ← 0;
    fnroot();
    GOTO gps;
(loadfil):
    EXECUTE rfbp2;
    EXECUTE ioct1;
    savrff();
    frcrff();
    curfn ← opnfil(0,72000000B,0);
    cpysr($stnxx,, $fnmxx);
    BRS $20(rffn);
    rffn ← 0;
    rdhdr(curfn);
    GOTO gps
END.
%todas i/o routines% %todas%
(inpcuc)PROCEDURE;
    %input a character and translate it to upper case IF necessary%
    IF inputc() IN (100B,132B) THEN buff ← .AR - 40B;
    RETURN END.
(inputc)PROCEDURE;
    %read one character from tty or buffer into lastchr%
    %retrun chr in a%
    buff←lookc(); bufffg ← .BR; buffct←+-1; RETURN(buff)
    END.
(lookc)PROCEDURE;
    %look at next haracter in input buf or tty%
    %RETURN in a%
    %RETURNS a flag in b register: 1=chr literal, 0=chr normal, -1=chr
    control%
    IF buffct THEN RETURN(buff/buffct/,bufffg(buffct/));
    tinptc(); BUMP buffct; buff/buffct/←.AR;
    bufffg/.XR/←.BR;
    RETURN END.
(todco)POP 173B PROCEDURE;
    %output a character to tty%
    popret ← $0;
    IF putchr(/$0/,carpos,0) = -1 THEN % carrage RETURN time % crlf()
    ELSE BEGIN
        IF carpos+.AR > pccolx THEN crlf() ELSE
            carpos←.AR
    END;
    BRR popret END.
(putchr)PROCEDURE(putch1,putch2,putch3);
    %put a character onto tty or a-string IF .XR # 0%
    IF .AR = $asctab THEN BEGIN % tab in here %
        IF gettab(putch2+1) > pccolx THEN RETURN(-1);%tab wont fit on
        line%
        putch1←putch2←.AR-putch2; %number of spaces to emit%
        (pttab1):
            IF NOT putch3 THEN TCO 0 ELSE

```

where tested?

```

        apchr(0,,putch3);
        putch1 ←+-1;
        IF putch1 DO-SINGLE GOTO pttabl;
    RETURN(putch2)
END;
IF .AR = $asccr THEN RETURN(-1); %indicate crlf%
IF prcapf THEN BEGIN % all upper case, slash before caps%
    IF putch1 NOT IN (40B,72B) DO-SINGLE GOTO putclc;
    IF NOT putch3 THEN TCO '/' ELSE
    apchr('/',,putch3);
    .AR ← putch1; GOTO putcr2;
    (putclc): IF .AR NOT IN (100B,132B) DO-SINGLE GOTO putcsc;
    .AR ← .AR - 40B;
    GOTO putcr1 END;
    (putcsc): IF putch1 > 132B THEN BEGIN
    IF NOT putch3 THEN TCO '& ELSE
    apchr('&',putch3);
    .AR ← putch1-100B; GOTO putcr2 END;
    (putcr1):
    .BR ← .AR;
    IF NOT putch3 THEN BEGIN putch3 ← .BR; TCO putch3 END ELSE
    apchr(.BR,,putch3); RETURN(1);
    (putcr2):
    .BR ← .AR;
    IF NOT putch3 THEN BEGIN putch3 ← .BR; TCO putch3 END ELSE
    apchr(.BR,,putch3); RETURN(2)
END.
(answer) %this PROCEDURE accepts a yes or no answer from the
keyboard, and RETURNS with a 0 in the a register IF the answer was
negative, and a 1 IF it was positive%
PROCEDURE;
(ansstr):
CASE inptm() OF
    = $ascca: GOTO anspos;
    = 'Y: BEGIN (anspos):
        !echo $"Yes."; RETURN(1) END;
    = 'N: BEGIN
        !echo $"No."; RETURN(0) END;
    = $asccd: GOTO gps
ENDCASE BEGIN
    !todco '?;
    !todco 0;
    GOTO ansstr
END
END.
(rdlit)PROCEDURE(rdlit2,rdlit1);
%read a literal from the keyboard, doing all of the control
things, plus breaking on any control character identified by the
list in rdlit1%
%RETURN 0 IF error (buffer overflow), 1 IF ok%
IF .AR THEN BEGIN %ok .. a-string here%
    (rdlit1):
        inputc(); IF NOT NEG buffig DO-SINGLE GOTO litchr; %not a

```

CROSS INDEX!

5-20-66 here

```

control character%
CASE .AR OF
  = $asccd: GOTO gps;
  = $ascca: RETURN(1);
  = $ascbc: BEGIN
    bkachr(rdilit2); GOTO rdilit1 END;
  = $ascbw: BEGIN
    IF ldchr(rdilit2,$1/.AR/) = 0 THEN %space past blanks%
      WHILE ($1/rdilit2/ > -1) AND (NOT bkachr(.XR)) DO NULL;
    WHILE bkachr(rdilit2) DO NULL;
    GOTO rdilit1 END;
  = $ascbst: BEGIN $1/rdilit2/←-1; crlf(); GOTO rdilit1 END;
  = $ascct: RETURN(1) % centerdot%
ENDCASE BEGIN
  (litchr):
  IF rdilit1 AND chrpos(,buff,.AR) THEN RETURN(2);
  (rdentc): apchr(buff,rdilit2);
  IF $1/rdilit2/ = $0/.XR/ THEN RETURN(0);
  GOTO rdilit1 END
END; rerror() END.
(restor)PROCEDURE(restol);
  %put character in a back ijto the input buffer%
  %CALL rerror IF buff ovrflw%
  IF buffsz <= buffct DO-SINGLE rerror();
  BUMP buffct; buff/buffct/←restol;
  buffig/.XR/ ← .BR;
  RETURN END.
(tinptc)PROCEDURE;
  %This is the main character input routine.
  It does all of the character translation, RETURNING a 0 inthe IF
  the character was a literal, and a -1 IF it was a control
  character. The character is RETURNED in the a%
  %todas control character defitions%
  DECLARE todbc = 150B, %control h%
    todbw = 167B, %control w%
    todbst = 160B, %control q%
    todlf = 152B, %line feed%
    todca = 144B, %control d%
    todcdt = 142B, %cotrol b%
    todcd = 171B, %control x%
    todshc = '/',
    todlit = '!; % exclamation point%
  tinptl←0;%no special character%
  (tinst):
  IF NOT tinpt2 THEN inptc() ELSE BEGIN .BR←0; tinpt2←.BR END;
  tinpt3←.AR;
  %first check to see IF lower case%
  IF .AR IN (100B,132B/ DO-SINGLE
    GOTO tintrl;
  %now see IF it is a special character%
  (inpnlc):
  IF .AR NOT IN (40B,72B/ DO-SINGLE GOTO tincon;%by here, upper
  case%

```



```

        IF prcapf THEN tinpt3 ←+40B; %really lower case%
        GOTO tintrl;
(tincon):
IF .AR NOT > 132B DO-SINGLE GOTO tinspc; %by here, control
character%
    tinpt1 ← .BR ← -1;%mark as control character%
    CASE .AR OF
        = todbc: BEGIN
            (tinbsp): .BR ← '↑; tinpt3 ← $ascbc; GOTO tinecl
            END;
        = todlf: BEGIN
            (tinlff): IF echofg # 3 THEN BEGIN TCO $ascrr; carpos←0
            END; RETURN(0,0) %linefeed becomes space%
            END;
        = todca: BEGIN
            (tinca): RETURN(,-1)
            END;
        = todcd: BEGIN
            (tinca): .AR ← $ascdd; GOTO tinca
            END;
        = todbw: BEGIN
            (tinbsw): .BR ← '\; GOTO tinecl
            END;
        = todbst: BEGIN
            (tinbst): .BR ← '←; GOTO tinecl
            END;
        = todcdt: BEGIN
            (tinca): .AR ← $ascddt; GOTO tinca
            END
    ENDCASE BEGIN
        tinpt1 ← .BR ← 0; %mark as not control character%
        GOTO tinech
    END;
(tinspc):
    CASE .AR OF
        = todshc: BEGIN
            (tinshc): IF prcapf THEN
                BEGIN
                    IF (tinpt3 ← inptc()) IN (40B,72B) THEN GOTO
                    tintr2;
                    IF .AR IN (100B,132B) THEN BEGIN %lower case
                    character%
                        tinpt3 ← .AR-40B;
                        TCO tinpt3; GOTO tintr2
                    END
                END;
            %by here, next character was not alpha%
            tinpt2 ← .AR;
            tinpt3 ← '/;
            GOTO tintrl
        END;
        = todlit: BEGIN
            (tinlit): %by here, literal%

```

```

        tinpt1 ← 1;
        IF echofg ≤ 2 DO-SINGLE BUMP carpos;
        IF (tinpt3 ← inptc()) > 132B DO-SINGLE GOTO tinech;
        IF .AR = 100B DO-SINGLE GOTO tinech; %centerdot
        non-printing%
        GOTO tintr1
    END
ENDCASE GOTO tintr1;
(tintr2):
    IF echofg ≤ 2 DO-SINGLE BUMP carpos;
(tintr1):
    IF 3 ≤ echofg DO-SINGLE GOTO tintr;
    carpos←carpos+1; IF .AR > prcolx THEN crlf();
    (tintr): RETURN(tinpt3,tinpt1);
(tinech):
    .BR ← tinpt3;
    (tinecl):
    IF echofg ≠ 3 THEN outptc(.BR);
    RETURN(tinpt3,tinpt1)
END.
(txtlit)% This PROCEDURE reads a literal into the literal buffer
(litc litlcl) for stateme usage.
It checks for overflow of the short literal buffer, and gets the
long one IF necessary%
PROCEDURE;
IF litc = 0 THEN rerror();
echol(); %turn off break on every characer%
IF NOT rdlit(litc,0) THEN % short lit buffer full%
    BEGIN gt2lit(); %get big lit buffer%
    IF NOT rdlit(litc,0) DO-SINGLE !err $3 %big buffer full%
    END;
echo3(); % turn break back on%
RETURN
END.
(gt2lit)%gets the second literal buffer, checking for errors and
copying string from name register across%
PROCEDURE;
IF litrf DO-SINGLE !err $6; %big lit already got%
EXECUTE prmspc;
gtlit2(); % get big lit reg%
cbystr($stnxs,,litc); %copy old part across%
RETURN END.
%todas character and command echo control routines % %todas%
(echo) %This is the pop which does all of the echoing for todas
It expects the address of a cell containing the string to be
echoed in the address field of the pop.
It types characters from the string onto the tty until either
(a) a period is encountered in the string
(b) the number of characters typed is equal to the value in the
variable 'feedbk'.
%
POP 174B PROCEDURE;
.XR←[%0]; %address of string to be echoed%

```

```

echot3 ← feedbk;
LOOP BEGIN
  echot2 ← 2;
  .BR ← $0[.XR/];
  (echol): echot1 ← LSH $8() .A 377B; %chr to be echoed%
  IF .AR = '. THEN EXIT;
  IF SKIP (SKR echot3) THEN EXIT; %end of string or feedbk%
  TCO echot1; %type chrs%
  IF NOT DECNEG echot2 DO-SINGLE GOTO echol;
  %by here..done with word..get next in string%
  .XR←.XR+1
END;
RETURN END.
(echo0)PROCEDURE;
%echo each character with itself; each character is a break
character%
BRS $12(echofg←0,, -1); RETURN END.
(echol)PROCEDURE;
%echo all characters, break on all but space, letter and digits%
BRS $12(echofg←1,, -1); RETURN END.
(echo3)PROCEDURE;
%no echo for any character; break on all characters%
BRS $12(echofg←3,, -1); RETURN END.
(pushsp) %Push the A-B registers on the spec stack.%
PROCEDURE(pshspl);
IF spskt ≤ spsk DO-SINGLE rerror();
spsk←+2;
.XR ← spsk;
.AR←pshspl;
$0[.XR/←.AR; $1[.XR/←.BR;
RETURN END.
%q-marks, arrow, bug setting, cfl, lt%
(qmon) %This displays a ? in the command feedback line.%
PROCEDURE;
DECLARE qmon1 = " ? ";
d1pqb ← d1pqb .V qmon1;
RETURN END.
(qmoff) %This turns the ? off.%
PROCEDURE;
d1pqb ← d1pqb .A 77600000B;
RETURN END.
(cflarw) %Given a position integer in the B, this routine puts an up
arrow under the string in that position of the command feedback
line.%
PROCEDURE(,cflptr);
DECLARE cflara = "↑ ";
d1pqa←(d1pqa .A 77770000B) .V ((RSH $12(cfltbl/cflptr/)) *(dlxcfl
.A 77B) + cdcflh);
d1pqb ← cflara;
RETURN END.
(san) %Turn the command feedback arrow on.%
PROCEDURE;
d1pqb ← d1pqb .V cflara; % "↑ " %

```

*cf
so not
as seen
by this
level*

why do st.?

