LIBE

```
FILE mlibe  % (arcsubsys,l109,) to (l10,libe9,) %
ALLOW!                                                                  2
NOMESS;                                                                 3
% condition flags %
   SET TENEX=1;
% entry addresses for NLS %
   DECLARE l300k=$nstart, l300k1=$stkint;
 % registers %
    REGISTER  g=0, % lstack %
    x=1, % hash index : node testing & generation %
    c=2, % current input character %
    t=12 %A3%, % node testing scratch%
    w=4, % binary word being formed %
    m=5, % mstack % k=3, % kstack % n=7, % nstack %
   mreg=6, % L10 sucess/fail register %
    me=8, % pointer to running node in output rule %
    r= 15, % libe calls & returns %
    lm=14, % l10 call stack mark %
    a1=10, a2=11, a3=12, a4=13, % accumulators %
   rg0=0, rg1=1, rg2=2, rg3=3, rg4=4, %fixed register numbers%
      rg5=5, rg6=6, rg7=7, rg8=8;
 % sets %
    SET uuo=41B, symhice=400000B, symkeep=116B;
    SET arsz=2048; %window size in characters%
 % declarations %
   DECLARE CONSTANT strbfx=180;
   DECLARE strbfn, strbfp, linebr;
   DECLARE STRING strbuf[strbfx];
   DECLARE STRING locsr[100];
    DECLARE savlm, pa, outpg;
    DECLARE EXTERNAL fnumi, fnumo, relocc, outbfa, errorn;
   DECLARE inproc, inbcnt, inbptr;
   DECLARE chradr, savpop, frmnls, tonls; REF chradr, inproc;
    DECLARE patch[20] % patch space %, wbuf[20], eof=0;
    DECLARE filchc, fixb2p, outb2p;
    DECLARE EXTERNAL chrflg=1B6, sr1flg=1B6;
    DECLARE EXTERNAL obptr, outp, outcnt; %preserve order - buffer
   pointers%
    DECLARE ibptr, inchp, incnt; %preserve order - buffer
   pointers%
    DECLARE wttyvl, wttv, wttc, litsfg,strad,xtime,rtime;
    DECLARE fildes[3], inf[3], outf[3], lincnt;
    DECLARE lc,nmark,kmark,lochtp,loclin, nummfg, iwp, bcw;
    DECLARE top, temp, ring[arsz], mrsiz, modrsz;
    DECLARE ssp1, sspp, sspc, symlen, rad, back, lcc, pcc, lccx;
    DECLARE litw, litbcw, litrn, litrnx, udefrf, sne2,link1, rsiz, iwpx;
    DECLARE savc, temp2,numv, numba1, numba, numsc, nump;
    DECLARE strptr, strc; %pointer to literal string%
    DECLARE fixbw[18],fixbp, fixb2, fixtyp;
    DECLARE outbw[18], outbp, outb2;
    DECLARE htlim, htmx, initat, rdsrfg=0;
    DECLARE htcntl, htcnt, htcntm, htr;
   DECLARE srg, srg1, srg2, srg3, srg4, srg5, srg6, srg7, srgend;
   DECLARE inpgno, inporg, inbtct, inppgo, tempc;
   DECLARE dsptab=(0,100000101B,0,0,0,0,0,0,0), brkloc;
   DECLARE levtab=($lv1wd,$lv2wd,$lv3wd), lv1wd, lv2wd, lv3wd,
```

```
    chntab[36];
  PAGE 8; DECLARE inpbuf[512], outbuf[512];
   DECLARE EXTERNAL htv[2048], htp[2048], hta[2048]; % hash table %
   EXTERNAL pcc, pa, nmark, kmark, callr, gochk, prj, mkgo, krummy, lc,
  fnumi, fnumo,frmnls, ppaprt,
     rdsrfg,
      htmx, temp, start, initll, xerror, bcw, errorn, tonls;
PAGE 8;
 % pop branch table %
   DECLARE poptab=
      (0, $qserr, $qserr, $qserr, $qppj, $qssck, $qmkdd, $qndlb,
      $qgen, $qmkls, $qstrf, $qcntst, $qritm, $qstitm, $qitmg, $qdoit,
      $qdfl, $qdfv, $qdgn, $qentr, $qpsr, $qpgn, $qserr, $qpas, $qget,
      $qserr, $qcerr, $qmkd, $qpav, $qnamt, $qbkupj, $qtst);
% UUO'S %
  SET PPJ=4, SSCK=5, MKDD=6, NDLB=7, GEN=10B, MKLS=11B,
  STRF=12B, CNTST=13B, RITM=14B, STITM=15B, ITMG=16B,
  DOIT=17B, DFL=20B, DFV=21B, DGN=22B, ENTR=23B,
  PSR=24B, PGN=25B, PAS=27B, GET=30B, SERR=31B,
  CERR=32B, MKD=33B, PAV=34B, NAMT=35B, BKUPJ=36B,
  TST=37B, CALLI=47B;
 % constants %
   DECLARE flgmsk=77B6, ssmsk=7B7,
      idflg=11B6, ptrflg=02B6, labflg=03B6,
      srflg=13B6, ssflg=1B7, uidflg=12B6;
   DECLARE gblrg=60B10,extsy=04B10, locsy=10B10,eolchr=37B;
   DECLARE device=44635B6, htamsk=11M, htrmsk=13M;
  DECLARE pgsize=1000B, pgcsize=5000B;
   DECLARE chrp=1070006, chr0=4407006, wrdp=143006;
 % fields %
   DECLARE FIELD lbcw=[litr(c),2:19], % literal bcw %
      ludfrf=[litr(c),1:18], litstr=[litr(c),1:21];
     % CAUTION: index must be in reg c (2) %
   DECLARE EXTERNAL FIELD defned=[htp(x),1:24],
      relcca=[htp(x),1:23], all=[htp(x),12:13],
      opcode=[htp(x),1:22], extern=[htp(x),1:21],
      linked=[htp(x),1:20], noddt=[htp(x),1:19],
     prefx=[htp(x),1:18]
      ;
   DECLARE FIELD regn3=[pa(0),3:23], regn4=[pa(0),4:23],
      gbit=[pa(0),1:26], popreg=[40B(0),4:23];
   DECLARE EXTERNAL FIELD htax=[htp(x),12:13],
      htlx=[htp(x),11:25], htpx=[htp(x),13:0];
   DECLARE FIELD cp1=[1(c),24:12], cp2=[1(c),8:4],
      cp3=[2(c),16:20], inpsiz=[inchp(0),6:24];
   DECLARE EXTERNAL FIELD fsqc=[x(0),8:0];
   DECLARE FIELD ob2pwd=[outb2(0),2:34], fb2pwd=[fixb2(0),2:36];
 % message writing %
   (qserr) PROCEDURE;                                                  14
      % system error %
     tcrlf();
      wstri($"COMPILER SYSTEM ERROR: ",-1);
      CASE pa.RH OF
         =0: wstri($"END OF FILE ",-1);
         =1: wstri($"HASH TABLE FULL --TOO MANY SYMBOLS",-1);
         =2: wstri($"BACKUP TOO FAR ",-1);
```

```
        =3: wstri($"SYMBOL TOO LONG (limit=2047 chars) ",-1);
        =4: wstri($"INPUT TOO LONG ",-1);
        =6: wstri($"SYMBOL STO FULL --TOO MANY SYMBOLS",-1);
        =7, =9: wstri($"I/O ERROR ",-1);
        =14: wstri($"LITERAL TABLE FULL ",-1)
        ENDCASE wvali(pa.RH, 8); xfin();
    %+TENEX%                                                          14A5
       (stkint):                                                      14A5A
          S _ -gstksz; !HRL S,S; !HRRI S,gstack;
             % in case gstack overflow %
          temp _ [brkloc]; % save it %
          [brkloc] _ $stkovr;
          !debrk();
       (iglint): % illegal instr %                                    14A5B
          temp _ [brkloc];
          [brkloc] _ $iglmsg;
          temp2 _ $"ILLEGAL COMPILER INSTR AT ";
          !debrk();
       (memint): % illegal mem ref %                                  14A5C
          temp _ [brkloc];
          [brkloc] _ $iglmsg;
          temp2 _ $"ILLEGAL COMPILER MEMORY REF AT ";
          !debrk(); %+TENEX%
    (stkovr): %come here on stack overflow%                           14A6
       tcrlf();
       R1 _ [temp.RH]; !LSH R1,-23; temp _ R1;
       CASE temp .A 4M OF
            =$m: a1 _ $"M STACK OVERFLOW ";
            =$k: a1 _ $"K STACK OVERFLOW ";
            =$n: a1 _ $"N STACK OVERFLOW--statement or rule too long ";
            =$g: a1 _ $"G STACK OVERFLOW ";
            =$r: a1 _ $"L10 STACK OVERFLOW ";
            ENDCASE a1 _ $"STACK OVERFLOW ";
       wstri(a1,-1);
       xfin();
    (iglmsg): % here on igl instr and igl mem ref %                   14A7
       tcrlf();
       wstri(temp2, -1);
       wvali(temp.RH-1, 8);
       xfin();
    END.
  (qcerr) PROCEDURE;                                                  14B
    tcrlf();
     wstri($"COMPILER ERROR: ",-1);
     CASE pa .A 18M OF
        =1001: wstri($"ILLEGAL NODE REF ",-1);
        =1002: wstri($"* FAILED ",-1);
        =1003: wstri($"NON-TERMINAL NODE VALUE ",-1);
        =1004: wstri($"ADDRESS SMASHING ",-1);
        =1005: wstri($"DUMMY RULE ",-1);
        =1006: wstri($"ILLEGAL REF CALL ",-1);
        =1000: wstri($"UDEF GEN LABEL ",-1);
        =0: wstri($"OUTRULE FAILED ",-1)
        ENDCASE wvali(pa,10);
     locw(); xerror() END.
  (qxerr) PROCEDURE;                                                  14C
```

```
    tcrlf();
     wstri($"SYNTAX ERROR: ",-1);
    locw();
    xerrcr() END.
(xerror) PROCEDURE;                                                    14D
     % syntax & compiler errors recover here %
    CALL errbuf; BUMP errorn;
     bcw_w _ 0; CALL wprep; n_nsx; k_ksx; m_msx; g_gsx;
     mknk();
    IF NOT frmnls THEN
       BEGIN
       S _ -gstksz; !HRL S,S; !HRRI S,gstack; M _ S;
       END
    ELSE
       BEGIN
       S _ M; !POP S,M; S _ M; !POP S,M;
       END;
    GOTO qsynqq;
    END.
 (xfin) PROCEDURE;                                                     14E
    BUMP errorn; locw(); errbuf(); !POP r,temp; !POP r,lm;
     CALL tables; CALL finish END.
 (tcrlf) PROCEDURE;                                                    14F
     % type cr, lf on message file%
    savr(); outchr(CR); outchr(LF); wstrf(); rstr();
    RETURN END.
 (errbuf) PROCEDURE;                                                   14G
     LOCAL wschar;
     % write last 30 chars of input %
     temp2 _ iwp-30;
     IF temp2 < $ring THEN temp2 _ temp2 + rsiz;
     temp _ 0;
     WHILE temp <= 30 DO BEGIN
         wschar _ [temp2]; outchr(wschar); BUMP temp2,temp;
         IF temp2 > iwpx THEN temp2 _ $ring END;
    outchr(SP);
    IF locsr.L THEN
       BEGIN
       outchr("");
       wstri($locsr);
       outchr("");
       END;
    tcrlf(); % sout the string %
     RETURN END.
 (locw) PROCEDURE;                                                     14H
     % write location set by &LABEL in compiler %
     wstri($" LINE ",-1); wvali(lincnt,10);
     outchr(" "); IF lochtp # -1 THEN wstrhi(lochtp);
    outchr("+"); wvali(lincnt-loclin,10);
    outchr(CR); outchr(LF);
     RETURN;
    END.
 (locset) PROCEDURE;                                                   14I
     % called by &LABEL %
     lochtp _ [k]; loclin _ lincnt; RETURN END.
 (finish) PROCEDURE;                                                   14J
```

```
    % close output, write parameters and quit %
    IF NOT tonls THEN
      BEGIN
       purgeb(); purgef();
       out36(586+1); out36(2811); out36(lc);
       IF outcnt < 512 THEN BEGIN
         savr();
         !sout(fnumo, $outbuf+444488, outcnt-512, 0);
          rstr();
         END
      END;
   IF NOT tonls THEN BEGIN
      % set eof pointer %
      savr();
      rg1 _ 1286 + fnumo; rg2 _ -1;
      rg3 _ outpg*512+(512-outcnt);
      !JSYS chfdb;
      !pmap(-1,4811+($outbuf/512),1006);
      rstr() END;
   IF NOT frmnls THEN
      !pmap(-1,4811+inppgo,188);
   IF frmnls THEN BEGIN
      uuo _ savpop; lm _ savlm; RETURN(errorn,lc) END;
   terlf();
   wstri($"FINISHED");
   terlf();
   IF errorn = 0 THEN wstri($"no") ELSE wvali(errorn,10);
   IF errorn = 1 THEN wstri($" error") ELSE wstri($" errors");
   terlf();
   rg1_485; !JSYS runtm; xtime _ (rg1-xtime)/rg2;
   !JSYS time; rtime _ (rg1-rtime)/rg2;
    wvali(lc,8); wstri($" WORDS",-1); terlf();
    wpair('t,xtime,rtime);
    wpair('s,(ssp1 .A 18M)-$ss, (sspx .A 18M)-$ss);
    wpair('p,htcntm+htcntl,htcntl);
    wpair('h,htcnt,htlim);
   terlf();  fndtop(ksx,'k);
    fndtop(msx,'m); fndtop(nsx,'n);
    wpair('l,litrn,litx); fndtop(gsx,'g);
   terlf();
   wstrf(); % do the sout %
   IF frmnls THEN BEGIN
      uuo _ savpop; lm _ savlm;
      RETURN(errorn,lc) END;
   !clzff(485);
   R1 _ errorn;
   !JSYS haltf;
    END.
(fndtop) PROCEDURE(ftemp,letr);                                    14K
    % find top cell used in stack (initial pointer)%
    a1 _ ftemp;
    !HLRO a3,a1; !HRLI a1,0; a3_-a3; a1 _ a1+a3-1; !HRL a1,a3;
    !HRLOI x,0;
    DO !POP a1,temp2 WHILE temp2=0 AND a1>x;
    !HLRZ x,a1; !HLRO a3,ftemp; ftemp_-a3;
    wpair(letr,x,ftemp); RETURN END.
```

```
 (wstrhi) PROCEDURE(wx);                                                14L
    % write symbol given (hash #) %
    x _ wx; wstri(.htpx+$ss, .htlx);
    RETURN END.
 (wstrh) PROCEDURE(wx);                                                 14M
    % write symbol given (hash #) %
   wstrhi(wx); % same for now %
    RETURN END.
 (wstr) PROCEDURE(wstrad,strln);                                        14N
   wstri(wstrad, strln);
    RETURN END.
 (wstri) PROCEDURE(wstrad REF,strln);                                   14O
    LOCAL indx, max, source;
   source _ &wstrad + 4407B8+1;
   max _ wstrad.L;
   IF max IN [1,128] AND max<=wstrad.M THEN NULL
   ELSE max _ strln;
   IF strbfn + max > strbfx THEN wstrf();
   FOR indx _ 1 UP  UNTIL > max DO
      BEGIN
      BUMP strbfn;
      |strbfp _ |source;
      END;
    RETURN END.
% io stuff %
 (wstrf) PROCEDURE; % write string buffer to terminal %                 15A
    IF strbfn THEN
      BEGIN
      |strbfp _ 0;
      !psout($strbuf + 4407B8+1);
      END;
   strbfp _ $strbuf+4407B8+1;
   strbfn _ 0;
   RETURN;
   END.
 (wvali) PROCEDURE(wval1,base);                                         15E
   LOCAL oldp;
   oldp _ strbfp;
   savr();
   rg1 _ strbfp; rg2 _ wval1; rg3 _ base;
   !JSYS nout; !SERR 10; strbfp _ rg1; rstr();
   strbfn _ strbfn + (strbfp.RH-oldp.RH)*5 + 5;
      % this is too big, but only used to check overflow %
    IF base=8 THEN outchr("B);
    RETURN END.
 (wval) PROCEDURE(wval1,base);                                          15O
   wvali(wval1,base);
    RETURN END.
 (wpair) PROCEDURE(wx,wtemp,wpair1);                                    15P
    outchr(wx); outchr("=);
    wvali(wtemp,10); outchr("/); wvali(wpair1,10); outchr(" );
    RETURN END.
 (outchr) PROCEDURE(wschar);                                            15E
   |strbfp _ wschar;
   BUMP strbfn;
   IF strbfn >= strbfx THEN wstrf();
```

```
    RETURN END.
(opnfl) PROCEDURE;                                                    15F
    LOCAL ptr;
    % open input with .TXT ext, output with same string
    and ext of .REL %
    savr();
    (opnfst): rg1_-1; !JSYS rljfn; !SERR 9;                           15F5
    wstr($"-INPUT: ",-1); wstrf();
       dsptab _ 16B10; dsptab[5] _ 18M6 .V $"TXT"+1;
       rg1 _ $dsptab; rg2 _ 0;
       IF NOT SKIP !JSYS gtjfn THEN GOTO opnfst;
       fnumi _ x; ptr _ chr0 .V $wbuf;
       rg1 _ ptr; rg2 _ fnumi; rg3 _ 1B9+1; !JSYS 30B;
       tcrlf();
    wstr($"-OUTPUT: ",-1); wstrf(); dsptab _ 66B10;
       dsptab[5] _ $"REL" .V 18M6+1; rg2 _ 0;
       dsptab[4] _ ptr; rg1 _ $dsptab;
       IF NOT SKIP !JSYS gtjfn THEN GOTO opnfst; fnumo _
       rg1;
    rg1 _ fnumi; rg2 _ 0700002B5;
    IF NOT SKIP !JSYS openf THEN GOTO opnfst;
    rg1 _ fnumo; rg2 _ 4400001B5;
    IF NOT SKIP !JSYS openf THEN GOTO opnfst;
    % set byte size %
    rg1 _ 11B6+fnumo; rg2 _ 77B8; rg3 _ 44B8;
    !JSYS chfdb;
    tcrlf(); rstr(); RETURN END.
(savr) PROCEDURE;                                                     15G
    srg1 _ rg1; srg2 _ rg2; srg3 _ rg3; srg4 _ rg4;
    RETURN END.
(rstr) PROCEDURE;                                                     15H
    rg1 _ srg1; rg2 _ srg2; rg3 _ srg3; rg4 _ srg4;
    RETURN END.
% input routines %
(get1) PROCEDURE; % get 1 input character %                           16A
    UNTIL (inbcnt _ inbcnt - 1) >= 0 DO
       BEGIN
       savr();
       inbptr _ inproc(:inbcnt);
       rstr();
       END;
    c _ |inbptr;
    RETURN(c);
    END.
(putin) PROCEDURE;                                                    16B
    LOCAL locsrp;
    % put a char into window %
    LOOP % til we find a good char to return with %
       BEGIN
       get1();
       IF linebr AND c=0 AND get1()="" THEN
          BEGIN
          locsrp _ $locsr + 4407B8 + 1;
          locsr.L _ 0;
          DO
             BEGIN
```

```
            get1();
            |locsrp _ c;
            locsr.L _ locsr.L + 1;
            END
            WHILE c # '"';
         locsr.L _ locsr.L - 1; % remove the " %
         get1();
         END;
      linebr _ FALSE;
      [iwp] _ c;
      IF c IN ['' , 'z] THEN RETURN;
      CASE c OF
         >=SP: RETURN; % nearly all of them... %
         =EOL, =CR:
            BEGIN
            IF c=EOL THEN BUMP linebr;
            BUMP lincnt;
            IF NOT rdsrfg THEN c _ [iwp] _ SP;
            RETURN;
            END;
         =LF:
            BEGIN
            BUMP linebr;
            IF rdsrfg THEN RETURN; % else next char %
            END;
         =TAB:
            BEGIN
            IF NOT rdsrfg THEN c _ [iwp] _ SP;
            RETURN;
            END;
         =0: NULL; % next char %
         ENDCASE RETURN;
      END;
   END.
(inc) PROCEDURE;                                                    16C
   % increment window and input char %
   BUMP iwp;
   IF iwp > iwpx THEN iwp _ $ring;
   IF back<0 THEN BEGIN
      BUMP back; RETURN(c_[iwp]) END;
   BUMP lcc;  IF lccx <= lcc THEN !SERR 4;
   putin();
   RETURN END.
(delb) PROCEDURE;                                                   16C
   %delete blanks and fix window pointers%
   IF c = SP THEN inc();
   LOOP BEGIN
      WHILE c = SP DO putin();
      IF c # '% THEN
         BEGIN
         pcc _ lcc + back;
         lccx _ lcc + rsiz;
         RETURN;
         END;
      delcmt();
      END;
```

```
    END.
 (delcmt) PROCEDURE;                                                    16E
    LOCAL idhsh;
    IF NOT rdcmnt() THEN
       BEGIN skiper(); putin(); RETURN END;
    idhsh _ x; putin();
    DO skiper() UNTIL rdcmnt() AND idhsh=x;
    putin(); RETURN END.
 (rdcmnt) PROCEDURE;                                                    16F
    putin(); IF NOT rdcm1() THEN RETURN(FALSE);
    IF c#"% THEN RETURN(FALSE); RETURN(TRUE) END.
 (rdcm1) PROCEDURE;                                                     16G
    CASE c OF
       ="+: IF cid() AND htv[x]=0 THEN RETURN(TRUE);
       ="-: IF cid() AND htv[x]#0 THEN RETURN(TRUE);
       ENDCASE;
   RETURN(FALSE) END.
 (skiper) PROCEDURE;                                                    16H
    UNTIL c="% DO putin(); RETURN END.
 (cid) PROCEDURE;                                                       16I
    putin(); IF c NOT IN ["A,"Z] THEN RETURN(0);
    entrc(); putin();
    WHILE c IN ["A,"Z] OR c IN ["0,"9]
       OR c IN ["a,"z] DO putin(entrc());
    entrs(); IF .cefned THEN RETURN(1);
    RETURN(0) END.
 (wprep) PROCEDURE;                                                     16J
    % prepair window for input %
    IF (back_pcc-lcc) NOT IN (mrsiz,0] THEN !SERR 2;
    iwp _ pcc .A modrsz + $ring; c _ [iwp];
    lccx _ lcc + rsiz;
    RETURN END.
 (bsav) PROCEDURE;                                                      16K
    % save for backup %
    !PUSH m,pcc; !PUSH m,n; !PUSH m,k; !PUSH m,kmark;
    RETURN END.
 (brstr) PROCEDURE;                                                     16L
    % restore for backup %
    !POP m,kmark; !POP m,k; !POP m,n; !POP m,pcc; wprep();
    RETURN END.
 (bpop) PROCEDURE;                                                      16M
    % pop for backup %
    !POP m,temp; !POP m,temp; !POP m,temp; !POP m,temp;
    RETURN END.
 (qbkupj) PROCEDURE;                                                    16N
    brstr(); GOTO prj END.
% input routines %
 (innls) PROCEDURE;                                                     17A
   % input from NLS - return byte ptr and count %
   !MOVEI a1,srg; !BLT a1,srgend; %save registers%
   tempc _ chradr( : inbtct);
   !MOVSI a1,srg; !BLT a1,8; %restore registers%
   IF tempc=-1 THEN !SERR 0;
   &inproc _ $innl2; %get an EOL next time%
   RETURN(tempc, inbtct) END.
 (innl2) PROCEDURE; % fake an EOL between NLS statements %              17B
```

```
    &inproc _ $innls; %get next statement after this EOL%
    tempc _ EOL;
    RETURN(070700B6+$tempc, 1) END.
  (intxt) PROCEDURE; % caller must save registers %                    17C
    %+TENEX%                                                           17C1
    % input from TXT file - return byte ptr and count %
    rg1 _ fnumi; !JSYS sizef; !SERR 9; inbtct _ rg2;
    IF (inpgno+1)*pgcsize>=inbtct THEN !SERR 0;
    mapin(inpgno+1);
    tempc _ MIN(pgcsize, inbtct - inpgno*pgcsize);
    RETURN(440700B6+inporg,tempc) END.
    %+TENEX%                                                           17C8
    %-TENEX%                                                           17C9
    IF eof THEN !SERR 0;
    !INPUT chnin,0;
    !STATZ chnin,34B4;
    !SERR 7;
    !STATZ chnin,2B4;
    BUMP eof;
    RETURN(inchp, incnt) END.
    %-TENEX%                                                           17C17
  %+TENEX%                                                             17D
   (mapin) PROCEDURE(pgno);                                            17E
    % map page into buffer %
    inpgno _ pgno;
    !HRL rg1,fnumi; !HRR rg1,inpgno;
    rg2 _ 4B11 %this fork% + inppgo;
    rg3 _ 1B11 %read access%; !JSYS pmap;
    RETURN END.
  %+TENEX%                                                             17F
% basic recognizers %
   (pid) PROCEDURE(precod);                                            18A
     LOCAL gotlower, savsp, savspc;
      % lower case identifier %
      IF c NOT IN ['a,'z] AND c NOT IN ['A, 'Z] THEN RETURN [FALSE];
     gotlower _ IF c IN ['a, 'z] THEN TRUE ELSE FALSE;
     savsp _ sspp; savspc _ sspc;
     %entrc% |sspp _ precod; BUMP sspc; IF sspp=sspx THEN !SERR 6;
     %duplicate the code for entrc to speed up ic's%
     %entrc% |sspp _ c; BUMP sspc; IF sspp=sspx THEN !SERR 6;
     inc();
      UNTIL c NOT IN ['A,'Z] AND c NOT IN ['a,'z] AND c NOT IN ['0, '9] DO
        BEGIN
        IF c IN ['a, 'z] THEN BUMP gotlower;
        %entrc% |sspp _ c; BUMP sspc; IF sspp=sspx THEN !SERR 6;
        inc();
        END;
     IF NOT gotlower THEN
        BEGIN
        sspp _ savsp; sspc _ savspc;
        wprep();
        RETURN [FALSE];
        END;
      entrs(); x_x .V idflg; !PUSH k,x; delb();
      IF .htax=0 THEN .reloca_1;
     .prefx _ 1;
```

```
    RETURN [TRUE] END.
(p1id) PROCEDURE;                                                    18B
   LOCAL f;
   pid(*. : [f]);
   RETURN [f];
   END.
(p2id) PROCEDURE;                                                    18C
   LOCAL f;
   pid(*$ : [f]);
   RETURN [f];
   END.
(p3id) PROCEDURE;                                                    18D
   LOCAL f;
   pid(*% : [f]);
   RETURN [f];
   END.
 (id) PROCEDURE;                                                     18E
   LOCAL gotlower, savsp, savspc;
    % lower case identifier %
    IF c NOT IN [*a,*z] AND c NOT IN [*A, *Z] THEN RETURN [FALSE];
   gotlower _ IF c IN [*a, *z] THEN TRUE ELSE FALSE;
   savsp _ sspp; savspc _ sspc;
   %duplicate the code for entrc to speed up ic*s%
   %entrc% |sspp _ c; BUMP sspc; IF sspp=sspx THEN !SERR 6;
   inc();
    UNTIL c NOT IN [*A,*Z] AND c NOT IN [*a,*z] AND c NOT IN [*0, *9] DO
      BEGIN
      IF c IN [*a, *z] THEN BUMP gotlower;
      %entrc% |sspp _ c; BUMP sspc; IF sspp=sspx THEN !SERR 6;
      inc();
      END;
   IF NOT gotlower THEN
      BEGIN
      sspp _ savsp; sspc _ savspc;
      wprep();
      RETURN [FALSE];
      END;
    entrs(); x_x .V idflg; !PUSH k,x; delb();
    IF .htax=0 THEN .reloca_1;
    RETURN [TRUE] END.
 (uid) PROCEDURE;                                                    18F
    % upper case identifier %
    IF c NOT IN [*A, *Z] THEN RETURN [FALSE];
    inc(entrc());
    UNTIL c NOT IN [*A,*Z] AND c NOT IN [*0,*9] DO inc(entrc());
    entrs(); x_x .V uidflg; !PUSH k,x; delb();
    RETURN [TRUE] END.
 (psr) PROCEDURE(precod);                                            18G
   LOCAL f;
    % strings include CR LF etc %
    IF c # *" THEN RETURN [FALSE];
   %entrc% |sspp _ precod; BUMP sspc; IF sspp=sspx THEN !SERR 6;
   sr(: [f]);
   IF f THEN .prefx _ 1;
   RETURN [f];
    END.
```

```
(p1sr) PROCEDURE;                                                    18H
   LOCAL f;
   psr(".  : [f]);
   RETURN [f];
   END.
(p2sr) PROCEDURE;                                                    18I
   LOCAL f;
   psr("$  : [f]);
   RETURN [f];
   END.
(p3sr) PROCEDURE;                                                    18J
   LOCAL f;
   psr("%  : [f]);
   RETURN [f];
   END.
 (sr) PROCEDURE;                                                     18K
    % strings include CR LF etc %
    IF c # "" THEN RETURN [FALSE];
    rdsrfg _ -1; CALL inc;
    WHILE c # "" DO
      BEGIN
      %entrc% |sspp _ c; BUMP sspc; IF sspp=sspx THEN !SERR 6;
      inc();
      END;
    rdsrfg _ 0; CALL inc;
    CALL entrs; x_x .V srflg; !PUSH k,x; CALL celb;
    RETURN [TRUE]
    END.
 (sr1) PROCEDURE;                                                    18L
    % single character string %
    IF c # "" THEN RETURN [FALSE];
    CALL inc; a1 _ c .V sr1flg; !PUSH k,a1;
    delb(inc());
    RETURN [TRUE]
   END.
 (num) PROCEDURE;                                                    18M
   LOCAL f;
    % retain only value %
    xnum(:[f]);
   IF NOT f THEN RETURN [FALSE];
   numval(); !PUSH k,numv;
    delb();
    RETURN [TRUE]
   END.
 (xnum) PROCEDURE;                                                   18N
    IF c NOT IN ["0,"9] THEN RETURN [FALSE];
    nummfg_numsc_0; numba1_numba;
   %entrc% |sspp _ c; BUMP sspc; IF sspp=sspx THEN !SERR 6;
   inc();
    UNTIL c NOT IN ["0,"9] DO
      BEGIN
      %entrc% |sspp _ c; BUMP sspc; IF sspp=sspx THEN !SERR 6;
      inc();
      END;
    IF c="M THEN BUMP nummfg
    ELSE IF c="8 THEN numba1_8
```

```
    ELSE IF c=*0 THEN BEGIN
      %look ahead for a number following the 0%
      numba1 _ iwp; %ugly save%
      inc();
      iwp _ numba1; back _ back-1;
      IF c NOT IN [*0,*9] THEN RETURN [TRUE](numba1 _ numba);
      numba1_10;
      END
    ELSE RETURN [TRUE];
    inc(); numsc_0;
    WHILE c IN [*0,*9] DO BEGIN
       numsc _ numsc*10 + c-*0; inc() END;
    RETURN [TRUE]; END.
 (numval) PROCEDURE;                                                    180
    % evaluate number %
    nump _ ssp1; numv _ 0;
    !JFCL 15,numlb1; (numlb1):
    WHILE nump # sspp DO
       numv _ numv*numba1 -*0 + |nump;
    IF nummfg THEN BEGIN
       !HRRZI a1,0; !HRROI a2,-1; !LSHC a1,@numv;
       numv _ a1; numba1 _ 8 END;
    WHILE (numsc:=numsc-1)>0 DO numv _ numv*numba1;
    !JFCL 8,numov; GOTO numlb2;
       (numov): numv _ numv .V 4B11;                                    1807A
    (numlb2): sspp _ ssp1; sspc_0; c _ [iwp]; RETURN END.               1808
 (chr) PROCEDURE;                                                       18P
    % input a character %
    a1 _ c .V chrflg; !PUSH k,a1; RETURN(delb(inc())) END.
(cvdate) PROCEDURE;                                                     18Q
   %+TENEX%                                                             18Q1
   % converts ( DATE ) to a number on k stack %
   LOCAL savr1, savr2;
   IF c # *( THEN RETURN [FALSE];
   inc(); delb();
   WHILE c # *) DO inc(entrc());
   |sspp _ 0;
   inc(); delb();
   savr1 _ rg1; savr2 _ rg2;
   rg1 _ ssp1; rg2 _ 0;
   IF SKIP !JSYS 221B %date conversion% THEN
      BEGIN
      rg1 _ 112000011B;;                                                18Q11B
      IF SKIP !CALLI rg1,41B % returns info on operating system in use %
      THEN                                                              18Q11C
         BEGIN                                                          18Q11C1
         IF rg1 = 40000B % TOPS20 % THEN                                18Q11C2
            BEGIN                                                       18Q11C2A
            % Convert time to TENEX format %
               % get time ( in fraction of days * 2**18 ) into right half
               of register 1.  Left half of r2 has days since system 0. %
                  !HRRZ rg1,rg2;
               % multiply by 86,400: number of seconds in a day %
                  !IMULI rg1,250600B;
               % left half of one now has number of seconds in day * 2 **
               18: but round up if right half indicates more than 1/2
```

```
                second remains in right half %
                   !ADDI rg1,400000B;
                % move tenex time from left half of 1 to right half of 2 %
                   !HLR rg2,rg1;
             END;                                                    18Q11C2C
          % Register 2 is now in TENEX format date and time %
          !PUSH k,rg2;
          rg1 _ savr1; rg2 _ savr2;
          sspp _ ssp1;
          sspc _ 0;
          %+TENEX%                                                  18Q11C8
          RETURN [TRUE];
          END;                                                      18Q11C1C
       % ELSE do error return %
       END;
     % error: restore registers and return FALSE %
       rg1 _ savr1; rg2 _ savr2;
       sspp _ ssp1;
       sspc _ 0;
       RETURN [FALSE];
     END.
   (advsk) PROCEDURE;                                                   18F
     % delete blanks and SKIP RETURN %
      delb(inc());
     BUMP [M.RH-1]; % SKIP RETURN !!!!!! %
     RETURN;
     END.
   (qtst) PROCEDURE;                                                    18S
      % test for given string & advance %
     %duplicate spset for speed%
         !HRRZ a1,pa; strptr _ a1 .V chrp;
         IF (x_[pa])<0 THEN BEGIN x_-x; strc_.htlx END
         ELSE strc _ x .A 18M;
     x _ |strptr;
      WHILE c=x AND (strc_strc-1)>0 DO BEGIN
         x _ |strptr; inc() END;
      IF strc>0 THEN BEGIN
        %copy wprep for speed%
         IF (back_pcc-lcc) NOT IN (mrsiz,0] THEN !SERR 2;
         iwp _ pcc .A modrsz + $ring; c _ [iwp];
         lccx _ lcc + rsiz;
        RETURN END;
     % SKIP RETURN !!!!!! %
        BUMP [M.RH-1];
      RETURN(delb(inc())) END.
   (spset) PROCEDURE;                                                   18T
      !HRRZ a1,pa; strptr _ a1 .V chrp;
      IF (x_[pa])<0 THEN BEGIN x_-x; strc_.htlx END
      ELSE strc _ x .A 18M;
      RETURN END.
% string storage & hash table control %
   (entrs) PROCEDURE;                                                   19A
      % enter string indicated by sspp, sspc %
      savc_c; c_0; symlen _ sspc;
      IF symlen>2047 THEN !SERR 3;
      temp _ ((sspc+4)/5) * 5;
```

```
    WHILE sspc < temp DO
      BEGIN
      %blank fill%
      %entrc% |sspp _ c; BUMP sspc; IF sspp=sspx THEN !SERR 6;
      END;
    IF sspc<=5 THEN [ssp1+2]_0;
    c _ ssp1;
    !LDB a1,cp1; !LDB a2,cp2; !LDB a3,cp3;
    !LSH a2,16; !IOR a2,a3; !ADDI a1,200401B;
    !ADDI a2,200401B; !MUL a1,a2;
    !LSHC a1,20; !IDIVI a1,4B3;
    !MOVE x,a2; htr _ a1; x _ (x + htr) .A htamsk;
    IF x=0 THEN x_1;
    BUMP htcntl;
    WHILE htp[x] # 0 DO BEGIN
       IF streqq(ssp1,symlen) THEN GOTO entrs1;
       BUMP htcntm;
       x _ (x+(htr_htr+31)) .A htamsk;
       IF x=0 THEN x_1 END;
    IF htlim <= htcnt THEN !SERR 1;
    .htpx _ ssp1-$ss; .htlx _ symlen; .htax _ initat;
    ssp1 _ sspp; BUMP htcnt;
    (entrs1): sspc _ 0; sspp _ ssp1; c_savc; RETURN END.                    19A19
(streqq) PROCEDURE(snep,snec);                                              19B
    % return true if strings are equal - hash# in x %
    IF snec # .htlx THEN RETURN (FALSE); snec _ (snec+4)/5;
    snep _ (snep .A 18M) .V wrdp;
    sne2 _ (.htpx+$ss ) .V wrdp;
    DO
       IF |snep#|sne2 THEN RETURN (FALSE) WHILE (snec_snec-1)>0;
    RETURN (TRUE) END.
(entrc) PROCEDURE;                                                          19C
    % enter a character on end of ss %
   % this routine is duplicated in various places for speed %
    |sspp _ c; BUMP sspc; IF sspp=sspx THEN !SERR 6;
    RETURN END.
(qentr) PROCEDURE;                                                          19D
  LOCAL savt;
    % enter a given literal string %
    IF [pa]<0 THEN RETURN(x_ (-[pa]) .V srflg);
  savt _ t;
    entrs(entrsr()); [pa] _ -x; x_ x.V srflg;
  t _ savt;
    RETURN END.
(entrsr) PROCEDURE;                                                         19E
    % move literal string to ss %
    spset(); savc _ c;
    WHILE (strc _ strc-1) >= 0 DO BEGIN
       c _ |strptr; entrc() END;
    c _ savc; RETURN END.
% output routines %
(out36) PROCEDURE(word);                                                    20A
    % put 36 bit word into buffer %
  [outp] _ word; outp _ outp+1;
    IF (outcnt_outcnt-1)=0 THEN BEGIN
      IF tonls THEN !SERR 8
```

```
      ELSE BEGIN
         savr();
         !sout(fnumo, $outbuf+444488, -512, 0);
         outpg _ outpg + 1;
         rstr(); outcnt_512; outp _ $outbuf
         END END;
   RETURN END.
(outb) PROCEDURE (outmp,outbc);                                    20B
   % output binary code word %
  %+TENEX%                                                         20B2
      IF tonls THEN BEGIN
         IF outbc .A 1B THEN outmp.RH_ outmp.RH+relocc;
         IF outbc .A 2B THEN outmp.LH_ outmp.LH+relocc;
         out36(outmp) END
      ELSE %+TENEX%
  BEGIN
      IF outbp=0 THEN BEGIN
         outbw _ lc; BUMP outbp; outb2_2B11;
         outb2p _ ob2pwd END;
      outbw[outbp] _ outmp;
      |outb2p_outbc;
      BUMP outbp; IF 18= outbp THEN purgeb() END;
   RETURN END.
(fixup) PROCEDURE(linka, aval, cb);                                20C
   % (address of link, value, control bits %
  aval _ aval.RH; % use only low order 18 bits %
  %+TENEX%                                                         20C3
      IF tonls THEN BEGIN
         IF cb .A 1=1 THEN aval _ aval + relocc;
         x _ linka + outbfa;
         LOOP
            IF (x_ ([x]:= ([x] .A 18M6 ).V aval) .A 18M)
               =0 THEN EXIT
            ELSE x _ x-relocc+outbfa END
      ELSE %+TENEX%
  BEGIN
      x _ linka; !LSH x,18;
      fixbfr(x .V aval, cb) END;
   RETURN END.
(fixbfr) PROCEDURE (fixw,fixbc);                                   20D
   % store in fixup buffer - args like outb %
   fixbw[fixbp] _ fixw;
   |fixb2p_fixbc;
   BUMP fixbp;
   IF 18>fixbp THEN RETURN;
   purgeb(); purgef();
   RETURN END.
(purgeb) PROCEDURE;                                                20E
   % purge binary code buffer %
   IF outbp=0 THEN RETURN;
   out36(186+outbp);
   out36(outb2);
   temp _ outbp; outbp _ 0;
   WHILE outbp<temp DO BEGIN
      out36(outbw[outbp]);
      BUMP outbp END;
```

```
    outbp_ 0; RETURN END.
  (purgef) PROCEDURE;                                                    20F
    % purge fixup buffer %
    IF fixbc=0 THEN RETURN;
    out36(fixtyp+fixbp);
    out36(fixb2);
    temp _ fixbp; fixbp _ 0;
    WHILE fixbp<temp DO BEGIN
      out36(fixbw[fixbp]);
      BUMP fixbp END;
    fixbp_fixb2_ 0; fixb2p_ fb2pwd;  RETURN END.
% calls, returns %
  (savlb) PROCEDURE;                                                     21A
    % save generated labels %
    !PUSH m,gn1; !PUSH m,gn2; !PUSH m,gn;
    gn1 _ gn2 _ gn _ 0;
    RETURN END.
  (qcoit) PROCEDURE;                                                     21E
    % call subnode or output value %
    IF [pa] .A flgmsk # ptrflg THEN BEGIN
      IF [pa] .A ssmsk # ssflg THEN !CERR 1006
      ELSE BEGIN BUMP [M.RH-1]; qpav(); RETURN; END END;
    !PUSH k,@pa;
   % move return from L10 stack to Mstack %
     S _ M;
     !POP S,M;
     !POP S,a1;
    !PUSH m,a1; GOTO callr END.
  (gopop) PROCEDURE;                                                     21C
    % call output rule -- collapse tree upon return %
   (gopop1): % same as gopop, but can GOTO it %                          21C2
    !PUSH n,nmark; nmark _ n;
   % move return from L10 stack to Mstack %
     S _ M;
     !POP S,M;
     !POP S,a1;
     !PUSH m,a1;
    !PUSH m,=trclap;
    (callr): me _ [k]; top_me; GOTO [[me]];                              21C6
   (gochk): % check bit on makenode UUO for GO condition %               21C7
    IF .gbit THEN GOTO gopop1 ELSE RETURN END.
  (labrst) PROCEDURE;                                                    21D
    % restore labels %
    WHILE (gn:=gn-1)>0 DO BEGIN
      !POP g,a2; IF a2<0 THEN !CERR 1000 END;
    !POP m,gn; !POP m,gn2; !POP m,gn1; RETURN END.
 (rtnqqq) PROCEDURE; % meta return labels here %                         21E
    (rtnlt):                                                             21E1
      labrst();
      (rtnt): BUMP [m];                                                  21E1F
      (rtnf): !POPJ m,;                                                  21E1G
      (rtnlf): labrst(); !POPJ m,;                                       21E1H
    (ortnlt):                                                            21E2
      temp _ a1; labrst(); a1 _ temp;
      (ortnt): !AOS 0(m);                                                21E2F
     (ortnf): !POP k,temp; !MOVE me,0(k); !MOVEM me,top; !POPJ m,;       21E2G
```

```
      (ortnlf): labrst(); GOTO ortnf;                                    21E2C
    (trclap):                                                            21E3
      %collapse tree (pop mark and reset n to previous mark)%
      %preserve skipping%
      !SKIPA; !AOS 0(m); !MOVE n,nmark; !POP n,nmark; !MOVE
      n,nmark;
      !POPJ m,;
    (bkuprl):                                                            21E4
      brstr();
      labrst();
      !POPJ m, ;
      (bkupr): brstr(); !POPJ m,0;  END.                                 21E4C
  EXTERNAL rtnt, rtnf, rtnlt, ortnlt, rtnlf, ortnt, ortnf, ortnlf, bkupr,
 bkuprl;
% stack control %
  (qppj) PROCEDURE;                                                      22A
    % pcp & jump after ritm %
    !POP m,t; !POP m,top;
    (prj): % change return to UUO address !!!!!! %                       22A3
      !HRR a1,pa; !HRRM a1,-1(M);
    RETURN END.
  (mknk) PROCEDURE;                                                      22E
    % mark n and k stacks %
    !PUSH n,nmark; nmark _ n;
    !PUSH k,kmark; kmark _ k; RETURN END.
% code production %
  (produce) PROCEDURE;                                                   23A
    IF link1 THEN BEGIN bcw _ bcw .A 2; w _ w .A 18M6 END;
    outb(w,bcw); bcw_w_link1_0; BUMP lc;
    RETURN END.
  (lit) PROCEDURE;                                                       23E
    % produce literal %
    litbcw _ bcw;
    IF bcw = 1 THEN !CERR 1004;
    litw _ w; w_bcw_udefrf_litsfg _ 0; RETURN END.
  (litpd) PROCEDURE;                                                     23C
    LOCAL litpd1;
    % finish literal %
    litpd1 _ c; % save it. use a reg for speed %
    IF udefrf THEN w _ (w .A 18M6) .V udefrf;
    litv[litrn] _ w; c_litr[litrn]_0;
    UNTIL litv[litrn]=litv[c]
        AND ( litr[c]=0
            OR ( (IF udefrf THEN .ludfrf ELSE NOT .ludfrf)
                AND .lbcw=bcw AND .litstr=litsfg))
        DO BUMP c;
    IF c = litrn THEN BEGIN
        litr[c] _ lc;
        .lbcw _ bcw; .ludfrf _ IF udefrf THEN 1 ELSE 0;
        .litstr _ IF litsfg THEN 1 ELSE 0;
        BUMP litrn,link1; IF litrnx <= litrn THEN !SERR 14 END
    ELSE BEGIN
        a1 _ litw; !HRR a1,litr(c); litw _ a1;
        litr[c] _ litr[c] .A 18M6 .V lc END;
    bcw _ litbcw .V 1; udefrf _ 1; w_ litw;
    c _ litpd1; % restore %
```

```
  RETURN END.
(qpgn) PROCEDURE;                                                    23D
   % produce generated label %
   IF [pa] = 0 THEN qgen(); bcw _ 1; x _ [pa];
   IF [x] = 0 THEN BUMP link1; % first link abs %
   IF [x] > 0 THEN w _ w+[x] % relocatable %
   ELSE w_ w-([x]:=-lc); RETURN END.
(qpsr) PROCEDURE;                                                    23E
   %produce string, hash # in [pa]%
   x _ [pa]; psrs(); RETURN END.
(psrs) PROCEDURE;                                                    23F
   % produce string - hash in x %
  LOCAL psrcnt, psrptr, psrwd, psrcn, psrp;
   IF udefrf=0 THEN BEGIN
      litsfg_1; udefrf_x .A 18M; RETURN END;
  IF .prefx THEN
     BEGIN
     psrcnt _ .htlx-1;
     psrptr _ .htpx+1+$ss + 350788;
     WHILE psrcnt > 0 DO
        BEGIN
        psrp _ $psrwd + 440788;
        psrwd _ 0;
        FOR psrcn _ 1 UP UNTIL > 5 DO
           IF (psrcnt:=psrcnt-1) > 0 THEN |psrp _ |psrptr
           ELSE |psrp _ 0;
        outb(psrwd,0);
        BUMP lc;
        END;
     IF psrcnt=0 THEN
        BEGIN
        outb(0,0); % ensure zero at end %
        BUMP lc;
        END;
     END
  ELSE
     BEGIN
      psrcnt _ (.htlx+4)/5;
      psrptr _ .htpx+1+$ss;
      WHILE (psrcnt:=psrcnt-1)>0 DO BEGIN
         outb([psrptr],0); BUMP psrptr,lc END;
      % ensure null at end of string %
      IF (.htlx/5)*5=.htlx THEN BEGIN
         outb(0,0); BUMP lc END;
     END;
   RETURN END.
(qpas) PROCEDURE;                                                    23G
   % produce string - addressed by pop %
   qentr(); psrs(); RETURN END.
(qgen) PROCEDURE;                                                    23H
   % generate a label %
   BUMP gn;
   !PUSH g,=0; [pa] _ (g .A 18M) .V labflg; RETURN END.
(pdate) PROCEDURE;                                                   23I
   % produce date string %
   RETURN END.
```

```
  (advlc) PROCEDURE(atemp);                                          23J
    % advance lc (BSS) %
    lc _ lc + atemp;
    %+TENEX%                                                         23J3
       IF tonls THEN WHILE (atemp_atemp-1)>=0 DO out36(0)
       ELSE %+TENEX%
     purgeb();
     RETURN END.
  (qpav) PROCEDURE;                                                  23K
     % produce value of symbol - addresses hash # %
     qssck(); !HRRZ x,@pa;
     IF NOT .linked AND udefrf#0 THEN BEGIN
        link1_.linked _ 1; htv[x] _ 0 END;
     IF .gbit THEN w _ w - htv[x] ELSE w _ w + htv[x];
     IF NOT .defned THEN BEGIN
        IF (udefrf:=x)#0 THEN htv[x] _ lc;
        IF (bcw:=1) = 1 THEN !CERR 1004 END
     ELSE IF  .reloca AND (bcw:=1) =1 THEN !CERR 1004;
     RETURN END.
% definitions %
  (qdfl) PROCEDURE;                                                  24A
     % define addressed value to be LC %
     x _ [pa]; .reloca _ 1;
     define(lc);  RETURN END.
  (qdfv) PROCEDURE;                                                  24B
     % define symbol given value %
     x _ [pa]; define(a1); RETURN END.
  (define) PROCEDURE(dtemp);                                         24C
     % define a symbol given (value, hash #)%
     IF .defned THEN BEGIN %mulitple definition%
        IF htv[x]#dtemp THEN BEGIN
           tcrlf();
           htv[x] _ dtemp;
           wstrh(x);
           wstr($" redefined at ",-1); locw() END END
     ELSE IF NOT .linked THEN BEGIN
        .linked _ .defned _ 1;
        htv[x] _ dtemp END
     ELSE BEGIN
        .defned _ 1;
        fixup(htv[x]:=dtemp,dtemp,IF .reloca THEN 3 ELSE 2)
       END;
     RETURN END.
  (qqgn) PROCEDURE;                                                  24D
    LOCAL savt;
     % define generated label %
    savt _ t;
     temp _ a1;
     IF NOT .gbit THEN temp _ lc;
     IF [pa] = 0 THEN qgen(); x _ [pa];
     IF [x] = 0 THEN [x] _ temp
     ELSE BEGIN
        IF (a2_[x]:=temp)<0 THEN fixup(-a2,temp,3) END;
     t _savt;
     RETURN END.
% tree testing support %
```

```
    (qitmg) PROCEDURE; % caution: t is really A3 %                          25
      LOCAL item, savt;
       % test for generated label item %
       !POP t,item; savt _ t;
       IF item .A flgmsk # labflg THEN BEGIN t _ savt; GOTO prj; END;
       gn[.regn3] _ item; t _ savt; RETURN END.
    (qcntst) PROCEDURE; % caution: t is really A3 %                         25
       % item count test %
       t _ top; !HLRZ a2,0(t); !HRL t,a2;
       IF a2 # .regn4 THEN GOTO prj;
       BUMP DOWN t;
       RETURN END.
    (qritm) PROCEDURE; % caution: t is really A3 %                          25
      LOCAL savt;
       % recursive item - subnode structure test %
       !POP t,x; savt _ t;
      IF x .A flgmsk # ptrflg THEN
         BEGIN t _ savt; RETURN; END;
       pa_pa .A 18M; IF pa#0 AND [x] .A 18M # pa THEN
         BEGIN t _ savt; RETURN; END;
       !PUSH m,top; t _ savt; !PUSH m,t; top_x;
      % SKIP RETURN % !AOS -1(M); RETURN;
      END.
    (qnamt) PROCEDURE; % caution: t is really A3 %                          25
      LOCAL savt;
       % sub node name test %
       !POP t,x; savt _ t;
      IF x .A flgmsk # ptrflg THEN
         BEGIN t _ savt; RETURN; END;
       pa_pa .A 18M; IF pa#0 AND [x] .A 18M # pa THEN
         BEGIN t _ savt; RETURN; END;
      t _ savt;
      % SKIP RETURN % !AOS -1(M); RETURN;
      END.
    (qstitm) PROCEDURE; % caution: t is really A3 %                         25
      LOCAL savt;
       % string item test %
       !POP t,x; savt _ t;
      IF x .A ssmsk # ssflg THEN
         BEGIN t _ savt; RETURN; END;
       !HRRZ x,x;
       IF [pa] < 0 THEN BEGIN
          IF -[pa] = x THEN
            BEGIN t _ savt; !AOS -1(M); % SKIP % RETURN; END;
        t _ savt; RETURN;
        END;
       IF NOT streqq(pa,[pa] .A 18M) THEN
         BEGIN t _ savt; RETURN; END;
      t _ savt; !AOS -1(M); % SKIP % RETURN;
      END.
  % node referencing support %
    (qget) PROCEDURE;                                                       26
       % get subnode of given node ( node pointer) %
       IF x .A flgmsk # ptrflg THEN !CERR 1001;
       x _ [pa]; RETURN END.
    (qssck) PROCEDURE;                                                      26
```

```
    % ss flag check %
    IF [pa] .A ssmsk # ssflg THEN !CERR 1003;
    RETURN END.
% node construction support - need to save A3(=t) here %
  (qndlb) PROCEDURE;                                                    27A
    % label a new node %
    x _ (n - t) .A 18M;
    (mkgo):                                                             27A3
      % push node word and go -count in x %
      !HRLZ x,x; !HRR x,pa;
      !PUSH n,x; !HRRZ x,n; x_x .V ptrflg; !PUSH k,x;
      GOTO gochk
  END.
  (qmkls) PROCEDURE;                                                    27B
    % make list of subnodes %
    % 0 means a $ sequence %
    !HRRZ a2,pa;
    IF a2=0 THEN BEGIN WHILE k>kmark DO BEGIN
      !POP k,a1; !PUSH n,a1 END; !POP k,kmark END
    ELSE DO BEGIN !POP k,a1; !PUSH n,a1 END WHILE (a2_a2-1)>0;
    RETURN END.
  (qmkd) PROCEDURE;                                                     27C
    % make node of n things %
    x _ .regn3;
   a2 _ x;
    WHILE (a2_a2-1)>=0 DO BEGIN
      !POP k,a1; !PUSH n,a1 END;
    GOTO mkgo END.
  (qmkdd) PROCEDURE;                                                    27D
    % make a node of $ things %
    x _ 0;
    WHILE k>kmark DO BEGIN
      !POP k,a1; !PUSH n,a1; BUMP x END; !POP k,kmark;
    GOTO mkgo END.
  (qstrf) PROCEDURE;                                                    27E
    LOCAL savt;
    % string reference %
   savt _ t;
    qentr(); !PUSH n,x;
   t _ savt;
    RETURN END.
% symbol table production %
  (tables) PROCEDURE; % output tables for compiler %                    28A
    literals();
    symblt();
    RETURN END.
  (literals) PROCEDURE; % output literal table %                        28B
    LOCAL val, litsav;
    % literals %
   litsav _ c;
   c _ 0;
    WHILE c < litrn DO BEGIN
      fixup(litr[c] .A 18M,lc,3);
      IF .litstr THEN BEGIN
        x _ litv[c] .A 18M;
       val _ .htlx;
```

```
        IF .prefx THEN val _ val-1; % remove prefix char %
        val.LH _ val;
         outb(val,0); BUMP lc;
         csrs(); END
      ELSE BEGIN
         IF .ludfrf THEN BEGIN
            % make literal part of a link %
            x _ litv[c] := litv[c] .A 18M6;
            IF NOT .defned THEN BEGIN
               IF NOT .linked THEN BEGIN
                  htv[x] _ .lbcw _ 0; .linked _ 1;
                  .ludfrf _ 1 END;
               val _ htv[x]:=lc END
            ELSE BEGIN
               val _ htv[x]; .lbcw _ (IF .relcca THEN 1 ELSE
               0) END;
            litv[c] _ val .V litv[c] END;
         outb(litv[c],.lbcw); BUMP lc END;
      BUMP c END;
   RETURN END.
(symblt) PROCEDURE;                                              28C
   % write  symbols %
   CALL purgeb; CALL purgef; fixtyp _ 286; x _ 0;
   WHILE x <= htmx DO BEGIN
      IF htp[x] # 0 AND NOT .noddt  AND .linked THEN dump1();
      BUMP x END;
   CALL purgef; RETURN END.
(dump1) PROCEDURE;                                               28D
  LOCAL dmpval;
   % dump one radix 50 symbol given hash # %
   getrad();
   rad _ rad .V (
      IF NOT .defned AND .linked THEN gblrq
      ELSE IF .extern THEN extsy ELSE locsy);
   temp _ IF .reloca THEN 1
      ELSE IF NOT .defned AND .linked THEN 1 ELSE 0;
   fixbfr(rad,0);
   dmpval _ htv[x];
  IF .cpcode THEN
     BEGIN
     a1 _ dmpval; !LSH a1,27; dmpval _ a1;
     END;
   fixbfr(dmpval,temp);
   RETURN END.
(getrad) PROCEDURE;                                              28E
   LOCAL cptr;
   % get radix50 of symbol (hash in x) %
   cptr _ (.htpx+$ss) .V chrp;
   temp2 _ rad _ 0;
   temp _ IF .htlx>6 THEN 6 ELSE .htlx;
   WHILE temp2<temp DO BEGIN radix50(|cptr); BUMP temp2 END;
   RETURN END.
(pgname) PROCEDURE;                                              28F
   % produce program-name block for loader %
   IF tonls THEN RETURN;
   getrad(); out36(6B6+1);
```

```
    out36(0); out36(rad); RETURN END.
  (radix50) PROCEDURE(radtmp);                                          28G
    % convert char code & multipy %
    radtmp _ (radtmp .A 177B) -40B;
    IF radtmp > 40B THEN radtmp _ radtmp -26B
      ELSE IF radtmp > 0 THEN radtmp _ radtmp -17B;
    IF radtmp > 47B THEN radtmp _ radtmp -40B;
    rad _ rad*50B + radtmp;
    RETURN END.
% initialization %
  (main) PROCEDURE;                                                     29A
    (envect): GOTO l10start; GOTO l10start;                             29A1
    (initll):                                                           29A2
     % starting code %
       S _ -gstksz; !HRL S,S; !HRRI S,gstack; M _ S; uuo _krummy;
       symhide _ symkeep;
       &inproc _ $intxt;
        htcnt _ 0; ssp1_ sspp _ $ss .V chrp; wstrf(); CALL opnfl;
       rg2 _ 2B6 .V $envect; rg1 _ 4B5; !JSYS sevec;
       initl(0,0); CALL recops;
       startup _ $startp; recover _ $finish;
       finish(); END.
  (startp) PROCEDURE; % starting procedure %                            29B
    frmnls _0; uuo _krummy;
    savlm _ lm;
    &inproc _ $intxt;
    savr(); rg1 _ 4B5; !HRRI rg2,chntab; !HRLI rg2,levtab;
    !JSYS sir; !JSYS eir; rg1 _ 4B5; rg2 _ 406B6;
    !JSYS aic; rstr();
    brkloc _ $lv1wd;
    chntab[9] _ 1B6 .V $stkint;
    chntab[15] _ 1B6 .V $iglint;
    chntab[16] _ 1B6 .V $memint;
    CALL opnfl; initl(0,0); GOTO begin END.
  (nstart) PROCEDURE(ofn,chrad,obf,symt,bkloc,byc,byc);                 29C
    % entry from NLS %
    savlm _ lm; %save mark for later return to nls %
    IF ofn=-1 THEN BUMP tonls ELSE fnumo_ofn;
    savpcp _ uuo := krummy;
    &chradr _ chrad; outbfa _ obf; %IF symt THEN getsym(symt);%
    brkloc _ bkloc;
    &inproc _ $innls;
    frmnls _ 1; initl(byp,byc);
    IF tonls THEN BEGIN
       outcnt _ [outbfa]; relocc _ outbfa;
       outp _ outbfa .V 4444B8 END;
    GOTO begin END.
  (getsym) PROCEDURE(symadr);                                           29D
    WHILE [symadr]#0 DO BEGIN
       pa _ symadr; entrs(entrsr());
       symadr _ symadr + ([symadr]+5)/5+1;
       a1 _ [symadr]; !DFV x; BUMP symadr END;
    RETURN END.
  (popapart) PROCEDURE;                                                 29E
    (ppaprt): % preserve a1 and a3 %                                    29E1
     !MOVE a2,40B; !MOVEM a2,pa;
```

```
    !HRRI a2,0;
    !ROT a2,9;
    !JRST @opotab(a2);
     (krummy): !PUSHJ S,ppaprt;                                    29E6
    END.
  (initl) PROCEDURE(ptr, cnt);                                     29F
    LOCAL i1;
     %initialize everything%
    FOR i1 _ 0 UP UNTIL > $qrsz DO
       ring[i1] _ 0;
    wstrf(strbfn _ 0); % init tty output buffer %
     outbp_fixbp_fixb2_lincnt_htcntl_htcntm_bcw_0;
    linebr _ TRUE;
     fixb2p _ fb2pwd;
     w_gn1_gn2_gn_ sspc _ 0;
     lc _ link1 _ eof  _ outpg _ 0;
     lochtp_ -1;
     lcc_pcc_udefrf _ 1;
    inpgno _ -1; inporg _ $inpbuf;
    inbcnt _ cnt; inbptr _ ptr;
    inppgo _ $inpbuf / pgsize;
     rsiz _ $qrsz; mrsiz _ -$qrsz; modrsz _ $qrsz-1;
     numba _ 10;
     iwpx _ $ring+rsiz-1;
     htmx _ htamsk;
     htlim _ htamsk -20;
     litrn _ 0; litrnx _ litx;
     m_msx; n_nsx; k_ksx; g_gsx;
     fixtyp _ 1086; mknk();
    %-TENEX%  IF NOT tonls THEN BEGIN                              29F23
       !OUTPUT chnout,0; BUMP outp END; %-TENEX%
    outcnt _ 512; outp _ $outbuf;
     incnt _ 0;
     a4_7; !DPB a4,inpsiz;
     IF NOT tonls THEN BEGIN out36(486); out36(0) END;
     CALL wprep; CALL inc; CALL delb;
    savr(); !JSYS time; rtime _ rg1;
    rg1 _ 485;  !JSYS runtm; xtime _ rg1;
    rstr();
     RETURN END.
  (redops) PROCEDURE;                                              29C
     % read ops for symbol initialization %
    (redop1):                                                      29G2
     uid(); !SKIPN mreg; RETURN; num(); !SKIPN mreg; RETURN;
        !MKD 12B,iniqy; !CERR 0; GOTO redop1;
     (iniqy): !MOVE a1,-2(me); !DFV 0,-1(me);                      29G4
     !MOVE x,-1(me); !HRRZI a1,1; !DPB a1,noddt; !DPB a1,opcode;
     !CAIN c,"|; advsk(); GOTO ortnt; a1 _ 0; !DPB a1,noddt;
     a1 _ 1; !DPB a1,extern; GOTO ortnt END.
FINISH of TREE META LIBRARY
```

