C Gram

(gramend)	<compsrc, cgram, 068>	LOCAL	1A46
(grammax)	<compsrc, cgram, 08>	CONSTANT =crmsize / 2	1A2C
(gramstuf)	<compsrc, cgram, 0399>	PROCEDURE	188
(gramtop)	<compsrc, cgram, 066>	LOCAL	1A4E
(crmsize)	<compsrc, cgram, 010>	CONSTANT =300008	1A29
(nvarstart)	<compsrc, cgram, 0109>	FIELD - address	1A7A10
(nvtab)	<compsrc, cgram, 089>	LOCAL	1A5F
(nvtabl)	<compsrc, cgram, 0826>	LOCAL	1A5N
(helprule)	<compsrc, cgram, 099>	FIELD - address	1A7A2C
(hlprgrule)	<compsrc, cgram, 01264>	FIELD - address	1A7A2G
(indexof)	<compsrc, cgram, 01263>	CONSTANT =658	1A2J54
(initinst)	<compsrc, cgram, 0100>	FIELD - address	1A7A2D
(inlste)	<compsrc, cgram, 01254>	CONSTANT =628	1A2J51
(inst)	<compsrc, cgram, 071>	LOCAL	1A4J
(instlen)	<compsrc, cgram, 072>	LOCAL	1A4K
(instrec)	<compsrc, cgram, 0120>	RECORD	1A7C
(instvar)	<compsrc, cgram, 0613>	PROCEDURE	1B11
(ivarr)	<compsrc, cgram, 0139>	RECORD	1A8C
(keyop)	<compsrc, cgram, 021>	CONSTANT =1	1A2J2
(keyop1)	<compsrc, cgram, 01019>	CONSTANT =358	1A2J30
(kword)	<compsrc, cgram, 0104>	FIELD - address	1A7A4
(kwtab)	<compsrc, cgram, 084>	LOCAL	1A5D
(kwtabl)	<compsrc, cgram, 0830>	LOCAL	1A5E
(lastfield)	<compsrc, cgram, 054>	CONSTANT =2	1A2L3
(lastnext)	<compsrc, cgram, 055>	CONSTANT =3	1A2L4
(lastnone)	<compsrc, cgram, 053>	CONSTANT =1	1A2L2
(lclste)	<compsrc, cgram, 01253>	CONSTANT =608	1A2J49
(load)	<compsrc, cgram, 036>	CONSTANT =208	1A2J17
(loadi)	<compsrc, cgram, 01069>	CONSTANT =438	1A2J36
(local)	<compsrc, cgram, 050>	CONSTANT =0	1A2K1
(lone)	<compsrc, cgram, 0836>	FIELD - 1	1A8B2
(loop)	<compsrc, cgram, 01077>	CONSTANT =518	1A2J42
(lsel)	<compsrc, cgram, 025>	CONSTANT =5	1A2J6
(lvtab)	<compsrc, cgram, 088>	LOCAL	1A5K
(lvtabl)	<compsrc, cgram, 0827>	LOCAL	1A5L
(mailbox)	<compsrc, cgram, 0116>	FIELD - subfields	1A7B3
(makrad50)	<compsrc, cgram, 01092>	PROCEDURE	1B17
(mapper)	<compsrc, cgram, 0337>	PROCEDURE	1B4
(maxalts)	<compsrc, cgram, 012>	CONSTANT =64	1A2F
(maxinst)	<compsrc, cgram, 011>	CONSTANT =crmsize / 8	1A2E
(mvbfbf)	<compsrc, cgram, 01274>	LOCAL	1B23
(nextab)	<compsrc, cgram, 087>	LOCAL	1A5J
(nmtab)	<compsrc, cgram, 01015>	LOCAL	1A5G
(nmtabl)	<compsrc, cgram, 01016>	LOCAL	1A5R
(noitems)	<compsrc, cgram, 014>	CONSTANT =-1	1A2I
(notlast)	<compsrc, cgram, 052>	CONSTANT =0	1A2L1
(nuadd)	<compsrc, cgram, 073>	LOCAL	1A4L
(nuadd2)	<compsrc, cgram, 075>	LOCAL	1A4N
(nuinstrec)	<compsrc, cgram, 0137>	RECORD	1A8A
(nums)	<compsrc, cgram, 0111>	FIELD - address	1A7A13
(nuopcode)	<compsrc, cgram, 0835>	FIELD - 6	1A8A1
(nuval)	<compsrc, cgram, 074>	LOCAL	1A4M
(ocall)	<compsrc, cgram, 01040>	CONSTANT =418	1A2J34
(ocllrule)	<compsrc, cgram, 01041>	CONSTANT =428	1A2J35
(oldcram)	<compsrc, cgram, 061>	LOCAL	1A3C
(opcod)	<compsrc, cgram, 0133>	FIELD - 6	1A7C6

```
FILE cgram  %(arcsubsys,l109,) (fe,cgram,) % %(arcsubsys,xl10,) (fe,cgram,) %
   %Declarations%
      ALLOW!
      % Constants %
         (pagelen) CONSTANT = 512; %words in pdp10 page%                 1A2
         (grmsize) CONSTANT = 30000B; % maximum size of uncompacted gram%   1A2
         (grammax) CONSTANT = grmsize / 2; % maximum size of compacted gram
         (sharable)%                                                      1A2
         (varsmax) CONSTANT = grammax / 10B; % maximum size of compacted gram
         (private)%                                                       1A2
         (maxinst) CONSTANT = grmsize / 2; % maximum number of insts in gram %
                                                                          1A2
         (maxalts) CONSTANT = 64; %max number of alts at a node in gram tree%
                                                                          1A2
         (opmax) CONSTANT = 64; % number of different opcodes %           1A2
         (bytemag) CONSTANT = 256; %magnitude of 8-bit byte%              1A2
         (noitems) CONSTANT = -1; %empty table indicator%                 1A2
         % CML opcodes %
            (abortop) CONSTANT =0;                                        1A2
            (keyop) CONSTANT =1;                                          1A2
            (confirm) CONSTANT =2;                                        1A2
            (ssel) CONSTANT =3;                                           1A2
            (dsel) CONSTANT =4;                                           1A2
            (lsel) CONSTANT =5;                                           1A2
            (pusharg) CONSTANT =6;                                        1A2
            (answer) CONSTANT =7;                                         1A2
            (option) CONSTANT =10B;                                       1A2
            (pfcllop) CONSTANT =11B;                                      1A2J1
            (execute) CONSTANT =12B;                                      1A2J1
            (call) CONSTANT =13B;                                         1A2J1
            (fbclear) CONSTANT =14B;                                      1A2J1
            (echo) CONSTANT =15B;                                         1A2J1
            (recho) CONSTANT =16B;                                        1A2J1
            (storeop) CONSTANT =17B;                                      1A2J1
            (load) CONSTANT =20B;                                         1A2J1
            (enter) CONSTANT =21B;                                        1A2J1
            (test) CONSTANT =22B;                                         1A2J1
            (showop) CONSTANT = 23B;                                      1A2J2
            (entercw) CONSTANT = 24B;                                     1A2J2
            (enternull) CONSTANT = 25B;                                   1A2J2
            (entertrue) CONSTANT = 26B;                                   1A2J2
            (enterfalse) CONSTANT = 27B;                                  1A2J2
            (resume) CONSTANT = 30B;                                      1A2J2
            (append) CONSTANT = 31B;                                      1A2J2
            (testtrue) CONSTANT = 32B;                                    1A2J2
            (testnull) CONSTANT = 33B;                                    1A2J2
            (callhelp) CONSTANT = 34B;                                    1A2J2
            (keyop1) CONSTANT = 35B;                                      1A2J3
            (vkeyop) CONSTANT = 36B;                                      1A2J3
            (pushvar) CONSTANT = 37B;                                     1A2J3
            (rptop) CONSTANT = 40B;                                       1A2J3
            (ocall) CONSTANT = 41B;                                       1A2J3
            (ocllrule) CONSTANT = 42B;                                    1A2J3
            (loadi) CONSTANT = 43B;                                       1A2J3
            (showcon) CONSTANT = 44B;                                     1A2J3
            (showstat) CONSTANT = 45B;                                    1A2J3
```

```
      (resumnull) CONSTANT = 46B;                                    1A2J3
      (testntrue) CONSTANT = 47B;                                    1A2J4
      (testnnull) CONSTANT = 50B;                                    1A2J4
      (loop) CONSTANT = 51B;                                         1A2J4
      (retloop) CONSTANT = 52B;                                      1A2J4
      (exitloop) CONSTANT = 53B;                                     1A2J4
      (untilloop) CONSTANT = 54B;                                    1A2J4
      (whileloop) CONSTANT = 55B;                                    1A2J4
      (ifcllop) CONSTANT = 56B;                                      1A2J4
      (clearstat) CONSTANT = 57B;                                    1A2J4
      (lclste) CONSTANT = 60B;                                       1A2J4
      (stlste) CONSTANT = 61B;                                       1A2J5
      (inlste) CONSTANT = 62B;                                       1A2J5
      (dllste) CONSTANT = 63B;                                       1A2J5
      (enterstr) CONSTANT = 64B;                                     1A2J5
      (indexof) CONSTANT = 65B;                                      1A2J5
   %inst var types%
      (local) CONSTANT =0;                                           1A2K
      (global) CONSTANT =2;                                          1A2K
      (bltin) CONSTANT =3;                                           1A2K
   % Altsuc field values%
      (notlast) CONSTANT = 0;                                        1A2L
      (lastnone) CONSTANT = 1;                                       1A2L
      (lastfield) CONSTANT = 2;                                      1A2L
      (lastnext) CONSTANT = 3;                                       1A2L
% Loading areas%
   (sharstg) [grammax + pagelen]; %area for compacted gram (sharable)%
                                                                     1A3
   (varstg) [varsmax + pagelen]; %area for compacted gram (private)%  1A3
   (oldgram) [grmsize]; %area for uncompacted gram%                   1A3
% Globals %
   % register save area - MUST REMAIN IN ORDER %
      (sv1ac1);                                                      1A4A
      (sv1acs)[15];                                                  1A4A
      (sv1ace);                                                      1A4A
   (record) REF; % page bound address of compacted grammar sharable) %
                                                                     1A4
   (vars) REF; % page bound address of compacted grammar private) %  1A4
   (opfreq) [ opmax ]; %array for grammar opcode frequencies%        1A4
   (gramtop); %number of last used byte in gram%                     1A4
   (grambot) REF; %points to first free loc in gram%                 1A4
   (gramend); %address of end of gram area%                          1A4
   (varsbot) REF; %points to first free loc in vars%                 1A4
   (varsend);  %address of end variable area%                        1A4
   (inst);  %instruction being built%                                1A4
   (instlen);  %final length of inst in 8-bit bytes%                 1A4
   (nuadd);  % addr field of inst%                                   1A4
   (nuval);  % val field of inst%                                    1A4
   (nuadd2);  % addr2 field for inst used for help code%            1A4
   (suc1);  %first byte of suc field%                                1A4
   (suc2);  % second byte of suc field%                             1A4
   (relsucadd);  % address of successor rel to current address%     1A4
   (sucfieldadd);  % address of current suc field%                  1A4
   (suclen);  % number of bytes in current suc field%              1A4
% Global tables and pointers %
   (addtab) [ maxinst ]; % inst addresses%                          1A4
```

```
    (altab) [ maxalts ]; % current alternatives%                          1A5
    (altabptr);                                                           1A5
    (kwtab) [bytemag]; % command words%                                   1A5
    (kwtabl);                                                             1A5
    (ectab) [bytemag]; % echo words%                                      1A5
    (ectabl);                                                             1A5
    (extab) [bytemag]; % old execute addresses%                           1A5
    (extabl);                                                             1A5
    (nextab) [bytemag]; % new execute addresses%                          1A5
    (lvtab) [bytemag]; % local variables%                                 1A5
    (lvtabl);                                                             1A5
    (gvtab) [bytemag]; % global variables%                                1A5
    (gvtabl);                                                             1A5
    (fntab) [bytemag]; % functions%                                       1A5
    (fntabl);                                                             1A5
    (nmtab) [bytemag]; % numeric constant table %                         1A5
    (nmtabl);                                                             1A5
    (pftab) [bytemag];                                                    1A5
    (pftabl);                                                             1A5
  % Strings %
    (string) STRING[80];                                                  1A6
  % Uncompacted gram record defs %
    (subr) RECORD % subsystem dispatch record %                           1A7
      sbnam[ADDRESS],    % ptr to subsystem name string %
      %the byte numbers below relative to gram%
        firstinst [ADDRESS],  % byte num of start of COMMANDS %
        sqekrule[ADDRESS],    % squeak rule for mouse buttons %
        helprule[ADDRESS], % global help rule/0 %
        initinst[ADDRESS],   % initialization rule /0 %
        reeninst[ADDRESS],   % rentry rule/0 %
        termrule[ADDRESS],   % termination rule/0,%
        hlprqrule[ADDRESS],   % help request rule/0,%
      prsrec[ADDRESS],  % ptr to process records/0,%
      kword[ADDRESS], %ptr to keywords%
      echoword[ADDRESS], %ptr to echo strings%
      execvector [ADDRESS], %ptr to vector of execute byte nums%
      pfuncs [ADDRESS], %ptr to parse function addrs in private part of
      grammar%
      pfuncl [ADDRESS], %number of parsefunction entries in pfuncs table%
      funcs [ADDRESS], %ptr to function records%
      gvarstart [ADDRESS], %start of global vars in private part of gram%
      varcnt[ADDRESS], % count of total number of variables %
      sharl [ADDRESS], % number of 32 word blocks in grammar sharable
      part%
      nums [ADDRESS], % pointer to numeric constants table %
      privl [ADDRESS]; % number of 32 word blocks in private part of
      grammar%
    (processr) RECORD %process/package record%                            1A7
      pkgstr[ADDRESS], %address of package name string%
      procstr[ADDRESS], %address of process name string%
      mailbox[
        pkgh[ADDRESS], %package handle%
        prcn[ADDRESS]]; %process handle/inchasn%
          %word prior to this record is link to next record%
    (instrec) RECORD  % CML instruction format %                          1A7
      alternative[ADDRESS],       %address of alternative instruction %
```

```
        successor[ADDRESS], % address of successor instruction %
        accr[ADDRESS],                % address/value field %
        val2[ tstrel[3], tstint[1], tstnot[1], tsttrue[1], tstnull[1],
        fillr2[1]],   %secondary value%
           % subfields are for XTEST:
              TSTNULL: test if accum contains null value.
              TSTTRUE: test if accum has non-null, non-zero value.
              TSTNOT: TRUE means to negate the test .
              TSTINT: implies test against integer in addr field.
              TSTREL: =, >, <, <=, >=.%
        filler[1],          % fill empty space in record %
        opcod[6],           % operation code %
        vtype[2]; %type of var if a var is referenced%
   % Compacted gram record defs %
      (nuinstrec) RECORD                                                1A8
        nuopcode[6],
        altsuc[2]; % alternative / successor field %
      (sucrec) RECORD                                                   1A8
        sucadd[7],
        long[1];
      (ivarr) RECORD                                                    1A8
        varind[6],
        vartype[2]; %for local vars%
   % misc. records %
      (b8rec) RECORD                                                    1A9
        b8rh[8],
        b8lh[8];
%Procedures%
   (cgraminit) %inits compacter and buffer for compiler output%
   PROCEDURE;                                                           1E
      %save registers%
        sv1ac1 _ R1;
        R1 _ $sv1acs;
        !BLT R1, sv1ace;
      % setup compiler globals %
        outcnt _ grmsize; % tell compiler how big compiled gram can be %
        relocc _ $oldgram; % relocation constant %
        outbfa _ $oldgram; % compiler relocation constant %
        outp _ 444488 + $oldgram; % compiler output buffer pointer %
      %restore registers%
        R1.LH _ $sv1acs;
        R1.RH _ 0;
        !BLT R1, 17B;
        R1 _ sv1ac1;
      RETURN;
      END.

   (cmgram) %compact the grammar - called by compiler%
   PROCEDURE;                                                           1F
      %save registers%
        sv1ac1 _ R1;
        R1 _ $sv1acs;
        !BLT R1, sv1ace;
      IF errorn = 0 THEN % no compile errors - do the compaction %
        BEGIN
        typeal($"");
```

```
        typeal($"-COMPACTING");
        compgram( fnumo );
        END;
    (terminate): closfile( fnumo, FALSE );                                  182
    %restore registers%
        R1.LH _ $sv1acs;
        R1.RH _ 0;
        !xLT R1, 17B;
        R1 _ sv1ac1;
    RETURN
    END.

(compgram) %compact grammar%
PROCEDURE ( outputjfn ) ;                                                   1E
    LOCAL CONSTANT
        blocksize = 32, %size of gramar blocks%
        blksperpage = pagelen / blocksize,
        bytperword = 4, %8-bit bytes per odp10 word%
        pplen = 3,
        pmappriv = 160400B6; %read, write, exe, copy on write%
    LOCAL i, j, word REF, bndis, lc, disrec REF, privpage, temp,
    nuinitadd, nurentadd, nutermadd, nufirstadd, nuhelpadd, nusqekadd,
    nuhlprqadd, errnum, ok;
    % Initialize globals %
        clrbf( $aoctab, maxinst);
        clrbf( $opfreq, opmax);
        kwtabl _ ectabl _ extabl _ lvtabl _ gvtabl _ fntabl _ nmtabl _
        noitems; % tables are empty %
        pftabl _ noitems + 1; % parsefuns, cannot have an index of zero as
        this indicates absence of a pf in xselect%
        &record _ divru($sharstg, pagelen) * pagelen; %record points to
        first page boundary in sharstg%
        &vars _ divru($varstg, pagelen) * pagelen; %vars points to first
        page boundary in varstg%
        &disrec _ oldgram[1];
        gramtop _ (grammax - subr.SIZE) * bytperword +1;
        gramend _ &record + grammax - 1;
        varsend _ &vars + varsmax - 1;
    nuinitadd _ compaltS( disrec.initinst, FALSE ); %initialization rule%
    nurentadd _ compaltS( disrec.reeninst, FALSE );% reentry rule %
    nutermadd _ compaltS( disrec.termrule, FALSE ); %termination rule%
    nuhelpadd _ compaltS( disrec.helprule, FALSE ); %help rule%
    nuhlprqadd _ compaltS( disrec.hlprqrule, FALSE ); %help request rule%
    nusqekadd _ compaltS( disrec.sqekrule, FALSE ); %squeak rule%
    nufirstadd _ compaltS( disrec.firstinst, TRUE );
    %process executes%
        FOR i _ 0 UP UNTIL > extabl DO
            IF NOT nextab[ i] _ getadd( extab[ i] ) THEN
                nextab[ i] _ compaltS( extab[ i], FALSE );
    (stuff): %Stuff compacted grammar into buffer%                          183
        % move grammar from end of buffer to beginning of buffer %
            &grambot _ &record + subr.SIZE; % address for first byte of
            grammar %
            lc _ rbntobp(gramtop, &record + subr.SIZE).RH; % address of
            first word currently occupied by grammar %
            i _ gramend - lc + 1; % size of grammar %
```

```
    mvbfbf( lc, &grambot, i); % move the grammar %
    &grambot _ &grambot + i;
bndis _ gramtop - (((gramtop - 1) MOD bytperword) + 1) - (subr.SIZE
* bytperword);
&varsbot _ &vars;
record.firstinst _ IF nufirstadd THEN nufirstadd - bndis ELSE 0;
record.initinst _ IF nuinitadd THEN nuinitadd - bndis ELSE 0;
record.reeninst _ IF nurentadd THEN nurentadd - bndis ELSE 0;
record.termrule _ IF nutermadd THEN nutermadd - bndis ELSE 0;
record.helprule _ IF nuhelpadd THEN nuhelpadd - bndis ELSE 0;
record.hlprgrule _ IF nuhlprgadd THEN nuhlprgadd - bndis ELSE 0;
record.seekrule _ IF nuseekadd THEN nuseekadd - bndis ELSE 0;
record.kword _ gramstuf( $kwtab, kwtabl + 1 );
record.echoword _ gramstuf( $ectab, ectabl + 1 );
record.nums _ gramstuf( $nmtab, nmtabl + 1 );
%Stuff execute vector and relocate addresses%
    FOR i _ 0 UP UNTIL > extabl DO
        nextab[ i] _ nextab[i ] - bndis;
    record.execvector _ gramstuf( $nextab, extabl + 1 );
%stuff kw's and relocate addresses%
    FOR &word _ record.kword + &record UP UNTIL = record.kword
    +kwtabl + &record +1 DO
        IF [word - 1] NOT = -1 THEN %selector%
            BEGIN
            FOR i _ word -3 UP UNTIL >  word - 1 DO
                BEGIN
                IF [i] THEN
                    temp _  getind( [i], $pftab, $pftabl)
                ELSE
                    temp _ 0;
                gramstuf( $temp, 1 );
                END;
            word _ gramstuf( word, wlen( [word] ) );
            END
        ELSE %regular kword%
            BEGIN
            word _ gramstuf( word, wlen( [word] ) );
            END;
record.funcs _ gramstuf( $fntab, fntabl + 1 );
%stuff echo strings and relocate addresses%
    FOR &word _ record.echoword + &record UP UNTIL = record.echoword
    +ectabl + &record +1 DO
        word _ gramstuf( word, wlen( [word] ) );
%stuff subsys name%
    record.sbnam _ gramstuf( disrec.sbram, wlen( [disrec.sbnam]) );
%stuff variables%
    FOR i _ 0 UP UNTIL > lvtabl DO
        BEGIN
        varsbot _ [ lvtab[ i ] ];
        BUMP &varsbot;
        END;
    record.gvarstart _ &varsbot - &vars;
    FOR i _ 0 UP UNTIL > gvtabl DO
        BEGIN
        varsbot _ [ gvtab[ i ] ];
        BUMP &varsbot;
```

```
            END;
         record.varcnt _ lvtabl + gvtabl + 2;
      %stuff process/package records%
         IF &word _ disrec.prsrec THEN
            BEGIN
            record.prsrec _ &varsbot - &vars + 1;
            DO
               BEGIN
               IF word.procstr THEN %stuff%
                  word.procstr _ gramstuf( word.procstr, wlen(
                  [word.procstr] ) );
               IF word.pkgstr THEN %stuff%
                  word.pkgstr _ gramstuf( word.pkgstr, wlen(
                  [word.pkgstr] ) );
               IF (temp _ [ &word - 1]) THEN
                  [ &word - 1] _ &varsbot + pplen - &vars + 1;
               putnewadd( &word, varstuf( &word - 1, pplen ) +1 );
               END
            WHILE ( &word _ temp );
            END
         ELSE
            record.prsrec _ 0;
      % change pf names to radix50 and stuff%
         FOR i _ 1 UP UNTIL > pftabl DO pftab[i] _ makrad50( pftab[i]);
         record.pfuncs _ varstuf( $pftab, pftabl + 1 );
         record.pfuncl _ pftabl; % number of pf entries %
      %stuff function records%
         FOR &word _ record.funcs + &record UP UNTIL = record.funcs +
         &record + fntabl +1 DO
            BEGIN
            [ word ] _ getadd( [ word ] );
            word _ gramstuf( word, wlen( [ word +1 ]) +1);
            END;
   (fillais): %Fill in dispatch record%                                  183
      record.sharl _ divru( (&grambot - &record), blocksize);
      privpage _ &vars / pagelen;
      record.privl _ divru( (&varsbot - &vars), blocksize );
   (file): %Output to file%                                              183
      mapper( outputjfn, privpage, divru(record.privl, blksperpage),
      divru(record.sharl, blksperpage), pmappriv );
      mapper( outputjfn, &record / pagelen, divru(record.sharl,
      blksperpage), 0, pmappriv );
   RETURN;
   END.

(mapper) %pmaps into a file%
PROCEDURE ( jfn, srcepage, pages, destpage, access );                    18
   LOCAL CONSTANT
      remove = -1; %remove page from fork%
   LOCAL
      srce _ 400000000000B, %source is current fork%
      dest, %destination%
      lastpage _ srcepage + pages;
   dest.LH _ jfn;
   dest.RH _ destpage;
   FOR srce.RH _ srcepage UP UNTIL >= lastpage DO
```

```
      BEGIN
      !pmap( srce, dest, access);
      !pmap( remove, srce, access);
      BUMP dest.RH;
      END;
   RETURN;
   END.

(closfile) %close file%
PROCEDURE ( jfn, release );                                             1E
   IF NOT release THEN jfn _ jfn + 4B11; % set high bit not release jfn %
   !closf( jfn);
   !JFCL;
   RETURN;
   END.

(divru) % divide and rounds up to next highest integer %
PROCEDURE ( dividend, divisor );                                        1E
   LOCAL quot, remain;
   DIV dividend / divisor, quot, remain;
   RETURN ( IF remain THEN quot + 1 ELSE quot);
   END.

(wlen) %gives length of a string in words%
PROCEDURE ( s );                                                        1E
   LOCAL CONSTANT charperword = 5; %chars in pdp10 word%
   RETURN ( divru( s.L, charperword) + 1 );
   END.

(gramstuf) %stuff table into gram buffer. Return ptr to first word
relative to start of gram%
PROCEDURE (blok REF, blokl );                                           19
   LOCAL k, ptr;
   ptr _ &grambot - &record;
   FOR k _ 0 UP UNTIL = blokl DO
      BEGIN
      grambot _ blok[ k ];
      BUMP &grambot;
      IF &grambot = gramend THEN errzzz( $"Compacted grammar too large");
      END;
   RETURN [TRUE] ( ptr );
   END.
(varstuf) %stuff table into vars buffer. Return ptr relative to begin of
vars to first word%
PROCEDURE (blok REF, blokl );                                           1E
   LOCAL k, ptr;
   ptr _ &varsbot - &vars;
   FOR k _ 0 UP UNTIL = blokl DO
      BEGIN
      varsbot _ blok[ k ];
      BUMP &varsbot;
      IF &varsbot > varsend THEN
         errzzz( $"Compacted grammar too large");
      END;
   RETURN ( ptr );
   END.
```

```
(compalts) %compact alternatives%
PROCEDURE ( firstalto, revorder );                                    1B:
   LOCAL CONSTANT
      smallsuccdis = 128,
      maxsuccdis = smallsuccdis * bytemax,
      varkwbit = 2006;
   LOCAL
      newaddr,
      i,
      frstinst, lastinst,
      alto REF _ firstalto;
   IF NOT firstalto THEN RETURN( 0 );
   DO %process successors for all alternatives if not already processed%
      IF alto.successor THEN
         IF alto.alternative NOT = alto.successor THEN
            IF NOT getadd( alto.successor ) THEN
               compalts( alto.successor, FALSE )
   WHILE &alto _ alto.alternative;
   %put current alts into grammar%
   &alto _ firstalto;
   IF revorder THEN %reverse the order of the alts%
      BEGIN
      altabptr _ $altab;
      DO
         BEGIN
         IF altabptr < $altab + maxalts THEN
            [altabptr] _ &alto
         ELSE
            errzzz( $"Too many parallel alternatives");
         BUMP altabptr;
         END
      WHILE  &alto _ alto.alternative;
      frstinst _ $altab; lastinst _ altabptr;
      END
   ELSE %just put in%
      BEGIN
      altabptr _ $altab + maxalts;
      DO
         BEGIN
         BUMP DOWN altabptr;
         IF altabptr >= $altab THEN
            [altabptr] _ &alto
         ELSE
            errzzz( $"Too many parallel alternatives");
         END
      WHILE $alto _ alto.alternative;
      frstinst _ altabptr; lastinst _ $altab + maxalts;
      END;
   FOR i _ frstinst UP UNTIL >= lastinst DO %put current alts into gram%
      BEGIN
      &alto _ [i];
      inst.nuopcode _ alto.opcoc;
      CASE inst.nuopcode OF
         = storeop, = load, = append, = loadi, = ldlste, = stlste, =
         inlste, = dllste, = indexof: %variable%
```

```
    BEGIN
    instlen _ 2;
    nuacc _ instvar( &alto);
    END;
= keyco1, = enterc%, = keyco: %command word%
    BEGIN
    instlen _ 4;
    nuadd _ getind( alto.addr, $kwtab, $kwtabl );
    nuval _ [&alto + 2].b8lh;
    nuadd2 _ [&alto + 2].b8rh;
    END;
= pushvar:
    BEGIN
    instlen _ 2;
    nuadd _ instvar( &alto);
    END;
= echo, = recho, = enterstr: %string%
    BEGIN
    instlen _ 2;
    nuadd _ getind( alto.addr, $ectab, $ectabl );
    END;
= execute, = loop: %index to extab%
    BEGIN
    instlen _ 2;
    nuadd _ getind( alto.addr, $extab, $extabl );
    END;
= call, = ocall: %function%
    BEGIN
    instlen _ 3;
    nuadd _ getind( alto.addr, $fntab, $fntabl );
    nuval _ alto.val2;
    END;
= callhelp, = ocllrule: %function with rule%
    BEGIN
    instlen _ 4;
    nuadd _ getind( alto.addr, $fntab, $fntabl );
    nuval _ alto.val2;
    nuadd2 _ getind( [ &alto + 2], $extab, $extabl);
    END;
= test:   %test may be test, testtrue, or testnull%
    CASE TRUE OF
        = alto.tsttrue :
            BEGIN
            instlen _ 1;
            inst.nuopcode _ IF alto.tstnot THEN testntrue ELSE
            testtrue;
            END;
        = alto.tstnull :
            BEGIN
            instlen _ 1;
            inst.nuopcode _ IF alto.tstnot THEN testnnull ELSE
            testnull;
            END;
        = alto.tstint : %addr field points to integer%
            BEGIN
            instlen _ 3;
```

```
                nuadd _ getind( [alto.addr], $nmtab, $nmtabl );
                nuval _ alto.val2;
                END;
            ENDCASE %variable in addr field%
                BEGIN
                instlen _ 3;
                nuadd _ firstvar(&alto);
                nuval _ alto.val2;
                END;
        = pfcllop, = ffcllop: %parsefunction or fefunction%
            BEGIN
            instlen _ 3;
            nuadd _ getind( alto.addr, $pftab, $pftabl );
            nuval _ alto.val2;
            END;
        = enter:
            BEGIN
            instlen _ 2;
            nuadd _ getind( [alto.addr], $nmtab, $nmtabl );
            END;
        = rptloop, = exitloop:
            BEGIN
            instlen _ 2;
            nuadd _ alto.addr; % nesting level %
            END;
        ENDCASE
            instlen _ 1;
    BUMP opfreq[ inst.nuopcode ];
    IF i NOT= firstinst THEN %not last alt%
        inst.altsuc _ notlast
    ELSE %last alt%
        BEGIN
        IF alto.successor THEN %see if next is suc%
            BEGIN
            IF getadd( alto.successor ) = gramtop THEN %next is suc%
                inst.altsuc _ lastnext
            ELSE %suc isnt next %
                inst.altsuc _ lastfield;
            END
        ELSE %last alt has no suc%
            inst.altsuc _ lastnone;
        END;
    suclen _ 0;
    IF inst.altsuc = notlast OR inst.altsuc = lastfield THEN %make suc
    field%
        BEGIN
        IF alto.successor THEN %calc new suc address%
            BEGIN
            sucfieldadd _ gramtop - 1;
            relsucadd _ getadd( alto.successor ) - sucfieldadd;
            CASE relsucadd OF
                IN [1,smallsucdis):
                    BEGIN
                    suc1.long _ FALSE;
                    suc1.sucadd _ relsucadd;
                    suclen _ 1;
```

```
                    END;
                IN [smallsuccis,maxsucdis): %sucadd is rel to 2nd byte of
                sucfld%
                    BEGIN
                    suc1.long _ TRUE;
                    DIV relsucadd / bytemag, suc1.sucadd, suc2;
                    suclen _ 2;
                    END;
                ENDCASE errzzz( $"Bad new successor addr");
                END
            ELSE %zero suc field%
                BEGIN
                suc1 _ 0;
                suclen _ 1;
                END;
            END;
        newaddr _ putingram();
        putnewadd( $alto, newaddr );
        END;
    RETURN( newaddr %of first alt%);
    END.

(instvar) %makes var field for an oldinst%
PROCEDURE( oldinst REF );                                                    18
    LOCAL field;
    CASE oldinst.vtype OF
        = local:
            IF (field _ getind( oldinst.addr, $lvtab, $lvtabl )) >= 128 THEN
            errzzz( $"Too many local vars");
        = global:
            BEGIN
            field.vartype _ global;
            IF (field.varind _ getind(oldinst.addr,$gvtab, $gvtabl)) >= 64
            THEN errzzz( $"Too many global vars");
            END;
        = bltin:
            BEGIN
            field.vartype _ bltin;
            field.varind _ oldinst.addr;
            END;
        ENDCASE;
    RETURN(field);
    END.

(getind) %get table index; if none found, insert it%
PROCEDURE ( oldaddr, table REF, tablen REF);                                 18
    LOCAL i;
    FOR i _ 0 UP UNTIL > tablen DO
        IF table[ i] = oldaddr THEN
            RETURN( i );
    BUMP tablen;
    IF tablen >= bytemag THEN  errzzz( $"More than 256 entries in a symbo
    table");
    table[ tablen ] _ oldaddr;
    RETURN( tablen );
    END.
```

```
(outingram) %out current inst in grammar%
PROCEDURE:                                                          131
   LOCAL fd;
   gramtop _ gramtop - (instlen + suclen);
   IF gramtop <= 0 THEN errzzz( $"Compacted grammar too large");
   fd _ rbntobp( gramtop, %record + subr.SIZE );
   .fd _ inst;
   CASE instlen OF
      = 2: %includes addr field%
         |fd _ nuadd;
      = 3: %includes addr and val fields%
         BEGIN
         |fd _ nuadd;
         |fd _ nuval;
         END;
      = 4: %callhelp opcode%
         BEGIN
         |fd _ nuadd;
         |fd _ nuval;
         |fd _ nuadd2;
         END;
      ENDCASE;
   CASE suclen OF
      = 1: %small suc field%
         |fd _ suc1;
      = 2: %long suc field%
         BEGIN
         |fd _ suc1;
         |fd _ suc2;
         END;
      ENDCASE; %no suc field%
   RETURN( gramtop );
   END.

(rbntobp) %convert byte number relative to base to a byte pointer%
PROCEDURE ( rbn, base );                                            181
   % byte numbers start at 1 %
   LOCAL CONSTANT byperwd = 4; % 4 8-bit bytes per word %
   LOCAL words, bp;
   bp _ 341C0086 + base;
   BUMP DOWN rbn;
   CASE rbn OF
      <= byperwd: % just loop to increment %
         FOR rbn DOWN UNTIL <= 0 DO !IBP bp; % increment the byte ptr %
      ENDCASE
         BEGIN
         DIV rbn / byperwd, words, rbn;
         bp.RH _ bp.RH + words;
         REPEAT CASE;
         END;
   RETURN( bp );
   END.

(getadd) %get new address of argument%
PROCEDURE ( oldaddr );                                              191
```

```
    LOCAL indhash, ind;
    indhash _ ( oldaddr MOD maxinst );
    ind _ indhash;
    DO
       CASE addtab[ ind ].RH OF
          = oldaddr: RETURN( addtab[ ind ].LH );
          = 0: RETURN( 0 );
          ENDCASE NULL
    UNTIL (ind _ ((ind+1) MOD maxinst)) = indhash;
    RETURN( 0 );
    END.

(putnewadd) %put new address in table%
PROCEDURE ( oldaddr, nwaddr );                                  181
    LOCAL indhash, ind;
    indhash _ ( oldaddr MOD maxinst );
    ind _ indhash;
    DO
       CASE addtab[ ind ].RH OF
          = 0: %empty entry%
             BEGIN
             addtab[ ind ].RH _ oldaddr;
             addtab[ ind ].LH _ nwaddr;
             RETURN;
             END
          ENDCASE %used entry% NULL
    UNTIL (ind _ ((ind+1) MOD maxinst)) = indhash;
    errzzz( $"Too many new addresses");
    RETURN;
    END.
(makrad50) % change an 110 string into rad50 %
PROCEDURE ( str REF);                                           181
    LOCAL CONSTANT chbmty = 440700000001B;
    LOCAL rad50, i, char, charptr;
    rad50 _ 0;
    charptr _ $str + chbmty;
    FOR i _ 1 UP UNTIL > MIN( 6, str.L) DO
       BEGIN
       CASE char _ |charptr OF
          IN ["A, "Z]:      char _ char - 54;
          IN ["a, "z]:      char _ char - 86;
          IN ["0, "9]:      char _ char - 47;
          ENDCASE errzzz( $"Bad rad50 char");
       rad50 _ rad50*506 + char;
       END;
    RETURN ( rad50);
    END.%%
```

```
(errzzz) PROCEDURE ( astrng REF );                                      1B1
   cmerr();
   printerror( &astrng);
   GOTO terminate; % simulate a catchphrase termination%
   END.
(cmerr) PROCEDURE;                                                      1B1
   BUMP errorn; % inc compilers error count %
   RETURN;
   END.
(printerror) PROCEDURE ( astrng REF );                                  1B2
   typeas($"ERROR: ");
   typeal( &astrng );
   RETURN;
   END.
(typeal) PROCEDURE ( astrng REF);                                       1B2
   IF astrng.L > 0 THEN
      wstri( &astrng ); % have compiler buffer string %
   tcrlf();
   RETURN;
   END.
(typeas) PROCEDURE (astrng REF);                                        1B2
   IF NOT astrng.L THEN RETURN;
   wstr( &astrng); % have compiler write string %
   RETURN;
   END.
(mvbtbf) %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.%
                                                                        1B2
   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.

(clrbf) % clear a buffer%
PROCEDURE (buf REF, nw);                                                1B2
LOCAL lw, bufadr;
```

```
    IF nw = 0 THEN RETURN;
    buf[0] _ 0;
    lw _ &buf + nw -1; %last word to be transferred to%
    butads.LH _ &buf;
    butads.RH _ &buf + 1;
    A1 _ butads;
    !BLT A1,@lw;
    RETURN;
    END.

    (flcpfs) PROCEDURE; RETURN;END.                                    183
    (filesc) PROCEDURE; RETURN; END.                                   183
    (flfechc1) PROCEDURE; RETURN; END.                                 183
FINISH
```

(doc) Documentation

The compacter is combined with the compiler at load time. The compiler must be used with mini-l10runtime and fullsize xl10data. Therefore, the compacter can only use features that are supported by the mini-l10runtime system.