```

RETURN END.
(saf) %Turn the command feedback arrow off.%
PROCEDURE;
  dlpqb ← dlpqb .A 177777B;
  RETURN END.
(delul) %This routine turns any bug marks off.%
PROCEDURE;
  dllul ←+ -40000B;
  cduhpt ←+ -1;
  RETURN END.
(uloff) %This PROCEDURE removes all of the marks (currently circles)
that provide feedback of bug selects on the screen.%
PROCEDURE;
  cduhpt ← $dlbul;
  dllul ← (dllul .A 37777B) .V 100000B;
  RETURN END.
(sbmfb) %The character that tracks the mouse position is set by this
routine (Set Bug Mark Feedback Character). The PROCEDURE is called
with the character in the A.%
PROCEDURE (cdtemp);
  dlxbug ← (dlxbug .A 76007777B) .V LSH $12(cdtemp,0);
  RETURN END.
(cflatc) %This routine is called with an A-string address in the A,
and an integer N in the B. It moves the string to the Nth position
in the command feedback line.%
PROCEDURE(cflfrm,cflptr);
  % takes the a string (addr on a) and puts into the cfl in
  position%
  %specified by the b%
  IF 9 <= cflptr DO-SINGLE !err $6;
  cfltoo ← $dlbcfl + cfltbl/cflptr/ .A 7777B;
  %cfltoo is the first unused location in the cfl%
  IF cdabfl THEN BEGIN
    [cfltoo]+$2(cflfrm) .A 77600000B .V 77577B;
    cflbnd+cfltoo END
  ELSE cflbnd ← asrbuf(cflfrm,cfltoo);
  %cflbnd is the last loc used%
  IF [cflbnd] .A 177B = 177B THEN [cflbnd] ← [cflbnd] .A 77777400B
  ELSE BEGIN
    BUMP cflbnd;
    [cflbnd] ← 177777B END ;
  BUMP cflfrm;
  $1/$cfltbl + cflptr/ ← $0/.XR/ + LSH $12(IF cdabfl THEN 2
  ELSE [cflfrm]+2,0) .V (cflbnd - cfltoo+1);
  dllcfl ← dllcfl .A 37777B .V LSH $14(cflbnd - $dlbcfl +3,0);
  RETURN END.
(ltlg) %Several characters is the upper left corner of the screen
provide feedback of the current view specs. Calling this routine
will display those characters in a large character size.%
PROCEDURE;
  dlxlt ← dlxlt .A 77776000B .V 1722B;
  RETURN END.
(ltsm) %Calling this routine will set the display of the view spec

```

```

feedback to a small character size.%
PROCEDURE;
  d1brlv ← 37700000B;
  d1xlt ← d1xlt .A 77776000B .V 1116B; us dk
  RETURN END.
%a-string stuff%
(from cvsno) %An A-string containing digits integer can be converted to a binary
integer by calling CVSN0 with the A-string address in the A. The
conversion is done to the base 10, and all characters are assumed to
be digits, and are not checked. The A-string is not changed, and the
integer is RETURNED in the .A. If the first character is a minus
sign, the number will be converted to a negative number correctly.%
PROCEDURE(cvsno1); %convert a-string to integer%
  cvsno3←cvsno2+cvsno4 ← 0;
  IF 1dchr(cvsno1,0) = '-' THEN BUMP cvsno2,cvsno4; slay
  LOOP BEGIN
    cvsno3 ← cvsno3*10 + 1dchr(cvsno1,cvsno2)-'0';
    IF (cvsno2 ← cvsno2 + 1) > $1/cvsno1/ DO-SINGLE EXIT
  END;
  RETURN(IF cvsno4 THEN
    -cvsno3 ELSE cvsno3)
  END.
(asrnam) %An A-string is displayed in the name area by calling
ASRNAM with the address of the A-string in the A. The routine
converts the case of letters in the string for the display, by moving
the A-string character by character (and converting) to an A-string
named NUMREG. It THEN calls ASRBUF to move NUMREG to the name
register display buffer.%
PROCEDURE (astnm1);
  BUMP astnm1;
  IF [astnm1] > d1bnmx THEN astnm4 ← d1bnmx
  ELSE astnm4 ← .AR;
  numreg ← 60;
  astnm2 ← 0;
  numrg1 ← -1;
  numreg ← 60;
  WHILE astnm2 <= astnm4 DO BEGIN
    astnm3 ← 1dchr(astnm1 -1,astnm2);
    BUMP astnm2;
    apchr( astnm3,,#numreg) END;
  asrbuf($numreg,$d1bnam);
  .AR ← .AR - $d1bnam +3;
  d1lnam ← LSH $14(,0) .V $d1pnam;
  RETURN END.
(bkachr) %Given the address of an A-string in the A, this routine
wil subtract one from the length, and THEN RETURN the last character
of the string in the A.%
PROCEDURE; %(a-string address in a)%
  IF $1/.AR/ > -1 THEN $1/.XR/ ←+ -1
  ELSE RETURN(0);
  IF carpos > 0 THEN carpos ←+-1; %carriage position for todas%
  RETURN(1dchr(.XR, $1/.XR/)) ) END.
(numdpy) %This routine is used to convert a binary integer to an ASCII

```

viewchange "bug"

number. The integer is provided in the A, and the routine RETURNS with the number in the A, in ASCII. This PROCEDURE is used primarily for displaying the view specs. Hence "all" is RETURNed IF a number less than zero or greater than 100 is given in the A. The ASCII number is left justified with blanks added at the END.%

```

PROCEDURE (numd1);
  IF (numd1 < 0) OR (numd1 = 100) THEN RETURN("ALL");
  numd3 ← 0;
  numd4 ← 3;
  WHILE numd4 DO BEGIN
    numd4 ←+ -1;
    numd1 ← numd1/10;
    .AR ← .BR + 20B;
    .AR ← RSH $8(,numd3);
    numd3 ← .BR;
    IF numd1 = 0 THEN RETURN(numd3) END;
  RETURN (numd3) END.

```

strategy on this (conventions comprehensible to a user?)

(xintr9) %This routine carries out all the details of finishing up when an INPUT QED BRANCH command is done. It is activated by pseudo-interrupt 209. It stops the fork that listens for rubouts WHILE the input is done, stops the input fork and relabels out the PASS1 code, closes the input file, removes the message, and recreates the display. It does not RETURN from the interrupt but goes to previous state.%

```

PROCEDURE;
  .AR ← $outfpt;
  IF NEG outfpt6 DO-SINGLE BRS $32();
  BRS $32($inpft);
  BRS $121(tpass1);
  %delete when change data page% tcalc ← toutov;
  inpfpt ← inpr11;
  inpfpt5 ← inpr12;
  inpfpt ← $inptf1;
  paslfg←inptfg ← 0;
  inptbo ← inptbi;
  BRS $9($inpft .V 66000000B);
  BRS $78(intmsk);
  BRS $20(outfn);
  dismes(,0);
  .AR ← -1;
  lany txt;
  GOTO gps END.

```

CS for selectivity

OK! CS earlier

(cptstr) %The function of this routine is to copy a T-string to an A-string. The calling arguments are the address of a T-pointer in the A and another in the B, and an A-string address in the X. It copies the T-string determined by the two T-pointers into the A-string, after setting the A-string to null. The two T-pointers must point to the same statement, or RERROR is called. An external flag is associated with this routine, named CPTSRC. If it is true, the string is forced to upper case. The flag is always reset before the routine RETURNS. Note: IF the T-pointers point to the same character, one character will be moved into the A-string. The END characters of the T-string are included in the resultant A-string.%

```

PROCEDURE(cptsr1,cptsr2,cptsr3);

```

```

%copy t-str to a-str%
IF [cptrs1/ # [cptrs2/ DO-SINGLE !err $3;
swork ← [cptrs1/;
BUMP cptrs1;
swork1 ← [cptrs1/;
BUMP cptrs2;
$1/[cptrs3/ ← -1;
%null string%
cptrs2 ← [cptrs2/+1;
.AR ← fechcl(1,,swork);
BRM $2[.XR/;
cptrs1 ← .AR;
WHILE (.AR # endcar) AND (swork1 ≤ cptrs2) DO BEGIN
  IF cptrsre THEN BEGIN
    IF cptrs1 IN (100B,132B/ THEN cptrs1 ← .AR-40B
    END;
    apchr(cptrs1,,cptrs3);
    .XR ← swork;
    BRM $2[.XR/;
    cptrs1 ← .AR END;
  cptrsre ← 0;
  RETURN END.
(chkfrz) %This routine calls ERROR IF any core blocks are frozen.
That is, the FRZCPT (frozen core page table) must contain -1 in all
positions.%
PROCEDURE;
  frzrfl←-1; WHILE (frzrfl←frzrfl+1) ≤ rficbx DO IF frzcpt/frzrfl/ #
  -1 DO-SINGLE rerror();
  RETURN END.
%pops%
(lec) %Load Entity Character. The entity character (SMEC) is set to
the character in the address field of the pop.%
POP 103B PROC; %load entity character %
  EAX [%0/ (,,0);
  smec←.XR;
  RETURN END.
(les) %Load Entity String. The entity string (SMES) is set to the
A-string address provided in the pop address field.%
POP 104B PROC; % load entity string %
  EAX [%0/ (,,0);
  smes←.XR;
  RETURN END.
(gset) %Group SET. The current group is set to the group number
given in the address of the pop. If the group has changed, the
entity character and string are set to the initial values for that
group. The initial values are in IGRPC and IGRPS.%
POP 110B PROC; %set group and entity %
  EAX [%0/ (,,0);
  IF .XR = curgrp DO-SINGLE RETURN;
  IF .AR > $groupx DO-SINGLE rerror();
  smec←igrpc[.XR/;
  smes←igrps[.XR/;
  curgrp←.XR;

```

```

RETURN END.
(mlf1) %Move Literal to Feedback line, first position. This pop
sets the feedback line position counter (CFLPOS) to -1 and THEN con
executes the code for a MLF pop.%
POP 111B PROC;
    %move literal to feedback - first pos %
    cflpos←-1 END. goto(mlf) (?)
    (mlf) %Move Literal to Feedback line. This pop increments the
Sbt correct feedback line position counter, by one and THEN moves the A-string,
whose address is in the address field of the pop, to the feedback
line by calling CFLATC.%
POP 111B PROC;
    %move literal to feedback - next%
    BUMP cflpos END.
(mlfr) %Move Literal to Feedback line and Replace last. This pop is
the same as MLF except that it does not increment the position
counter before moving the string.%
POP 112B PROCEDURE;
    %move literal -over last%
    popret ← .BR ← $0;
    .BR←cflpos;
    EAX [$0];
    BRR popret(cflatc(.XR .A 37777B)) END.
epf (rep) %REpeat a case statement. The REP pop removes the overlay
address from the circular case save array, resets the pointer (CASV)
to the previous entry, and branches to the saved location using
routine OVLGO. The address of the pop contains an integer which
indicates how many cases to back up over. A REP 0 is the normal
edition, which RETURNS to the instruction following the last CSE pop,
and does not change the case array pointer. A REP 1 backs up the
pointer by one. The big problem with the current case business is
that the pointer is not moved back when a repeat is NOT done. Hence
REP 1 or REP 2 will not RETURN to the correct location IF someone
branched out of an inner case.%
POP 113B PROC; %repeat in case statement%
    smcreg←smbreg;
    .XR←casv;
    IF [$0] > 0 THEN BEGIN
        casv←(-(.AR-casv)) .A $ncasv1;
        .XR←.AR END;
    IF NOT NEG casvb[.XR] DO-SINGLE BRR casvb[.XR];
    ovlgo(casvb[.XR]) END.
(dfs) %DeFINE State. The overlay address is saved in STATE, and
the current group is set to the integer given in the address of the
pop. A ZAP is THEN executed.%
POP 114B PROC; % define a state %
    state←abtgt←ovladr($0);
    EAX [$0](,,0);
    curgrp←.XR;
    GOTO zap END.
(cta) POP 117B PROCEDURE;
    popret ← $0; %save return loc%
    IF [$0] THEN BUMP ctsrc;

```



```

BRR popret(cptstr($sptr1,$sptr2,$sarxn)) END.
(anypop) POP 177B PROC;
EXTERNAL creg,breg,sreg,sba,sbp,ta, mrf,mrfr,mrkl,mrk, kin,
kct,pbm,cse, spcb,spcr, spcrno,spcrp,spcrna, mrs,bkc,dpn,spca,
spcf,spcn,cct, cct,zero,zap,zapsp,feedlt;
popret ← .AR; .AR←$0; XMA popret;
GOTO [$0];
(creg): %append the character in C-REG to string in A-reg. The
character in SMCREG is appended to the register whose number is in
SMAREG.%
    regadr(smareg);
    BRR popret(apchr(smcreg));
(breg): % append char in b to reg no. in a %
    regadr(smareg);
    BRR popret(apchr(smbreg));
(sreg): % append sar to reg no. in a %
    regadr(smareg);
    BRR popret(apsr($sarxn));
(sba): % set bug to arrow %
    BRR popret (sbmfbc(rmdcr));
(sbp): % set bug to plus %
    BRR popret (sbmfbc(drmdcr));
(ta): %display in text area%
    IF (NOT litrf) AND (stnrxn ≤ stnl+1) THEN gt2lit();
    IF (smbreg ≤ 132B) OR (.AR = 151B) OR (.AR = 155B)
    THEN apchr(smbreg,,litlc);
    IF NOT paslfg THEN aplit(smbreg);
    BRR popret;
(mrf): % move register to feedback line %
    BUMP cflpos;
(mrfr): % move register to feedback (same spot) %
    regadr(smareg);
    .BR←cflpos;
    BRR popret (cflatc(.XR .A 37777B));
(mrkl): %MARK feedback line for up arrow in first position. This
calls CFLARW to display an up arrow under the first position of the
feedback line.%
    BRR popret(cflarw(0,0));
(mrk): %MARK feedback line for up arrow. This pop has CFLARW
display an up arrow under the NEXT position in the feedback line.%
    BRR popret(cflarw(0,cflpos+1));
(kin): %INPUT a character. The routine INPTM is called to read a
character. The character is stored in SMCREG. QMOFF is also
called, to make sure a "?" is not being displayed.%
    smcreg←inptm();
    BRR popret(qmoff());
(kct): %Character Test. This compares the character in SMBREG with
the input character in SMCREG and sets FLAG IF they are equal,
resets it IF not. The input character is forced to upper case
before the compare.%
    .BR←1;
    IF smcreg IN (100B,132B) DO-SINGLE .AR←.AR-40;
    IF .AR # smbreg DO-SINGLE .BR←0;

```

```

    flag←.BR;
    BRR popret;
    (pbm): %Process Bug Mark. This pop resets the flag IF the input
    character (in SMCREG) is not a command accept. Otherwise it sets
    the flag, reads the bug coordinates and stores them in BUGREG.%
    flag←0;
    IF smcreg # $ascca DO-SINGLE BRR popret;
    BUMP flag;
    readbm();
    STP bugreg;
    BRR popret;
    (cse): %CaSE statement. This pop causes the overlay address to be
    "pushed" onto the case save stack (which is circular). The flag
    is set to false.%
    casv←(casv+1) .A $ncasvl;
    csel←.AR;
    casvb/csel/←ovladr($0);
    flag ← 0;
    BRR popret END.
    (spcb)PROCEDURE; %SPEC a Bug mark. If BUGREG does not already
    contain a T-pointer (a -1 in the first word) THEN either PBUG or
    TREBUG in TREE is called to PROCESS the bugmark into a T-pointer.
    The T-ponter is pushed onto the spec stack.%
    LDP bugreg;
    IF .AR = -1 THEN BEGIN
        IF NOT cdref THEN pbug()
        ELSE BEGIN
            txtloc ← ovladr($0);
            EXECUTE rfbp2;
            EXECUTE cdsply;
            EXECUTE disbuf;
            trebug(,bugrgl);
            pushsp();
            ovlg(txtloc) END END;
    BRR popret (pushsp());
    (spcr): % SPEC a Register. This is a rather complicated pop that
    converts any string register to a T-pointer or number for the spec
    stack. Depending on which register is indicated by SMAREG,
    (STNO) It converts the string (a statement number) to a PSID by
    calling FECHUX.
    (NUMN) It converts the string to an integer by calling CVSTNO,
    and pushes (0,number) on the spek stack.
    Otherwise, IF the first character of the string is a digit, it
    converts it as a statement number as above, or ELSE it treats
    it as a name and finds the corresponding PSID by calling
    LKNAMH. %
    smtemp ← $0;
    EXECUTE rfbp2;
    $0 ← smtemp;
    IF smareg = $stnon DO-SINGLE GOTO spcrno;
    IF .AR # $numnn DO-SINGLE GOTO spcrna;
    cvsno($numnxxn);
    COPY (AB,A);

```

move about this?

?