Libe DOP

```
FILE dlibe   % <ARCSUBSYS>XL10  <nineporgen>dlibe % % (arcsubsys,xL10,)
<nineporgen,dlibe,> %

ALLOW!
% registers %
   REF crersr, mersig, makdc, makdtc, makid, getirt;
   REGISTER
   creg=4,        % current input character %
   kreg=3,        % kstack %
   mereg=8,       % pointer to running node in output rule %
   mreg=5,        % mstack %
   nreg=7,        % nstack %
   rp = 9,
   treg=12,       % node testing scratch%
   rg1=1, rg2=2, %fixed register numbers%
   xreg=1;        % hash index : node testing % generation %
SET CALL1=478;
(getinfo) PROCEDURE;                                                    5
   typ _ ntv[hashnm].type; ix _ ntv[hashnm].indx;
   tmax _ cmax[ix];  tmin _ cmin[ix];
   RETURN END.
(lodval) PROCEDURE;                                                     6
   exnval _ avalue[ix]; RETURN END.
(reqsav) PROCEDURE( array REF);                                         7
   array[1] _ R1;
   array[2] _ R2;
   array[3] _ R3;
   array[4] _ R4;
   array[5] _ R5;
   array[6] _ R6;
   array[7] _ R7;
   array[8] _ R8;
   array[9] _ R9;
   array[10] _ R10;
   array[11] _ R11;
   array[12] _ R12;
   array[13] _ R13;
   RETURN;
   END.
(regrstr) PROCEDURE( array REF);                                        8
   R1 _ array[1];
   R2 _ array[2];
   R3 _ array[3];
   R4 _ array[4];
   R5 _ array[5];
   R6 _ array[6];
   R7 _ array[7];
   R8 _ array[8];
   R9 _ array[9];
   R10 _ array[10];
   R11 _ array[11];
   R12 _ array[12];
   R13 _ array[13];
   RETURN;
   END.
(getsig) PROCEDURE;                                                     9
```

```
    LOCAL adr;
    (array) [14];                                                    9E
    REF adr;
    regsav($array);
    str0d.chradr _ 5*(1+ (&adr _ maksid()) );
    str0d.l _ adr.RH;
    expval _ $str0d;
    regrstr($array);
    RETURN;
    END.

(getdc) PROCEDURE;                                                   10
    LOCAL adr;
    (array) [14];                                                    10E
    REF adr;
    regsav($array);
    str0d.chradr _ 5*(1+ (&adr _ makdc()) );
    str0d.l _ adr.RH;
    expval _ $str0d;
    regrstr($array);
    RETURN;
    END.

(getdtc) PROCEDURE;                                                  11
    LOCAL adr;
    (array) [14];                                                    11E
    REF adr;
    regsav($array);
    str0d.chradr _ 5*(1+ (&adr _ makdtc()) );
    str0d.l _ adr.RH;
    expval _ $str0d;
    regrstr($array);
    RETURN;
    END.

(getid) PROCEDURE;                                                   12
    LOCAL adr;
    (array) [14];                                                    12E
    REF adr;
    regsav($array);
    str0d.chradr _ 5*(1+ (&adr _ makid()) );
    str0d.l _ adr.RH;
    expval _ $str0d;
    regrstr($array);
    RETURN;
    END.

(idstuff) PROCEDURE;                                                 13
    xreg _ expval;
    expval _ [$ss + .htpx + 1];
    RETURN;
    END.

(idtst) PROCEDURE;                                                   14
    LOCAL var;
    (array) [14];                                                    14E
```

```
    regsav(Sarray);
    var _ getint();
    regrstr(Sarray);
    RETURN( var = cid );
    END.

(cvdate) PROCEDURE;                                                    15
    %+TENEX%                                                           15A
    % converts ( DATE ) to a number on kreg stack %
    LOCAL savr1, savr2;
    IF creg # *( THEN RETURN [FALSE];
    inc(); celb();
    WHILE creg # *) DO inc(entrc());
    |sspp _ 0;
    inc(); celb();
    savr1 _ rg1; savr2 _ rg2;
    rg1 _ ssp1; rg2 _ 0;
    IF SKIP !JSYS 2210 %date conversion% THEN
       BEGIN
       rg1 _ 112000011B;;                                                15K2
       IF SKIP !CALLI rg1,41B % returns info on operating system in use % THEN
                                                                         15K
          BEGIN                                                          15K3
          IF rg1 = 400C0B % TOPS20 % THEN                                15K3E
             BEGIN                                                       15K3B
             % Convert time to TENEX format %
                % get time ( in fraction of days * 2**18 ) into right half of
                register 1.  Left half of r2 has days since system 0. %
                   !BPRZ rg1,rg2;
                % multiply by 86,400: number of seconds in a day %
                   !IMULI rg1,2506005;
                % left half of one now has number of seconds in day * 2 ** 18:
                but round up if right half indicates more than 1/2 second
                remains in right half %
                   !ADDI rg1,400000B;
                % move tenex time from left half of 1 to right half of 2 %
                   !HLR rg2,rg1;
             END;                                                        15K3B
          % Register 2 is now in TENEX format date and time %
          !PUSH kreg,rg2;
          rg1 _ savr1; rg2 _ savr2;
          sspp _ ssp1;
          sspc _ 0;
          %+TENEX%                                                       15K3
          RETURN [TRUE];
          END;                                                           15K3
       % ELSE no error return %
       END;
    % error: restore registers and return FALSE %
       rg1 _ savr1; rg2 _ savr2;
       sspp _ ssp1;
       sspc _ 0;
       RETURN [FALSE];
    END.
(staval) PROCEDURE;                                                    1
    avalue[ix] _ expval; RETURN END.
```

```
(genend) PROCEDURE;                                                    1
   [rscan](-1); RETURN END.
(ststrl) PROCEDURE;                                                    1
   % store string length %
   [dvalue[ix]].l _ expval; RETURN END.
(pushstr) PROCEDURE;                                                   1
   % push string name onto string name stack %
   [expval].cpos _ 1; 80 _ smpskp _ smpsko;
   !PUSH 80,expval; smpskp _ 80; RETURN END.
(clrbta) PROCEDURE;                                                    2
   % clear bit array %
   [dvalue[ix]] _ 0;
   RETURN END.
(stdmak) PROCEDURE;                                                    2
   IF udfscp >= $udfsdp-1 THEN qxerr(); xreg _ hashnm;
   [udfsdp].chradr _ 5*($ss + .htpx +1);
   [udfsdp].l _ .htlx;
   udfsdp _ 2 + expval _ udfsdp;
   RETURN END.
(arystf) PROCEDURE;                                                    2
   [dvalue[ix]+index] _ expval; RETURN END.
(arystall) PROCEDURE;                                                  2
   FOR index _ arymin UP UNTIL > arymax
      DO [dvalue[ix]+index] _ expval;
   RETURN END.
(arysto) PROCEDURE;                                                    2
   starry(dvalue[ix] , index, expval);
   RETURN END.
(aryld) PROCEDURE;                                                     2
   expval _ [dvalue[ix]+index]; RETURN END.
(arylod) PROCEDURE;                                                    2
   expval _ ldarry(dvalue[ix],index); RETURN END.
(arysti) PROCEDURE;                                                    2
   FOR index _ lbnd UP UNTIL > ubnd DO arystd();
   RETURN END.
(mkpans) PROCEDURE;                                                    2
   nbrmak (expval, pn, index, 0 % not an SID %, TRUE % append spaces if
   desired %); RETURN; END.
(mkanum) PROCEDURE;                                                    2
   nbrmak (expval, numtoc, gntype, 0 % not an SID %, TRUE % append spaces if
   desired %); RETURN; END.
(setxpc) PROCEDURE;                                                    3
   xpc _ charsp (SP, bfont);
   RETURN; END.
(setxpm) PROCEDURE;                                                    3
   xpm _ charsp ('M, bfont);
   RETURN; END.
(setypl) PROCEDURE;                                                    3
   ypl _ bfont.fsize + ybl;
   RETURN; END.
(cleartabs) PROCEDURE;                                                 3
   lasttab _ -1;
   RETURN; END.
(settab) PROCEDURE;                                                    3
   IF lasttab < maxtabs THEN BEGIN
      BUMP lasttab;
```

```
      tabtable[lasttab] _ expval;
      END;
   RETURN; END.
(fixtabstops) PROCEDURE;                                                    35
   LOCAL increment;
   increment _ tabtable[lasttab] - tabtable[lasttab-1];
   UNTIL lasttab = maxtabs DO
      BEGIN
      BUMP lasttab;
      tabtable[lasttab] _
         MAX ( MIN ( tabtable[lasttab-1] + increment, tmax ), tmin );
      END;
   RETURN; END.
(colmbase) PROCEDURE;                                                       36
   LOCAL maxxdir;
   lmbase _ MIN (xmax, MAX (0, expval));
   maxxdir _ xmax - lmbase;
   IF blm > maxxdir THEN blm _ maxxdir;
   IF brm > maxxdir THEN brm _ maxxdir;
   IF flm > maxxdir THEN flm _ maxxdir;
   IF frm > maxxdir THEN frm _ maxxdir;
   IF hlm > maxxdir THEN hlm _ maxxdir;
   IF hrm > maxxdir THEN hrm _ maxxdir;
   IF hjlm > maxxdir THEN hjlm _ maxxdir;
   IF hjrm > maxxdir THEN hjrm _ maxxdir;
   IF imax > maxxdir THEN imax _ maxxdir;
   IF lm > maxxdir THEN lm _ maxxdir;
   IF rm > maxxdir THEN rm _ maxxdir;
   IF sigf > maxxdir THEN sigf _ maxxdir;
   IF snf > maxxdir THEN snf _ maxxdir;
   IF xbc > maxxdir THEN xbc _ maxxdir;
   RETURN; END.
(setfont) PROCEDURE;                                                        37
   CASE fontsize OF
      = -1 : NULL;
      < minsize : expval.fsize _ ( 1000 * fontsize + 36 ) / 72;
      >= maxsize : expval.fsize _ maxsize;
      ENDCASE expval.fsize _ fontsize;
   CASE fontface OF
      = -1 : NULL;
      <= minface : expval.fface _ minface;
      >= maxface : expval.fface _ maxface;
      ENDCASE expval.fface _ fontface;
   CASE fontstyle OF
      = -1 : NULL;
      <= minstyle : expval.fstyle _ minstyle;
      >= maxstyle : expval.fstyle _ maxstyle;
      ENDCASE expval.fstyle _ fontstyle;
   RETURN;
   END.
(setslant) PROCEDURE;                                                       38
   ldfont();
   newfont.fstyle.styslant _ IF expval THEN 1 ELSE 0;
   stfont();
   RETURN;
   END.
```

```
(setunderline) PROCEDURE;                                                    3
   ldfont();
   newfont.fstyle.styunoline _ IF expval THEN 1 ELSE 0;
   stfont();
   RETURN;
   END.
(setmono) PROCEDURE;                                                         4
   ldfont();
   newfont.fstyle.stymono _ IF expval THEN 1 ELSE 0;
   stfont();
   RETURN;
   END.
(setlight) PROCEDURE;                                                        4
   ldfont();
   newfont.fstyle.styface _ IF expval THEN 1 ELSE 0;
   stfont();
   RETURN;
   END.
(setbold) PROCEDURE;                                                         4
   ldfont();
   newfont.fstyle.styface _ IF expval THEN 2 ELSE 0;
   stfont();
   RETURN;
   END.
(setsize) PROCEDURE;                                                         4
   ldfont();
   newfont.fsize _ CASE expval OF
      < minsize : ( 1000 * expval + 36 ) / 72;
      >= maxsize : maxsize;
      ENDCASE expval;
   stfont();
   RETURN;
   END.
(setface) PROCEDURE;                                                         4
   ldfont();
   newfont.fface _ CASE expval OF
      <= minface : minface;
      >= maxface : maxface;
      ENDCASE expval;
   stfont();
   RETURN;
   END.
(setstyle) PROCEDURE;                                                        4
   ldfont();
   newfont.fstyle _ CASE expval OF
      <= minstyle : minstyle;
      >= maxstyle : maxstyle;
      ENDCASE expval;
   stfont();
   RETURN;
   END.
(ldfont) PROCEDURE;                                                          4
   IF fmthdr
      THEN newfont _ (
         IF tflag THEN tfont ELSE
         IF hjflag THEN hjfont ELSE
```

```
        IF h1flag THEN h1font ELSE
        IF h2flag THEN h2font ELSE
        IF h3flag THEN h3font ELSE
        IF h4flag THEN h4font ELSE
           error() ;
     ELSE newfont _ bfont;
  RETURN;
  END.
(stfont) PROCEDURE;                                                    47
  IF fmthcr
     THEN BEGIN
        IF fflag THEN ffont _ newfont ELSE
        IF hjflag THEN hjfont _ newfont ELSE
        IF h1flag THEN h1font _ newfont ELSE
        IF h2flag THEN h2font _ newfont ELSE
        IF h3flag THEN h3font _ newfont ELSE
        IF h4flag THEN h4font _ newfont ELSE
        error() END
     ELSE bfont _ newfont;
  chngfont _ 1;
  RETURN;
  END.
(setdefaultfonts) PROCEDURE;                                           48
  expval _ 0;
  expval.fsize _ sizedefault;
  expval.fface _ facedefault;
  expval.fstyle _ styledefault;
  CALL setfont;
  bfont _
  ffont _
  hjfont _
  h1font _
  h2font _
  h3font _
  h4font _
  sigffont _
  mcstfont _
  actfont _
  v1font _
  v2font _
  v3font _ expval;
  FOR index _ arymin UP UNTIL > arymax DO BEGIN
     pxfont[index] _ expval;
     pxnfont[index] _ expval;
     snfont[index] _ expval;
     snffont[index] _ expval;
     END;
  chngfont _ 1;
  RETURN;
  END.
FINISH
```