Compacted grammar file format with n sharable pages and m private pages:
Pages 0 to n-1: sharable, Pages n to n+m-1: private

```
FILE cml10 CHECK % (arcsubsys,meta9,) (fe,cml10.rel,) %
   META file                                                                    1
   % DECLARATIONS %
      DUMMY: dispatch remote helpcall funres alter succ succargs option pseudo
      pushi ulcop ulistemanip;                                                  1F
      ERROR: -> *;                                                              1F
         (?IF errtype = 1 $condcomp $dcls $rule #subsys "FINIS" /
         $(command / rule) "END." $subsys "FINIS");
      SIZE: S=3000 M=100 K=100 N=1500 L=300 G=100;                              1F
      SET:
         VALIDSUBSYS = 110474B  %validation code for subsys dispatch record%
         IDFLG = 1186 % kludge to make hash work %
         % TEST OPCODE BITS %
            TSTNULL=100B
            TSTTRUE=40B
            TSTNOT=20B
            TSTINT=10B
         % RECOGNIZERS %
            ABORTOP=0   %abort op code%
            KEYOP=189   % keyword recognition operator (level 2) %
            KEYOP1=3589 % keyword recognition operator (level 1) %
            VKEYOP=3689 % variable keyword recognition (level 1) %
            CONFIRM=289 % get command confirmation %
            SSEL=389    % source selection %
            DSEL=489    % destination selection %
            LSEL=589    % literal selection %
            ANSWER=789  % gets viewspecs %
            OPTION=1089 % option character %
            RPTOP=4089  % repeat character %
         % CONTROL ELEMENTS %
            PFCALL=1189 % parsing function %
            EXECUTE=1289        % transfer to another point in tree %
            CALL=1389   % Backend function call with no associated help rule %
            CALLHELP=3489       % Backend function call with associated help
            rule %
            OCALL=4189  % Outofline Backend function call with no continue rul
            %
            OCLLRULE=4289       % Outofline Backend function call with continu
            rule %
            FECALL=5689 % call on function in Frontend %
         % FEEDBACK ELEMENTS %
            FECLEAR=1489        % clear feedback buffer %
            ECHO=1589   % echo noise word string %
            RECHO=1689  % replace last thing echoed %
            SHOW=2389   % display a variable to user %
            SHOWCON=4489        % display a variable to user and require
            confirm char %
            SHOWSTAT=4589       % display in tty window a variable to user %
            CLEARSTAT=5789      % clear tty window %
            LOOP=5189   % Beginning of a CML loop %
            RPTLOOP=5289        % repeat a CML loop %
            EXITLOOP=5389       % exit a CML loop %
            UNTILLOOP=5489      % until clause of a CML loop %
            WHILELOOP=5589      % while clause of a CML loop %
         % VALUE MANIPULATIONS %
            PUSHARG=689 % push accum on arg stack %
```

```
        PUSHVAR=3789          % push variable on arg stack %
        STORE=1789  % store accum value into variable %
        APPEND = 3189         % append accum to list variable %
        LOAD=2089             % load variable into accum %
        LOADI=4389            % load variable address into accum %
        ENTER=2189  % enter constant into accum %
        LOLSTE=6089 % load a list element into accum %
        STLSTE=6189                   % store list element into accum %
        INLSTE=6289                   % insert a list element following giver
        element %
        DLLSTE=6389           % delete a list element from a list %
        INDEXOF=6589          % find index of expression in list  %
      % TEST A VARIABLE %
        TEST=2289    % test for TRUE variable %
      % ENTER CML VALUE INTO ACCUMULATOR %
        ENTERCW=2489          % enter command word into accum %
        ENTERSTR=6489         % enter string into accum %
        ENTERNULL = 2589       %set accum to null%
        ENTERTRUE = 2689     %BOOLEAN = TRUE%
        ENTERFALSE = 2789     %BOOLEAN = FALSE%
      % RESUME HELP%
        RESUME= 3089          %resume help call%
        RESUMNULL= 4689       %resume help call with no value%
        ;
    FLAGS:                                                              1
        attr              % attribute values %                           18
        initloc           % ptr to INITIALIZATION rule %                 18
        termloc           % ptr to TERMINATION rule %                    18
        hlprqloc          % ptr to HELPREQUEST rule %                    18
        rentryloc         % ptr to RENTRY rule %                         18
        helploc           % ptr to HELP rule %                           18
        scekloc           % ptr to SQUEAK rule called when user pushes on mouse
                                                                          18
        sav               % id value for command %                       18
        value             % temporary accumulator %                     18
        testtype          % true or false test %                         18
        prcr              % addr of current process/package record %     18
        pointrtn                %address of coroutine to call for a pointing
        selection%                                                        18
        typeinrtn               %address of typeir collection routine%    18
        addressrtn              %address of address collection routine%   18
        ctlbits           % temporary accumulator %                      18
        top               % address of last generated node %             18
        alt               % address of alternative node %                18
        suc               % address of successor node %                  18
        errtype;                % type of error %                        18
          %1 => scan for ; then expect more dcls or commands or rules
          2 => scan for ; then expect more commands or rules%           185
    FIELDS: OP=[6:27] VAL=[8:18] LH=[18:18] RH=[18:0] VARTYPE=[2:33];    1
    ATTRIBUTES:                                                          1
        parallel                % BE fn can be called in parallel with parsing
        %                                                                18
        pfunctname      % name of external parsing function %           18
        ffunctname      % name of external frontend function %          18
        functname                % name of external L10 function %      18
        rulename                 % name of a parse rule %               18
```

```
    varname                 % temporary variable %                      187
    constant                %constant (equate)%                         187
    litstr          % .ID has literal associated with it %               187
    bltinvar                %var is a builtin%                          187
    strconst        % id is a string constant %                         187
    globalvar;      %scope of var is entire grammar%                    187
% PARSING RULES %
  % file definition %
    file =                                                              1C1
      ("FILE"
        &tonls _ 1 % output to compacter in same address space %
        &cgraminit % initialize the compacter %
        .ID <"-CML COMPILER 7/7/76">
        <"-FILE "*1> &NAME &DISCARD
        :[VALIDSUBSYS, G#dispatch,] % output address of dispatch record
        %
         &prcr _ 0 &errtype _ 1 $condcomp $dcls $rule  % can have global
        rules or declarations %
        #(subsys) % must have at least one subsystem def %
        "FINIS" :fin[] *
        &cmgram) % compact the grammar %
      /"SYMBOLS"
        $<',> (.ID '= .NUM :xbivar[2]* ) ';  "FINIS";
      xbivar [-,-] => >*1 _ *N2 @S varname *1 @S bltinvar *1 @P reloca
      *1;                                                               1C1A
      fin => % fin processing and output the literals %                 1C1A
        % output a L10 string for all keyword strings which are not
        external %
          >|"SEGLIT"
          $SYMS(
            ?NOT ?@ defneo *$ (
              ?NOT ?@ extern *$
                (?@ litstr *$
                  -1, % not a selector cw %                     1C1A4A2A1A
                  >*$
                  *L$|LH *L$,
                  *S$
                /?$ varname *$ >*$ 0\0
                / (?@ pfunctname *$ / ?@ ffunctname *$) % put out
                the string %
                  >*$
                  *L$|LH *L$,
                  *S$
                /?@ rulename *$ <"ERROR: " *$ " not declared or
                defined"> &cmerr
                /TRUE)
              /TRUE)
            /TRUE)
        % put out any literals %
          &literals;
    % Rule name definitions %
      rulname[.ID] => % define the rule as a local symbol %             1C2
        @S rulename *1                                                  1C2A
        >*1 _ top @S reloca *1;
    % DCL definitions %
      condcomp = %conditional compilation switches%                     1C7
```

```
        "SET"
        #<*,> ( .UID *=
            ("TRUE" >*1 _ 1 / "FALSE" >*1 _ 0) &DISCARD)
        *; ;                                                          1C3A
    dcls = (                                                          1C3
        ("DCL" / "DECLARE")
            &attr _ 1 % default is VARIABLE %                         1C3D1
            (
            dclcw                                                     1C3R1
            / (dclfun  .#<*,> dclitem :dcllist[$] *; *)
            / ("CONSTANT"
               #<*,> (.ID *= .NUM :xconst[2] *) *;)
            / ("STRING"
               #<*,> (.ID *= .SR :xstring[2] *) *;)
            / ([dclattr]  .#<*,> .ID :dcllist[$] *; *)
            )
        / "CONSTANT"
            #<*,> (.ID *= .NUM :xconst[2] *) *;);
    dclcw = % declare command words%                                  1C3
        "COMMAND" "WORD"
        #<*,>
            ( .SR [?@ defned <"ERROR: Command word " *1 " previously
            defined"> &cmerr]
            ( *= .NUM [?IF *N2 > 177777B <"ERROR: Command word " *1 " has
            number greater than 177777B"> &cmerr] :xdclcw[2]
            / :xdclcw[1, =0])
            (( "SELECTOR" &pointrtn _ 0 &typeinrtn _ 0 &addressrtn _ 0
               (*= biselectors
               / $(
                  "POINT" *=
                     (.ID @S pfunctname &pointrtn _ HASH(*2) + IDFLG
                     &DISCARD
                     / !typeinrtn, addressrtn (biselectors))
                  / "TYPEIN" *=
                     (.ID @S pfunctname &typeinrtn _ HASH(*2) + IDFLG
                     &DISCARD
                     / !pointrtn, addressrtn (biselectors))
                  / "ADDRESS" *=
                     (.ID
                     / "TEXT" :"ttext"
                     / "CHARACTER" :"acharacter")
                     @S pfunctname &addressrtn _ HASH(*2) + IDFLG &DISCARD
                                                                      1C3C2D2
                     )
               )
            :sel[1, =typeinrtn, =pointrtn, =addressrtn] )
            / :[ -1 , ] %distinction of cws for compactor%)
            *)
        *; ;                                                          1C3C
    biselectors =                                                     1C3
        :"ttext" @S pfunctname &typeinrtn _ HASH(*1) + IDFLG &DISCARD
        :"acharacter" @S pfunctname &addressrtn _ HASH(*1) + IDFLG &DISCARD
        ((
            ("CHARACTER" :"pcharacter"  /
            "WORD" :"pword"  /
            "VISIBLE" :"pvisible"  /
```

```
        "INVISIBLE" :"pinvisible"  /
        "TEXT" :"ttext" @S pfunctname &addressrtn _ HASH(*1) + IDFLG
        &DISCARD :"ptext"  /
        "INTEGER" :"ttint" @S pfunctname &typeinrtn _ HASH(*1) + IDFLG
        &DISCARD :"pinteger"  /
        "NUMBER" :"pnumber"  /
        "STRING" :"pstring"  /
        "OLDFILENAME" :"toldfilename" @S pfunctname &typeinrtn _
        HASH(*1) + IDFLG &DISCARD :"poldfilename"  /
        "NEWFILENAME" :"tnewfilename" @S pfunctname &typeinrtn _
        HASH(*1) + IDFLG &DISCARD :"pnewfilename"  /
        "FILENAME" :"tfilename" @S pfunctname &typeinrtn _ HASH(*1) +
        IDFLG &DISCARD :"pfilename"  /
        "CHARPOS" ["ITION"] :"pcharpos")
     @S pfunctname &pointrtn _ HASH(*1) + IDFLG &DISCARD)  /          1C3D
     "PASSWORD" :"tpassword" @S pfunctname &typeinrtn _ HASH(*1) + IDFLG
     &DISCARD &pointrtn _ 0) ;
  sel[xdclcw[-,-], -,-,-] =>                                          1C3
     (?IF *N3 = 0 0, / *V3,)
     (?IF *N2 = 0 0, / *V2,)
     (?IF *N4 = 0 0, / *V4,)
     *1;
  dclattr = % declaration attributes %                                1C3
     ("VARIABLE" &attr _ 1
     / "PARSEFUNCTION" &attr _ 3
     / "GLOBAL" [ "VARIABLE" ] &attr _ 4
     / ( "FE" / "FRONT" "END" ) "FUNCTION" &attr _ 5);
  dclfun = % declare function %                                       1C3
     "FUNCTION" &attr _ 2
        ("PROCESS" *= .SR
           ( *, "PACKAGE" *= .SR *:  :xfnhdr[2]*
           / *: :xfnhdr[1]* )
        / :xfnhdr[]*);
  xdclcw                                                              1C3
     [-,-] =>
        >*1 _ LSH(*N2)18+. @S reloca *1 *L1|LH *L1, *S1;
  xconst [-,-] => >*1 _ *N2  @S constant *1 @R reloca *1;              1C3
  xstring [-,-] =>                                                    1C3
     >*2 *L2|LH *L2, *S2
     >*1 _ *V2 @S strconst *1;
  xfnhdr                                                              1C3
     [.SR,.SR] =>
        >*1 *L1|LH *L1, *S1 >*2 *L2|LH *L2, *S2
        ( ?IF prcr = 0 0\0 / prcr\1) &prcr _ .
           %fnrecord -1 = addressrtn of next fnrecord or zero%
        *V1|LH *V2\3 HASH(*1)|LH HASH(*2),:
           %fnrecord =
              ADDRESS(process),,ADDRESS(package),                     1C3K1C1
              HASH(process),,HASH(package)%                           1C3K1C1
     [.SR] =>
        >*1 *L1|LH *L1, *S1
        ( ?IF prcr = 0 0\0 / prcr\1) &prcr _ .
           %fnrecord -1 = addressrtn of next fnrecord or zero%
        *V1|LH\2 HASH(*1)|LH ,;
     [] =>
        ( ?IF prcr = 0 0\0 / prcr\1) &prcr _ .
```

```
            %fnrecord -1 = addressrtn of next fnrecord or zero%
         $BSS 2,;                                                    1C3K3
            %null fnrecord =
               0, 0 %                                                1C3K381
      cclitem = .ID [("OUT" "OF" "LINE" @S parallel) / ("PSEUDONYM" *= .ID
      ["OUT" "OF" "LINE" @S parallel] :pseudo[2])] ;                    1C3
      dollist                                                           1C3
         [pseudo[.ID,.ID]] =>
            [ ?@ defned *1:2 <"ERROR: " *1:2 " previously defined" LOC LINE>
            &cmerr]
            ?IF attr = 2 % FUNCTION %
               @S functname *1:2 >*1:2 prcr\1 *L1:1|LH *L1:1, *S1:1 ; 1C3M1B
                  %FNname = ADDRESS(fnrecord), STRING(fn)%
         [$] =>
            ( ?IF attr = 1 % VARIABLE %
               $( ?@ defned *$ <"ERROR: " *$ " previously defined" LOC LINE>
               &cmerr/ @S varname *$ >*$ &BSS 1,)
            / ?IF attr = 2 % FUNCTION %
               $( ?@ defned *$ <"ERROR: " *$ " previously defined" LOC LINE>
               &cmerr/ @S functname *$ >*$ prcr\1 *L$|LH *L$, *S$  )
                  %FNname = ADDRESS(fnrecord), STRING(fn)%
            / ?IF attr = 3 % PARSEFUNCTION %
               $( ?@ defned *$ <"ERROR: " *$ " previously defined" LOC LINE>
               &cmerr/ @S pfunctname *$ >*$ *L$|LH *L$, *S$ )
            / ?IF attr = 5 % FE FUNCTION %
               $( ?@ defned *$ <"ERROR: " *$ " previously defined" LOC LINE>
               &cmerr/ @S ffunctname *$ >*$ *L$|LH *L$, *S$ )
            / ?IF attr = 4 % GLOBALVAR %
               $( ?@ defned *$ <"ERROR: " *$ " previously defined" LOC LINE>
               &cmerr/ @S globalvar,varname *$ >*$ &BSS 1, ));
   % SUBSYSTEM definition %
      subsys = "SUBSYSTEM" .ID &LABEL &errtype _ 2                      1C4
         "KEYWORD" .P1SR % recognize as a prefixed string to distinguish
         from cw %
         :subs2[ 2 ]
         &MARK &sav _ 0 &initloc _ 0  &termloc _ 0  &rentryloc _ 0 &helploc
         _ 0  &spekloc _ 0  &hlprqloc _ 0
         #(command / rule)
         &UNMARK &top _ sav *  "END.";
      subs2 [.ID, .SR] =>                                               1C4
         % define the subsystem identifier as an external symbol whose value
         is the address of the subsystem dispatch record %
            >|*1
            >#dispatch
         % output the subsystem dispatch record %
            % word 1:
               RH = ptr to subsystem name string
               LH = ptr to start of COMMANDS %
                  (?IF top = 0 =*S2\1 /
                  top|LH =*S2\3)
            % word 2:
               RH = ptr to squeak rule / 0
               LH = ptr to help rule/0, %
                  header[ =helploc, =spekloc ]
            % word 3:
               RH = ptr to initialization rule /0
```

```
            LH = ptr to reentry rule/0, %
               header[ =rentryloc, =initloc ]
         % word 4:
            RH = ptr to termination rule/0
            LH = ptr to helprequest rule/0 %
               header[ =hlprqloc, =termloc ]
         % word 5:
            RH = ptr to process records/0
            LH = 0 %
               header[ =0, =prcr ];
   % command definitions %
      % rules and commands are the same, except that the parse tree for a
      command is linked onto an alternative list for a subsystem, and the
      alternative to a rule is NULL %
      command =                                                          1C5
         (( "COMMAND" .ID &LABEL/ _ .ID "COMMAND" &LABEL)
            :defcmd[1] &MARK
            *= exp &alt_sav  &suc_0  :eval[1] * &sav _ top               1C5B1
            &UNMARK *  *;
         / "INITIALIZATION"
            :[?IF initloc # 0 <"ERROR: multiple INITIALIZATIONs"> error2]
            rule &initloc _ top                                          1C5B2
         / "TERMINATION"
            :[?IF termloc # 0 <"ERROR: multiple TERMINATIONs"> error2]
            rule &termloc _ top                                          1C5B3
         / "HELPREQUEST"
            :[?IF hlprqloc # 0 <"ERROR: multiple HELPREQUESTs"> error2]
            rule &hlprqloc _ top                                         1C5B4
         / "HELP"
            :[?IF helploc # 0 <"ERROR: multiple HELPs"> error2]
            rule &helploc _ top                                          1C5B5
         / "SQUEAK"
            :[?IF sqekloc # 0 <"ERROR: multiple SQUEAKs"> error2]
            rule &sqekloc _ top                                          1C5B6
         / ("RENTRY" / "REENTRY" )
            :[?IF rentryloc # 0 <"ERROR: multiple REENTRYs"> error2]
            rule &rentryloc _ top ) ;                                    1C5B7
      rule =                                                             1C5
         .ID &LABEL
         *= exp &alt_0  &suc_0  :eval[1] *                               1C5C
          :defcmd[1] *  *; ;
      defcmd                                                             1C5
         [.ID]       =>
            [ ?@ defned *1 <"ERROR: Previously defined name used as a rule:
            " *1 LOC> &cmerr]
            rulname[*1];                                                 1C5D1
   % expression definition %
      exp = .#<*/>subexp :alter[$];                                      1C6
      subexp =                                                           1C6
         .#factor :succ[$];
      factor =                                                           1C6
         *( exp *)/                                                      1C6C
         *[ exp *] :option[1]/                                           1C6C
         term;
   % output production rules %
      % the principal production element is the unparse rule eval.  It
```

```
expects a node of 1 element:  an expression node
(alter/succ/option/terminal).  The global variables "alt" and "suc"
are used to identify the alternative and successor paths respectively.
%
% the entire parse tree for the expression is built, then it is
processed in reverse order (from "right to left") so that no fixups
are required to address any alternative or successor nodes. %
%--------%
eval                                                                1C7
   [alter] => % process a sequence of alternatives %
      ?*01 = 1 % single subnode %
         eval[*1:1]                                                 1C7D1A
      / % multiple subnodes %
         $*1_( % select all alternatives beginning with the last,
         invoking the eval rule recursively to process them.%
            ! suc ( eval[*$] )
            &alt _ top
            );
   [succ] => % process a sequence of successors %                   1C7D
      ?*01 = 1 % single subnode %
         eval[*1:1]                                                 1C7D2A
      / % multiple subnodes %
         $*1_( % select all successors beginning with the last,
         invoking the eval rule recursively to process them.%
            ((?LAST  eval[*$])
            / (! alt (&alt _ 0 eval[*$]) ))
            &suc _ top
            );
   [succargs] => % process a sequence of successors as arguments to a
   call %                                                           1C7D
      ?*01 = 1 % single subnode %
         (?IF [succargs[pushi[-]]] eval[push[*1:1:1]] /
         ! alt (&alt _ 0 eval[push[]]) &suc _ top eval[*1:1])
      / % multiple subnodes %
         $*1_( % select all successors beginning with the last,
         invoking the eval rule recursively to process them.%
            (?LAST lastarg[*$]
            / ntlastarg[*$])
            &suc _ top
            );
   [assign] => *1; % generate code for an assignment %
   [function] => *1; % generate code for a function %
   [testac] => *1; % generate code for a test%
   [option[-]] => % generate code for an optional expression %
      ! alt, suc ( &alt _ 0 &suc _ 0   eval[*1:1] )
      header[=suc, =suc]
      EXECUTE top|RH\1
      &top _ .-2;
   [showac] => *1; % generate code for a show%
   [funres] => % generate code for x function results %             1C7D
      ?*01 = 1 % single subnode %
         eval[*1:1]                                                 1C7D9A
      / % multiple subnodes %
         $*1( % results are loaded last result first so that result1
         will end in the accum.%
            ((?LAST  eval[*$])
```

```
                / (! alt (&alt _ 0 eval[*$]) ))
                &suc _ top
                );
        [ulistemanip[-,-]] => % list element with expression %
            !alt (&alt _ 0
               eval[*1:1] % list element inst %                          1C7D10A
               &suc _ top
               eval[ *1:2] % value for the list element - a CML exp %
                                                                         1C7D10A
               &suc _ top )
            eval[ succargs[ *1:1:2]] % the list element index - a CML exp %
                                                                         1C7D1C
            &suc _ top;
        [ulistemanip[-]] => % list element %
            !alt (&alt _ 0 eval[*1:1]) % list element inst %
            &suc _ top
            eval[*1:1:2] % the list element index - a CML exp %          1C7D11
            &suc _ top;
        [uloop[-]] => % generate code for a loop construct %              1C7D1
            ! alt, suc ( &alt _ 0 &suc _ 0   eval[*1:1] )
            header[=suc, =alt]
            LOOP top|RH\1
            &top _ .-2;
        [-] => % generate code for any terminal nodes. %
            &top _ .
            header[=suc, =alt]
            *1;
    lastarg                                                               1C7
        [pushi[-]] =>
            eval[push[*1:1]];
        [-] =>
            ! alt (&alt _ 0 eval[push[]]) &suc _ top eval[*1];
    ntlastarg                                                             1C7
        [pushi[-]] =>
            ! alt (&alt _ 0 eval[push[*1:1]]);
        [-] =>
            ! alt (&alt _ 0 eval[push[]] &suc _ top &alt _ 0 eval[*1]);
%---------
(header): header generates a one word header consisting of two pointers.
 The relocation bits are appropriately set to relocate either the right
or left half words independently %
    header                                                                1C9
        [0,0]       => 0\0;
        [0,-]       => 0|LH  *N2|RH\1;
        [-,0]       => *N1|LH  0|RH\2;
        [-,-]       => *N1|LH  *N2|RH\3;
% terminal type  productions %
    assign                                                                1C1C
        [-,-] =>
            !alt (&alt _ 0 eval[store[*1]])
            &suc _ top
            eval[ *2 ]                                                   1C10A1
            &suc _ top;
    apond                                                                 1C1C
        [.ID,-] =>
            !alt (&alt _ 0 eval[append[*1]])
```

```
        &suc _ top
        eval[ *2 ]                                                    1C1081
        &suc _ top;
    call                                                               1C10
      [.ID,-,0,0] =>
        CALL *N2|VAL *V1,;                                            1C10C1
      [.ID,-,0,1] =>
        CALLHELP *N2|VAL *V1,;                                        1C10C2
      [.ID,-,1,0] =>
        OCALL *N2|VAL *V1,;                                           1C10C3
      [.ID,-,1,1] =>
        OCLLRULE *N2|VAL *V1,;                                        1C10C4
    echo[.SR] =>                                                       1C10
      @S noddt *1                                                     1C10D
      ECHO =*S1,;                                                     1C10E
    xconfirm[] =>                                                      1C10
      CONFIRM,;
    answ[] =>                                                          1C10
      ANSWER,;
    opt[] =>                                                           1C10
      OPTION,;
    rpt[] =>                                                           1C10
      RPTOP,;
    enter                                                              1C10
      [-] =>
        ENTER =*N1,;
    entern [] =>                                                       1C10
      ENTERNULL,;                                                     1C10J
    entert [] =>                                                       1C10
      ENTERTRUE,;                                                     1C10K
    enterf [] =>                                                       1C10
      ENTERFALSE,;                                                    1C10L
    execute[-] =>                                                      1C10
      EXECUTE *V1,;
    fbclear[] =>                                                       1C10
      FBCLEAR,;
    statclear[] =>                                                     1C10
      CLEARSTAT,;
    push                                                               1C10
      [-] => % push variable %
        (?@ varname *1
          [ ?@ globalvar *1 2|VARTYPE / ?@ bltinvar *1 3|VARTYPE ]
        PUSHVAR *V1,);                                                1C10P1
      [] =>
        PUSHARG,;
    show                                                               1C10
      [0] => SHOW,;
      [1] => SHOWCON,;
      [2] => SHOWSTAT,;
    showac[-,-] =>                                                     1C10
      !alt(&alt _ 0 eval[show[*2]]) &suc _ top
      eval[*1] &suc _ top;                                            1C10F
    abort                                                              1C10
      [-] =>
        !alt(&alt _ 0 eval[abortmsg[]]) &suc _ top
        eval[*1] &suc _ top;                                          1C10S1
```

```
      [] =>
         ABORTOP ,;                                                    1C10S2
      abortmsg [] =>                                                     1C1(
         ABORTOP 1|VAL,;                                                 1C10
      resume                                                             1C1(
         [-] =>
            !alt(&alt _ 0 eval[rsmparam[]]) &suc _ top
            eval[*1] &suc _ top;                                         1C10U1
         [] =>
            RESUMNULL,;                                                  1C10U2
      rsmparam [] =>                                                     1C1(
         RESUME,;                                                        1C10\
      function                                                           1C1(
         [remote[helpcall[-,-],-],succargs] =>
            % generate the  call  %
               ( ?@ functname *1:1:1 eval[call[*1:1:1,=*Q2, *1:2, =1]]
                  [ ?& defned *1:1:2 ?AND ( ?NOT ?@ rulename *1:1:2)
                  <"ERROR: Illegal use of " *1:1:2 LOC LINE> &cmerr ]
                  @S rulename *1:1:2 *V1:1:2,                            1C10W1A1
               / <"ERROR: Undeclared function: " *1:1:1 LOC> &cmerr)
            &suc _ top
            !alt ( &alt _ 0 eval[*2])
            &suc _ top;
         [remote[-,-],succargs] =>
            % generate the call%
               ( ?@ functname *1:1 eval[call[*1:1,=*Q2, *1:2, =0]]
               / <"ERROR: Undeclared function" *1:1 LOC> &cmerr)
            &suc _ top
            !alt ( &alt _ 0 eval[*2])
            &suc _ top;
         [internal,succ] =>
            eval[*1]                                                     1C10W3
            &suc _ top
            !alt ( &alt _ 0 eval[*2])
            &suc _ top;
      prsfunction                                                        1C1(
         [.ID,succargs] =>
            pfcall[*1,=*Q2]                                              1C10X1
            &suc _ top
            !alt ( &alt _ 0 eval[*2])
            &suc _ top;
         [-] =>
            pfcall[*1]                                                   1C10X2
            &suc _ top;
      fefunction                                                         1C1(
         [.ID,succargs] =>
            ffcall[*1,=*Q2]                                              1C10Y1
            &suc _ top
            !alt ( &alt _ 0 eval[*2])
            &suc _ top;
         [-] =>
            ffcall[*1]                                                   1C10Y2
            &suc _ top;
      internal[-] =>                                                     1C1(
         *N1,;
      varkw [-,-] =>                                                     1C10/
```

```
      !alt (&alt _ 0 eval[xvkw[ ]])
      &suc _ top eval[ *1]
      &suc _ top;
   xvkw[ ] => VKEYOP, ;                                                1C10A
   xindof [-,-] =>                                                     1C10A
      !alt (&alt _ 0 eval[xindop[*2]])
      &suc _ top eval[ *1]
      &suc _ top;
   xindop[-] =>                                                        1C10A
      INDEXOF [ ?@ globalvar *1 2|VARTYPE / ?@ bltinvar *1 3|VARTYPE ]
      *V1, ;                                                           1C10AE
   keyw                                                                1C10A
      [.SR,1,-] =>
         @S litstr *1  % mark it as a literal %                        1C10AE1
         KEYOP1 (?@defned *1 *R1\1 / *V1,) *N3,;                       1C10AE1
      [.SR,0,-] =>
         @S litstr *1  % mark it as a literal %                        1C10AE2
         KEYOP (?@defned *1 *R1\1 / *V1,) *N3,;                        1C10AE2
   test                                                                1C10A
      [.ID,-] =>
         TEST *N2|VAL                                                  1C10AF1
            ( ?@ constant *1 =*V1, /
            [ ?@ globalvar *1 2|VARTYPE / ?@ bltinvar *1 3|VARTYPE ]
            *V1,);
      [.SR,-] => % builtin var%
         TEST *N2|VAL ?@ bltinvar *1 3|VARTYPE *V1,;                   1C10AF2
      [-,-] =>
         TEST *N2|VAL =*N1,;
      [-] =>
         TEST *N1|VAL,;
   testac                                                              1C10A
      [-,-,-] =>
         !alt(&alt _ 0 eval[test[*2,*3]]) &suc _ top
         eval[*1] &suc _ top;                                          1C10AG1
      [-,-] =>
         !alt(&alt _ 0 eval[test[*2]]) &suc _ top
         eval[*1] &suc _ top;                                          1C10AG2
   uptloop[-] =>                                                       1C10A
      RPTLOOP *N1,;                                                    1C10AH
   uexitloop[-] =>                                                     1C10A
      EXITLOOP *N1,;                                                   1C10A1
   uuntilloop[ ] =>                                                    1C10A
      UNTILLOOP,;                                                      1C10A.
   uwhileloop[ ] =>                                                    1C10A
      WHILELOOP,;                                                      1C10Ar
   perform % looping construct %                                       1C10A
      [.ID,-,-] =>
         % check to make sure .ID is a rule %
            [(?NOT ?@ defned *1 @S  rulename *1 @S reloca *1
            /?NOT ?@ rulename *1 <"ERROR: PERFORM must use a rule name"
            *1 LOC> &cmerr)]
         % generate code to (1) EXECUTE the rule and (2) evaluate the
         test expression.  Like all CML code, this sequence is generated
         in reverse order so that tree meta can do the appropriate
         linking %
            ! alt, suc ( &alt _ 0
```

```
            eval[ succ[ *2 ] ]                                          1C10AL1B1
            &suc _ top &alt _ . eval[ execute[*1] ] )
         &suc _ top;
   pfcall                                                                 1C10A
      [.ID,-] =>
         PFCALL *N2|VAL *V1,;                                            1C10AM1
      [-] =>
         PFCALL *V1,;                                                    1C10AM2
   ffcall                                                                 1C10A
      [.ID,-] =>
         FECALL *N2|VAL *V1,;                                            1C10AN1
      [-] =>
         FECALL *V1,;                                                    1C10AN2
   recho[.SR] =>                                                          1C10A
      @S noddt *1                                                         1C10AO
      RECHO =*S1,;                                                        1C10AO
   store[-] =>                                                            1C10A
      STORE [ ?@ globalvar *1 2|VARTYPE / ?@ bltinvar *1 3|VARTYPE ]
      *V1,;                                                            1C10AP
   append[.ID] =>                                                         1C10A
      APPEND [?@ globalvar *1 2|VARTYPE / ?@ bltinvar *1 3|VARTYPE ]
      *V1,;                                                            1C10AQ
   uinlste[.ID,-] =>                                                      1C10A
      (?@ varname *1 [ ?@ globalvar *1 2|VARTYPE / ?@ bltinvar *1
      3|VARTYPE ] INLSTE *V1,);
   udllste[.ID,-] =>                                                      1C10A
      (?@ varname *1 [ ?@ globalvar *1 2|VARTYPE / ?@ bltinvar *1
      3|VARTYPE ] DLLSTE *V1,);
   uldlste[.ID,-] =>                                                      1C10A
      (?@ varname *1 [ ?@ globalvar *1 2|VARTYPE / ?@ bltinvar *1
      3|VARTYPE ] LDLSTE *V1,);
   ustlste[.ID,-] =>                                                      1C10A
      (?@ varname *1 [ ?@ globalvar *1 2|VARTYPE / ?@ bltinvar *1
      3|VARTYPE ] STLSTE *V1,);
   valueof                                                                1C10A
      [.SR] =>
         @S litstr *1                                                     1C10AV1
         (?@defned *1 ENTERCW *R1\1 *H1, / ENTERCW *V1, 0,);
      [.SR,-] =>
         @S litstr *1                                                     1C10AV2
         (?@defned *1 ENTERCW *R1\1 / ENTERCW *V1,) *N2,;
   xentrstr[-] => ENTERSTR *V1,;                                          1C10A
   loadbi [-] =>                                                          1C10A
      @S extern *1 @S varname *1 @S bltinvar *1 load[*1];                  1C10AX
   loadres [-] =>                                                         1C10A
      ( ?IF *N1 = 1 loadbi[*'feresult']
      / ?IF *N1 = 2 loadbi[*'fe2result']
      / ?IF *N1 = 3 loadbi[*'fe3result']
      / ?IF *N1 = 4 loadbi[*'fe4result']
      / ?IF *N1 = 5 loadbi[*'fe5result']
      / ?IF *N1 = 6 loadbi[*'fe6result']
      / ?IF *N1 = 7 loadbi[*'fe7result']
      / ?IF *N1 = 8 loadbi[*'fe8result']);
% terminal nodes for compiler %
   term =                                                                 1C11
      (subname/ recognition/ feedback/ confirm/ conditional/ answer/
```

```
      optchar/ rptchar/ variablecw/ sloop/ showuser/ listemanip/
      abortcmd/ helpresume/ sizeof/ perfrm/ tparam):
   conditional =                                                    1C1
      "IF" &testtype _ TSTTRUE ["NOT" &testtype _ testtype + TSTNOT]
         !testtype ( param)
         (
            ( %binary relation%
               '= &testtype _ testtype + 1/                         1C11B1B
               ">=" &testtype _ testtype + 3B/
               "<=" &testtype _ testtype + 5B/
               "< " &testtype _ testtype + 4B/
               "> " &testtype _ testtype + 2B)
            (
               idornum                                              1C11B1B
               &testtype _ testtype - 40B :testac[2, = testtype]    1C11B1B
            /
               "NULL"
               &testtype _ testtype + TSTNULL - 40B :testac[1, =
               testtype])                                           1C11B1B
         /
            :testac[1, = testtype]);
   idornum =                                                        1C1
      ( .ID
         (?@ defned
            (?@ varname/
            ?@ constant &testtype _ testtype + TSTINT /
            <"ERROR: Illegal operand in IF: " *1 LOC> error2)
         / <"ERROR: Undeclared variable: " *1 LOC> @S varname @S defned
         &cmerr) /
      .NUM &testtype _ testtype + TSTINT/
      bivar1 /
      bivar2);
   showuser =                                                       1C1
      "SHOW" &value _ 0
         ["CONFIRM" &value _ 1 / "STATUS" &value _ 2]
         *( !value (param) :showac[1, = value] *);                  1C11D
   subname =                                                        1C1
      (.ID / bivar1) (
         (?@ functname % function invocation %
            (?@ parallel &ctlbits _ 1 / &ctlbits _ 0)
            [ *[ ["IN" "LINE" &ctlbits _ 0 / "OUT" "OF" "LINE" &ctlbits _
            1] [ .ID :helpcall[2] ] *] ] :remote[1, = ctlbits]
            *( .$<*,> ( varparam / param) :succargs[$] :function[2]
            [sfunres] *)                                            1C11E1
         )
         /(?@ pfunctname *( .$<*,> ( varparam / pfparam) :succargs[$]
         :prsfunction[2] [sfunres] *) ) % parsefunction invocation %
         /(?@ ffunctname *( .$<*,> ( varparam / pfparam) :succargs[$]
         :fefunction[2] [sfunres] *) ) % frontend function invocation %
         / (*_ % assignment statement %
            % check lhs variable type %
               ( ?@ varname / <"ERROR: Undeclared variable: " *1 LOC> @S
               varname @S defned &cmerr)
            param  :assign[2] )
         / (*:*_ %append statement %
            % check lhs variable type %
```

```
                ( ?@ varname / <"ERROR: Undeclared variable: " *1 LOC> @S
                varname @S defned &cmerr)
             param  :apond[2] )
          / :loadid[1]  )
       / bifun :fefunction[1];
       loadic                                                          1C11E
          [.ID] =>
              (?@ rulename *1
                @S reloca *1 execute[*1])                            1C11E3A1
             / (?@ varname *1
                load[*1])
             / (?@ constant *1 enter[=*V1])
             / (?@ strconst *1 xentrstr[*1])
             / (?NOT ?@ defned *1
                @S rulename *1 @S reloca *1                          1C11E3A5
                execute[*1])                                         1C11E3A5
             / <"ERROR: Illegal use for identifier: " *1 LOC LINE>;
       load                                                            1C11E
          [-] =>
             (?@ varname *1 [ ?@ globalvar *1 2|VARTYPE / ?@ bltinvar *1
             3|VARTYPE ] LOAD *V1,);
       loadaddr                                                        1C11
          [-] =>
             (?@ varname *1 [ ?@ globalvar *1 2|VARTYPE / ?@ bltinvar *1
             3|VARTYPE] LOADI *V1,);
    sfunres = "->" &value _ 1
       .#<*,>( :loadres[=value] (.ID / oivar1)
          % check lhs variable type %
             ( (?@ defned / ?@ bltinvar)
                ( ?@ varname
                /  <"ERROR: Illegal assign to variable" LOC> error2)
             / <"ERROR: Undeclared variable" LOC LINE> @S varname @S
             defned &cmerr)
          :store[1] &value _ value + 1 :succ[2])
       :funres[1] :succ[2];
    recognition = keyword/ builtinrec;                                 1C11
       keyword = .SR &ctlbits _ 1B % set level1 bit % (?@defned &value _
       *H1 / &value _ 0)                                               1C11(
          [ *! #qualifier *! ]
          :keyw[1, =ctlbits, =value];
       builtinrec =                                                    1C11(
          ( ("SSEL" &value _ SSEL/
             "DSEL" &value _ DSEL/
             "LSEL" &value _ LSEL)
             *( !value ( param)  :function[internal[=value],succ[1]] *) );
                                                                     1C11G2A
    listemanip =                                                       1C11
       "INSERT" % insert an element following the given element %
          *# .ID *#                                                    1C11H
             ( ?@ varname / <"ERROR: Undeclared variable " *1 LOC> @S
             varname &cmerr)
          *[ param *] :uinlste[2]                                      1C11H
          *_ param                                                     1C11H
          :ulistemanip[2]
       /"DELETE" % delete given element %
          *# .ID *#                                                    1C11H
```

```
            ( ?@ varname / <"ERROR: Undeclared variable " *1 LOC> @S
            varname &cmerr)
         *[ param *] :udllste[2]                                    1C11H2
         :ulistemanip[1]
      / _ *# .ID *# % load/store a list element %
         ( ?@ varname / <"ERROR: Undeclared variable " *1 LOC> @S varname
         &cmerr)
         *[ param *]                                                1C11H2
         ( *_ :ustlste[2] % store into list element %
            param
            :ulistemanip[2]
         / :uldlste[2] :ulistemanip[1] ); % load the list element %
variablecw =                                                        1C11
   ( "CW" / "COMMAND" [ "WORD" ] ) *: param
   &ctlbits _ 18 : varkw[1,=ctlbits];                               1C11
sindof = % syntax rule for INDEX OF x IN y %                        1C11
   "INDEX" "OF" param "IN" .ID
   % check variable type %
      ( ?@ varname / <"ERROR: Undeclared variable: " *2 LOC> @S
      varname @S cefned &cmerr)
   : xindof[2];
feedback =                                                          1C11
   *<                                                               1C11H
      (  "..." .SR *> :recho[1] /
         .SR       *> :echo[1] )  /
   "CLEAR"
      ("STATUS" :statclear[]
      / :fbclear[]);
confirm =                                                           1C11
   "CONFIRM"
   :xconfirm[] ; % call routine to get a command accept %
answer =                                                            1C11
   "ANSWER"
   :answ[] ; % call routine to process yes/no answer %
abortcmd =                                                          1C11
   "ABORT" (*(param*) :abort[1] / :abort[]);
helpresume =                                                        1C11
   "RESUME" (*( param *) :resume[1] / :resume[]);
optchar =                                                           1C11
   "OPTION"
   :opt[] ; % call routine to process option character %
rptchar =                                                           1C11
   "RPT" ["CHAR"]
   :rpt[] ; % get the REPEAT char from user %
sloop =                                                             1C11
   "LOOP" factor
      ( "UNTIL" factor :uuntilloop[] :succ[3]
      / "WHILE" factor :uwhileloop[] :succ[3]
      / :urptloop[=1] :succ[2] )
   :uloop[1]
   / "EXIT" "LOOP" (.NUM :uexitloop[1] / :uexitloop[=1])
   / "REPEAT" "LOOP" (.NUM :urptloop[1] / :urptloop[=1]);
perfrm =                                                            1C11
   "PERFORM" <"ERROR: PERFORM construct not implemented" LOC LINE>
   error2
      .ID
```

```
        "UNTIL"  &value _ 1
        *( exp *) :perform[2,=value];                                   1C11S1
qualifier =                                                             1C11
    ("L2" &ctlbits _ 0 % clear level1 bit % /
    .NUM
        [?IF *N1 > 177777B <"ERROR: Command word token greater than
        177777B" LOC> &cmerr]
        &value _ *N1 &DISCARD);
varparam = _ .ID  ?& varname :pushi[1];                                 1C11
param =                                                                 1C11
    tparam / factor;
pfparam =                                                               1C11
    param / addrparam;                                                  1C11
addrparam = %pass grammar variable address to parsefunction%           1C11
    *$ (.ID / bivar1 / bivar2)                                          1C11
    :loadaddr[1];
tparam =                                                                1C11
    "VALUEOF" *( .SR
        ( *! .NUM *! [?IF *N2 > 177777B <"ERROR: Command word " *1 " has
        number greater than 177777B"> &cmerr] :valueof[2] / :valueof[1]
        *) )
    / _ *% .SR
        ( *! .NUM *! [?IF *N2 > 177777B <"ERROR: Command word " *1 " has
        number greater than 177777B"> &cmerr] :valueof[2] / :valueof[1])
    / .NUM :enter[1]
        %should allow negative numbers also%
    / "NULL" :entern[]
    / "TRUE" :entert[]
    / "FALSE" :enterf[]
    / (bivar1 / bivar2) : loadbi[1];
bivar1 = %assignable builtin vars%                                      1C11
    ("TERMINATORS" : "fetrminators"
    )
    &S varname &S bltinvar;                                             1C11
bivar2 = %non-assignable builtin vars%                                  1C11
    ("TERMCHAR" : "fetrmch"
    / "DISPLAY" :"fedisplay"
    / "TYPEWRITER" :"fetypwriter"
    / "HELPCODE" :"fehelpcode"
    / "RESULT2" :"fe2result"
    / "RESULT3" :"fe3result"
    / "RESULT4" :"fe4result"
    / "RESULT5" :"fe5result"
    / "RESULT6" :"fe6result"
    / "RESULT7" :"fe7result"
    / "RESULT8" :"fe8result"
    / ("RESULT1"/"RESULT") :"feresult"
    / "HELPSYNTAX" :"fehlpsyn"
    / "BREAKPOINT" :"febreakpoint"
    / "LINEATATIME" :"felineatatime"
    / "HALFDUPLEX" :"fehalfduplex"
    )
    &S varname &S bltinvar;                                             1C11AA
bifun = % builtin functions %                                           1C11
    "WINDOW" :"fewindow"
    &S ffunctname;                                                      1C11AB
```

```
      %ERROR RULES%
         error2 = &errtype _ 2 &xerror;                                    1C12
   END of CML                                                                1
SYMBOLS % branch only output to sequential < fe, fevarsym.txt,> %
   feresult = 1, %RESULT1/RESULT%                                            2
   fe2result = 2, %RESULT2%                                                  2
   fe3result = 3, %RESULT3%                                                  2
   fe4result = 4, %RESULT4%                                                  2
   fe5result = 5, %RESULT5%                                                  2
   fe6result = 6, %RESULT6%                                                  2
   fe7result = 7, %RESULT7%                                                  2
   fe8result = 8, %RESULT8%                                                  2
   febreakpoint = 9, %BREAKPOINT%                                            2
   fecurrenttool = 10, %CURRENTTOOL%                                         2
   fehalfduplex = 11, %HALFDUPLEX%                                           2
   fehelpcode = 12, %HELPCODE%                                               2
   felineatatime = 14, %LINEATATIME%                                         2
   fetypwriter = 15, %TYPEWRITER%                                            2
   fedisplay = 16, %DISPLAY%                                                 2
   fenode = 17, %NODE%                                                       2
   feproject = 18, %PROJECT%                                                 2
   feuserid = 19, %USERID%                                                   2
   fetools = 20, %TOOLS%                                                     2
   feactivetools = 21, %ACTIVETOOLS%                                         2
   fetrmch = 22, %TERMCHAR%
   fetrminators = 23; %TERMINATORS%
   FINISH
(doc)
   TODO                                                                      3
      ANYOF expressions                                                      3A
         %ANY NUMBER OF%
            ANYOF= 3789 %zero to infinite occurrences of this expression%
         % generate code for a '$ expression %
            [anyof[-]]  =>
               ! alt, suc ( &alt _ 0 &suc _ 0    eval[*1:1] )
               header[=suc, =suc]
               ANYOF top|RH\1                                          3A15
               &top _ .-2;
         DUMMY: anyof;                                                    3A
      List element manipulation:                                          3A
         allow list#index# as lhs, rhs, bin-rel rhs                       3A2
         produce (loadelem/storeelem) addr= $list, val = index            3A2
      List element description:                                           3A
         This describes a facility for manipulating list elements in NLS10 CML.
          Functionally it is a subset of the L10 list package.  Comments
         requested.                                                       3A3
         The syntax and semantics for appending to a CML list will remain
         unchanged.  The syntax for a CML list element expression is      3A3
            CML-list-name#CML-index-expression#                           3A3E
         The CML-list-name must be an identifier declared to be a variable.
         The CML-index-expression may be an arbitrary CML expression whose
         value is a number.  Its semantics are as follows.                3A3
         When a CML list element expression is found on the left side of an
         assignment, the value of the element is replaced by the value of the
         expression on the right side of the assignment.  For example.    3A3
            listvar#2# _ 5                                                3A3E
```