```

GOTO spcrp;
(spcrno):
regadr();
(spcrnl):
fechux(.XR);
!abort $2;
(spcrp):
BRR popret(pushsp());
(spcrna):
regadr();
IF NOT NEG $1/.XR/ THEN BEGIN
  IF LRSH $16($2/.XR/) IN (17B,31B/ DO-SINGLE GOTO spcrnl END;
.AR ← 0;
IF NEG $1/.XR/ DO-SINGLE GOTO spcrp;
hash(.XR);
lknamb();
!abort $3;
GOTO spcrp;
(mrs): %Move Register to String area. This used CPYSR to move the
A-string indicated by SMAREG to SAR.%
  regadr(smareg);
  BRR popret (cpysr(.XR,,$sarxn));
(bkc): %Backspace Character. The A-string indicated by SMAREG is
backed up one character via routine bkachr.%
  regadr(smareg);
  BRR popret(bkachr(.XR));
(dpn): %Display in Name area. The A-string indicated by SMAREG is
displayed in the name area.%
  regadr(smareg);
  BRR popret(asrnam(.XR)); (hashes)
(spcn): %This gets the hash of the A-string indicated by SMAREG
and puts (-1,hash) on the stack.%
  regadr(smareg);
  BRR popret(pushsp(-1,hash(.XR))) END.
(cct)PROCEDURE; %Character Class test. This pop does the
character class testing for the CASE statement. The letter class
includes both upper case and lower case letters. The classes are:
(4) Letter
(5) Digit other?
(6) Any character.%
.BR ← 1;
IF smareg = 6 THEN NULL
ELSE IF .AR = 5 THEN BEGIN
  IF smbreg NOT IN (17B,31B/ DO-SINGLE .BR←0 END
ELSE IF .AR = 4 THEN BEGIN
  IF smbreg NOT IN (40B,72B/ THEN BEGIN
    IF .AR NOT IN (100B,132B/ DO-SINGLE .BR←0 END END
ELSE rerror();
flag ← .BR;
BRR popret;
(zero): %ZERO the spec stack. This does sets the stack pointer to
the bottom of the spec stack.%
  spsk←spskl;

```

```

BRR popret END.
(zap)PROCEDURE; %ZAP everything. This includes:
Collapse the general stack.
Release the long literal register, IF it exists.
Reset the spec stack.
Relabel out the two random file block pages (RFBP1 and RFBP2).
Call CHKFRZ, LTSM, RESDPY, and ULOFF. %
stack←stack1;
(zapsp): %ZAP the SPec stack. This is exactly the same as ZAP,
except that the general stack is not changed.%
spsk←spsk1;
rellit();
smtmp ← $0;
EXECUTE nrfbp2;
$0 ← smtmp;
chkfrz();
IF todas DO-SINGLE BRR popret; %todas%
BRR popret (ltsm(resdpy(uloff())));
(feeldt): %FEED LT specs. This pop requires that PRMSPC be in,
since it calls SETLT. However, it does not check. The characters
in the A-string indicated by SMAREG are fed to SETLT one at a
time.%
regadr(smareg);
feeldt3 ← .XR;
feeldt2 ← $1/.XR/;
feeldt1←-1; WHILE (feeldt1←feeldt1+1) <= feeldt2 DO BEGIN
    lachr(feeldt3,feeldt1);
    setlt() END;
BRR popret END.
(tch) %Test character. Sets SMBREG to the character looking for, THEN
goes to kct.%
POP 146B PROCEDURE;
EAX ($0/(/,0);
smbreg ← .XR;
GOTO kct END.

```

wheel

:NMOLR, 12/12/69 0029:04 WHP ; .DPR=0; †P1 SE(P1) < \$NP -'. ;
/"con"); %%

TITLE: MOL-940: An Algol-like Machine Oriented Language for the SDS 940.

ABSTRACT .

This report describes a programming language developed at Stanford Research Institute for the Scientific Data Systems 940 computer.

The name MOL is an acronym for "Machine-Oriented Language". MOL-940 is an ALGOL-like language with extensions to make it suitable for systems programming. The added syntax strongly reflects the internal design of the SDS 940, in accordance with the name MOL.

The compiler is written using a meta-compiler system, TREE META. The formal specification of the language given in the report is actually the TREE META program for the compiler.

MOL-940

CONTENTS.

Abstract.
Contents.
Foreword.
Introduction.
Structure of the language.
 General.
 Primitives.
 Program.
 Procedure.
 Coroutine.
 Pipe.
 Statements.
 Arithmetic Expressions.
 Logical Expressions.
 Logical Evaluation.
 Declarations.
 Miscellaneous.
Library.
Bibliography.

:INPFBK, 10/31/69 1740:14 WSD ; INPUT FEEDBACK .SCR=1; .PLO=1; .MCH=75;
.RTJ=0; .DSN=1; .LSP=0; .MIN=70; .INS=3; ["syserr"];

Description of the INPFBK overlay

The INPFBK overlay is one core page of MOL PROCEDURES that are used by other overlays that DO user terminal input and display output. The display buffer page is always relabeled in when INPFBK is in, since many display feedback routines are in INPFBK. Also, tables in the display buffer page are used to PROCess bug marks. The tables for decoding the binary handset strokes are also in INPFBK. The INPFBK page is read-only and shared.

```

%.HED="INPFBK,  inpfbk overlay %
DECLARE origin(17777B);
DECLARE inpkch = (
  35660536B,220501B,421102B,
  621503B,1022104B,1222505B,
  1423106B,1623507B,2024110B,
  2224511B,10025112B,2625513B,
  3226114B,2426515B,3627116B,
  17427517B,4030120B,4230521B,
  4431122B,4631523B,5032124B,
  5232525B,5433126B,5633527B,
  6034130B,6234531B,7235132B,
  16616014B,17217016B,17615033B,
  27236037B,33264400B);
DECLARE inpkht = (0,
  1000101B,1000102B,1000103B,
  1000104B,1000105B,4000000B,
  1000107B,1000110B,1000111B,
  1000112B,1000113B,1000114B,
  1000115B,1000116B,1000117B,
  1000120B,1000121B,1000122B,
  1000123B,1000124B,1000125B,
  1000126B,1000127B,1000130B,
  1000131B,1000132B,1000000B,
  1000000B,1000000B,100B,
  1000000B);
%input characters%
(readbm) %This PROCEDURE RETURNS in the A and B the mouse
coordinates from the last command accept character or pointer.  In
the case of a bug mark or CA key, the 24 bit coordinates are RETURNed
in the B, with a -1 in the A.  In the case of a pointer, the
T-pointer is RETURNed in the A and B.%
PROCEDURE;
  RETURN(inptpr,inptmr) END.
(inptO) %This routine RETURNS a character from the keyboard,
handset, or mouse.  If no characters are in the system's input
buffer, the PROCEDURE dismisses (the main fork) on the highest
priority queue until there is a character waiting.  This is done with
BRS's -10 and 72 which are EXEC only.  The format of the character
RETURNed is the same as from the system.  The clock reading is saved
in INPTCL, the mouse coordinates are saved in INPTCM, and the
character is RETURNed in the A.%
PROCEDURE;
  %high priority- lowest level input routine%
  (inptO1):
  WSI ttyno;
  IF .AR CB 100000B DO-SINGLE GOTO inptOw; %no character read%
  inptOc ← .AR .A 77077777B;
  inptcl ← .XR;
  inptcm ← .BR;
  RETURN(inptOc);
  (inptOw):
  BRS $-10(, ,ttyno);

```



```

    BRS $72(); %wait for workstation input%
    GOTO inpt01 END.
(inptc) %INPTC is the single character input routine. It RETURNS
characters from the input devices in the A, in the following format:
A command accept button going down is a -1, coming up is a -2, and
all other characters are in ASCII. This routine takes care of the
binary handset. Button strokes or special handset chords are
PROCESSED, but INPTC does not RETURN until a real character for the
main part of NLS is read. INPTC functions approximately in this
way:
    First a character is obtained.
        If there is a character in the input buffer, it is taken.
        Otherwise, IF Pass1 is running, THEN the routine dismisses
        itself (using BRS's -10 and 72), after setting a flag for the
        input fork to create pseudo-interrupt 207.
        Or IF the input fork is active, INPTC waits (BRS 45) and goes
        back to its BEGINning.
        And finally, it calls INPT0 to read the character IF all the
        above fail.
    Having obtained a character, it checks to see IF it is a rubout
    and PROCESSES it IF it is. Or IF it is a command accept code of
    some kind, a flag is set to indicate that and the coordinates are
    saved.
    Then the character is PROCESSED through a switch, depending on the
    source of the character (mouse, keyboard, handset).%
PROCEDURE;
%RETURNS a character from input devices%
DECLARE inptyp(7)=(inptb,inptb,inptk,inptcx,
    inptbg,inptp,inptx,inptcx);
DECLARE inpksh(4)=
    (inptb,inplts,inptcx,inptcx,inpcd);
EXTERNAL inptcx;
(inptcx):
IF inptbi # inptbo THEN BEGIN
    inptcl ← /inptbo/;
    BUMP inptbo;
    IF inptbo >= inptbe THEN inptbo ← $inptbf END
ELSE IF paslfg THEN BEGIN
    BUMP inptfg;
    BRS $72(,100200B,0) %suspend until interrupt%
    END
ELSE IF inptfa THEN BEGIN %input fork is running, wait for it to
stop%
    BRS $45();
    GOTO inptcx END
ELSE BEGIN
    BUMP inptfg;
    IF todas THEN TCI inptcl ELSE
    inptcl ← inpt0();
    inptfg ← 0 END;
IF inptcl = 1000137B THEN BEGIN
    IF inptrf THEN BEGIN
        BUMP inptfg;

```

```

    intr5() END
ELSE BEGIN
    BUMP inptrf;
    IF todas THEN EXECUTE todmnc ELSE EXECUTE mnctr1;
    IF paslfg DO-SINGLE ursnls();
    IF todas THEN GOTO reset ELSE GOTO cmdrst
    END END
ELSE BEGIN
    IF inptcl = 1000144B or .AR = 4000001B THEN BEGIN
        inptmr+inptcm;
        inptpr+-1 END;
        inptrf ← 0; IF todas THEN RETURN(inptcl) END;
GOTO (inptyp/(RSH $18(inptcl,0) .A 7));
(inptb):
    BUMP inp;
    IF incse = 3 THEN GOTO inplts
    ELSE RETURN(LSH $18(0) .A 177B);
(inptk):
    BUMP inp;
    IF incse = 3 THEN GOTO [inpksh/(LRSH $18(inpkht/(LSH
    $18(0)),0))]
    ELSE RETURN((LRSH $0[.XR]/(inpkch/(LSH $18(0) .A 37B)),, LSH
    $3(incse,0)) .A 177B);
(inptbg):
    RETURN(IF .BR CB -1 THEN -1
    ELSE -2);
(inptx):
    inbbtn+2;
    GOTO inptck;
(inptp):
    inbbtn+1;
    % GOTO inptck; %
(inptck):
    IF .BR CB -1 THEN BEGIN
        inp,inwait+0;
        incse+incse .V inbbtn END
    ELSE BEGIN
        incse+incse .A (inbbtn .X 3);
        IF NOT inp THEN BEGIN
            IF incse = 0 THEN BEGIN
                IF inwait THEN RETURN(LRSH $16(inpkch))
                ELSE IF inbbtn = 1 THEN RETURN(inpkch .A 177B)
                ELSE RETURN(LRSH $8(inpkch) .A 177B) END
            ELSE BUMP inwait END END;
        GOTO inptcx;
(inpcd):
    inptcl ← cortb[5];
    inptc2 ← cortb[6];
    inptc3 ← $0;
    .AR ← -1;
    !any txt;
    ovldn(, ,inptcl);
    ovldn(, ,inptc2);

```

```

        BUMP ovrelf;
        EXECUTE inpfbk;
        $0 ← inptc3;
        GOTO inptcx;
(inplts):
        inptc2 ← LSH $18(0);
        inptc3 ← $0;
        inptc1 ← cortb/6);
        EXECUTE prmspc;
        setlt(inptc2);
        ovldn(,,inptc1);
        BUMP ovrelf;
        EXECUTE inpfbk;
        $0 ← inptc3;
        GOTO inptcx END.
(inptm) %This is the main program's input routine. It is one
logical level higher than INPTC in that it PROCesses pointers and
turns command accept buttons up-down into command accept characters
(144b). The pointer names are read into an A-string named INPPR,
and LKPTR is called to obtain a T-pointer. %
PROCEDURE;
%main programs input routine%
(inptml):
        IF todas THEN RETURN(tinptc());
        inptml ← inptc();
(inptmx):
IF .NEG inptml THEN BEGIN
        IF .AR = -1 THEN BEGIN
                inppr ← 2;
                inppr1 ← -1;
                inppr2 ← 0;
                inptml ← inptc();
                WHILE inptml # -2 DO BEGIN
                        IF .NEG inptml THEN labort $5;
                        apchr(inptml,,$inppr);
                        inptml ← inptc() END;
                IF .NEG inppr1 THEN RETURN($ascca);
                lkptr(inppr2);
                labort $5;
                inptmr ← .BR;
                inptpr ← .AR;
                inptml ← $ascca END
                ELSE IF .AR = -2 THEN GOTO inptml
                ELSE rerror() END;
        RETURN(inptml) END.
(lkptr) %LKPTR takes a three character pointer name in the A and
RETURNS the T-pointer for that pointer in the A and B. IT skips IF
that pointer exists, does not skip IF it fails to find the pointer in
the table.%
PROCEDURE(lkptr1); %3 character name in .AR%
        FOR lkptr2 FROM 0 INC 3 TO ptrtb1 DO IF ptrtb/lkptr2/ = lkptr1
        THEN BEGIN
                %skip RETURN%

```

```

        SKIP RETURN (ptrtb/lkptr2+1/, ptrtb/.XR+1/) END;
RETURN %no skip%
END.
%todas kludge for opening and loading files%
(tdiokd)PROCEDURE; %todas io kludge%
%the following is a kludge for todas...very very temporary%
EXTERNAL outckp,outfil,lodekp,lodfil;
(outckp):
    EXECUTE rfbp2;
    EXECUTE ioctl;
    frecor();
    opnckp();
    copfil(rffn,ckptfn);
    BRS $20(ckptfn);%close file%
    ckptfn ← 0;
    GOTO gps;
(outfil):
    EXECUTE rfbp2;
    EXECUTE ioctl;
    frecor();
    LOAD(1,72000000B);
    IF fnml > -1 THEN BEGIN % type file name and look for ca%
        typeas fnmxn;
        IF inputc() # $ascca DO-SINGLE GOTO outfil1;
        curfn ← opnfil(1,72000000B,$fnmxn)
    END ELSE BEGIN
        (outfil1): smcreg ← .AR; %save off character%
        curfn ← opnfil(1,72000000B,0);
        cpysr($stnrxn,, $fnmxn) END;
    ofdate();
    lany mks; ilas =pl; lsov txtedt; lsbc setrot;
    EXECUTE ioctl;
    frecor();
    copfil(rffn,curfn);
    BRS $20(curfn);
    curfn ← 0;
    GOTO gps;
(lodekp):
    EXECUTE rfbp2;
    EXECUTE ioctl;
    frerff();
    opnckp();
    rdhdr(ckptfn);
    copfil(ckptfn,rffn);
    BRS $20(ckptfn);
    ckptfn ← 0;
    fnroot();
    GOTO gps;
(lodfil):
    EXECUTE rfbp2;
    loctl;
    savrff();
    frerff();

```