Li be MUF

```
(LibeMOP) FILE                                                          1
   ALLOW! % all kinds of stuff in here %
% registers %
   REGISTER
   creg=2,        % current input character %
   kreg=3,        % kstack %
   mereg=4,       % pointer to running node in output rule %
   mreg=6,        % mstack %
   nreg=7,        % nstack %
   rp = 9,
   treg=12 % A3 %,      % node testing scratch%
   xreg=1;        % hash index : node testing & generation %
% UUO'S %
   SET PPJ=4, SSCK=5, MKDD=6, NDLB=7, MKLS=115,
   STRF=12B, CNTST=13B, RITM=14B, STITM=15B, ITMG=16B,
   DOIT=17B, DFL=20B, DFV=21B, ENTR=23B,
   GET=30B, SERR=31B,
   CERR=32B, MKD=33B, NAMT=35B, BKUPJ=36B,
   TST=37B;
% Other SETs %
   EXTERNAL prj;
   EXTERNAL mkpo;
   EXTERNAL krummy, gnaport, gcall, gcall1, gcallm;
   EXTERNAL rtnt, rtnf, rtnlf, ortnt, ortnf, ortnlf, bkuprl, ortnlt, rtnlt,
   trclap, gochk, bkupr;

PAGE;
% pop branch table %
   (poptab) =                                                           7/
      (0,$error, $error, $error, $appj, $assck, $amkdd, $andlb,
      $error, $amkls, $astrf, $acntst, $aritm, $astitm,$error, $acoit,
      $error, $acfv,$error, $aentr,$error,$error,$error,$error, $aget,
      $error, $error, $amkd,$error, $anamt, $abkupj, $atst);
% string storage & hash table control %
   (entrs) PROCEDURE;                                                   8/
      % enter string indicated by sspp, sspc %
      save_creg; creg_0; symlen _ sspc;
      IF symlen>7M THEN !SERR 3;
      temp _ ((sspc+4)/5) * 5;
      WHILE sspc < temp DO entrc(); %blank fill%
      IF sspc<=5 THEN [ssp1+2]_0;
      creg _ ssp1;
      !LDB A1,cp1; !LDB A2,cp2; !LDB A3,cp3;
      !LSH A2,16; !IOR A2,A3; !ADDI A1,200401B;
      !ADDI A2,200401B; !MUL A1,A2;
      !LSHC A1,20; !IDIVI A1,483;
      !MOVE xreg,A2; htr _ A1;
      xreg _ ( xreg + htr ) .A htamsk;
      IF xreg=0 THEN xreg_1;
      BUMP htcntl;
      WHILE htp[xreg] # 0 DO BEGIN
         IF strcmp(ssp1,symlen) THEN GOTO entrs1;
         BUMP htcntm;
         xreg _ (xreg+(htr_(htr+31))) .A htamsk;
         IF xreg=0 THEN xreg_1 END;
      IF htlim <= htcnt THEN !SERR 1;
```

```
    .htpx _ ssp1-$ss; .htlx _ symlen; .htax _ iritat;
    ssp1 _ sspp; BUMP htcnt;
    (entrs1): sspc _ 0; sspc _ ssp1; creg_savc;                          8A2(
    RETURN
    END.
(checks) PROCEDURE;                                                      8F
    % check if string indicated by sspp, sspc is in has table and has defined
    attribute %
    LOCAL savt;
    savt _ treg;
    savc_creg; creg_0; symlen _ sspc;
    IF symlen>7M THEN !SERR 3;
    temp _ ((sspc+4)/5) * 5;
    WHILE sspc < temp DO entrc(); %blank fill%
    IF sspc<=5 THEN [ssp1+2]_0;
    creg _ ssp1;
    !LDB A1,cp1; !LDB A2,cp2; !LDB A3,cp3;
    !LSH A2,16; !IOR A2,A3; !ADDI A1,200401B;
    !ADDI A2,200401B; !MUL A1,A2;
    !LSHC A1,20; !IDIVI A1,483;
    !MOVE xreg,A2; htr _ A1;
    xreg _ ( xreg + htr ) .A htamsk;
    IF xreg=0 THEN xreg_1;
    BUMP htcntL;
    WHILE htp[xreg] # 0 DO
       BEGIN
       IF streqq(ssp1,symlen) AND .defned AND NOT .option THEN
          BEGIN
          sspc _ 0; sspp _ ssp1; creg_savc;
          treg _ savt;
          RETURN[TRUE](TRUE); %success -- it is in hash table and defined%
          END;
       xreg _ (xreg+(htr_(htr+31))) .A htamsk;
       IF xreg=0 THEN xreg_1
       END;
    sspc _ 0; sspp _ ssp1; creg_savc;
    treg _ savt;
    RETURN[FALSE](FALSE) %failure -- not in hash table%
    END.
(streqq) PROCEDURE(snep,snec);                                           8(
    % return true if strings are equal - hash# in xreg %
    IF snec # .htlx THEN RETURN(FALSE);
    snec _ (snec+4)/5;
    snep _ (snep .A 18M) .V wrdp;
    sne2 _ (.htpx+$ss ) .V wrdp;
    DO
       IF |snep#|sne2 THEN RETURN(FALSE) WHILE (snec_snec-1)>0;
    RETURN(TRUE)
    END.
(entrc) PROCEDURE;                                                       8[
    % enter a character on end of ss %
    |sspp _ creg; BUMP sspc; IF sspp=sspx THEN !SERR 6;
    RETURN
    END.
(centr) PROCEDURE;                                                       8[
    % enter a given literal string %
```

```
    LOCAL savt;
    IF [pa]<0 THEN RETURN(xreg_ (-[pa]) .V srflg);
    savt _ treg;
    entrs(entrsr()); [pa] _ -xreg; xreg_ xreg.V srflg;
    treg _ savt;
    RETURN
    END.
(entrsr) PROCEDURE;                                                    8F
    % move literal string to ss %
    soset(); savc _ creg;
    WHILE (strc _ strc-1) >= 0 DO BEGIN
       creg _ |strptr; entrc() END;
    creg _ savc;
    RETURN
    END.
% calls, returns %
(savlb) PROCEDURE;                                                     9A
    % save generated labels %
    RETURN
    END.
(gdoit) PROCEDURE;                                                     9E
    % call subnode or output value %
    IF [pa] .A tlgmsk # ptrflg THEN BEGIN
       IF [pa] .A ssmsk # ssflg THEN !CERR 1006
       ELSE error() END;
    !PUSH kreg,dpa;
    % move return from L10 stack to mstack %
       S _ M;
       !POP S,M;
       !POP S,A1;
    !PUSH mreg,A1;
    GOTO callr
    END.
 (gopop) PROCEDURE;                                                    9C
     % call output rule -- collapse tree upon return %
    (gopop1): % same as gopop, but can GOTO it %                       9C2
     !PUSH nreg,nmark; nmark _ nreg;
    % move return from L10 stack to Mstack %
       S _ M;
       !POP S,M;
       !POP S,A1;
       !PUSH mreg,A1;
     !PUSH mreg,=trclap;
     (callr): mereg _ [kreg]; top_mereg; GOTO [[mereg]];               9C6
    (gochk): % check bit on makenode UUO for GO condition %            9C8
     IF .gbit THEN GOTO gopop1 ELSE RETURN
    END.
(labrst) PROCEDURE;                                                    9D
    % restore labels %
    RETURN
    END.
(trclp1) PROCEDURE;                                                    9H
(trclap):                                                              9H
    %collapse tree (pop mark and reset nreg to previous mark)%
    %preserve skipping%
    !SKIPA; !AOS 0(mreg); !MOVE nreg,nmark; !POP nreg,nmark; !MOVE
```

```
      nreg,nmark;
      !POP[ mreg, ;
   (rtnlt):                                                                    9C
      labrst();
      (rtrt): BUMP [kreg];                                                     9G2
      (rtnf): !POPJ mreg,;                                                     9G3
      (rtnlf): labrst(); !POPJ mreg, ;                                         9G4
   (ortnlt):                                                                   9H
      temp _ A1; labrst(): A1 _ temp;
      (ortrt): !ADD 0(mreg);                                                   9H2
      (ortnf): !POP kreg,temp; !MOVE mereg,0(kreg); !MOVEM mereg,top; !POPJ
      mreg,;                                                                   9H3
      (ortnlf): labrst(): GOTO ortnf ;                                         9H4
   (bkuprl):                                                                   9I
      labrst();
      (bkupr): brstr(); !POPJ mreg, ;                                          9I2
      END.
% stack ccrtrcl %
   (qppj) PROCEDURE;                                                          10A
      % pop & jump after ritm %
      !POP mreg,treg; !POP mreg,top;
      (prj): % change return address to UUO address %                         10A3
      !HRR A1,ca; !HRRP A1,-1(M);
      RETURN;
      END.
   (mknk) PROCEDURE;                                                          10E
      % mark nreg and kreg stacks %
      !PUSH nreg,nmark; nmark _ nreg;
      !PUSH kreg,kmark; kmark _ kreg;
      RETURN
      END.
% definitions %
   (qdfv) PROCEDURE;                                                          11A
      % define symbol given value %
      xreg _ [ca];
      define(A1);
      RETURN
      END.
   (define) PROCEDURE(dtemp);                                                 11E
      % define a symbol given (value, hash #)%
      IF .defned THEN BEGIN %mulitple definition%
         IF htv[xreg]=dtemp THEN htv[xreg] _ dtemp END
      ELSE IF NOT .linked THEN BEGIN
         .linked _ .defned _ 1;
         htv[xreg] _ dtemp END
      ELSE error();
      RETURN
      END.
% tree testing support %
   (qcntst) PROCEDURE;                                                        12A
      % item count test %
      treg _ top; !HLRZ A2,0(treg); !HRL treg,A2;
      IF A2 # .regn4 THEN GOTO prj;
      BUMP DOWN treg;
      RETURN
      END.
```

```
(qritm) PROCEDURE;                                                    12
   LOCAL savt;
   % recursive ite- - subnode structure test %
   !POP treg.xreg;
   savt _ treg;
   IF xreg .A flgmsk # ptrflg THEN
      BEGIN
      treg _ savt;
      RETURN;
      END;
   pa_pa .A 18M;
   IF pa#0 AND [xreg] .A 18M # pa THEN
      BEGIN
      treg _ savt;
      RETURN;
      END;
   !PUSH mreg,top;
   treg _ savt;
   !PUSH mreg,treg: top_xreg;
   % SKIP RETURN % !AOS -1(M); RETURN;
   END.
(qnamt) PROCEDURE;                                                    12(
   LOCAL savt;
   % sub node name test %
   !POP treg.xreg;
   savt _ treg;
   IF xreg .A flgmsk # ptrflg THEN
      BEGIN treg _ savt; RETURN; END;
   pa_pa .A 18M;
   IF pa#0 AND [xreg] .A 18M # pa THEN
      BEGIN treg _ savt; RETURN; END;
   treg _ savt;
   % SKIP RETURN% !AOS -1(M); RETURN;
   END.
(qstitm) PROCEDURE;                                                   12[
   LOCAL savt;
   % string item test %
   !POP treg,xreg;
   savt _ treg;
   IF xreg .A ssmsk # ssflg THEN
      BEGIN treg _ savt; RETURN; END;
   !HRRZ xreg.xreg;
   IF [pa] < 0 THEN BEGIN
      IF -[pa] = xreg THEN
         BEGIN treg _ savt; !AOS -1(M); % SKIP % RETURN; END;
      treg _ savt;
      RETURN END;
   IF NOT streqq(pa,[pa] .A 18M) THEN
      BEGIN treg _ savt;  RETURN; END;
   treg _ savt;
   % SKIP % !AOS -1(M); RETURN;
   END.
% node referencing support %
   (qget) PROCEDURE;                                                  13/
      % get subnode of given node ( node pointer) %
      IF xreg .A flgmsk # ptrflg THEN !CERR 1001;
```

```
    xreg _ [ra];
    RETURN
    END.
(qssck) PROCEDURE;                                                    13E
    % ss flag check %
    IF [ra] .A qsmsk # ssflg THEN !CERR 1003;
    RETURN
    END.
% node construction support %
(qndlb) PROCEDURE;                                                    14A
    % label a new node %
    xreg _ (nreg - treg) .A 18M;
    (mkgc):                                                           14A3
        % push node word and go -count in xreg %
        !HRLZ xreg,xreg; !HRR xreg,pa;
        !PUSH nreg,xreg; !HRRZ xreg,nreg; xreg_xreg .V ptrflg; !PUSH
        kreg,xreg;
        GOTO gochk
        END.
(qmkls) PROCEDURE;                                                    14E
    % make list of subnodes %
    % 0 means a $ sequence %
    !HRRZ A2,pa;
    IF A2=0 THEN BEGIN WHILE kreg>kmark DO BEGIN
        !POP kreg,A1; !PUSH nreg,A1 END; !POP kreg,kmark END
    ELSE DO BEGIN !POP kreg,A1; !PUSH nreg,A1 END WHILE (A2_A2-1)>0;
    RETURN
    END.
(qmkd) PROCEDURE;                                                     14C
    % make node of nreg things %
    xreg _ .regn3;
    A2 _ xreg;
    WHILE (A2_A2-1)>=0 DO BEGIN
        !POP kreg,A1; !PUSH nreg,A1 END;
    GOTO mkgo
    END.
(qmkdd) PROCEDURE;                                                    14D
    % make a node of $ things %
    xreg _ 0;
    WHILE kreg>kmark DO BEGIN
        !POP kreg,A1; !PUSH nreg,A1; BUMP xreg END; !POP kreg,kmark;
    GOTO mkgc
    END.
(qstrf) PROCEDURE;                                                    14E
    LOCAL savt;
    % string reference %
    savt _ treg;
    qentr(); !PUSH nreg,xreg;
    treg _ savt;
    RETURN
    END.
% initialization %
(initll) PROCEDURE;  % starting code %                                15A
    LOCAL ex1;
    % Check if bounds within processor area %
        IF bOutPrc NOT IN [bPrcArea, ePrcArea] OR eOutPrc NOT IN [bPrcArea,
```

```
      ePrcArea] THEN
         BEGIN
         !psout($loaderr);
         !halt*();
         END;
   % byte pointers to output mapped buffer %
      startbp _ 440700B6 + $buffer;
      bufbp   _ 440700B6 + $buffer;
      endbp   _ 01070086 + $buffer+10008-1;
      pmapfork_ 40000086 + ($buffer/10008);
   % set up string and array descriptors %
      strstd.chradr _ 5 * $strstr;
      str0d.chradr  _ 5 * $str0st;
      str1d.chradr  _ 5 * $str1st;
      hjstd.chradr  _ 5 * $hjsst;
      h1std.chradr  _ 5 * $h1sst;
      h2std.chradr  _ 5 * $h2sst;
      h3std.chradr  _ 5 * $h3sst;
      h4std.chradr  _ 5 * $h4sst;
      fstd.chradr   _ 5 * $fsst;
      mcfstd.chradr _ 5 * $mcfsst;
      hjestd.chradr  _ 5 * $hjesst;
      h1estd.chradr  _ 5 * $h1esst;
      h2estd.chradr  _ 5 * $h2esst;
      h3estd.chradr  _ 5 * $h3esst;
      h4estd.chradr  _ 5 * $h4esst;
      festd.chradr   _ 5 * $fesst;
      mcfestd.chradr _ 5 * $mcfesst;
      mstd.chradr _ 5 * $msst;
      mcmstd.chradr _ 5 * $mcmst;
      romand.chradr _ 5 * $roman;
      crstrd.chradr _ 5 * $crstr;
      spstrd.chradr _ 5 * $spstr;
      tbstrd.chradr _ 5 * $tbstr;
      bigmsd.chradr _ 5 * $bigsst;
   FOR ex1 _ 0 UP UNTIL > 177B DO code[ex1] _ ex1;
   uuo _krummy: usual _ 3;
   htcrt _ 0; ssa1_ sspp _ $ss .V chrp: CALL opnfl;
   initl();
   wprep(): inc(); delb();
   !PUSHJ mreg,rdaval; error();
   opapchr( $mstd, "|");
   opapchr( $mcmstd, "|");
   !JSYS haltf;
   (stkint): [brkloc] _ $error1; !JSYS debrk;                          15A14
   (finish): RETURN;                                                   15A15
   END.
(loaderr) = ( % string contruction NOT in OP! Used in PSOUT if OP Not in
Bounds %                                                               15E
   4274562677448, %Error %
   2032356202308, % in l%
   677024464734B, %oadin%
   634000000008); %o%
EXTERNAL stkint, finish;
(cllp) PROCEDURE;                                                      15E
   (opaprt): % preserve A1 and A3 %                                    15D1
```

```
        %get here by a !PUSHJ S,ppaprt in location 41B%
        !MOVE A2,40B;
        !MOVEM A2,pa;
        !HRRI A2,0;
        !HRLT A2,0;
        !BLT A2,@(A2);
    (krummy): !PUSHJ S,ppaprt ;                                          15D2
    END.
(initL) PROCEDURE;                                                       15E
    %initialize everything%
    snmskp _ snmskm _ (-1286 ,A 18M6) ,V $snmstk;
    htcntl _ htcntm _ 0;
    sspc _ 0;
    lcc _ pcc _ 1;
    rsiz _ grsz; mrsiz _ -grsz; modrsz _ grsz-1;
    iwpx _ $opring+rsiz-1;
    htmx _ htamsk;
    htlim _ htamsk -20;
    mreg_msx; nreg_nsx; kreg_ksx;
    mknk(); savall();
    RETURN
    END.
FINISH
```

LibeOpDInit

(dinuid) <compsrc, libeopdinit, 0323> PROCEDURE 6

```
FILE opdthings
% special identifiers for OPDINIT compiler %
ALLOW!
DECLARE idflg = 1186, uidflg = 1286;
% registers %
    REGISTER  g=0, % lstack %
    x=1, % hash index : node testing & generation %
    c=2, % current input character %
    t=12 %A3%, % node testing scratch%
    w=4, % binary word being formed %
    m=5, % mstack % k=3, % kstack % n=7, % nstack %
   mreg=6, % L10 sucess/fail register %
    me=8, % pointer to running node in output rule %
    r= 15, % libe calls & returns %
    lm=14, % L10 call stack mark %
    a1=10, a2=11, a3=12, a4=13, % accumulators %
   rg0=0, rg1=1, rg2=2, rg3=3, rg4=4, %fixed register numbers%
      rg5=5, rg6=6, rg7=7, rg8=8;
(dinuid) PROCEDURE;                                                    6
   % upper case identifier %
   IF c NOT IN ['A, 'Z] THEN RETURN [FALSE];
   inc(entrc());
   LOOP
      BEGIN
      IF c NOT IN ['A, 'Z] THEN
         IF c NOT IN ['a, 'z] THEN
            IF c NOT IN ['0, '9] THEN EXIT LOOP;
      inc(entrc());
      END;
   entrs(); x_x .V uidflg; !PUSH k,x; delb();
   RETURN [TRUE] END.
 (dinid) PROCEDURE;                                                    7
    % lower case identifier %
    IF c NOT IN ['a,'z] THEN RETURN [FALSE];
   inc(entrc());
   UNTIL c NOT IN ['a,'z] AND c NOT IN ['0, '9] DO
      BEGIN
      inc(entrc());
      END;
    entrs(); x_x .V idflg; !PUSH k,x; delb();
    IF .htax=0 THEN .reloca_1;
    RETURN [TRUE] END.
FINISH
```