If the CML variable is currently not a list, it will be made into an EMPTY list before the assignment. If it is a list with less elements than indicated by the index expression, NULL elements will be added to make up the difference. For example. 3A3

listvar _ NULL listvar#1# _ 5 results in a list with one element whose value is 5 3A3E

listvar _ NULL listvar#1# _ 5 listvar #3# _ #"A" results in a list of three elements: the first element has value 5, the second NULL and the third the string "A" 3A3E

When the CML list element expression is found in any other context, its semantics are to generate the value of that list element. If the list variable is not currently a list or if it does not currently have the element indicated by the index, ie. the list is too short, a runtime error will occur. 3A3

It is possible to simplify the implementation by allowing the index for a CML list element expression on the left side of an assignment to be only a constant, ie. a literal number or a constant declared to be a literal number. However, I think that this would be unnecessarily restrictive. 3A3

N. B. This suggested implementation does not allow several common features of the L10 list package. In particular, it does not allow the removal of one or more elements from a list to make it shorter. 3A3

make builtin variable, selection fns, and fefns indices so wont have to link load grammar at load time but can compact it and suck it in. 3A

REWRITE EVAL USING GEN LABELS

$(>#1 ... G#1, ..)

OUT OF LINE EXECUTION

add inst to load spclargs[chvar] and spclargs[parcallrule]. add syntax for these and such that chvar gets result of call when return finally happens. can test chvar in the meantime for FALSE. add a WAIT UNTIL chvar construct.

SYNTAX

add DECLARE SYNTAX .ID = .SR,...

INPUT FROM, OUTPUT TO,

DECLARE TERMINAL CLASS FUNCTION

Syms-L10

```
SYMBOLS %FILE%
% This file contains initializing symbols for L10 and L1011 compilers.  It
should only contain CONSTANT and ADDRESS statements. %
% signal type names %
   (helptype)  CONSTANT = 1; % HELP %                                        3A
   (notetype)  CONSTANT = 2; % NOTE %                                        3E
   (aborttype) CONSTANT = 3; % ABORT %                                       3C
% world-wide signal names %
   % System signal names are greater than 10000 octal.  Program-wide signal
   names should be less than 10000 ocatl %
   (nohelp)          CONSTANT = 10000B; % on resume, means no help obtained %  4E
      % happens when HELP is not caught %
   (return)          CONSTANT = 10001B; % NOTE: procedure returning %          4C
      % Whenever a procedure or its coroutine did an INVOKE, and then does
      RETURN %
   (unwind)          CONSTANT = 10002B; % NOTE, this routine will vanish %     4D
      % happens when higher routine does TERMINATE %
      % also when runtime package does a recover %
   (gothelp)         CONSTANT = 10003B; % on resume, means help obtained %     4E
      % Should be standard first argument for RESUME %
   (stringoverflow)CONSTANT = 10004B; % HELP, string overflowed %             4F
      % low level string handlers generate this %
      % arg2 is string address %
      % they expect resume(gothelp,new-string-address) %
      % IF program not able to provice new string, treat as ABORT %
   (changestring)  CONSTANT = 10005B; % NOTE, arg2=old addr, arg3=new addr % 4G
      % NOTE issued by low level string handlers after they got a new string
      address %
   (saroverflow)   CONSTANT = 10006B; % ABORT, SAR string overflow %          4H
      % generated by runtime routines when SAR overflows %
      % perhaps later treated like other strings %
   (stkoverflow)   CONSTANT = 10007B; % ABORT, program defined stack overflow
   %                                                                          4I
   (stkunderflow)  CONSTANT = 10010B; % ABORT, program defined stack underflow
   %                                                                          4J
   (listspace)     CONSTANT = 10011B; % ABORT, list allocation zone full %   4K
% error type codes (from (nls,xl10runtime,sysrcv) ) %
   % error type codes are greater than 20000 octal %
   % this is the first argument passed to recover procedure and indicates the
   type of error involved %
   (programbug)    CONSTANT = 20001B; % a program bug involved %             5G
   (uncaughtabort) CONSTANT = 20002B; % uncaught ABORT %                     5I
   % stkoverflow=general stack overflow -- use same symbol defined above %
% LIST dscriptor type codes %
   (lnull)  CONSTANT = 0; % NULL %                                           6A
   (linteg) CONSTANT = 2; % an intteger %                                    6B
   (lstrin) CONSTANT = 3; % a string %                                       6C
   (llist)  CONSTANT = 4; % a list %                                         6D
   (lblock) CONSTANT = 1; % a 'getblk' type block of storage %               6E
FINISH
```

Sys Gd - Arc Ldr.

< PROGSUP, SYSGD-ARCLDR.NLS;50, >, 10-Aug-79 11:50 BLP ;;;;

FORMAL PARAMTERS(chain, a)

(fixuplh) <compsrc, arcldr, 0574> PROCEDURE 5E
FORMAL PARAMTERS(chain, a)

(getjfn) <compsrc, arcldr, 0192> PROCEDURE 3I

(glblok) <compsrc, arcldr, 0442> PROCEDURE 4F
FORMAL PARAMTERS(s, a)

(gothelp) <compsrc, syms-l10, 015> CONSTANT =10003B 4E
% on resume, means help obtained %

(gtvlu) <compsrc, arcldr, 0263> PROCEDURE 3K

(hashsize) <compsrc, arcldr, 070> EXT CONSTANT =359 2D1

(hashtabaddr) <compsrc, arcldr, 0931> EXT 2A4

(helptype) <compsrc, syms-l10, 06> CONSTANT =1 .3A
% HELP %

(henter) <compsrc, arcldr, 0343> PROCEDURE 4B
FORMAL PARAMTERS(s, a, type)

(InitILdr) <compsrc, arcldr, 0918> PROCEDURE 3B
FORMAL PARAMTERS(newcorpg)
% initialize "internal" loader %

(l10set) <compsrc, rt-minimain, 044> LOCAL 7C6A
% local label %

(l10start) <compsrc, rt-minimain, 039> LOCAL 7C5A

(lblock) <compsrc, syms-l10, 053> CONSTANT =1 6E
% a 'getblk' type block of storage %

(ldstblk) <compsrc, arcldr, 0619> PROCEDURE 5I

(ldwerr) <compsrc, arcldr, 0729> PROCEDURE 6E
FORMAL PARAMTERS(s)

(linit) <compsrc, arcldr, 089> PROCEDURE 3A

(linker) <compsrc, arcldr, 0864> PROCEDURE 4D
FORMAL PARAMTERS(s, a)
% link address a to symbol s %

(linteg) <compsrc, syms-l10, 050> CONSTANT =2 6B
% an intteger %

(listspace) <compsrc, syms-l10, 054> CONSTANT =10011B 4K
% ABORT, list allocation zone full %

(llist) <compsrc, syms-l10, 052> CONSTANT =4 6D
% a list %

```
(lnull)         <compsrc, syms-l10, 049>          CONSTANT =0           6A
   % NULL %

(loadfl)        <compsrc, arcldr, 0498>           PROCEDURE             5A

(lstrin)        <compsrc, syms-l10, 051>          CONSTANT =3           6C
   % a string %

(NLSArcLdr)     <compsrc, arcldr, 01031>          PROCEDURE             7B
   FORMAL PARAMTERS( njfn, eCode, ubCode, bSym, eSym, lbSym, statusHash, bHash,
eHash, lbHash, outstr REF)
  % CL: ;  load NLS user program into program buffer %

(nohelp)        <compsrc, syms-l10, 012>          CONSTANT =10000B      4B
   % on resume, means no help obtained %

(notetype)      <compsrc, syms-l10, 07>           CONSTANT =2           3B
   % NOTE %

(ownSymLoc)     <compsrc, arcldr, 0989>           EXT                   2A1

(pcall)         <compsrc, rt-minimain, 089>       LOCAL                 7C8D
   % port-call code (trace point) %

(pgname)        <compsrc, arcldr, 0610>           PROCEDURE             5H

(printm)        <compsrc, arcldr, 0667>           PROCEDURE             6C

(printud)       <compsrc, arcldr, 0682>           PROCEDURE             6D
   % list undefined symbols %

(programbug)    <compsrc, syms-l10, 044>          CONSTANT =20001B      5C
   % a program bug involved %

(r3bits)        <compsrc, arcldr, 01020>          FIELD - 6             2E2C

(readsw)        <compsrc, arcldr, 0200>           PROCEDURE             3J

(rec2)          <compsrc, arcldr, 086>            RECORD                2E1

(rec3)          <compsrc, arcldr, 087>            RECORD                2E2

(reloc)         <compsrc, arcldr, 01297>          LOCAL                 2C2

(return)        <compsrc, syms-l10, 013>          CONSTANT =10001B      4C
   % NOTE: procedure returning %

(saroverflow)   <compsrc, syms-l10, 034>          CONSTANT =10006B      4H
   % ABORT, SAR string overflow %

(skip)          <compsrc, arcldr, 0598>           PROCEDURE             5G

(stable)        <compsrc, arcldr, 062>            EXT                   2A3

(start1)        <compsrc, arcldr, 0124>           PROCEDURE             3C
```

```
(start2)        <compsrc, arcldr, 0127>          PROCEDURE              3D

(status)        <compsrc, rt-minimain, 0113>     LOCAL                  7D1A
   % TRUE if recovering %

(stkoverflow)   <compsrc, syms-l10, 037>         CONSTANT =10007B       4I
   % ABORT, program defined stack overflow %

(stkunderflow) <compsrc, syms-l10, 038>          CONSTANT =10010B       4J
   % ABORT, program defined stack underflow %

(stringoverflow) <compsrc, syms-l10, 016>        CONSTANT =10004B       4F
   % HELP, string overflowed %

(symbl)         <compsrc, arcldr, 01025>         FIELD - 32             2E1A

(syms)          <compsrc, arcldr, 0541>          PROCEDURE              5C

(syscent)       <compsrc, rt-minimain, 084>      LOCAL                  7C8C
   % port entry (trace point) %

(syscer)        <compsrc, rt-minimain, 0102>     LOCAL                  7C9B
   % coroutine called as procedure %

(sysent)        <compsrc, rt-minimain, 075>      LOCAL                  7C8A
   % procedure entry point (trace point) %

(sysnxp)        <compsrc, rt-minimain, 0100>     LOCAL                  7C9A
   % non-existant port called %

(sysovr)        <compsrc, rt-minimain, 081>      LOCAL                  7C8B
   % stack overflow (follows sysent) %

(sysrcv)        <compsrc, rt-minimain, 018>      LOCAL                  7C

(sysrtf)        <compsrc, rt-minimain, 068>      LOCAL                  7C7C
   % return false %

(sysrtn)        <compsrc, rt-minimain, 063>      LOCAL                  7C7B
   % return, value in mreg %

(sysrtt)        <compsrc, rt-minimain, 061>      LOCAL                  7C7A
   % return true %

(syssys)        <compsrc, rt-minimain, 0111>     LOCAL                  7D

(sysufl)        <compsrc, rt-minimain, 0108>     LOCAL                  7C9C
   % stack underflow (strange) %

(udef)          <compsrc, arcldr, 0403>          PROCEDURE              4E
   FORMAL PARAMTERS( s, a)

(uncaughtabort) <compsrc, syms-l10, 045>         CONSTANT =20002B       5D
   % uncaught ABORT %
```

```
(undefinecmask) <compsrc, arcldr, 0999>          CONSTANT =76543286    2D2

(undefinedtype) <compsrc, arcldr, 0997>          CONSTANT =2           2D4

(unwinc)        <compsrc, syms-l10, 014>         CONSTANT =100028      4D
   % NOTE, this routine will vanish %

(wn)            <compsrc, arcldr, 0734>          PROCEDURE             6F
   FORMAL PARAMTERS( n)

(ws)            <compsrc, arcldr, 0738>          PROCEDURE             6G
   FORMAL PARAMTERS( s)

(wsym)          <compsrc, arcldr, 0742>          PROCEDURE             6H
   FORMAL PARAMTERS( s, a)

(xbinwd)        <compsrc, arcldr, 0635>          PROCEDURE             5K

(xlook)         <compsrc, arcldr, 0306>          PROCEDURE             4A
   FORMAL PARAMTERS( s)
   % => symPtr, hashPtr/-1 %

(xreloc)        <compsrc, arcldr, 0645>          PROCEDURE             5L
   % user JSYS reloc to read next word and relocate it per rlcwd %

(xxx)           <compsrc, arcldr, 01018>         FIELD - 24            2E2A
```

< PROGSUP, SYSGD-ARCLDR.NLS;50, >, 10-Aug-79 11:50 BLP ;;;;

FORMAL PARAMTERS(chain, a)

(fixuplh) <compsrc, arcldr, 0574> PROCEDURE 5E
FORMAL PARAMTERS(chain, a)

(getjfn) <compsrc, arcldr, 0192> PROCEDURE 3I

(glblok) <compsrc, arcldr, 0442> PROCEDURE 4F
FORMAL PARAMTERS(s, a)

(gothelp) <compsrc, syms-l10, 015> CONSTANT =10003B 4E
% on resume, means help obtained %

(gtvlu) <compsrc, arcldr, 0263> PROCEDURE 3K

(hashsize) <compsrc, arcldr, 070> EXT CONSTANT =359 2D1

(hashtabaddr) <compsrc, arcldr, 0931> EXT 2A4

(helptype) <compsrc, syms-l10, 06> CONSTANT =1 3A
% HELP %

(henter) <compsrc, arcldr, 0343> PROCEDURE 4B
FORMAL PARAMTERS(s, a, type)

(InitILdr) <compsrc, arcldr, 0918> PROCEDURE 3B
FORMAL PARAMTERS(newcorpg)
% initialize "internal" loader %

(l10set) <compsrc, rt-minimain, 044> LOCAL 7C6A
% local label %

(l10start) <compsrc, rt-minimain, 039> LOCAL 7C5A

(lblock) <compsrc, syms-l10, 053> CONSTANT =1 6E
% a 'getblk' type block of storage %

(ldstblk) <compsrc, arcldr, 0619> PROCEDURE 5I

(ldwerr) <compsrc, arcldr, 0729> PROCEDURE 6E
FORMAL PARAMTERS(s)

(linit) <compsrc, arcldr, 089> PROCEDURE 3A

(linker) <compsrc, arcldr, 0864> PROCEDURE 4D
FORMAL PARAMTERS(s, a)
% link address a to symbol s %

(linteg) <compsrc, syms-l10, 050> CONSTANT =2 6B
% an intteger %

(listspace) <compsrc, syms-l10, 054> CONSTANT =10011B 4K
% ABORT, list allocation zone full %

(llist) <compsrc, syms-l10, 052> CONSTANT =4 6D
% a list %

```
(lnull)          <compsrc, syms-l10, 049>          CONSTANT =0            6A
   % NULL %

(loadfl)         <compsrc, arcldr, 0498>           PROCEDURE              5A

(lstrin)         <compsrc, syms-l10, 051>          CONSTANT =3            6C
   % a string %

(NLSArcLdr)      <compsrc, arcldr, 01031>          PROCEDURE              7B
   FORMAL PARAMTERS( njfn, eCode, ubCode, bSym, eSym, lbSym, statusHash, bHash,
eHash, lbHash, outstr REF)
   % CL: ;  load NLS user program into program buffer %

(nohelp)         <compsrc, syms-l10, 012>          CONSTANT =10000B       4B
   % on resume, means no help obtained %

(notetype)       <compsrc, syms-l10, 07>           CONSTANT =2            3B
   % NOTE %

(ownSymLoc)      <compsrc, arcldr, 0989>           EXT                    2A1

(pcall)          <compsrc, rt-minimain, 089>       LOCAL                  7C8D
   % port-call code (trace point) %

(pgname)         <compsrc, arcldr, 0610>           PROCEDURE              5H

(printm)         <compsrc, arcldr, 0667>           PROCEDURE              6C

(printud)        <compsrc, arcldr, 0682>           PROCEDURE              6D
   % list undefined symbols %

(programbug)     <compsrc, syms-l10, 044>          CONSTANT =20001B       5C
   % a program bug involved %

(r3bits)         <compsrc, arcldr, 01020>          FIELD - 6              2E2C

(readsw)         <compsrc, arcldr, 0200>           PROCEDURE              3J

(rec2)           <compsrc, arcldr, 086>            RECORD                 2E1

(rec3)           <compsrc, arcldr, 087>            RECORD                 2E2

(reloc)          <compsrc, arcldr, 01297>          LOCAL                  2C2

(return)         <compsrc, syms-l10, 013>          CONSTANT =10001B       4C
   % NOTE: procedure returning %

(saroverflow)    <compsrc, syms-l10, 034>          CONSTANT =10006B       4H
   % ABORT, SAR string overflow %

(skip)           <compsrc, arcldr, 0598>           PROCEDURE              5G

(stable)         <compsrc, arcldr, 062>            EXT                    2A3

(start1)         <compsrc, arcldr, 0124>           PROCEDURE              3C
```

```
(start2)        <compsrc, arcldr, 0127>          PROCEDURE              3D

(status)        <compsrc, rt-minimain, 0113>     LOCAL                  7D1A
   % TRUE if recovering %

(stkoverflow)   <compsrc, syms-l10, 037>         CONSTANT =10007B       4I
   % ABORT, program defined stack overflow %

(stkunderflow) <compsrc, syms-l10, 038>          CONSTANT =10010B       4J
   % ABORT, program defined stack underflow %

(stringoverflow) <compsrc, syms-l10, 016>        CONSTANT =10004B       4F
   % HELP, string overflowed %

(symbl)         <compsrc, arcldr, 01025>         FIELD - 32             2E1A

(syms)          <compsrc, arcldr, 0541>          PROCEDURE              5C

(syscent)       <compsrc, rt-minimain, 084>      LOCAL                  7C8C
   % port entry (trace point) %

(syscer)        <compsrc, rt-minimain, 0102>     LOCAL                  7C9B
   % coroutine called as procedure %

(sysent)        <compsrc, rt-minimain, 075>      LOCAL                  7C8A
   % procedure entry point (trace point) %

(sysnxp)        <compsrc, rt-minimain, 0100>     LOCAL                  7C9A
   % non-existant port called %

(sysovr)        <compsrc, rt-minimain, 081>      LOCAL                  7C8B
   % stack overflow (follows sysent) %

(sysrcv)        <compsrc, rt-minimain, 018>      LOCAL                  7C

(sysrtf)        <compsrc, rt-minimain, 068>      LOCAL                  7C7C
   % returr false %

(sysrtn)        <compsrc, rt-minimain, 063>      LOCAL                  7C7B
   % return, value in mreg %

(sysrtt)        <compsrc, rt-minimain, 061>      LOCAL                  7C7A
   % return true %

(syssys)        <compsrc, rt-minimain, 0111>     LOCAL                  7D

(sysufl)        <compsrc, rt-minimain, 0108>     LOCAL                  7C9C
   % stack underflow (strange) %

(udef)          <compsrc, arcldr, 0403>          PROCEDURE              4E
   FORMAL PARAMTERS( s, a)

(uncaughtabort) <compsrc, syms-l10, 045>         CONSTANT =20002B       5D
   % uncaught ABORT %
```

```
(undefinedmask) <compsrc, arcldr, 0999>          CONSTANT =76543286   2D2

(undefinectype) <compsrc, arcldr, 0997>          CONSTANT =2          2D4

(unwind)        <compsrc, syms-l10, 014>         CONSTANT =10002B     4D
   % NOTE, this routine will vanish %

(wn)            <compsrc, arcldr, 0734>          PROCEDURE            6F
   FORMAL PARAMTERS( n)

(ws)            <compsrc, arcldr, 0738>          PROCEDURE            6G
   FORMAL PARAMTERS( s)

(wsym)          <compsrc, arcldr, 0742>          PROCEDURE            6H
   FORMAL PARAMTERS( s, a)

(xbinwd)        <compsrc, arcldr, 0635>          PROCEDURE            5K

(xlook)         <compsrc, arcldr, 0306>          PROCEDURE            4A
   FORMAL PARAMTERS( s)
   % => symPtr, hashPtr/-1 %

(xreloc)        <compsrc, arcldr, 0645>          PROCEDURE            5L
   % user uSYS reloc to read next word and relocate it per rlcwd %

(xxx)           <compsrc, arcldr, 01018>         FIELD - 24           2E2A
```