```

    curfn ← opnfil(0,72000000b,0);
    cpysr($stnxxn,, $fnmxxn);
    BRS $20(rifn);
    rffn ← 0;
    rdhdr(curfn);
    GOTO gps
END.
%todas i/o routines% %todas%
(inpcuc)PROCEDURE;
    %input a character and translate it to upper case IF necessary%
    IF inputc() IN (100B,132B) THEN buff ← .AR - 40B;
    RETURN END.
(inputc)PROCEDURE;
    %read one character from tty or buffer into lastchr%
    %retrun chr in a%
    buff←lookc(); bufffg ← .BR; buffct←+-1; RETURN(buff)
    END.
(lookc)PROCEDURE;
    %look at next haracter in input buf or tty%
    %RETURN in a%
    %RETURNS a flag in b register: 1=chr literal, 0=chr normal, -1=chr
    control%
    If buffct THEN RETURN(buff/buffct/,bufffg/buffct/);
    tintpc(); BUMP buffct; buff/buffct/←.AR;
    bufffg/.XR/←.BR;
    RETURN END.
(todco)POP 000B PROCEDURE;
    %output a character to tty%
    IF putchr(/$0/,carpos,0) = -1 THEN % carrage RETURN time % crlf()
    ELSE BEGIN
        IF carpos+.AR > prcolx THEN crlf() ELSE
            carpos←.AR
    END;
    RETURN END.
(putchr)PROCEDURE(putch1,putch2,putch3);
    %put a character onto tty or a-string IF .XR # 0%
    IF .AR = $asctab THEN BEGIN % tab in here %
        IF gettab(putch2+1) > prcolx THEN RETURN(-1);%tab wont fit on
        line%
        putch1←putch2←.AR-putch2; %number of spaces to emit%
        (pttab1):
            IF not putch3 THEN TCO 0 ELSE
                apchr(0,,putch3);
                putch1 ← +-1;
                IF putch1 DO-SINGLE GOTO pttab1;
            RETURN(putch2)
    END;
    IF .AR = $ascrcr THEN RETURN(-1); %indicate crlf%
    IF prcapf THEN BEGIN % all upper case, slash before caps%
        .AR ← putch1;
        IF NOT IN (40B,72B) DO-SINGLE GOTO putclc;
        IF NOT putch3 THEN TCO '/' ELSE
            apchr('/',,putch3);

```

```

        .AR ← putcl; GOTO putcr2;
        (putcl): IF NOT IN (100B,132B) DO-SINGLE GOTO putcsc;;
        .AR ← .AR - 40B;
        GOTO putcr1 END;
    (putcsc): IF putcl > 132B THEN BEGIN
        IF NOT putc3 THEN TCO '& ELSE
        apchr('&,putc3);
        .AR ← putcl-100B; GOTO putcr2 END;
    (putcr1):
        .BR ← .AR;
        IF NOT putc3 THEN BEGIN putc3 ← .BR; TCO putc3 END ELSE
        apchr(.BR,,putc3); RETURN(1);
    (putcr2):
        .BR ← .AR;
        IF NOT putc3 THEN BEGIN putc3 ← .BR; TCO putc3 END ELSE
        apchr(.BR,,putc3); RETURN(2)
    END.
    (answer) %this PROCEDURE accepts a yes or no answer from the
    keyboard, and RETURNS with a 0 in the a register IF the answer was
    negative, and a 1 IF it was positive%
    PROCEDURE;
    (ansstr):
    CASE inptm() OF
        = $ascca: GOTO anspos;
        = 'Y: BEGIN (anspos):
            lecho "Yes."; RETURN(1) END;
        = 'N: BEGIN
            lecho "No."; RETURN(0) END;
        = $asccd: GOTO gps
    ENDCASE BEGIN
        !todco '?;
        !todco 0;
        GOTO ansstr
    END
    END.
    (rdlit)PROCEDURE(rdlit2,rdlit1);
    %read a literal from the keyboard, doing all of the control
    things, plus breaking on any control character identified by the
    list in rdlit1%
    %RETURN 0 IF error (buffer overflow), 1 IF ok%
    IF .AR THEN BEGIN %ok .. a-string here%
    (rdlit1):
        inputc(); IF NOT .NEG bufffg DO-SINGLE GOTO litchr; %not a
        control character%
        CASE .AR OF
            = $asccd: GOTO gps;
            = $ascca: RETURN(1);
            = $ascbc: BEGIN
                bkachr(rdlit2); GOTO rdlit1 END;
            = $ascbw: BEGIN
                IF ldchr(rdlit2,$1/.AR) = 0 THEN %space past blanks%
                WHILE $1/rdlit2/ > -1 AND NOT bkachr(.XR) DO NULL;
                WHILE bkachr(rdlit2) DO NULL;

```

```

        GOTO rdlit1 END;
        = $ascbst: BEGIN $1/rdlit2/←-1; crlf(); GOTO rdlit1 END;
        = $ascdc: RETURN(1) % centerdot%
    ENDCASE BEGIN
        (litchr):
        IF rdlit1 and chrpos(,buff,.AR) THEN RETURN(2);
        (rdentc): apchr(buff,.rdlit2);
        IF $1/rdlit2/ = $0/.XR/ THEN RETURN(0);
        GOTO rdlit1 END
    END; RERROR() END.
(restor)PROCEDURE(restol);
    %put character in a back ijto the input buffer%
    %CALL rerror IF buff ovrflw%
    IF buffsz <= buffct DO-SINGLE rerror();
    BUMP buffct; buff/buffct/←restol;
    bufffg/.XR/ ← .BR;
    RETURN END.
(tinptc)PROCEDURE;
    %This is the main character input routine.
    It does all of the character translation, RETURNing a 0 inthe IF
    the character was a literal, and a -1 IF it was a control
    character. The character is RETURNed in the a%
    %todas control character defiiitions%
    DECLARE todbc = 150B, %control h%
        todbw = 167B, %control w%
        todbst = 160B, %control q%
        todlf = 152B, %line feed%
        todca = 144B, control d%
        todcdt = 142B, %cotrol b%
        todcd = 171B, %control x%
        todshc = '/',
        todlit = '!; % exclamation point%
    tinptl←0;%no special character%
    (tinst):
    IF NOT tinpt2 THEN inptc() ELSE BEGIN .BR←0; tinpt2←.BR END;
    tinpt3←.AR;
    %first check to see IF lower case%
    IF .AR IN (100B,132B/ DO-SINGLE
        GOTO tintrl;
    %now see IF it is a special character%
    (inpnlc):
    IF .AR NOT IN (40B,72B/ DO-SINGLE GOTO tincon;%by here, upper
    case%
        IF precapf THEN tinpt3 ←+40b; %really lower case%
        GOTO tintrl;
    (tincon):
    IF .AR NOT > 132B DO-SINGLE GOTO tinspc; %by here, control
    character%
    tinptl ← .BR ← -1;%mark as control character%
    CASE .AR OF
        = $todbc: BEGIN
            (tinbsp): .BR ← '↑ tinpt3 ← $ascbc; GOTO tinecl
            END;

```

```

= $todlf: BEGIN
  (tinlf): IF echofg # 3 THEN BEGIN TCO $asccr; carpos←0
  END; RETURN(0,0) %linefeed becomes space%
  END;
= $todca: BEGIN
  (tinca): RETURN(,-1)
  END;
= $todcd: BEGIN
  (tincd): .AR ← $asccd; GOTO tinca
  END;
= $todbw: BEGIN
  (tinbsw): .BR ← '\'; GOTO tinecl
  END;
= $todbst: BEGIN
  (tinbst): .BR ← '← GOTO tinecl
  END;
= $todcdt: BEGIN
  (tincdt): .AR ← $asccdt; GOTO tinca
  END;
ENDCASE BEGIN
  tinpt1 ← .BR ← 0; %mark as not control character%
  GOTO tinech
  END;
(tinspc):
CASE .AR OF
= $todshc: BEGIN
  (tinshc): IF precapf THEN
  BEGIN
    IF tinpt3 ← inptc() IN (40B,72B) THEN GOTO tintr2;
    IF .AR IN (100B,132B) THEN BEGIN %lower case
    character%
      tinpt3 ← .AR-40B;
      TCO tinpt3; GOTO tintr2
    END;
  END;
  %by here, next character was not alpha%
  tinpt2 ← .AR;
  tinpt3 ← '/';
  GOTO tintr1
  END;
= $todlit: BEGIN
  (tinlit): %by here, literal%
  tinpt1 ← 1;
  IF ECHOFG ≤ 2 DO-SINGLE BUMP carpos;
  IF tinpt3 ← inptc() > 132b DO-SINGLE GOTO tinech;
  IF .AR = 100b DO-SINGLE GOTO tinech; %centerdot
  non-printing%
  GOTO tintr1;
  END
ENDCASE GOTO tintr1;
(tintr2):
  IF ECHOFG ≤ 2 DO-SINGLE BUMP carpos;
(tintr1):

```



```

        IF 3 <= echofg DO-SINGLE GOTO tintr;
        carpos+carpos+1; IF .AR > prcolx THEN crlf();
        (tintr): RETURN(tinpt3,tinpt1);
(tinech):
        .BR ← tinpt3;
        (tinecl):
        IF echofg # 3 THEN outptc(.BR);
        RETURN(tinpt3,tinpt1)
END.
(txtlit)% This PROCEDURE reads a literal into the literal buffer
(litlc litlcl) for stateme usage.
It checks for overflow of the short literal buffer, and gets the
long one IF necessary%
PROCEDURE;
IF litlc = 0 THEN rerror();
echol(); %turn off break on every characer%
IF NOT rdlit(litlc,0) THEN % short lit buffer full%
        BEGIN gt2lit(); %get big lit buffer%
        IF NOT rdlit(litlc,0) DO-SINGLE !err $3 %big buffer full%
        END;
echo3(); % turn break back on%
RETURN
END.
(gt2lit)%gets the second literal buffer, checking for errors and
copying string from name register across%
PROCEDURE;
IF litrf DO-SINGLE !err $6; %big lit already got%
EXECUTE prm spc;
gtlit2(); % get big lit reg%
cpysr($stxn,.,litlc); %copy old part across%
RETURN END.
%todas character and command echo control routines % %todas%
(echo) %This is the pop which does all of the echoing for todas
It expects the address of a cell containing the string to be
echoed in the address field of the pop.
It types characters from the string onto the tty until either
(a) a period is encountered in the string
(b) the number of characters typed is equal to the value in the
variable feedbk.
%
POP 174B PROCEDURE;
.XR+/$0; %address of string to be echoed%
echot3 ← feedbk;
LOOP BEGIN
        echot2 ←2;
        .BR ← $0/.XR;
        (echol): echot1 ← LSH $8() .A 377B; %chr to be echoed%
        IF .AR = '. OR .DECNEG echot3 THEN EXIT; %end of string or
        feedbk%
        TCO echot1; %type chrs%
        IF NOT .DECNEG echot2 DO-SINGLE GOTO echol;
        %by here..done with word..get next in string%
        .XR+.XR+1;

```

```

END;
RETURN END.
(echo0)PROCEDURE;
%echo each character with itself; each character is a break
character%
BRS $12(echofg+0,, -1) RETURN END.
(echo1)PROCEDURE;
%echo all characters, break on all but space, letter and digits%
BRS $12(echofg+1,, -1) RETURN END.
RETURN END.
(echo3)PROCEDURE;
%no echo for any character; break on all characters%
BRS $12(echofg+3,, -1) RETURN END.
(pushsp) %Push the A-B registers on the spec stack.%
PROCEDURE(pshspl);
IF spskt <= spsk DO-SINGLE rerror();
spsk<+2;
.XR ← spsk;
.AR←pshspl;
$0[.XR]←.AR; $1[.XR]←.BR;
RETURN END.
%q-marks, arrow, bug setting, cfl, lt%
(qmon) %This displays a ? in the command feedback line.%
PROCEDURE;
dlpqb ← dlpqb .V 17400B; %%
RETURN END.
(qmoff) %This turns the ? off.%
PROCEDURE;
dlpqb ← dlpqb .A 77600000B;
RETURN END.
(cflarw) %Given a position integer in the B, this routine puts an up
arrow under the string in that position of the command feedback
line.%
PROCEDURE (,cflptr);
dlpqa←(dlpqa .A 77770000B) .V ((RSH $12(cfltbl/cflptr)) *(dlxcfl
.A 77B) + cdcflh);
dlpqb ← 17400000B; %↑%
RETURN END.
(san) %Turn the command feedback arrow on.%
PROCEDURE;
dlpqb ← dlpqb .V 17400000B;
RETURN END.
(saf) %Turn the command feedback arrow off.%
PROCEDURE;
dlpqb ← dlpqb .A 177777B;
RETURN END.
(delul) %This routine turns any bug marks off.%
PROCEDURE;
dllul ←+ -40000B;
cdulpt ←+ -1;
RETURN END.
(uloff) %This PROCEDURE removes all of the marks (currently circles)
that provide feedback of bug selects on the screen.%

```