Rt - Data

(blcstk)	<compsrc, rt-data, 0109>	EXT STACK	8B
(btwstk)	<compsrc, rt-data, 051>	EXT STACK	7D
(catchsiz)	<compsrc, rt-data, 094>	CONSTANT =75	3B
(catsiz)	<compsrc, rt-data, 095>	CONSTANT =10	3D
(cinit)	<compsrc, rt-data, 050>	EXT	4A1
(clchng)	<compsrc, rt-data, 021>	EXT STACK	5A
(clmoty)	<compsrc, rt-data, 019>	EXT	4E4
(copyright)	<compsrc, rt-data, 0163>	STRING	2A
(flag)	<compsrc, rt-data, 011>	EXT	4B1
(gstack)	<compsrc, rt-data, 023>	EXT	6F
(gstkdesc)	<compsrc, rt-data, 0128>	EXT	6E
(gstksz)	<compsrc, rt-data, 039>	EXT CONSTANT =4608	3F
(mkrstay)	<compsrc, rt-data, 040>	EXT	4E3
(modeset)	<compsrc, rt-data, 053>	EXT	4E1
(modeshift)	<compsrc, rt-data, 054>	EXT	4E2
(nsdbpt)	<compsrc, rt-data, 017>	EXT	4C1D
(pstack)	<compsrc, rt-data, 012>	EXT	4B2
(pstksz)	<compsrc, rt-data, 038>	EXT CONSTANT =20	3G
(rplsid)	<compsrc, rt-data, 015>	EXT	4C1B
(sar)	<compsrc, rt-data, 026>	EXT STRING	7A
(scndir)	<compsrc, rt-data, 034>	EXT	4C1A
(signalsiz)	<compsrc, rt-data, 092>	CONSTANT =10	3A
(sigsiz)	<compsrc, rt-data, 093>	CONSTANT =12	3C
(sptr1)	<compsrc, rt-data, 016>	EXT	4C1C1
(sptr2)	<compsrc, rt-data, 0161>	EXT	4C1C2
(stackdecsize)	<compsrc, rt-data, 0153>	CONSTANT =10	3E
(stcwr1)	<compsrc, rt-data, 045>	EXT	4D2A
(stcwrk)	<compsrc, rt-data, 044>	EXT	4D2
(swork)	<compsrc, rt-data, 042>	EXT	4D1
(swork1)	<compsrc, rt-data, 043>	EXT	4D1A
(sysce)	<compsrc, rt-data, 0143>	EXT	6D
(syscstk)	<compsrc, rt-data, 0142>	EXT	6C
(sysge)	<compsrc, rt-data, 0141>	EXT	6B
(sysgstk)	<compsrc, rt-data, 0140>	EXT	6A

```
(RtData) FILE                                                                    1
   SET TRACE=1; % 1 to compile trace globals, 0 tc omit them %
% copyright string %
   (copyright) STRING = "COPYRIGHT by Tymshare Incorporated 1978";              2A
% Special stuff %
   (signalsiz) CONSTANT=10; % number of nested signals allowed %                3A
   (catchsiz) CONSTANT=75; % number of catchphrases that can be invoked at
   once %                                                                       3B
   (sigsiz) CONSTANT=12; % number of words in signal frame %                    3C
   (catsiz) CONSTANT=10; % number of words in catch frame %                     3D
   (stackdecsize) CONSTANT=10; % number of words in call stack descriptor %     3E
   (gstksz) EXTERNAL CONSTANT =  4608;  %length of general call stack%         3F
   (pstksz) EXTERNAL CONSTANT = 20; %length of pattern stack%                  3G
%...KEEP THESE THINGS TOGETHER -- DONT ADD NEW DATA HERE...%
   %...user related data...%
      (cinit) EXTERNAL;   % current initials of user %                         4A1
   %...string analysis stuff...%
      (flag) EXTERNAL _ 1;        %the general flag%                           4B1
      (pstack) EXTERNAL [pstksz];     %pattern stack%                          4B2
   %...editing global variables...%
      %...text editing global variables...%
         (scndir) EXTERNAL;  % analyzer compiler scan direction %              4C1A
         (rplsid) EXTERNAL;  % psid of statement being replaced in sc %        4C1B
         % for append T-string %
            (sptr1) EXTERNAL [2];                                              4C1C1
            (sptr2) EXTERNAL [2];                                              4C1C2
         (nsobpt) EXTERNAL;  % byte pointer for statement being constructed%
                                                                               4C1D
   %...often used work areas...%
      (swork) EXTERNAL; % string reading work area %                           4D1
         (swork1) EXTERNAL [6];                                                4D1A
      (stcwrk) EXTERNAL; % string construction work area %                     4D2
         (stcwr1) EXTERNAL [6];                                                4D2A
   %...set command and aptstr routine...%
      (modeset) EXTERNAL;        %  option set  %                              4E1
      (modeshift) EXTERNAL;      %  case set    %                              4E2
      (mkrstay) EXTERNAL;        %true if do not move markers%                 4E3
      (clmpty) EXTERNAL;   %reset value for clchng stack%                      4E4
% accessed in statement construction%
   (clchng) EXTERNAL STACK [30];                                               5A
% general call stack + stack descriptor%
   (sysgstk) EXTERNAL [signalsiz*sigsiz]; % signal in progress stack %          6A
   (sysge) EXTERNAL; % absolute end of signal stack %                          6B
   (syscstk) EXTERNAL [catchsiz*catsiz]; % catchphrase invoked stack %         6C
   (sysce) EXTERNAL; % absolute end of catchphrase stack %                     6D
   (gstkdesc) EXTERNAL [stackdecsize];                                         6E
   (gstack) EXTERNAL [gstksz];                                                 6F
% DECLARE's %
   (sar) EXTERNAL STRING [2000];        % DO NOT USE -used by string
   primitives%                                                                 7A
   DECLARE EXTERNAL
      %  signal info-- keep in this order!  do not add without changing
      runtime! %
         sysfrm, % current signaler's PORT ID %
         sysgcc, % "" "" saved cc-loc from his stack frame %
         syssig, % signal value%
```

```
    sysgtp, % signal type %
    sysg2, sysg3, sysg4, % signal arguments %
    syscat, % previous catchphrase pointer - for nesting signals; search
    starting location.  set in syshelp.  usec in looping over catchphrase
    stack %
    sysbot, % stack pointer of bottom of the stack for signal
    propogation: may be sysbtm, or may be higher on the stack if the ange
    of propogation is to be restricted.  cf. systrm%
    syssstk, % stack pointer to bas of signalling stack %
    nestscan, % used in sysctn.  set in sysctn, popsig.  FALSE => not a
    nested signal.  TRUE => nested signal. %
    sysgn, % signal in progress on current stack: also the number of the
    current signal since the top signal for a particular stack is the one
    in progress %
    tsysgn, % total number of signals in progress in system %
    sysinfo, % the address of the current signal on the signal stack %
% catchphrase info %
    syscpm, % catchphrase parameter %
    systloc, % termiation location, global for systrm %
% stack pointers to catchphrase stacks %
    sysctchall, % contains pointer to system catchphrase, syscatch, on
    catchphrase stack %
    syscpe, % pointer to end of catchphrase stack (dynamic end) %
    syscpt, % pointer to current catchphrase frame %
% stack pointers to signal stacks %
    syssptr, % current signal stack top pointer %
% stack pointers to call stack(s) %
    pcstack, % stack pointer to bottom of "previous" current stack %
    sysbtm, % bottom stack frame ID for catch searches: a stack pointer
    to the bottom of the current stack %
    savebtm, % used to save sysbtm around calls on sigrestore %
    sysse, % pointer for (current) stack end, used by sysctn %
% inter-procedure/coroutine communication globals %
    sysfctn, % TRUE if pcall comes from sysfctn (CONTINUE, SYSHLP, etc.);
    FALSE otherwise %
    sysbfg, % flag for signal routine - to set sysbot to restrict signal
    propogation.  TRUE only for NOTE(unwind or return).  Otherwise false
    => use sysbtm.  Used in SYSHELP %
    sysbcon, % gobal flag for syshelp/sysctn communication %
%...general global variables...%
    linlink = 0, % stack pointer to gstack (if there is more than one
    stack) or 0 %
    syshlp, % port ID for HELP %
    sysabt, % port ID for ABORT %
    sysres, % port ID for RESUME %
    sysnot, % port ID for NOTE %
    sysinv, % port ID for INVOKE %
    syscon, % port ID for CONTINUE %
    % --- %
    startup, % REF for startup procedure %
    recover, % REF for recover procedure %
    % --- %
    syswhy, % global for runtime error reasor string address %
    sysloc, % runtime error location %
    sysrip, % flag, "recover-in-progress" sec sysrcv %
    sysetc, % error-type code
```

```
            (programbug, stkoverflow, uncaughtabort) %
         % --- %
            syszpn, % saved SIGNAL for uncaught ABORT %
            syszpt, % " for SIGNALTYPE %
            syszp2, syszp3, syszp4, % " for arguments %
         tempalsave, % to preserve a1 in pcall code %
         tempmregsave, % to preserve mreg in pcall code %
         srgsav[14], % to preserve registers 0-13 in pcall code %
         srg14, % to preserve register 14 in pcall code %
         srg15, % to preserve register 15 in pcall code %
         drpcat = 0,
         %+TRACE% % trace globals %                                          7B14
            systen=0, % TRACE ENABLE FLAG, set true to trace %
            systfg=0, % trace code flag, TRUE if trace in progress %
            systjfn, % TRACE.DATA file jfn , in left half %
            systpc, % trace.data file page count %
            systmd, % mode bits %
            systpn, % core page number to store trace data into %
            systpt, % AOBJN pointer into systpn %
            systck, % jobtm clock reading %
            systr1, % place to save R1 %
            systtp, % teletype output jfn/flag for trace %
            %+TRACE%                                                        7B14K
         %...other work areas...%
            ps, pe; %cells for BETWEEN construct%
      DECLARE EXTERNAL
         %collector sorter%
            lngflg; %true if length is considered befor value in sort%
      (btwstk) EXTERNAL STACK [10];                                          7D
      %...spec stack...%
         DECLARE EXTERNAL
            spsk = $spsk,     % spec stack pointer %
            spsk1= $spsk,     % initial pointer %
            b1[2], b2[2], b3[2], b4[2], b5[2],
            spskt = $b5;      % top of spec stack %
      %...correspondence list...%  %PASSED%
         DECLARE EXTERNAL
            clhead[4],  %header, see clhdr record%
            clistaddr,  %addr of allocated corres. list block%
            clsize,  %current size of clist block%
            clstac=$clhead;  %address of clist header%
   % L10 LIST runtime globals %
      DECLARE EXTERNAL lstzone, blclst;
      (blcstk) EXTERNAL STACK [10];                                          8B
   FINISH of RtData
```

Rt - LRP

```
FILE l10lrp % (arcsubsys,xl10,) or (arcsubsys,l109,) to (L10,l10lrp,) %
   ALLOW!
% L10 LIST RUNTIME PACKAGE (part of L10RUNTIME) for PDP-10 %
% Internal Format %
   % The internal PDP-10 format of a list is:
      storwd
      list: XWD M,,L
      led1 (list element descriptor)
         ...
      ledM %
   % Where storwd is a runtime package word that is used to indicate if (and
   where) the list resides in allocated storage. A value of zero means that
   the list resides in declared storage or has not been referenced. In any
   case M designates the number of (following) words that may be used for
   elements.
      Another value storwd means
         The address of the list (itself in allocated storage).
         The original list.M stays even tho the max is larger. It is in the
         allocated list.M The list.L and the allocated list.L are the same and
         indicate the length.
      %
% Definitions %
   REF blclst; % declared in l10data %
   (ledr) RECORD % PDP-10 list element descriptor%                          41
      ledval [18], %address/value of element%
      ledubv[ ledtyp[3], ledbits[3] ], % type and user bits %
      ledalo  [1]; %element space allocated if TRUE%
   DECLARE EXTERNAL CONSTANT % descriptor types %
      %
      lnull  = 0 ,   NULL
      linteg = 2 ,   INTEGER
      lstrin = 3 ,   STRING
      llist  = 4 ,   LIST
      lblock = 1 ,   BLOCK
      %
      ldescr = 101 , % DESCRIPTOR -- never stored as type %
      lnewde = 100 ; % NEWDESCRIPTOR -- never stored as type  %
   DECLARE CONSTANT
      nulldescr = 0B;
      % ledval = 0 %
      % ledtyp = 0 %
      % ledalo = 0 %
   DECLARE CONSTANT
      blkadj=0, % result of getblk plus blkadj is first free word in block %
      lsthdr=1, % number of words in list header (stowrd) %
      storwd=18M, % index to get storwd %
      listmx = 120, % 2 times max list size (= 60 ) %
      blksiz=30; % initial size of construction workspace %
(nulist) % set a list or sublist to NULL %
   PROCEDURE (
      list REF, % address of list %
      sublist, % boolean: TRUE if e1,e2 or FALSE if entire list %
      e1, e2) ; % nullify e1 thru e2 if sublist=TRUE %
   % If "sublist" is FALSE, effectively sets list.L to zero.  Otherwise,
   discards elements "e1" through "e2", shifing forward any elements behind
   them. Written so nulist(list,FALSE) will work !! %
```

```
    LOCAL
       i, % element index %
       nlength, % new list length %
       alist REF, % allocated list address %
       top, % highest element to delete %
       bot; % lowest %
    IF list.L =0 THEN RETURN; % don't waste time %
    IF sublist THEN BEGIN bot _ MAX(1,e1); top _ MIN(e2,list.L); END
    ELSE BEGIN bot _ 1; top _ list.L; END;
    nlength _ bot-1; % number of elements in lower segment %
    % free all allocated storage in the elements to be eliminated %
    FOR i _ bot UP UNTIL > top DO
       IF top < list.L THEN % move down top elements %
          BEGIN
          wrlelm($list, lnewde, rdlelm($list, TRUE, top+1), TRUE, i);
          BUMP top;
          BUMP nlength; % one more real element %
          END
       ELSE
          BEGIN
          IF (&alist _ list[storwd]) = 0 THEN &alist _ &list;
          freelm(alist[i]); % free elements %
          END;
    list.L _ nlength;
    IF list[storwd] # 0 THEN % allocated list to take care of %
       BEGIN
       &alist _ list[storwd];
       alist.L _ nlength; % set length both places %
       lstrst($list, $alist); % move back if possible %
       END;
    RETURN;
    END.
(rdlelm) % read list element, return descriptor, element value %                    (
    PROCEDURE (
       list REF,  % address of list %
       delete, % boolean: delete element if true %
       index % list element index % ) ;
    % Returns the descriptor and value (descr, value) for (addr of L10 string,
    list, or block; or integer) of element "index" of list "list".  If "delete"
    is TRUE, the source element descriptor is replaced with a null descriptor.
    The element itself is left. This is how a "MOVE" is done. %
    LOCAL
       descr, % the descriptor %
       value, % the value %
       alist REF; % the address of list elements %
    IF (&alist _ list[storwd]) = 0 THEN &alist _ &list;
    IF index NOT IN [1,alist.L] THEN RETURN(nulldescr,0);
    descr _ alist[index] ;
    IF descr.ledtyp = linteg AND descr.ledalo THEN
       value _ [descr.ledval]
    ELSE
       value _ descr.ledval ;
    IF delete THEN
       alist[index] _ nulldescr;
    RETURN ( descr, value );
    END.
```

```
(wrlelm) % write list element %
   PROCEDURE (
      list REF, % list address %
      type, % shows what type 'value' is %
      value, % the value itself %
      replace, % boolean: TRUE= store in list[index], FALSE= append %
      index) ; % list element index %
   % If "replace" is FALSE, appends element of type "type" and value "value"
   to list "list". Otherwise, replaces element "index" with it. Written so
   that the fifth argument (index) may be omitted if replace=FALSE %
   LOCAL
      alist REF, % address of list elements %
      i, % list element index %
      size, % size of new list %
      myindex; % real list element index %
         % cannot touch index if not provided ! %
   IF (&alist _ list[storwd]) = 0 THEN &alist _ &list;
   CASE replace OF
      = FALSE: % not replacing an element %
         BEGIN
            IF alist.L >= alist.M THEN
               BEGIN
               IF (size _ MAX(blcsiz, list.L*2) ) > listmx THEN
                  sysrcv(programbug, $"excessive list overflow", sysclr());
               CASE &list OF
                  = &blclst: % update &list after len changes %
                     BEGIN
                     setllen( &list, size);
                     &list _ &blclst;
                     END;
                  ENDCASE setllen(&list, size);
               IF (&alist _ list[storwd]) = 0 THEN &alist _ &list;
               END;
            BUMP alist.L ;
            IF &alist # &list THEN list.L _ alist.L;
            myindex _ alist.L ;
         END;
      ENDCASE
         BEGIN
         CASE index OF
            <= list.L: % replacing an existing element %
               BEGIN
               freelm (alist[index]) ;
               myindex _ index;
               END ;
            ENDCASE % extend list up to this element and then append %
               BEGIN
               FOR i_list.L+1 UP UNTIL >= index DO
                  wrlelm($list, lnull, 0, FALSE);
               IF (&alist _ list[storwd]) = 0 THEN &alist _ &list;
                  % alist may have changed %
               REPEAT CASE 2 ( FALSE); % do the append %
               END;
         END ;
   alist[myindex] _ makelm ( type, value ) ;
   RETURN;
```

```
   END.
(setllen) % set list (allocated) length %                                    8
   PROCEDURE(
      list REF, % list address %
      size); % desired size %
   LOCAL
      i, % list elment index %
      alist REF,
      nlist REF; % for new list %
   % makes sure that list has max length 'size' %
   IF (&alist _ list[storwd]) = 0 THEN &alist _ &list;
   IF alist.M < size THEN
      BEGIN
      &nlist _ getlst(size);
      nlist.L _ alist.L;
      FOR i_1 UP UNTIL > alist.L DO
         nlist[i] _ alist[i];
      list[storwd] _ &nlist; % fix up stowrd %
      CASE &alist OF
         =&blclst: % from construction workspace %
            BEGIN
            frelst(&blclst);
            &blclst _ &nlist;
            END;
         = &list: NULL; % just moved to alocated sto %
         ENDCASE
            frelst($alist);
      END;
   RETURN;
   END.
(blc) % begin list construction (initialization) %                           9
   PROCEDURE;
   % push current list construction state and begin a new list. The list is
   built in allocated storage, but may not stay there. See (elc). %
   PUSH &blclst ON blcstk;
   &blclst _ getlst(blcsiz);
   blclst.L _ 0;
   RETURN;
   END.
(aplelm) % append list element (to construction workspace) %                 10
   PROCEDURE(
      type, % shows what type 'value' is %
      value); % the value to be stored %
   % Appends element of type "type" and value "value" to list in list
   construction workspace. %
   wrlelm($blclst, type, value, FALSE);
   RETURN;
   END.
(apsubl) % append sublist (or entire list) to construction workspace %      11
   PROCEDURE(
      flags, % determines mode: see below for meaning %
      list REF, % address of source list %
      e1, e2); % use list[e1] thru list[e2] %
   % flags
        bit 35
           0 - append only elements e1 through e2 of list to the worklist
```

```
        1 - append all elements of list to the worklist
      bit 34
        0 -
        1 - move
      bit 33
        0 -
        1 - copy
      bit 32
        0 -
        1 - values of source elements are stored as integers
    %
  % appends the sublist e1 thru e2, or the entire list, to the construction
  workspace. Flags indicate copy/move and sublist/all modes. %
  LOCAL
    descr, value, % for element begin appended %
    top, % last element to append %
    i; % list element index %
  IF flags .A 1 THEN % means append whole list %
    BEGIN
    i _ 1;
    top _ list.L;
    END
  ELSE
    BEGIN
    i _ MAX(1,e1);
    top _ MIN(list.L, e2);
    END;
  FOR i UP UNTIL > top DO
    BEGIN
    descr _ rdlelm($list, flags .A 2 % move %, i: value);
    wrlelm( $blclst, IF flags .A 8 THEN linteg ELSE IF flags .A 2 THEN
    lnewde ELSE ldescr, IF flags .A 8 THEN value ELSE descr, FALSE);
    END;
  RETURN;
  END.
(elc) % enc list construction and store list somewhere %                    1;
  PROCEDURE (list REF); % list address or zero %
  % If argument "list" is present (i.e. non-zero), stores list in list
  construction workspace as list "list" and releases the workspace.
  Otherwise, simply returns addr "worklist" of and responsibility for list in
  workspace. %
  LOCAL
    descr, % a new descriptor %
    i; % list element index %
  descr _ 0;
  CASE &list OF
    =0: % just return the workspace address %
      BEGIN
      descr.ledval _ &blclst;
      descr.ledalo _ TRUE;
      END;
    ENDCASE
      BEGIN
      list.L _ blclst.L;
      lstrst($list, $blclst); % return list if possible %
      descr.ledval _ &list;
```

```
        END;
    POP blcstk TO &blclst;
    descr.ledtyp _ llist;
    RETURN(descr);
    END.
(lstrst) % restore list : return to declared storage if possible %                13
    PROCEDURE (
        list REF, % declared list %
        alist REF); % allocated list %
    % move the list in 'alist' to 'list' if it will fit. if moved, release the
    storage for 'alist' %
    LOCAL
        i; % list element index %
    CASE list.M OF
        >= alist.L: % can move it %
            BEGIN
            list[storwd] _ 0;
            FOR i_1 UP UNTIL > alist.L DO
                list[i] _ alist[i];
            frelst(&alist);
            END;
        ENDCASE list[storwd] _ $alist; % point to it %
    RETURN;
    END.
(makelm) % make new list element given type and value %                           14
    PROCEDURE (
        type, % shows type of value provided %
        value REF ) ; % the value depends on type, see case stmt. %
    % makelm makes a descriptor for a new list element, including allocating
    storage and copying data, and returns it. %
    LOCAL
        descr_0, % the result %
        i, % list element index %
        size, blkadr; % size and address of block obtained %
    CASE type OF
        = lnewde: % value is a new descriptor, just use it if ok %
            IF ckdesc ( &value ) THEN descr _ &value
            ELSE sysrcv(programbug, $"bad new-descriptor for list, makelm",
            sysclr());
        = lnull: % value ignored. he wants a null element %
            descr _ nulldescr  ;
        = ldescr: % value is existing element descr. copy it %
            BEGIN
            descr _
                makelm ( &value.ledtyp, IF &value.ledtyp = linteg AND
                &value.ledalo THEN [&value.ledval] ELSE &value.ledval) ;
            descr.ledbits _ &value.ledbits;
            END;
        = linteg: % value is 36 bit integer. check size etc. %
            IF &value NOT IN [0, 18M] THEN
                BEGIN
                blkadr _ getblk ( 1, lstzone ) ;
                IF blkadr=0 THEN ABORT(listspace, $"exceed LIST allocation zone");
                [blkadr+blkadj] _ &value ;
                descr.ledval _ blkadr + blkadj ;
                descr.ledtyp _ linteg ;
```

```
            descr.ledalo _ TRUE ;
            END
         ELSE
            BEGIN
            descr.ledval _ &value ;
            descr.ledtyp _ linteg ;
            descr.ledalo _ FALSE ;
            END ;
      = lstrin: % value is string address. copy the string %
         BEGIN
         blkadr _ getstring(value.L, lstzone);
         *[blkadr]* _ *value* ;
         descr.ledval _ blkadr ;
         descr.ledtyp _ lstrin ;
         descr.ledalo _ TRUE ;
         END ;
      = llist: % value is list address. copy entire list!! %
         BEGIN
         blkadr _ getlst(value.L) ;
         [blkadr].L _ value.L;
         FOR i _ 1 UP UNTIL > value.L  DO
            [blkadr+i] _ makelm ( ldescr, value[i] ) ;
         descr.ledval _ blkadr;
         descr.ledtyp _ llist ;
         descr.ledalo _ TRUE ;
         END ;
      = lblock: % value is block address (adr of word with size) copy %
         BEGIN
         size _ value[-1].blklength -1 ;
         blkadr _ getblk ( size, lstzone ) ;
         IF blkadr=0 THEN ABORT(listspace, $"exceed LIST allocation zone");
         blkadr _ blkadr + blkadj;
         FOR i _ 0 UP UNTIL >= size DO
            [blkadr+i] _ value[i] ;
         descr.ledval _ blkadr ;
         descr.ledtyp _ lblock ;
         descr.ledalo _ TRUE ;
         END ;
   ENDCASE sysrcv(programbug, $"bad type given to makelm", sysclr());
   RETURN ( descr ) ;
   END.
(freelm) % free any storage associated with given list element %                1
   PROCEDURE ( descr ) ; % the list element descriptor %
   % frees any allocated storage associated with this element %
   LOCAL
      blkadr; % address of block to free %
   IF descr.ledalo THEN
      CASE descr.ledtyp OF
         = linteg:
            BEGIN
            blkadr _ descr.ledval - blkadj ;
            IF freeblk ( blkadr, lstzone ) =0 THEN sysrcv(programbug, $"cannot
            deallocate a list", sysclr());
            END ;
         = lstrin:
            BEGIN
```

```
            blkadr _ descr.ledval ;
            freestring(blkadr, lstzone);
            END ;
         = llist:
            BEGIN
            nulist ( descr.ledval, FALSE ) ;
            frelst(descr.ledval);
            END;
         = lblock:
            BEGIN
            blkadr _ descr.ledval ;
            IF freeblk ( blkadr-blkadj, lstzone ) =0 THEN sysrcv(programbug,
            $"cannot deallocate a list", sysclr());
            END ;
         ENDCASE sysrcv(programbug, $"bad type given to freelm", sysclr());
   RETURN(nulldescr);
   END.
(ckdesc) % check list element descr for reasonable values %                    1(
   PROCEDURE ( descr ) ;
   %ckdesc checks a list element descriptor to see that it meets some
   criteria, for example that the type is in range and the *!ag bits are
   reasonable. ckdesc returns true if all is well, false otherwise. %
   IF descr.ledtyp NOT IN [lnull, llist] THEN RETURN ( FALSE ) ;
   %one could check adddress range depending on local or global declared vs
   allocated storage. %
   % one could ckeck to see that no unused bits are on. %
   RETURN(TRUE);
   END.
(getlst) % get a storage block for a list %                                    1'
   PROCEDURE (size); % number of list elements, max %
   % Uses getblk to obtain block of storage *size*+1 words long %
   LOCAL adr REF; % address %
   &adr _ getblk( size+1 + lsthdr, lstzone);
   IF %adr=0 THEN ABORT(listspace, $"exceed LIST allocation zone");
   &adr _ &adr + blkadj + lsthdr;
   adr.M _ size;
   RETURN(&adr);
   END.
(frelst) % free an allocated list element %                                    1(
   PROCEDURE (list REF); % address of list %
   % Uses freblk to free up the allocated block of storage %
   IF freeblk( &list - lsthdr - blkadj, lstzone) =0 THEN sysrcv(programbug,
   $"cannot deallocate a list", sysclr());
   RETURN;
   END.
(lreadb) % read user bits from list element descr %                            1'
   PROCEDURE(
      list REF, % list adress %
      index); % element index %
   LOCAL
      descr,
      alist REF;
   % return composite type, user bits %
   IF (&alist _ list[storwd]) = 0 THEN &alist _ &list;
   IF index NOT IN [1,alist.L] THEN RETURN(0,0);
   descr _ alist[index];
```

```
    RETURN(descr.ledubv, descr.ledbits);
    END.
(lsetb) % set user bits in list elment descr %                                  20
    PROCEDURE(
        list REF, % lis address %
        index, % element index %
        value); % new user-bits value %
    % stores new value in user bits, return TRUE if OK %
    LOCAL
        alist REF;
    IF (&alist _ list[storwd]) = 0 THEN &alist _ &list;
    IF index NOT IN [1,alist.L] THEN RETURN(FALSE);
    alist[index].ledbits _ value;
    RETURN(TRUE);
    END.
(sysfll) %Free Local Lists. Call produced by "NULL-LISTS" %                     21
    PROCEDURE(
        ptr REF); % address of array of local lists %
    % L10 compiler produces a call to sysfll for NULL-LISTS statement. The
    argument is the address of an array of M relative list locations in the
    calling procdure. All those lists are nulled. A zro ends the array. %
    WHILE ptr#0 DO
        BEGIN
        nulist( ptr.RH + [M].RH, FALSE);
        BUMP &ptr;
        END;
    RETURN;
    END.
FINISH L10LRP
```

Rt - Main

(pcloop)	<compsrc, rt-main, 03612>	LOCAL	5B6D12C
(pclpend)	<compsrc, rt-main, 03625>	LOCAL	5B6D12D
(pdc)	<compsrc, rt-main, 0931>	PROCEDURE	9A
(pid)	<compsrc, rt-main, 04183>	LOCAL	5H1C
(popsig)	<compsrc, rt-main, 02070>	LOCAL	5G3
(port)	<compsrc, rt-main, 02103>	LOCAL	5H2B1
(port)	<compsrc, rt-main, 02095>	LOCAL	5H1B1
(rcpsh)	<compsrc, rt-main, 01162>	LOCAL	10B
(rcpto)	<compsrc, rt-main, 01191>	LOCAL	10E
(rcummy)	<compsrc, rt-main, 02366>	FIELD - 1	4I6
(repchr)	<compsrc, rt-main, 0687>	LOCAL	8M
(repet)	<compsrc, rt-main, 02367>	FIELD - 3	4I7
(repstr)	<compsrc, rt-main, 0825>	LOCAL	8AE
(respos)	<compsrc, rt-main, 0483>	PROCEDURE	7D
(rhf)	<compsrc, rt-main, 02368>	FIELD - 1	4I8
(ring)	<compsrc, rt-main, 02360>	RECORD	4I
(rinst1)	<compsrc, rt-main, 02364>	FIELD - 7	4I4
(rinst2)	<compsrc, rt-main, 02365>	FIELD - 7	4I5
(rnamef)	<compsrc, rt-main, 02370>	FIELD - 1	4I10
(rnameh)	<compsrc, rt-main, 02373>	FIELD - 30	4I13
(rnull)	<compsrc, rt-main, 02372>	FIELD - 1	4I12
(rpop)	<compsrc, rt-main, 01172>	LOCAL	10C
(rsdb)	<compsrc, rt-main, 02363>	FIELD - 18	4I3
(rsio)	<compsrc, rt-main, 02374>	FIELD - 30	4I14
(rstsk)	<compsrc, rt-main, 01185>	LOCAL	10D
(rsub)	<compsrc, rt-main, 02361>	FIELD - 18	4I1
(rsuc)	<compsrc, rt-main, 02362>	FIELD - 18	4I2
(rtf)	<compsrc, rt-main, 02369>	FIELD - 1	4I9
(rtorgin)	<compsrc, rt-main, 02371>	FIELD - 1	4I11
(savpos)	<compsrc, rt-main, 0479>	PROCEDURE	7C
(schars)	<compsrc, rt-main, 02380>	FIELD - 11	4J3
(scnlft)	<compsrc, rt-main, 01072>	PROCEDURE	9L
(scnrht)	<compsrc, rt-main, 01075>	PROCEDURE	9M
(scp)	<compsrc, rt-main, 01067>	PROCEDURE	9K
(sdbhdl)	<compsrc, rt-main, 01468>	EXT CONSTANT =5	3J
(sdbhead)	<compsrc, rt-main, 02377>	RECORD	4J
(sgarb)	<compsrc, rt-main, 02378>	FIELD - 1	4J1
(sgbot)	<compsrc, rt-main, 03904>	CONSTANT =8	5A7I
(sgcat)	<compsrc, rt-main, 03903>	CONSTANT =7	5A7H
(sgfrm)	<compsrc, rt-main, 03896>	CONSTANT =0	5A7A
(sgg2)	<compsrc, rt-main, 03900>	CONSTANT =4	5A7E
(sgg3)	<compsrc, rt-main, 03901>	CONSTANT =5	5A7F
(sgg4)	<compsrc, rt-main, 03902>	CONSTANT =6	5A7G
(sggco)	<compsrc, rt-main, 03897>	CONSTANT =1	5A7B
(sggn)	<compsrc, rt-main, 03907>	CONSTANT =11	5A7L
(sggtp)	<compsrc, rt-main, 03899>	CONSTANT =3	5A7D
(sgnst)	<compsrc, rt-main, 03906>	CONSTANT =10	5A7K
(sgsig)	<compsrc, rt-main, 03898>	CONSTANT =2	5A7C
(sgstk)	<compsrc, rt-main, 03905>	CONSTANT =9	5A7J
(sigrestore)	<compsrc, rt-main, 03683>	PROCEDURE	5G2
(sigsave)	<compsrc, rt-main, 03665>	PROCEDURE	5G1
(sigsiz)	<compsrc, rt-main, 02030>	CONSTANT =12	5A7N
(sinit)	<compsrc, rt-main, 02386>	FIELD - 21	4J9
(sitpsid)	<compsrc, rt-main, 02394>	FIELD - 18	4J12
(slength)	<compsrc, rt-main, 02379>	FIELD - 9	4J2
(slngth)	<compsrc, rt-main, 01262>	PROCEDURE	8X

(sysena)	<compsrc, rt-main, 01604>	LOCAL	5E6
(sysenc)	<compsrc, rt-main, 02260>	LOCAL	5F2
(sysent)	<compsrc, rt-main, 02205>	LOCAL	5B6A
(syshelp)	<compsrc, rt-main, 01797>	LOCAL	5E1
(sysnxp)	<compsrc, rt-main, 01586>	LOCAL	5B7B
(sysovr)	<compsrc, rt-main, 01782>	LOCAL	5B6B
(syspc1)	<compsrc, rt-main, 03647>	LOCAL	5B6D9
(syspc2)	<compsrc, rt-main, 03608>	LOCAL	5B6D12
(syspc3)	<compsrc, rt-main, 03658>	LOCAL	5B6D12B
(syspc4)	<compsrc, rt-main, 04162>	LOCAL	5B6D12C5(
(syspop)	<compsrc, rt-main, 01936>	LOCAL	5E4H
(sysprc)	<compsrc, rt-main, 02092>	LOCAL	5H1
(sysrcv)	<compsrc, rt-main, 01569>	LOCAL	5B
(sysrsume)	<compsrc, rt-main, 01794>	LOCAL	5E3
(sysrtf)	<compsrc, rt-main, 01572>	LOCAL	5B5C
(sysrtn)	<compsrc, rt-main, 01573>	LOCAL	5B5B
(sysrtt)	<compsrc, rt-main, 01570>	LOCAL	5B5A
(syssov)	<compsrc, rt-main, 02165>	LOCAL	8R
(syssys)	<compsrc, rt-main, 01711>	LOCAL	5C
(syster)	<compsrc, rt-main, 01733>	LOCAL	5B7E
(systi1)	<compsrc, rt-main, 03319>	LOCAL	5D1E2
(systi2)	<compsrc, rt-main, 03328>	LOCAL	5D1E3
(systi3)	<compsrc, rt-main, 03329>	LOCAL	5D1E4
(systi4)	<compsrc, rt-main, 03327>	LOCAL	5D1E5
(systi5)	<compsrc, rt-main, 03330>	LOCAL	5D1E6
(systj3)	<compsrc, rt-main, 03318>	LOCAL	5D1E1
(systoff)	<compsrc, rt-main, 03375>	LOCAL	5D2
(systrc)	<compsrc, rt-main, 03296>	LOCAL	5D1
(systrm)	<compsrc, rt-main, 01628>	LOCAL	5E5
(systwd)	<compsrc, rt-main, 03331>	LOCAL	5D1E7
(systz1)	<compsrc, rt-main, 01576>	LOCAL	5B5B4
(systz2)	<compsrc, rt-main, 01580>	LOCAL	5B5C5
(systz3)	<compsrc, rt-main, 02208>	LOCAL	5B6A3
(systz4)	<compsrc, rt-main, 02402>	LOCAL	5B6C4
(systz5)	<compsrc, rt-main, 02218>	LOCAL	5B6D10
(sysufl)	<compsrc, rt-main, 01780>	LOCAL	5B7F
(sysvoke)	<compsrc, rt-main, 01795>	LOCAL	5E4
(tcatp)	<compsrc, rt-main, 03795>	REF	5C2A
(tchf)	<compsrc, rt-main, 01127>	PROCEDURE	9R
(temp)	<compsrc, rt-main, 03861>	LOCAL	5G3C
(teststk)	<compsrc, rt-main, 04184>	LOCAL	5H1D
(tsigptr)	<compsrc, rt-main, 03862>	REF	5G3E
(tstf)	<compsrc, rt-main, 0961>	PROCEDURE	9D
(tstsr)	<compsrc, rt-main, 0949>	PROCEDURE	9C
(xxreadc)	<compsrc, rt-main, 0487>	PROCEDURE	7E
(zhash)	<compsrc, rt-main, 04315>	PROCEDURE	8V

```
FILE RtMain
   ALLOW! % system interface stuff here %
   SET TRACE=1; % 1 to compile trace code, 0  to omit it %

% NOTES ABOUT RUNNING STANDALONE L10 PROGRAMS %
   % the following files should be loaded with your program to generate a
   standalone L10 program:
      Rt-Data    - runtime data package
      Rt-Main    - main part of runtime package
      Stenex             - tenex jsys definitions
   The following procedures must be provided:
      A startup procedure
         This procedure is given control after the L10 environment is
         established.  The address of this procedure must be stored in system
         variable "startup". No arguments are passed.
      A recovery procedure
         This procedure is given control when the runtime package encounters
         ar error and cannot contnue.  At the entry point, the stack has been
         totally reset, but non-system globals remain unchanged from their
         condition at the time of the error.  Three arguments are passed: the
         first indicates the type of error and the second is the address of a
         string describing the error and the third provides an address to help
         locate the offending code. The address of this procedure must be
         stored in system variable "recover".
         The last two arguments may be passed to procedure "errmsg" in order
         to get a user-readable error string printed.  The arguments are:
         (jfn, arg2, arg3) where arg2 and arg3 are the arguments passed to the
         recover routine.
   the following will be undefined unless they are provided by the user:
      filesc
         this procedure is called from procedure esc (end string construction)
         when esc detects that the strings being dealt with do not have the
         stastr bit set (i.e., they are not astrings)
         if a user is not dealing with any strings other than literal, local,
         or global strings (for example, strings within a data file), then
         this procedure need not be suoplied.
         to see how NLS handles this see <nls, utilty, filesc>
      flcpfse
         this procedure is called from procedure cpfse when cpfse detects that
         the strings being dealt with do not have the stastr bit set (i.e.,
         they are not astrings)
         if a user is not dealing with any strings other than literal, local,
         or global strings (for example, strings within a data file), then
         this procedure need not be supplied.
         to see how NLS handles this see <nls, utilty, flcpfse>
      flfechc1
         this procedure is called from procedure fechc1 when fechc1 detects
         that the strings being dealt with do not have the stastr bit set
         (i.e., they are not astrings)
         if a user is not dealing with any strings other than literal, local,
         or global strings (for example, strings within a data file), then
         this procedure need not be supplied.
         to see how NLS handles this see <nls, utilty, flfechc1>
   The starting point for the program is the procedure whose address is in
   system variable "startup". However, execution must start at location
   L10start in order to initialize the program environment.
```

To run a program, load it and go into DDT. Then place the address of the starting procedure in system variable "startup", the address of the recover procedure in location "recover", and start executing at location "l10start".