ARCLDR

```
FILE ArcLdr
   ALLOW!
% DECLARATIONS %
   % The following variables are set in Load-ArcLdr.run %
      (ownSymLoc) EXTERNAL;                                                    2A1
         % copy of symloc after symbols BLTed; for use with DAD or DDT %
      (corpg) EXTERNAL;                                                        2A2
         % Input file is pmaped in page [corpg]. %
      (stable) EXTERNAL;                                                       2A3
         % Symbol table is built from [stable] downward and may be relocated
         using the /<address>s switch. %
      (hashtabaddr) EXTERNAL;                                                  2A4
         % Hash table is built from [hashtabaddr] downward and may be
         relocated using the /<address>h switch. %
      % The /N switch makes a new symbol table and hash table in core right
      below last new one. stpbot contains address of lowest hash table. %
   % misc % DECLARE EXTERNAL
      moveSymsFlag_TRUE, blktyp, blksz, nlsflag_FALSE, nlslimit, noht_FALSE,
      storeHash_TRUE, lc, rlc, stp, stpst, stpbot,
      sssflg_FALSE, mflg_FALSE, globfg_TRUE, pname, ploc, psym, ulink_0,
      string[10], sp, altfg, udfnum_0,
      initlptr, filpg, word, sr2, sr3, phash REF, hashbot, hashtop,
      errorn_0, loadub_777777B, symlb, hashlb_0, wdnum, jfn,
      iodesignator=101B, maxout_377777777777B, newline=(2000002B, 64240B6),
      ary[6], csprel="REL", jfntab=(1B11, 377777377777B, 0,0,0,$dsprel,0,0);
   % user JSYS words - must be >= 1000B %
      (binwd);                                                                 2C1
      (reloc);                                                                 2C2
   % hash table %
      (hashsize) EXTERNAL CONSTANT = 359;                                      2D1
         % HASHED SYMBOL TABLE:  a hash table of length hashsize is based at
         location phash.  Each element of the hash table is a pointer to a
         chained list of single word hash table ertries for all symbols on the
         same hash chain.  Each hash table entry (1 word) contains two
         pointers.  The LH points to the symbol table entry (in the DDT table
         or in the undefined table) and the RH is the pointer to the next
         entry on the hash chain.  Hashbot is a low water mark for allocating
         space for hash table entries and undefnec symbol table entries.  The
         space used by undefined symbol table entries is recovered for reuse
         by linking on a chain beginning with ulink %
      (undefinedmask) CONSTANT = 765432B6;                                     2D1
      (definectype) CONSTANT = 1;                                              2D1
      (undefinedtype) CONSTANT = 2;                                            2D1
         % undefinedmask appears in the LH of symbols that are undefined (in
         the dot symbol table) and is changed when they are defined -- it is
         unlikely that it will ever appear in the LH of a good defined symbol
         %
   % RECORDS %
      (rec2) RECORD                                                            2E
         symbl[32],
         codbit[4];
      (rec3) RECORD                                                            2E
         xxx[24],
         byte[6],
         r3bits[6];
   ADDRESS
```

```
   symloc  = 116B,  % symbol table pointer %
   codeLoc = 120B;  % loaded code pointer %
REGISTER
   a2=11, a1=10, a3=12, r3=3, r4=4, a4=13, r1=1, r2=2, r5=5, r6=6,
   pointr=A, rlcwd=7;
      % pointr points into input page, rlcwd contains relocation bits for a
      block.  These are kept in registers throughout loading of a file %
      % registers zero through six are occationally used for executing code
      - see xlook, fixup, udef %
EXTERNAL sysovr;
% INITIALIZATION and PARSING ROUTINES %
(linit) PROCEDURE;                                                    3A
   % initialize here before ssave of ArcLdr %
      binwd.LH _ $a1; binwd.RH _ $binloc; % setup user JSYS %
      reloc.LH _ $a2; reloc.RH _ $relocl;
      initlptr.LH _ -1000B; initlptr.RH _ corpg*1000B;
      r1 _ 4B5: r2 _ 2B6+$envect;
      !JSYS sevec;
      !haltf;
   (start): % starting location %                                     3A2
      !JSYS reset; r1 _ 4B5; !JSYS clzff;
      r1 _ 100B; !JSYS rfcoc; r2 _ r2 .A 6377777777777B; %|a%
      r3 _ r3 .A 7774777777777B; %|w% !JSYS sfccc;
      startup _ $start1;
      GOTO l10start;
   (rstart): % reenter loc %                                          3A3
      startup _ $start2;
      GOTO l10start;
   (envect): % entry vector %                                         3A4
      GOTO start; GOTO rstart;
   END.
(InitILdr) PROCEDURE (newcorpg); % initialize "internal" loader %     3E
   % If a .sav file has ArcLdr.rel as one of its modules, this procedure
   should be called to initialize this "internal" version of the loader.
      Note that the hash table is NOT initialized here. %
   % setup user JSYSs %
      IF $binloc < 1000B OR $relocl < 1000B THEN
         ABORT (pSavErr, $"USER JSYSs WON'T WORK");
      binwd.LH _ $a1; binwd.RH _ $binloc;
      reloc.LH _ $a2; reloc.RH _ $relocl;
   IF newcorpg # 0 THEN corpg _ newcorpg;
   initlptr.LH _ -1000B; initlptr.RH _ corpg*1000B;
   nlslimit _ symloc.RH+1;
   RETURN;
   END.
(start1) PROCEDURE;                                                   3(
   cmndparse(TRUE);
   END.
(start2) PROCEDURE;                                                   3(
   cmndparse(FALSE);
   END.
(cmndparse) PROCEDURE(startf);                                        3(
   % cmndparse parser %
   IF startf THEN % initial startup %
      BEGIN
      stp _ stpst _ stable;
```

```
      stpbot _ stpst +1; % stpbot should be same as phash %
      % initialize hash table %
         hashtop _ hashtabaddr;
         &phash _ hashtop-hashsize;
         hashbot _ &phash-1;
         FOR r1_ 0 UP UNTIL >= hashsize DO phash[r1] _ 0;
      rlc _ 1408;
      loaqub _ 777777B;
      errorn _ 0;
      END;
   altfg _ FALSE;
   DO
      BEGIN
      r1 _ "*; !JSYS pbout;
      collsr(); % collect string from user %
      CASE |sp OF
         ="/: readsw();
         =37B: NULL;
         =0: NULL;
         ENDCASE
            BEGIN
            getjfn();
            loadfl()
            END;
      END
   UNTIL altfg;
   done();
   nlslimit _ symloc.RH+1; %save limit away for NLSLOADER %
   cr(); !JSYS haltf;
   END.
(collsr) PROCEDURE;                                                    3F
   % collect string from user - term. on cr, ca, alt %
   LOCAL x;
   sp _ $string .V 440708;
   LOOP CASE (x_binchr()) OF
      =CA: BEGIN cr(); EXIT END;
      =37B: EXIT;
      =12B: EXIT;
      =15B: NULL;
      =ALT: BEGIN altfg _ TRUE; cr(); EXIT END;
      =1: %|A% sp_bakstr(sp,$string);
      =27B: %|W% BEGIN cr(); GOTO rstart END;
      ENDCASE |sp_x;
   |sp _ 0; sp _ $string .V 440708;
   RETURN
   END.
(bakstr) PROCEDURE(sptr,a);                                            3(
   % backup sptr one char and echo to tty %
   % for 7 bit chars only %
   IF sptr= a .V 440708 OR sptr.RH<a THEN BEGIN
      r1 _ 7 %bell%; !JSYS pbout END
   ELSE BEGIN
      r1 _ "\; !JSYS pbout; !LDB 1,sptr; !JSYS pbout;
      IF sptr.r3bits<29 THEN BEGIN
         sptr.r3bits _ sptr.r3bits + sptr.byte END
      ELSE BEGIN
```

```
        sptr _ sptr-1; sptr.r3bits _ 1 END
      END;
   RETURN(sptr)
   END.
(pinchr) PROCEDURE;                                                    31
   !JSYS pbin;
   RETURN(r1)
   END.
(getjfn) PROCEDURE;                                                    3
   r1 _ $jfntab; r2 _ $string .V 4407B8;
   IF NOT SKIP !JSYS gtjfn THEN error(r1);
   jfn _ r1; r1 _ r1 .A 18M;
   r2 _ 4400002B5;
   IF NOT SKIP !JSYS openf THEN error(r1);
   wdnum _ filpg _ pointr _ lc _ 0;
   RETURN
   END.
(readsw) PROCEDURE;                                                    3.
   % parse switch and set flags, etc. %
   LOCAL x, count, symval, char;
   x _ sp;
   CASE (char _ |sp) OF
      ="S, ="s:  sssflg_TRUE;
      ="M, ="m:  mflg _ TRUE;
      ="W, ="w:  sssflg_FALSE;
      ="G, ="g:  globfg_TRUE;
      ="X, ="x:  globfg_FALSE;
      ="A, ="a:  moveSymsFlag_FALSE;
      ="B, ="b:
         BEGIN
         bltable();
         rlc _ symloc.RH + stpst - stp;
         END;
      ="P, ="p: % go to next page boundary unless on one now %
         IF rlc .A 9M THEN rlc _ rlc .A 9M3 + 1B3;
      ="I, ="i: % go to next page boundary regardless %
         rlc _ rlc .A 9M3 + 153;
      ="J, ="j: % print undefined symbols %
         printud();
      =";:  NULL;  % comment %
      ENDCASE
         BEGIN
         sp _ x; % backup over first char %
         IF NOT gtvlu( :char, x) THEN RETURN;
         CASE char OF
            ="E, ="e: [x] _ $ArcLdr;
            ="L, ="l: [x] _ $linker;
            ="O, ="o: rlc _ x;
            ="P, ="p: rlc _ [x].LH;
            ="A, ="a: [x] _ rlc;          % next free location %
            ="F, ="f: [x] _ rlc - 1;      % last used location %
            ="R, ="r: [x].LH _ rlc;       % next free location %
            ="O, ="o: [x].LH _ rlc - 1;   % last used location %
            ="N, ="n: % New symbol table %
               BEGIN
               [x] _ stp+1; % store pointer in given loc %
```

```
            [x].LH _ (stp-stpst);
            [x+1].RH _ &phash;
            IF stpbot = stpst+1 THEN stpbot _ stp - hashsize;
            &phash _ stpbot;
            stpst _ stp _ stpbot - 1;
            END;
         ="U, ="u: % go back to old table %
            BEGIN
            stp _ [x].RH - 1;
            stpst _ stp - ([x].LH .V 18M6);
            &phash _ [x+1].RH;
            END;
         ="S, ="s: IF stp=stpst THEN stpst _ stp _ x;
         ="C, ="c: [x](stp+1, stpst);
         ="K, ="k: [x] _ stpst-stp; % current length of symbol table %
         ="Q, ="q: [x] _ stp+1;     % current low point of symbol table
         %
         ="T, ="t: [x] _ stpst;     % current high point of symbol table
         %
         ="H, ="h: IF hashbot = &phash-1 THEN % set new hash table area
         %
            BEGIN
            hashtop _ x;
            &phash _ hashtop - hashsize;
            hashbot _ &phash-1;
            FOR x _ 0 UP UNTIL >= hashsize DO phash[x] _ 0;
            END;
         ="V, ="v: [x] _ hashtop-hashbot; % cur length of hash table %
         ="Y, ="y: [x] _ hashbot+1;   % current low point of hash table
         %
         ="Z, ="z: [x] _ hashtop;     % current high point of hash table
         %
         ="_:
            BEGIN
            IF NOT gtvlu( : char, count) THEN RETURN;
            [x] _ count;
            END;
         ENDCASE BEGIN ws($"illegal switch - ignored"); cr() END
      END;
   RETURN
   END.
(gtvlu) PROCEDURE;                                                    31
   % evaluate symbol in switch %
   LOCAL y, count, symval, char, sv, indir_0;
   REF sv;
   symval _ 0;
   LOOP
      BEGIN
      y _ sp; % save 'previous' pointer %
      CASE (char _ |sp) OF
         ="[: % set flag to get contents of word %
            BEGIN
            BUMP indir;
            REPEAT LOOP;
            END;
         =".: % collect symbol name %
```

```
            BEGIN
            count _ 0;
            % read symbol name and convert to radix 50 value %
               LOOP
                  BEGIN
                  CASE char _ |sp OF
                     IN ['A, 'Z]:     char _ char - 54;
                     IN ['a, 'z]:     char _ char - 86;
                     IN ['0, '9]:     char _ char - 47;
                     = SP:            REPEAT CASE; % ignore spaces %
                     =0, ='.:                 EXIT LOOP;
                  ENDCASE BEGIN
                     ldwerr( $"illegal function name");
                     RETURN (FALSE); END;
                  BUMP count;
                  IF count <= 6 THEN symval _ symval*50B + char;
                  END;
               IF char = '. THEN char _ |sp;
            % look up the symbol in the hash table %
               IF NOT (y _ xlook( symval .V 4B10 ) ) THEN
                  BEGIN ldwerr( $"symbol not found" ); RETURN(FALSE); END;
            % get value of symbol and return %
               &sv _ y;
               &sv _ sv[1];
            END;
         IN ['C, '7]:
            BEGIN
            IF NOT SKIP !nin(y, 0, 8) THEN error(R3);
            sp _ R1;
            char _ .r1;
            &sv _ R2;
            END;
         ENDCASE
            BEGIN
            ldwerr($"illegal switch value");
            RETURN(FALSE);
            END;
      EXIT LOOP;
      END;
   IF irdir THEN &sv _ sv; % perform indirection if requested %
   RETURN(TRUE, char, &sv);
   END.
(done) PROCEDURE;                                                    31
   % finish up tasks: undefined symbols printed, map, blt symbol table if
   desired %
   printud();
   IF stpst=stable AND NOT nlsflag AND moveSymsFlag THEN bltable()
   ELSE
      BEGIN
      symloc _ stp+1;
      symloc.LH _ (stp-stpst);
      END;
   IF nlsflag=2 AND ddtPtr#0 THEN [ddtPtr] _ symloc;
   IF mflg THEN printm();
   RETURN
   END.
```

```
(bltable) PROCEDURE; % BLT symbol table to follow prog %                    3M
   (lSymTab);  % length of sybol table %                                   3M1
   (newEndLocP1);  % new ending location of symbol table plus 1 %          3M2
   lSymTab _ stpst - stp;
   newEndLocP1 _ (rlc + lSymTab + 777B) .A 9M3;
   r1 _ newEndLocP1 - lSymTab; r1.LH _ stp+1;
   symloc _ r1.RH; symloc.LH _ -lSymTab;
   r2 _ newEndLocP1;
   !BLT 1,0(2) ;
   moveSymsFlag _ FALSE;
   RETURN;
   END.
% SYMBOL TABLE MANIPULATION ROUTINES %
   (xlook) PROCEDURE(s % => symPtr, hashPtr/-1 %);                          4A
      % lookup global symbol s in hash/symbol table.  Return address of symbol
      table entry and address of pointer word in hash table (-1 if none) %
      LOCAL ptr;
      IF nlsflag AND NOT storeHash THEN
         % looks up s in the part of the symbol table built since Nls*.sav was
         made.  The current value of stp and the global value nlslimit (saved
         in routine cmndparse when loading Nls*.sav) are used to define the
         current bounds of the symbol table %
         BEGIN
         ptr _ stp + 1;
         WHILE ptr < nlslimit DO
            BEGIN
            IF [ptr] = s THEN RETURN(ptr, -1);
            ptr _ ptr + 2;
            END;
         END;
      IF ncht THEN % do not use hash table %
         BEGIN
         ptr _ stp+1;
         WHILE ptr < stpst DO
            BEGIN
            IF [ptr] = s THEN RETURN(ptr, -1);
            ptr _ ptr + 2;
            END;
         RETURN(0, -1);
         END;
      % registers r1-r3 are used in the search loop as follows
         r1      pointer to entry in hash table chain
         r2      pointer to symbol table entry
         r3      local copy of symbol value %
      % hash the symbol by using integer divide %
         r3 _ s;
         r1 _ IF r3 >= 0 THEN r3 ELSE -r3; % make sure its positive %
         !IDIVI r1,hashsize; % remainder to r2 %
      % set initital pointer to hash table entry %
         r1 _ phash[r2];
      % loop down the hash chain, looking for the symbol %
      (lookuploop):                                                         4A
         !JUMPE r1,trySymTab; % jump if hash chair exhausted %
         !HLRZ          r2,0(r1); % fetch pointer to symbol table entry %
         !CAMN          r3,0(r2);% compare with symbol %
         !JRST          lookupdone; % found the symbol %
```

```
      !HRRZ           r1,0(r1); % link to next entry in hash table %
      !JRST           lookuploop; % try again %
   (lookupdone):                                                          4A10
      RETURN (r2, r1);
   (trySymTab):                                                           4A11
      IF storeHash THEN RETURN (0, -1);  %no use looking in symbol table %
      ptr _ stp+1;
      WHILE ptr < nlslimit DO  % stuff after nlslimit was in hash table %
         BEGIN
         IF [ptr] = s THEN RETURN (ptr, -1);
         ptr _ ptr + 2;
         END;
      RETURN(0, -1);
   END.
(henter) PROCEDURE(s,a,type);                                             4E
   % this routine makes a symbol table entry in the symbol table and links
   the entry into the hash table;  the leftmost bit of the value word is
   set to TRUE if the symbol table entry is undefined %
   LOCAL
      x, % chain link in the hash chain %
      y; % pointer to the symbol table entry %
   % check for an unexpected value %
      IF a.LH # 0 AND type = undefinedtype THEN
         BEGIN % LH of undefined symbol already set! %
         lowerr( $"LOADER CONFLICT- L.H. of undef symbol non-zero " );
         wsym(s,a); cr();
         END;
   % set ptr to new entry into y (depends on type) and make new symbol
   table entry %
      IF type = undefinedtype THEN
         BEGIN
         IF NOT storeHash OR hashbot<=hashlb THEN
            % We must make the undefined symbol table entry in the DDT
            symbol table even though it is in the wrong block so that we
            can find it and fix it up later %
            BEGIN
            IF stp<=symlb THEN
               BEGIN
               HELP(symtype : symlb);
               noPT _ TRUE;
               storeHash _ FALSE;
               END;
            y _ stp - 1;
            stp _ stp - 2;
            END
         ELSE
            IF (y _ ulink) = 0 THEN
               BEGIN
               y _ hashbot-1; % allocate a new two word entry %
               hashbot _ hashbot - 2;
               END
            ELSE ulink _ [ulink];
         % make an entry at address y %
         [y+1] _ a.RH .V undefinedmask;
         [y] _ s;
         END
```

```
      ELSE
         BEGIN % make entry in normal ddt smbol table %
         y _ stp - 1;
         entr(s,a); % entry is made at location stp %
         END;
   % make entry in the hash table if in use %
      IF storeHash THEN
         BEGIN
         % check if past lower bound for hash table %
            IF nlsflag AND hashbot<hashlb THEN
               BEGIN
               HELP(hashtype);
               storeHash _ FALSE;
               RETURN;
               END;
         % hash the symbol %
            r1 _ IF s >= 0 THEN s ELSE -s; % make sure its positive %
            !IDIVI r1,hashsize; % result to r2 %
         % create a link in the hash table %
            x _ phash[ r2 ];
            phash[ r2 ] _ hashbot;
         % make a new entry in the hash table %
            [hashbot].LH _ y;
            [hashbot].RH _ x;
            hashbot _ hashbot - 1;
         END;
   RETURN;
   END.

(entr) PROCEDURE(s,a);                                                    4C
   % enter symbol s in symbol table, value a %
   IF stp<=symlb THEN
      IF nlsflag THEN
         BEGIN
         HELP(symtype : symlb);
         noht _ TRUE;
         storeHash _ FALSE;
         END
      ELSE ldwerr($"symbol table overflow");
   [stp-1] _ s; [stp] _ a; stp _ stp-2;
   RETURN
   END.
(linker) PROCEDURE(s, a); % link address a to symbol s %                  4[
   % externally callable interface for udef. Used to store an address of a
   symbol somewhere (or put it on end of a undefined chain) given the
   symbol  s  and the address of the word to be fixed   a.    (s is a
   radix50 symbol )   %
   % store the address of this procedure in location <n> by using switch
   /<n>L   %
   % make sure symbol table pointers stp and stpst are correct %
      stp _ symloc.RH-1;
      stpst _ (stp - symloc.LH) .A 18M;
   a.LH _ 0;
   [a].RH _ 0;  % ensure address zero now %
   udef(s, a);
   % now update location 116 (may have entered undefined ref) %
```

```
      symloc _ stp+1;
      symloc.LH _ (stp-stpst);
   RETURN(Ca].RH);
      % this is zero if undefined, value of s if defined %
   END.
(ucef) PROCEDURE(s,a);                                                    4E
   % handles global request symbols - except deferred global stuff.  IF
   it's defined, fix it up, else enter in undefined table (if possible) %
   LOCAL y_0,p;
   IF a<0 THEN BEGIN
      ldwerr($"global request bit zero on  -improper linking done");
      cr() END;
   s.codbit _ 1; % make the undefined global into a global symbol %
   IF (y _ xlook(s)) # 0 THEN % symbol already in table %
      IF [y+1] .A 18M6 # undefinedmask THEN fixup(a,[y+1])
      ELSE  % have found an undefined symbol table entry %
         BEGIN
         p _ a; % append to (already existing) undefined link%
         %WHILE [ p ] .A 18M # 0 DO p_[p];% %find end of chain%
         % execute this in registers ! %
         !MOVSI a4,udef1; !BLT a4,3;
         a2 _ p; !JRST 1; % setup a2 and go %
         (udef1): !MOVEI a2,0(a1); !HRRZ a1,0(a2): !JUMPN a1,0;          4E5B7
            !JRST udef2;
         (udef2):                                                        4E5B8
         p _ a2;
         [p].RH _ [y+1];
         [y+1].RH _ a;
         END
   ELSE henter(s,a.RH,undefinedtype);
   RETURN;
   END.
(glblok) PROCEDURE(s,a);                                                  4E
   % check symbol s for global def: mulitple definition results in message
   on tty (unless same value), undef symbol table gets searched for
   reference to s, fixups are done if necessary.%
   LOCAL x, y;
   IF (y _ xlook(s : x)) THEN
      IF [y+1] .A 18M6 # undefinedmask THEN % symbol previously defined %
         % if we are called via the nlsloader, (or previous definition in
         another symbol table) then we will define a new external symbol %
         IF [y+1]#a THEN
            IF NOT nlsflag AND (y+1) IN [stp,stpst] THEN
               BEGIN % multiply defined symbol %
               ldwerr($"multiply defined global ");
               cr();
               wsym(s,a);
               ws($" (was and is =)  "); wn([y+1]); cr();
               END
            ELSE
               henter(s,a,definedtype)
         ELSE NULL
      ELSE  % symbol is in table, but undefinec %
         BEGIN
         fixup([y+1].RH,a);
         IF NOT noht AND y IN [hashbot,hashtop] THEN
```

```
            BEGIN % fix up undefined symbols %
            % the entry must be deleted from the hash table and moved to
            the normal DDT symbol table, the space recovered is linked onto
            the chain ulink %
            IF x#-1 THEN [x].LH _ sto - 1; % fixup entry in hash table %
            entr(s,a); % put symbol entry into DDT table %
            [y] _ ulink; % link freed entry to ulink chain %
            ulink _ y;
            END
         ELSE % symbol found in local ddt table %
            [y+1] _ a; % stick real value in ddt symbol table entry %
         END
      ELSE henter(s,a,definedtype);
      RETURN;
      END.
% LOADER PROCESSING ROUTINES %
   (loadfl) PROCEDURE;                                                    5
      % main file-loading loop. parse block type %
      LOCAL w;
      LOOP BEGIN
         w _ !binwd(); blktyp _ w.LH; blksz _ w.RH;
         CASE blktyp OF
            =1: code();
            =2: syms();
            =4: skip();
            =5: BEGIN endblk(); RETURN END;
            =6: pgname();
            =7: ldstblk();
            =10B: fixblk();
            ENDCASE BEGIN
               wdnum _ (filpg-1)*1000B + pointr.RH - corpg*1000B;
               ldwerr($"format error, word "); wn(wdnum);
               ws($" block type "); wn(blktyp); cr();
               RETURN END END
      END.
   (code) PROCEDURE;                                                      5
      % read a code block - load into core %
      LOCAL c, savrlcwd, savpointr;
      LOOP BEGIN
         rlcwd _ !binwd();
         c _ MIN(blksz,18); blksz _ blksz-c;
         lc _ !reloc().RH;
         % check if past boundaries for loading program; try and get help;
         NOTE: c includes first data word which is location counter, not code
         %
            IF lc IN [0,17B] THEN RETURN(ldwerr($"Outside allowed area"));
            WHILE lc+c-1 > loadub DO
               IF nlsflag THEN
                  BEGIN
                  savrlcwd _ rlcwd; savpointr _ pointr;
                  HELP(uspgtype : loadub);
                  rlcwd _ savrlcwd; pointr _ savpointr;
                  END
               ELSE RETURN(ldwerr($"Outside allowed area"));
         % LOOP BEGIN
            IF (c_c-1)<=0 THEN EXIT;
```

```
        [lc] _ !reloc();
        BUMP lc
        END; %
      % execute this loop in registers %
      !MOVSI a4,code1; !ADDI a4,2; !BLT a4,5;
      !MOVN r6,c; !MOVSI r6,1(r6); !HRR r6,lc; !JRST 2; % setup r6 and go %
      (code1):                                                              5B31
        !reloc(); !MOVEM r1,0(r6); !AOBJN r6,2; !JRST code2;
      (code2):                                                              5B3<
      IF blksz=0 THEN RETURN
      END
   END.
(syms) PROCEDURE;                                                           5(
   % read symbol block - parse symbol types %
   LOCAL c, w1, w2;
   LOOP BEGIN
      rlcwd _ !binwd();
      c _ MIN(blksz,18); blksz _ blksz-c;
      LOOP BEGIN
         IF (c_c-2)<0 THEN EXIT;
         w1 _ !reloc();
         w2 _ !reloc();
         CASE w1.codbit OF
            =1: %global% IF globfg THEN glblok(w1,w2);
            =14B: %global req.% udef(w1,w2);
            IN [2,3]: IF sssflg THEN henter(w1,w2,definedtype);
            ENDCASE henter(w1,w2,definedtype)
         END;
      IF blksz=0 THEN RETURN
      END
   END.
(fixup) PROCEDURE(chain, a);                                                5[
   % fixup chain with value a (right half) %
   LOCAL x;
   %DO BEGIN
      x _ [chain]; [chain].RH _ a;
      chain _ x.RH END
      WHILE chain.RH # 0;%
   % execute the DO loop in the registers %
   !MOVSI a4,f1; !BLT a4,4; % BLT into registers %
   a1 _ chain; a2 _ a; !JRST 0; % setup and go %
   (f1):                                                                    5D'
      !MOVE a3,0(a1); !HRRM a2,0(a1); !MOVEI a1,0(a3); !JUMPN a1,0;
      !JRST sysrtn;
   NULL %return is in above loop %
   END.
(fixuplh) PROCEDURE(chain, a);                                              5I
   % fixup chain with value a (left half) %
   LOCAL x;
   DO BEGIN
      x _ [chain]; [chain].LH _ a;
      chain _ x.LH END
      WHILE chain.RH # 0;
   RETURN
   END.
(fixblk) PROCEDURE;                                                         5
```

```
   % read fixup block - handles -1 (left half fixup) entries %
   LOCAL p, c, w, f;
   f _ 0;
   LOOP BEGIN
      rlcwd _ !binwd(); c _ 0;
      LOOP BEGIN
         IF blksz=0 THEN RETURN;
         IF c=18 THEN EXIT;
         w _ !reloc();
         blksz _ blksz-1;
         IF w=-1 THEN f_1
         ELSE IF f THEN BEGIN fixuplh(w.LH, w.RH); f_0 END
         ELSE fixup(w.LH, w.RH) END
      END
   END.
(skip) PROCEDURE;                                                    5G
   % skip over block taking no action %
   LOCAL c;
   LOOP BEGIN
      !binwd(); c _ 0;
      LOOP BEGIN
         IF blksz=0 THEN RETURN;
         IF c=18 THEN EXIT;
         !binwd(); BUMP c; blksz_blksz-1
         END
      END
   END.
(pgname) PROCEDURE;                                                  5H
   % enter program name and save necessary stuff for endblk %
   LOCAL w;
   !binwd();
   w _ !binwd();
   IF blksz=2 THEN !binwd();
   entr(w,lc+rlc);
   pname_w; ploc_stp+2; psym_stp+1;
   RETURN
   END.
(lastblk) PROCEDURE;                                                 5I
   % set starting location in codeLoc %
   rlcwc _ !binwd();
   codeLoc.RH _ !reloc();
   RETURN
   END.
(endblk) PROCEDURE;                                                  5J
   % enc block: correct reloc const. an fixup prog. name info for DDT%
   rlcwd _ !binwd();
   rlc _ !reloc().RH;
   entr(0,rlc); % program break %
   [ploc].LH _ stp-ploc;
   IF rlc IN [stp-2,stpst) THEN BEGIN
      ldwerr($"symbol table and code overlap"); cr() END;
   BUMP [initlptr]; % creates private copy of file page so can close; done
   like this rather than via [initlptr] _ 0; because of bug on 2020. %
   IF NOT SKIP !closf(jfn) THEN error(rl);
   RETURN
   END.
```

```
(xbinwd) PROCEDURE;                                                    5K
   (binloc): % called by !binwd() (user JSYS) %                        5K1
      % return is kept in register a1 %
      !MOVE r1,1(pointr); !AOBJN pointr,0(a1); % return word %
         sr2 _ r2; sr3 _ r3; % save r2, r3 %
         !pmap(jfn*1B6+filpg,4B11+corpg,100400B6);
         !ADD filpg; pointr _ initlptr;
         r2 _ sr2; r3 _ sr3;
         !MOVE r1,0(pointr); !JRST 0(a1)
   END.
(xreloc) PROCEDURE; % user JSYS reloc to read next word and relocate it per
rlcwd %                                                                5L
   (relocl): % relocate word according to rlcwd and return value in word%
                                                                       5L1
      % called with !reloc() - return kept in a2 %
      !binwd();
      !MOVS a3,rlc;
      !TLNE rlcwd,4B5;  % left half relocation %
      !ADD r1,a3;
      !MOVSS 0,r1;  % exchange halves %
      !TLNE rlcwd,200000B;  % right half relocation %
      !ADD r1,a3;
      !MOVS r1,r1; % get it right side up %
      !LSH rlcwd,2; % shift rlcwd for next input word %
      !JRST 0(a2);  % return %
   END.
% I/O UTILITY ROUTINES %
(cr) PROCEDURE;                                                        6
   ws( $newline );
   RETURN
   END.
(error) PROCEDURE(x);                                                  6
   r1 _ jocesignator; r2 _ 4B11+x; r3 _ 0;
   !JSYS 11B%erstr%; !JFCL; !JFCL; cr();
   iodesignator _ r1;
   BUMP errorn;
   IF nlsflag=2 THEN err($"Fatal error in loading") ELSE GOTO rstart;
   END.
(printm) PROCEDURE;                                                    6
   % print map (start location and length) %
   LOCAL x;
   cr(); ws($"MAP:"); cr();
   x _ stpst+2;
    WHILE (x_x-2)>stp DO
      IF [x-1] AND [x-1].codbit=0 THEN BEGIN
         wsym([x-1],[x].RH);
         ws($" to "); wn( [ [x].LH+x+2] ); cr() END;
   cr();
   ws($"symbol table "); wn(symloc.RH);
      ws($" to "); wn(symloc.RH-1-(symloc.LH .V 18M6));
   cr();
   ws($"last free adr = "); wn(rlc);
   RETURN
   END.
(printuc) PROCEDURE;  % list undefined symbols %                       6
   LOCAL ptr, hptr, i, sw _ FALSE;
```

```
   IF NOT nobt THEN
      FOR i _ 0 UP UNTIL >= hashsize DO
         BEGIN % link down the i'th hash chain %
         hptr _ phash[ i ];
         WHILE hptr DO
            BEGIN
            ptr _ [hptr].LH; % get pointer to 2 word sym tab entry %
            IF [ptr+1] .A 18M6 = undefinedmask AND [ptr].codbit = 1 THEN
               BEGIN
               IF NOT sw THEN
                  BEGIN
                  sw _ TRUE;
                  ws( $"*** Uncefined Globals ***");
                  END;
               cr();
               wsym([ptr], [ptr+1].RH);
               BUMP udfnum;
               END;
            hptr _ [hptr].RH; % link to next entry in the hash chain %
            END;
         END;
   IF NOT storeHash THEN
      BEGIN
      ptr _ stb + 1;
      WHILE ptr < nlslimit DO
         BEGIN
         IF [ptr+1] .A 18M6 = undefinedmask AND [ptr].codbit = 1 THEN
            BEGIN
            IF NOT sw THEN
               BEGIN
               sw _ TRUE;
               ws( $"*** Undefined Globals ***");
               END;
            cr();
            wsym([ptr], [ptr+1].RH);
            BUMP udfnum;
            END;
         ptr _ ptr + 2;
         END;
      END;
   IF sw THEN cr();
   RETURN;
   END.
(ldwerr) PROCEDURE(s);                                                  61
   % write error string and bump errorn %
   BUMP errorn;
   ws(s);
   cr();
   RETURN
   END.
(wn) PROCEDURE(n);                                                      61
   IF maxout <=0 THEN RETURN;
   r1 _ iodesignator; r2 _ n; r3 _ 8;
   IF NOT SKIP !JSYS nout THEN error(r3);
   iodesignator _ r1;
   maxout _ MAX(maxout-6, 0);
```

```
    RETURN
    END.
(ws) PROCEDURE(s);                                                    6(
    % do sout %
    IF maxout <=0 THEN RETURN;
    r1 _ iodesignator; r2 _ (s+1) .V 18M6; r3 _ -[s].RH; r4 _ 0;
    !JSYS sout; iodesignator _ r1; maxout _ maxout-[s].RH;
    RETURN
    END.
(wsym) PROCEDURE(s,a);                                                6F
    % write symbol s on tty, followed by value a %
    LOCAL j;
    j _ 6; r1 _ s.symbl;
    WHILE (j_j-1) >= 0 DO BEGIN
        !IDIVI 1,50B;
        ary[j] _ (CASE r2 OF
            =47B: "%;
            =46B: "$;
            =45B: ".;
            >12B: r2+66B;
            >0: r2+57B;
            ENDCASE SP) END;
    r1 _ iodesignator; r2 _ $ary .V 4444B8; r3 _ -6;
    !JSYS sout; r2 _ SP; !JSYS bout;
    iodesignator _ r1;
    maxout _ MAX( maxout-7, 0 );
    wn(a);
    RETURN
    END.
% ArcLdr INTERFACE ROUTINE %
(ArcLdr) PROCEDURE(njfn, nrlc, nub, arg4, outstr REF);                7
    % The switch /<adr>E is used to store the address of this routine in
    location adr when a file is being loaded.  The loader is then saved with
    the file. %
    % njfn is a open REL file.  nrlc is the location to start loading in.
    nub is the upper bound for safe loading.  the symbol table pointer in
    symloc is used and changed, and the LH of codeLoc is used for the symbol
    table bound %
    % the LH of nub is taken as a special flag (normaly zero). arg4 is only
    referenced if nub.LH is special value
        =777777B: set nlsflag FALSE (nlslimit unchanged)
        =777776B: set nlsflag TRUE and take nlslimit from arg4
        =777775B: set nlsflag FALSE and take nlslimit from arg4
        =777774B: set nlsflag TRUE and don't use hash table %
    LOCAL save1,save2, save3; % save away regs 7 and 8 %
    save1 _ rlcwd; save2 _ pointr; save3 _ iodesignator;
    noht _ FALSE;
    storeHash _ FALSE;
    IF &outstr THEN
        BEGIN
        iodesignator _ 440700000001B+&outstr;
        maxout _ outstr.M;
        END;
    CASE nub.LH OF
        =777775B: nlsflag _ FALSE;
        =777776B:
```

```
            BEGIN
            nlsflag _ 1;
            nlslimit _ arg4.RH;
            END;
         =777775B:
            BEGIN
            nlsflag _ FALSE;
            nlslimit _ arg4.RH;
            END;
         =777774B:
            BEGIN
            nlsflag _ 1;
            noht _ TRUE;
            END;
         ENDCASE nlsflag _ 1;
      mflg _ altfg _ FALSE;
      udfnum _ 0;
      sssflg _ globfg _ TRUE;
      jfn _ njfn; rlc _ nrlc; loadub _ hub.RH;
      stp _ symloc.RH-1;
      stost _ (stp - symloc.LH) .A 18M;
      symlb _ codeLoc.LH;
      wdnum _ filbg _ pointr _ errorn _ lc _ 0;
      loadfl(); done();
      rlcwc _ save1; pointr _ save2; % restore regs 7 and 8 %
      iodesignator _ save3; % restore iodesignator %
      IF %outstr THEN outstr.L _ outstr.M - maxout;
      maxout _ 377777777777B; %infinity%;
      RETURN(errorn, rlc, udfnum)
      END.
(NLSArcLdr)    % CL: ;  load NLS user program into program buffer %
PROCEDURE (njfn, eCode, ubCode, bSym, eSym, lbSym, statusHash, bHash,
eHash, lbHash, outstr REF % => -errorn/nUndefs, eUsPgBuf, bSymTable,
bHashTable %);                                                          7E
   % Procedure description
      ARGUMENTS
         njfn    - jfn of an open REL file
         eCode   - the location (address) after which to start loading
         ubCode  - the upper bound (address) for safe loading.
         bSym    - beginning (lowest)  address of symbol table
         eSym    - end       (highest) address of symbol table
         lbSym   - lower address boundary for symbol table
         statusHash - see <NlsBeSrc,BData,hashStatus>
         bHash   - beginning (lowest)  address of hash table
         eHash   - end       (highest) address of hash table
         lbHash - lower address boundary for hash table
         outstr - err string
      RESULTS
         if there were errors, then -#errors else #undefined-symbols
         eUsPgBuf    - last used user buffer location
         bSymTable   - new beginning address of symbol table
         bHashTable - new beginning address of hash table
      %
   % Declarations %
      (save1) = rlcwd;          % save away regs 7 (rlcwd) %          7B2
      (save2) = pointr;         % save away regs 8 (pointr) %         7B2
```

```
    (save3) = iodesignator;  % save I/O designator %                    7B2(
  % initialize some of ArcLdr's global variables %
    nlsflag _ 2;
    mflg _ altfg _ FALSE;
    udfnum _ 0;
    sssflg _ globfg _ TRUE;
    wcnum _ filpo _ pointr _ errorn _ lc _ 0;
  % set ArcLdr's global variables from input paramters %
    jfn _ njfn;
    rlc _ eCode+1;
    loadub _ ubCode;
    stp _ bSym - 1;
    stpst _ eSym;
    symlb _ lbSym;
    noht _ statusHash<0;
    storeHash _ statusHash>0;
    hashbot _ bHash - 1;
    &phash _ eHash - hashsize;
    hashtop _ eHash;
    hashlb _ lbHash;
    IF &outstr THEN
       BEGIN
       iodesignator _ 440700000001B+&outstr;
       maxout _ outstr.M;
       END;
  % do load %
    loadfl();
    done();
  % restore things %
    rlcwd _ save1;
    pointr _ save2;
    iodesignator _ save3;
    IF &outstr THEN outstr.L _ outstr.M - maxout;
    maxout _ 377777777777B; %infinity%;
  % Return %
    RETURN((IF errorn#0 THEN -errorn ELSE udfnum),
       rlc-1, stp+1, hashbot+1);
  END.
    %%                                                                 7B8A
```