```

PROCEDURE;
    cdulpt ← $dlbul;
    dllul ← (dllul .A 37777B) .V 100000B;
    RETURN END.
(sbmfb) %The character that tracks the mouse position is set by this
routine (Set Bug Mark FeedBack Character). The PROCEDURE is called
with the character in the A.%
PROCEDURE (cdtemp);
    dlxbug ← (dlxbug .A 7600777B) .V LSH $12(cdtemp,0);
    RETURN END.
(cflatc) %This routine is called with an A-string address in the A,
and an integer N in the B. It moves the string to the Nth position
in the command feedback line.%
PROCEDURE(cflfrm,cflptr);
    % takes the a string (addr on a) and puts into the cfl in
    position%
    %specified by the b%
    IF 9 ≤ cflptr DO-SINGLE !err $6;
    cfltoo ← $dlbcfl + cfltbl/cflptr/ .A 7777B;
    %cfltoo is the first unused location in the cfl%
    IF cdabfl THEN BEGIN
        [cfltoo]+$2[cflfrm/ .A 77600000B .V 77577B;
        cflbnd+cfltoo END
    ELSE cflbnd ← asrbuf(cflfrm,cfltoo);
    %cflbnd is the last loc used%
    IF [cflbnd/ .A 177B = 177B THEN [cflbnd/ + [cflbnd/ .A 77777400B
    ELSE BEGIN
        BUMP cflbnd;
        [cflbnd/ ← 177777B END ;
    BUMP cflfrm;
    $1/$cfltbl + cflptr/ ← $0[.XR/ + LSH $12(IF cdabfl THEN 2
    ELSE [cflfrm/ +2,0) .V (cflbnd - cfltoo+1);
    dllcfl ← dllcfl .A 37777B .V .LSH $14(cflbnd - $dlbcfl +3,0)1
    RETURN END.
(ltlg) %Several characters is the upper left corner of the screen
provide feedback of the current view specs. Calling this routine
will display those characters in a large character size.%
PROCEDURE;
    dlxlt ← dlxlt .A 77776000B .V 1722B;
    RETURN END.
(ltsm) %Calling this routine will set the display of the view spec
feedback to a small character size.%
PROCEDURE;
    dlbrlv ← 37700000B;
    dlxlt ← dlxlt .A 77776000B .V 1116B;
    RETURN END.
%a-string stuff%
(cvsno) %An A-string containing digits can be converted to a binary
integer by calling CVSNO with the A-string address in the A. The
conversion is done to the base 10, and all characters are assumed to
be digits, and are not checked. The A-string is not changed, and the
integer is RETURNed in the A. If the first character is a minus
sign, the number will be converted to a negative number correctly.%

```

```

PROCEDURE(cvsno1); %convert a-string to integer%
  cvsno3,cvsno2,cvsno4 ← 0;
  IF ldchr(cvsno1,0) = 15B THEN BUMP cvsno2,cvsno4;
  LOOP BEGIN
    cvsno3 ← cvsno3*10 + ldchr(cvsno1,cvsno2)-20B;
    IF cvsno2 ← cvsno2 + 1 > $1/cvsno1/ DO-SINGLE EXIT
  END;
  RETURN(IF cvsno4 THEN
    -cvsno3 ELSE cvsno3)
END.
(asrnam) %An A-string is displayed in the name area by calling
ASRNAM with the address of the A-string in the A. The routine
converts the case of letters in the string for the display, by moving
the A-string character by character (and converting) to an A-string
named NUMREG. It THEN calls ASRBUF to move NUMREG to the name
register display buffer.%
PROCEDURE (astnml);
  BUMP astnml;
  IF /astnml/ > d1bnmx THEN astnm4 ← d1bnmx
  ELSE astnm4 ← .AR;
  numreg ← 60;
  astnm2 ← 0;
  numrg1 ← -1;
  numreg ← 60;
  WHILE astnm2 <= astnm4 DO BEGIN
    astnm3 ← ldchr(astnml -1,astnm2);
    BUMP astnm2;
    apchr( astnm3,, $numreg) END;
  asrbuf($numreg,$d1bnam);
  .AR ← .AR - $d1bnam +3;
  d1lnam ← LSH $14(,0) .V $d1pnam;
  RETURN END.
(bkachr) %Given the address of an A-string in the A, this routine
wil subtract one from the length, and THEN RETURN the last character
of the string in the A.%
PROCEDURE; %(a-string address in a)%
  IF $1/.AR/ > -1 THEN $1/.XR/ ←+ -1
  ELSE RETURN(0);
  IF carpos > 0 THEN carpos ← +-1; %carrage position for todas%
  RETURN(ldchr(.XR, $1/.XR)) ) END.
(numdpy) %This routine is used to convert a binary integer to an ASCII
number. The integer is provided in the A, and the routine RETURNS with
the number in the A, in ASCII. This PROCEDURE is used primarily for
displaying the view specs. Hence "all" is RETURNed IF a number less
than zero or greater than 100 is given in the A. The ASCII number is
left justified with blanks added at the END.%
PROCEDURE (numd1);
  IF numd1 < 0 OR numd1 = 100 THEN RETURN(10226054B);
  numd3 ← 0;
  numd4 ← 3;
  WHILE numd4 DO BEGIN
    numd4 ←+ -1;
    numd1 ← numd1/10;

```

```

        .AR ← .BR + 20B;
        .AR ← RSH $8(,numd3);
        numd3 ← .BR;
        IF numd1 = 0 THEN RETURN(numd3) END;
RETURN (numd3) END.
(xintr9) %This routine carries out all the details of finishing up when
an INPUT QED BRANCH command is done. It is activated by
pseudo-interrupt 209. It stops the fork that listens for rubouts WHILE
the input is done, stops the input fork and relabels out the PASS1 code,
closes the input file, removes the message, and recreates the display.
It does not RETURN from the interrupt but goes to previous state.%
PROCEDURE;
        .AR ← $outfpt;
        IF .NEG outfpt/6/ DO-SINGLE BRS $32();
        BRS $32($inpfpt);
        BRS $121(tpass1);
        %delete when change data page% tcalc ← toutov;
        inpfpt/4/ ← inpr11;
        inpfpt/5/ ← inpr12;
        inpfpt ← $inptfl;
        paslfg+inptfg ← 0;
        inptbo ← inptbi;
        BRS $9($inpfpt .V 66000000B);
        BRS $78(intmsk);
        BRS $20(outfn);
        dismes(,0);
        .AR ← -1;
        lany txt;
        GOTO gps END.
(cptstr) %The function of this routine is to copy a T-string to an
A-string. The calling arguments are the address of a T-pointer in the A
and another in the B, and an A-string address in the X. It copies the
T-string determined by the two T-pointers into the A-string, after
setting the A-string to null. The two T-pointers must point to the same
statement, or RERROR is called. An external flag is associated with
this routine, named CPTSRC. If it is true, the string is forced to
upper case. The flag is always reset before the routine RETURNS. Note:
IF the T-pointers point to the same character, one character will be
moved into the A-string. The END characters of the T-string are
included in the resultant A-string.%
PROCEDURE(cptstr1,cptstr2,cptstr3);
        %copy t-str to a-str%
        IF [cptstr1] # [cptstr2] DO-SINGLE rerror();
        swork ← [cptstr1];
        BUMP cptstr1;
        swork1 ← [cptstr1];
        BUMP cptstr2;
        $1/cptstr3/ ← -1;
        %null string%
        cptstr2 ← [cptstr2]+1;
        .AR ← fechcl(1,,$swork);
        BRM $2[.XR];
        cptstr1 ← .AR;

```

```

WHILE .AR # endchr and swork1 <= cptsr2 DO BEGIN
  IF cptsr3 THEN BEGIN
    IF cptsr1 IN (100B,132B) THEN cptsr1 ← .AR-40B
    END;
    apchr(cptsr1,,cptsr3);
    .XR ← $swork;
    BRM $2[.XR];
    cptsr1 ← .AR END;
  cptsr3 ← 0;
  RETURN END.
(chkfrz) %This routine calls RERROR IF any core blocks are frozen.
That is, the FRZCPT (frozen core page table) must contain -1 in all
positions.%
PROCEDURE;
  FOR frzrfl FROM 0 INC 1 TO rfcibx DO IF frzcpt(frzrfl) # -1 DO-SINGLE
  rerror();
  RETURN END.
%pops%
(lec) %Load Entity Character. The entity character (SMEC) is set to
the character in the address field of the pop.%
POP 103B PROC; %load entity character %
  EAX [$0](,,0);
  smec←.XR;
  RETURN END.
(les) %Load Entity String. The entity string (SMES) is set to the
A-string address provided in the pop address field.%
POP 104B PROC; % load entity string %
  EAX [$0](,,0);
  smes←.XR;
  RETURN END.
(gset) %Group SET. The current group is set to the group number
given in the address of the pop. If the group has changed, the
entity character and string are set to the initial values for that
group. The initial values are in IGRPC and IGRPS.%
POP 110B PROC; %set group and entity %
  EAX [$0](,,0);
  IF .XR = curgrp DO-SINGLE RETURN;
  IF .AR > $groupx DO-SINGLE rerror();
  smec←igrpc[.XR];
  smes←igrps[.XR];
  curgrp←.XR;
  RETURN END.
(mlf1) %Move Literal to Feedback line, first position. This pop
sets the feedback line position counter (CFLPOS) to -1 and THEN
executes the code for a MLF pop.%
POP 111B PROC;
  %move literal to feedback - first pos %
  cflpos←-1 END.
(mlf) %Move Literal to Feedback line. This pop increments the
feedback line position counter by one and THEN moves the A-string,
whose address is in the address field of the pop, to the feedback
line by calling CFLATC.%
POP 111B PROC;

```

```

    %move literal to feedback - next%
    BUMP clpos END.
(mlfr) %Move Literal to Feedback line and Replace last. This pop is
the same as MLF except that it does not increment the position
counter before moving the string.%
POP 142B PROCEDURE;
    %move literal -over last%
    .BR+cflpos;
    EAX [$0];
    RETURN(cflatc(.XR .A 37777B)) END.
(rep) %REpeat a case statement. The REP pop removes the overlay
address from the circular case save array, resets the pointer (CASV)
to the previous entry, and branches to the saved location using
routine OVLGO. The address of the pop contains an integer which
indicates how many cases to back up over. A REP 0 is the normal
edition, which RETURNS to the instruction following the last CSE pop,
and does not change the case array pointer. A REP 1 backs up the
pointer by one. The big problem with the current case business is
that the pointer is not moved back when a repeat is NOT done. Hence
REP 1 or REP 2 will not RETURN to the correct location IF someone
branched out of an inner case.
POP 113B PROC; %repeat in case statement%
    smcreg←smbreg;
    .XR←casv;
    IF [$0] > 0 THEN BEGIN
        casv←(-(.AR-casv)) .A $ncasv1;
        .XR←.AR END;
    IF NOT .NEG casvb[.XR] DO-SINGLE BRR casvb[.XR];
    ovlgo(casvb[.XR]) END.
(dfs) %DeFINE State. The overlay address is saved in STATE, and
the current group is set to the integer given in the address of the
pop. A ZAP is THEN executed.%
POP 144B PROC; % define a state %
    state←abtgt←ovladr($0);
    EAX [$0](,,0);
    curgrp←.XR;
    GOTO zap END.
(cta) POP 147B PROCEDURE;
    IF [$0] THEN BUMP cptsr;
    RETURN (cptstr($spt1,$spt2,$sarxn)) END.
(anypop) POP 177B PROC;
EXTERNAL creg,breg,sreg,sba,sbp,ta,mrf,mrfr,mrkl,mrk,kin,
kct,pbm,cse,spcb,spcr,spcrno,spcrp,spcrna,mrs,bkc,dpn,spca,
spcf,spcn,cct,cct,zero,zap,zapsp,feedlt;% %
GOTO [$0];
(creg): %append the character in C-REG to string in A-reg. The
character in SMCREG is APPENDED to the register whose number is in
SMAREG.%
    regadr(smareg);
    RETURN (apchr(smcreg));
(breg): % append char in b to reg no. in a %
    regadr(smareg);
    RETURN ( apchr(smbreg));

```

```

(sreg): % append sar to reg no. in a %
        regadr(smareg);
        RETURN( apsr($sarxn));
(sba): % set bug to arrow %
        RETURN (sbmfbcrmdcr);
(sbp): % set bug to plus %
        RETURN (sbmfbcrmdcr);
(ta): %display in text area%
        IF NOT litrf AND stnxn <= stnl+1 THEN gt2lit();
        IF smbreg <= 132B or .AR = 151B or .AR = 155B
            THEN apchr(smbreg,,litlc);
        IF NOT paslfg THEN aplit(smbreg);
        RETURN;
(mrf): % move register to feedback line %
        BUMP cflpos;
(mrfr): % move register to feedback (same spot) %
        regadr(smareg);
        .BR+cflpos;
        RETURN (cflatc(.XR .A 37777B));
(mrkl): %MaRK feedback line for up arrow in first position. This
calls CFLARW to display an up arrow under the first position of the
feedback line.%
        RETURN(cflarw(0,0));
(mrk): %MaRK feedback line for up arrow. This pop has CFLARW
display an up arrow under the NEXT position in the feedback line.%
        RETURN(cflarw(0,cflpos+1));
(kin): %INput a character. The routine INPTM is called to read a
character. The character is stored in SMCREG. QMOFF is also
called, to make sure a "?" is not being displayed.%
        smcreg<inptm();
        RETURN(qmoff());
(kct): %Character Test. This compares the character in SMBREG with
the input charcter in SMCREG and sets FLAG IF they are equal,
resets it IF not. The input character is forced to upper case
before the compare.%
        .BR+1;
        IF smcreg IN (100B,132B) DO-SINGLE .AR<.AR-40;
        IF .AR # smbreg DO-SINGLE .BR+0;
        flag<.BR;
        RETURN;
(pbm): %Process Bug Mark. This pop resets the flag IF the input
character (in SMCREG) is not a command accept. Otherwise it sets
the flag, reads the bug coordinates and stores them in BUGREG.%
        flag+0;
        IF smcreg # $ascca DO-SINGLE RETURN;
        BUMP flag;
        readbm();
        STP bugreg;
        RETURN;
(cse): %CaSE statement. This pop causes the overlay address to be
"pushed" onto the case save stack (which is curcular). The flag
is set to false.%
        casv+(casv+1) .A $ncasv1;

```



```

    csel←.AR;
    casvb/csel/←ovladr($0);
    flag ← 0;
    RETURN;
(spcb): %SPeC a Bug mark.  If BUGREG does not already contain a
T-pointer (a -1 in the first word) THEN either PBUG or TREBUG in
TREE is called to PROCess the bugmark into a T-pointer.  The
T-ponter is pushed onto the spec stack.%
    LDP bugreg;
    IF .AR = -1 THEN BEGIN
        IF NOT cdtref THEN pbug()
        ELSE BEGIN
            txtloc ← ovladr($0);
            EXECUTE rfbp2;
            EXECUTE cdsply;
            EXECUTE disbuf;
            trebug(,bugreg/l/);
            pushsp();
            ovlgo(txtloc) END END;
    RETURN (pushsp());
(spcr): % SPeC a Register.  This is a rather complicated pop that
converts any string register to a T-pointer or number for the spec
stack.  Depending on which register is indicated by SMAREG,
    (STNO) It converts the string (a statement number) to a PSID by
calling FECHUX.
    (NUMN) It converts the string to an integer by calling CVSTNO,
and pushes (0,number) on the spek stack.
Otherwise, IF the first character of the string is a digit, it
converts it as a statement number as above, or ELSE it treats
it as a name and finds the corresponding PSID by calling
LKNAMH. %
    smtemp ← $0;
    EXECUTE rfbp2;
    $0 ← smtemp;
    IF smareg = $stnon DO-SINGLE GOTO spcrno;
    IF .AR # $numnn DO-SINGLE GOTO spcrna;
    cvsno($numn);
    COPY (AB,A);
    GOTO spcrp;
    (spcrno):
    regadr();
    (spcrnl):
    fechux(.XR);
    EXECUTE abort $2;
    (spcrp):
    RETURN(pushsp());
    (spcrna):
    regadr();
    IF NOT .NEG $1/.XR/ THEN BEGIN
        IF LRSH $16($2/.XR/) IN (17B,31B/ DO-SINGLE GOTO spcrnl END;
    .AR ← 0;
    IF .NEG $1/.XR/ DO-SINGLE GOTO spcrp;
    hash(.XR);

```