To make use of the TENEX entry vector, do one of the following:

1) Set "startup" to the address of an initializing procedure that will set the entry vector to a table of JRST instructions and then HALTF. For each entry point, the JRST goes to a code branch that sets "startup" to the desired starting procedure and then does a GOTO location "l10start".

Note that, while executing the code branches that set the "startup" location, the runtime environment is not intitialized. Therefore it is impossible to do procedure calls or reference locals. However, a simple assignment such as the following is OK:

```
startup _ $procedure;
```

Set "recover" to the desired procedure address and start executing at "l10start" and when your program halts, save the core immage. At startup time, the proper code branch will set the "startup" location and pass control to L10 initialization which will pass control to the startup procedure.

2) If your program does no initializatior before you save it, it may be desirable to prepair the entry vector as above and use the EXEC's ENTVEC command to establish your entry vector.

Load the program, setup "recover", go to the EXEC and do the ENTVEC comand and then save the core immage.

if you have problems or comments getting standalone runtime programs to the point where the bugs are in your program and not associated with the L10 environment, contact a systems programmer at SRI-ARC

.Pes;%

```
% DECLARATIONS %
   REGISTER
      a1 = 10, %working register%
      a2 = 11, %working register%
      a3 = 12, %working register%
      a4 = 13, %working register%
      r1 = 1,  %working register%
      r2 = 2,  %working register%
      r3 = 3,  %working register%
      r4 = 4,  %working register%
      r5 = 5,  %working register%
      rp = 9, %record pointer%
      p = 7,   %pattern stack%
      wa = 8,   %work area stack%
      mreg = 6, % "meta" result register, success/fail and port %
      s = 15,   %general call stack pointer%
      m = 14;  %mark stack pointer%
   EXTERNAL
      readc,       %entry to READC routine%
      L10start; % entry to initialization code %
   (errmsbase) EXTERNAL CONSTANT = 1400;                                    3C
   (empty) EXTERNAL CONSTANT = 0;                                           3D
   (endfil) EXTERNAL CONSTANT = -1;                                         3E
   (origin) EXTERNAL CONSTANT = 2;  % =fbhdl %                              3F
   (nullch) EXTERNAL CONSTANT = 0;                                          3G
   (chbmty) EXTERNAL CONSTANT = 440700000001B;                              3H
      % $"xxx"+chbmty is byte pointer to first byte %
   (bits) EXTERNAL = ( % 36 bit table: bits[n] is a mask with 1 in the nth bit
   from the left; bits[0] is the leftmost bit %                             3J
      4B11, 2B11, 1B11,
      4B10, 2B10, 1B10,
      4B9, 2B9, 1B9,
      4B8, 2B8, 1B8,
      4B7, 2B7, 1B7,
      4B6, 2B6, 1B6,
      4B5, 2B5, 1B5,
      4B4, 2B4, 1B4,
      4B3, 2B3, 1B3,
      4B2, 2B2, 1B2,
      4B1, 2B1, 1B1,
      4, 2, 1 );
   (scbhdl)  EXTERNAL CONSTANT = 5;  % length of sdb header %              3.
   (forward) EXTERNAL CONSTANT = 1;  % used by READC constructs %          3K
   %.Pes;%
```

```
% RECORDS %
   (byteptr) RECORD  % field definitions for PDP-10 byte pointer word %      4A
      bpadr[18],
      bpindx[6],
      bpsize[6],
      bpbitpos[6];
   (stidr) RECORD                                                            4B
      stpsid[18],
      stfile[6],
      stastr[1];
      % -- (filmnp, lodrng) and (filmnp, lodsdb) cheat and make use of the
      format of an stid -- so please don't change this without  checking those
      routines and telling WHP %
   (stbw) RECORD                                                             4C
      stwc[9],
      stblk[9];
   (stsdb) RECORD                                                            4D
      stpscb[18];
   (stast) RECORD                                                            4E
      stadr[18];
   (stsidr) RECORD                                                           4F
      stsidf[6],
      stsid[30];

   (clistr) RECORD %clist entry%                                             4G
      clst1[36],  %original or updated stid%
      clst2[36],  %successor stid%
      clcc1[12],  %character count for first%
      clcc2[12],  %character count for second%
      clfixed[1]; %for aptstr%

      (cll) EXTERNAL CONSTANT = 3;                                           4G6
      (clmax) EXTERNAL CONSTANT = 50;
                                                                             4G7
   (clhdr) RECORD %clist header%                                             4H
      clcnt[9],   %count of entries in use%
      clfno1[7],  %type used to generate clist%
      clfno2[7],  %type used to generate clist%
      cltype[6],  %type used to generate clist%
      clbuff[18]; %address of buffer%

   (ring) RECORD % *** ring1 is length%                                      4I
      rsub[18],     %psid of sub of this statment%
      rsuc[18],     %psid of suc of this statement%
      rsdb[18],     %psdb of sdb for this statement%
      rinst1[7],    %DEX interpolation string-- scratch space%
      rinst2[7],    %DEX interpolation string-- scratch space%
      rdummy[1],    %DEX dummy flag-- scratch space%
      repet[3],     %DEX repetition-- scratch space%
      rhf[1],       %head flag, true if this is head of plex%
      rtf[1],       %tail flag, true if tail of plex%
      rnamef[1],    %name flag, true if statement has a name%
      rtorgin[1],   %inferior tree origin flag, true if origin%
      rnull[1],     %unused%
      rnameh[30],   %name hash for this statement%
      rsid[30];     %statement identifier%
```

```
   %although only neec four words, use five so that have room
   to grow%

(sdbhead) RECORD % *** sdbhdl is length%                                    4
   sgarb[1],     %true if this sdb is garbage%
   slength[9],   %number of words in this sdb%
   schars[11],   %number of characters in this statement%
   slnmdl[7],    %left name delimiter for statement%
   srnmdl[7],    %right name delimiter for statement%
   spsid[18],    %psid of the statement for this sdb%
   sname[11],    %position of character after name%
   stime[36],    %date and time when this sdb created%
   sinit[21],    %initials of user who created this sdb%
   sptype[15],   %property type of this data block%
      %
      txttyp    = text data block (SDB)
      dhdtyp    = diagram header block
      segtyp    = segment data block
      %
   spsdb[18],    %PSDB of the next property data block 0=tail%
   sitpsid[18]; %PSID to head of inferior tree if any%
   %sgarb and slength must be in the first word of the header
   for newsdb%

(frmtctrl) RECORD %format control for "STRING" construct %                  4F
   fmncols[7],        % # columns (with punctuation) / 0: as many as needed
   %
   fmjstfy[1],        % TRUE: right justify; FALSE: left justity %
   fmfill[1],         % fill field with   TRUE: 0s;  FALSE: spaces %
   fmlgcl[1],         % TRUE: treat as 36 bit ursigned quantity %
   fmsgne[1],         % TRUE: ignore LH of value and extend sign of RH %
   fmpsgn[1],         % TRUE: print + if val > 0 / val is 36 bit quantity %
   fmdecm[1],         % TRUE: print terminating decimal point %
   fmovrf[3],         % column overflow control
      0 - print nothing
      1 - print as many most significant digits (MSDs) as fit
      2 - print as many least significant digits (LSDs) as fit
      3 - print blanks
      4 - print astericks in entire field
      5 - print as many MSDs as fit followed by one asterick
      6 - print as many LSDs as fit preceded by one asterick %
   fmflcat[20];       % reserved for eventual floating point format control
   %

%.Pes;%
```

```
% system startup, returns, signal and fatal error code & declarations %
   % DECLARATIONS for CALL/RETURN mechanisms %
      EXTERNAL % labels (for runtime "procedures") %
         bptrcv, bptctn, bptres, bptinv, bpttrm, bptrtn, bptabt,
         sysrtn, sysrtt, sysrtf, syster,
         sysnxp, syscer, syscte, sysufl,
         sysovr, syspop, sysent, syscent, pcall;
      EXTERNAL % labels for trace package %
         systz1, systz2, systz3, systz4, systz5, systi1, systi2, systi3,
         systi4, systi5, systwd; % trace labels %
      % special code for enable count %
         (dropped) CONSTANT=-100000; % this enable count means catchphrase
         dropped %                                                    5A3
      % size of allocated stack %
         (callstksize) EXTERNAL CONSTANT = 2000B;                     5A4
      % catchphrase frame indices %
         (catloc) CONSTANT=0; % location of catchphrase %             5A5
         (catfrm) CONSTANT=1; % (frame) PORT id for catchphrase %     5A5
         (catenc) CONSTANT=2; % enable count for catchphrase %        5A5
         (cats) CONSTANT=3; % S at invoke time %                      5A5
         (catrtn) CONSTANT=4; % real return location for owning procedure %
                                                                      5A5
            % zero if not first invoked catchphrase owned by procedure %
         (cattloc) CONSTANT=5; % termination location %               5A5
         (catprm) CONSTANT=6; % single catchphrase parameter %        5A5
         (catown) CONSTANT=7; % (frame) PORT id for owning procedure % 5A5
         (catmrk) CONSTANT=8; % indicates which signals this catchphrase has
         seen %                                                       5A5
         (catstk) CONSTANT=9; % stack id of stack which owns this catchphrase
         %                                                           5A5
         % ------- %
         (catsiz) CONSTANT=10; % entries in catch frame %             5A5
      % stack descriptor indices %
         (stkmsave) CONSTANT = -1; % saved M ptr %                    5A6
         (stkssave) CONSTANT = -2; % saved S ptr %                    5A6
         (stklink) CONSTANT = -3; % forward link to next stk or 0 %   5A6
         (stkblink) CONSTANT = -4; % backward link to previous stk or 0% 5A6
         (stkzone) CONSTANT = -5; % allocation zone if allocated stack; 0
         otherwise%                                                   5A6
         (stksize) CONSTANT = -6; % size of usable stack size of usable stack
         area%                                                        5A6
         (stksig) CONSTANT = -7; % Number of the current signal in progress on
         this stack.%                                                 5A6
         (stkdalo) CONSTANT = -8; % Stack which is deallocating this stack: 0
         except in dalostk%                                           5A6
         (stkmdalo) CONSTANT = -9; % M for Stack which is deallocating this
         stack: 0 except in dalostk%                                  5A6
         (stksdalo) CONSTANT = -10; % S for Stack which is deallocating this
         stack: 0 except in dalostk%                                  5A6
         % ------- %
         (stackdecsize) CONSTANT=10; % entries in stack descriptor frame at
         bottom of each stack %                                       5A6
      % signal frame indices: BLT'd from globals indicated.  See their
      decalrations for further information %
         (sgfrm) CONSTANT = 0; % signalling routine's fram (port id)% 5A7
            % SYSFRM %
```

```
    (sggco) CONSTANT = 1; %called catchphrase's co-return location %  5A7E
       % SYSGCO %
    (sgsig) CONSTANT = 2; %signal value/name %                        5A7C
       % SYSSIG %
    (sgctp) CONSTANT = 3; % signal type %                             5A7D
       % SYSCTP %
    (sgc2) CONSTANT = 4; % signal parms %                             5A7E
       % SYSG2 %
    (sgg3) CONSTANT = 5; % signal parms %                             5A7F
       % SYSG3 %
    (sgg4) CONSTANT = 6; % signal parms %                             5A7G
       % SYSG4 %
    (sgcat) CONSTANT = 7;% catchphrase stack pointer for dispatched
    catchphrase %                                                     5A7H
       % &SYSCAT %
    (sgbot) CONSTANT = 8; % "bottom" of stack for signal propogation% 5A7I
       % SYSBOT %
    (sgstk) CONSTANT = 9; % stack pointer to base of stack propogating
    the signal %                                                      5A7J
       % SYSSSTK %
    (sgnst) CONSTANT = 10; % nested scan flag %                       5A7K
       % NEXTSCAN %
    (sggn) CONSTANT = 11; % the signal number %                       5A7L
       % SYSGN %
    % ------- %
    (sigsiz) CONSTANT=12; % in signal in progress frame %             5A7M
REF syscpt, syscat, syscpe, sysgpt, startup, recover;
%.Pes;%
```

```
(sysrcv) % recover/startup/call/return code here %                              5E
   PROCEDURE % called for fatal error %
      (type, % error type code %
      why, % error code %
      loc): % associated location %
      (lsb) REF; % local stack bottom: used in looping over stacks %     5B1C
      (lst) REF; % local stack top: used in looping over stacks %        5B1E

   (bstrcv): % see arguments - label for debugging convenience %          5B2
      % check for recovery loop %
         IF NOT sysrip THEN % recovery in progress ? %
            BEGIN % first try - issue NOTE %
            sysrip _ TRUE;
            sysetc _ type; % error-type code %
            syswhy _ why;
            IF NOT linlink THEN
               BEGIN
               IF loc.RH IN [sysbtm.RH, (sysbtm.RH-sysbtm.LH).RH] THEN % a
               port %
                  loc _ [ loc.RH+1 ].RH; % get its last exit point %
               END
            ELSE % other stacks: is this a port in this stack? %
               BEGIN
               &lsb _ $gstack;
               DO % loop over all stacks %
                  BEGIN
                  &lst _ &lsb + lsb[stksize];
                  IF loc.RH IN [&lsb, &lst] THEN
                     BEGIN
                     loc _ [loc.RH + 1].RH;  % coreturn location %
                     EXIT LOOP;
                     END;
                  END
               WHILE (&lsb _ (lsb[stklink]).RH);
               END;
            sysloc _ loc;
            NOTE(unwind);
            END;
      % store error parameters and start over %
         !JSP R1,L10set; % reset stacks %
         syssys(TRUE); % recover %
      %.Pes;%
```

```
(l10start): % initial startup here %                                    5B3
   !JSP R1,l10set; % reset stacks %
   sysric _ FALSE;
   syssys(FALSE); % recover %

(l10set): % initialize stack pointers (local label) %                   5B4
   %general call stack%
      S _ -gstksz; !HRL S,S; !HRRI S,gstack;
      M _ S;
      sysbtm _ S;
      gstack _ $sysufl;
      sysse _ $gstack+gstksz;
      % indicate only one stack thusfar %
         linlink _ 0;
      % Set up gstack stack descriptor %
         gstack[ stkmsave ] _
         gstack[ stkssave ] _
         gstack[ stklink ] _
         gstack[ stkblink ] _
         gstack[ stkzone ] _
         gstack[ stkcalo ] _
         gstack[ stkmcalo ] _
         gstack[ stkscalo ] _
         gstack[ stksig ] _ 0;
         gstack[ stksize ] _ gstksz;
   %pattern stack%
      p _ -pstksz; !HRL p,p; !HRRI p,pstack;
      swork _ endfil; % initialize for patterns %
   % spec stack %
      spsk _ spsk1;
   % reset clist change stack %
      RESET clchng; % before any string constructions are done%
      clmpty _ clchng;
   % return %
      !JRST 0(R1);
   %.Pes;%
```

```
   % returns, get here by JRST produced by compiler %
      (sysrtt): % return true %                                          5B5A
         !MOVEI mreg,1;

      (sysrtn): % return, value in mreg %                                5B5E
         !MOVE R4,S; % save old S for syspop - just in case %
         !MOVE S,M;
         !POP S,M;
         (systz1): !POPJ S,0;                                            5B5B4
         %+TRACE%                                                        5B5B8
            !MOVEM R1,systr1; !jobtm(); % get clock %
            !SUBM R1,systck; !EXCH R1,systck; % elapsed time %
            !SKIPGE R1; !MOVEI R1,0; % check wraparound %
            !HRLI R1,0; !JSR systwd; % write 0,,time: means return %
            !MOVE R1,systr1; !POPJ S,0; % return %
            %+TRACE%                                                     5B5B5F

      (sysrtf): % return false %                                         5B5C
         !MOVEI mreg,0;
         !MOVE R4,S; % save old S for syspop %
         !MOVE S,M;
         !POP S,M;
         (systz2): !POPJ S,0;                                            5B5C5
         %+TRACE%                                                        5B5C6
            !JRST systz1; % for tracing %
            %+TRACE%                                                     5B5C6E
      %,Pes;%
```

```
% procedure entry, pcall. get here by JSP A4 produced by compier %
   (sysent): % procedure entry point (trace point) %                    5B6A
      !PUSH S,M; % push old mark %
      !MOVE M,S; % set up new one %
      (systz3): !AOBJN S,0(A4); % return to procedure %                 5B6A3
         % AOB puts blank on stack (coroutine return) %
         % also check for overflow (positive) %

   (sysovr): % stack overflow (follows sysent) %                        5B6E
      S _ sysbtm; M _ S;
      sysrcv(stkoverflow,$"general stack overflow",0);

   (syscent): % port entry (trace point) %                              5B6C
      !PUSH S,M; % push old mark %
      !MOVE M,S; % get new one %
      !PUSH S,A4; % setup co-return loc for ddt %
      (systz4): !JRST 0(A4); % return to port entry %                   5B6C4
      %+TRACE% % make it look like pcall %                              5B6C5
         !MOVEM R1,systr1;
         !MOVE R1,systmd; !TRNN R1,1; !JRST notrce1; %check mode %
         !jobtm(); % get clock %
         !SUBM R1,systck; !EXCH R1,systck; % elapsed time %
         !SKIPGE R1; !MOVEI R1,0; % check wraparound %
         !HRLI R1,1; !JSR systwd; % write 1,,time: means pcall %
         !HRLI R1,0(A4); !HRR R1,1(M); !JSR systwd; %source,,dest%
         !MOVEI R1,0; !JSR systwd; % a zero for now %
         (notrce1):                                                    5B6C5I
         !MOVE R1,systr1; !JRST 0(A4); % return to port entry %
         %+TRACE%                                                      5B6C5K
      %.Pes;%
```

```
(pcall): % port-call code (trace point) %                              5B6C
   % mreg has port id to call; A4 contains return address; M has
   calling port id %
   !MOVEM A1,tempa1save;        % save A1 (argument/result) %
   !MOVEM mreg,tempmregsave;    % save mreg (destination port) %
   !HRRZS mreg;
   !MOVE A1,sysbtm;
   !CAILE mreg,0(A1);  % new port in current stack? %
   !CAILE mreg,0(S);   % It is if sysbtm < mreg < S %
      !JRST syspc2;    % New port not in current stack %
   !MOVEM A1,pcstack;  % set up previous stack pointer %
   (syspc1):           % pulse coroutine %                               5B6D9
      !SETZM sysfctn;  % reset system incicator %
      !MOVEM A4,1(M);  % save return in old frame %
      !MOVE A1,tempa1save;     % restore A1 %
      !MOVE mreg,tempmregsave; % restore full port %
      !EXCH mreg,M;    % setup new frame, old is argument %
   (systz5):                                                           5B6D10
      !JRST @1(M);     % call new one %
   %+TRACE%                                                            5B6D11
      !MOVEM R1,systr1;
      !MOVE R1,systmc; !TRNN R1,1; !JRST notrce; % check mode %
      !jobtm(); % get clock %
      !SUBM R1,systck; !EXCH R1,systck; % elapsed time %
      !SKIPGE R1; !MOVEI R1,0; % check wraparound %
      !HRLI R1,1; !JSR systwd; % write 1,,time: means pcall %
      !HRLI R1,0(A4); !HRR R1,1(M); !JSR systwd; %source,,dest%
      !MOVEI R1,0; !JSR systwd; % a zero for now %
      (notrce):                                                        5B6D11I
      !MOVE R1,systr1; !JRST @1(M); % call port %
      %+TRACE%                                                         5B6D11K
   (syspc2): % New port not in this stack.  Loop through all stacks %
                                                                       5B6D12
      % save state (M, S, signal status) of current stack %
         !MOVEM A1,pcstack;     % save calling stack ID %
         !MOVEM S,stkssave(A1);         % Save current S %
         !MOVEM M,stkmsave(A1);         % save M %
         % save sig state, if any in progress, if this is not a
         continue PCALL.  (sigsave checks to see if there is any
         signal in progress.)  SYSFCTN is TRUE if the PCALL comes
         from continue (i.e., HELP, NOTE, ABORT, or CONTINUE) or
         RESUME, FALSE otherwise. %
            !SKIPE sysfctn;
               !JRST syspc3;
            % save registers around sigsave %
               !MOVEM R15,srg15;
               !MOVEI R15,srgsav;
               !BLT R15,srg14;
               !MOVE R15,srg15;
            sigsave( A4);
            % restore registers %
               !HRLZI R15,srgsav;
               !BLT R15,R15;
      (syspc3):         % point to first stack %                        5B6D12B
         !SKIPN A1,linlink;     % more than 1 stack? %
            !JRST pclerr;       % no - give error %
```

```
(pcloop):                                                          5B6D12C
   !EXCH S,stkssave(A1); % get old S %
   !CAILE mreg,0(A1);    % in this stack? %
   !CAILE mreg,0(S);
      !JRST pclpend;     % no. look at next stack. %
   !MOVEM S,stkssave(A1);        % restore saved S %
   % we have found the proper stack for the new port %
      !MOVEM A1,sysbtm;  % set up sysbtm to new stack %
      % restore signal state for new stack if this is not a
      continue pcall %
         !SKIPE sysfctn;
            !JRST syspc4;
         % save registers around sigrestore %
            !MOVEM R15, srg15;
            !MOVEI R15, srgsav;
            !BLT R15, srg14;
            !MOVE R15, srg15;
         sigrestore();
         % restore registers %
            !HRLZI R15, srgsav;
            !BLT R15, R15;
      (syspc4):                                                    5B6D12C5C
         !JRST syspc1;            % dispatch PCALL %
(pclpend):                                                         5B6D12D
   !EXCH S,stkssave(A1);
   !SKIPE A1,stklink(A1);        % any more stacks? %
      !JRST pcloop;      % Yes. %
(pclerr):                                                          5B6D12E
   !MOVE A1,tempa1save;  % No. Restore A1 %
   !MOVE mreg,tempmregsave;      %. Restore mreg %
   !JRST sysnxp;         % error %
%.Pes;%
```

```
% fatal error points, JSP,A4 produced by compiler %

    (sysrxp): % non-existant port called %                                  5B71
       sysrcv(programbug,$"non-existant port called",A4);

    (syscer): % coroutine called as procedure %                             5B71
       sysrcv(programbug,$"coroutine called as procedure",A4);

    (syscte): % catchphrase termination error %                             5B71
       sysrcv(programbug,$"no catchphrase termination specified",A4);

    (syster): % catchphrase fall-thru error %                               5B71
       sysrcv(programbug,$"catchphrase did not dispatch signal",A4);

    (sysufl): % stack underflow (strange) %                                 5B71
       sysrcv(programbug,$"general stack underflow",0);

END.  % of sysrcv procedure above %
%.Pes;%
```

```
(syssys) % initialize system coroutines, etc %                                  5C
   PROCEDURE % this must be done from a procedure %
      (status); % TRUE if recovering %                                          5C1A
   % LOCAL  for catchphrase %
      (tcatp) REF; % temporary pointer to catchframe stack %                    5C2A
      (mask); % used by catchphrase to contain a mask which will cause  the
      bit corresponding to the signal which has been in progress to be shut
      off. %                                                                    5C2E
   % initialize catchphrase stack %
      &syscpe _ &syscat _ $syscstk;
      &sysgpt _ $sysgstk;
      sysgn _ tsysgn _ sysbfg _ 0;
      syshcon _ FALSE;
   % open system coroutines %
      OPENPORT syshelp( helptype :[syshlp]); % HELP %
      OPENPORT sysctn(:[syscon]); % CONTINUE %
      OPENPORT syshelp( aborttype :[sysabt]); % ABORT %
      OPENPORT syshelp( notetype :[sysnot]); % NOTE %
      OPENPORT sysrsume(:[sysres]); % RESUME %
      OPENPORT sysvoke(:[sysinv]); % INVOKE %
   % invoke catch-all catchphrase %
      INVOKE(syscatch);
      sysctchall _ &syscpe-catsiz; % save system catch pointer %
   % give control to program %
      sysrip _ FALSE;
      IF status THEN recover(sysetc,syswhy,sysloc)
      ELSE startup();
         % recover and startup are the addresses of appropriate procedures
         specified for the L10 program.  They<LF> should never return. %
      sysrcv(programbug,$"startup/recover procedure returned",0); % don't
      return %

   (syscatch) CATCHPHRASE;  % TOP CATCHPHRASE in L10 ENVIRONMENT %            5C7
      BEGIN
      % unmark catchphrase stack for the signal which was in progress %
         mask _ bits[sysgn] .% -1;
         FOR %tcatp _ $syscstk UP catsiz UNTIL >= &syscpe DO
            IF tcatp[catstk] = syssstk THEN
               tcatp[catmrk] _ tcatp[catmrk] .A mask;
      CASE SIGNALTYPE OF
         =notetype: % this is where NOTEs are resumed %
            BEGIN
            sysgtp _ helptype; % smash type !! %
            RESUME;
            END;
         =helptype: RESUME(nohelp);
         ENDCASE
            BEGIN
            % params in global safe cells %
               syszgn _ SIGNAL; syszgt _ SIGNALTYPE;
               syszg2 _ sysg2; syszg3 _ sysg3; syszg4 _ sysg4;
            sysrcv(uncaughtabort,$"ABORT to rurtime code",sysfrm);
            END;
      END;
   END.
   %.Pes;%
```

```
% L10 Runtime TRACE FACILITY %
   (systrc) % system trace invoke routine %                                    5D:
      PROCEDURE(
         mempg, % memory page number to use for trace %
         mode, % trace mode specifier %
         typeout); % zero or a jfn to write start/fin messages on %
      % call/return etc. trace setup. trace code currently not protected
      from traced events inside PSI's! %
      %+TRACE%                                                                 5D1(
         IF systen=0 OR systfg THEN RETURN; % no-op if already on %
         % setup globals %
            systpn _ mempg;
            systmd _ mode;
            systpc _ 0; % output file page count %
            systtp _ typeout;
            systpt _ systpn*512; % AOBJ pointer into page %
            systpt.LH _ -512; % count %
         % open file %
            IF NOT SKIP !gtjfn(600001B6, $"TRACE.DATA"+chbmty) THEN
               BEGIN
               IF systtp THEN !sout(systtp, $"gtjfn failed -trace"+chbmty,
               -19);
               RETURN;
               END;
            systjfn.LH _ R1;
            IF NOT SKIP !openf(R1, 2200001B5) THEN
               BEGIN
               IF systtp THEN !sout(systtp, $"openf failed -trace"+chbmty,
               -19);
               RETURN;
               END;
            !chfdb(11B6+R1, 77B8, 22B8); % byte size = 18 %
         % put out message %
            IF systtp THEN
               BEGIN
               !sout(systtp, $"Tracing "+chbmty, -8);
               !bout(systtp, EOL);
               END;
         % replace secret cells %
            systi1 _ systz1 := systi1; % replace secret cells %
            systi2 _ systz2 := systi2;
            systi3 _ systz3 := systi3;
            systi4 _ systz4 := systi4;
            systi5 _ systz5 := systi5;
         % go... %
            systfg _ TRUE;
            systck _ !jobtm();
         %+TRACE%                                                             5D1C
      RETURN;
      %+TRACE%                                                                5D1I
         (systj3):                                                            5D1E:
            !MOVEM R1,systr1; !jobtm(); % get clock %
            !SUBM R1,systck; !EXCH R1,systck; % elapsed time %
            !SKIPGE R1; !MOVEI R1,0; % check wraparound %
            !HRLI R1,-1(A4); !JSR systwd; % write adr,,time: call %
            !MOVE R1,systr1; !AOBJN S,0(A4); !JRST sysovr; % go %
```

```
        (systi1): !JFCL;                                                    5D1E2
        (systi2): !JFCL;                                                    5D1E3
        (systi3): !JRST systj3;                                             5D1E4
        (systi4): !JFCL;                                                    5D1E5
        (systi5): !JFCL;                                                    5D1E6
        (systwd): % JSR routine to write word into trace page %             5D1E7
           !JFCL; % the return location stored here %
           !EXCH R3,systpt; % get pointer/save R3 %
           !MOVEM R1,0(R3); % store word %
           !AOBJN R3,nomap; % count %
           !MOVSI R1,485; % end of page, pmap it %
           !HRR R1,systpn; % core page number %
           !EXCH R2,systjfn; % get jfn in LH, save R2 %
           !HRR R2,systpc; % file page number %
           !MOVSI R3,140400B; % write access, copy on write %
           !pmap();
           !EXCH R2,systjfn; % restore R2 %
           !AOS systpc; % up page count %
           !HRRZ R3,systpn; % initialize pointer again %
           !LSH R3,9; % make page number an address %
           !HRLI R3,-1000B; % initialize count %
           !jobtm(); !MOVEM R1,systck; % make pmap invisible %
           (nomap):                                                         5D1E7G
              !EXCH R3,systpt; % restore pointer %
              !JRST @systwd; % return %
        %+TRACE%                                                            5D1E8
    END.
    %.Pes;%
```

```
(systoff) % system trace - turn it off %                                5D2
   PROCEDURE; % no arguments %
   %+TRACE%                                                             5D2E
      IF systcr=0 OR systfg=0 THEN RETURN; % no-op if already off %
      % set EOF byte count for file %
         R3 _ systbc; !LSH R3,10; % byte count already pmapped %
         R4 _ systpn; !LSH R4,9; % page address %
         !HRRZ R2,systpt; !SUBI R2,0(R4); % words in this page %
         !LSH R2,1; % times 2 for bytes %
         !ADDI R3,0(R2); % total bytes in file %
         !chfdb(12B6+systjfn.LH, -1, R3);
      % map out last page %
         systpt.LH _ 0; % force it % !JSR systwd;
         R4 _ systpn; !LSH R4,9; !MOVEM R4,0(R4); % make it private %
         IF NOT SKIP !closf(systjfn.LH) THEN
            IF systtp THEN !sout(systtp, $"closf failed -trace"+chbmty,
            -19);
      % restore secret cells %
         systi1 _ systz1 := systi1;
         systi2 _ systz2 := systi2;
         systi3 _ systz3 := systi3;
         systi4 _ systz4 := systi4;
         systi5 _ systz5 := systi5;
      % done %
         systfg _ FALSE;
         IF systtp THEN
            BEGIN
            !sout(systtp, $"Trace off"+chbmty, -9);
            !bout(systtp, EOL);
            END;
      %+TRACE%                                                          5D2BE
   RETURN;
   END.
   %.Pes;%
```

```
% Signal and Catchphrase Coroutines and Procedures %
   (syshelp)  % ABORT/NOTE/HELP coroutine %                                    5E1
      COROUTINE ( type);      % signal type %
      LOCAL
         sigval, % temp slot for signal value %
         s2, s3, s4; % other args %
      PORT ENTRY EXIT sigval _ PCALL (:s2, s3, s4);
      LOOP
         BEGIN
         % save global signal info (only needed if we didn't switch stacks
         to get here) %
            IF pcstack = sysbtm THEN sigsave( PORT );
         % set up global info to reflect signal being generated now %
            syssig _ sigval; % set SIGNAL %
            sysg2 _ s2;      % global args %
            sysg3 _ s3;
            sysg4 _ s4;
            sysfrm _ PORT;   % signalling port id %
            sysgtp _ type;   % set signal type %
            syssstk _ pcstack;       % signalling stack id %
            sysinfo _ 0; % the global signal information is not on the
            signal stack %
         % keep track of number of signals in progress %
            BUMP sysgn;      % for generating stack %
            BUMP tysysgn;    % for entire system %
         % search for proper catachphrase from top of catchphrase stack %
            &syscat _ &syscpe;
         % indicate for sysctn that we came from here and not from a
         CONTINUE in a catchphrase %
            syshcon _ TRUE;
         % set "bottom" for this signal (to gererating frame for NOTEs
         Unwind or Return).  SYSBFG is set to a non-zero value in SYSPOP
         (=> RETURN with catchphrases to be dropped) and SYSTRM (=>
         UNWIND).  This value delimits the range of the frames on the call
         stack which will see these signals.  Otherwise, SYSBFG is FALSE:
         we therefore set SYSBOT to the bottom of the signalling stack.
         Signals will propogate only in the stack in which they are
         generated! %
            IF sysbfg THEN sysbot _ sysbfg := 0
            ELSE sysbot _ syssstk;
         % provide breakpoint for aborts %
            IF type = aborttype THEN
               (bptabt): !JFCL;                                               5E1D8A1
         % now call CONTINUE %
            sigval _ PCALL [syscon] (:s2, s3, s4);
         END;
      END.
      %.Pes;%
```

```
(sysctn) % CONTINUE coroutine %                                          5E2
   % This coroutine searches the catch stack from syscat to bottom and
   calls the first eligible catchphrase it finds. syscat setup by
   syshelp, sysabort or sysnte - or a previous CONTINUE. Starts with
   next catchphrase DOWN from syscat. %
   COROUTINE;
   LOCAL port, nur, tcatp REF;
   PORT ENTRY EXIT PCALL(: [port] );
      % Can't reference PORT.RH %
   LOOP
      BEGIN
      % if from a catchphrase, must get proper signal info from sig
      stack into global cells (info will already be in global cells if
      we didn't have to switch stacks to get here) and must restore
      coroutine location in stack frame ( [PORT+1] is co-loc on PDP-10
      ): syshcon  is TRUE if NOTE, ABORT, or HELP; FALSE if CONTINUE,
      i.e. from a catchphrase %
         IF syshcon THEN syshcon _ FALSE
         ELSE
            BEGIN
            IF pcstack # sysbtm THEN
               % we switched stacks to get here; thus we must put signal
               info from sig stack into global area %
               BEGIN
               savebtm _ sysbtm := pcstack;
               sigrestore();
               sysbtm _ savebtm;
               END;
            [port.RH+1] _ sysgco;
            END;
      % If Note (Unwind or Return), we will propogate the signal only to
      those frames more recently placed in the flow of control then the
      frame generating the signal (i.e., its subframes) %
         nur _ (IF sysbot=sysbtm THEN FALSE ELSE TRUE);
      LOOP % over catchphrase stack: find eligible cachphrase %
         BEGIN
         % get to next catchphrase in catstack.  If at bottom, set to
         generate SYSCATCHALL; if not, decice if it should be
         dispatched. %
         IF (&syscat _ &syscat - catsiz) >= $syscstk THEN
            BEGIN % not at bottom: should this be dispatched? %
            % only look at catchphrases that belong to the same stack
            that generated the signal %
               IF syscat[catstk] # syssstk THEN REPEAT LOOP;
            IF NOT nestscan AND syscat[catmrk] THEN
               BEGIN
               % we are not yet in the middle of a nested signal scan,
               but since this catchphrase has been seen by some signal
               (since syscat[catmrk] is non-zero), we start a scan from
               the bottom of the stack and go up the stack until we find
               the first catchphrase seen by the immediately preceding
               signal: that one is a candidate for seeing the current
               signal.  This catchphrase defines where a branch occurred
               in the thread of control. %
               nestscan _ TRUE;
               FOR &tcatp _ $syscstk UP catsiz UNTIL >= &syscat DO
```

```
                IF tcatp[catmrk] .A bits[sysgn-1] THEN EXIT LOOP;
              &syscat _ &tcatp;
              END;
            % This catchphrase is marked as having seen this signal. %
              syscat[catmrk] _ syscat[catmrk] .V bits[sysgn];
            % if this catachphrase is disabled, or if this is a Note
            (Unwind or Return) and this catachphrase is out of range,
            then we must find another catchphrase, otherwise we exit
            this loop and dispatch this catchphrase %
              IF (syscat[catenc] <= 0) OR (nur AND (syscat[catown] <
              sysbot))
                THEN REPEAT LOOP % check another catchphrase %
                ELSE EXIT LOOP; % dispatch the catchphrase %
            END
          ELSE  % no more catchphrases on stack: dispatch the catchall
          catchphrase %
            BEGIN
            &syscat _ sysctchall; % system catchall - will resume NOTE %
            EXIT LOOP;
            END;
          END; % of catchphase scan loop %
        % save coreturn location, then change it to point to catchphrase
        so catchphrase is actually dispatched by PCALLing the owning frame
        %
          sysgco _ [syscat[catfrm]+1] := syscat[catloc];
        % setup catchphrase parameter %
          syscpm _ syscat[catprm];
        (bptctn): % syscat points to catch frame being activated %    5E2E7
        % call catchphrase %
          sysfctn _ TRUE;
            % checked in pcall to see whether the global signal stuff
            should be changed.  IT WILL NOT BE SINCE THIS PCALL IS A
            "SYSTEM" PCALL TO EXECUTE THE SIGNAL! %
          PCALL [syscat[catfrm]] (0,sysg2,sysg3,sysg4:[port]);
        END; % of coroutine body "loop" %
    END.
    %.Pes;%
```

```
(sysrsume) % RESUME coroutine %                                          5E:
   % This coroutine RESUMEs code when a catchphrase does a RESUME %
   CCROUTINE;
   LOCAL
      port, % for port.RH references %
      r, % flag %
      res1, res2, res3, res4, % results to pass on %
      sport; % temp cell for port %
   PORT ENTRY EXIT res1 _ PCALL (: [port] res2, res3, res4);
   LOOP
      BEGIN
      % get signal info into global area (already there if we didn't
      switch stacks) %
         IF pcstack # sysbtm THEN
            BEGIN
            savebtm _ sysbtm := pcstack;
            sigrestore();
            sysbtm _ savebtm;
            END;
      % check signal status for screwups %
         IF SIGNALTYPE # helptype THEN
            sysrcv(programbug,$"attempt to resume a
            NOTE/ABORT",[port.RH+1]);
         IF syscn <= 0 THEN % in progress count %
            sysrcv(programbug,$"attempt to RESUME with no signal in
            progress",[port.RH+1]);
      % restore co-loc in catchphrase frame %
         [port.RH+1] _ sysgco;
      % get current values before poping %
         sport _ sysfrm; % signal from this port %
      popsig();
      syscpm _ syscat[catprm]; % restore parameter %
      (bptres): % (M)+4 is res1, others follow %                         5E3E8
      sysfctn _ TRUE;
         % checked in pcall to see whether the global signal stuff
         should be changed.  IT WILL NOT BE SINCE THIS PCALL IS A
         "SYSTEM" PCALL TO EXECUTE THE SIGNAL! %
      res1 _ PCALL [sport] (res1, res2, res3, res4
         : [port] res2, res3, res4); % RESUME %
      END;
   END.
   %.Pes;%
```

```
(sysvoke) % INVOKE coroutine %                                          5E4
   % This coroutine invokes a catchphrase and sets termination location.
    It also smashes return location for procedures.  NOTE: if caller is
   a coroutine, must locate owning procedure in order to smash its
   return location.  Real return locations are kept in the catchphrase
   frame %
   COROUTINE:
   % PCALL arguments:
      cloc, catchphrase location
      tloc: TERMINATE in this catchphrase instance means GOTO here %
   LOCAL port; % temp store for port id %
   PORT ENTRY
      EXIT syscpe[catloc] _ PCALL(: syscpe[cattloc], syscpe[catprm]);
   LOOP
      BEGIN
      % initialize catchphrase frame %
         % catchphrase location stored in PCALL %
            syscpe[catfrm] _ PORT; % catchphrase port (frame) %
         % termination location stored in PCALL %
         syscpe[catenc] _ 1; % enable it %
         % save current stack pointer of stack invoking catchphrase %
            syscpe[cats] _
               IF pcstack = sysbtm THEN S
               ELSE [pcstack.RH][stkssave];
         % indicate it hasn't seen any signals yet %
            syscpe[catmrk] _ 0;
         % indicate which stack owns this catchphrase %
            syscpe[catstk] _ pcstack;
      (bptinv): % syscpe points to catch frame %                        5E4F3
      % find owning procedure and save/change return location %
         port _ PORT;
         WHILE (NOT sysprc(port)) AND ([port.RH] # $sysufl) DO
            port _ sysbak(port);
         syscpe[catown] _ port;
         IF ([port.RH-1].RH # $syspop) AND ([port.RH] # $sysufl) THEN %
         save/change return %
            syscpe[catrtn] _ [port.RH-1] := $syspop
         ELSE syscpe[catrtn] _ 0; % zero means already changed %
      % bump catchphrase frame pointer %
         IF ( &syscpe _ &syscpe + catsiz) > $sysce THEN
            sysrcv(programbug,$"catchphrase stack overflow",PORT);
      .syscpe[catloc] _ PCALL (:syscpe[cattloc], syscpe[catprm]);
      END;
   %.Pes;%
```

```
(syspop): % label to drop catchphrases or RETURN %                    5E4H
   % first "undo" the return so sig will work %
      !MOVE M,S; !ADD M,=20000028; % M _ S+2 %
      !MOVE S,R4;
   % first save args and mreg %
      !PUSH S,A1; !PUSH S,mreg; %return values that will be smashed %
      !PUSH S,R1; !PUSH S,R2; !PUSH S,R3;
   % put out "return" NOTE %