FINISH of ArcLdr

META

```
% Compiler header. (arcsubsys,meta9,) to (l10,meta9,) %

   FILE meta CHECK
   META program

    ERROR: -> *; $( *[ -> *;) $( rule * ) "END"  :endr[] *;
    SIZE: S=1900 M=200 K=75 N=2500 L=500 G=50;

    FLAGS: genfg first check negw symseq nmt dolrev
      bkup dolr inval parse kref erfg szfg nestc1 nestcn;

    DUMMY: alter xp backup nt attrn mknode
       and nreg subitem ndt dfg p1;

    SET: X=1 C=2 W=4 M=5 K=3 N=7 D=2 ASM=0 MREG=6
      ME=8 R=15 Y=10 A2=11 A3=12 T=12 LABF=3 PTRF=2
      CHRF=1 IDF=11B UIDF=12B SRF=13B SSF=10B;

    FIELDS: A=[4:23] XR=[4:18] OP=[9:27] SF=[6:24] PF=[6:30]
      GBIT=[1:26] LH=[18:18] MOD=[3:27];

    ATTRIBUTES: libry valrul valref;

    OPCODES: MOVNS=213B HRL=504B AOBJN=253B JUMPGE=325B IDIV=230B
      TLO=661B IOR=434B XOR=430B;

% Compiler header syntax. %
    program  =                                                          3
      "FILE" .ID &LABEL <"-TREE 3.1 7-Jul-77 "> :namep[1]*
         &FR check ["CHECK" &FS check] "META" .ID  :strt[1]*
         $(headelt) :hack[]*  $(rule * ) "END"  :endr[]*/
      "LIBRARY" $( .ID @S libry &DISCARD) /
      <"NOT A META PROGRAM">;

    strt [-] => >|begin PUSHJ M|A *V1, pj["qxerr"] JRST 'finish',       31
      &attrn_0 >|gn, >|gn1, >|gn2, &nreg_N;

    mkn => PUSH N|A 'nmark', MOVEM N|A 'nmark',;                        3(
    mkk => PUSH K|A 'kmark', MOVEM K|A 'kmark',;                        3(
    pnm => MOVE N|A 'nmark', POP N|A 'nmark',;                          3(

    namep [-] => &NAME *1 <"-FILE " *1>                                 3(
      &FR nestcn, erfg, szfg;

    endr =>  $SYMS(                                                     3(
      ?@ libry *$ [?@ defned *$
         <"library symbol '" *$ "' used"> ] /
      ?@ valref *$ (?@ valrul *$ / <*$ " used as value rule" > )
      ) &TABLES;

    headelt =                                                           31
      "DUMMY:" $( .ID :[>|*1 CERR 1005,] &DISCARD ) *;  /
      "ERROR:" &FS erfg,parse scn  exp   :erul[2]* *;  /
      "FLAGS:" $( .ID :[ >|*1 ,] &DISCARD ) *;  /
      "ATTRIBUTES:" .$ .ID :attrib[$]* *;  /
      "OPCODES:" .$( .UID *= .NUM :opdf[2]*) *; /
```

```
    "SET:" .$(.UID *= .NUM :setdf[2]*) *; /
    "SIZE:" &FS szfg $( siz1 ) *; /
    "FIELDS:" $(.UID *= *[ .NUM *: .NUM *] :fldd[3]*) *; ;

  siz1 =                                                                    2
    *S *= .NUM :[>|sspx . *N1 7|SF 1|PF, >|ss &BSS *N1,] &DISCARD/
    *M *= .NUM :[>|msx -*N1|LH . 1, >|mstack &BSS *N1,] &DISCARD/
    *K *= .NUM :[>|ksx -*N1|LH . 1, >|kstack &BSS *N1,] &DISCARD/
    *N *= .NUM :[>|nsx -*N1|LH . 1, >|nstack &BSS *N1,] &DISCARD/
    *G *= .NUM :[>|gsx -*N1|LH . 1, >|lstack &BSS *N1,] &DISCARD/
    *L *= .NUM :[>|litx *N1, >|lity &BSS *N1,
       >|litr &BSS *N1,] &DISCARD;

  hdck => ?F erfg ?AND ?F szfg /                                            3
    <"size parameters and/or error statement missing">;

  erul [-,-] => >|qsynqq *1 nestxp[*2,=0,=0,=0] JRST *finish*,;             3

  attrib [$] =>                                                             3
    $(>*$ attrn|PF 1|SF *hta* X|XR, &attrn_attrn+1)
    [? attrn>35 <"only 36 attributes">];

  opdf [-,-] => >*1_*N2 @S opcode, noddt *1 @R reloca *1;                   3
  setdf [-,-] => >*1_*N2 @R reloca,opcode *1 @S noddt *1;                   3
  fldd [-,-,-] => >*1 *N3|PF *N2|SF W,;                                     3

%  Rules. %
  rule = .ID &LABEL                                                         4
    &FR  bkup, genfg, parse, kref, inval
    (( genpart / *= &FS parse exp) *; /
       outrul) :prule[2];

  prule [-,alter] => >|*1 [?F genfg pj["savlb"]]                            4
    [?F inval @S valrul *1]
    (?F parse ?AND ?F kref PUSH M|A ME, MOVE ME|A K,
       nestxp[*2,G#1,=1,=0] POP M|A ME, retrnt[]
       >#1 POP M|A ME, brnchf[=0] /
    nestxp[*2,=0,=1,=1] retrnt[]);

  retrnt => ?NOT ?F parse retot[] / ?F genfg JRST *rtnlt*, /                4
    JRST *rtnt*,;

  retot => ?F genfg JRST *ortnlt*, / JRST *ortnt*,;                         4

  brnchf % produce a branch false ( one instr ) %                           4
    [0] => ?F bkup (?F genfg JRST *bkuprl*, /JRST *bkupr*,)/
        ?F genfg JRST *rtnlf*, / POPJ M|A,;
    [#1] => ?F bkup BKUPJ #1, / JRST #1,;

  berr => ?F parse pj["qxerr"] / CERR,;                                     4

  ftadr  % failure target address %                                         4
    [0] => ?F genfg *ortnlf*/ *ortnf*;
    [#1] => #1;

% Syntax expressions. %
```

```
    exp = subexp .$( '/ subexp) :alter[1,$];                                4K

   % These outrules compile expressions. %
      %   Arguments are
          [expression, false label,
          =0: error if exp fails,
          =1 if can return after exp]   %

    concat % concatenate a list of subexpression elements %                 4K
      [backup,-,-,-] => &FS bkup pj["bsav"] $*1( ele[*$,*2,=1,=0])
          &FR bkup pj["bpop"];
      [xp[-],-,-,-] => ele[*1:1,*2,*3,*4];
      [xp,-,-,-] => &FS first $*1( (?F first ele[*$,*2,*3,=0]/
          ?LAST ele[*$,=0,=0,*4] /
          ele[*$,=0,=0,=0]) &FR first);
      [ndt,-,-,-] => nct[*1:1,*2]
          [ ?[ndt[list],-,-,-] $*1:1( ndt1[*$,*2])];
      [mt,-,-,-] => TRUE;

    nct [-,-] => CNTST cknum[=*Q1,=4M] ftaar[*2],;                          4K
    cknum  [-,-] =>                                                         4K
      ?IF *N1>*N2 <"too many nodes " LOC LINE> / *N1|A;

    ele % produce code for a subexpression element %                        4K
      [nt[option[-]],-,-,-] => ele[*1:1:1,G#1,=1,*4] >#1;
      [nt,-,-,-] => *1:1;
      [alter,-,-,-] => ?NOT ?F bkup nestxp[*1,*2,*3,*4]/
          bnest[*1,G#1,=1,=0] JRST #2, >#1 brf[*2,*3] >#2;
      [and,-,-,-] => $*1 ( ?LAST ele[*$,*2,*3,*4] /
          ele[*$,*2,*3,=0]);
      [not[alter],-,-,-] => bnest[*1:1,G#1,=1,=0]
          brf[*2,*3] >#1;
      [arb,-,-,-] => arb[*1:1,*1:2,*1:3,*2,*3];
      [-,-,-,-] => *1 brf[*2,*3];

    brf [-,-] => ?*N2=1 brnchf[*1] / berr[];                                4K

    bnest [-,-,-,-] => !bkup( &FR bkup nestxp[*1,*2,*3,*4]):                4K1

    nestxp  % nested expression - possible alternations %                   4K1
      [alter,-,-,-] => $*1( ?LAST concat[*$,*2,*3,*4] /
          concat[*$,G#1,=1,*4]
              (?*N4=1 retrnt[] / JRST #2,) >#1)
          >#2;

% Syntax subexpressions. %

    subexp = '_ .$ssbxp :backup[$]/ ssbxp1 .$ssbxp :xp[1,$];                4M

    ssbxp =  -'/ -'; -') -'] (ntest :nt[1] / stest);                        4M

    ssbxp1 = chrchk .$( chrchk ) [ssbxp] :and[1,$] / ssbxp;                 4M
    chrchk = ("+'" :CAIE /"-'" :CAIN) .CHR :pchck[2];                       4M
    pchck [-,-] => *N1|OP C|A *C2,;                                         4M
```

```
% Non-testing syntax elements. %

    ntest = *: ( *[ genexp *] :option[1]/ node[1 :mkunit[1])/          4M1
      *[ exp *] :option[1] /
      ("$ :=0 / ".$" :=1) delimr stest :arb[3] /
      ".CHR" :pj["chr"] /
      "%UNMARK" :pnm[] / "%MARK" :mkn[] /
      "%LABEL" :pj["locset"] /
      "%DISCARD" :pk[] / ** :go[] /
      *! ( %+ASM%  .UID instr / %+ASM%
         idseq *( exp *) :savid[2]) /
      contrl / scn;

    scn = "->" stest :scan[1];                                        4M1

    delimr = *< stest *> / :=0;                                       4M1
    idseq = .ID .$( *, .ID) :list[1,$];                               4M1

    mkunit [srref[-]] => ent[*1:1] PUSH K|A X,;                       4M1
      [sbcn[node]] => *1:1 [?[ ?[?[-,-,1,-]]] CERR 1002,] ;
      [sbcn[valid]] => valid [*1:1:1,K];
      [-] => &nreg_K *1 &nreg_N;

    option [-] => ele[*1,G#1,=1,=0] >#1;                              4M1
    go => pj["gopop"] CERR 1002,;                                     4M1
      % it will skip return for the rule %
    pj [-] => PUSHJ R|A *V1,;                                         4M1
      [-,-] => pj[*1] SUB R|A =*N2+LSH(*N2)18,;
    pk => POP K|A *temp*,;                                            4M1
    ktop => MOVE X|A K|XR,;                                           4M1
    list [$] => $(*$ );                                               4M1
    mt => TRUE;                                                       4M1

    scan [-] => JRST . 3, >#1 pj["inc"] pj["delb"]                    4M1
      ele[*1,#1,=1,=0];

    arb                                                               4M1
      [-,-,-] => [?*N1=1 mkk[]]
         (?*N2=0 >#1 ele[*3,G#2,=1,=0] JRST #1, >#2/
         ele[*3,G#2,=1,=0] >#1 ele[*2,#2,=1,=0]
            ele[*3,=0,=0,=0] JRST #1, >#2);
      % in following, cannot fail directly to *4 if K marked - must
      remove mark first! %
      [-,-,-,-,-] => [?*N1=1 mkk[]]
         (?*N2=0 ele[*3,*4,*5,=0] >#1
            ele[*3,G#2,=1,=0] JRST #1, >#2 /
         ele[*3,*4,*5,=0] >#1 ele[*2,G#2,=1,=0]
            ele[*3,=0,=0,=0] JRST #1, >#2);
    mesgelt = .SR :msg[1] /                                           4M1
      "LOC" :pj["locw"] / "LINE" :pj["errbuf"] /
      _ ** simpl :wrtsr[1] /
      value (*8 :msgv[1,=8] / :msgv[1,=10]);

    msglst [$] => pj["terlf"] $(*$);                                  4M1
    msg [-] => PUSH R|A =*L1, HRRZI Y|A =*S1, PUSH K|A Y,             4M1
      pj["wstr",=2];
```

```
        wrtsr [-] => *1 PUSH R|A, pj["wstrh",=1];                          4M10
        msgv [-,-] => *1 PUSH R|A =*N2, load[*1] PUSH R|A Y,               4M10
          pj["wval",=2];

    % Testing syntax elements. %

        stest =                                                            4M12
          .ID :cll[1] /
          .SR :srtst[1] /
          '( exp ') /
          .SR1 :ctst[1] /
          chrchk /
          ('# :=0 / ".#" :=1) delimr stest :arb[3] /
          '. .UID :pjtest[1] /
          '? (
             'F  .ID  :fgtst[1] /
             '@ .ID :att[1,ktop[]] /
             ["IF"] otest2 ) ;

        pjtest [-] => pj[*1] SKIPN MREG, ;                                 4M12
        srtst [-] => TST =*S1,;                                            4M12
        ctst [-] => CAIN C|A *C1, pj["acvsk"];                             4M12
        fgtst [-] => SKIPN *V1,;                                           4M12

        not [fgtst[-]] => SKIPE *V1:1,;                                    4M12
          [att[-,-]] => *1:2 LDB Y|A *1:1, TRNE Y|A 1,;
          [-] => *1 JRST . 2,;

        att [-,-] => *2 LDB Y|A *V1, TRNN Y|A 1,;                          4M12
        cll [-] => PUSHJ M|A *V1,;                                         4M12

    % Node construction. %

        nodecn = _ .ID '[ cnlist;                                          4(
        cnlist = %FR nestcn nodlist '] :node[2,=1,nestcn] %FS nestcn;      4(

        nodlist = nodelt [nodmany] :mknode[1]/ :mknode[mkl[=0]]:           4(
        nodmany = ', nodelt .$( ', nodelt) :blist[2,$];                    4(

        node [-,-,-,-] => [?*N4#0 lookahd[*2:1] ] gnode[*1,*2,*3];         4(

        gnode                                                              4(
          [-,mknode[mkl[-]],-] => MKD cknum[=*N2:1:1,=3M] *V1
          gbt[*3],;
          [-,mknode[mklst[]],-] => MKDD *V1 gbt[*3],;
          [-,mknode[-],-] => HRRZ T|A N, *2:1 NDLB *V1 gbt[*3],;

        lookahd [blist] => $*1(lookahd[*$]);                               40:
          [sbcn[valld]] => valcll[*1:1:1,M];
          [sbcn[-]] => nodem[*1:1] / <"misplaced * " LOC LINE>:
          [$] => TRUE;

        nodem [node[-,-,0,-]] => *1 POP K|A Y, PUSH M|A Y,;                40:
        valcll [-,-] => mkn[] load[*1] pnm[] PUSH *N2|A Y,;                40:

        gbt [1] => 1|BBIT ;                                                40:
```

```
      [0] => TRUE;

    blist [$] => $_(*$);                                                401

    % Node elements. %

        nodelt = -*] (                                                  401
          .NUM ?F parse :mkl[1] /
          simp :eltld[1] /
          *$ ?F parse :mklst[] /
          node11);

        node11 =                                                        4018
          glabel :eltld[1] /
          .UID defck :opload[1] /
          *= &FR nestc1 value :valld[1]
             [?F nestc1 :sbcn[1] &FS nestcn] /
          .SR :srref[1] /
          .ID (
             *[ &FR nestcn nodlist *]
                (?F parse (** :=1/ :=0)/ :=0)
                :node[3,nestcn] &FS nestcn :sbcr[1] /
             :eltld[1])/
          .SR1 :chrld[1];

        glabel = *# gref / "G#" gref :genlab[1];                        4018
        gref = (.NUM :[?*N1>2 <"illegal gen label " LOC LINE>]          4018
          :gnum[1] / .ID) &FS genfg;
        gnum [-] => *gn*  *N1;                                          4018
        genlab [-] => GEN *1, *1;                                       4018

        srref [-] => STRF =*S1,;                                        4018
        eltld [-] => *1 PUSH nreg|A,;                                   4018
        chrld [-] => PUSH nreg|A =*C1 CHRF|XR,;                         4018
        opload [-] =>  PUSH nreg|A =*V1,;                               4018
        sbcn [-] => POP M|A Y, PUSH N|A Y,;                             4018
        imed [-] => *N1;                                                4018

        defck = ?@ defned / <"udef symbol at " LOC LINE>;               4018

        valld [-] => valad[*1] PUSH nreg|A,;                            4018
        valad [imed] => =*N1:1;                                         4018
          [neg[imed]] => =-*N1:1:1;
          [p1] => *V1:1;
          [p2[-]] => =*V1:1;
          [p2[*N,-]] => *1:2;
          [-] => *1 Y;

        mkl [-] => MKLS *N1,;                                           4018
        mklst => MKLS,;                                                 4018

    % Output rules. %

        outrul = +*[ .$( *[ part ) :alter[$];                           4(

        part = ntpart genpart *; :xp[2];                                4(
```

```
    genpart = "=>" genexp / ":=" &FS inval genexp;                          46
    ntpart = noct .$( *[ noct ) :alter[1,$];                                46

  % Node testing. %

    noct = *$ *] :mt[] / *] :not[mt[]] /                                    409
      &FS nmt item .$( *, item) *]
        (?F nmt :list[1,$] / :mt[1,$]) :not[1];

  % Node item tests. %

    item =  -*, -*]                                                         409C
      *- :mt[] /
      ( (.ID/*? :mt[]) ( *[ ntpart  :subitem[2] /
        :namitm[1]) /
      simp :refitem[1] /
      *. .UID  :fitem[1] /
      .SR :stritem[1] /
      .NUM :nitem[1] /.UID defck :nitem[1] /
      .SR1 :citem[1] /
      *# pref  :gitem[1] ) &FS nmt;

    py => POP TIA Y,;                                                       409C

    not1                                                                    409C
      [subitem[-,alter],-] => RITM *1:1, stcode[*1:2,*2];
      [mt,-] => py[];
      [gitem[-],-] => gitem [*1:1,*2];
      [-,-] => *1 JRST ftadr[*2],;

    stcode [-,-] => JRST ftadr[*2], nestxp[*1,G#2,=1,=0]                    409C
      PPJ . 2, >#2 PPJ ftadr[*2],;

    refitem [-] => POP TIA *temp*, *1 MOVE Y|A, CAME Y|A                    409C
    *temp*,;
    namitm [-] => NAMT *1,;                                                 409C
    citem [-] => py[] AND IMMED Y|A 8M, CAIE Y|A *C1,;                      409C
    fitem [-] => py[] HLRZ A2|A Y, CAIE A2|A fname[*1],;                    409C1
    stritem [-] => STITM =*S1,;                                             409C1
    nitem [-] => py[] compr[*1] Y|A,;                                       409C1

    compr [.UID] => compr[=*V1];                                            409C1
      [-] => ?IF *N1>18M CAME =*N1 / CAIE *N1;

    gitem                                                                   409C1
      [.ID,-] => fitem["LABL"] JRST ftadr[*2], MOVEM Y|A
      *V1,;
      [-,-] => ITMG *N1:1|A ftadr[*2],;

    fname ["UID"] => UIDF;                                                  409C1
      ["ID"] => IDF;
      ["SR"] => SRF;
      ["SR1"] => CHRF;
      ["CHR"] => CHRF;
      ["LABL"] => LABF;
```

```
        [-] => <"NO ITEM FLAG " *1 LOC LINE>;

    % Output expressions. %

        genexp = osbxp .$(*/ osbxp) :alter[1,$];                              4Q11
        osbxp = oelt .$oelt :xp[1,$];                                         4Q11
        oelt = -*; -*) -*] -*/ (otest / genelt);                              4Q11

    % Outexp tests. %

        otest = otest1 (-*? / .$("?AND" otest1) :and[1,$]);                   4Q13

        otest1 = nodecn /                                                     4Q13
          +** ( _ ** simpl  :donode[1] ) /
          *( genexp *)  /
          *? (
             "NOT" otest1 :not[1] /
             *F .ID :fgtst[1] /
             *@ .ID simp :att[1,hx[1]] /
             ["IF"] otest2);

        otest2 = *$ .ID :ojtest[1] / *[ ntpart /                              4Q13
          "LAST" ?F dolr :ltest[] /
          value (
             ">=" :=5 / "<=" :=3 /
             *> :=7 / *< :=1 /
             *= :=2 / *# :=6)
             value :skiprltn[3];

        donode [-] => *1 DOIT,;                                               4Q13

        skiprltn                                                              4Q13
          [-,-,-] => skip1[*3,*2,*1] /
             skip1[*1,=relv[*2],*3] /
             token[*3] load[*1] tad[*3] CAM Y|A *N2|MOD,/
             token[*1] load[*3] tad[*1] CAM Y|A +relv[*2]|MOD,/
             load[*3] PUSH R|A Y, load[*1]
             POP R|A A2, CAM Y|A *N2|MOD A2,;

        skip1                                                                 4Q13
          [imed,-,-] => smnum[=*N1:1] skipi[=*N1:1,*2,*3] / &FAIL;
          [p2[-],-,-] => smnum[=*V1:1] skipi[=*V1:1,*2,*3] / &FAIL;

        skipi [-,-,-] => ?*N1=0 skipz[*2,*3] /                                4Q13
          load[*3] CAI Y|A *N2|MOD *N1,;

        skipz [-,-] => token[*2] tad[*2] SKIP *N1|MOD,/                       4Q13
          load[*2] SKIP *N1|MOD Y,;

        relv [-] := ? *N1#2 ?AND ? *N1#6 8-*N1/ *N1;                         4Q13
        ltest => HLRZ Y|A D, Y|A CAIE [?F dolrev 18M],;                       4Q13

    % Code generation expressions. %

        genelt = -*; -*/ -*) -*] ( ?F inval value  :nt[load[1]]/              4Q15
          outelt / contrl :nt[1]);
```

```
% Definitions. %

    def = ( *1 def1 :extrnl[1] / def2)                                  4Q15C
      (*_ value :dfnv[2] / :dfn[1]);

    def1 = (.ID / .SR) :ent[1] / simp ;                                 4Q15C

    def2 = def1 / glabel :dfg[1];                                       4Q15C

    dfn [dfg] => *1:1 DGN,;                                             4Q15C
      [ent]
      [extrnl] => *1 DFL X,;
      [-] => ckh[*1] DFL,;

    dfnv                                                                4Q15C
      [ent,-]
      [extrnl,-] => *1 load[*2] DFV X,;
      [dfg,-] => load[*2] DGN *1:1 1|GBIT,;
      [-,-] => [?F check *1 SSCK,] load[*2] *1 DFV,;

    extrnl                                                              4Q15C
      [-] => hx[*1] exset[];

    exset => HRRZI Y|A 1, DPB Y|A *exterr*,;                            4Q15C
    ent [-] => ENTR =*S1,;                                              4Q15C1

% Controls. %

    control = *> def /                                                  4Q15E
      *& (
         "BSS" value *, :pbss[1] /
         cllasgn /
         "NAME" simpk :nmblk[1] /
         "TABLES" :pj["tables"] /
         "FAIL" :pfail[] /
         "G#" gref  :geng[1] /
         frfs) /
      *< .$mesgelt *> :msglst[$] /
      *$ ( *S :=1 / *R :=0) idseq simpk :ats[3];

    frfs = *F (*R :=0 / *S :=1) idseq :fgset[2];                        4Q15E

    fgset [-,-] => $*2( (?IF *N1=0 SETZM / SETOM) *V$,);                4Q15E
    simpk = ?F parse :ktop[] / simp :hx[1];                             4Q15E

    geng [-] => GEN *1,;                                                4Q15E
    ats                                                                 4Q15E
      [0,list["all"],-] => *3 SETZM *hta* X|XR, lv[=0] DPB Y|A *all*,;
      [-,-,-] => *3 lv[=*N1] $*2(DPB Y|A *V$,);

    cllasgn = .ID ( *_ (value :loclasgn[2] /                            4Q15E1
      glabel :gsave[2]) / :pj[1] ) ;

    loclasgn [-,imed[0]] => SETZM *V1,;                                 4Q15E1
      [-,-] => ?[-,?[p1[*1],imed[1]]] ?AND
```

```
        (?[-,add] AOS / ?[-,sub] SOS) *V1, /
        load[*2] MOVEM Y|A *V1,;

    osave [-,-] => MOVE Y|A *2 MOVEM Y|A *V1,;                          4Q15E
    obss [-] => load[*1] PUSH R|A Y, pj["advlc",=1];                    4Q15E

    pfail => [ ?F dclr                                                  4Q15E
      <"%FAIL not legal in % "LOC LINE>]
      (?F parse brnchf[=0] / JRST ftadr[=0],);

    nmblk [-] => *1 pj["pgname"];                                       4Q15E

% Output elements. %

    outelt = ( outerm /                                                 4Q15(
      "DATE" :pj["pdate"] /
      "TRUE" :mt[] /
      *= .$ outunit :litrl[$] ) :nt[1] /
      outunit;

    outunit = otest1 / (                                                4Q15(
      *[ genexp *] :option[1] /
      .SR :srout[1] /
      *$ dolexp /
      *! ( %+ASM% .UID instr / %+ASM%
         idseq *( genexp *) :savid[2]) /
      glabel :genadr[1] /
      "*S" simpl :simpout[1] /
      pvalue) :nt[1];

    savid [-,-] => $*1( PUSH R|A *V$,) nestxp[*2,=0,=0,=0]              4Q15(
      $*1_( POP R|A *V$,);

    dolexp = (simp :dolref[1] /"SYMS" :dolsy[]/ :doll[])                4Q15(
      !dolr( (*_ :=1 / :=0 ) *( &FS dolr genexp) *) :dolseq[3];

    dolseq  % note $SYMS not implemented for nesting %                  4Q15(
      [dolsy[],-,-] => &FS symseq PUSH M|A D, *1 >#1
         SKIPN X|A *htp* Y|XR, SOJGE Y|A #1, JUMPL Y|A #2,
         HRR D|A Y, option[*3] HRR Y|A D, SOJGE Y|A #1,
         >#2 &FR symseq POP M|A D,;
      [-,-,-] => PUSH M|A D, *1
         ! dolrev (
            (? *N2=0 &FR dolrev  HLL D|A D|XR, >#2 HLRZ Y|A D,
               JUMPE Y|A #1, POP D|A X, option[*3] JRST #2,/
            &FS dolrev HLRZ A2|A D|XR, SUB D|A A2, MOVNS A2,
               HRL D|A A2, JUMPGE D|A #1, >#2 option[*3]
               AOBJN D|A #2,))
         >#1 POP M|A D,;

    doll  => MOVE D|A ME,;                                              4Q15(
    dolref [-] => *1 MOVE D|A,;                                         4Q15(
    dolsy => HRLI D|A SSF, MOVE Y|A *htmx*,;                            4Q15(

    srout [-] => PAS =*S1,;                                             4Q15(
    simpout [-] => *1 PSR,;                                             4Q15(
```

```
neg [-] => &FS negw *1;                                          4Q15G;
genadr [-] => *1 PGN,;                                           4Q15G;

litrl [$] => pj["lit"] $(ele[*$,=0,=0,=0])                       4Q15G;
  >*1 pj["litpc"];

pvalue = '+ pvalue / ( '- value :neg[1] / value)                 4Q15G;
  ( '| .UID :field[2]/ :primop[1]);

p2                                                               4Q15G;
  [.UID] => lv[=*V1];
  ['N,-] => *2 mop[],;
     % node is a number %
  ['V,-] => hx[*2] mop[] 'htv' X|XR,;
     % hash table value, 36 bits %
  ['R,-] => hx[*2] HRRZ 'htv' Y|A X|XR, negy[];
     % hash table value, RH %
  ['H,-] => hx[*2] HLRZ 'htv' Y|A X|XR, negy[];
     % hash table value, LH (High order half) %
  ['C,-] => *2 MOVE X|A, LDB Y|A 'fscc', negy[];
     % node conatins a character %
  ['Q,-] => *2 MOVE X|A, HLRZ Y|A X|XR,;
     % number of subnodes %
  ['*,-] => p2['V,*2];
  ['L,-] => ckh[*2] MOVE X|A,
     LDB Y|A 'htlx', negy[];
     % length of hash table symbol in characters %

negy => [?F negw MOVN Y|A Y, &FR negw];                          4Q15G;

fielc [-,-] => load[*1] DPB Y|A *V2,;                            4Q15G;

prim = .NUM :imed[1] /                                           4Q15G;
  .UID defck :p2[1] /
  '* (legl .CHR simpl :p2[2] / simpl :p2['*,1]) /
  '' .ID ''  :srhsn[1] :p2['V,1] /
  '. :p1["lc"] /
  .ID ( '[ cnlist &FS nestc1 :valnod[1] /  :p1[1]) ;

legl = +'V / +'L / +'N / +'C / +'Q / +'H / +'R;                  4Q15G;

primop [imed[-]] => pv[*1:1];                                    4Q15G;
  [p1[-]] => prim1[*1:1];
  [p2[-]] => (?@ opcode *1:1 TLO LSH(*V1:1)9 W|A, / pv[=*V1:1]);
  [p2['N,-]] => *1:2 aop[],;
  [p2['V,-]] => *1:2 PAV negbit[],;
  [p2['*,-]] => *1:2 DOIT negbit[], CERR,;
  [neg[-]] => &FS negw primop[*1:1] &FR negw;
  [-] => *1 aop[] Y,;

prim1                                                            4Q15G;
  ["lc"] => lv[=1] MOVEM Y|A 'bcw', aop[] *V1,;
  [-] => aop[] *V1,;

negbit => [?F negw 1|GBIT &FR negw];                             4Q15G;
pv [-] => aop[] numad[*1],;                                      4Q15G;
```

```
    load [imed[-]] => lv[*1:1];                                          4Q15G3
      [p1[-]] => mop[] *V1:1, [ ?[p1["lc"]] poser[] ];
      [neg] => &FS negw load[*1:1] &FR negw;
      [-] => *1 neg[];

    poser => %<"CAUTION: LC TREATED AS VALUE" LOC LINE>%                 4Q15G3
      TRUE;

    lv [-] => mop[] numac[*1],;                                          4Q15G3

    numad [-] => smnum[=*N1] IMMED *N1 / =*N1;                           4Q15G3
    smnum [-] => ?IF RSH(*N1)18=0 / &FAIL;                               4Q15G3

    hx [ent] => *1;                                                      4Q15G3
      [srhsh] => *1  MOVE X|A,;
      [-] => ckh[*1] MOVE X|A,;

    ckh [-] => *1 [?F check SSCK, *1];                                   4Q15G3

    tad [imed] => numad[=*N1:1];                                         4Q15G4
      [p1[-]] => *V1:1;
      [p2[-]] => smnum[=*V1:1] IMMED *V1:1 / =*V1:1;
      [p2['N,-]] => *1:2;

    token                                                                4Q15G4
      [imed]  [p1[-]]  [p2[-]]  [p2['N,-]] => TRUE;

    aop => (?F negw SUB &FR negw / ADD) W|A;                             4Q15G4
    mop => (?F negw MOVN &FR negw / MOVE) Y|A;                           4Q15G4

    idref [-] => PAS =*S1,;                                              4Q15G4

    outerm = ', :pj["produce"] / '\ .NUM :nterm[1];                      4Q15G4
    nterm [-] => lv[=*N1] MOVEM Y|A 'bcw', pj["produce"];                4Q15G4

  % Node references. %

    simp = '* simpl;                                                     4Q15I

    simpl = simp1 &FS kref (simpll :getitm[2]/ :getitm[1])/              4Q15I
       .ID  :idhsh[1] /
      '' .ID '' :srhsh[1];

    simpll = ': .NUM .$(': .NUM) :chase[1,$];                            4Q15I
    simp1 = .NUM / '$ ?F dolr :mt[0] ;                                   4Q15I

  % Produce address of hash number. %

    getitm [mt] => (?F symseq D / D|XR);                                 4Q15I
      [-] => ME|XR (?F parse *N1/ MASK(-*N1)18M);
      [-,-] => getitm[*1] MOVE X|A, *2;

    chase [$] => $(?LAST nadr[*$] / chase1[*$]);                         4Q15I1
    chase1 [-] => ?F check GET nadr[*1], /                               4Q15I1
      MOVE X|A nadr[*1],;
```

```
    nadr  [-] => X|XR MASK(-*N1)18M;                                    4Q15I

    idhsh [-] => *V1;                                                   4Q15I
    srhsh [-] => ent[*1] X;                                             4Q15I

% Values. %

    value = term $( '+ value :add[2] /                                  4Q15I
       '- value :sub[2] );                                              4Q15K

    term = factor $(";*" factor :mult[2]  /                             4Q15I
      ":/" factor :divd[2] );
    factor = '- fctn :neg[1] / fctn;                                    4Q15I

    fctn =                                                              4Q15I
      "RSH(" value ( ', value ') / ') .NUM :imed[1] ) :prsh[2] /
      "LSH(" value ( ', value ') / ') .NUM :imed[1] ) :plsh[2] /
      "MASK(" value ( ', value ') / ') .NUM :imed[1] ) :pmsk[2] /
      "HASH(" simp ') :phsh[1] /
      "XOR(" value ', value ') :pxor[2]/
      "OR(" value ', value ') :pior[2] /
      prim;

    prsh                                                                4Q15I
      [-,imed] => load[*1] LSH MASK(-*N2:1)18M Y|A, ;
      [-,-] => load[*2] MOVN A3|A Y, LSH Y|A A3|XR, ;
    plsh                                                                4Q15I
      [-,imed] => load[*1] LSH *N2:1 Y|A,;
      [-,-] => load[*2] MOVE A3|A Y, LSH Y|A A3|XR, ;
    pior [-,-] =>                                                       4Q15I
      token[*1] load[*2] tad[*1] IOR Y|A, /
      token[*2] load[*1] tad[*2] IOR Y|A, /
      load[*1] PUSH R|A Y, load[*2] POP R|A A3, IOR Y|A A3,;

    pxor [-,-] =>                                                       4Q15I
      token[*1] load[*2] tad[*1] XOR Y|A, /
      token[*2] load[*1] tad[*2] XOR Y|A, /
      load[*1] PUSH R|A Y, load[*2] POP R|A A3, XOR Y|A A3,;

    pmsk [-,-] =>                                                       4Q15K
      token[*1] load[*2] tad[*1] AND Y|A, /
      token[*2] load[*1] tad[*2] AND Y|A, /
      load[*1] PUSH R|A Y, load[*2] POP R|A A3, AND Y|A A3,;

    phsh [-] => *1 HRRZ Y|A,;                                           4Q15K

    add [-,-] =>                                                        4Q15K
      token[*1] load[*2] tad[*1] ADD Y|A, /
      token[*2] load[*1] tad[*2] ADD Y|A, /
      load[*1] PUSH R|A Y, load[*2] POP R|A A3, ADD Y|A A3,;

    sub [-,-] => token[*2] load[*1] SUB Y|A tad[*2], /                  4Q15K
      token[*1] &FS negw load[*2] tad[*1] ADD Y|A, /
      load[*2] PUSH R|A Y, load[*1] POP R|A A3, SUB Y|A A3,;
```

```
        mult [-,-] => token[*1] load[*2] tad[*1] IMUL Y|A, /          4Q15K
          token[*2] load[*1] tad[*2] IMUL Y|A, /
          load[*1] PUSH R|A Y, load[*2] POP R|A A3, IMUL Y|A A3,;

        divo [-,-] => token[*2] load[*1] tad[*2] IDIV Y|A, /          4Q15K
          load[*2] PUSH R|A Y, load[*1] POP R|A A2, IDIV Y|A A2,;

        valnod [-] =>                                                 4Q15K
          (?@ defneo *1:1 [?NOT ?@ valrul *1:1
             <*1:1 " not value rule " LOC LINE>] /
             @S valref *1:1)
          *1 CERR, negy[];

      % Assembly code. % %+ASM%
        instr = defck (                                               4Q15I
          regstr (*, address index :inst[5] / index :inst[3])/
          :=0 address index :inst[5]) *; ;
        regstr = (.UID defck / .NUM);                                 4Q15I
        address = (*@ :imed[=287] / :mt[])                            4Q15I
          (*= adr :adlit[1] / adr :adr1[1]);
        index = *( regstr *);                                         4Q15I
        adr = (.ID / *- liter :negadr[1] / regstr / .SR1);            4Q15I
        liter =  .NUM :imed[1];                                       4Q15I

        inst [-,-,-] => *V1|OP +reg[*2] +reg[*3]|XR,;                 4Q15I
          [-,-,-,-,-] => *V1|OP +reg[*2]|A *3 +reg[*4]|XR,;

        negadr [-] => MASK(-*N1:1)18M;                                4Q15I

        adr1 [.ID]                                                    4Q15L
          [.UID] => *V1;
          [negadr[-]] => *1;
          [-] => +reg[*1];

        adlit [-] => =adr[*1];                                        4Q15L

        reg                                                           4Q15L
          [.UID] := *V1;
          [-] := *N1; %+ASM%
END of TREE META
;RUNFIL to load tree meta on tops-20 (l10,meta.rur,)
comm l10
<arcsubsys>l10ldr
/s
/m
/400010o
libe9
meta9
minil10data
minil10runtime
sys:monsym
<ESC>
ddt
initll<ESC>cmetop.txt
<ESC>
sav <l10>meta5.exe 400 554
```

```
del metop.rel
;
```