```

lknamh();
labort $3;
GOTO spcrp;
(mrs): %Move Register to String area. This used CPYSR to move the
A-string indicated by SMAREG to SAR.%
    regadr(smareg);
    RETURN (cpysr(.XR,, $sarxn));
(bkc): %Backspace Character. The A-string indicated by SMAREG is
backed up one character via routine bkachr.%
    regadr(smareg);
    RETURN(bkachr(.XR));
(dpn): %Display in Name area. The A-string indicated by SMAREG is
displayed in the name area.%
    regadr(smareg);
    RETURN(asrnam(.XR));
(spcn): %This gets the hash of the A-string indicated by SMAREG
and puts (-1,hash) on the stack.%
    regadr(smareg);
    RETURN(pushsp(-1,hash(.XR)));
(cct): %Character Class test. This pop does the character class
testing for the CASE statement. The letter class includes both
upper case and lower case letters. The classes are:
    (4) Letter
    (5) Digit
    (6) Any character.%
    .BR ← 1;
    IF smareg = 6 THEN NULL
    ELSE IF .AR = 5 THEN BEGIN
        IF smbreg NOT IN (17B,31B) DO-SINGLE .BR←0 END
    ELSE IF .AR = 4 THEN BEGIN
        IF smbreg NOT IN (40B,72B) THEN BEGIN
            IF .AR NOT IN (100B,132B) DO-SINGLE .BR←0 END END
    ELSE rerror();
    flag ← .BR;
    RETURN;
(zero): %ZERO the spec stack. This does sets the stack pointer to
the bottom of the spec stack.%
    spsk←spskl;
    RETURN;
(zap): %ZAP everything. This includes:
    Collapse the general stack.
    Release the long literal register, IF it exists.
    Reset the spec stack.
    Relabel out the two random file block pages (RFBP1 and RFBP2).
    Call CHKFRZ, LTSM, RESDPY, and ULOFF. %
    stack←stackl;
(zapsp): %ZAP the SPec stack. This is exactly the same as ZAP,
except that the general stack is not changed.%
    spsk←spskl;
    rellit();
    smtemp ← $0;
    EXECUTE nrfbp2;
    $0 ← smtemp;

```

```

chkfrz();
IF todas DO-SINGLE RETURN; %todas%
RETURN (ltsm(resdpy(uloff())));
(feedlt): %FEED LT specs. This pop requires that PRMSPC be in,
since it calls SETLT. However, it does not check. The characters
in the A-string indicated by SMAREG are fed to SETLT one at a
time.%
    regadr(smareg);
    fedlt3 ← .XR;
    fedlt2 ← $1/.XR/;
    FOR fedlt1 FROM 0 INC 1 TO fedlt2 DO BEGIN
        ldchr(fedlt3,fedlt1);
        setlt() END;
    RETURN END.
(tch) %Test character. Sets SMBREG to the character looking for, THEN
goes to kct.%
POP146B PROCEDURE;
    EAX /$O/ (,,0);
    smbreg ← .XR;
    GOTO kct END.

```

finish

```
:MNCTRL, 09/17/69 1301:36 CHI ; .HED="MAIN CONTROL OVERLAY"; .SCR=1;
.PLO=1; .MCH=75; .RTJ=0; .DSN=1; .LSP=0; .MIN=70; .INS=3;
```

```
file mnctrl(orgmct)
(+wait) bp af . case
  (cd) an goto [s]
  (ca) an return
  endcase repeat 0(.)
(+caqm) case
  (ca) goto [s]
  (cd) goto [s]
  endcase qm repeat 1(.)
(+dmain) display() goto main
(+setdum) +gdmys,txtedt/pl/ goto dmain
(+cmdrst) s**zz,spec dsp(<command reset +)
  cdflag(incse,0) goto main
(+mdispec) mdreset (mdnxt): . case
  (b) dsp (boldface ↑) boldface goto mdnxt
  (c) dsp (capital ↑) upper-case goto mdnxt
  (f) dsp (flicker ↑) flicker goto mdnxt
  (i) dsp (italics ↑) italics goto mdnxt
  (l) dsp (lower ↑) lower-case goto mdnxt
  (n) dsp (noboldface ↑) no-boldface goto mdnxt
  (r) dsp (roman ↑) roman goto mdnxt
  (s) dsp (solid ↑) solid goto mdnxt
  (u) dsp (underline ↑) underline goto mdnxt
  (w) dsp (nounderline ↑) no-underline goto mdnxt
  (cd) mdreset goto cmdrst
  (ca) dsp (+) return
  endcase repeat 0(.)
(+main) bp zap . goto wc
(+wc) bp an zap case
  (a) s**xa,spec dsp(+< append statement) -qas,stredt
  (b) S**bs,spec dsp(+< break statement) -qbs,stredt
  (c) [edit] dsp(<copy tes*) . case
    (c) s**cc,edit dsp(+ < copy character) e**c,character
      +bug2spec,prmspc -qcc,txtedt
    (d) -qcd,vctedt
    (w) s**cw,edit dsp(+ < copy word) e**w,word
      +bug2spec,prmspc -qcw,txtedt
    (n) s**cn,edit dsp(+ < copy number) e**n,number
      +bug2spec,prmspc -qcn,txtedt
    (v) s**cv,edit dsp(+ < copy visible) e**v,visible
      +bug2spec,prmspc -qcv,txtedt
    (i) s**ci,edit dsp(+ < copy invisible) e**i,invisible
      +bug2spec,prmspc -qci,txtedt
    (t) s**ct,edit dsp(+ < copy text) e**t,text
      +bug3spec,prmspc -qct,txtedt
    (s) s**cs,edit dsp(+ < copy statement) e**s,statement -qcs,stredt
    (b) s**cb,edit dsp(+ < copy branch) e**b,branch -qcb,stredt
    (p) s**cp,edit dsp(+ < copy plex) e**p,plex -qcp,stredt
    (g) s**cg,edit dsp(+ < copy group) e**g,group -qcg,stredt
  (ca) repeat 0(e*)
  endcase goto caqm
```

MAIN CONTROL OVERLAY

```

(d) /edit/ dsp(<delete ↑es*) . case
  (c) s*=dc,edit dsp(+ < delete character) e*=c,character
      +buglspec,prmspc -qdc,txtedt
  (d) -qdd,vctedt
  (w) s*=dw,edit dsp(+ < delete word) e*=w,word
      +buglspec,prmspc -qdw,txtedt
  (n) s*=dn,edit dsp(+ < delete number) e*=n,number
+buglspec,prmspc -qdn,txtedt
  (v) s*=dv,edit dsp(+ < delete visible) e*=v,visible
      +buglspec,prmspc -qdv,txtedt
  (i) s*=di,edit dsp(+ < delete invisible) e*=i,invisible
      +buglspec,prmspc -qdi,txtedt
  (t) s*=dt,edit dsp(+ < delete text) e*=t,text
      +bug2spec,prmspc -qdt,txtedt
  (s) s*=ds,edit dsp(+ < delete statement) e*=s,statement
      -qds,stredt
  (b) s*=db,edit dsp(+ < delete branch) e*=b,branch -qdb,stredt
  (p) s*=dp,edit dsp(+ < delete plex) e*=p,plex -qdp,stredt
  (g) s*=dg,edit dsp(+ < delete group) e*=g,group -qdg,stredt
  (ca) repeat O(e*)
endcase goto caqm
(e) dsp (<execute ↑) . case
  (a) -eap,ioctl
  (c) dsp(<content analyzer+) +buglspec,prmspc
      getrf -cacmpg,cacmpl
  (d) dsp(<declare file ownership+) +wait
      !" lda cuno; sta funo" goto [s]
  (f) -fcln,clnup
  (l) goto qaq,ioctl
  (o) dsp(oops) +wait;
  (p) dsp(+compactor) +wait af getrf
      goto cmptrl,txtedt
  (s) dsp(+status) getrf +sttus0,auxcod
      +wait display() goto[s]
  (v) -diddx,diddl
  (x) -calcm,calc
endcase goto caqm
(f) /special/ dsp(< freeze ↑statement) . case
  (s) s**kf,spec dsp(+ < freeze statement) -qkf,auxcod
  (r) s**kr,spec dsp(+ < release statement) -qkr,strmnp
  (a) dsp(+< release all) +wait -qka,auxcod
  (ca) repeat O(s)
endcase goto caqm
(i) /edit/ dsp(<insert ↑es*) . case
  (c) s**ic,edit dsp(+ < insert character) e*=c,character
      call getlit lit*=0 +bugltspec,prmspc -qic,txtedt
  (w) s**iw,edit dsp(+ < insert word) e*=w,word
      call getlit lit*=sp +bugltspec,prmspc -qiw,txtedt
  (n) s**in,edit dsp(+ < insert number) e*=n,number
      call getlit lit*=sp +insacc,calc +buglspec,prmspc -qiw,txtedt
  (v) s**iv,edit dsp(+ < insert visible) e*=v,visible

```

MAIN CONTROL OVERLAY

```

    call getlit lit*=sp +bugltspec,prmspc -qiv,txtedt
(i)  s*=ii,edit dsp(+ < insert invisible) e*=i,invisible
    call getlit lit*=0 +bugltspec,prmspc -qii,txtedt
(t)  s*=it,edit dsp(+ < insert text) e*=t,text
    call getlit lit*=0 +bugltspec,prmspc -qit,txtedt
(s)  s*=is,edit dsp(+ < insert statement) e*=s,statement
-qis,prmspc
(b)  s*=ib,edit dsp(+ < insert branch) e*=b,branch -qis,prmspc
(p)  s*=ip,edit dsp(+ < insert plex) e*=p,plex -qis,prmspc
(g)  s*=ig,edit dsp(+ < insert group) e*=g,group -qis,prmspc
(q)  s*=iq,edit dsp(+<insert qed branch) e*=b,branch
    +specit,prmspc goto qiq,ioc1
(ca) repeat 0(e*)
endcase goto caqm
(j)  -jmpagn,vctedt
(k)  -kmain,keywd
(l)  goto qlq,ioc1
(m)  /edit/ dsp(<move ↑es*) . case
(c)  s*=mc,edit dsp(+ < move character) e*=c,character
    +bug2spec,prmspc -qmc,txtedt
(d)  -qmd,vctedt
(w)  s*=mw,edit dsp(+ < move word) e*=w,word
    +bug2spec,prmspc -qmw,txtedt
(n)  s*=mn,edit dsp(+ < move number) e*=n,number
    +bug2spec,prmspc -qmn,txtedt
(v)  s*=mv,edit dsp(+ < move visible) e*=v,visible
    +bug2spec,prmspc -qmv,txtedt
(i)  s*=mi,edit dsp(+ < move invisible) e*=i,invisible
    +bug2spec,prmspc -qmi,txtedt
(t)  s*=mt,edit dsp(+ < move text) e*=t,text
    +bug3spec,prmspc -qmt,txtedt
(s)  s*=ms,edit dsp(+ < move statement) e*=s,statement
    -qms,stredt
(b)  s*=mb,edit dsp(+ < move branch) e*=b,branch -qmb,stredt
(p)  s*=mp,edit dsp(+ < move plex) e*=p,plex -qmp,stredt
(g)  s*=mg,edit dsp(+ < move group) e*=g,group -qmg,stredt
(ca) repeat 0(e*)
endcase goto caqm
(o)  goto qoq,ioc1
(p)  /special/ dsp(<pointer ↑fix) . case
(f)  s*=pf,spec dsp(+<pointer fix) call getlit lit*=0
    +bugltspec,prmspc -qpf,txtedt
(m)  s*=pm,spec dsp(+ < pointer move) call getlit lit*=0
    +bugltspec,prmspc +qpml,auxcod -qpf,txtedt
(l)  dsp(+ ... list show) +qpls1,auxcod -qpls2,prmspc
(r)  dsp(... release ↑all) . case
(a)  dsp(+ +wait delptr(all) display() goto /s/
(t)  s*=prt,spec dsp(+<pointer release text)
    +bug2spec,prmspc -qprt,txtedt
(ca) repeat 0(a)
endcase goto caqm

```

MAIN CONTROL OVERLAY

```

(ca) repeat 0(f)
endcase goto caqm
(q) dsp(<view set↑) +ltspec,prmspc display() goto [s]
(r) /edit/ dsp(<replace ↑es*) . case
(c) s**=rc,edit dsp(+<replace character) e**=c,character
    call getlit lit*=0 +rplbug,prmspc -qrc,txtedt
(w) s**=rw,edit dsp(+<replace word) e**=w,word
    call getlit lit*=0 +rplbug,prmspc -qrw,txtedt
(n) s**=rn,edit dsp(+ < replace number) e**=n,number
    call getlit lit*=0 +rplbug,prmspc -qrn,txtedt
(v) s**=rv,edit dsp(+<replace visible) e**=v,visible
    call getlit lit*=0 +rplbug,prmspc -qrv,txtedt
(i) s**=ri,edit dsp(+<replace invisible) e**=i,invisible
    call getlit lit*=0 +rplbug,prmspc -qri,txtedt
(t) s**=rt,edit dsp(+<replace text) e**=t,text
    call getlit lit*=0 +rplbug,prmspc -qrt,txtedt
(s) s**=rs,edit dsp(+<replace statement) e**=s,statement
    -qrs,stredt
(b) s**=rb,edit dsp(+<replace branch) e**=b,branch -qrb,prmspc
(p) s**=rp,edit dsp(+<replace plex) e**=p,plex -qrp,prmspc
(g) s**=rg,edit dsp(+<replace group) e**=g,group -qrg,prmspc
(ca) repeat 0(e*)
endcase goto caqm
(s) dsp(< set ↑ ) . case
(c) s**=sc,spec dsp(< set character ↑)
    +mdispec (scl): zap +buglspec,prmspc +qsc,txtedt goto scl
(w) s**=sw,spec dsp(< set word ↑)
    +mdispec (swl): zap +buglspec,prmspc +qsw,txtedt goto swl
(n) s**=sn,spec dsp(< set number ↑)
    +mdispec (snl): zap +buglspec,prmspc +qsn,txtedt goto snl
(v) s**=sv,spec dsp(< set visible ↑)
    +mdispec (svl): zap +buglspec,prmspc +qsv,txtedt goto svl
(i) s**=si,spec dsp(< set invisible ↑)
    +mdispec (sil): zap +buglspec,prmspc +qsi,txtedt goto sil
(t) s**=st,spec dsp(< set text ↑)
    +mdispec (stl): zap +bug2spec,prmspc +qst,txtedt goto stl
(s) s**=ss,spec dsp(< set statement ↑)
    +mdispec (ssl): zap +buglspec,prmspc +qss,txtedt goto ssl
(ca) repeat 0(e*)
endcase goto caqm
(v) -vmain,vctrl
(lw) -scmain,vctedt
(x) dsp(< substitute ↑ branch) . case
%(e) s**=xe,edit dsp( +< edit statement) +specit,prmspc
    +xedit,inpfbk +stlit,txtedt[bl] goto [s]%
(s) s**=xs,spec dsp( +< substitute statement) -qxs,stredt
(b) s**=xb,spec dsp( +< substitute branch) -qxb,stredt
(g) s**=xg,spec dsp( +< substitute group) -qyg,stredt
(p) s**=xp,spec dsp( +< substitute plex) -qxp,stredt
(ca) repeat0(b)
%how do we do the wait?

```


MAIN CONTROL OVERLAY