      sysbfg _ M; % above and here only %
      NOTE(return);
      !POP S,R3; !POP S,R2; !POP S,R1; % restore those args now %
   &syscpt _ &syscpe;
   systloc _ 0; % this will be return loc %
   WHILE (&syscpt _ &syscpt - catsiz) >= $syscstk DO
      IF syscpt[catfrm] IN [M, S) THEN
         % NOTE:
            Old mark was S+2. Two PUSH's make it =S. Want to remove
            all his catchphrases and those of all routines above him
            in stack. There may be phrases for routines below him in
            stack, above his phrases in catch stack. %
         BEGIN
         IF syscpt[catrtn] AND syscpt[catown] = M THEN
            systloc _ syscpt[catrtn];
         % mark the catchphrase to be dropped %
            drpcat _ syscpt[catenc] _ dropped;
         END;
   IF NOT tsysgn AND drpcat THEN sysdpall();
   IF systloc THEN
      BEGIN
      [M.RH-1] _ systloc; % resore return loc %
      !POP S,mreg;
      !POP S,A1; % restore results %
      (bptrtn): % break just before real return %                     5E4H8E
      % just like sysrtn, not traced %
         !MOVE S,M;
         !POP S,M;
         !POPJ S,0; % go %
      END;
   sysrcv(programbug,$"syspop: cannot find return for
   procedure",S.RH);
   % means did not find return for this procedure - major screwup %
END.
%.Pes;%
```

```
(systrm) % terminate this signal (and returr to termloc) %                5E5
   % This procedure performs TERMINATE command.  It terminates signals
   and gives control to the TERMINATinc routine at the location
   soecifiec when the catchphrase was invoked (i.e. termloc ) %
   PROCEDURE;
   LOCAL
      tloc, % to save termination loc - global no good here %
      savs, savm, trmown; % temps to save S, M, TERM owner's port  %
   tloc _ systloc _ syscat[cattloc]; % save the termloc in global %
   trmown _ syscat[catown]; % position in flow of control for TERMINATOR
   %
   (bpttrm): % here, systloc is term. location %                          5E5F
   IF sysgn <= 0 THEN
      sysrcv(programbug,$"TERMINATE with no sig in progress",[M.RH-1]);
   IF sysbtm # syssstk THEN
      sysrcv(programbug,$"TERMINATE for signal in another
      stack",[M.RH-1]);
   savs _ syscat[cats];
   savm _ syscat[catfrm];
   IF SIGNALTYPE = notetype THEN
      sysrcv(programbug,$"attempt to TERMINATE a NOTE",[M.RH-1]);
   popsig(); % pop 1 signal %
   WHILE sysgn>0 AND syscat[catown] >= trmown DO popsig();
      % remove all signals generated by routines lower in flow of
      control than the routine that is TERMINATING %
   sysbfg _ savm+1; % set lower limit for NOTE (NOT this frame) %
   NOTE(unwind);
   % now remove all catchphrases for lower routines %
      &syscpt _ &syscpe;
      WHILE (&syscpt _ &syscpt - catsiz) >= $syscstk DO
         IF syscpt[catfrm] > savm THEN
            % mark the catchphrase to be dropped %
               drpcat _ syscpt[catenc] _ dropped;
      IF NOT tsysgn AND drpcat THEN sysdpall();
   systloc _ tloc; % restore that global, may have been used %
   S _ savs; % restore stack %
   M _ savm;
   GOTO [systloc]; % GOTO termination location %
   END.
   %.Pes;%
```

```
(sysena) % enable catchphrase %                                    5E6
   PROCEDURE
      (catchnam); % address of catchphrase %                       5E6A1
   sysabl(catchnam, [M] % caller's mark %, 1);
   RETURN;
   END.
   %.Pos;%
```

```
(sysdis) % disable catchphrase %                                    5E7
   PROCEDURE
      (catchnam); % address of catchphrase %                        5E7A1
   sysabl(catchnam, [M] % caller's mark %, -1);
   RETURN;
   END.
   %.Pes;%
```

```
% Signal and Catchphrase Utility Procedures %
   (sysabl) % manipulate catchphrase enable count %                    5F1
      PROCEDURE
         (catchname, % address of catchphrase %
         oldmark, % mark for caller %
         n); % count %
      LOCAL
         catp REF: % catchphrase stack pointer %
      &catp _ &sysctp:
      % now search for most recent instance of catchname %
         WHILE (&catp _ &catp - catsiz) > $syscstk DO
            IF % right one %
               catp[catloc] = catchname AND
               catp[catfrm] = oldmark AND
               catp[catenc] # dropped THEN
                  BEGIN
                  catp[catenc] _ catp[catenc] + n;
                  EXIT;
                  END;
         % no-op if not found %
      RETURN;
      END.
      %.Pes;%
```

```
(sysenc) % return enable count given catchphrase adr %                    5F2
   PROCEDURE
      (catchname); % address of catchphrase %                             5F2A1
   LOCAL
      catp REF, % catchphrase stack pointer %
      oldmark; % caller's mark %
   oldmark _ [M];
   %catp _ &syscpe:
   % now search for instance of catchphrase %
      WHILE (&catp _ &catp - catsiz) > $syscstk DO
         IF catp[catloc] = catchname AND catp[catfrm] = oldmark THEN
            RETURN(catp[catenc]);
      % zero if not found %
   RETURN(0);
   END.
   %.Pes;%
```

```
(sysdrp) % drop catchphrase %                                          5F3
   PROCEDURE
      (catchname); % address of catchphrase %                          5F3A1
   % drops a catchphrase given address, or drops all catchphrases
   belonging to calling routine if argument is -1 %
   LOCAL
      prevcat REF, % pointer to previous catchphrase for caller %
      catp REF, % catchphrase stack pointer %
      oldmark; % caller's mark %
   oldmark _ [N];
   &catp _ &syscpe;
   &prevcat _ 0;
   % now search for instance of catchphrase and delete it %
      WHILE (&catp _ &catp - catsiz) > $syscstk DO
         IF catp[catfrm] = oldmark AND (catp[catenc] # dropped) AND
            ( catp[catloc] = catchname OR catchname=-1 ) THEN
               BEGIN
               % restore return location or move it to previous
               catchphrase %
               IF catp[catrtn]#0 THEN % return loc in it %
                  BEGIN
                  IF &prevcat THEN prevcat[catrtn] _ catp[catrtn]
                  ELSE [catp[catown].RH-1] _ catp[catrtn];
                  % mark the catchphrase to be dropped %
                     drpcat _ catp[catenc] _ dropped;
                  EXIT; % we know we are done here %
                  END
               ELSE
                  % mark the catchphrase to be dropped %
                     drpcat _ catp[catenc] _ dropped;
               IF catchname#-1 THEN EXIT;
               END
         ELSE IF catp[catown] = oldmark THEN &prevcat _ &catp;
            % save prev cat pointer %
         % ignore if does not belong to oldmark %
      IF NOT tsysgn AND drpcat THEN sysdpall();
      % no-op if not found %
   RETURN;
   END.
   %.Pes;%
```

```
(sysdpall) % drop all marked catchphrases %                          5F4
   PROCEDURE;
   % to be used by runtime routines only; reset drpcat falg to FALSE %
   (catp) REF; % temporary pointer for looping over catchphrase stack %
                                                                     5F4C
   &catp _ $sysctpe;
   WHILE (&catp _ &catp - catsiz) > $syscstk DO
      BEGIN
      IF catp[catenc]=dropped THEN
         sysdp1(&catp);
      END;
   drpcat _ FALSE;
   RETURN;
   END.
   %.Pes:%
```

```
(syscp1) % drop one catchphrase given catch stack pointer %                5F5
   PROCEDURE
      (catp REF); % catchphrase sack pointer of one to drop %
   % to be used by runtime routines only %
   IF &catp = &syscpe - catsiz THEN &syscpe_&catp % on top %
   ELSE
      BEGIN
      &syscpe _ &syscpe - catsiz; % step down one %
      % want to move from &catp+catsiz to &catp till we store in
      &syscpe-catsiz %
      R4.LH _ &catp+catsiz;
      R4.RH _ &catp;
      A1 _ &syscpe+catsiz;
      !BLT R4,@A1;
      END;
   RETURN;
   END.
   %.Pes;%
```

```
% Signal and Catchphrase State Saving Primitives %
   (sigsave) PROCEDURE  % save global signal state in signal stack %     5G1
      % FORMALS %
         (rloc);      % return location for error messages %            5G1A1
      % does nothing if no signal in progress for previous stack.  Uses
      sysstk (signalling stack).  Thus, if sysgn is non-zero sysstk MUST
      be set up.  This is done in syshelp.  It is reset in sigrestore.  %
      % DECLARATIONS %
         (tsigptr) REF;                                                  5G1C1
         (endadr);                                                       5G1C2
      IF sysgn THEN
         BEGIN
         [sysstk.RH][stksig] _ sysgn;
         FOR &tsigptr _ $sysgstk UP sigsiz UNTIL >= &sysgpt DO
            IF (tsigptr[sggn] = sysgn) AND (tsigptr[sgstk] = sysstk) THEN
               EXIT LOOP;
         IF &tsigptr + sigsiz > $sysge THEN
            sysrcv(stkoverflow,$"signal stack overflow",rloc);
         A1.LH _ $sysfrm;   A1.RH _ sysinfo _ &tsigptr;
         endadr _ &tsigptr + sigsiz - 1;
         !BLT A1,@endadr;
         IF &tsigptr = &sysgpt THEN &sysgpt _ &sysgpt + sigsiz;
            % We have added a new signal to the stack %
         END;
      RETURN;
      END.

   (sigrestore) PROCEDURE;  % restore global signal state from sig stack %
                                                                        5G2
      % NOTE: *** CAN NOT USE ANY REGISTERS EXCEPT A1 *** %
      % Uses sysbtm, absolute bottom of the cutrent stack %
      % DECLARATIONS %
         (tsigptr) REF;                                                  5G2C1
         (endadr);                                                       5G2C2
         (signo);                                                        5G2C3
      IF signo _ [sysbtm.RH][stksig] THEN
         BEGIN
         FOR &tsigptr _ $sysgstk UP sigsiz UNTIL >= &sysgpt DO
            BEGIN
            IF (tsigptr[sggn] = signo) AND (tsigptr[sgstk] = sysbtm) THEN
               EXIT LOOP;
            END;
         END
      ELSE &tsigptr _ &sysgpt;
      IF &tsigptr = &sysgpt THEN
         BEGIN
         sysinfo _ sysstk _ sysfrm _ 0;
         R1.LH _ $sysfrm;   R1.RH _ $sysfrm + 1;
         END
      ELSE
         BEGIN
         R1.LH _ sysinfo _ &tsigptr;   R1.RH _ $sysfrm;
         sysstk _ sysbtm;
         END;
      endadr _ $sysfrm + sigsiz - 1;
      !BLT R1,@endadr;
```

```
    RETURN;
    END.

(oopsig) % pop signal-in-progress frame from sysgstk: called from systrm
and sysresume %                                                          5G:
    % This routine reverses pushsig operation - recover on underflow %
    PROCEDURE;
    (temp);                                                              5G3(
    (svsybtm);                                                           5G3[
    (tsigptr) REF;                                                       5G3{
    (endadr);                                                            5G3}
    % unmark all catchphrases that saw this signal %
       temp _ bits[sysgn] .X -1;
       FOR &tsigptr _ $syscstk UP catsiz UNTIL >= &syscpe DO
          IF tsigptr[catstk] = syssstk THEN
             tsigptr[catmrk] _ tsigptr[catmrk] .A temp;
    IF sysinfo % the signal is on the stack: remove it % THEN
       BEGIN
       % check for underflow %
          IF (&sysgpt-sigsiz) < $sysgstk THEN
             sysrcv(programbug,$"signal stack underflow",0);
       % find this signal on sig stack and collapse the sig stack %
          IF sysinfo = &sysgpt - sigsiz THEN
             &sysgpt _ sysinfo
          ELSE
             BEGIN
             &sysgpt _ &sysgpt - sigsiz;
             R4.LH _ sysinfo + sigsiz;
             R4.RH _ sysinfo;
             endadr _ &sysgpt;
             !BLT R4,@endadr;
             % sysinfo will be set in sigrestore; should be set by here
             to reflect true state of the world %
             END;
       END;
    % restore prior signal %
       svsybtm _ sysbtm := syssstk;
    % move from sig stack to globals %
       sigrestore();
       % sysgn was restored by sigrestore %
       IF [sysbtm.RH][stksig] THEN BUMP DOWN [sysbtm.RH][stksig];
       IF tsysgn THEN BUMP DOWN tsysgn;
       sysbtm _ svsybtm;
    RETURN;
    END.
    %.Pes;%
```

```
% Signal and Catchphrase State Query Primitives %
   (syspro) % is given stack frame for a PROCEDURE? %                    5H1
      % Return TRUE if port id P (frame pointer) is that of a procedure,
      false if that of a coroutine. Method: look at return location - if
      coroutine, it contains port id of caller which points into stack
      rather than to code (and, of course, there is no code on the stack.
      We look first at stack pointed to by sysbtm, then go on through all
      other stacks if necessary. %
      PROCEDURE
         (port): % the port id %                                        5H1B1
      (pid);                                                            5H1C
      (teststk);                                                        5H1D
      IF (pid _ [port.RH-1].RH) IN [sysbtm.RH, (sysbtm.RH-sysbtm.LH).RH]
      THEN
         RETURN (FALSE);
      IF NOT (teststk _ linlink) THEN RETURN(TRUE);
      LOOP % over all stacks from the beginning: must end when stkLink
      field 0 at end of chain %
         BEGIN
         IF pid IN [teststk.RH, (teststk.RH-teststk.LH).RH] THEN
            RETURN (FALSE);
         IF NOT (teststk _ [teststk.RH][stklink]) THEN RETURN(TRUE);
         END;
      END.

   (sysbak) % given stack frame,, back up one frame %                    5H2
      % Return frame pointer (port id) P for the frame that is "back" one
      frame in stack (back=toward bottom) %
      PROCEDURE
         (port); % the port id = frame pointer %                        5H2B1
      RETURN ( [port.RH] ); % gets previous mark %
      END.

   (sysclr) % return a procedure's caller's address %                   5H3
      % sysclr() inside a procedure will give you your return address %
      PROCEDURE;
      RETURN( [ [M.RH].RH-1 ] );
      END.
      %.Pes;%
```

```
(errmsg) % error message writing procedure %                                51
   PROCEDURE
      (jfn, % output jfn %
      why REF. % reason %
      loc): % associated location %
   % put out CRLF %
      !bout(jfn,EOL);
      !sout(jfn,$"L10 RUNTIME ERROR: "+chbmty,0);
   % find reason %
      IF why.L > why.M OR why.L NOT IN [1,200] THEN
         &why _ $"bad error string passed to ERRMSG";
      !sout(jfn, &why+chbmty,0);
   % put out location %
      !bout(jfn,EOL);
      !sout(jfn,$"LOCATION: "+chbmty,0);
      CASE loc OF
         =0: !sout(jfn,$"unknown location (zero given)"+chbmty,0);
         ENDCASE
            BEGIN
            IF NOT SKIP !nout(jfn,loc,8) THEN !JFCL;
            !sout(jfn,$" octal"+chbmty,0);
            END;
      !bout(jfn,EOL);
   RETURN;
   END.
   %.Pes;%
```

```
% Stack allocation/deallocation %
   (alostk)   % alocate & openport coroutine stack %
   PROCEDURE (astk REF,nstksize, zone, acoroutine REF, nargs %, + nargs
   arguments for the coroutine % % => [port] coroutine's results %);     5J1
      % Procedure description
         FUNCTION
            A call on this procedure is logically equivalent to allocating
            a new stack and then OPENPORTing a coroutine that lives at the
            base of the new stack
         ARGUMENTS
            none
         RESULTS
            proc-value
         NON-STANDARD CONTROL
            will generate an ABORT signal if no space available for new
            stack
         GLOBALS
            none
         %
      % Declarations %
         (stkptr);                                                        5J1B1
         (constk);                                                        5J1B2
         (temp);                                                          5J1B3
         (i);                                                             5J1B4
      % make sure a coroutine address was passed to us %
         IF NOT &acoroutine THEN
            ABORT( programbug, $"ALOSTK: no coroutine address passed" );
      % get a new stack of size nstksize + overhead words from specified
      zone %
         IF NOT &astk THEN
            IF NOT (&astk _ getblk( nstksize + stackdecsize, zone )) THEN
               ABORT( programbug, $"Insufficient room in zone for new
               stack");
      % initialize the stack descriptor for this new stack %
         &astk _ &astk + stackdecsize;
         A1 _ -nstksize; !HRL A1,A1; A1.RH _ &astk;
         stkptr _ A1;
         astk[stkmsave] _ astk[stkssave] _ astk[stksig] _ astk[stkcalo] _
         astk[stkmcalo] _ astk[stksdalo] _ 0;
         astk[stkzone] _ zone;
         astk[stksize] _ nstksize;
         IF ( constk _ astk[stklink] _ ([sysbtm.RH][stklink] := stkptr) )
         THEN
            BEGIN
            [constk.RH][stkblink] _ stkptr;
            END;
         astk[stkblink] _ sysbtm;
      % set linlink to point to first stack in chain %
         IF NOT linlink THEN
            BEGIN
            linlink _ $gstack;
            linlink.LH _ -gstksz;
            END;
      % put first word on bottom of stack %
         astk _ $sysufl;
      % save state of old stack (M, S, & signal status) %
```

```
    [sysbtm][stkssave] _ M - 20000028;
    [sysbtm][stkmsave] _ [M];
    sigsave( [M-1] );
% indicate which stack we came from, and indicate where new stack is
%
    pcstack _ sysbtm := stkptr;
    sysse _ &astk + nstksize;
% set up signal status of new stack (no signals in progress) %
    sigrestore();
% move S to 1st entry of new stack %
    S _ stkptr;
% move arguments for coroutine to new stack in reverse order %
    WHILE (i _ 1) <= nargs DO
        BEGIN
        temp _ [$nargs - 1 - nargs + i];
        !PUSH S,temp;
        END;
% set CORETURN location in our caller ([[M]+1]) to be our return
location ([M-1]); in otherwords, make it look like our caller did an
OPENPORT to the new coroutine on the new stack .  The coroutine
location is one cell beyond the chain word of a frame (pointed to by
M for the current frame.)  It is NOT a stack pointer, but rather has
a right half which is a pointer to code which presumably did an
openport.  The word one cell before the chain word is a return
location if the frame is a procedure.  If the frame is for a
coroutine, the word is a stack pointer tcthe frame which openported
it (if it has not been stored away in a local).  This word is
accessible by the word PORT.%
    [[M.RH] + 1] _ [M.RH - 1];
    R6 %mreg% _ [M.RH];
% setup to OPENPORT coroutine %
    A1 _ &acoroutine;
% switch stacks; i.e., make new stack current %
    M _ stkptr;
% now OPENPORT new coroutine %
    !PUSHJ S,1(A1);
% we should never get here %
    ABORT( programbug, $"ALOSTK: System screwed up");
END.

%.Pes;%
```

```
(dalcstk)   % deallocate stack %
PROCEDURE (astk);                                                    5J2
   % Procedure description
      FUNCTION
         A call on this procedure is logically equivalent to deallocates
         a stack; it propogates a note unwind through the entire stack,
         unchains it from the stack chain, then frees associated
         allocated storage.
      ARGUMENTS
         none
      RESULTS
         proc-value
      NON-STANDARD CONTROL
         An ABORT signal is generated if an attempt is made to
         deallocate the default system stack, GSTACK;  also, if the
         storage allocator generates and error; also, if a stack tries
         to deallocate itself.
      GLOBALS
         none
      %
   % Declarations %
      (fptr);                                                        5J2B1
      (bptr);                                                        5J2B2
      (sigp) REF;                                                    5J2B3
      (endadr);                                                      5J2B4
      (catp) REF;                                                    5J2B5
   % Generate ABORT if this is gstack or if we stack is trying to
   deallocate itself or if stack is currently being dealocated by anyone
   else %
      IF [astk.RH] = $gstack THEN
         ABORT(programbug, $"Attempt to deallocate system stack.");
      IF [astk.RH] IN [sysbtm.RH, (sysbtm.RH-sysbtm.LH).RH] THEN
         ABORT(programbug, $"Attempt by stack to deallocate itself.");
      IF [astk.RH][stkdalo] THEN
         ABORT(programbug, $"Attempt to deallocate a deallocated
         stack.");
   % Propogate NOTE(unwind) down entire stack to permit cleanup %
      sigsave( M );
      R1.LH _ - [astk.RH][stksize];   R1.RH _ astk;
      astk _ R1; % to insure astk LH set %
      [R1.RH][stkdalo] _ sysbtm;
      [sysbtm.RH][stkssave] _ [R1.RH][stksdalo] _ S :=
      [R1.RH][stkssave];
      [sysbtm.RH][stkmsave] _ [R1.RH][stkmdalo] _ M :=
      [R1.RH][stkmsave];
      sysbtm _ R1;
      %sigrestore(); not actually done: we aren't interested in (other)
      nested signals on the stack being deallocated %
      NOTE(unwind);
      M _ [sysbtm.RH][stkmdalo];
      S _ [sysbtm.RH][stksdalo];
      sysbtm _ [sysbtm.RH][stkdalo];
      sigrestore();
   % Get rid of signals in progress associated with stack being
   deallocated %
      FOR &sigp _ &sysgpt-sigsiz DOWN sigsiz UNTIL < $sysgstk DO
```

```
        BEGIN
        IF sigp[sgstk] = astk THEN % signal in stack being deallocated
        %
           BEGIN
           IF tsysgn THEN BUMP DOWN tsysgn;
           IF sysinfo >= &sigp THEN
              sysinfo _ sysinfo-sigsiz; % the global signal will move
              down the stack %
           IF &sigp = &sysgpt-sigsiz THEN
              &sysgpt _ &sysgpt -sigsiz
           ELSE
              BEGIN
              % check for underflow %
                 IF (&sysgpt-sigsiz) < $sysgstk THEN
                    sysrcv(programbug,$"signal stack underflow",0);
              &sysgpt _ &sysgpt - sigsiz;
              R4.LH _ &sigp + sigsiz;
              R4.RH _ &sigp;
              endadr _ &sysgpt;
              !BLT R4,@endadr;
              END;
           END;
        END;
     % Drop catchphrases associated with stack being deallocated %
       FOR &catp _ $syscstk UP catsiz UNTIL >= &syscpe DO
        IF catp[catstk] = astk THEN
           drpcat _ catp[catenc] _ dropped;
       IF NOT tsysgn AND drpcat THEN sysdpall();
     % Unchain stack %
       fptr _ [astk.RH][stklink];
       bptr _ [astk.RH][stkblink];
       IF bptr THEN [bptr.RH][stklink] _ fptr;
       IF fptr THEN [fptr.RH][stkblink] _ bptr;
       IF NOT gstack[stklink] THEN linlink _ 0;
     % Deallocate %
       IF [astk.RH][stkzone] THEN
          BEGIN % stack was allocated %
          IF NOT freeblk([astk.RH]-stackdecsize, [astk.RH][stkzone]) THEN
             ABORT( programbug, $"Error on stack deallocation");
          END;
     RETURN;
     END.

     %.Pes;%
```

```
% String Construction.Post=Off;%
   (bsc) %Begin string construction.  Called with STID (or pointer to string
   with stastr TRUE) of the destination.%
                                                                        6A
      PROCEDURE (dest);
      rplsid _ dest;
      %set up byte pointer and SAR for string construction%
      nsdbpt _ chbmty + $sar;
      sar.L _ empty;
      %reset stack for modified clist entries%
      clchng _ clmpty;
      RETURN;
      END.

   (esc) %End string construction.%
                                                                        6E
      PROCEDURE;
      IF rplsid.stastr THEN %A-string%
         BEGIN
         *[rplsid.stadr]* _ *sar*;
         RETURN;
         END;
      filesc();
      RETURN;
      END.

   (kps) %argument points to string to be appended to the statement being
   constructed%
                                                                        6D
      PROCEDURE (asfrom);
      REF asfrom;
      LOCAL bptr; %byte pointer for reading characters%
      IF (sar.L _ sar.L + asfrom.L) > sar.M THEN
         ABORT(saroverflow,$"string too long");
      bptr _ chbmty + $asfrom;
      a2 _ asfrom.L;
      a3 _ nsdbpt;
      a4 _ bptr;
      UNTIL (a2 _ a2-1) < 0 DO |a3 _ a1 _ |a4;
      nsdbpt _ a3;
      RETURN;
      END.

   (apachr) %passed character as argument.  appends to the statement being
   constructed in sar.%
                                                                        6C
      PROCEDURE (ch);
      IF (sar.L _ sar.L+1) > sar.M THEN ABORT(sarcverflow,$"string too long");
      |nsdbpt _ ch;
      RETURN;
      END.

   (aptstr) %Append tstring.  Argument is pointer to two T-pointers which
   delimit the substring to be appended to the statement currently being
   constructed.%
                                                                        6B
```

```
PROCEDURE (tp);
LOCAL
   cl,       %location of record for a cl to be updated%
   end,      %location of end of cl table%
   cltab,    %location of cl table%
   flhc,     %file header loc%
   ch,       %character read from the tstring%
   bfrom,     %source byte pointer%
   lenstat,  %number of chars from string%
   lenblank; %number of chars from string%
REF cl;
IF [tp] # [tp+2] OR [tp+1] <= empty THEN
   BEGIN
   modeset _ 0; %reset global modeset flag %
   sysrcv(programbug,$"illegal text-pointer pair",sysclr());
   END;
IF [tp+1] >= [tp+3] THEN
   BEGIN
   modeset _ 0; %reset global modeset flag %
   RETURN;
   END;
INVOKE(catch);
stcwrk _ [tp];
stcwr1 _ [tp+1];
fechcl (forward, $stcwrk);
% stcwrk[2] now contains length + 1 of source string %
% calculate number of chars to take from source string %
CASE stcwrk[2] OF
   <= [tp+1] :
      BEGIN
      lenstat _ 0;
      lenblank _ [tp+3] - [tp+1];
      END;
   < [tp+3] :
      BEGIN
      lenstat _ stcwrk[2] - [tp+1];
      lenblank _ [tp+3] - stcwrk[2];
      END;
   ENDCASE
      BEGIN
      lenstat _ [tp+3] - [tp+1];
      lenblank _ 0;
      END;
&cl _ cltab _ [clstad].clbuff;
   %address of start of clist%
end _ &cl + [clstad].clcnt * cll;
   %end of clist%
DROP(catch);
IF modeset THEN
   BEGIN % mode set now in operation %
   modeset _ 0; %reset the global flag%
   UNTIL (lenstat _ lenstat-1) < 0 DO
      BEGIN
      ch _ READC($stcwrk);
      %now append the character%
      CASE modeshift OF %check for forced case%
```

```
            =0: apachr(IF ch IN ['a,'z] THEN ch-40B ELSE ch);
               %forced upper case%
            =1: apachr(IF ch IN ['A,'Z] THEN ch+40B ELSE ch);
               %forced lower case%
            ENDCASE apachr(ch); %no case change%
         END;
      END
   ELSE %no mode change. just copy across%
      BEGIN
      IF NOT mkrstay THEN
         IF NOT rplsid.stastr THEN
            BEGIN
            IF rplsid.stfile = [tp].stfile THEN
               UNTIL &cl = end DO
                  BEGIN
                  IF cl.clst1 = [tp] AND
                     cl.clcc1 IN [[tp+1],[tp+3]) AND
                     NOT cl.clfixed THEN
                        BEGIN
                        cl.clfixed _ TRUE;
                        cl.clcc1 _
                           sar.L + 1 + cl.clcc1 - [tp+1];
                        PUSH &cl ON clchng;
                        END;
                  &cl _ &cl + cll;
                  END
            ELSE cldsr(tp);
            END;
      IF (sar.L _ sar.L + lenstat) > sar.M THEN
         ABORT(saroverflow,$"string too long");
      a4 _ stcwrk[4];  %byte pointer made by fechc1%
      a2 _ lenstat;
      a3 _ nsdbpt;
      UNTIL (a2 _ a2-1) < 0 DO |a3 _ a1 _ |a4;
      nsdbpt _ a3;
      END;
   UNTIL (lenblank _ lenblank-1) < 0 DO apachr(SP);
   mkrstay _ FALSE;
   RETURN;
   (catch) CATCHPHRASE;                                          6E21
      % make sure modeset is zero if the world gets away from us %
      BEGIN
      IF SIGNAL=unwind  THEN
         modeset _ 0;
      CONTINUE;
      END;
   END.

(cldsr) %Called with address of two tpointers.  For all clist entries, if
between these two pointers, then set to "deleted".%
                                                                 6F
   PROCEDURE (tp);
   LOCAL end, cl;
   REF cl;
   &cl _ [clstad].clbuff;
      %address of start of clist%
```

```
    end _ &cl + [clstad].clcnt * cll;
       %end of clist%
    UNTIL &cl >= end DO
       BEGIN
       IF cl.clst1 = [tp] AND
          cl.clcc1 IN [[tp+1],[tp+3]) AND
          NOT cl.clfixed THEN
          BEGIN
          cl.clst2 _ cl.clst1 := endfil;
          cl.clcc2 _ [tp+1];
          END;
       &cl _ &cl + cll;
       END;
    RETURN;
    END.

% Reading characters from SDB%
   (fechc1)                                                    71
      %Documentation%
         %This routine is called to initialize a work area for reading
         characters from a statement.  Arguments are: direction of reading
         characters (=0 then backwards) and the address of the 7 word work
         area (of which the last 2 words are no longer used).
         If characters are to be read from a statement then
            when calling FECHC1, the first two cells of the work area must
            contain a Tpointer.  A character count of one indicates the first
            character of the statement. FECHC1 will initialize the rest of the
            work area. The first word of the word area always has the PSID of
            the statement.  The second word has the character count.  The
            third word contains a bound on characters to be read.  ENDCHR's
            are returned after reach this bound.  The fourth word has the
            direction of readout for use by readc.  A READC(x) actually
            results in the value of x being loaded into register wa followed
            by a JSP a4,readc.
            The fifth word of the work area contains a byte pointer to the
            character last read from the statement. Thus an ILDB instruction
            may be used to get the next character if the direction is forward.
             If the direction of reading is backward, then the byte pointer is
            decremented by in-line code.
            To read characters from the statement execute a READC(x) where the
            value of x is the address of the work area to be used.  The
            character is returned as the value of the READC.  Subsequent
            READC's will return the following characters.
            To change position or direction within the statement the work area
            must be reinitialized by calling FECHC1 again, as described above.
             There may however be more than one work area currently in use,
            and these may be changed independently.
         If characters are to be read from an A-string then
            the first word of the work area contains the address of the
            A-string instead of a PSID.  The second word is 1 if the first
            character of the string is to be read next, two if the second,
            etc.  Characters may be read out of ar A-string in either
            direction, just like a statment.  Endcharacters are returned when
            the string is exhausted.%
      PROCEDURE (dir, worka);
      LOCAL
```

```
      wp,   %word postion of the starting character%
      cp,   %character postion of the starting character%
      stdb,  %stdb for statement%
      rng,  %location of ring element%
      addr; %address of the word containing starting character%
   REF rng;
   %set work+2 to bound%
   IF [worka] = endfil THEN %set to null string%
      BEGIN
      [worka+3] _ 2; %readc will return ENDCHR%
      RETURN;
      END
   ELSE IF [worka].stastr THEN %A-string%
      BEGIN
      addr _ [worka].staddr;
      [worka+2] _ IF pe THEN pe ELSE [addr].L + 1;
      addr _ addr + 1;
      END
   ELSE %SDB%
      addr _ flfechc1( dir, worka : stdb );
   %find word and character position%
   IF dir THEN %scan forward%
      BEGIN
      [worka+3] _ 1; %direction%
      END
   ELSE %scan backwards%
      BEGIN
      [worka+2] _ IF ps THEN ps ELSE 1; %change bound%
      [worka+3] _ 0; %direction%
      END;
   %make byte pointer to the previous (forward) or current (backward)
   character%
   DIV ([worka+1]-[worka+3])/5, wp, cp;
   [worka+4] _ 440700B6 - cp * 7000086 + addr + wp;
   RETURN;
   END.

(openswork) PROCEDURE;                                                  7E
   fechc1(1, $swork);
   RETURN;
   END.

(savpos) PROCEDURE;                                                     7C
   PUSH ps ON btwstk;
   PUSH pe ON btwstk;
   RETURN;
   END.

(respos) PROCEDURE;                                                     7D
   POP btwstk TO pe;
   POP btwstk TO ps;
   RETURN;
   END.

(xxreadc) PROCEDURE; %This is the routine that is called to read a
character by the READC construct.  Code to call is JSP a4,readc so return
```

```
loc is in A4.%                                                          7E
   (readc):                                                             7E1
   %cant have locals since is not called by usual procedure linkage.  wa is
   a register containing the address of the work area.%
   !SKIPG a1,3(wa);
      !JRST readc1;
   !SUBM a1,readend;
   %forward scan%
      !AOS a1,1(wa);
      !CAMLE a1,2(wa);
         !JRST rdend1;
      !ILDB a1,4(wa); %get the character%
      !JRST 0(a4);
   (readc1):                                                            7E6
   !JUMPN a1,readend;
   %backward scan%
      !SOS a1,1(wa);
      !CAMGE a1,2(wa);
         !JRST rdend2;
      %back up byte pointer%
      !MOVE a1,4(wa);
      !ADD a1,=7B10;
      !CAIG a1,0;
      !SUB a1,=430000000001B;
      !MOVEM a1,4(wa);
      !LDB a1,a1; %get the character%
      !JRST 0(a4);
   (rdend2): !AOSA 1(wa); %adjust for backward overrun, NOTE SKIP%      7E9
   (rdend1): !SOS 1(wa); %adjust for forward overrun, SKIPPED OVER%     7E10
   (readend): %out of bounds%                                           7E11
      a1 _ ENDCHR;
      !JRST 0(a4);
   END.

% A-string routines%
   (asrref) %create a pointer to astring with stastr field set.%
                                                                        8A
      PROCEDURE (ast);
      ast.stastr _ TRUE;
      RETURN (ast);
      END.

   DECLARE % byte pointers for chbptr %
      chbarray=(
         350700000001B,
         260700000001B,
         170700000001B,
         100700000001B,
         107000000001B);

   DECLARE cshift=(7,14,777761B %-15%, 777770B %-8%, 777777B %-1%);
   DECLARE ascomm=(0,77489,777767,7777777B5,-400B);
   EXTERNAL cmbstr, comstr, apstore; % inter-procedure refs %
   (lochr) % load character from a-string %                             8F
      PROCEDURE
         (astr, % a-string address %
```

```
      charno); % index, 1=first char %
   % return a specified character from an astring %
      !MOVE A1,charno; % get index %
      !MOVEI A1,-1(A1);
      !IDIVI A1,5;  % char-1/5 %
      !ADD A1,astr; % add address of string %
      !MOVE A1,1(A1);  % get word from string %
      !ROT A1,wcshift(A2);  % rotate and mask %
      !ANDI A1,177B;  % faster than LDB %
      RETURN;
   END.
(chbptr) % return a-string char byte-pointer given index %                    8G
   PROCEDURE
      (charno); % character index 1=first char  if LDB %                      8G1A
         % NOTE: 0=first char if do ILDB on resulting pointer %
   % return a partial byte pointer given a character number.  The resulting
   byte pointer needs to have the address of the astring added to it %
      !SKIPG A1,charno;  %check for zero char count %
      !JRST chbpt1;
      !MOVEI A1,-1(A1);  %subtract one %
      !IDIVI A1,5;
      !ADD A1,chparray(A2);  %get partial pointer %
      RETURN;
      (chbpt1):                                                               8G2C
         RETURN(chbmty);
   END.
(apchr) % append a character to a-string %                                    8H
   PROCEDURE
      (char, % the character to append %
      astr); % the a-string address %
   % Append the given character to the specified astring.  If the string is
   at it's max length a HELP(stringoverflow,astr) results %
      LOOP
         BEGIN
         A2 _ [astr];  % get max,,len from string %
         !MOVSI A1,1(A2); % get len+1 %
         !CAMG A1,A2;  % over max ? %
         EXIT LOOP;
         astr _ syssov(astr, astr.L+1); % get new string %
         END;
      !HLRM A1,@-3(M);  % store len+1 %
      !MOVEI A2,0(A2);  % isolate new length-1 = old len %
      (apstore):                                                              8H2D
         !IDIVI A2,5;
         !ADD A2,chparray(A3);  % get partial pointer %
         !ADD A2,astr;  % plus string address %
         !MOVE A1,char;  % get character %
         !DPB A1,A2;  % and store it %
         RETURN;
   END.
(stbptr) % non-astring character byte pointer%                                81
   PROCEDURE (charno);
   %creates a partial byte pointer to the character determined by the
   character number passed as argument.  The resulting byte pointer needs
   to have the address of the string (NOT ASTRING) added to it.%
   LOCAL wp, cp;
```

```
    IF charno = empty THEN RETURN (440700000000B);
    DIV (charno-1) / 5, wp, cp;
    RETURN (cnbarray[cp] + wp - 1);
    END.

(ascom) PROCEDURE(astr1, astr2, relation);                                    8,
    % accepts addresses of 2 a-strings.  returns logical value of astr1
    <relation> astr2 %
    LOCAL reltabi;
    r1 _ -1;  % set flag in r1 - means must do all of compare %
    r2 _ relation;
    (comstr):  % called by ascom and compas %                                 8J5
       % r1 is a flag: =0 means return false if a mismatch (compas call). <0
       means must compare entire string and return reltab (ascom call). >0
       means return the sign of astr1 = astr2 (cmpstr call) %
       IF [astr1].L # [astr2].L THEN BEGIN
          !JUMPE r1,ascom6;  % RETURN(FALSE) if called from compas %
          !SKIPE lngflg;  % cmpstr: IF lngflg AND lengths not equal %
          !JUMPG r1,ascom5;  % cmpstr call - compare lengths %
          !CAIN r2,2;  % is relation = ? %
          !JRST ascom6;  % yes.  RETURN(FALSE) %
          !CAIN r2,6;  % if relation # ? %
          !JRST ascom7  % yes.  RETURN(TRUE) %
          END;
       !IMULI r2,3;  % r2 _ r2*3+1 %
       !MOVEI r2,1(r2);  % plus one %
       a3 _ [m-2]; % $astr1 %
       a4 _ [m-3];  % $astr2 %
       !HRRZ a1,0(a3);  % len of astr1 %
       !HRRZ a2,0(a4);  % len of astr2 %
       !CAIG a2,0(a1);  % find min len %
       !MOVEI a1,0(a2);  % now a1 has the min %
       !IDIVI a1,5;  % length/5: a1=number of words, a2=mask index %
       (ascom2):                                                              8J5I
          !MOVEI a3,1(a3);  % add one to pointer %
          !MOVEI a4,1(a4);  % ditto %
          !MOVE r3,0(a3);  % get a word of string %
          !TRZ r3,1;  % turn off low order bit %
          !SOJL a1,ascom1;  % last word of string ? %
          !MOVE r4,0(a4);  % get other word %
          !TRZ r4,1;  % and turn of low order bit %
          !SUB r3,r4;  % now, take difference %
          !JUMPE r3,ascom2;  % if equal, keep going %
          !JUMPE r1,ascom6;  % quit here if we can (i.e. from compas) %
       (ascom3):                                                              8J5I
          !TLNE r3,485;  % compute sign of result:  sign bit on? %
          !SKIPA r3,=-1;  % yes, get -1 %
          !MOVEI r3,1;  % no, get one %
       (ascom4):                                                              8J5I
          !JUMPG r1,ascom8;  % cmpstr call - return sign, not reltab %
          !ADDI r3,0(r2);  % add in reltab index %
          RETURN(reltab[r3]);  % return result from table %
       (ascom1):                                                              8J5I
          !JUMPE a2,ascom5;  % last word of str.  any chars in it? %
          !AND r3,ascomm(a2);  % and off extraneous chars %
          !MOVE r4,0(a4);  % get word from other string %
```

```
        !AND r4,ascomm(a2);  % and it also %
        !SUB r3,r4;  % take difference %
        !JUMPN r3,ascom3;  % check sign if not equal %
      (ascom5):                                                    8J5F
        !HRRZ r3,0-2(m);  %Last words equal.  compare lengths %
        !HRRZ r4,0-3(m);  % get other length %
        !SUBI r3,0(r4);  % take difference %
        !JUMPE r3,ascom4;  % equal, return result %
        !JRST ascom3;  % not equal, compute sign %
      (ascom6):                                                    8J5G
        RETURN(FALSE);
      (ascom7):                                                    8J5F
        RETURN(TRUE);
      (ascom8):                                                    8J5S
        RETURN(r3); % return sign for cmpstr call %
  NULL;
  END.
(compas) PROCEDURE(astr1, astr2);                                  8K
  % compares the contents of two a-strings.  Returns true if the contents
  match, false otherwise %
  r1 _ 0; r2 _ 2; % relation = true %
  GOTO comstr;
  END.
(cmpstr) PROCEDURE(astr1, astr2);                                  8L
  % return -1 if astr1 < astr2, 0 if equal, 1 if astr1 > astr2 %
  r1 _ 1; % set flag >0 means return sign value %
  R2 _ 0; % condition cannot be 2 or 6 %
  GOTO comstr;
  END.
(repchr) %replace a character in an astring%
                                                                   8M
  PROCEDURE (char, ast, chrno);
  %------------------%
  LOCAL cnt;
  REF ast;
  !MOVE a2,chrno; % the char number %
  !MOVEI a2,-1(a2);  % minus one %
  !JRST apstore
  END.

(apblnk) PROCEDURE(ast, cnt);                                      8N
  % Append cnt blanks to the designated astring. %
  REF ast;
  FOR cnt DOWN UNTIL < 1 DO *ast* _ *ast*, SP;
  RETURN;
  END.

(apsr) %The purpose of this routine is to apperd one A-string onto another.
 Arguments are two A-string addresses.  The first string is appended to the
second.  A HELP signal (stringoverflow) is generated if the maximum length
of the latter string is too short to contain the result.%
                                                                   8O
  PROCEDURE (asfrom, asto);
  LOCAL
    nlen, % new string length %
    wp,    % word position %
```

```
      cp,    % character position %
      bfrom, % byte pointer in source string %
      bto;   % byte pointer in destination string %
   REF asfrom, asto;
   IF (nlen _ asto.L + asfrom.L) > asto.M THEN
      &asto _ syssov(&asto, nlen); % get help %
   bfrom _ chbmty + &asfrom;
   bto _ chbptr(asto.L) + &asto;
   asto.L _ nlen;
   a2 _ asfrom.L;
   UNTIL (a2 _ a2-1) < 0 DO |bto _ a1 %use a1% _ |bfrom;
   RETURN;
   END.
```