L10

```
FILE l10   % (arcsubsys,meta9,)  to  (l10,l10,)  %
   CHECK
   META program
      SET: % note, S=15 %
         P=7 %A=8 RP=9 A1=10 A2=11 A3=12 A4=13 M=14 S=15
         MREG=6 R1=1 R2=2
         SEGR=5 % segment base register %
         L=1 F=2 LE=3 GE=5 N=6 G=7 I=2B7
         IDFLG=11B6
         DESCR=101 NEWDESCR=100 LSTRNG=3 LINTEG=2 LNULL=0 LLIST=4 %LIST types %
         LMOVE=2 LCOPY=4 LELEM=8 LALL=1 % bits for apsubl arg %
         ;
      DUMMY:
         pnam % hash of current procedure or coroutine name %
         actf argn % (act) arg checking varriables %
         lllists % "local list list" - list of lists to be nulled (label) %
         lstyp lindx lstmt % used by LIST stuff (lstor) (leledo) %
         lsb lele listnm % dummys for LIST stuff %
         disj conj not skip addr adsr con minus ec inalt date instat intemp
         recloc % ugly save for lc in (rechd) %
         dfport % node for default PORT ID %
         val % temp cell for a vlaue %
         val2 % "" %
         resltn % number of results %
         actaln % number of actual args %
         bcount % count of 'blank' cells when defining locals %
         astroom % for astring room declare %
         declst % list of declared values %
         recbit % record field displacement %
         recdsp % record field displacement (bits to right ) %
         recref % dummy rule for recptr node %
         boolean lvar gvar gvarindexed fof incbyt rvar rvarindexed indir pbound
         btimes clltest cexp fgtest sf se astest ident notele rcon;
      ATTRIBUTES:
         local constant waslocal litsr ref ptr register
         recnam % symbol is record name %
         recptr % symbol is record pointer %
         llist % symbol is local list %
         import export % IMPORTed and EXPORTed symbols %
         ;
      FLAGS:
         caflg rchsh rchlc rccnt idhash ut decf islab pshcnt mrcnt first isund
         sysfg % true if ! syntax allowed %
         ctime % true while compiling compiletime expression %
         coroutine % true while compiling coroutine %
         openok % true if OPENPORT is legal %
         catchf % true when in catchphrase %
         listpres % true if routine delcared list(s) %
         stkok % TRUE if code produced to check S overflow %
         segflag % TRUE if compiling segment %
         nomess undok skipcll;
      FIELDS:
         A=[4:23] X=[4:18] OP=[9:27] MOD=[3:27] POS=[6:30] SIZ=[6:24]
         LH=[18:18]
         ;
      OPCODES:
```

```
    AOBJP=252B AOBJN=253B
    ILDB=134B LDB=135B IDPB=136B DPB=137B
    MOVE=200B MOVEI=201B MOVEM=202B MOVN=210B MOVNI=211B MOVNM=212B
    MOVNS=213B IMUL=220B IMULI=221B IMULB=223B IDIV=230B IDIVI=231B
    IDIVB=233B LSH=242B EXCH=250B JSYS=104B HRRM=542B;
  OPCODES:
    JRST=254B XCT=256B PUSHJ=260B PUSH=261B POP=262B POPJ=263B
    JSP=265B ADD=270B ADDI=271B ADDM=272B ADDB=273B SUB=274B
    SUBB=277B SUBI=275B;
  OPCODES:
    CAI=300B CAIE=302B CAIN=306B CAM=310B CAME=312B JUMP=320B JUMPG=327B
    JUMPE=322B CAML=311B
    JUMPN=326B SKIP=330B SKIPE=332B SKIPN=336B AOJ=340B AOS=350B
    SOJ=360B SOS=370B SETZM=402B SETZB=403B SETOM=476B SETOB=477B AND=404B
    ANDI=405B ANDB=407B XOR=430B XORB=433B IOR=434B IORB=437B HRLM=506B
    HRLZI=515B
    IMMEDIATE=1;
  ERROR: ->
    (?F caflg / "END." &FR ctime.catchf :xreset[]* $parts "FINIS")
    :fin[]*;
  SIZE: S=5000 N=1400 M=400 K=800 L=750 G=60;

  stat =                                                              1B
    catchphrase /
    simple* &FR ut / utr* &FS ut /
    special* &FR ut &resetall / otherstat;
catchphrase =                                                         1B
    cathed @S local catbody ;                                         1B1
xcatch =                                                              1B
    "CATCHPHRASE" @S extern catbody [*;];
catbody =                                                             1B
    &FS catchf
    %G#1 (?F ut / :ljump[#1]* )
    ( *( mresults *) / :mt[] )
    *; :catch[2]*
    stat :catch[#1]* &FR catchf , ut;                                 1B1
cathed =                                                              1B
    _ *( .ID *) "CATCHPHRASE" &LABEL ;
catch % produce catchphrase code %                                    1B
    [-,-] => % id, mresults %
        >*1 % define local label %
        *2 % store results % ;
    [#1] => % end of catchphrase %
        pcallp[gvar["syscon"],mt[],mt[]] % continue %
        >#1;
otherstat =                                                           1B
    ("LOOP" loop / "WHILE" while /
      "UNTIL" until / "DO" do / "FOR" for /
      "IF" if / "CASE" case /  "DIV" divid /
      "OPEN" "PORT" openport /
      "INVOKE" invoke* /
      "DROP" [?F catchf  <"DO NOT use DROP in a Catchphrase" LOC> ]
        *( :gvar["sysdrp"] (
          "ALL" *) :act[con[=-1]] :fcnref[2,mt[]]* /
          sigable* ) /
      "ENABLE" *( :gvar["sysena"] sigable* /
```

```
        "DISABLE" *( :gvar["sysdis"] sigable*
        ) &FR ut /
      "BEGIN" block / ["NULL"] [?F islab &FR ut];

   invoke = % invoke catchphrase %                                          1B
      *( :gvar["sysinv"] name1 :load[1,A1]
      (
         *, (
            name1 /                                                       1B199
            "RETURN" [?F coroutine :syntx[" ","RETURN not allowed in
            coroutine"]* ] (
               "[FALSE]" :gvar["sysrtf"] /
               "[TRUE]" :gvar["sysrtt"] /
               :gvar["sysrtt"] ) :loadad[1] /
            :gvar["syster"] :loadad[1] ) :load[1,R1]
            ( *, exp / :con[=0] ) :load[1,R2] /
         :gvar["syster"] :loadad[1] :load[1,R1] :load[con[=0],R2] )
      *)
      :loclist[3] :pcallp[2,mt[]];
   sigable = % enable/disable signal %                                      1B
      name1 *) :act[1] :fcnref[2,mt[]];                                    1B2(
   name1 = % local label to be passed as arg %                              1B
      .ID :gvar[1] :loadad[1];
   catchk = % in catchphrase? %                                             1B
      ( ?F catchf /
         :syntx[" ","statement allowed only in catchphrase"]* );
   notcatch = % not allowed in catchphrse %                                 1B
      [ ?F catchf :syntx[" ","statement not allowed in catchphrase"]*];
   fcn1 = % procedure call, no args, no results %                           1B
      :fcnref[1,mt[],mt[]];
   openport =                                                               1B
      [ ?F openok /
         :syntx[" ","OPENPORT not allowed in coroutines"] ]
      lhs (                                                                1B2
         '_ fwlhs *( args *) :openp[4]* /
         *( args *) :openp[3]*  );
   openp % open port code produced here %                                   1B
      [-,-,-,-] => % assign, coroutine name, args, results %
         openp[*2,*3,*4]
         store[*1,rgcon[A1]] % store 1st result %
         :
      [-,-,-] => % with no assign %
         MOVEI A4|A G#1 [?F segflag segx[] ],                              1B26
            % compute address of #1 %
         MOVEM A4|A M|X 1, acreset[A4] % setup co-return %                 1B26
         *2 % push args %
         loadad[*1,A1] % coroutine address %
         MOVE MREG|A M, % this port id as meta-arg %                       1B26
         PUSHJ S|A A1|X 1, % the call %                                    1B26
         >#1 % define co-return location %
         &resetall % unknown reg contents %
         *3 % store results %
         :
    loop =  >#1 &G#2 &lpush labeled :ljump[#1]* >#2 &lpop;                 1B

    while = exp whil1;                                                     1B
```

```
until = exp :boolean[not[1]] whil1;                                    1B3

   whil1 =                                                             1B3
     "DO" >#1 &G#2 &lpush :brf[1,#2]* &lpbxit
     labeled :ljump[#1]* >#2 &lpop;

do =                                                                   1B3
  >#1 &G#2 &lpush labeled
  ("WHILE" exp / "UNTIL" exp :not[1] :boolean[1])
  :brt[1,#1]* &lpdxit >#2 &lpop;

for =                                                                  1B3
  lhs  [ '_ exp :store[2] ]
  ("UP" :=1 / "DOWN" :=0) (exp / :con[=1]) "UNTIL"
  ('= :E / '# :N / ">=" :GE / "<=" :LE / '> :G / '< :L) exp
  "DO" &G#1 &G#2 &pushr &lpush &popr :dofor[5,#1,#2]*
  &lpbxit labeled :ljump[#1]* >#2 &lpop;

   dofor
     [store,-,-,-,-,-,-] => %store,dir,inc,rel,bound,repeat,exit%
         simstat[*1] dofor[*1:1,*2,*3,*4,*5,*6,*7];                    1B34G1
     [-,-,-,-,-,#1,-] => %lhs,dir,inc,rel,bound,repeat,exit%
        &pushr ljump[G#2] >#1 &resetall forstep[*1,*2,*3]
        >#2 &compr btrue[logical[*1,*5,*4],*7];
   forstep  %lhs,direction,inc%                                        1B3
     [-,1,-] => simstat[store[*1,addnode[*1,*3]]];
     [-,0,-] => simstat[store[*1,subnode[*1,*3]]];

if =                                                                   1B3
  exp "THEN" :if1[1,G#1]* &FR ut labeled
  ("ELSE" :if2[#1,G#2]* &FR ut labeled :if3[#2]* / :if3[#1]*);

   if1 [-,-] => brf[*1,*2] &pushb;                                     1B3
     %push what the registers will be if branch over the true
     part%

   if2 [#1,#2] => (?F ut / ljump[#2]) &xchr >#1;                       1B3
     %the call to xchr stacks what know about the registers when
     exit from the true part and sets the registers to what know
     when enter the false part%

   if3 [#1] => >#1 (?F ut &popr / &compr);                             1B3
     %if there was an ELSE part, then stack contains what know
     when exit true part (by if2) and registers contain what
     know when exit false part.  If ended false part with an
     unconditional branch, then restore registers from stack,
     else compare registers from the two parts.  If there was
     no ELSE part then stack contains what in registers if
     branched over the true part (by if1) and registers contain
     what know when exit the true part.  If the true part ended
     in an unconditional transfer, then restore registers from
     stack, else compare registers.%

case = >#1 &cstrt cstat;                                               1B
```

```
    cstat =                                                              1B37
      exp :load[1,A1]* "OF" >#1 &G#2 &coush
      $(-'E casest [';]) "ENDCASE" &FR ut labeled &cpop >#2 ;

    casest =                                                             1B37
      label casehead :brf[1,G#1]* &oushb &FR ut labeled
      :[?F ut / &lplev _ 1 &cgeta2 JRST #2, &caseck] >#1 &copr;

    casehead =                                                           1B37
      :mt[] binrel $(', :mt[] binrel :disj[2] :boolean[1]) ':;

  utr = %unconditional transfer%                                         1B3
    (
      "RETURN"
        [ ?F coroutine :syntx[" ","RETURN not allowed in coroutine"]* ]
        result :crtn[2] /
      "EXIT"
        ("CASE" (.NUM / :=1) :xitcase[1] /
        ["LOOP"] (.NUM / :=1) :xit[1]) /
      "REPEAT"
        ("LOOP" (.NUM / :=1) :rptloop[1] /
        ["CASE"] (.NUM / :=1) oneval :rpt[2]) /
      "GO""TO" fwlhs :cgoto[1]
      ) notcatch /
    "ABORT" :gvar["sysabt"] hcall2 /
    "CONTINUE" catchk :gvar["syscon"] :hcallp[1,mt[],mt[]] /
    "TERMINATE" catchk :gvar["systrm"] fcn1 /
    "RESUME" catchk :gvar["sysres"] hcall2 ;

    result =                                                             1B38
      ( metarg / :load[con[=1],MREG] % Success % )
      (
        '( exp :load[1,A1] &resltn _ R1
          .$('; exp :load[1,=resltn] &resltn _ resltn+1 )
          ') :lodlist[1,$] /
        :mt[] );
    metarg = % 'meta' argument, success/fail %                           1B38
      '[ expression '] :load[1,MREG] ;
    oneval = '( exp ') / :mt[];                                          1B38
    cgoto                                                                1B38
      [lvar] => JRST *V1:1 [?F segtlag segx[] ],; % oper will index by %
      %
      [-] => oper[*1,JRST,=0];
    crtn                                                                 1B38
      [load[con[1],-],-] =>
        *2 JRST ( ?@ export *pnam 'syssrt' / 'sysrtt' ) ,;
      [load[con[0],-],-] =>
        *2 JRST ( ?@ export *pnam 'syssrf' / 'sysrtf' ) ,;
      [-,-] =>
        *2 *1 JRST ( ?@ export *pnam 'syssrn' / 'sysrtn' ) ,;
    cklrtn % check for local lists on return %                           1B38
      =>
        fcnref[gvar["sysfll"],act[#lllists],mt[]]
        &FS listpres % set this if NULL-LISTS used % ;
    lodlist % load list of results in A1, R1... %                        1B3
      [$] => &FS isund % borrow a flag %
```

```
        !jsysrg( &jsysrg _ 0
        $( ?F isuno &FR isuno *$ &acpush / *$ &jsysrg _ jsysrg+1 )
           % first one loads A1: save contents! %
           )
        &acpop:
      xit [-] => &lplev _ *N1  &lgetg2  JRST #2, &lpexit;                    1838
      xitcasc [-] => &lplev _ *N1  &cgetc2  JRST #2, &casxit;                1838
      rptloop [-] => &lplev _ *N1  &lgetg1  JRST #1,;                        1838
      rpt                                                                    1838
        [-,mt] => &lplev _ *N1  &cgetsc  JRST #1,;
        [-,-] => load[*2,A1]  &lplev _ *N1  &cgetst  JRST #1,;

  block = labeled $(*; labeled) "END";                                       183
 simple =                                                                    184
    sims / "CALL" procall / "BUMP" bump /
    "PCALL" notcatch pcall2 /
    "HELP" :gvar["syshlp"] hcall /
    "NOTE" :gvar["sysnot"] hcall2 /
    "NULL-LISTS"
       [ ?lllists=0 :syntx[" ","NULL-LISTS only inside routines"]* ]
       :cklrtn[] /
    *!
       (?F sysfg /
          <"syscall/assembly not allowed in this file"> &xerror )
       (.ID jsysx / builtin) /
    stkrng;

  sims =                                                                     184
    %simple assignment or call%
       lhs tail :simstat[1] /
    %multiple assignment%
       *( lhs $(*, lhs :popmult[2]) *) *_
       *( exp $(*, exp :pushmult[2]) *) :massgn[2];

    massgn [-,-] =>                                                          184
      &pshcnt _ 0 pushlist[*2] poplist[*1]
      (? pshcnt = 0 / <"Sides unequal" LOC> &xerror);

    pushlist                                                                 184
      [pushmult] => *1;
      [-] => load[*1,A1];
    pushmult                                                                 184
      [pushmult,-] => *1 pushit[*2];
      [-,-] => pushit[*1] pushit[*2];
    pushit [-] => push[*1,S] &pshcnt _ pshcnt+1;                             184
    poplist                                                                  184
      [popmult] => *1;
      [-] => simstat[store[*1,mt[]]];
    popmult                                                                  184
      [popmult,-] => popit[*2] *1 &pshcnt _ pshcnt-1;
      [-,-] => popit[*2] popit[*1] &pshcnt _ pshcnt-2;
    popit [-] =>                                                             184
      oper[*1,POP,S] modify[*1] /
      POP S|A ac, acreset[ac] store[*1,mt[]];

  lhs = (fwlhs / pwlhs) $qualifiers;                                         18
```

```
    fwlhs = %full word left hand side%                                  1843
      (.ID / *< link)
         (
            ".SIZE" % record size notation % (
               ?@ recnam &val_*M1 %DISCARD :con[=val] /
               ?@ recptr &val_*M1 %DISCARD :gvar[=val+IDFLG] /
               ?@ defned :syntx[1," not a record designator"] /
               :gvar[1] % assume undefined record name % ) /
            ?@ constant
               &val _ *V1 &DISCARD :con[val] /
            ?@ ptr reftype
               [( *[ indexexp :indir[addnode[2]] /
               (&ptrklg _ *N1) :indir[1]
                  (qualifier / &DISCARD :ptrklg) )] /
            ?@ ref reftype [*[ indexexp :addnode[2]] :indir[1] /
            ?@ local
               (*[ :addr[1] indexexp :indir[addnode[2]] / :lvar[1]) /
            ?@ register (*[ indexexp :rvarindexed[2] / :rvar[1])
            /
            ?@ waslocal :locerr[1]* /
            %global% (*[ indexexp :gvarindexed[2] / :gvar[1]) ) /
      *[ indexexp [ *[ indexexp :addnode[2] ] :indir[1] /
      *% .ID (?@ ref / ?@ ptr / :syntx[1," not REF or POINTER"]*) reftype
      /
      _ .UID ?@ register syschk
         (*[ indexexp :rvarindexed[2] / :rvar[1]) ;

    locerr [-] => <*1 " was local " LOC> &xerror;                       1843
   syntx % syntax error message routine %                               1843
      [-,-] => <"ERROR: "*1 *2 LOC> &xerror;

    pwlhs = %part word left hand side%                                  1843
      *. .ID :for[mt[],1] /
      *| .ID :incbyt[1] /
      _ *@ twlhs *@ *[ exp *] :ch[2];

    reftype =                                                           1843
      ?@ local :lvar[1] / ?@ waslocal :locerr[1]* /
      ?@ register :rvar[1] / :gvar[1];

    qualifiers =                                                        1843
      _ *. (
      .ID  :fof[2] /
      (*M / "LH" )   :fof[1,"sysm"]/
      (*L / "RH" )   :fof[1,"sysl"] );

    link =                                                              1843
      [.UID @S noddt &DISCARD] [*,] [.UID @S noddt &DISCARD]
      [*,] .ID (*> / *: -> *>);

    indexexp = _ sum *] / exp *];                                       1843

  tail =                                                                184
    *_ exp :store[2] /
    ":=" exp :exchange[2] /
```

```
      *( args :fcnref[3] catch1;

    args = arglist mresults ;                                          154
   catch1 = % invoke catchphrase over 1 call %                         154
      *: name1 ( *, *, exp / :con[=0] ) *) :invok1[3] /
      *) :
   invok1 % invoke *2, do *1, drop %                                   154
      [-,-,-] =>
         pcallp[gvar["sysinv"],
            lodlist[ load[*2,A1],
               load[G#1,R1], load[*3,R2]],
            mt[] ] % invoke %
         *1 % the call or whatever %
         >#1 % terminate goes here %
         PUSH S|A A1,                                                  1B47A
         fcnref[gvar["sysdrp"],act[*2],mt[]]
         POP S|A A1, ;                                                 1B47A

       arglist = exp .$(*, arg) :act[1,$] / :mt[];                     1B48
       mresults =                                                      1B48
         *:
            [?F skipcll <"no result syntax allowed here " LOC> &xerror]
            &resltn _ R1
            ( metares / :mt[] ) [*,]
            .$<*,>( lhs :store[1,rgcon[resltn]] &resltn _ resltn+1 )
            :mr[1,$]
            [?resltn>4 :syntx[" ","too many results"]*] /
         :mt[];
       act                                                             1B48
         [-] => push[*1,S] ;
         [$] =>
            !actf,argn( &FS actf &argn _ 0
               $( (actok[*$] / &FR actf) &argn_argn+1 )
               (
                  ?F actf $_( push[*$,S] ) /
                  ADD S|A =argn+LSH(argn)18,
                  &argn _ 0
                  $( load[*$] MOVEM acnum|A S|X MASK(-argn)18M,
                     &argn_argn+1 )
                  )
               );
      actrev % always do actual args in reverse for LIST stuff %
         [$] => $_( push[*$,S] ) ;
      actok % is actual arg OK? not if store, etc. %                   1B48
         [store] [exchange] [fcnref] [invok1] [pcallp] [jsyscll] => &FAIL;
         [?[$]] => % check sub structure %
            !actf( &FS actf $*1( actok[*$] / &FR actf) &val_actf)
               (?F val / &FAIL);
         [-] => TRUE; % terminal nodes OK %
       arg = _ sum (+*) / +*, / +*:) / exp / :con[=0];                 1B48
       mr % store multiple results %                                   1B48
         [$] =>
            &acpush % save A1 %
            % this is not adequate - if proc call to evaluate lhs %
            $( *$ )
            &acpop ;
```

```
    mrcount                                                              184
      [popmult] => mrcount[*1:1] &mrcnt _ mrcnt+1;
      [-] => &mrcnt _ 1;

  pushac                                                                 18
    [incir] => push[*1:1,S];
    [-] => loadad[*1,ac] p[] ac,;
  push %exp,ac%                                                          18
    [addr,-] =>
      ?& local *1:1 loadad[lvar[*1:1],ac] PUSH *N2|A ac, /
      ?F segflag load[*1,ac] PUSH *N2|A ac, /
      PUSH *N2|A =*V1:1,;
    [con,-] => PUSH *N2|A =*N1:1,;
    [genint,-] [genstr,-] [ldmove,-] [lmove,-] [ldcopy,-]
    [lcopy,-] [genlst,-] [ltype,-] [lvcopy,-] [lstnul,-] =>
      *1 % they do the push %;
    [-,-] => oper[*1,PUSH,*2] / adrget[*1] PUSH *N2|A,;

  procall =                                                              18
    fwlhs ('( args :fcnref[3] catch1 / :mt[] :mt[] :fcnref[3]);

  jsyscll = ( '( args ') / :mt[] :mt[]) :jsysref[3];                     18
  jsysx = jsyscll :load[1]; % simple jsys - avoids load into a1 %        18
  fcnref                                                                 18
    [-,mt,-] => %lhs,no args,results%
      &savreg oper[*1,PUSHJ,S] fcnres[*3];
    [-,-,-] => %lhs,args,results%
      &savreg   *2
      oper[*1,PUSHJ,S] cntargs[*2]
      SUB S|A =pshcnt+LSH(pshcnt)18,
      fcnres[*3];

    cntargs                                                              185
      [actrev[$]]
      [act[$]] =>
        &pshcnt _ 0  $*1(&pshcnt _ pshcnt+1);
      [mt] =>
        &pshcnt _ 0 ;
    p => PUSH S|A;                                                       185
    cll [-] => PUSHJ S|A [?@ ref *1 I] loadr[*1] &resetall;              185
    cll1 [-] => cll[*1] SUB S|A =1000001B,;                              185
    cllm [-,-] => cll[*2] SUB S|A =*N1+LSH(*N1)18,;                      185
    jspto [-] => JSP A4|A *V1, &resetall;                                185
    fcnres [-] => &resetall *1 %mr or mt% &resreg;                       185

  jsysref % jsys call %                                                  18
    [-,-,-] => % lhs, args, results %
      &savreg !jsysrg( % push jsysrg on a stack for nested calls %
        &jsysrg_0
        $*2( load[*$,=jsysrg+1] &jsysrg_jsysrg+1)
        [ ? jsysrg>8 <"8 args max " LOC> &xerror]
        JSYS *V1, &resetall jsysrtn[*3] )
      &jrstreg;
  jsysrtn  % handles results %                                           18
    [mt] => TRUE;
    [mr[-]] => mrcount[*1:1] &jsysrg_mrcnt jsysr[*1:1];
```

```
jsysr                                                                   185
  [popmult] => jsysr[*1:2] jsysr[*1:1]; %reverse order%
  [fof] [incbyt] [ch] =>
     store[*1,rgcon[jsysrg]] &jsysrg_jsysrg-1;
  [-] => oper[*1,MOVEM,jsysrg] &jsysrg_jsysrg-1;
simstat                                                                 185
  %This is for stores that are not restricted as to which
  register use to evaluate.  Leave the global variable acnum
  designating the register that is used.%

  [store[rvar,-]] => regload[*1:1,*1:2] &acnum _ *V1:1:1;
  [store[-,addnode[-,con[1]]]] =>
     maybmp[*1:1,*1:2:1,AOS,=1] / sass[*1];
  [store[-,subnode[-,con[1]]]] =>
     maybmp[*1:1,*1:2:1,SOS,=1] / sass[*1];
  [-] => sass[*1];

   maybmp [-,-,-,-] => % lhs,exp,op,flag %                              1858
     samvar[*1,*2] bmpit[*1,*3,*4] / &FAIL;
   sass                                                                 1858
     [store[fof,-]] =>
        stoken[*1:1:1] load[*1:2]
           [? acnum = RP MOVE ac|A RP, acreset[ac] &acnum _ ac]
           stfof[*1:1,acnum] /
        doset[*1];
     [store[incbyt,-]] =>
        load[*1:2] IDPB acnum|A doadr[*1:1:1] mdfy[*1:1:1];
     [store[-,rvar]] =>
        oper[*1:1,MOVEM,=*V1:2:1] &acnum _ *V1:2:1 modify[*1:1] /
        sass2[*1];
     [store[-,lvar]]
     [store[-,gvar]] =>
        svset[*1:2:1] (
        ? &isinac ?AND stoken[*1:1]
           oper[*1:1,MOVEM,acnum] modify[*1:1] /
        sass2[*1]);
     [-] => sass2[*1];
   svset [-] => &actype _ acvar  &acval _ HASH(*1);                     1858
   sass2                                                                1858
     [store[lvar,-]]
     [store[gvar,-]] =>
        onereg[*1:2] svset[*1:1:1]
           &pickac &rgstrt *1 &rgend /
        doset[*1];
     [-] =>
        stoken[*1:1] ?AND onereg[*1:2]
           &actype _ acmoty  &acval _ 0  &pickac
           &rgstrt *1 &rgend /
        doset[*1];
   stfof
     [-,-] =>
     stfof2[*1,*2] /
     loadad[*1:1,RP] DPB *N2|A doadr[*1:2] &acnum _ *N2;
   stfof2                                                               1858
     [fof[-,"sysl"],-] =>token[*1:1] oper[*1:1,HRRM,=*N2] &acnum_*N2
     modify[*1:1] / &FAIL;
```

```
        [fof[-,"sysm"],-] =>token[*1:1] oper[*1:1,HRLM,=*N2] &acnum_*N2
          modify[*1:1] / &FAIL;
        coset [-] => *1 &acnum _ ac;                                    1858

    regload  %rvar,exp%                                                 185
      [-,?[rvar[*1:1],-]] => rgld[*1,*2];
      [-,-] => load[*2,=*V1:1];

       rgld                                                             1859
         [-,andnode] => regop[*1,*2:2,AND];
         [-,iornode] => regop[*1,*2:2,IOR];
         [-,xornode] => regop[*1,*2:2,XOR];
         [-,addnode] => regop[*1,*2:2,ADD];
         [-,subnode] => regop[*1,*2:2,SUB];
         [-,mulnode] => regop[*1,*2:2,IMUL];
         [-,-] => load[*2,=*V1:1];
       regop [-,-,-] => %rvar,exp,op%                                   1859
         doadc[*2,*3,=*V1:1] /
         oper[*2,*3,=*V1:1] /
         adrget[*2] *N3|OP *V1:1|A,;

    store %lhs,exp%                                                     186
      [fof[fof,-],-] [fof[incbyt,-],-] =>
         load[*2,ac] &pac fst1[*1:1] MOVEI RP|A ac, acreset[RP] &rpac
         DPB accum|A doadr[*1:2] fst2[*1:1];
      [fof,-] =>
         fofssmp[*1,*2] /
         load[*2,ac] &acpush loadad[*1:1,RP] &acpop
         DPB ac|A doadr[*1:2];
      [incbyt,-] => load[*2,ac] IDPB ac|A doadr[*1:1] mdfy[*1:1];
      [ch,-] =>
         &savreg push[*1:2,S] pushad[*1:1] push[*2,S]
         cllm[=3,"repchr"] &resreg &resetall;
      [rgcon,-] =>
         load[*2,=*N1:1];
      [rvar,-] =>
         load[*2,=*V1:1];
      [con,-] => syntx[" ","attempt to store into constant"];
      [-,addnode[-,con[1]]] =>
         maybmp[*1,*2:1,AOS,=0] / strcont[*1,*2];
      [-,subnode[-,con[1]]] =>
         maybmp[*1,*2:1,SOS,=0] / strcont[*1,*2];
      [-,-] => strcont[*1,*2];
    fofssmp  % simple partial word store %                              186
      [fof[-,"sysl"],-] =>
         token[*1:1] load[*2,ac] oper[*1:1,HRRM,ac] modify[*1:1]/ &FAIL;
      [fof[-,"sysm"],-] =>
         token[*1:1] load[*2,ac] oper[*1:1,HRLM,ac] modify[*1:1]/ &FAIL;
    strcont                                                             186
      [-,subnode] =>
         samvar[*1,*2:2] doassign[*1,*2:1,SUBB] /
         samvar[*1,*2:1] doassign[*1,minus[*2:2],ADDB] /
         doassign[*1,*2,MOVEM];
      [-,jsysref] => load[*2,=1] oper[*1,MOVEM,=1];
      [-,?[-,-]] =>
         samvar[*1,*2:1] str[*1,*2] /
```

```
    ( ?[-,store[-,rgcon]] / ?[-,store[-,rvar]] )
       *2 store[*1,*2:1] / % use destination of first store %
          % still wrong for x_ |y _ reg and x_ [geta()] _ reg %
    doassign[*1,*2,MOVEM];
[-,con[0]] => doassign[*1,mt[],SETZB];
[-,minus[con[1]]] => doassign[*1,mt[],SETOB];
[-,minus] =>
    samvar[*1,*2:1] doassign[*1,mt[],MOVNS] /
    doassign[*1,*2,MOVEM];
[-,rgcon] => oper[*1,MOVEM,=*N2:1] stoacc[*1,=*N2:1];
[-,rvar] => oper[*1,MOVEM,=*V2:1] stoacc[*1,=*V2:1];
[-,-] => doassign[*1,*2,MOVEM];

fst1                                                              186
  [fof[fof,-]] [fof[incbyt,-]] =>
     fst1[*1:1] MOVEI RP|A ac, acreset[RP] &acpush
     LDB ac|A doadr[*1:2] acreset[ac];
  [incbyt] =>
     &acpush ILDB ac|A doadr[*1:1] acreset[ac] mdfy[*1:1];
  [fof] =>
     &acpush loadad[*1:1,RP] LDB ac|A doadr[*1:2] acreset[ac]
     (stoken[*1:1] / PUSH S|A RP,);
fst2                                                              186
  [fof[fof,-]] [fof[incbyt,-]] =>
     &acpop MOVEI RP|A ac, acreset[RP] &acpush
     DPB ac|A doadr[*1:2] &acpop fst2[*1:1];
  [incbyt] => DPB ac|A doadr[*1:1] &acpop;
  [fof] =>
     (stoken[*1:1] loadad[*1:1,RP] / POP S|A RP, acreset[RP])
     DPB ac|A doadr[*1:2] &acpop;
samvar                                                            186
  [mt,mt]
  [lvar,lvar[*1:1]]
  [gvar,gvar[*1:1]]
  [con,con[*1:1]]
  [minus[con],minus[con[*1:1:1]]]
  [addr,addr[*1:1]] => TRUE;
  [indir,indir] => samvar[*1:1,*2:1] / &FAIL;
  [gvarindexed,gvarindexed[*1:1,-]]
  [rvarindexed,rvarindexed[*1:1,-]] =>
     samvar[*1:2,*2:2] / &FAIL;
  [fof,fof[-,*1:2]] => samvar[*1:1,*2:1] / &FAIL;
  [-,-] => sameop[*1,*2] / &FAIL;
sameop                                                            186
  [addnode,addnode]
  [subnode,subnode]
  [mulnode,mulnode]
  [divnode,divnode]
  [andnode,andnode]
  [iornode,iornode]
  [xornode,xornode] => bothsame[*1,*2] / &FAIL;
bothsame [-,-] =>                                                 186
  samvar[*1:1,*2:1] ?AND samvar[*1:2,*2:2] / &FAIL;
str                                                               186
  [-,andnode] => doassign[*1,*2:2,ANDB];
  [-,iornode] => doassign[*1,*2:2,IORB];
```

```
      [-,xornode] => doassign[*1,*2:2,XORB];
      [-,addnode] => doassign[*1,*2:2,ADDB];
      [-,mulnode] => doassign[*1,*2:2,IMULB];
      [-,-] => doassign[*1,*2,MOVEM];
    doassign [-,-,-] => %lhs,tail,op%                                    1862
      load[*2,ac] oper[*1,*3,ac] stoacc[*1,ac];
    stoacc %have stored acc(*2) into *1%                                 1863
      [lvar,-]
      [gvar,-] => stoac1[*1:1,*2];
      [-,-] => acreset[*2];
    stoac1 [-,-] =>                                                      1863
      &actype _ acvar  &acval _ HASH(*1)
      &acnum _ *N2  &stoac;

  exchange [-,-] => %lhs,tail%                                           1865
    token[*1] load[*2,ac] oper[*1,EXCH,ac] modify[*1] acreset[ac] /
    load[*1,ac] &acpush simstat[store[*1,*2]] &acpop;

  builtin = .UID                                                         1866
    ((.ID / .UID / .NUM :con[1])
      ('', address :bltn[5] / index :bltn[3]) /
    address :bltn[4]);

    address = ('@ :indir[] / :mt[])                                      1866
      (adel / '= adel :eq[1] / :mt[]) index;
    index = '( (.ID / .UID / .NUM :con[1]) ') / :mt[];                   1866
    adel = .ID / .UID / '- literal :minus[1] / literal;                  1866
    bltn                                                                 1866
      [-,-,-,-,-] => %op,acc,indir,address,index%
        bltac[*2] bltn[*1,*3,*4,*5];
      [-,-,-,-] => %op,indir,address,index%
        bltindir[*2] bltn[*1,*3,*4];
      [-,-,-] => %op,address,index%
        check[*1] *V1|OP bltndx[*3] bltad[*2] &resetall;
    bltndx                                                               1866
      [mt] => TRUE;
      [.ID] => check[*1] *V1|X;
      [.UID] => check[*1] *V1|X;
      [con] => *N1:1|X;
    bltac                                                                1866
      [.ID] => check[*1] *V1|A;
      [.UID] => check[*1] *V1|A;
      [con] => *N1:1|A;
    bltindir                                                             1866
      [mt] => TRUE;
      [-] => I;
    check [-] =>                                                         1866
      ?@ defned *1
        [?@ reloca *1 <*1 " relocatable " LOC> &xerror] /
      <*1 " undefined " LOC> &xerror;
    bltad                                                                1866
      [.UID]
      [.ID] => doadr[*1];
      [eq[.ID]] => ?@ local *1:1 noloc[*1:1] / =*V1:1,;
      [con] => *N1:1,;
      [eq[con]] => =*N1:1:1,;
```

```
        [minus[con]] => MASK(-*N1:1:1)18M,;
        [eq[minus[con]]] => =-*N1:1:1:1,;
        [mt] => ,;
      doadr [-] =>                                                    1864
        ?% local *1 V|X MASK(*V1)18M,/
        ?% waslocal *1 locerr[*1]/
        ?% defrec *1 (
           *P1 ( ?% reloca *1 [?F segflag segx[]] \1 / , ) ) /
              % above omits bits from LH of value %
        *V1 [?F segflag segx[]],;

    stkrng =                                                          186
      "PUSH" fwlhs "ON" fwlhs :stkpsh[2] /
      "POP" fwlhs ("TO" fwlhs :stkpto[2] / :stkp[1]) /
      "RESET" fwlhs :stkrst[1];

      stkpsh [-,-] =>                                                 1865
        pushac[*2] pushad[*1] cllm[=2,"rcpsh"] &resetall;
      stkp [-] => pushad[*1] cll1["rpop"];                            1865
      stkpto [-,-] =>                                                 1865
        pushac[*2] pushac[*1] cllm[=2,"rcpto"] &resetall;
      stkrst [-] => pushad[*1] cll1["rstsk"];                         1865

    bump =                                                            186
      ("DOWN" lhs .$(*, lhs) :bmpdwn[1,$] /
      lhs .$(*, lhs) :bmp[1,$]);

      bmp [$] => $(bmpit[*$,AOS,=1]);                                 1866
      bmpdwn [$] => $(bmpit[*$,SOS,=1]);                              1866
      bmpit  %lhs,op,flag%                                            1866
        % if flag is true then use any register, else use ac %
        [lvar,-,-]
        [gvar,-,-] => bmpsv[*1:1,*2,*3];
        [rvar,-,-] =>
           (? *N2=AOS  AOJ / SOJ) *V1:1|A,
           [? *N3=0 load[*1,ac]] &acnum _ *V1:1;
        [-,-,-] =>
           oper[*1,*2,ac] acreset[ac] &acnum _ ac /
           (? *N2=AOS load[adonode[*1,con[=1]],ac] /
              load[subnode[*1,con[=1]],ac])
           store[*1,mt[]] &acnum _ ac;
      bmpsv [-,-,-] => %ic,op,flag%                                   1866
        &actype _ acvar &acval _ HASH(*1)
        (? *N3=0 &acnum _ ac / &pickac)
        *N2|OP acnum|A doadr[*1] &stoac;

    divid =                                                           186
      bitor */ bitor *, lhs %quotient% *, lhs %remainder% :dodiv[4]*;

      dodiv [-,-,-,-] =>                                              1867
        divnode[*1,*2] &acpush store[*4,mt[]] &acpop store[*3,mt[]];

    special =                                                         186
      "CCPOS" (
        pos /
        ** fwlhs ** (*[ exp *] / :con[=1]) :ch[2]) :ccpos[1] /
```

```
      "READC" (*( exp *) :readc[1] / :readc[]) /
      "FIND" pexp /
      "ST" (pair *_ plist :subscon[2] /
         pos *_ plist :sconstrn[2]) /
      *+ fwlhs *+ (
         *[ exp "TO" exp *] *_ plist :subsasgn[4] /
         *_ plist :sasgn[2])/
      *# fwlhs *# (
         *[ exp (
            "TO" exp *] *_ listrh :lsbsto[4] /
            *] *_ ( listele / :lstmt[] ) :lelesto[3] ) /
         "!_" listrh :lapnd[2] /
         *_ listrh :lstor[2] );

   listrh = % list assign right hand side %                          186
      +*; :lstnul[]
      / .#<*,> listseg :listrh[$] ;
   listseg = % list segment %                                        187
      "ELEM" listpart :lvcopy[1] /
      "COPY" ( listpart :lcopy[1] / "DESCR" exp :ldcopy[1] ) /
      "MOVE" listpart :lmove[1] /
      listpart :lvcopy[1] / % shorthand for ELEM %
      listele;
   listele = % list element %                                        187
      "NULL" :lstnul[] / % null element %
      "COPY" ( listref :lcopy[1] / "DESCR" exp :ldcopy[1] ) /
      "MOVE" listref :lmove[1] /
      "USE" (
         "DESCR" :syntx[" ","may not USE DESCR"]* /
         exp :ldmove[1]) /
      "LIST" *( listrh *) :genlst[1] /
      ( .SR :adsr[1] / *+ fwlhs *+ :loadadd[1] ) :genstr[1] /
      exp :genint[1];                                                1971
   listpart = % list or sublist designation %                        187
      *# fwlhs *# (
         *[ exp (
            *] :lele[2] /
            "TO" exp *] :lsb[3] ) /
         :listnm[1] );
   % LIST production rules %
      lelesto % list element store e.g. #list#[i] _ ... %           1873
         [-,-,lstmt] => % remove one element %
            lsbsto[*1,*2,*2,lstnul[]];
         [-,-,-] =>
            leledo[*1,*2,*3,=1];
      leledo % do list element operation %                          1873
         [-,-,-,-] => %list, index, RHside, replace? %
            fcnref[gvar["wrlelm"],
               actrev[loadadd[*1], % list %
                  ltype[], % type %
                  *3, % value %
                  con[=*N4], % replace flag %
                  *2], % index %
               mt[] ] % results %
            ;
      ltype % load list element type %                              1873
```

```
      => PUSH S|A =lstyp,;
   lcopy % put out code for list ref copy %                                  1B7
      [lele] => push[rdlelm[*1:1,=0,*1:2],S]
         &lstyp _ DESCR;
   ldcopy % COPY DESCR expression %                                          1B7
      [-] => push[*1,S]
         &lstyp _ NEWDESCR;
   lvcopy % list 'value' copy, same as lgval %                               1B7
      [lele] => rdlelm[*1:1,=0,*1:2] % push value %
         PUSH S|A R1, &lstyp _ LINTEG;                                       1B73F
   lmove % list element move %                                               1B7
      [lele] => rdlelm[*1:1,=1,*1:2]
         PUSH S|A A1, % push descr %                                         1B73G
         &lstyp _ NEWDESCR % BUT DON'T COPY ELEMENT % ;
   ldmove % USE DESCR expression syntax %                                    1B7
      [-] => push[*1,S]
         &lstyp _ NEWDESCR % BUT DON'T COPY %;
   genstr % generate string element for list %                               1B7
      [-] => push[*1,S]
         &lstyp _ LSTRNG;
   genint % generate integer string element for list %                       1B7