```
    also how do we get th count displayed? %
    endcase goto caqm
    (sp) repeat 0(.)
    endcase goto caqm
words( text, word, number, visible, invisible, branch, plex, all,
group, delete, insert, replace, move, copy,
show, dont, list, file,
pointer, fix, view, set, reset,
command, release, break, append,
boldface, capital, flicker, italics, lower, noboldface,
roman, solid, underline, nounderline,
content, analyzer, execute, qed, freeze, substitute,
compactor, oops, status, declare,
ownership
)
end of mnctrl
```

:PRMSPC, 11/04/69 1139:17 WHP ; PARAMETER SPECIFICATION .SCR=1; .PLO=1;
.MCH=75; .RTJ=0; .DSN=1; .LSP=0; .MIN=70; .INS=3;
%.HED="PRMSPC, SPL CODE"; .RES;

PRMSPC, SPL CODE

%

```

file prmspc(orgpsp)
%odds and ENDS%
  (+recre) display() GOTO [s]
  (+nuview) (dl) display(dl) GOTO [s]
  (+setvsp) feedlt(num*) RETURN
  (+specbug) spec(bug) RETURN
  (+waitp) bp af . case
    (cd) an GOTO [s]
    (ca) an RETURN
    (bc) CALL softcd ba repeatl(.)
  ENDcase repeat0(.)
  (+bcp) case
    (bc) CALL softcd repeat 2(.)
  ENDcase GOTO wc,mnctrl
  (+sugt12) CALL gtlit2 RETURN %called by substitute in strmp%
%bug acceptors%
  (+bug1spec) ba . case
    (bug) spec(bug) bp +waitp RETURN
  ENDcase GOTO bcp
  (+bug2spec) ba . case
    (bug) spec(bug) +bug1spec RETURN
  ENDcase GOTO bcp
  (+bug3spec) ba . case
    (bug) spec(bug) +bug2spec RETURN
  ENDcase GOTO bcp
  (+bugltspec) ba . case
    (bug) spec(bug) bp +atext an RETURN
  ENDcase GOTO bcp
  (+bugltspec) ba . (bgltspec): case
    (bug) bp spec(bug) +ltspec RETURN
  ENDcase GOTO bcp
  (+rplbug) ba . case
    (bug) spec(bug) . case
      (bug) spec(bug) +waitp cdflag(rplb,1) RETURN
      (bc) CALL softcd repeatl(.)
    ENDcase af bp +atext1 an cdflag(rplb,0) RETURN
  ENDcase GOTO wc,mnctrl
  (+rpltbug) ba . case
    (bug) spec(bug) . case
      (bug) spec(bug) . case
        (bug) spec(bug) +waitp cdflag(rplb,1) RETURN
      ENDcase GOTO bcp
      (bc) CALL softcd repeatl(.)
    ENDcase af bp +atext1 an cdflag(rplb,0) RETURN
  ENDcase GOTO bcp
  ENDcase GOTO bcp
%number, name, literal, and lt acceptors%
  (+atext) af . (atext1): case
    (cd) an GOTO [s]

```

PRMSPC, SPL CODE

```

(ca) an RETURN
(c.) an RETURN
(bc) back(lit*) dt(c*) repeat 0(.)
(bw) backw(lit*) dt(c*) repeat 0(.)
ENDcase dt(c*) repeat 0(.)
(+integer) af .
(integl): !"integ1 ext" num*=0 dn(num*) case
  (ca) an RETURN
  ($d) &num* dn(num*) repeat 0(.)
  (cd) an GOTO [s/
  (bc) back(num*) dn(num*) repeat 0(.)
  (bw) num*=0 dn(num*) repeat 0(.)
  ENDcase qm repeat 0(.)
(+ltspec) ltl af cdflag(ltflag,l) . case
  (cd) lts an GOTO [s/
  (ca) lts an RETURN
  (bc) CALL softcd an ba repeat 1(.)
  ENDcase !" lda smcreg; sbrm setlt" repeat 0(.)
(+statno) stno*=0 dn(stno*) af case
  (cd) an GOTO [s/
  (ca) an RETURN
  (bc) back(stno*) dn(stno*) repeat 0(.)
  (bw) stno*=0 dn(stno*) repeat 0(.)
  ENDcase &stno* dn(stno*) repeat 0(.)
(+statnam) stn*=0 dn(stn*) af case
  (cd) an GOTO [s/
  (ca) an RETURN
  (bc) back(stn*) dn(stn*) repeat 0(.)
  (bw) stn*=0 dn(stn*) repeat 0(.)
  ENDcase
  !" lda smcreg; skg =132b; skg =100b; bru **+2; sub =40b; sta
  smcreg"
  &stn* dn(stn*) repeat 0(.)
%statement spec%
(+specn) ba . case
  (bug) spec(bug) bp +wdr2,txtedt/bl,pl-4/ stn*=(p1 p2)↑
  dn(stn*) spec() spec(stn*) RETURN
  ENDcase GOTO spectl
(+jpspcn) ba . case
  (bug) spec(bug) +wdr2,txtedt/bl,pl-4/ stn*=(p1 p2)↑
  dn(stn*) spec() spec(stn*) bp CALL jspush GOTO ltspec
  ENDcase GOTO jpsptl
(+jpsplt) ba .
  (jpsptl): case
  (bug) spec(bug) bp CALL jspush GOTO ltspec
  ($d) +statno spec(stno*) stno*=0 bp CALL jspush GOTO ltspec
  (sp) af . +statnam spec(stn*) stn*=0 bp CALL jspush GOTO ltspec
  ENDcase GOTO newjmp,vctedt
(+kgtwd) ba . case
  (bug) spec(bug) +wdr2,txtedt/bl,pl-4/
  stn*=(p1 p2)↑ dn(stn*) spec() spec(stn*) RETURN

```

PRMSPC, SPL CODE

```

(sp) af . +statnam spec(stn*) RETURN
($1) GOTO wc,mnctrl
(cd) GOTO wc,mnctrl
ENDcase qm repeat 0(.)
(+specit) ba .
(spect1): case
  (bug) spec(bug) bp RETURN
  ($d) +statno spec(stno*) stno*=0 bp RETURN
  (sp) af . +statnam spec(stn*) stn*=0 bp RETURN
  (bc) CALL softcd repeat 1(.)
  ENDcase GOTO wc,mnctrl
(+gt2adj) +specit +specit +adj/rbl/ RETURN
(+gt3adj) +specit +specit +specit
+adj/rbl/ RETURN
%insert and replace control%
(+qib) +adj/rbl/ +atexti +recre;
(+qis) +specit +adj/rbl/ +atexti +recre;
(+qrb) +specit +setgrp, strmnp/rbl/ +atextr +recre;
(+qrg) +specit +specit +grptst, strmnp/rbl,rb2/ +atextr +recre;
(+qrp) +specit +plxset, strmnp/rbl/ +atextr +recre;
(+qpls2) . case
  (ca) display() GOTO /s/
  ENDcase display() GOTO wc,mnctrl
%atexti and r adj%
(+atexti) dummy dl
  (atexla): CALL getlit lit*=0 +atext
  +inslit, strmnp
  CALL rellit
  case
    (ca) RETURN
  ENDcase %c.% display() +adj/@adjpos/ GOTO atexla
(+atextr) dummy dl,d2,d3
  CALL getlit lit*=0 +atext
  +rpllit, strmnp
  CALL rellit
  case
    (ca) RETURN
  ENDcase %c.% display() +adj/@adjpos/ +atexti RETURN
(+adj) (dl) dummy d2,d3 stn*=0
:s set adjdir; dl => adjpos;
IF not dl THEN reset adjdir; reset d2:
(adjla):
:s GOTO adjlc;
(adjlb):
IF adjpos THEN BEGIN
  set adjdir; dl => adjpos;
  IF neg d2 THEN BEGIN
    reset adjdir,d2;
    dec d2 END
  ELSE BEGIN
    d2 => d3; reset d2;

```

PRMSPC, SPL CODE

```
        WHILE d3 DO BEGIN
            IF adjpos THEN BEGIN
                +adjklg, strmnp;
                dec d3;
                BUMP d2 END
            ELSE reset d3 END
        END
    END;
    GOTO adjla :
    (adjlc): . case
        (u) :s BUMP d2: &stn* dn(stn*) GOTO adjlb
        (d) :s dec d2: &stn* dn(stn*) GOTO adjlb
        (sp) RETURN
        (ca) RETURN
        (cd) GOTO [s]
    ENDcase repeat0(.)
END of prmspc
```

PRMSPC, SPL CODE

%.HED="PRMSPC OVERLAY, MOL PROCEDURES"; .RES;

PRMSPC OVERLAY, MOL PROCEDURES

%

```

DECLARE POP sov=137B, sbc=107B %POP's in UTILITY%;
(setlt) PROCEDURE(setlt1);
  IF .AR = '. DO-SINGLE BUMP setlt3;
  IF .AR NOT > 17B DO-SINGLE RETURN;
  IF .AR <= 32B THEN BEGIN
    IF setlt3 THEN BEGIN
      setlt3+0;
      IF setlt1-20B NOT IN (0,10B/ DO-SINGLE RETURN;
      .BR+.AR; pushsp(0,.BR);
      !sov didd1; !sbc ltrstr END
    ELSE setlt2+setlt1-20B;
    RETURN END;
  IF .AR = 136B %command delete% THEN BEGIN
    IF cdrlev DO-SINGLE GOTO rstlev;
    RETURN END;
  IF .AR IN (100B,132B/ THEN GOTO [alpha[.AR-101B/]
  ELSE IF .AR IN (40B,'L/ THEN GOTO [capalpha[.AR-41B//];
(setlt1): setlt2+1; setlt3+0; RETURN;
(a): % 1+1-1 %
  cdlev+--setlt2;
  .AR + IF cdrlev THEN -99 ELSE 1;
  IF .AR > cdlev DO-SINGLE cdlev + .AR;
  CALL lnmdpy; GOTO setlt1;
(b): % 1+1+1 %
  IF 100 <= cdlev DO-SINGLE GOTO setlt1;
  cdlev++setlt2;
  CALL lnmdpy(); GOTO setlt1;
(c): % 1+all %
  cdlev+100;
  cdrlev+0;
  dlbrlv + 37700000B;
  dblev+10226054B; GOTO setlt1;
(d): % 1+1 %
  cdlev+1;
  cdrlev+0;
  dlbrlv + 37700000B;
  dblev+4200000B; GOTO setlt1;
(e): % 1=rel %
  IF NOT cdrlev THEN BEGIN
    cdrlev + cdlev;
    cdlev + 0;
    dlbrlv + 14405600B;
    CALL lnmdpy() END
  ELSE BEGIN
    (rstlev):
    cdlev + cdrlev;
    cdrlev + 0;
    dlbrlv + 37700000B;
    CALL lnmdpy() END; GOTO setlt1;
(f): GOTO setlt1; % for new view now %

```


PRMSPC OVERLAY, MOL PROCEDURES

```

(g): % branch only on %
    cdbrf+1;
    dlbscl+dlbscl .A 177777B .V 11600000B; GOTO setl1;
(h): % branch only off %
    cdbrf+0;
    dlbscl+dlbscl .A 177777B .V 12000000B; GOTO setl1;
(i): % content analyzer on %
    cdctaf+1;
    dlbscl+dlbscl .A 77600377B .V 24400B; GOTO setl1;
(j): % content analyzer off %
    cdctaf+0;
    dlbscl+dlbscl .A 77600377B .V 25000B; GOTO setl1;
(k): % trail on %
    cdtrlf+1;
    dlbscl+dlbscl .A 77777400B .V 53B; GOTO setl1;
(l): % trail off %
    cdtrlf+0;
    dlbscl+dlbscl .A 77777400B .V 54B; GOTO setl1;
(m): cdstnf+1; GOTO setl1; % location numbers on %
(n): cdstnf+0; GOTO setl1; % location numbers off %
(o): cdfrzf+1; GOTO setl1; % frozen on %
(p): cdfrzf+0; GOTO setl1; % frozen off %
(q): % t+t-1 %
    IF cdtrn = -1 THEN cdtrn ← 100;
    cdtrn+--setl2;
    IF cdtrn < 1 THEN cdtrn+1;
    dlbtrn+numdpy(cdtrn); GOTO setl1;
(r): % t+t+1 %
    IF cdtrn = -1 DO-SINGLE GOTO setl1;
    cdtrn++setl2;
    dlbtrn+numdpy(cdtrn); GOTO setl1;
(s): % t=all %
    cdtrn+-1;
    dlbtrn+10226054B; GOTO setl1;
(t): % t+1 %
    cdtrn+1;
    dlbtrn+4200000B; GOTO setl1;
(u): % markers on %
    cdptr+1;
    dlbsc2+dlbsc2 .A 177777B .V 15200000B; GOTO setl1;
(v): % markers off %
    cdptr+0;
    dlbsc2+dlbsc2 .A 177777B .V 15400000B; GOTO setl1;
(w): % l=t=all %
    cdtrn+-1;
    cdlev+100;
    dlbrlv+37700000B;
    cdrlev+0;
    dlbtrn,dlblev+10226054B; GOTO setl1;
(x): % l=t=1 %
    cdtrn+cdlev+1;

```

PRMSPC OVERLAY, MOL PROCEDURES