(cpysr) %To copy one A-string into another, call this routine with the address of the source string and the address of the destination string (in that order). An ERROR is generated if the maximum length of the destination string is shorter than the source string.%

```
   PROCEDURE (asfrom, asto);
   REF asfrom, asto;
   CASE asfrom.L OF
      > asto.M :
         BEGIN
         &asto _ syssov(&asto, asfrom.L); % get help %
         REPEAT CASE;
         END;
      <= empty : asto.L _ empty;
      ENDCASE
         BEGIN
         asto.L _ asfrom.L;
         mvbfbf(&asfrom+1, &asto+1, (asfrom.L+4)/5);
         END;
   RETURN;
   END.
```

(choair) %extract substring and append to another string%

```
   PROCEDURE(ast1, lower, upper, ast2);
   LOCAL nlen, bto, bfrom, lenstr, lenblank;
   REF ast1, ast2;
   IF upper < lower THEN RETURN;
   lower _ MAX(1, lower);
   CASE ast1.L OF
      < lower :
         BEGIN
         lenstr _ 0;
         lenblank _ upper-lower+1;
         END;
      < upper :
         BEGIN
         lenstr _ ast1.L-lower+1;
         lenblank _ upper-ast1.L;
         END;
      ENDCASE
         BEGIN
         lenstr _ upper-lower+1;
```

```
        lenblank _ 0;
        END;
    LOOP
        IF (nlen _ ast2.L + lenstr + lenblank ) <= ast2.M THEN EXIT
        ELSE &ast2 _ syssov(&ast2, nlen); % get help %
    bto _ chbptr(ast2.L) + &ast2;
    bfrom _ chbptr(lower-1) + &ast1;
    ast2.L _ nlen;
    a2 _ lenstr;
    a3 _ bto;
    a4 _ bfrom;
    UNTIL (a2 _ a2 - 1) < 0 DO |a3 _ a1 _ |a4;
    a2 _ lenblank;
    a1 _ SP;
    UNTIL (a2 _ a2 - 1) < 0 DO |a3 _ a1;
    RETURN;
    END.

(syssov) % string overflow - get HELP routine %                                  8F
    % attempt to get new string address from routine up in thread of
    control.  If get it, broadcast new address via NOTE and return it.
    Otherwise ABORT. %
    PROCEDURE
        (astr, % A-string address of too-short string %
        nlen); % required string length (total length) %
    LOCAL nastr; % new astring address %
    IF HELP(stringoverflow,astr, nlen: nastr) = gothelp THEN
        NOTE(changestring,astr,nastr)
    ELSE ABORT(stringoverflow,$"string too long");
    RETURN(nastr);
    END.
(mvbfbf) %This routine moves a buffer of words.  Arguments are address of
source, address of destination, and number of words to transfer.  It
returns the address of the last word into which data was moved.%
                                                                                 8S
    PROCEDURE (bfrom, bto, nw);
    LOCAL lw;
    IF nw = 0 THEN RETURN;
    lw _ nw + bto - 1; %last word to be transferred to%
    IF bto IN (bfrom, bfrom+nw) THEN
        BEGIN
        a1 _ bfrom;
        a2 _ nw; a3 _ a2;
        a2 _ a2 + a1; %pointer into source%
        a3 _ a3 + bto; %pointer into destination%
        UNTIL (a2 _ a2-1) < a1 DO
            BEGIN
            a3 _ a3 - 1;
            [a3] _ [a2];
            END;
        END
    ELSE %can use block transfer instruction%
        BEGIN
        !HRL a1,bfrom; !HRR a1,bto; !BLT a1,@lw
        END;
    RETURN (lw);
```

```
    END.

DECLARE hshmsk=(
    774B9,
    7777687,
    777777755,
    77777777742,
    7777777777768);

(hash) %An a-string address as argument.  The hash code for that string is
returned.%                                                                  8L
    PROCEDURE (ast);
    LOCAL nw, i, oldl, cp;
    REF ast;
    IF (oldl _ ast.L) <= empty THEN RETURN(0);
    DIV (oldl + 4)/5, nw, cp;
    FOR i _ 1 UP UNTIL = nw DO ast[i] _ ast[i] .A hshmsk[4];
    ast[nw] _ ast[nw] .A hshmsk[cp];
    RETURN(zhash(&ast));
    END.

(zhash)   % hashes string assuming extra bits are zero %
PROCEDURE (ast REF);                                                        8V
    % Procedure description
        FUNCTION
            This procedure  is called by the middle end when nlsbe is started
            up rather than hash to prevent the creation of dirty pages.
            Specifically, when the dispatch table for a subsystem is set up
            "zhash" will not insert zeros, thus zhash will not cause a dirty
            page.
        ARGUMENTS
            address of string to be hashed
        RESULTS
            hash value
        NON-STANDARD CONTROL
            none
        %
    % Declarations %
        (gen1);                                                             8V2A
        (gen2);                                                             8V2E
        (oldl); % number of characters in the string %                      8V2C
        (nw); % number of words used to store characters %                  8V2D
        (cp);                                                               8V2E
    % check for empty string %
        IF (oldl _ ast.L) <= empty THEN RETURN(0);
    % determine number of words contained in string %
        DIV (oldl + 4)/5, nw, cp;
    % compute the hash value %
        gen1 _ 0;
        gen2 _ 1;
        UNTIL nw = 0 DO
            BEGIN
            gen1 _ ast[nw] * gen2 + gen1 / 17;
            gen2 _ gen2 * 43;
            nw _ nw - 1;
```

```
        END;
      a1 _ gen1;
      !LSH a1,-6;
   % Return %
      RETURN(a1);
   END.
(astruc) PROCEDURE(astrng); %a-string to upper case%                    8L
   % convert a-string (in astrng) to upper case %
   %-----------------%
   LOCAL length, bytptr, char;
   REF astrng;
   IF (length _ astrng.L) = empty THEN RETURN(&astrng);
   bytptr _ chbptr(empty) + &astrng;
   DO IF (char _ |bytptr) IN ['a,'z] THEN
      .bytptr _ char - 40B
   UNTIL (length _ length - 1) = empty;
   RETURN(&astrng);
   END.

(slngth) PROCEDURE (bp1, bp2); %string length%                          8)
   %returns the lenght of the string  represented by the passed byte
   pointers, the first of which is assumed to be set so that an increment
   and load byte will get first char of the string and the second pointing
   to the last char in the string.  NOTE: the index field is ignored and
   the only the size field of bp1 is used!%
   %bpadr, bpsize, bpbitpos%
   %-----------------%
   LOCAL length;
   length _ (bp2.bpadr - bp1.bpadr)*(36 / bp1.bpsize) +
      (bp1.bpbitpos - bp2.bpbitpos) / bp1.bpsize;                      8X5/
   RETURN(MAX(length, 0));
   END.

(srmake) %make string from value%                                       8'
   PROCEDURE(value,astrng,base,format);
   LOCAL char, bpntr, b2pntr, b3pntr, mag, psign, ovrflw, cp, npad, nchars,
   i, sign, ncols, rcols, fill, decml;
   LOCAL STRING locstr[40];
   REF astrng;
   % make sure no floating point to confuse things %
      cp _ ovrflw _ npad _ 0;
   % initialize byte pointers %
      bpntr _ b2pntr _ $locstr + 1 .V 44070086;
   % extend sign of RH if requested %
      IF (format.fmsgne := FALSE) THEN !HRRES value;
   % setup for nout %
      mag _ IF (format.fmlgcl := FALSE) THEN 4B11 ELSE 0;
      psign _ IF (format.fmpsgn := FALSE) THEN 2B11 ELSE 0;
   % get number string (& do nothing if bad radix passed) %
      IF NOT SKIP !nout( bpntr, value, base .V mag .V psign) THEN RETURN
      ELSE nchars _ locstr.L _ slngth( bpntr, R1 );
   % pick up sign char (if any) and adjust pointer %
      CASE sign _ |b2pntr OF
         = '+, = '-:
            BEGIN
            BUMP DOWN nchars;
```

```
            bpntr _ b2pntr;
            END;
         ENDCASE sign _ FALSE;
(srovr):                                                        8Y11
% pick up other parameters & compute # needed columns %
   ncols _ locstr.L + (decml _ format.fmdecm);
   IF NOT (rcols _ format.fmncols) THEN rcols _ ncols;
   fill _ format.fmjstfy;
% compute number of padding characters required %
   npad _ rcols - ncols;
% check for overflow %
   IF npad < 0 THEN
      CASE format.fmovrf OF
         = 0:  RETURN;     % do nothing %
         = 1, = 5:         % MSDs only (& optionally a *) %
            BEGIN
            nchars _ nchars + npad;
            IF nchars <= 0 THEN REPEAT CASE( 4 );
            locstr.L _ locstr.L + (npad := 0);
            *locstr*[locstr.L + 1] _ 0;
            IF format.fmovrf = 5 THEN *locstr*[locstr.L] _ "*;
            END;
         = 2, = 6:         % (optionally a * &) LSDs only %
            BEGIN
            nchars _ nchars + npad;
            IF nchars <= 0 THEN REPEAT CASE( 4 );
            b3pntr _ b2pntr _ bpntr;
            FOR i _ -npad DOWN UNTIL = 0 DO char _ |b3pntr;
            FOR i _ (nchars + 1) DOWN UNTIL = 0 DO
               |b2pntr _ |b3pntr;
            IF format.fmovrf = 6 THEN |bpntr _ "*;
            locstr.L _ locstr.L + (npad := 0);
            END;
         = 3:      % spaces in entire field %
            BEGIN
            WHILE (rcols := rcols - 1) DO
               IF &astrng THEN *astrng* _ *astrng*, SP
               ELSE apachr( SP );
            RETURN;
            END;
         = 4:      % *s in entire field %
            BEGIN
            WHILE (rcols := rcols - 1) DO
               IF &astrng THEN *astrng* _ *astrng*, "*
               ELSE apachr( "* );
            RETURN;
            END;
         ENDCASE;
% leading spaces %
   IF fill AND (NOT format.fmfill) THEN srm3( &astrng, npad := 0, SP);
% sign if required %
   IF sign THEN
      BEGIN
      IF &astrng THEN *astrng* _ *astrng*, sign
      ELSE apachr( sign );
      cp _ 1;
```

```
        END;
    % leading zeros %
        IF fill AND format.fmfill THEN srm3( &astrng, npad := 0, '0);
    % significant digits %
        srm1( &astrng, $locstr, cp );
    % trailing zeros %
        IF format.fmfill THEN srm3( &astrng, npad := 0, '0);
    % terminating decimal %
        IF decml THEN
            IF &astrng THEN *astrng* _ *astrng*, '.
            ELSE apachr('.);
    % final spaces %
        srm3( &astrng, npad, SP);
    RETURN;
    END.

(srm3) PROCEDURE( astrng REF, npad, pchar);                                    8Z
    WHILE (npad := npad-1) > 0 DO
        IF &astrng THEN *astrng* _ *astrng*, pchar
        ELSE apachr( pchar );
    RETURN;
    END.

(srm1) PROCEDURE( astrng REF, str2 REF, cp);                                   8AA
    LOCAL TEXT POINTER tp1, tp2;
    IF &astrng THEN *astrng* _ *astrng*, *str2*[cp+1 TO str2.L]
    ELSE
        BEGIN
        tp1 _ tp2 _ &str2;
        tp1.stastr _ tp2.stastr _ TRUE;
        tp1[1] _ cp+1;
        tp2[1] _ str2.L + 1;
        aptstr( $tp1 );
        END;
    RETURN;
    END.

(srmk) PROCEDURE(value,base, format);                                          8AE
    %In string construction use apachr to append string to sar.%
    srmake(value, 0, base, format);
    RETURN;
    END.

(srval) PROCEDURE(astrng, base);                                               8A(
    %convert string to value%
    LOCAL value, cnt, char;
    REF astrng;
    value _ 0;
    cnt _ 1;
    UNTIL cnt > astrng.L DO
        BEGIN
        char _ *astrng*[cnt];
        char _ (CASE char OF
            IN [ '0, '9]: char - '0;
            IN ['A, 'Z]: char - 'A + 10;
            IN ['a, 'z]: char - 'a + 10;
```

```
        ENDCASE char - "0);
      value _ value*base + char;
      BUMP cnt;
      END;
   RETURN (value);
   END.

DECLARE reltab = (0,0,0, %no 0 relational%
   TRUE, FALSE, FALSE, %1: <%
   FALSE, TRUE, FALSE, %2: =%
   TRUE, TRUE, FALSE, %3: <=%
   0, 0, 0, %no 4 relational%
   FALSE, TRUE, TRUE, %5: >=%
   TRUE, FALSE, TRUE, %6: #%
   FALSE, FALSE, TRUE); %7: >%

(repstr) %The routine replaces the characters in one a-string, OLDAST, with
those in another, NEWAST, beginning at CHARNO. If the new a-string is
empty, the routine returns FALSE; otherwise it returns TRUE.%
                                                                    8AE
   PROCEDURE (newast, oldast, charno);
   %------------------%
   LOCAL astrl, repcnt;
   REF newast, oldast;
   IF (astrl _ newast.L) = empty THEN RETURN (FALSE);
   IF charno + newast.L > oldast.L THEN
      BEGIN
      IF charno + newast.L > oldast.M THEN
         RETURN (FALSE);
      oldast.L _ charno + newast.L;
      END;
   repcnt _ empty + 1;
   UNTIL repcnt > astrl DO
      BEGIN
      *oldast*[charno] _ *newast*[repcnt];
      BUMP charno, repcnt;
      END;
   RETURN (TRUE)
   END.

(srlset)  %To set an A-string to a single character, call this procedure
with the character and the address of the string.%
                                                                    8AF
   PROCEDURE (char, ast);
   [ast].L _ empty;
   apchr (char, ast);
   RETURN;
   END.

(mkbptr) %Make Byte Pointer with the specified position, size and address.
The address is 23 bits wide to allow indexing and indirection.%
                                                                    8AG
   PROCEDURE (pos, size, addr);
   a1 _ addr; !LSHC a1,-24;
   a1 _ size; !LSHC a1,-6;
   a1 _ pos;  !LSHC a1,-6;
```

```
    RETURN (a2);
    END.

% Statement scanning & cntent-analysis support%
  (pdc) PROCEDURE (pdcnum, pntloc);                                    9A
    %pointer decrement. arguments are number of postions to
    move and address of pointer to be decremented.  %

    LOCAL
      end;   %count at end of statement%
    IF scndir = forward THEN
      [pntloc+1] _ MAX([pntloc+1]-pdcnum, 1)
    ELSE
      BEGIN
      cpfse([pntloc]);
      end _ a2;
      [pntloc+1] _ MIN([pntloc+1]+pdcnum, end);
      END;
    RETURN;
    END.

  (qptr) PROCEDURE(pntloc);                                            9E
    pdc(1, pntloc);
    RETURN;
    END.

  (tstsr) PROCEDURE (ast);                                             9(
    %Test string.  Compare the T-string specified by SWORK with the string
    whose address is passed as arg.  Fetches characters from string by
    calling READC.  On a successful match does RETURN TRUE and SWORK is left
    pointing to the character after the pattern.  If the match fails, FALSE
    return.%

    LOCAL
      cnt,    %counter for characters in test string%
      char,   %character from source string%
      tsp;    %pointer into test string%
    REF ast;
    cnt _ ast.L;
    tsp _ chbmty + &ast;
    UNTIL (cnt _ cnt-1) < 0 DO
      IF (char _ READC) = ENDCHR OR !tsp # char THEN RETURN [FALSE];
    RETURN [TRUE];
    END.

  (tstf) PROCEDURE (ast);                                              9I
    %Like tst except looks for any occurence of the test string in the
    remaining portion of the T-string.  This is done by first scanning thru
    looking for the first character of the strirg, then if find it the rest
    of the string is?compared.  If get a mismatch SWORK is reset to
    character?after the start of the current partial match and the process
    is repeated.  Failure occurs only when the T-string is exhausted.
    Returns with RETURN TRUE on success, FALSE return on failure.%

    LOCAL
      cnt,    %counter for characters in test string%
```

```
      cpos,   %value of swork1 where got match%
      nchar,  %number of characters in test string%
      char,   %character from source string%
      char1,  %first character of test string%
      tsp,    %pointer into the test string%
      tsp1;   %pointer to first character of test string%
   REF ast;
   nchar _ ast.L;
   IF nchar < 1 THEN RETURN [TRUE];
   tsp1 _ chbmty + &ast;
   char1 _ |tsp1; %first character of string%
   LOOP %scan until get match for first character in string%
      CASE READC OF
         = ENDCHR : EXIT; %have reached the end%
         = char1 : %have a match on the first character%
            BEGIN
            cpos _ swork1;
            cnt _ 1;
            tsp _ tsp1;
            LOOP
               BEGIN
               IF (cnt _ cnt+1) > nchar THEN RETURN [TRUE];
               IF (char _ READC) = ENDCHR THEN EXIT 2;
               IF |tsp # char THEN EXIT;
               END;
            %failure -- back to scanning for first character%
            swork1 _ cpos;
            fechc1(scndir, $swork);
            END;
         ENDCASE;
   RETURN [FALSE];
   END.

(bfs) %branch false and scan %

   % Resets the character position, then reads another character from the
   statement.  If this is not an ENDCHR the character number is put on the
   stack and control is transfered to the location specified in the address
   of the pop.%

      % The above is such an ancient statement, it has been left as is.
      (The PDP-10 has UUO's but the SDS 940 has POPs -- WHP got carried
      away). Now, if the char is not an ENDCHR, the number is put on the
      stack and the procedure does a failure return. The compiler puts out
      a JUMPE R6, to the address in question (the beginning of the floating
      scan). %
   PROCEDURE;
   swork _ [p-1];
   swork1 _ [p];
   fechc1(scndir, $swork);
   IF READC # ENDCHR THEN
      BEGIN
      [p] _ swork1;
      RETURN[FALSE];
      END;
   p _ p - 20000028;
```

```
   RETURN
   END.

(fxswork) PROCEDURE;                                                    9F
   fechc1(scnptr, %swork);
   RETURN
   END.

(srsup) PROCEDURE; %cummy procedure for string support code%            9C
   %these are all called by JSP a4,x and returr by JRST (a4)%
   (fcp):                                                               9G2
      swork _ a1;
      swork1 _ a2;
      !JRST (a4);
   (pcp):                                                               9G3
      !PUSH p,swork;
      !PUSH p,swork1;
      !JRST (a4);
   (sptr):                                                              9G4
      !MOVE  a2, swork;
      !MOVEM a2,0(a1);
      !MOVE  a2, swork1;
      !MOVEM a2,1(a1);
      !JRST (a4);
   (begarb):                                                            9G5
      !PUSH p,swork;
      !PUSH p,swork1;
      !HLRZ a2,a1; %lower bound%
      !PUSH p,a2;
      !HRRZ a2,a1; %upper bound%
      !PUSH p,a2;
      a2 _ 0; %count%
      !PUSH p,a2;
      !JRST (a4);
   (fxsp1): %leave a1 and a2 unchanged%                                 9G6
      sptr1 _ a1;
      a3 _ 1;
      sptr1[a3] _ a2;
      !JRST (a4);
   (fxsp2):                                                             9G7
      sptr2 _ a1;
      a3 _ 1;
      sptr2[a3] _ a2;
      !JRST (a4);
   (lpr):                                                               9G8
      a1 _ [a2];
      a2 _ [a2+1];
      !JRST (a4);
   END.

EXTERNAL fcp, pcp, sptr, begarb, fxsp1, fxsp2, lpr;

(incarb) PROCEDURE;                                                     9
   %update position on stack where last succeeced%
   [p-4] _ swork;
   [p-3] _ swork1;
```

```
   %increment count and compare to upper bound%
   RETURN((Cp] _ Cp]+1) < Cp-1])
   END.

(erdarb) PROCEDURE;                                                    9L
   p _ p - 30000035;
   scp();
   p _ p - 20000025;
   RETURN(Cp+5] IN C[p+3],Cp+4]])
   END.

(scp) PROCEDURE;                                                       9K
   swork _ Cp-1];
   swork1 _ Cp];
   fechc1(scndir, $swork);
   RETURN
   END.

(scnlft) PROCEDURE;                                                    9L
   fechc1(scndir _ 0, $swork);
   RETURN
   END.

(scnrht) PROCEDURE;                                                    9M
   fechc1(scndir _ 1, $swork);
   RETURN
   END.

(chrct) PROCEDURE (char, chrcls); %Character class test%
                                                                       9N
   CASE chrcls OF
      =1: %CH%
         IF char IN C0,177B] THEN GOTO cctyes;
      =4: %LD%
         IF char IN C'A,'Z] OR
            char IN C'a,'z] OR
            char IN C'0,'9] THEN GOTO cctyes;
      =11: %NLD%
         IF char # ENDCHR AND
            char NOT IN C'A,'Z] AND
            char NOT IN C'a,'z] AND
            char NOT IN C'0,'9] THEN GOTO cctyes;
      =7: %L%
         IF char IN C'a,'z] OR
            char IN C'A,'Z] THEN GOTO cctyes;
      =8: %D%
         IF char IN C'0,'9] THEN GOTO cctyes;
      =9: %PT%
         IF char IN C41B,174B] THEN GOTO cctyes;
      =10: %NP%
         IF char # ENDCHR AND
            char NOT IN C41B,174B] THEN GOTO cctyes;
      =5: %UL%
         IF char IN C'A,'Z] THEN GOTO cctyes;
      =6: %LL%
         IF char IN C'a,'z] THEN GOTO cctyes;
```

```
      =2: %ULD%
         IF char IN ['A,'Z] OR
            char IN ['0,'9] THEN GOTO cctyes;
      =3: %LLD%
         IF char IN ['a,'z] OR
            char IN ['0,'9] THEN GOTO cctyes;
      ENDCASE;
   RETURN [FALSE] (char);
   (cctyes): RETURN [TRUE] (char)                                    9N3
   END.

(cct) PROCEDURE(chrcls);                                             9C
   chrct(READC, chrcls);
   RETURN(mreg)
   END.

(cctf) PROCEDURE(chrcls);                                            9F
   WHILE (a1 _ READC) # ENDCHR DO
      BEGIN
      chrct(a1, chrcls);
      IF mreg THEN RETURN(TRUE);
      END;
   RETURN (FALSE)
   END.

(span) PROCEDURE(chrcls);                                            9G
   LOCAL sw1;
   DO
      BEGIN
      sw1 _ swork1;
      chrct(READC, chrcls);
      END
      WHILE mreg;
   swork1 _ sw1;
   fxswork();
   RETURN
   END.

(tchf) PROCEDURE(char);                                              9I
   CASE READC OF
      =char: RETURN(TRUE);
      =ENDCHR: RETURN(FALSE);
      ENDCASE REPEAT CASE;
   END.

(cpfse) PROCEDURE (stid);                                            9S
   %statement end t-pointer%
   %called by SPL compiler only%

   IF stid.stastr THEN
      BEGIN
      a2 _ [stid.stadr].L + 1;
      a1 _ stid;
      RETURN;
      END;
   flcpfse( stid ):
```

```
    RETURN
    END.

% storage manipulation.  stacks, rings, buffers%
 DECLARE EXTERNAL FIELD
    systkp = [0(rp), 18:0], %pointer%
    systkt = [0(rp), 13:23], %type (1=stack, 2=ring)%
    systks = [1(rp), 18:18], %size of element%
    systke = [1(rp), 18:0]; %end of store%
    % rcpsh, rpop, rstsk, and <IDLIBE>gbotids use the fact that the stack
    body starts in word 2. %

. (rcpsh) %Record push.  Arguments are (1) address of record, and (2) address
 of stack or ring buffer.  The record is pushed on the store.%
                                                                        10
    PROCEDURE (recadr, st);
    REF st;
    IF st.systkt NOT IN [1,2] THEN
       sysrcv(programbug,$"bad address in stack/ring manipulation",
       sysclr());
    IF (st.systkp _ st.systkp + st.systks) = st.systke THEN
       IF st.systkt = 1 THEN ABORT(stkoverflow,$"stack overflow")
       ELSE st.systkp _ &st + 2; %ring wrap around%
    IF st.systks = 1 THEN [st.systkp] _ [recadr]
    ELSE mvbfbf (recadr, st.systkp, st.systks);
    RETURN
    END.

 (rpop) %Record pop.  Argument is the address of a store.  Pop the top
 element.%
                                                                        10
    PROCEDURE (st);
    REF st;
    IF st.systkt = 1 %stack% THEN
       IF st.systkp < &st+2 THEN ABORT(stkunderflow,$"stack underflow")
       ELSE NULL
    ELSE IF st.systkt = 2 %ring% THEN
       IF st.systkp <= &st+2 THEN
          st.systkp _ st.systke
       ELSE NULL
    ELSE sysrcv(programbug,$"bad address in stack/ring manipulation",
    sysclr());
    st.systkp _ st.systkp - st.systks;
    RETURN
    END.

 (rstsk) %Reset store%
                                                                        10
    PROCEDURE (st);
    REF st;
    IF st.systkt NOT IN [1,2] THEN
       sysrcv(programbug,$"bad address in stack/ring manipulation",
       sysclr());
    st.systkp _ &st + 2 - st.systks;
    RETURN
    END.
```