      [-] => push[*1,S] &lstyp _ LINTEG ;
   genlst % generate list for list element %                                 1B7
      [-] =>
         dolistrh[con[=0],*1]
         PUSH S|A A1, % push resulting address %                             1B73K
         &lstyp _ NEWDESCR;
   lstnul % generate null list element %                                     1B7
      => PUSH S|A =0, &lstyp _ LNULL;
   lgdescr % list - get DESCR **LOAD it** %                                  1B7
      [-] => rdlelm[*1:1,=0,*1:2] &acnum _ A1;
   lgval % list - get element value **LOAD it** %                            1B7
      [-] => rdlelm[*1:1,=0,*1:2] &acnum _ R1;
   rdlelm % read element %                                                   1B7
      [-,-,-] => fcnref[gvar["rdlelm"],
         actrev[loadad[*1], con[=*N2], *3],
         mt[] ];
   lstor % list store, general case %                                        1B7
      [-,-] => % list, rhside %
         leasy[*1,*2] % easy case? %
            fcnref[gvar["nulist"],
               actrev[loadad[*1], con[=0]], mt[]]
            ( ?[-,lstnul] % done %
            /
            &lindx _ 1
            $*2(leledo[*1,con[lindx],*$,=1]
               &lindx _ lindx+1 % append elements % ))
         / dolistrh[loadad[*1],*2] ;
   dolistrh % do list RH side using blc, elc %                               1B7
      [-,-] => % elc argument (loadad already done), RH side %
         fcnref[gvar["blc"], mt[], mt[]]
            $*2( elecon[*$] )
            % if storing in list, null it first %
               ( ?[con,-] /
                  fcnref[gvar["nulist"],
                     actrev[*1, con[=0]],
```

```
                    mt[]] )
            fcnref[gvar["elc"],
               actrev[*1], mt[] ];
leasy % check for each rh side %                                    187
   [-,lstnul] => TRUE;
   [-,listrh] => &val _ 0
      $*2( %val _ val + leasy1[*1,*$] ) [?val#0 %FAIL] ;
leasy1 % possible to do appends in place ? 0=TRUE %                 187
   [-,lcopy] [-,lvcopy] [-,lmove] := leasy2[*1,*2:1];
   [-,?[lele]] := samvar[*1,*2:1:1] 1 / 0;
   [-,lstnul] [-,ldmove] [-,ldcopy] [-,genlst] [-,genstr] [-,genint]
      := 0;
leasy2 % look for sublist, entire list %                            187
   [-,lsb] [-,listnm] := 1;
   [-,?[lele]] := samvar[*1,*2:1:1] 1 / 0;
   [-,-] := 0;
elecon % list element construction, use aplelm or apsubl %          187
   [lcopy[listnm]]
   [lcopy[lsb]] => apsubl[LCOPY,*1:1];
   [lvcopy[listnm]]
   [lvcopy[lsb]] => apsubl[LELEM,*1:1];
   [lmove[lsb]]
   [lmove[listnm]] => apsubl[LMOVE,*1:1];
   [lcopy[lele]] => aplelm[DESCR,rdlelm[*1:1:1,=0,*1:1:2]];
   [ldcopy] => aplelm[DESCR, *1:1];
   [lmove] => aplelm[NEWDESCR, rdlelm[*1:1:1, =1, *1:1:2]];
   [ldmove] => aplelm[NEWDESCR, *1:1];
   [genlst] => aplelm[NEWDESCR,
      dolistrh[con[=0], *1:1] ];
   [genstr] => aplelm[LSTRNG, *1:1];
   [genint] => aplelm[LINTEG, *1:1];
   [lvcopy] => aplelm[LINTEG, lgval[*1:1]];
apsubl % append sub list %                                          187
   [-,lsb] =>
      fcnref[gvar["apsubl"],
         actrev[con[=*N1],loadad[*2:1],*2:2,*2:3],
         mt[] ];
   [-,listnm] =>
      fcnref[gvar["apsubl"],
         actrev[con[=*N1+LALL], loadad[*2:1]],
         mt[] ];
aplelm % append list element to construction area %                 187
   [-,-] =>
      fcnref[gvar["aplelm"],
         actrev[con[=*N1], *2],
         mt[] ];
lapnd % list append %                                               187
   [-,lstnul] => TRUE;
   [-,-] => % list, RH side %
      leasy[*1,*2] ( % can just do it %
         $*2( leledo[*1, con[=0], *$, =0] ) )
      /
         fcnref[gvar["blc"],mt[], mt[]]
         apsubl[LMOVE,listnm[*1]] % move it %
         $*2( elecon[*$] ) % append %.
         fcnref[gvar["nulist"], actrev[loadad[*1], con[=0]], mt[]]
```

```
            fcnref[gvar["elc"], actrev[loadad[*1]], mt[] ];
      lsbsto % list sub list store %                                    1573
         [-,-,-,lstnul] => % list, e1, e2, rhside %
            fcnref[gvar["nulist"],
               actrev[loadad[*1], con[=1], *2,*3],
               mt[] ];
         [-,-,-,-] => fcnref[gvar["blc"], mt[], mt[]]
            apsubl[[LMOVE,lsb[*1,con[=1],subnode[*2,con[=1]]]]
               % move first portion %
            $*4( elecon[*$] ) % do it %
            apsubl[[LMOVE,lsb[*1,addnode[*3,con[=1]],fof[*1,"sysl"]]]
               % move end portion %
            fcnref[gvar["elc"], actrev[loadad[*1]], mt[] ];
   ccpos                                                                187
      [ch] =>
         setscn[] pushad[*1:1] cll1["asrref"]
         MOVEM A1|A *swork*, load[*1:2,A1]
         MOVEM A1|A *swork1*, fechc1[];
      [se] => fcp[*1];
      [-] => setscn[] fcp1[*1];

   plist = sspec $( *, sspec :sclist[2]);                               187
   sspec =                                                              187
      "NULL" :mt[] /
      .SR :cs[1] /
      "STRING" *( exp
         (*, exp
            ( *, exp :srmake[3] / :srmake[2,con[=0]] ) /
         :srmake[1,con[=10],con[=0]])
         *) /
      pair :cp[1] /
      *+ pair :cppx[1,=0] /
      ** fwlhs **
         (*[ exp ("TO" exp *] :chpair[3] / *] :ch[2]) / :cs[1]) /
      _ (*$ pair :cppw[1] / *- pair :cppx[1,=1]) /
      exp :cc[1];
   sconstrn                                                             187
      [-,-] =>
         PUSH S|A psid[*1] cll1["bsc"] *2 cll["esc"];
   subscon [-,-] =>                                                     187
      PUSH S|A psid[*1:1] cll1["bsc"] cp[pr[sf[*1:1],*1:1]] *2
      cp[pr[*1:2,se[*1:2]]] cll["esc"];
   subsasgn [-,-,-,-] =>                                                187
      pushad[*1] cll1["asrref"] PUSH S|A A1, cll1["bsc"]
      chpair[*1,con[=1],subnode[*2,con[=1]]] *4
      chpair[*1,addnode[*3,con[=1]],fof[*1,"sysl"]] cll["esc"];
   sasgn %lhs,sspec%                                                    188
      [-,-] =>
         srtest[*1,*2] simsrasgn[*1,*2] /
         pushad[*1] cll1["asrref"] PUSH S|A A1, cll1["bsc"]
            *2 cll["esc"];
   srtest                                                               188
      [-,sclist[sclist,-]] =>
         srtest[*1,*2:1] ?AND srisok[*1,*2:2] / &FAIL;
      [-,sclist] =>
         srok1[*1,*2:1] ?AND srisok[*1,*2:2] / &FAIL;
```

```
    [-,-] => srok1[*1,*2] / &FAIL;
  srok1                                                              188
    [-,mt] [-,ch] [-,cs] [-,cc] => TRUE;
    [-,chpair] => ?NOT samvar[*1,*2:1] / &FAIL;
  srisok                                                             188
    [-,mt] [-,cc] => TRUE;
    [-,cs] [-,chpair] [-,ch] =>
       ?NOT samvar[*1,*2:1] / &FAIL;
  simsrasgn                                                          188
    [-,sclist] => dosr1[*1,*2:1] dosr2[*1,*2];
    [-,-] => dosr1[*1,*2];
  dosr1  %take care of first item of sclist%                         188
    [-,sclist] => dosr1[*1,*2:1];
    [-,mt] => store[fof[*1,"sysl"],con[=0]];
    [-,cc] =>  ladcll[push[*2:1,S],*1,"sr1set"];
    [-,ch] =>  ladcll[push[*2,S],*1,"sr1set"];
    [-,cs] =>
       samvar[*1,*2:1] %do nothing% /
        ladcll[pushad[*2:1],*1,"cpysr"];
    [-,chpair] => dosr1[*1,mt[]] dosr2[*1,*2];
  dosr2 %append string in *2 to string in *1%                        188
    [-,sclist[sclist,-]] => dosr2[*1,*2:1] dosr2[*1,*2:2];
    [-,sclist] => dosr2[*1,*2:2];
    [-,cc] =>  ladcll[push[*2:1,S],*1,"apchr"];
    [-,cs] =>  ladcll[pushad[*2:1],*1,"apsr"];
    [-,ch] =>  ladcll[push[*2,S],*1,"apchr"];
    [-,chpair] =>
       pushad[*1] push[*2:3,S] push[*2:2,S] pushad[*2:1]
       cllm[=4,"chpair"];
    [-,mt] => TRUE;
  ladcll [-,-,-] => %do 1,load address and CALLM with 2 args%        188
    pushad[*2] *1 cllm[=2,*3];
  sclist [-,-] => *1 *2;                                             188
  cs [-] => pushad[*1] cll1["kps"];                                  188
  cp [-] => *1  MOVEI A1|A 'sptr1', p[] A1, cll1["aptstr"];          188
  cppx [-,-] =>                                                      188
    (? *N2 = 0 SETZM  / one[A1] MOVEM A1|A)
    'modeshift', AOS 'modeset', cp[*1];
  srmake [-,-,-] =>                                                  189
    push[*3,S] push[*2,S] push[*1,S] cllm[=3,"srmk"];
  cppw [-] => AOS 'mkrstay', cp[*1];                                 189
  cc [-] => push[*1,S] cll1["apachr"];                               189
  cn [-] => push[ch[*1],S] cll1["apachr"];                           189
  chpair [-,-,-] => %string, left index, right index%                189
    pushad[*1] cll1["asrref"] &acpush
    load[*2,A2] jspto["fxsp1"] %assumes fxsp1 leaves a1 unchanged%
    load[*3,A2] AOJ A2|A, jspto["fxsp2"] &acpop
    MOVEI A1|A 'sptr1', p[] A1, cll1["aptstr"];
  one [-] => MOVEI *N1|A 1, acreset[*1];                             189
  pair = _ pos pos :pr[2];                                           189
  pr [-,-] => ldptr[*1] jspto["fxsp1"] ldptr[*2] jspto["fxsp2"];     189
  fechc1 => cll["fxswork"];                                          1911
  ldptr                                                              1911
    [sf[sf]]
    [se[se]] => ldptr[*1:1];
    [sf[.ID]] => MOVE A1|A psid[*1:1] one[A2];
```

```
    [sf] =>
      pushad[*1:1] cll1["asrref"] one[A2];
    [se[.ID]] => p[] psid[*1:1] cll1["cpfse"];
    [se] =>
      pushad[*1:1] cll1["asrref"] p[] A1, cll1["cpfse"];
    [-] =>
      MOVE A2|A (?@ ref *1 / IMMEDIATE) doadr[*1]
      jspto["lpr"];
  psid                                                          1B1(
    [sf]
    [se] => psid[*1:1];
    [-] => [?@ ref *1 I] doadr[*1];
  pos =                                                         1B1(
    "SF(" stspec *) :sf[1] /
    "SE(" stspec *) :se[1] /
    tpointer;
  stspec = *+ fwlhs *+ / tpointer;                              1B1(
  tpointer = .ID;                                               1B1(

  pexp = union :punion[1];                                      1B1(
  union = inter [ "OR" union :por[2] ];                         1B1(
  inter = negtn [ "AND" inter :pand[2] ];                       1B1(
  negtn = "NOT" negtn :pnegtn[1] / alter;                       1B1)
  alter = concat [ */ alter :palter[2] ];                       1B1)
  concat = .%( nele / ele :ptunit[1]) :pconcat[$];              1B1)
  nele =                                                        1B1)
    "TRUE" :mt[] /
    *< :psd[=0] /
    *> :psd[=1] /
    *| tpointer :sptr[1] /
    *_ (.NUM / :=1) tpointer :dptr[2] /
    "FS" .ID :fs[1] /
    "FR" .ID :fr[1] /
    pos :fcp[1] /
    *+ (.ID / link) :cll[1];
  ele =                                                         1B1)
    (.SR / *+ fwlhs *+) :astest[1] /
    char / charclass /
    "FT" .ID :fgtest[1] /
    *( union *) :punion[1] /
    *- ele :notele[1] /
    "ID" (*= .UID :E / *# .UID :N) :ident[2] /
    "SINCE" ?&cvdate :date[1,GE] /
    "BEFORE" ?&cvdate :date[1,L] /
    "BETWEEN" pos pos *( union *) :pbound[3] /
    *? (.ID / link) :clltest[1] /
    *[ union *] :punach[1] /
    .NUM :con[1] (*$ uprbnd ele :parb[3] / ele :ptimes[2]) /
      _ lowbnd *$ uprbnd ele :parb[3];                          1B11
  lowbnd = (.NUM / :=0) :con[1];                                1B1
  uprbnd = (.NUM / :=1000) :con[1];                             1B1
  psd [-] => ? *N1 = 0 cll["scnlft"] / cll["scnrht"];           1B1
  fs [-] => one[A1] MOVEM A1|A [?@ ref *1 I] doadr[*1];         1B1
  fr [-] => SETZM [?@ ref *1 I] doadr[*1];                      1B1
  fcp                                                           1B1
    [sf] => setscn[] fcp1[*1];
```

```
    [se] => SETZM *scndir*, fcp1[*1];
    [-] => fcp1[*1];
  fcp1 [-] => ldptr[*1] jspto["fcp"] cll["fxswork"];                    1812
  setscn => one[A1] MOVEM A1|A *scndir*,;                              1812
  pos [-] => SUB P|A =LSH(*N1)18 *N1,;                                 1812
  pcp => jspto["pcp"];                                                 1812
  sptr [-] =>                                                          1812
    MOVE A1|A (?@ ref *1 / IMMEDIATE) doadr[*1] jspto["sptr"];
  dptr                                                                 1812
    [1,-] =>
      MOVE A1|A (?@ ref *2 / IMMEDIATE) doadr[*2] p[] A1, cll1["dptr"];
    [-,-] =>
      (?@ ref *2 p[] doadr[*2] / MOVEI A1|A doadr[*2] p[] A1,)
      p[] =*N1,
      cllm[=2,"pdc"];

%the arguments for the following output rules are
[thingy,false,true] or [thingy,false]%

  punion [-] =>                                                        1813
    &savreg pcp[] por[*1,G#1,G#2]
    >#1 scp[] SETOM *flag*,
    >#2 AOS ac|A *flag*, pps[=2] &resetall &resrec;
  por
    [por,-,-] =>
      pand[*1:1,G#1,*3] >#1 scp[] por[*1:2,*2,*3];
    [-,-,-] => pand[*1,*2,*3];
  pand                                                                 1813
    [pand,-,-] =>
      pnegtn[*1:1,*2,G#2] >#2 scp[] pand[*1:2,*2,*3];
    [-,-,-] => pnegtn[*1,*2,*3];
  pnegtn                                                               1813
    [pnegtn,-,-] => pnegtn[*1:1,*3,*2];
    [-,-,-] => palter[*1,*2,*3];
  palter                                                               1813
    [palter,-,-] =>
      pconcat[*1:1,G#1] ljump[*3] >#1
      scp[] palter[*1:2,*2,*3];
    [-,-,-] => pconcat[*1,*2] ljump[*3];
  scp => cll["scp"];                                                   1813
  pconcat [pconcat[$],-] => $*1(pcon[*$,*2]);                          1813
  pcon                                                                 1813
    [ptunit,-] => ptunit[*1:1,*2];
    [-,-] => *1;
  ptunit                                                               1813
    [astest,-] =>
      pushac[*1:1] cll1["tstsr"] gofail[*2];
    [con,-] =>
      readc[]
      (smlcon[*1:1] CAIE A1|A *N1:1, / CAME A1|A =*N1:1,) ljump[*2];
    [cct,-] =>
      p[] =*N1:1, cll1["cct"] gofalse[*2];
    [fgtest,-] =>
      SKIPN [?@ ref *1:1 I] doadr[*1:1] ljump[*2];
    [clltest,-] =>
      cll[*1:1] SKIPN *flag*, ljump[*2];
```

```
    [pbound,-] =>
       cll["savpos"] ldptr[*1:1] MOVEM A2|A 'ps',
       lptr[*1:2] MOVEM A2|A 'pe',
       pcp[] fcp[*1:1] por[*1:3,G#2,G#1]
       >#2 pps[=2] cll["respos"] ljump[*2]
       >#1 pps[=2] cll["respos"];
    [-,-] => ptuni[*1,*2];
  ptuni                                                          1813
    [punion,-] =>
       pcp[] por[*1:1,G#2,G#1] >#2 pps[=2]
       ljump[*2] >#1 pps[=2];
    [notele,-] => ptunit[*1:1,G#2] ljump[*2] >#2;
    [parb,-] => parb[*1:1,*1:2,*1:3,*2];
    [ptimes,-] => parb[*1:1,*1:1,*1:2,*2];
    [punach,-] =>
       punach[*1:1,*2] /
       xunach[*1:1,G#2] ljump[*2] >#2 pos[=2];
    [ident,#1] =>
       MOVEI A1|A =*S1:1, @S litsr *1:1 c[] A1, cll1["idtst"]
       JUMP *N1:2|MOD A1|A glbadr[#1],;
    [date,-] =>
       cll["txtdat"] CAM *N1:2|MOD A1|A =*N1:1, ljump[*2];
  part  %lowerbound,upperbound,test,false label%                 1814
    [con[0],con[1000],cct,-] =>
       p[] =*N3:1, cll1["span"];
    [con,con,-,-] =>
       MOVE A1|A =LSH(*N1:1)18 *N2:1, jsptop["begarb"]
       >#1 ptunit[*3,G#2]
       cll["incarb"] JUMPN A1|A glbadr[#1],
       % if incarb returns true then look for another instance by
       branching to #1, else have reached upper bound so fall thru to
       endarb.  Also come here if the test fails. %
       >#2 cll["endarb"] gofalse[*4];
       % if endarb returned false, then goto the false label location,
       else fall thru meaning that the $ succeeded %
  gofalse [#1] => JUMPE A1|A glbadr[#1],;
  gosucc [#1] => JUMPN MREG|A glbadr[#1],;                       181
  gofail [#1] => JUMPE MREG|A glbadr[#1],;                       181
   xunach [-,-] => %thingy,true%                                 181
     pcp[] >#2 por[*1,G#1,*2] >#1 cll["bfs"] gofail[#2];
   punach                                                        181
     [pconcat[ptunit[astest]],-] =>
        pushad[*1:1:1:1] cll1["tstf"] gofail[*2];
     [pconcat[ptunit[cct]],-] =>
        p[] =*N1:1:1:1, cll1["cctf"] gofalse[*2];
     [pconcat[ptunit[con]],-] =>
        p[] =*N1:1:1:1, cll1["tchf"] gofalse[*2];
   exp =                                                         181
     "IF" exp "THEN" exp "ELSE" exp :xpr[3] /
     "CASE" exp "OF"
        .$(-*E csexpitem [*;]) :citm[$]
        "ENDCASE" exp :csexp[3] /
     disjunct;

      xpr [-,-,-] =>                                            1814
        brf[*1,G#1] &pushb load[*2,ac] JRST G#2, &xchr
```

```
        >#1 load[*3,ac] >#2 &compr;

      csexpitem = casehead exp :csitm[2];                                    1B146
      csexp [-,-,-] =>                                                       1B146
        %cstrt load[*1,ac] &G#2 &cpush *2 load[*3,ac] &cpop >#2;
      csitm [$] => $(*$);                                                    1B146
      csitm [-,-] =>                                                         1B146
        brf[*1,G#1] &pushb load[*2,ac] &lplev _ 1 &cgetg2 JRST #2, &caseck
        >#1 &popr;

    disjunct = conjunct $("OR" conjunct :boolean[disj[2]]);                  1B14
    conjunct = negation $("AND" negation :boolean[conj[2]]);                 1B14
    negation = relation / "NOT" relation :boolean[not[1]];                   1B14
    relation =                                                               1B15
      sum [binrel] /
      string strelation :boolean[1] /
      "POS" posrel :boolean[1] /
      "FIND" pexp :boolean[1] /
      "SKIP" &FS skipcll  (*! syschk ( .ID jsyscll / builtin) / procall) &FR
      skipcll :boolean[skip[1]];
    string = .SR / *+ fwlhs *+;                                              1B15
    strelation = "NOT" strln :not[boolean[1]] / strln;                       1B15
    strln =                                                                  1B15
      (*= :E / *# :N / "<=" :LE / ">=" :GE / *< :L / *> :G)
      string :strl[3];
    binrel = ("NOT" binrl :not[boolean[1]] / binrl) :boolean[1];             1B15
    binrl =                                                                  1B15
      *=  (sum :logical[2,E] / charclass :cct[2]) /
      *#  (sum :logical[2,N] / charclass :cct[2] :not[boolean[1]])
      /
      (  ">=" sum :GE /
         "<=" sum :LE /
         *>   sum :G  /
         *<   sum :L  ) :logical[3] /
      "IN" intrel;
    charclass =  (                                                           1B15
      "CH"  :=1 /
      "ULD" :=2 /
      "LLD" :=3 /
      "NLD" :=11 /
      "LD"  :=4 /
      "UL"  :=5 /
      "LL"  :=6 /
      *L    :=7 /
      *D    :=8 /
      "PT"  :=9 /
      "NP"  :=10 ) :cct[1];
    intrel =  (                                                              1B15
      *( intbnd
         ( *] :G / *) :GE ) :G /
      *[ intbnd
         ( *] :G / *) :GE ) :GE ) :int[5];
    intbnd = sum *, sum;                                                     1B15

    posrel = pos ("NOT" posrl :not[boolean[1]] / posrl);                     1B16
    posrl =  (                                                               1B16
```

```
      ">=" pos :GE /
      "<=" pos :LE /
      *= pos :E /
      *# pos :N /
      *> pos :G /
      *< pos :L )  :plogical[3];

    sum =                                                              1B16
      _ prim  -*+  -*-  -**  -*/ -*M  -*, /
      prod $(*+ prod :addnode[2] / *- prod :subnode[2]);

       addnode                                                         1B162
          [minus,-] => subnode[*2,*1:1];                               1B162[
         [con,con] => load[con[=eval[*1]+eval[*2]],ac];
         [-,con[0]] => load[*1,ac];
         [con[0],-] => load[*2,ac];
          [-,-] => comop[*1,*2,ADD];                                   1B162[
       subnode
         [con,con] => load[con[=eval[*1]-eval[*2]],ac];
         [-,con[0]] => load[*1,ac];
         [-,-] => sub1[*1,*2] acreset[ac];
       sub1 [-,-] =>                                                   1B162
         isadc[*2] load[*1,ac] doadc[*2,SUB,ac] /
         isadc[*1] loadneg[*2,ac] doadc[*1,ADD,ac] /
         stoken[*2] ?AND onereg[*1] sub2[*1,*2] /
         stoken[*1] ?AND onereg[*2] sub3[*1,*2] /
         token[*2] sub2[*1,*2] /
         token[*1] sub3[*1,*2] /
         load[*1,ac] &acpush adrget[*2] &acpop SUB ac|A,;
       sub2 [-,-] => load[*1,ac] oper[*2,SUB,ac];                      1B162
       sub3 [-,-] => loadneg[*2,ac] oper[*1,ADD,ac];                   1B162

    prod = bitor                                                       1B16
      $(** bitor :mulnode[2] /
      "MOD" bitor :remnode[2] /
      */ bitor :divnode[2]);

       mulnode
         [con,con] => load[con[=eval[*1];*eval[*2]],ac];
         [-,con[1]] => load[*1,ac];
         [con[1],-] => load[*2,ac];
         [-,-] => comop[*1,*2,IMUL];
       remnode [-,-] => divnd2[*1,*2] MOVE ac|A ac+1,;                 1B16
       divnode
         [con,con] => load[con[=eval[*1];/eval[*2]],ac];
         [-,con[1]] => load[*1,ac];
         [-,-] => divnd2[*1,*2];

      divnd2 % actually do the divide %                                1B16
         [-,-] =>
            (opadc[*1,*2,IDIV] /
               load[*1,ac] (
                  oper[*2,IDIV,ac] /
                  &acpush adrget[*2] &acpop IDIV ac|A,))
         acreset[ac] acreset[=ac+1];
    bitor =                                                            1B1
```

```
    bitand $(".V" bitand :iornode[2] / ".X" bitand :xornode[2]);

     iornode [-,-] => comop[*1,*2,IOR];                                  18164
     xornode [-,-] => comop[*1,*2,XOR];                                  18164

  bitand = factor $(".A" factor :andnode[2]);                            1816

     andnode [-,-] => comop[*1,*2,AND];                                  18166

  factor =                                                               1816
    '- factor :minus[1] /
    '( exp ') $qualifiers /
    prim;
  prim =                                                                 1816
    lhs [tail] $qualifiers / rhs /
    "MKFD" '( exp ', exp ', exp ')
       :fcnref[gvar["mkbptr"],act[3],mt[]] /
    "MAX" explist :minmax[1,LE] /
    "MIN" explist :minmax[1,GE] /
    "DESCR" listref :lgdescr[1] /
    "ELEM" listref :lgval[1] /
    listref :lgval[1] / % shorthand for ELEM %
    "PCALL" notcatch pcall2 /
    "CCPOS" :gvar["swork1"] /
    "SIGNALTYPE" :gvar["sysgtp"] /
    "SIGNAL" :gvar["syssig"] /
    "PORT"
       (?F coroutine /
          :syntx[" ","PORT not allowed in procedure"]*)
       :dfport[] /
    "HELP" :gvar["syshlp"] hcall /
    "VALUE" '( exp ( ', exp :srval[2] / :srval[1,con[=10]]) ') /
    "READC" ('( exp ') :readc[1] / :readc[]) /
    '! syschk .ID jsyscll $qualifiers ;
  listref = % list element reference %                                   1816
    '# fwlhs '# '[ exp '] :lele[2];
  syschk = % check for sys allow %                                       1817
     ?F sysfg /
       :syntx[" "," system interface code not allowed in this file" ]*
       ;
  rhs =                                                                  1817
    '$ (fwlhs :loadad[1] / .SR :adsr[1]) /
    "CATCHPARAM"
       ( ?F catchf / :syntx[" "," only allowed in catchphrase"]* )
       :gvar["syscpm"] /
    literal;
  literal =                                                              1817
    .NUM :con[1] / "TRUE" :con[=1] / "FALSE" :con[=0] /
    "WORD" :con[=36] /
    "ADR-PER-WORD" :con[=1] /
    ("ADDRESS" / "ADDR" ) :con[=18] /
    ("CHARACTER" / "CHAR" ) :con[=7] /
    "CPU" :con[=10] /
    char;
  char =  (                                                              1817
    .SR1 /
```

```
      "ENDCHR" :=377B /
      ("BUG" / "CA") :=4 /
      "CD"  :=30B /
      "SP"  :=40B /
      "CR"  :=15B /
      "LF"  :=12B /
      "TAB" :=11B /
      "EOL" :=37B /
      "ALT" :=33B /
      "BC"  :=10B /
      "BW"  :=27B /
      "C."  :=140B ) :con[1];
   pcall2 = % port call syntax %                                    1B17
      portdes pcall3;                                               1B174
   pcall3 = % pcall without portdes syntax %                        1B17
      *( expreglist presults :pcallp[3] catch1 /
      :mt[] pcalres :pcallp[3] ;

   hcall = % pcall3 without port result %                           1B17
      *( expreglist mresults :pcallp[3] *) /
      :mt[] :mt[] :pcallp[3] ;

   hcall2 = % pcall3 without any results %                          1B17
      *( expreglist :mt[] :pcallp[3] *) /
      :mt[] :mt[] :pcallp[3] ;

   pcallp % port call code %                                        1B17
      [-,-,-] => % port id, args, results %
         &savreg
            !actaln(
               *2 % load arguments in reg's % )
            load[*1,MREG] % get new port ID %
            jspto["pcall"] % do the port call %
            fcnres[*3] % store results %
         ;
   expreglist =                                                     1B17
      !actaln(
         &actaln _ 0
         .$<*,> ( expression
            ( ?actaln=0 :load[1,A1] / :load[1,actaln] )
            &actaln _ actaln+1 ) :lodlist[$]
         [?actaln>5 :syntx[" ","too many arguments"]*]  );
   expression = exp;
   presults =                                                       1B1
      *:
         &resltn _ R1
         (
            metares /                                             1B181A
             ?F coroutine :store[dfport[],rgcon[MREG]] /
            :mt[]
            ) [*,]
         .$<*,> ( lhs
            :store[1,rgcon[resltn]]
            &resltn _ resltn+1 )
         :mr[1,$]
         [?resltn>4 :syntx[" ","too many results"]*] /
```

```
    pcalres;
  pcalres = % pcall result if not specified %                          1B18
    ?F coroutine :store[dfport[],rpcon[MREG]] /
    :mt[];
  metares = % 'meta' result, success/fail or port %                    1B18
    '[ lhs '] :store[1,rpcon[MREG]] ;
  explist = '( exp $(', exp :cexp[2]) '):                              1B18
  minmax                                                               1B18
    [cexp[cexp,-],-] => minmax[*1:1,*2] minmx2[*1:2,*2];
    [cexp,-] => load[*1:1,ac] minmx2[*1:2,*2];
    [-,-] => load[*1:1,ac];
  minmx2 [-,-] =>                                                      1B18
    &acpush load[*1,ac] CAM *N2|MOD ac|A &acpop ac,
    MOVE ac|A &acpush ac, &acpop acreset[ac];
  noloc [-] => <*1 " local " LOC> &xerror;                             1B18
  srval [-,-] =>                                                       1B18
    push[*2,S] push[*1,S] cllm[=2,"srval"];
  readc                                                                1B18
    [-] => load[*1,WA] readcgo[];
    [] => MOVEI WA|A 'swork', readcgo[];
  readcgo => JSP A4|A 'readc', &resetall;                              1B19

  brf                                                                  1B19
    [boolean,-] => bfalse[*1:1,*2];
    [-,#1] => acget[*1] JUMPE glbadr[#1], &setb;

    bfalse                                                             1B19
      [disj,-] => brt[*1:1,G#1] &pushb brf[*1:2,*2] >#1 &compr;
      [conj,-] => brf[*1:1,*2] &pushb brf[*1:2,*2] &compb;
      [not,-] => brt[*1:1,*2];
      [cct,-] => *1 gofail[*2] &setb;
      [strl,#1] => *1 JUMPE ac|A glbadr[#1], &setb;
      [logical,-] => logical[*1:1,*1:2,*2,*1:3] &setb;
      [plogical,-] => plogical[*1:1,*1:2,*2,*1:3] &setb;
      [punion,#1] => *1 JUMPE ac|A glbadr[#1], &setb;
      [skip,-] => load[*1:1] ljump[*2] &setb;
      [int,-] => *1 ljump[*2] &setb;

  brt                                                                  1B19
    [boolean,-] => btrue[*1:1,*2];
    [-,#1] => acget[*1] JUMPN glbadr[#1], &setb;

    btrue                                                              1B19
      [disj,-] => brt[*1:1,*2] &pushb brt[*1:2,*2] &compb;
      [conj,-] => brf[*1:1,G#1] &pushb brt[*1:2,*2] >#1 &compr;
      [not,-] => brf[*1:1,*2];
      [cct,-] => *1 gosucc[*2] &setb;
      [strl,#1] => *1 JUMPN ac|A glbadr[#1], &setb;
      [logical,-] =>
         logical[*1:1,*1:2,*2,=MASK(*N1:3+4)7] &setb;
      [plogical,-] =>
         plogical[*1:1,*1:2,*2,=MASK(*N1:3+4)7] &setb;
      [punion,#1] => *1 JUMPN ac|A glbadr[#1], &setb;
      [skip,-] =>
         load[*1:1] JRST . 2 [?F segflag segx[]], ljump[*2] &setb;
      [int,-] => *1 JRST . 2 [?F segflag segx[]], ljump[*2] &setb;
```

```
 cct [-,cct] =>                                                          1819
   a[] =*N2:1, push[*1,S] cllm[=2,"chrct"] &resetall;

 strl [-,-,-] =>                                                         1819
   %savreg p[] =*N2, pushac[*3] pushac[*1]
   cllm[=3,"ascom"] &resetall %resreg;

 ljump [#1] => JRST #1 [?F segflag segx[] ],;                            1819
   [-] => JRST *1 [?F segflag segx[] ],;

glbadr [#1] => [?F segflag segx[] ] #1;
                                                                         1819
segx => ?IF 0 = MASK(w)4M6 SEGB|X / indxfull[];                          1819
   % check to see if index already used %

indxfull => syntx["Compiler Error using Index",""];                      1819

 logical %exp,exp,label,modifier%                                        1820
   [-,con[0],-,-] => lzero[*1,*3,*4];
   [-,con,-,-] =>
      acget[*1]
      (smlcon[*2:1] CAI *N4|MOD *N2:1, / CAM *N4|MOD =*N2:1,)
      ljump[*3];
   [-,addr,-,-] =>
      acget[*1] CAI *N4|MOD doadr[*2:1] ljump[*3];
   [-,adsr,-,-] =>
      acget[*1] CAI *N4|MOD =*S2:1, @S litsr *2:1 ljump[*3];
   [store[rvar,-],-,-,-] =>
      regload[*1:1,*1:2]
      (oper[*2,=CAM+*N4,=*V1:1:1] /
      acget[*2] modsym[*4] CAM mode|MOD *V1:1:1,) ljump[*3];
   [-,-,-,-] => lgcl[*1,*2,*3,*4];
 lgcl                                                                    1821
   [rvar,-,-,-] =>
      oper[*2,=CAM+*N4,=*V1:1] ljump[*3] /
      modsym[*4] lop[*2,*1,*3,mode];
   [mt,-,-,-] => %for case statement% lop1[*2,*3,*4];
   [con,-,-,-]
   [addr,-,-,-]
   [adsr,-,-,-]
   [-,rvar,-,-]
   [-,store[rvar,-],-,-] =>
      modsym[*4] logical[*2,*1,*3,mode];
   [-,-,-,-] =>
      sv[*1] ?AND stoken[*2] lop[*1,*2,*3,*4] /
      sv[*2] ?AND stoken[*1] modsym[*4] lop[*2,*1,*3,mode] /
      token[*2] lop[*1,*2,*3,*4] /
      token[*1] modsym[*4] lop[*2,*1,*3,mode] /
      lop[*1,*2,*3,*4];

 sv [lvar] [gvar] [store[lvar,-]] [store[gvar,-]] => TRUE;                182
 modsym [-] => &mode _ *N1  &modechange;                                  182

 lop [-,-,-,-] => %exp,exp,label,modifier%                                182
   %know that will load *1.  compare to *2%
```

```
    %if *2 is easy then can load *1 in any accumulator, else load
    in ac%
    stoken[*2] load[*1] oper[*2,=CAM+*N4,acnum] ljump[*3] /
    load[*1,ac] lop1[*2,*3,*4];

  lop1 [-,-,-] => %exp,label,modifier%                              1B20
    %compare ac to the exp doing the modified CAM, then jump to
    label%
    (oper[*1,=CAM+*N3,ac] /
       &acpush adrget[*1] &acpop CAM *N3|MOD ac|A,)
    ljump[*2];

  lzero %exp,label,modifier%                                        1B20
    [lvar,-,-]
    [gvar,-,-] => lzero2[*1:1,*2,*3];
    [store,-,-] => lzst[*1,*2,*3];
    [rvar,-,-] => JUMP lzero3[*1:1,*2,*3];
    [-,-,-] =>
       oper[*1,=SKIP+*N3,=0] ljump[*2] /
       lzero1[*1,*2,*3];

  lzero2 [-,#1,-] => %id,label,modifier%                            1B20
    &actype _ acvar  &acval _ HASH(*1) (
    ? &isinac JUMP *N3+4|MOD acnum|A glbadr[#1], /
    &pickac SKIP *N3|MOD acnum|A
       doadr[*1] ljump[*2] &stoac);

  lzst %store,label,modifier%                                       1B21
    [store[rvar,-],-,-] => lzrst[*1,*2,*3];
    [store[-,subnode[-,con[1]]],-,-] =>
       token[*1:1] ?AND samvar[*1:1,*1:2:1]
          lzst1[*1:1,SOS,*2,*3] /
       lzero1[*1,*2,*3];
    [store[-,addnode[-,con[1]]],-,-] =>
       token[*1:1] ?AND samvar[*1:1,*1:2:1]
          lzst1[*1:1,AOS,*2,*3] /
       lzero1[*1,*2,*3];
    [-,-,-] => lzero1[*1,*2,*3];

  lzrst %store[rvar,-],label,modifier%                              1B21
    [store[rvar,subnode[rvar[*1:1:1],con[1]]],-,-] =>
       SOJ lzero3[*1:1:1,*2,*3];
    [store[rvar,addnode[rvar[*1:1:1],con[1]]],-,-] =>
       AOJ lzero3[*1:1:1,*2,*3];
    [-,-,-] =>
       regload[*1:1,*1:2] JUMP lzero3[*1:1:1,*2,*3];

  lzst1 %token,op (AOS or SOS),label,modifier%                      1B21
    [lvar,-,-,-]
    [gvar,-,-,-] =>
       svset[*1:1] &pickac
       oper[*1,=*N2+*N4,acnum] ljump[*3] stoacc[*1,acnum];
    [-,-,-,-] =>
       oper[*1,=*N2+*N4,=0] ljump[*3];

  lzero1 %exp,label,modifier%                                       1B21
```

```
    [-,#1,-] => acget[*1] JUMP *N3+4|MOD glbadr[#1],;

  lzero3 [-,#1,-] => *N3+4|MOD *V1|A glbadr[#1],;                    1821

  int [-,-,-,-,-] => %exp,lower,upper,skip1,skip2%                   1821
    stoken[*2]
      (stoken[*3] load[*1] int1[*2,*3,*4,*5,acnum] /
      load[*1,ac] intskp[*3,*4,ac] intskp[*2,*5,ac]) /
    load[*1,ac] &acpush load[*2,ac] intskp[*3,*4,=ac-1]
      ac &acpop CAM *N5|MOD ac|A,;
  int1 [-,-,-,-,-] => intskp[*2,*3,*5] intskp[*1,*4,*5];             1821
  intskp %exp,modifier,ac%                                           1821
    %compare ac and the exp doing the skip specified by the
    modifier%
    [con,-,-] =>
      *N3|A
      (smlcon[*1:1] CAI *N2|MOD *N1:1, / CAM *N2|MOD =*N1:1,);
    [addr,-,-] => CAI *N2|MOD *N3|A doadr[*1:1];
    [adsr,-,-] => CAI *N2|MOD *N3|A =*S1:1, &S litsr *1:1;
    [-,-,-] =>
      oper[*1,=CAM+*N2,*3] /
      &acpush adrget[*1] &acpop CAM *N2|MOD *N3|A,;

  stoken                                                             1821
    [minus[con]] [lvar] [rvar] [gvar] [mt]
    [rvarindexec[-,rvar]] [gvarindexed[-,rvar]] => TRUE;
    [indir] => simind[*1:1] / &FAIL;
    [-] => isadc[*1] / &FAIL;
  simind                                                             1822
    [lvar] [rvar] [gvar] => TRUE;
    [rvarindexed[-,rvar]] [gvarindexed[-,rvar]] => TRUE;
    [addnode[rvar,-]] => simcon[*1:2] / &FAIL;
    [subnode[rvar,-]] => simscon[*1:2] / &FAIL;
    [addnode[-,rvar]] => simcon[*1:1] / &FAIL;
  smlcon [-] => ? MASK(*N1)777777B6 = 0 / &FAIL;                     1822
  simcon                                                             1822
    [con] => smlcon[*1:1] / &FAIL;
    [addr] => TRUE;
  simscon                                                            1822
    [con] => smlcon[*1:1] / &FAIL;
    [addr] => ckmad[*1] / &FAIL;

  plogical %pos,pos,label,modifier%                                  1822
    [-,-,-,-] =>
      (? *N4 = N psidchk[*1,*2,G#1] / psidchk[*1,*2,*3])
      plog[*1,*2,*3,*4] [? *N4 = N >#1] &resetall;
    psidchk [-,-,-] =>                                               1822
      sampsid[*1,*2] %don't bother to check% /
      loadstrid[*1,A1] &acpush loadstrid[*2,A2] &acpop
      CAME A1|A A2, ljump[*3];
    sampsid                                                          1822
      [sf,-] [se,-] => sampsid[*1:1,*2] / &FAIL;
      [-,sf] [se,-] => sampsid[*1,*2:1] / &FAIL;
      [.ID,*1] => TRUE;
    loadstrid  %pos,ac%                                              1822
      [sf[.ID],-] [se[.ID],-] => MOVE *N2|A psid[*1:1];
```

```
      [sf,-] [se,-] => loadad[*1:1,*2];
      [-,-] => MOVE *N2|A psid[*1];
    plog %pos,pos,label,mod%                                        1B224
      [se,se,-,-]
      [sf,sf,-,-] => plog2[*3,*4];
      [-,se,-,-] => modsym[*4] plog1[*2,*1,*3,mode];
      [-,-,-,-] => plog1[*1,*2,*3,*4];
    plog1                                                           1B224
      [-,sf,-,-] => ldptr[*1] CAI *N4|MOD A2|A 1, ljump[*3];
      [-,-,-,-] =>
         ldptr[*1]
         MOVE (?@ ref *2 / IMMEDIATE) A1|A doadr[*2]
         CAM *N4|MOD A2|A A1|X 1, ljump[*3];
    plog2                                                           1B224
      [-,E] [-,GE] [-,LE] => TRUE;
      [-,N] [-,G] [-,L] => %false% ljump[*1];

 declare =  (decl/ext/equ/regdec/refd/ptrd/pgdec) % <"warning: declared
item(s) not names"> % *;;                                           1B2

    decl = "DECLARE" extchk (                                       1B22
      "FIELD" fdef $(*, fdef) /
      "STRING" astrings * /
      "TEXT" "POINTER" tpts * /
      "STACK" #<*,>( stkdef * ) /
      "RING" #<*,>( rngdef * ) /
      "LIST" #<*,>( listdef * ) /
      "CONSTANT" consts /
       items * ) ;                                                  1B226

       fdef =                                                       1B226A
         .ID syschk :exts[1]* fdef1*;
      fdef1 =                                                       1B226A
         [*= / *_] *[ address *, decval *: cecval *] :fld[5];
      flgrh =                                                       1B226A
         fdef1 :fld2[2];                                            1B226A1
      fld2                                                          1B226A
         [-,-] => exts[*1] *2;
      listdef = .ID lstgrh;                                         1B226A
      extchk =                                                      1B226A
         "EXTERNAL" &FS decf / &FR dect;
       cecval = .ID / .NUM;                                         1B226A
       fld [-,-,-,-,-] => %indirect,addr,incex,size,position%       1B226A
         fldp[*5] flds[*4] bltindir[*1] bltrdx[*3] bltad[*2];
       fldp                                                         1B226A
         [.ID] => check[*1] *V1|POS;
         [-] => *N1|POS;
       flds                                                         1B226A
         [.ID] => check[*1] *V1|SIZ;
         [-] => *N1|SIZ;
      tpts =                                                        1B226A
         .#<*,> tptdef :many[$] ;
      tptdef =                                                      1B226A
         .ID :tptd[1];
      tptd % text pointer definition %                              1B226A
         [-] => exts[*1] &BSS 2,;
```

```