```

dlbrlv←37700000B;
cdrlev←0;
dlbtrn←dblev←4200000B; GOTO setl1;
(y): cdblnk←1; GOTO setl1; % blank line on %
(z): cdblnk←0; GOTO setl1; % blank line off %
(capa): cdindf←1; GOTO setl1; % indenting on %
(capb): cdindf←0; GOTO setl1; % indenting off %
(capc): cdnamf←1; GOTO setl1; % names on %
(capd): cdnamf←0; GOTO setl1; % names off %
(cape): cdclpf←1; GOTO setl1; % clip pictures %
(capf): cdclpf←0; GOTO setl1; % dont show IF too big to fit %
(capg): cdtref←1; GOTO setl1; % text format display %
(caph): cdtref←0; GOTO setl1; % tree format display %
(capi): cdkeyf←1; GOTO setl1; % keyword reordering %
(capj): cdkeyf←0; GOTO setl1; % normal order %
(capk): cdidtf←1; GOTO setl1; % initials, date, on %
(capl): cdidtf←0; GOTO setl1; % initials, date, off %
NULL END.
DECLARE alpha=(a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,
u,v,w,x,y,z);
DECLARE capalpha=(capa,capb,capc,capd,cape,capf,capg,caph,capi,
capj,capk,capl);
(lnmdpy) PROCEDURE;
dlblev ← numdpy(IF .NEG cdlev THEN -cdlev ELSE cdlev);
dlbrlv← (IF .NEG cdlev THEN 2000B
ELSE 1000B) .V (dlbrlv .A 77774777B);
RETURN END.
(aplit) PROCEDURE (aplit1);
IF .AR = 141B or .AR = 167B THEN BEGIN
IF cdlitf THEN RETURN;
aplit2 ← 0;
cdlitf ← 1;
CALL clrln( RSH $1(cdlit1));
WHILE aplit2 ≤ $1/litlc/ DO BEGIN
CALL aplita(ldchr(litlc,aplit2));
BUMP aplit2 END END
ELSE RETURN(aplita(aplit1));
RETURN END.
(aplita) PROCEDURE(aplit1);
*** this is the literal input routine ***
IF cdlitf THEN BEGIN
(aplila):
cdlit ← cred5;
cdlit5 ← 1;
cdlit2 ← cdlit4 ← -1;
cdlitf ← cdlite ← [cdlit/ + 0;
CALL clrln(cdcdrs);
cdlit1 ← LSH $1(cdcdrs,0);
cdlit9 ← $dlrt + 2;
[cdlit9/ ← [cdlit9/ .A 77770000B;
dl1buf/cdlit1/ ← cdlit + 40000B END;

```

PRMSPC OVERLAY, MOL PROCEDURES

```

**** now enter the character ****%
IF aplit1 = 151B THEN BEGIN
  %tabs%
  cdtabl ← fndtab(cdlit4);
  IF cdlitc ≠ 0 THEN BEGIN
    BUMP cdlit,cdlit5;
    dllbuf/cdlit1/ ←+ 40000B END;
  IF cdtabl ≥ cdncol THEN BEGIN
    %terminate line%
    CALL terlin(cdlit5+0);
    dllbuf/cdlit1/ ← cdlit .V 140000B;
    cdlit2 ← -1;
    cdlit5 ← 3;
    cdlitc ← 1;
    cdlit4 ← cdtabl ← fndtab(0) END
  ELSE BEGIN
    dllbuf/cdlit1/ ←+ 40000B;
    cdlit5 ←+ 2;
    cdlit3 ← cdlitc ← 1;
    cdlit8 ← cdlit4 ← cdtabl;
    cdlit2 ← cdlit END;
    cdtab2 ← RSH $1(cdlit1);
    cdlit ← puttab(cdlit,cdtabl,cdtab2);
    cdlit4 ← cdtabl + 1;
    RETURN END;
  CALL nterch(aplit1);
  IF .AR = -1 THEN GOTO aplilb;
  IF aplit1 = 155B THEN BEGIN
    BUMP cdlit;
    cdlit1 ← terlin(0);
    dllbuf/cdlit1/ ← cdlit .V 40000B;
    cdlit4,cdlit2 ← -1;
    [cdlit/ ← 37677577B;
    cdlit5 ← 1;
    cdlitc ← 0 END
  ELSE BEGIN
    BUMP cdlit4;
    IF cdlit4 > cdncol THEN BEGIN
      IF cdlit2 > -1 THEN BEGIN
        %transfer to next line%
        cdlit6 ← cdlit7 ← cdlit - cdlit2;
        BUMP cdlit6;
        IF cdlit3 = 0 THEN BEGIN
          %blank in top of word%
          CALL mvdown(cdlit,cdlit6,1);
          [cdlit2/ ←+ 100000B;
          BUMP cdlit2;
          [cdlit2/ ← [cdlit2/ .V 37600000B;
          BUMP cdlit;
          (cdlll):
          %terminate the line%

```

PRMSPC OVERLAY, MOL PROCEDURES

```

        .AR ← terlin(cdlit5-cdlit7);
        dllbuf/cdlit1/ ← LSH $14(cdlit6,0) .V cdlit2;
        cdlit4 ← (cdlit6-2)*3+cdlitc+2-cdlit3;
        cdlit2 ← -1;
        cdlit5 ← cdlit6 END
    ELSE IF cdlit3 = 2 THEN BEGIN
        %blank in bottom of word%
        cdlit6 ← cdlit7;
        BUMP cdlit2;
        GOTO cdlll END
    ELSE BEGIN
        %blank in middle%
        CALL mvdown(cdlit,cdlit6,1);
        BUMP cdlit;
        /cdlit2/ ←+ 200B;
        BUMP cdlit2;
        /cdlit2/ ← /cdlit2/ .V 37677400B;
        GOTO cdlll END END
    ELSE BEGIN
        %cdlit2 = -1%
        cdlit4 ← 0;
        BUMP cdlit1;
        BUMP cdlit1;
        BUMP cdlit;
        cdlite ← 0;
        dllbuf/cdlit1/ ← cdlit .V 40000B END END END;

RETURN;
(aplilb):
/cred5/ ← 37677577B;
GOTO aplila END.
(terlin) PROCEDURE (trmlil);
trmlil2 ← IF trmlil = 0 THEN 77777777B
        ELSE 37777B;
dllbuf/cdlit1/ ← (dllbuf/cdlit1/ .A trmlil2).V LSH $14(trmlil,0);
BUMP cdlit1;
BUMP cdlit1;
cdlit9 ←+ dlrtx;
IF cdlit1 = 2*cdnlin THEN BEGIN
    cdlit1 ← LSH $1(cdcrls,0);
    cdlit9 ← $dlrt +2;
    CALL crlin(cdcrls) END
ELSE BEGIN
    /cdlit9/ ← /cdlit9/ .A 77770000B;
    CALL crlin( RSH $1(cdlit1)) END;
RETURN(cdlit1) END.
(nterch) PROCEDURE (ntercl);
CALL nterca;
IF ntercl = 0 THEN BEGIN
    cdlit2 ← cdlit;
    cdlit8 ← cdlit4;
    cdlit3 ← cdlitec END;

```

PRMSPC OVERLAY, MOL PROCEDURES

```

BUMP cdlitc;
IF cdlitc > 2 THEN BEGIN
  BUMP cdlit;
  IF cdlit = $dlebuf THEN RETURN(-1);
  cdlitc ← 0;
  [cdlit] ← 37677577B;
  BUMP cdlit5;
  dllbuf [.XR] ← dllbuf/cdlit1/ + 40000B END;
RETURN(0) END.
(nterca) PROCEDURE;
%enters the character in the a reg%
.BR ← 0;
EXECUTE chr1sh/cdlitc/;
cdtemp ← .AR;
[cdlit] ← cdtemp .V ([cdlit] .A chrmas (cdlitc/ ));
RETURN END.
(frzsav) PROC;
cdlev/1/←cdlev;
cdtrn/1/←cdtrn;
cdblnk/1/←cdblnk;
RETURN END.
(dpych) PROCEDURE;
CALL aplit(.AR);
BRS $41() END.
(jspush) PROCEDURE;
CALL jpsch;
[jstack] ← cdpsid;
CALL jpsch;
[jstack] ← savlt();
RETURN END.
(jpsch) PROCEDURE;
BUMP jstack;
IF jstack ≤ jstck1 DO-SINGLE RETURN;
jstack ← $jstckd;
RETURN END.
(savlt) PROCEDURE;
%create word to save view specs%
.AR ← LSH $18(,IF cdtrn > 62 THEN 0
  ELSE .AR);
.AR ← LSH $6(IF cdlev > 62 THEN 0
  ELSE .AR,.BR);
.BR ← -1;
IF NOT .BR CB cdbrf DO-SINGLE .BR ← 0;
.AR←LSH $1();
.BR ← -1;
IF NOT .BR CB cdnamf DO-SINGLE .BR ← 0;
.AR←LSH $1();
.BR ← -1;
IF NOT .BR CB cdstnf DO-SINGLE .BR ← 0;
.AR←LSH $1();
.BR ← -1;

```

PRMSPC OVERLAY, MOL PROCEDURES

```

IF NOT .BR CB cdctaf DO-SINGLE .BR ← 0;
.AR← LSH $1();
.BR ← -1;
IF NOT .BR CB cdindf DO-SINGLE .BR ← 0;
.AR← LSH $1();
.BR ← -1;
IF NOT .BR CB cdblnk DO-SINGLE .BR ← 0;
.AR← LSH $1();
.BR ← -1;
IF NOT .BR CB cdtref DO-SINGLE .BR ← 0;
.AR← LSH $1();
.BR ← -1;
IF NOT .BR CB cdptr DO-SINGLE .BR ← 0;
.AR← LSH $1();
.BR ← -1;
IF NOT .BR CB cdfrzf DO-SINGLE .BR ← 0;
.AR← LSH $1();
.BR ← -1;
IF NOT .BR CB cdkeyf DO-SINGLE .BR ← 0;
.AR← LSH $1();
.BR ← -1;
IF NOT .BR CB cdtrlf DO-SINGLE .BR ← 0;
.AR← LSH $1();
.BR ← -1;
IF NOT .BR CB cdclpf DO-SINGLE .BR ← 0;
.AR← LSH $1();
RETURN(.AR) END.
(fechux) PROCEDURE(fhuxa);
fhux3 ← fhux5 ← 0;
fhux1 ← $1/fhuxa/;
%fhux5 = start psid (0). fhux1 is length of st no%
fhux1 ← -1;
fhuxba ← 10;
fhuxz ← 20B;
% fhuxba is base (10 or 27), z is zero%
WHILE fhux1 ≤ fhux1 DO BEGIN
    BUMP fhux1;
    IF fhux1 > fhux1 THEN fhux2 ← 0
    ELSE fhux2 ← ldchr(fhuxa,fhux1);
    IF fhux2 > 77B THEN fhux2← -40B;
    fhux6 ← fhux2 - fhuxz;
    IF fhux6 ≤ -1 or fhux6 ≥ fhuxba THEN BEGIN
        fhux4 ← getsub(fhux5);
        IF fhux4 = fhux5 THEN RETURN;
        %no such st%
        WHILE fhux3 > 1 DO BEGIN
            IF getft1(fhux4) THEN RETURN;
            fhux4 ← getsuc(fhux4);
            fhux3 ← -1 END;
        fhux5 ← fhux4;
        fhux3 ← 0;
    
```

PRMSPC OVERLAY, MOL PROCEDURES

```

        IF fhuxba = 10 THEN BEGIN
            fhuxba+27;
            fhuxz+40B END
        ELSE BEGIN
            fhuxba+10;
            fhuxz+20B END END;
        fhux3 ← fhux3*fhuxba + fhux2 - fhuxz END;
    BUMP /fechux/;
    RETURN(fhux5,0) END.
(gtlit2) PROCEDURE;
    BUMP litrf;
    smtemp ← $0;
    EXECUTE rfbp2;
    $0 ← smtemp;
    CALL lodrfb(0,-1);
    litcb ← .XR;
    BUMP frzcpt/.XR/;
    litle ← crpgad/.XR/;
    litlcl ← .AR+1;
    /litle/ ← 2976;
    /litlcl/ ← -1;
    RETURN END.
(softcd) PROCEDURE;
    IF spsk = spsk1 DO-SINGLE GOTO gps;
    spsk ←+ -2;
    RETURN END.
(pbug) PROCEDURE (,pbug1);
    EAX 10; NOP 77B; %short tone%
    pbug2 ← - (LSH $12(0) ) + cdtop - cdvinc/4 ;
    %effective vert pos%
    pbug3←IF pbug1 .A 1777B > cdift THEN .AR-cdift ELSE 0;
    %horiz pos%
    pbug4 ← ( RSH $2(cdhinc) + pbug3) / cdhinc;
    IF .NEG pbug2 THEN pbug5 ← 0 %vert col number%
    ELSE BEGIN
        pbug5 ← ( RSH $2(cdvinc) + pbug2) / cdvinc ;
        pbug6 ← .BR;
        %remainder%
        IF pbug5 >= cdclin THEN BEGIN
            pbug5 ← cdclin;
            pbug6 ← 0 END;
        CALL gdlset(,,pbug5);
        IF NOT pbug5 AND gdldot() THEN BUMP pbug5
        ELSE BEGIN
            IF gdlnul() THEN BEGIN
                IF pbug6 <= RSH $2(cdvinc) THEN pbug5 ←+ -1
                ELSE BUMP pbug5 END;
            CALL gdlset(,,pbug5);
            WHILE gdlnul() DO BEGIN
                pbug5←+ -1;
                CALL gdlset(,,pbug5) END END END;

```

PRMSPC OVERLAY, MOL PROCEDURES

```

%vert line now in pbug5%
CALL gdlset(,,pbug5);
IF gdllcl() <= pbug4 THEN pbug4 ← .AR
ELSE IF gdllcl() > pbug4 THEN pbug4 ← .AR;
%now a potential row and col has been established%
%DO actual character selection%
IF gdlspc() THEN BEGIN
  %there is a special character in the line%
  pbug7 ← fndchr(pbug4,,pbug5);
  %col #,,line %%
  %RETURNS character # in the a and col number in the b%
  pbug4 ← .BR END
ELSE pbug7 ← pbug4 - gdllcl() + gdlsch();
CALL gdlset(,,pbug5);
/cdulpt/ ← (pbug4*cdhinc+cdlft) .V LSH $12(cdtop-cdvinc*pbug5,0) .V
4000B;
BUMP cdulpt;
dllul ← dllul + 40000B;
RETURN (gdlpsi(),pbug7) END.
(fndchr) PROCEDURE (fndchl,,fndch2); %col number, line number%
CALL gdlset;
fndch3 ← gdllcl();
%col counter%
fndch4 ← cdwork/1/ ← gdlsch();
fndch5 ← cdwork ← gdlpsi();
fndch7 ← $0;
EXECUTE rfbp2;
$0 ← fndch7;
CALL fechcl(1,,$cdwork);
%sets up the sdb%
WHILE fndch3 < fndchl DO BEGIN
  BRM $2/$cdwork/;
  fndch6 ← .AR;
  IF .AR > 177B THEN BEGIN
    BUMP fndch4,fndch4,fndch3;
    BRM $2/$cdwork/ END
  ELSE BEGIN
    BUMP fndch4,fndch3;
    BRM $2/$cdwork/;
    fndch6 ← .AR;
    IF .AR = 151B THEN fndch3 ← fndtab(fndch3);
    cdwork/1/ ←+ -1;
    CALL fechcl(1,,$cdwork) END END;
  IF fndch6 > 177B THEN BUMP fndch4;
  RETURN(fndch4,fndch3) END.
.FINISH of PRMSPC MOL

```