(rcpto) %Record Pop To. Arguments are (1) address of store, and (2) address of record. Pops the top record on the store to the designated location.%

```
    PROCEDURE (st, recadr);
    REF st;
    IF st.systkt NOT IN [1,2] THEN
       sysrcv(programbug,$"bad address in stack/ring manipulation",
       sysclr());
    IF st.systks = 1 THEN [recadr] _ [st.systkp]
    ELSE mvbfbf (st.systkp, recadr, st.systks);
    rpop (&st);
    RETURN
    END.
```

FINISH of Rt-Main

RE- Mini Data

(copyright) <compsrc, rt-minidata, 018> STRING 7A

```
(RtMiniData) FILE                                                              1
% mini data package for L10 mini runtime package %
% does NOT support signals, catchphrases, string manipulation, stacks %
DECLARE EXTERNAL CONSTANT
   gstksz = 100;
DECLARE EXTERNAL
   gstack[gstksz];
DECLARE EXTERNAL
   startup, % REF for startup procedure %
   recover, % " for recovery %
   syswhy, % runtime error reason %
   sysloc, % location of error %
   sysrip, % recover in procress %
   sysetc; % error type code %
% copyright string %
   (copyright) STRING = "COPYRIGHT by Tymshare Incorporated 1978";             7A
FINISH of RtMiniData
```

RE-Mini/Main

(bptrcv)	<compsrc, rt-minimain, 023>	LOCAL	7C2
(l10set)	<compsrc, rt-minimain, 044>	LOCAL	7C6A
(l10start)	<compsrc, rt-minimain, 039>	LOCAL	7C5A
(pcall)	<compsrc, rt-minimain, 089>	LOCAL	7C8D
(status)	<compsrc, rt-minimain, 0113>	LOCAL	7D1A
(syscent)	<compsrc, rt-minimain, 084>	LOCAL	7C8C
(syscer)	<compsrc, rt-minimain, 0102>	LOCAL	7C9B
(sysent)	<compsrc, rt-minimain, 075>	LOCAL	7C8A
(sysnxp)	<compsrc, rt-minimain, 0100>	LOCAL	7C9A
(sysovr)	<compsrc, rt-minimain, 081>	LOCAL	7C8B
(sysrcv)	<compsrc, rt-minimain, 018>	LOCAL	7C
(sysrtf)	<compsrc, rt-minimain, 068>	LOCAL	7C7C
(sysrtn)	<compsrc, rt-minimain, 063>	LOCAL	7C7B
(sysrtt)	<compsrc, rt-minimain, 061>	LOCAL	7C7A
(syssys)	<compsrc, rt-minimain, 0111>	LOCAL	7D
(sysufl)	<compsrc, rt-minimain, 0108>	LOCAL	7C9C

```
FILE l10runtime % (arcsubsys,xl10,) to (l10,minil10runtime,) %
% mini L10 runtime package %
% does NOT support signals, catchphrases, strings, stacks %
ALLOW!
% SEE (nls,xl10runtime,) for notes about use %
% DECLARATIONS %
   REGISTER mreg=6;
   EXTERNAL l10start;
   DECLARE EXTERNAL chbmty = 440700000001B;
% system code %
   EXTERNAL
      bptrcv,
      sysrtn, sysrtt, sysrtf, syster,
      sysnxp, syscer, syscte, sysufl,
      sysovr, sysent, syscent, pcall;
   REF startup, recover;
   (sysrcv) % recover/startup code here %                                    7C
      PROCEDURE % called for fatal error %
         (type, % error type code %
         why, % error code %
         loc); % associated location %
      (bptrcv): % see arguments %                                           7C2
      % check for recovery loop %
         IF NOT sysrip THEN % recovery in progress ? %
            BEGIN % first try - issure NOTE %
            sysrip _ TRUE;
            sysetc _ type; % error-type code %
            syswhy _ why;
            IF loc.RH IN [$gstack,$gstack+gstksz] THEN % a port %
               loc _ [ loc.RH+1 ].RH; % get its last exit point %
            sysloc _ loc;
            END;
      % store error parameters and start over %
         !JSP R1,l10set; % reset stacks %
         syssys(TRUE); % recover %
      % initial startup here %
          (l10start):                                                       7C5
         !JSP R1,l10set; % reset stacks %
         sysrip _ FALSE;
         syssys(FALSE); % recover %
      % initialize stack pointers%
         (l10set): % local label %                                          7C6
         %general call stack%
            S _ -gstksz; !HRL S,S; !HRRI S,gstack;
            M _ S;
            gstack _ $sysufl;
         % return %
            !JRST 0(R1);
      % returns, get here by JRST produced by compiler %
         (sysrtt): % return true %                                          7C7
            !MOVEI mreg,1;
         (sysrtn): % return, value in mreg %                                7C7
            !MOVE S,M;
            !POP S,M;
            !POPJ S,0;
         (sysrtf): % return false %                                         7C7
```

```
        !MOVEI mreg,0;
        !MOVE S,M;
        !POP S,M;
        !POPJ S,0;
    % procedure entry, pcall. get here by JSP A4 produced by compier %
      (sysent): % procedure entry point (trace point) %                    7C8A
        !PUSH S,M; % push old mark %
        !MOVE M,S; % set up new one %
        !AOBJN S,0(A4); % return to procedure %
          % AOB puts blank on stack (coroutine return) %
          % also check for overflow (positive) %
      (syscvr): % stack overflow (follows sysent) %                        7C8B
        S _ -$gstksz; !HRL S,S; !HRRI S,gstack;
        sysrcv(stkoverflow,$"general stack overflow",0);
      (syscent): % port entry (trace point) %                              7C8C
        !PUSH S,M; % push old mark %
        !MOVE M,S; % get new one %
        !PUSH S,A4; % setup co-return loc for ddt %
        !JRST 0(A4); % return to port entry %
      (pcall): % port-call code (trace point) %                            7C8D
        % mreg contains port id to call %
        % A4 contains return address %
        !EXCH mreg,M; % setup new frame, old is argument %
        !CAML M,S; % check to see if in stack %
        !JRST sysnxp; % no! badness! %
        !MOVEM A4,1(mreg); % save return in old frame %
        !JRST @1(M); % call new one %
        % this is 4 instr shorter than inline, costs 1 instr xeq %
        % primarily here to allow trace %
    % fatal error points, JSP,A4 produced by compiler %
      (sysnxp): % non-existant port called %                               7C9A
        sysrcv(programbug,$"non-existant port called",A4);
      (syscer): % coroutine called as procedure %                          7C9E
        sysrcv(programbug,$"coroutine called as procedure",A4);
      (sysufl): % stack underflow (strange) %                              7C9C
        sysrcv(programbug,$"general stack underflow",0);
    END.
  (syssys) % initialize system coroutines, etc %                             7C
    PROCEDURE % this must be done from a procedure %
      (status); % TRUE if recovering %                                     7D1A
    % give control to program %
      sysrip _ FALSE;
      IF status THEN recover(sysetc,syswhy,sysloc)
      ELSE startup();
      sysrcv(programbug,$"startup/recover procedure returned",0); % don't
      return %
    END.
FINISH
```

StgMgE

```
FILE stgmgt   % (arcsubsys,xl10,) to (l10, stgmgt,) (arcsubsys,l1011,) to
(l1011, stgmgt,)  %
(blkhdr) RECORD %  block header record%                                    2
   blklength[15],   %length of block (in words) requested%
   blkfree[1],      %true if block is on a free list%
   sysblength[15],  %actual length (in address units) of block%
   blkprevfree[1]; %true if previous block is on a free list%
%free list pointers% %INDEXES%
   (blksuc) CONSTANT = blkhdr.SIZE;  %addr of successor block%             3A
   (blkprd) CONSTANT = blkhdr.SIZE + 1;  %addr of predecessor block%       3B
%zone header% %INDEXES%
   (wrdsused) EXTERNAL CONSTANT = 0;                                       4A
   (blkmax) EXTERNAL CONSTANT = 1;                                         4B
   (blkmin) EXTERNAL CONSTANT = 2;                                         4C
   (blkmxi) EXTERNAL CONSTANT = 3;                                         4D
   (blksused) EXTERNAL CONSTANT = 4;                                       4E
   (zonend) EXTERNAL CONSTANT = 5;                                         4F
   (zfrelst) EXTERNAL CONSTANT = 6;                                        4G

% error strings %
   %+CPU10% % for 10, use actual strings %                                 5A
      (smezos) %400% = $"Storage Management Error: Zone out of space";     5A1
      (smefib) %401% = $"Storage Management Error: Attempt to free invalid
      block";                                                              5A2
      (smebip) %402% = $"Storage Management Error: bad index passed to
      REPLENISH";                                                          5A3
      %+CPU10%                                                             5A4
   %+CPU11% % for 11, use error numbers %                                  5B
      (smen) CONSTANT = 21400B;                                            5B1
      (smezos) CONSTANT = smen+0;                                          5B2
      (smefib) CONSTANT = smen+1;                                          5B3
      (smebip) CONSTANT = smen+2;                                          5B4
      %+CPU11%                                                             5B5
%storage allocator%
   (getblk) PROCEDURE  %allocate a blk of "size" words in "zone"%          6A
      (size,  %number of words desired in blk%
      zone REF);  %address of free storage zone%
      %------------------%
      %allocates a blk in zone of at least size+blkhdr.SIZE words.  The blk
      consists of a blk header and at least size words of zeros.  The address
      returned is that of the begining of the blk (the blk header) plus
      blkhdr.SIZE.  There are at least size usable words at this address.  If
      no blk can be allocated, a FALSE return of zero is made.%
      %------------------%
      LOCAL
         blk REF,    %address of blk%
         r,       %remainder temp%
         zero,    %temp%
         end,     %temp%
         index;  %index into zones free lists for blocks of
                  correct size%
      size _ size*ADR-PER-WORD;
         %size now in address units%
      IF zone[blkmin] = zone[blkmax] THEN
         IF zone[blkmax] >= size+blkhdr.SIZE *ADR-PER-WORD THEN
            index _ 0
```

```
      ELSE
         %RETURN[FALSE](0)% err(smezos)
   ELSE
      BEGIN  %round up to nearest available size%
      index _ (size + blkhdr.SIZE * ADR-PER-WORD + zone[blkmin] -
      1)/zone[blkmin] - 1;
         %index now in words%
      IF index NOT IN [0, zone[blkmxi]] THEN %RETURN[FALSE](0)%
      err(smezos);
      END;
   IF NOT zone[zfrelst + index] AND (zone[blkmin] = zone[blkmax] OR NOT
   replenish(index, &zone)) THEN
      %no free blocks of that index or larger%
      %RETURN[FALSE](0)% err(smezos);
   &blk _ zone[zfrelst + index];
   %unlink it%
      unlink(&blk, &zone);
   %zero the blk%
      zero _ &blk + blkhdr.SIZE * ADR-PER-WORD;
      end _ &blk + blk.sysblength;
      DO [zero] _ 0 UNTIL (zero _ zero + ADR-PER-WORD) >= end;
   blk.blklength _ size/ADR-PER-WORD + blkhdr.SIZE; %for caller's use only%
      %in WORD units%
   BUMP zone[blksused];
   zone[wrdsused] _ zone[wrdsused] + blk.sysblength;
      %actually address-units used%
   RETURN[TRUE](&blk + blkhdr.SIZE * ADR-PER-WORD);
   END.

(freeblk) PROCEDURE  %de-allocate a block in zone%                    6E
   (blk REF,  %address of block to be de-allocated%
   zone REF); %address of free storage zone%
   %----------------%
   %the block addressed by blk is marked as free.  It is not linked back
   into the appropriate free list in zone (as was done in the past).
   NOTE:  Freeblk will return FALSE if it finds a bad block address.  You
   may run out of space if you never issue another good free to replace a
   bad call.%
   %----------------%
   LOCAL
      previous REF, %address of previous block in the zone%
                %for combining free blocks%
      next REF;     %address of next block in zone%
                %for combining free blocks%
   IF (&blk _ &blk - blkhdr.SIZE * ADR-PER-WORD) NOT IN [&zone,
   zone[zonend]] THEN
      RETURN[FALSE](FALSE);
   IF blk.sysblength NOT IN [zone[blkmin], zone[blkmax]] OR blk.blkfree
   THEN
      err(smefib);
   zone[wrdsused] _ zone[wrdsused] - blk.sysblength;
   %  this is the old code to combine adjacent free blocks
   IF zone[blkmin] NOT= zone[blkmax] THEN
      BEGIN
      &next _ &blk + blk.sysblength;
      IF &next < zone[zonend] THEN
```

```
        BEGIN
        IF next.blkfree AND (blk.sysblength + next.sysblength <=
        zone[blkmax]) THEN
           BEGIN %%combine them%%
           unlink(&next, &zone);
           blk.sysblength _ blk.sysblength + rext.sysblength;
           next.sysblength _ 0;
           &next _ &blk + blk.sysblength;
           END;
        END;
     IF blk.blkprevfree AND (blk.sysblength + [&blk - ADR-PER-WORD] <=
     zone[blkmax]) THEN
        BEGIN %%combine them%%
        %%last word of previous block contains the length of that block%%
        &previous _ &blk - [&blk - ADR-PER-WORD];
        unlink(&previous, &zone);
        previous.sysblength _ previous.sysblength + blk.sysblength;
        blk.sysblength _ 0;
        &blk _ &previous;
        END;
     END;
  %
  linkup(&blk, &zone);
  BUMP DOWN zone[blksused];
  RETURN[TRUE](TRUE);
  END.

(makezone) PROCEDURE  %make a zone out of a block%                          6
  (zone REF,  %address of a block to be initialized as a zone%
  max,    %maximum size in words of a block -- integral multiple of min%
  min,    %minimum size in words of a block%
  size);  %size in words of the zone%
  %----------------%
  %This routine takes the address of a free storage block and constructs
  an appropriate header so that it will serve as a free storage zone.  If
  min and max are the same, then the zone will contain only fixed sized
  block and a single free list -- no combining of adjacent free blocks
  will be attempted by freeblk%
  %----------------%
  LOCAL
     blk REF,     %address of a block within the zone%
     lastfreelist,    %displacement to the lastfreelist free list for this
     zone%
     frstblk REF, %address of first block in the zone%
     base REF,    %address of a block within the zone%
     length;  %length of rest of block%
  % zero out this whole block before starting %
     FOR length _ (size-1) DOWN UNTIL < 0 DO
        IF zone[length] THEN zone[length] _ 0;
  size _ size*ADR-PER-WORD;
  min _ MAX(min, blkhdr.SIZE+3);
  max _ ((max+min-1)/min)*min;
  zone[blkmxi] _ (max/min) - 1;
  max _ max*ADR-PER-WORD; %in address units%
  min _ min*ADR-PER-WORD; %in address units%
  zone[blkmax] _ max;
```

```
    zone[blkmin] _ min;
    lastfreelist _ zone[blkmxi] + zfrelst; %index to last free list%
    &frstblk _ &base _ &zone + lastfreelist*ADR-PER-WORD + ADR-PER-WORD;
    length _ ((size - (&frstblk - &zone))/min)*min;
    zone[zonend] _ &frstblk + length;
    UNTIL length < max DO
       BEGIN
       &blk _ &base + max;
       base.sysblength _ max;
       [&blk-ADR-PER-WORD] _ max;
       base.blkfree _ base.blkprevfree _ TRUE;
       base[blksuc] _ zone[lastfreelist] := &base;
       base[blkprd] _ 0;
       IF &base _ base[blksuc] THEN
          base[blkprd] _ zone[lastfreelist];
       &base _ &blk;
       length _ length - max;
       END;
    frstblk.blkprevfree _ FALSE;
    IF length > 0 THEN
       BEGIN
       blk.sysblength _ length;
       IF length < min THEN
          blk.blkfree _ blk.blkprevfree _ FALSE
       ELSE
          BEGIN
          blk.blkfree _ blk.blkprevfree _ TRUE;
          blk[blksuc] _ zone[zfrelst + length/min - 1] := &blk;
          blk[blkprd] _ 0;
          IF &base _ blk[blksuc] THEN
             base[blkprd] _ &blk;
          [&blk + blk.sysblength -ADR-PER-WORD] _ length;
          END;
       END;
    zone[blksused] _ zone[wrdsused] _ 0;
    RETURN;
    END.

(replenish) PROCEDURE %replenish supply of free blocks in zone%          6[
    (index, %free list index for size blocks needed%
    zone REF);  %address of free storage zone%
    %--------------%
    %This routine will try to replenish the supply of blocks corresponding
    to freelist specified by index.  First it will try to find a larger
    block which can be subdivided to replenish the supply of blcks desired.
    It will try to find a block which is a multiple in size (so all pieces
    will go on same free list).  If this fails, a scan for any larger block
    is made.
    If no larger blocks are found, it will attempt to combine samller blocks
    into a block of the size desired.  It does this by searching freelists
    for a free block and then attempting to combine adjacent blocks as
    required.  If this is unsuccessful, it returns FALSE; otherwise, breakup
    is used to decompose the larger block and acd to the supply of desired
    blocks and this routine returns TRUE%
    %--------------%
    LOCAL
```

```
    indexplus1, %value of index+1 (used to speed up loop)%
    blk REF,   %address of a larger free block%
    newblk REF, %addr of blk returned when combining smaller blks %
    size, %words needed %
    maxi,  %value of zone[blkmxi] (used to speed up loop)%
    n,     %temp, used in loop%
    i;     %temp, used as loop variable ad free list index%
IF index NOT IN [0, zone[blkmxi]] THEN
    err(smebip);
maxi _ zone[blkmxi];
%try for multiples first%
    i _ indexplus1 _ index+1;
    n _ 2;
    UNTIL (i _ indexplus1*n - 1) > maxi DO
        BEGIN
        IF zone[i + zfrelst] THEN %got one%
            BEGIN
            unlink(&blk _ zone[i+zfrelst], &zone);
            breakup(&blk, &zone, index);
            RETURN(TRUE);
            END;
        BUMP n;
        END;
%Find any%
    i _ index;
    UNTIL (i _ i + 1) > maxi DO
        IF zone[zfrelst + i] THEN
            BEGIN %got one%
            &blk _ zone[zfrelst+i];
            unlink(&blk, &zone);
            breakup(&blk, &zone, index);
            RETURN(TRUE);
            END;
%Try to combine smaller adjacent blocks. Start with next smallest size
(since anything plus this will lead to the size needed)%
    size _ (index + 1) * zone[blkmin]; % words needed %
    FOR i _ (index - 1) DOWN UNTIL < 0 DO
        BEGIN
        IF &blk _ zone[zfrelst + i] THEN
            BEGIN %found free list%
            WHILE  &blk DO
                BEGIN
                % try this block and adjacent blocks up %
                    IF (&newblk _ combup(&blk, size - blk.sysblength, &blk,
                    &zone)) THEN
                        BEGIN % must replenish correct freelist %
                        breakup(&newblk, &zone, index);
                        RETURN(TRUE);
                        END;
                % try this block and adjacent blocks down %
                    IF (combdown(&blk, size - blk.sysblength, &blk, &zone))
                    THEN
                        BEGIN
                        breakup(&blk, &zone, index);
                        RETURN(TRUE);
                        END;
```

```
                &blk _ blk[blksuc];
                END;
            END;
        END;
    RETURN(FALSE);
    END.

(breakup) PROCEDURE  %break up blk and put in index free list%                61
    (blk REF,     %address of block to be subdivided%
    zone REF,     %address of free storage zone%
    index);  %free list index to be replenished%
    %-----------------%
    %This routine creates a number of blocks (by breaking up blk) of
    appropriate size for the free list indicated by index and links them
    into that free list in zone.%
    %-----------------%
    LOCAL
        length, %length of smaller blocks%
        base REF,    %temp, used in subdividing blk%
        i,       %loop control variable%
        r;       %temp, used to set up i%
    length _ (index+1)*zone[blkmin];
    DIV blk.sysblength/length, i, r;
    IF NOT r THEN BUMP DOWN i;
    UNTIL (i _ i -1) < 0 DO
        BEGIN
        &blk _ (&base _ &blk) + length;
        blk.sysblength _ (base.sysblength := length) - length;
        linkup(&base, &zone);
        END;
    &blk _ (&base _ &blk) + blk.sysblength;
    IF base.sysblength >= zone[blkmin] THEN
        linkup(&base, &zone)
    ELSE base.blkfree _ FALSE;
    RETURN;
    END.

(unlink) PROCEDURE  %unlink a block from its free list%                       61
    (blk REF,  %address of block to be unlinked from its free list%
    zone REF); %address of a free storage zone%
    %---------------%
    %this routine unlinks blk from the appropriate free list in zone%
    %---------------%
    LOCAL
        next REF, %address of next block in zone%
        suc REF,  %address of block pointed to by successo link%
        prd REF:  %address of block pointed to by predecessor link%
    blk.blkfree _ FALSE;
    IF (&next _ &blk + blk.sysblength) < zone[zonend] THEN
        next.blkprevfree _ FALSE;
    IF &suc _ blk[blksuc] THEN
        suc[blkprd] _ blk[blkprd];
    IF &prd _ blk[blkprd] THEN
        prd[blksuc] _ blk[blksuc] := 0
    ELSE
        zone[zfrelst + blk.sysblength/zone[blkmin] - 1] _ blk[blksuc] := 0;
```

```
    blk[blkord] _ 0;
    RETURN;
    END.

(linkup) PROCEDURE  %linkup blk into appropriate free list%                    6C
    (blk REF,  %address of block to be linked into free list%
    zone REF); %address of a free storage zone%
    %--------------%
    %this routine links blk into the begining of the appropriate free list
    in zone%
    %--------------%
    LOCAL
       next REF,   %address of next block in zone%
       index;  %free list index for blk%
    blk.blkfree _ TRUE;
    index _ blk.sysblength/zone[blkmin]-1 + zfrelst;
    blk[blksuc] _ zone[index] := &blk;
    blk[blkord] _ 0;
    [&blk+blk.sysblength-ADR-PER-WORD] _ blk.sysblength;
    IF (&next _ &blk+blk.sysblength) < zone[zonend] THEN
       next.blkprevfree _ TRUE;
    IF &blk _ blk[blksuc] THEN
       blk[blkord] _ zone[index];
    RETURN;
    END.

(combup)   % CL: :  combine up %
PROCEDURE (blk REF, wrdsneeded, strtblk REF, zone REF);                        6F
    % Procedure description
       FUNCTION
          blk--REF- addr of next block to consider.
          wrdsneeded--words needed to fulfill this request.
          strtblk--REF- addr of first block considered.
       ARGUMENTS
          none
       RESULTS
          proc-value
       NON-STANDARD CONTROL
          none
       GLOBALS
          none
       %
    % Declarations %
       LOCAL
          complength _ 0, nlength,
          next REF,
          newblk REF;
    % combine up %
       IF blk.blkprevfree = 0 THEN RETURN(FALSE);
       &newblk _ &blk - [&blk - ADR-PER-WORD];
       IF newblk.sysblength >= wrdsneeded THEN
          BEGIN % combine all blocks down to strtblk %
          &next _ &newblk;
          UNTIL &next > &strtblk DO
             BEGIN
             unlink(&next, &zone);
```

```
            nlength _ next.sysblength := 0;
            complength _ complength + nlength;
            $next _ &next + nlength;
            END;
         newblk.sysblength _ complength;
         RETURN(&newblk);
         END
      ELSE RETURN(combup(&newblk, wrdsneeded-newblk.sysblength, &strtblk,
      &zone));
   % Return %
      RETURN;
   END.

(combdown)   % CL: ;  combine down %
PROCEDURE (blk REF, wrdsneeded, strtblk REF, zone REF);                    61
   % Procedure description
      FUNCTION
         none
      ARGUMENTS
         blk--REF- addr of newblk block to consider.
         wrdsneeded--words needed to fulfill this request.
         strtblk--REF- addr of first block considered.
      RESULTS
         proc-value
      NON-STANDARD CONTROL
         none
      GLOBALS
         none
      %
   % Declarations %
      LOCAL
         nsize, csize, prevblk REF,
         newblk REF;
   % combine down %
      csize _ blk.sysblength;
      &newblk _ &blk + csize;
      nsize _ newblk.sysblength;
      IF newblk.blkfree AND (nsize + csize <= &zone) THEN
         BEGIN % newblk block free %
         IF nsize >= wrdsneeded THEN
            BEGIN % unlink the blocks we have checked %
            unlink(&strtblk, &zone);
            UNTIL &newblk <= &strtblk DO
               BEGIN
               unlink(&newblk, &zone);
               strtblk.sysblength _ strtblk.sysblength + newblk.sysblength;
               newblk.sysblength _ 0;
               &newblk _ &newblk - [&newblk - ADR-PER-WORD];
               END;
            RETURN(TRUE); % the block is strtblk %
            END
         ELSE % keep going down %
            RETURN(combdown(&newblk, wrdsneedec-newblk.sysblength,
            &strtblk, &zone));
         END;
   % Return %
```

```
        RETURN(FALSE);
      END.

%support routines to control storage allocator%
   %string handling%
      (getstring) PROCEDURE (length, zone);                                    7A1
         %allocate a block in "zone" of  "length" characters  plus  block
         header plus string header.  Initialize the string to empty.  Return
         string address.
         length        actual useable length of string desired
         zone          address of zone with free space in it
         SIGNALS if no more room in that block%
         %---------------%
         LOCAL blk REF;
         &blk _ getblk ((length + WORD/CHAR -1)/(WORD/CHAR)  + %string header%
         (2*ADDRESS)/WORD, zone);
         IF NOT &blk THEN
            err( smezos);
         IF CPU = 11 THEN &blk _ &blk + 4;
         blk.M _ length;
         *blk* _ NULL;
         RETURN (&blk);
         END.

      (freestring) PROCEDURE (string, zone);  %free this allocated string%  7A1
         LOCAL outcome;
         IF CPU = 11 THEN string _ string - 4;
         RETURN[outcome](freeblk(string, zone : [outcome]));
         END.

FINISH
```