many                                                                    1B226A
   [$] => $(*$) % unwind %;
consts =                                                                1B226A
   .#<',> ( const * ) ;
const =                                                                 1B226A
   .ID constrh ;
constrh =                                                               1B226A
   '= compexp :con2[2];
con2                                                                    1B226A
   [-,-] => [?F decf @S external *1]
      [ ?@ linked *1 syntx[*1," already referenced or defined"]]
      &val _ eval[*2]
      >*1 _ val @S constant *1 @R reloca *1
      [ ?val < 20B @S noddt *1 ] ;
items =                                                                 1B226A
   .#<',> item :many[$] ;
item = % declare item %                                                 1B226A
   .ID itemrh;
itemrh =                                                                1B226A
    spectype ( dimension / value / :itm[1] );
lstarh = % list def, RH side %                                          1B226A
   '[ compexp '] :listdf[2] / %initial size %
   :listdf[1,con[=0]];
listdf % define list given size %                                       1B226A
   [-,-] => % ID, size %
      0, % runtime package word %
      exts[*1] % define symbol %
      +eval[*2] [LH, % M,,L %
      &BSS eval[*2],; % room for it %
exts                                                                    1B226A
   [recref] => [?F decf @S extern *1:1]
      >*1:1 _ LSH(HASH(*1:2))18+. @S reloca *1:1 ;
   [.ID] => [?F decf @S extern *1] >*1;
dimension =                                                             1B226A
   '[ compexp '] :decbnd[2];
decbnd                                                                  1B226A
   [-,-] => exts[*1] &BSS eval[*2],;
value =                                                                 1B226A
   ('=/'_) ( '( .#<',> itemval ') :declst[$] / itemval :declst[1]
   :decv[2];
decv                                                                    1B226A
   [-,-] => exts[*1] $*2( itm2[*$] );
eval % evaluate compexp tree %                                          1B226A
   % later: xpr, csexp, boolean, andnode %
   [.UID]
   [.ID] :=
      ?@ defned *1
         ?AND ?NOT ?@ reloca *1
         ?AND ?NOT ?@ ref *1
         ?AND ?NOT ?@ recptr *1 *V1/
      syntx[*1," not legal compiletime identifier"] 0;
   [loadac[rvar]] := eval[*1:1:1];
   [lvar]
   [rvar] := eval[*1:1];
   [con] := *N1:1;
   [addnode] := 0 + eval[*1:1] + eval[*1:2];
```

```
      [subnode] := 0 + eval[*1:1] - eval[*1:2];
      [mulnode] := 0 + eval[*1:1] ;* eval[*1:2];
      [divnode] := 0 + eval[*1:1] ;/ eval[*1:2];
      [minus] := -eval[*1:1];
      [xpr] := ?eval[*1:1]=0 0+eval[*1:3] / 0+eval[*1:2];
      [boolean] := 0+eval[*1:1];
      [disj] := ?eval[*1:1]#0 1 / ?eval[*1:2]#0 1 / 0;
      [conj] := ?eval[*1:1]#0 ?AND ?eval[*1:2]#0 1 / 0;
      [not] := ?eval[*1:1]#0 0 / 1;
      [-] := syntx[" ", " invalid compiletime expression"] 0;
   compexp =                                                    1B226A3
      &FS ctime exp &FR ctime;
      % ctime will cause syntax error if udef symbols parsed %
   itemval =                                                    1B226A4
      .SR / '$ ( .SR :adsr[1] / fwlhs :loadad[1]) / exp;
    itm2                                                        1B226A4
      [.SR] => *S1 &S litsr *1; %dont need a comma%
      [adsr] => =*S1:1,;
      [loadad] => ?[loadad[gvar]] *V1:1:1,/
         ?[loadad[con]] *N1:1:1, /
         <"illegal $ construct " LOC> &xerror;
      [-] => +eval[*1],;
   itm                                                          1B226A4
      [-] => exts[*1] 0,;
   astrings =                                                   1B226A4
      .#<',> astrng :many[$] ;
   astrng =                                                     1B226A4
      .ID astrh;
   astrh =                                                      1B226A4
      ( ('=/'_) .SR :astr[2] / '[ compexp '] :srdec[2]);
    astr [-,-] => exts[*1] LSH(*L2)18 *L2, *S2 &S litsr *2;     1B226A4
    srdec [-,-] => exts[*1]                                     1B226A4
      &val _ eval[*2]
      &srsize _ val &srwords LSH(val)18, &BSS srsize,;
   stkdef =                                                     1B226A4
      .ID stkrh;
   stkrh =                                                      1B226A4
      '[ compexp ( ', compexp / :con[=1] ) :std[3] '] :
    std [-,-,-] => exts[*1] 1|A %for stacks% strd[*2,*3];       1B226A5
    strd [-,-] =>                                               1B226A5
      &val _ eval[*1] &val2 _ eval[*2]
      . 2-val2, LSH(val2)18 val;*val2 . 1, &BSS val;*val2,;
   rngdef =                                                     1B226A5
      .ID rngrh;
   rngrh =                                                      1B226A5
      '[ compexp ( ', compexp / :con[=1] ) :rnd[3] '] ;
    rnd [-,-,-] => exts[*1] 2|A %for rings% strd[*2,*3];        1B226A5

declname = % named declaration %                                  1B22(
   "DECLARE" extchk itemrh* '; /
   extchk (                                                       1B226(
      "STRING" astrh* '; /
      "CONSTANT" constrh* '; /
      "TEXT" "POINTER" :tptd[1]* '; /
      "STACK" stkrh* '; /
      "RING" rngrh* '; /
```

```
        "LIST" lstdrh * *; /
        "ADDRESS" equrh* *: /
        "SET" equrh* *; /
        "FIELD" syschk flcrh* *; /
        ["DECLARE"] itemrh* *; ) ;
    record = % Record Declaration %                                    1B226
        "RECORD"
            &recbit _ 0 &recdsp _ 0
            fieldlist *; :rechd[2]*:                                   1B226C1
        fieldlist =                                                    1B226C
            recdef .$( *, recdef ) :many[1,$];                         1B226C2
        recdef =                                                       1B226C
            _ .ID *[ ( fieldlist / compexp ) *] :recdo[2];
        rechd % record heading %                                       1B226C
            [-,-] => % ID, many[field defs] %
                @S recnam *1                                           1B226C4A
                >#1 % first field here %
                &recloc _ lc
                *2 % define all fields %
                >|*1 _ LSH(recdsp+1)18+. @S reloca *1
                recdsp+1,  % be size in words %
                (?recloc=0 0\1 /
                    #1, ) % pointer to first field %
                    %NOTE: gen labels wont work for zero%
                ;
        recdc % produce one field (nested?) def %                      1B226C
            [-,many] =>
                sum2[*2]
                fieldfit[*1,val] % val = field size, computed by sum2 %
                    % fieldfit will give error if won't go in one word %
                !recbit(
                    *2
                    &val _ recbit )
                fieldef[*1,=val-recbit];
            [-,-] =>
                fieldfit[*1,=eval[*2]]
                fieldef[*1,=eval[*2]];
        fieldfit % make sure field fits %                              1B226C
            [-,-] =>
                [ ?recbit >= 36 &recbit _ 0 &recdsp _ recdsp+1 ]
                [ ?*N2 <= 0 syntx[*1," field <= 0 not allowed"] ]
                [ ?*N2 > 36 syntx[*1," field too large"]]
                [ ?recbit+*N2 > 36 &recbit _ 0 &recdsp _ recdsp+1 ] % fit? %
                ;
        sum2 % return sum of *2 arguments %                            1B226C
            [many] =>
                !val2( &val2 _ 0
                    $*1( sum2[*$:2] [ ?LAST &val_val+val2 / &val2_val2+val ] )
                    ) ;
            [-] => &val _ eval[*1];
        fieldef % define record field %                                1B226C
            [-,-] => % field def %
                defext[*1]
                recbit|POS *N2|SIZ RP|X recdsp,
                &recbit _ recbit + *N2
                ;
```

```
    ext = "EXTERNAL" .ID .$(', .ID) :extt[1,$]*;                          1B22(

      extt [$] => $(@S extern *$);                                        1B226(

    equ =                                                                 1B22(
      ( "SET" / "ADDRESS" )
        ("EXTERNAL" &FS decf / &FR decf) equ1 $(', equ1);

      equ1 = (.ID / .UID) '=                                              1B226(
        compexp :equ2[2]*;
      equ2
        [-,-] => >*1 _ eval[*2] equ3[*1];
      equ3 [-] =>                                                         1B226(
        @R reloca *1 (?F decf @S extern *1 / @S noddt *1);

      equrh =                                                             1B226(
        '=  compexp :equ2[2] ;
    regdec = "REGISTER" rdec $(', rdec);                                  1B22(

      rdec = .ID '= .NUM :rdcl[2]*;                                       1B226
      rdcl [-,-] =>                                                       1B226
        ? *N2 > 15 <*1 " cannot be > 15 " LOC LINE> /
        >*1 _ *N2 @R reloca *1 @S register *1;

    refd = "REF" .ID .$(', .ID) :refas[1,$]*;                             1B22(
      refas [$] => $(@S ref *$);                                          1B226(

    ptrd = "POINTER" .ID .$(', .ID) :ptras[1,$]*;                         1B22(
      ptras [$] => $(@S ptr *$);                                          1B226(

    pgdec = "PAGE" (.NUM / :=0) :pgdecl[1]*;                              1B22(
      pgdecl [-] =>                                                       1B226(
        &pgleft _ *N1 &pgsz
        (? pgleft = 0 / &BSS pgleft,);
                                                                          1B226L
  xreset => &erreset bodfin[];                                            1B2

  program  =                                                              1B2(
    "FILE" .ID file1 /
    "SYMBOLS" <"-L10 symbol initialization">
      $symbols "FINIS" :fin[]* /
    "PROGRAM" .ID :[ JRST *V1, ] prog1 /
    _ '( .ID ')
      ("FILE" file1 /
      "PROGRAM" @S noddt &DISCARD .ID :[ JRST *V1, ] prog1) /
    <"No Program Found">;

     file1 =  &LABEL <"-L10 5-Apr-78 "> :head[1]* &sr1flg _ 0 &lllists _
    $parts                                                                1B22
      "FINIS" :fin1[]* :fin[]* ;
     prog1 =  :head[1]* &sr1flg _ 0 &lllists _ 0 $parts "FINIS"           1B22
      :fin1[]* :fin[]* ;

     head [-] =>                                                          1B22
      % put out FILE message for user %
        <"-FILE " *1> &NAME *1
```

```
      % reset sfilev symbol %
         @R all *'sfilev'
      defit["CPU11",=0] defit["CPU10",=1]
      % define registers for PDP-10 %
         def["SFGB",=5] % segment base register %
         def["R0",=0] def["R1",=1] def["R2",=2] def["R3",=3]
         def["R4",=4] def["R5",=5] def["R6",=6] def["R7",=7]
         def["RP",=8] def["A1",=10] def["A2",=11] def["A3",=12]
         def["A4",=13] def["S",=15] def["M",=14]
         def["R9",=9] defz["R10",=10] defz["R11",=11]
         defz["R12",=12] defz["R13",=13] defz["R14",=14]
         defz["R15",=15];
   defit % define symbol %                                          1B229
      [-,-] => >*1 _ *N2 @R reloca *1;
   def % define register %                                          1B229
      [-,-] => >*1_*N2 @S register *1 @R reloca *1;
   defz % define, set noddt %                                       1B229
      [-,-] => >*1_*N2 @S register,noddt *1 @R reloca *1;

   parts =                                                          1B229
      '( .ID &LABEL ')
         (procedure / record / cortn / xcatch / declname) /
      udlist /
      "NOMESS" ['';] &nomess _ 1 /
      declare/
      "ALLOW" '! ['';] &FS sysfg /
      "NLSSYM" ['';] :ud2[]*/
      "UND-OK" ['';] &undok _ 1;

   fin1 =>                                                          1B229
      @S noddt *'sysm'  @S noddt *'sysl'
      [?@ linked *'sysl'  >sysl 18|SIZ RP|X,]
      [?@ linked *'sysm'  >sysm 18|SIZ 18|POS RP|X,];
   fin =>                                                           1B229
      $SYMS(
         [?@ waslocal *$ @S noddt *$]
         [?@ local *$ @S noddt *$]
         [?@ constant *$ ?AND ?@ external *$
            % external constant:
               redefine to be location containing value %          1B229P1C1
            &val_*V$ @R defned,linked,noddt *$ @S reloca *$
            >|*$ val, ] )
      ( ?F tonls / >"sfilev" &sfilstr % store file name string % )
      [(? tonls = 0/? errorn = 0) &TABLES];

   udlist = "LIST" ["UNDEFINED" "SYMBOLS"] ['';] :udl[]*;           1B229
   udl =>                                                           1B229
      &isund _ 0
      $SYMS([?@ linked *$ ?AND ?NOT ?@ defned *$
         [? isund = 0  <"Undefined symbols are">]
         &isund _ isund + 1  <"    " *$>]);

   ud2 =>
      $SYMS([?@ linked *$ ?AND ?NOT ?@ defned *$ &idhash _ HASH(*$)
      [?&datget]]);
   procedure = ("PROCEDURE" / "PROC") :prhead[1]*                   1B229
```

```
   &FR skipcll,coroutine,listpres &FS openok &G#lllists
   :pentry[]*
   formal vardef body;
centry % coroutine entry code %                                    1B229
   =>
      jspto["syscent"] % does push mark, M_S, add cell to stack %
      % the exttra cell is co-return location %
      &FS stkok % stack overflow checked at this point %
      ;
pentry % procedure entry code %                                    1B229
   =>
      jspto["sysent"] % does push mark, M_S, add cell to stack %
      % the exttra cell is co-return location %
      &FS stkok % stack overflow checked at this point %
      ;

    prhead [.ID] => &pnam _ *N1 defext[*1] ;                       1B229V
    defext [-] => >|*1 [  ?@ waslocal *1 locerr[*1]] ;             1B229V

 body =                                                            1B229
   &resetall &FR ut labeled $(*; labeled) "END." :bodfin[]*;

    bodfin =>                                                      1B229Y
      >#lllists &lllists _ 0
      $SYMS(
         [?F listpres ?AND ?@ llist *$ *V$, ]
         [?@ local *$ @R all *$ @S waslocal *$])
      [?F listpres 0,] ;
      % if lists, drop out list of local displacemens for sysfll %

cortn =                                                            1B22
   "COROUTINE" :prhead[1]*
   &FR skipcll &FS coroutine,openok &resetall &G#lllists
   :oper[gvar["syscer"],JSP,A4]* % error if normal call %
   :centry[]*
   formal
   :store[dfport[],rgcon[MREG]]* % store port id %
    vardef cobody;
cobody =                                                           1B22
   "PORT" "ENTRY" portdes :stoport[1]* [*;]
   ( "EXIT" / labeled [*;] "EXIT" )
   (
      lhs *_ "PCALL" pcall2 :store[2]* /                           1B229Z
      "PCALL" pcall2* )
   *;
   &FR openok
   body;
stoport % store port id in *1 %                                    1B229
   [dfport] => TRUE;
   [-] => store[*1,dfport[]];
portdes = % port designator %                                      1B229
    _ *[ (
      "PORT" :dfport[] /
      fwlhs  )                                                     1B229AB
      *] /
   :dfport[] ;
```

```
formal = &lcrst &FR ut &fcount _ MASK(-2)18M % first argument here %
                                                                    1B2297
   [ *( #<*,>( .ID spectype :formdef[1]* ) *) ] *; ;
spectype =                                                          1B2297
   [
      "RECPTR" @S recptr .ID :recref[2] /
      "REF" @S ref spectype /
      "LIST" % no action as yet % /
      "POINTER" @S ptr ];
 vardef =                                                           1B2297
   &bcount _ 0 &fcount _ 2
   $(
      "LOCAL" locd *; /
      _ *( .ID *) -*: (
         "CATCHPHRASE" @S local catbody *; /
         locdname *; ) /                                           1B229AE2E
      "REF" refdef *; /
      "POINTER" ptrdef *;)
   :stkmov[]*
   $(? &lsrpop :[ HRLZI A1|A srsize, MOVEM A1|A M|X fcount,])
   :blanks[]* ;

locdname =                                                          1B2297
   "STRING"  @strh :lastr1[1]* /
   "TEXT" "POINTER" :ld[1,=2]* /
   "LIST" lstloc /
   "CONSTANT" constrh :lconst[1]* /
   ["LOCAL"]  itemrh :litem1[1]*;

    locd =                                                          1B229A(
      "STRING" @strings :lastr[1]* /
      "TEXT" "POINTER" ltxp $(*, ltxp) /
      "LIST" #<*,> ( .ID lstloc ) /
      "CONSTANT" lcons /
       items :litem[1]*;                                          1B229AG
   lstloc = % local list def/assign %                              1B229A(
      lstdrh [ *_ listrh / :mt[]]  :lstdfx[2]* ;
   lstdfx % local list assignment %                                1B229A(
      [-,-] => % listdf[.ID],RH side %
         blanks[] % put out code to correct S %
         push[con[=0],S] &fcount _ fcount+1
         &val _ eval[*1:2]
         ld[*1:1,=val+1]
         push[con[=LSH(val)18],S]
         &bcount _ val
         &FS listpres @S llist *1:1
         ( ?[-,mt] / lstor[lvar[*1:1],*2] ) ;
   lcons = % local constants %
      #<*,>( const :lconst[1]* ) ;
   lconst % local constant %                                       1B229A(
      [-] => @S local *1:1 *1;
   lastr % local A string %                                        1B229A(
      [-] => $*1(lastr1[*$]);
   lastr1                                                          1B229A
      [astr] => % push space, define, call routine %
         blanks[]
```

```
            lstrp[*1:1,=*L1:2]
            sasgn[lvar[*1:1],cs[*1:2]];
         [srcec] =>
            lstrp[*1:1,=eval[*1:2]];
       blanks => % bump up S to current usec stack position %      1B229AG
         [ ?bcount=1 AOBJP S|A *sysovr*, &FS stkok ]
         [?bcount > 1 ADD S|A = LSH(bcount)18 bcount, &FR stkok]
            % keep track of checks on S with stkok %
         &bcount _ 0;
      litem % local declared item %                                1B229AG
         [-] => $*1(litem1[*$]);
      litem1 % perform runtime assignment to locals %              1B229AG1
         [decbnd] =>
            ld[*1:1,=eval[*1:2]];
         [decv] =>
            blanks[] ld[*1:1,=*01:2]
            $*1:2(
               push[*$,S]
               % push value % )
            &bcount _ 0;
         [itm] =>
            ld[*1:1,=1];
       lstrp [-,-] =>                                              1B229AG1
         &srsize _ *N2 &lsrpush &srwords ld[*1,=srsize + 1];
       ltxp = .ID :ld[1,=2]*;                                      1B229AG1
      stkmov =>                                                    1B229AG1
         blanks[]
         ( ?F stkok / JUMPG S|A *sysovr*, ) :
      ld                                                           1B229AG1
         [recref,-] =>
            ?@ linked *1:1 <*1:1 " ext&local " LOC> &xerror /
            >*1:1 _ fcount + LSH(HASH(*1:2))18
            &fcount _ fcount + *N2
            &bcount _ bcount + *N2
            @S local *1:1 @R reloca *1:1;
         [-,-] =>
            ?@ linked *1 <*1 " ext&local " LOC> &xerror /
            >*1 _ fcount  &fcount _ fcount + *N2
            &bcount _ bcount + *N2
            @S local *1 @R reloca *1;
      formdef                                                      1B229AG1
         [recref] => localck[*1:1]
            >*1:1 _ MASK(fcount)18M+LSH(HASH(*1:2))18
            &fcount _ fcount - 1;
         [-] => localck[*1] >*1 _ MASK(fcount)18M
            &fcount _ fcount - 1;
      localck                                                      1B229AG1
         [-] => ?@ linked *1
            syntx[*1," previously used as global symbol "]/
            @S local *1 @R reloca *1;                             1B229AG164
       refdef = rfdf $(", rfdf);                                   1B229AG1
       rfdf = .ID @S ref &DISCARD;                                 1B229AG1
       ptrdef = ptdf $(", ptdf);                                   1B229AG1
       ptdf = .ID @S ptr &DISCARD;                                 1B229AG2

symbols =                                                         1B23
```

```
      '( .ID ') &LABEL (record / declname) /
      declare;
    labeled = label stat;                                          1823

      label =                                                      18232
        _ '( .ID '):' &LABEL &FS islab
          % ( ?@ extern / @S local ) %
             % labels should be local but need another attribute bit for
             it to work. buildins screw up addresses and con't know
             cifference between MOVE and JRST so can't fix easily %
          :defl[1]* &resetall /
        &FR islab;
      defl [-] => >*1;                                             18232

    comop  [-,-,-] => %exp,exp,op%                                 1823
      (opadc[*1,*2,*3] /
      opadc[*2,*1,*3] /
      stoken[*2] ?AND onereg[*1] dooper[*1,*2,*3] /
      stoken[*1] ?AND onereg[*2] dooper[*2,*1,*3] /
      token[*2] dooper[*1,*2,*3] /
      token[*1] dooper[*2,*1,*3] /
      dooper[*1,*2,*3]) acreset[ac];

      dooper [-,-,-] => %exp,exp,op%                               18233
        load[*1,ac] (
           oper[*2,*3,ac] /
           &acpush adrget[*2] &acpop *N3|OP ac|A,);

    opadc  %exp,isadc?,op%                                         1823
      [-,-,-] => isadc[*2] load[*1,ac] doadc[*2,*3,ac] / &FAIL;

    oper  %token,opcode,accumulator%                               1823
      [lvar,-,-] => *V1:1 0|LH *N2|OP *N3|A M|X,;
      [efport,-,-] => *N2|OP *N3|A M|X 777777B,;
      [rvar,-,-] => *V1:1 0|LH *N2|OP *N3|A,;
      [gvar,-,-] => *V1:1 0|LH *N2|OP *N3|A [?F segflag segx[] ],;
      [indir,-,-] => indirop[*1:1,*2,*3];
      [rvarindexed,-,-]
      [gvarindexed,-,-] => varind[*1] *N2|OP *N3|A,;
      [.SR,-,-] =>
         *N2|OP *N3|A =*S1 [?F segflag segx[] ], @S litsr *1;
      [minus[con],-,-] => *N2|OP *N3|A =-*N1:1:1,;
      [minus[addr],-,-] =>
         ckmad[*1:1] *N2|OP *N3|A =-*V1:1:1, / &FAIL;

      varind  %rvarindexed or gvarindexed%                         18235
        [?[-,minus]] =>
           (?@ defned *1:1 ?AND
              isminuscon[*1:2:1] minuscon[*1:2:1] /
           indexby[*1:2]) *V1:1;
        [-] =>
           (?@ defned *1:1 ?AND
              ispluscon[*1:2] pluscon[*1:2] /
           indexby[*1:2]) *V1:1;

      ckmad [addr] =>                                              18235
```

```
    ?% defned *1:1 ?AND
       ?NOT ?% reloca *1:1 ?AND
       ?NOT ?% local *1:1 /
    &FAIL;

  indirop %exp, opcode, accumulator%                                    1B235
    [lvar,-,-] => *V1:1 0|LH *N2|OP *N3|A I M|X,;
    [gvar,-,-] => *V1:1 0|LH *N2|OP *N3|A I,;
    [gvarindexed,-,-] [rvarindexed,-,-] =>
       varind[*1] *N2|OP *N3|A I,;
    [-,-,-] => refaddr[*1] *N2|OP *N3|A,;

     refaddr  %exp%                                                     1B235M
       [addnode] =>
          ispluscon[*1:2] indexby[*1:1] pluscon[*1:2] /
          ispluscon[*1:1] indexby[*1:2] pluscon[*1:1] /
          indexby[*1];
       [subnode] =>
          isminuscon[*1:2] indexby[*1:1] minuscon[*1:2] /
          indexby[*1];
       [-] => indexby[*1];
     ispluscon                                                          1B235M
       [con] => smlcon[*1:1] / &FAIL;
       [addr] => ?NOT ?% local *1:1 / &FAIL;
     isminuscon                                                         1B235M
       [con] => smlcon[*1:1] / &FAIL;
       [addr] => ckmad[*1] / &FAIL;
     pluscon                                                            1B235M
       [con] => *N1:1;
       [addr] => *V1:1;
     minuscon                                                           1B235N1
       [con] => MASK(-*N1:1)18M;
       [addr] => MASK(-*V1:1)18M;
     indexby %exp%                                                      1B235N1
       [rvar] => *V1:1|X;
       [-] => &acpush load[*1] acnum|X &acpop;

  adrget [-] => load[*1] acnum; %put address of exp%                    1B235

  acget [-] => load[*1] acnum|A;                                        1B235

  token  %things that oper will take%                                   1B235
    [lvar]
    [rvar] [rvarindexed]
    [.SR]
    [indir]
    [gvar] [gvarindexed]
    [minus[con],-,-] => TRUE;
    [minus[addr],-,-] => ckmad[*1:1] / &FAIL;

  isadc                                                                 1B235
    [addr] [adsr] [con] => TRUE;
  coadc  %exp,op,acc%                                                   1B235
    [con,-,-] =>
       *N2|OP *N3|A (smlcon[*1:1] IMMEDIATE *N1:1, / =*N1:1,);
    [addr,-,-] => *N2|OP IMMEDIATE *N3|A coadr[*1:1];
```

```
    [acsr,-,-] => *N2|OP IMMEDIATE *N3|A =*S1:1, @S litsr *1:1;

load                                                              1823
  %exp only -- means load in any register, leaving acnum pointing
  to the register that is used%
     [store] => simstat[*1];
     [lvar]
     [avar] => loacic[*1:1];
     [con] =>
        &actype _ accon  &acval _ *N1:1
        (? &isinac / &pickac doadc[*1,MOVE,acnum] &setac);
     [mt] => &acnum _ ac;
     [rvar] => &acnum _ *V1:1;
     [jsysref] => load[*1,=1] &acnum_1;
     [lgval] [lgdescr] => *1; % they do the load %
     [-] =>
        onereg[*1]  &actype _ acmpty
           &acval _ 0  &pickac
           &rgstrt load[*1,ac] &rgend /
        load[*1,ac] &acnum _ ac;
  %exp,acc%
     [mt,-] => TRUE;
     [#1,-] => MOVEI *N2|A #1, acreset[*2];
     [lvar,-]
     [avar,-] => loacsv[*1:1,*2];
     [con,-] =>
        &actype _ accon  &acval _.*N1:1 &acnum _ *N2
        (? &isin / doadc[*1,MOVE,*2] &setac);
     [boolean,-] =>
        (?F nomess / <"Boolean as operand at " LOC LINE>)
        brf[*1,G#1] MOVEI *N2|A 1, JRST . 2,
        >#1 MOVEI *N2|A, acreset[*2];
     [minus,-] => loadneg[*1:1,*2];
     [-,-] => xload[*1,*2];

  xload                                                           1823E
     [fof,-] =>
        foflsmp[*1,*2] /
        loacad[*1:1,RP] LDB *N2|A doadr[*1:2] acreset[*2];
        [lgval] [lgdescr] => *1 [?acnum#*N2 MOVE *N2|A acnum, &acnum _
        *N2 acreset[*2]];
     [incbyt,-] =>
        ILDB *N2|A doadr[*1:1] acreset[*2] mdfy[*1:1];
     [lgval,-] [lgdescr,-] =>
        *1 [?acnum#*N2 MOVE *N2|A acnum, &acnum_*N2 acreset[*2] ];
     [ch,-] =>
        &savreg push[*1:2,S] pushad[*1:1]
        cllm[=2,"ldchr"]
        (? *N2 = A1 / MOVE *N2|A A1,) &resreg &resetall;
     [rvar,-] =>
        &acnum _ *V1:1 &accum _ *N2
        (? acnum=accum / MOVE *N2|A *V1:1, acreset[*2]);
     [rgcon,-] =>
        &acnum_*N1:1 &accum_*N2
        (? acnum=accum / MOVE *N2|A *N1:1, acreset[*2]);
     [loadac, -] => loadac[*1:1, *2];
```

```
  [-,-] =>
     (doadc[*1,MOVE,*2] / oper[*1,MOVE,*2]) acreset[*2] /
     reqtst[*1,*2] &rgstrt *1 &rgend /
        %want to load expression into register (*2) in as
        direct manner as possible.  The predicate reqtst
        decides if the expression can be evaluated using (*2)
        as the prime accumulator.  It is possible that A1
        and other accumulators may contain intermediate
        results at this point.  The routines acpush and acpop
        must know what is going on and skip from (*2) to the
        next available accumulator.  To do this they must
        know what the next available accumulator was at the
        start of the evaluation of the expression and must
        also have a flag to show what is going on.  The
        routine rgstrt sets this up.  Rgend turns the flag
        off and resets ac to the value prior to the start
        of the eval.%
     *1 [?*N2#ac  MOVE *N2|A ac, &acnum _ *N2 &movac];
foflsmp  % simple partial word store %                          1B236
  [fof[-,"sysl"],-] =>
     token[*1:1] oper[*1:1,HRRZ,*2] acreset[*2]/ &FAIL;
  [fof[-,"sysm"],-] =>
     token[*1:1] oper[*1:1,HLRZ,*2] acreset[*2]/ &FAIL;
loacad                                                          1B236
  [mt,-] => TRUE; %for byte variables%
  [indir,-] => load[*1:1,*2];
  [lvar,-] [gvar,-] [rvar,-] =>
     &actype _ acaddr  &acval _ HASH(*1:1)  &acnum _ *N2
     (? &isin / oper[*1,MOVEI,*2] &setac) modify[*1];
     %"modify" since may change using address%
  [-,-] =>
     (oper[*1,MOVEI,*2] /
     acrget[*1] MOVEI *N2|A,)
     acreset[*2];
loacid [-] =>                                                   1B236
  &actype _ acvar  &acval _ HASH(*1)
  (? &isinac / &pickac MOVE acnum|A doacr[*1] &setac);
loacsv [-,-] => %id,acc%                                        1B236
  &actype _ acvar  &acval _ HASH(*1)  &acnum _ *N2
  (? &isin /
     (? &isinac MOVE *N2|A acnum, /
        MOVE *N2|A doadr[*1])
     &acnum _ *N2 &setac);
loadneg [-,-] =>                                                1B236
  (doadc[*1,MOVN,*2] /
  oper[*1,MOVN,*2] /
  acrget[*1] MOVN *N2|A,) acreset[*2];
onereg                                                          1B236
  %test if exp is simple enough to be loaded using only one
  register%
  [con]
  [lvar] [rvar] [gvar] [incbyt]
  [rvarindexed[-,rvar]]
  [gvarindexed[-,rvar]]
  [adar] [adsr] => TRUE;
  [incir]
```

```
        [fof] => stoken[*1:1] / &FAIL;
        [minus] => onereg[*1:1] / &FAIL;
        [-] => simop[*1] ?AND (
           stoken[*1:1] ?AND onereg[*1:2] /
           stoken[*1:2] ?AND onereg[*1:1]) / &FAIL;
      simop                                                          1B234
        [addnode]
        [subnode]
        [mulnode]
        [andnode]
        [iornode]
        [xornode] => TRUE;
      regtst [-,-] => &acnum _ *N2 (regok[*1] / &FAIL);              1B236
      regok  %can this exp be evaluated directly into acnum%         1B236
        [con] [lvar] [gvar] [addr]
        [aasr] [fof[mt,-]] [incbyt] => TRUE;
        [indir] [minus] => regok[*1:1] / &FAIL;
        [rvar] => ? *V1:1 # acnum / &FAIL;
        [gvarindexed] => regok[*1:2] / &FAIL;
        [jsysref] => ?acnum=1/ &FAIL;
        [-] =>
           simop[*1] ?AND regok[*1:1] ?AND regok[*1:2] / &FAIL;

    acreset [-] => &acnum _ *N1 &rstac;                              1B23
    modify                                                           1B23
      [lvar] [gvar] => mdfy[*1:1];
      [-] => TRUE;
    mdfy [-] =>                                                      1B23
      &actype _ acvar  &acval _ HASH(*1)  &modsvs;

    mt => TRUE;                                                      1B24

    END of L10                                                       1B24


FILE libe10 %  FOR XL10 (l10,) to (L10,libe10,) %
  UNO-OK
  SET rpnum=9, a1num=10, a4num=13, a4next=14, sknum=15;
  DECLARE EXTERNAL
     pglett, mode, fcount, ac=a1num, actype, acval, acnum, accon=1,
     acaddr=2, acvar=3, acmpty=-1, srsize, accs[16], accvs[16],
     bregs[16], bregvs[16], accum, ptrklg, jsysrg=0;
        % jsysrg is the highest register with partial results for a jsys call
       -- see savreg and rstreg, jrstreg %
  SET regnum=10, zero=0;
  SET sf=4000016; REF sf;
  DECLARE STRING sfstr[50];
  DECLARE pkac=a4next;
  REGISTER c=2, r1=1, r2=2, r3=3, w=4, k=6, a1=12 , a2=13, a3=14, a4=15;
  (sfilstr) PROCEDURE; % produce source file string %                  2
     LOCAL sfilp, sfiln;
     IF &sf # 0 THEN
        BEGIN
        w _ sf.L*1B6 + sf.L; produce();
        sfilp _ &sf+1; sfiln _ (sf.L+4)/5;
        WHILE (sfiln_sfiln-1)>=0 DO
```

```
            BEGIN
            w _ [sfilp];
            BUMP sfilp;
            produce();
            END;
         END
      ELSE IF frmnls THEN
         BEGIN
         w _ 0; produce();
         END
      ELSE % not from NLS, get file name %
         BEGIN
         !jfns($sfstr+440788+1, fnumi, 0);
         r1 _ r1 .A 18M;
         sfstr.L _ (r1-$sfstr)*5;
         &sf _ $sfstr;
         sfilstr();
         % put out some nulls at end of string... %
         END;
      &sf _ 0; % in case compiler run again %
      RETURN;
      END.
%ddt support routines%
      DECLARE ddtsr1, ddtsr2, ddtsr3, ddtv, ddtadr, ddtval;
      DECLARE symloc=116B, global=1;
      DECLARE loadlimit = 120B;
      DEFINE loadh = [loadlimit].lh#;
      (word) RECORD                                                    2.
         rh[18], % right half of word %
         lh[18]; % left half of word %
      (ddtdef) PROCEDURE;                                              2.
         LOCAL str, value;
         str _ ddtgsr();
         x _ idhash;  value _ IF .reloca THEN relocc ELSE 0;
         IF ddtenter(str,value + htv[idhash],global) THEN RETURN;
         % put out an error message %
         tcrlf(); wstr($"FAILURE TO DEFINE ",-1);
         wstrh(idhash); BUMP errorn; RETURN END.
      (ddtgsr) PROCEDURE;                                              2.
         x _ idhash;
         ddtsr1.lh _ .htlx; % fake a dummy NLS string header %
         ddtsr1.rh _ .htlx;
         ddtsr2 _ ss[.htpx+1];
         IF .htlx < 5 THEN ddtsr3 _ 0
         ELSE ddtsr3 _ ss[.htpx+2];
         RETURN($ddtsr1) END.
      (ddtget) PROCEDURE;                                              2.
         IF ddtlookup( ddtgsr()) THEN
            BEGIN
            ddtv _ ddtval;
            x _ idhash; .reloca _ FALSE; define(ddtv);
            SKIP RETURN END;
         RETURN END.
      (ddtlookup) PROCEDURE(astr);                                     2.
         LOCAL i,j,v,blockptr;
         REF astr, symloc;
```

```
(modechange) PROCEDURE;                                                 2
   CASE mode OF
      =2, =6: NULL
      ENDCASE mode _ 8 - mode;
   RETURN END.
(pickac) PROCEDURE;                                                     2
   %Called when there is a choice as to which accumulator to use
   next.  The variable acnum is set to the acc which will be used.
   The quantity specified by actype and acval is the target.
   If find in any acc, then use that one.  Else if find in any
   of the acc's in the set on the top of the acstack, then use
   that one.  Else just pick arbitrarily.%
   %set accum to lowest number for available accumulator%
   accum _ IF ac < $a1num THEN [rgstkp-2] %see rgstrt% ELSE ac;
   IF SKIP isinac() AND acnum >= accum THEN RETURN;
   % search for correspondence with stacked accums %
   IF acstkp > $acstk THEN
      BEGIN %see if can find on stack%
      acnum _ $a4next;
      lpt _ acstkp;
      WHILE (acnum _ acnum-1) >= accum DO
         BEGIN
         lpt _ lpt - 2;
         IF [lpt] = actype AND [lpt+1] = acval THEN RETURN;
         END;
      END;
   % look for an empty acc %
   acnum _ $a4next;
   WHILE (acnum _ acnum-1) >= accum DO
      IF accs[acnum] = acmpty THEN RETURN;
   %none empty%
   IF (pkac _ pkac-1) < accum THEN pkac _ $a4num;
   acnum _ pkac;
   RETURN END.
(isin) PROCEDURE;                                                       2
   %If (actype,acval) is in accumulator acnum according to the
   information in accs and accvs, then return true, else fail.%
   IF acnum NOT IN [$rpnum,$a4num] OR
      accs[acnum] # actype OR
      accvs[acnum] # acval THEN RETURN;
   SKIP RETURN END.
(isinac) PROCEDURE;                                                     2
   %True if the quantity specified by actype and acval is in any
   of the accumulators from RP to A4.  If it is found then acnum
   is left pointing to it.%
   acnum _ $a4num;
   DO IF accs[acnum] = actype AND
      accvs[acnum] = acval THEN SKIP RETURN
   UNTIL (acnum _ acnum-1) < $rpnum;
   RETURN END.
(setac) PROCEDURE;                                                      2
   %quantity specified by actype and acval is now known to be
   in the accumulator acnum.%
   IF acnum IN [$rpnum,$a4num] THEN
      BEGIN
      accs[acnum] _ actype;
```

```
      accvs[acnum] _ acval;
      END;
   RETURN END.
(stoac) PROCEDURE;                                                    2
   %Have stored accumulator acnum into the identifier acval.%
   actype _ acvar;
   modsvs();
   setac();
   RETURN END.
(movac) PROCEDURE;                                                    2
   %Have moved accumulator ac to accumulator acnum.%
   IF acnum IN [$rpnum,$a4num] THEN
      IF ac IN [$rpnum,$a4num] THEN
         BEGIN
         accs[acnum] _ accs[ac];
         accvs[acnum] _ accvs[ac];
         END
      ELSE
         BEGIN
         accs[acnum] _ acmpty;
         accvs[acnum] _ 0;
         END;
   RETURN END.
(modsvs) PROCEDURE;                                                   2
   %Identifier specified by acval has been changed.  Any
   accumulator that was holding it is set to undefined.%
   accum _ $rpnum;
   WHILE accum < $a4next DO BEGIN
      IF accvs[accum] = acval AND
         accs[accum] = acvar THEN
         BEGIN
         accs[accum] _ acmpty;
         accvs[accum] _ 0;
         END;
      BUMP accum END;
   RETURN END.
(resetall) PROCEDURE;                                                 2
   %Contents of all accumulators set to undefined.%
   accum _ $rpnum;
   WHILE accum < $a4next DO
      BEGIN
      accs[accum] _ acmpty;
      accvs[accum] _ 0;
      BUMP accum;
      END;
   RETURN END.
(rstac) PROCEDURE;                                                    2
   %Accumulator acnum set to undefined.%
   IF acnum IN [$rpnum,$a4num] THEN
      BEGIN
      accs[acnum] _ acmpty;
      accvs[acnum] _ 0;
      END;
   RETURN END.
DECLARE EXTERNAL
   wbitc, %number of bits used in current word%
```

```
   rwrdc, %number of words used in current record%
   fbitc; %number of bits used in current field%
(setfld) PROCEDURE;                                                    2
   %Checks to see is next field will overflow the current word
   of the record that is being declared.  If that is the case,
   then moves to the next word.%
   IF fbitc + wbitc > 36 THEN BEGIN wbitc _ 0; BUMP rwrdc END;
   RETURN END.
DECLARE lststk[30], lstrkp;
(ldrst) PROCEDURE;                                                     2A
   fcount _ 1;
   lstrkp _ $lststk;
   RETURN END.
(lsrpush) PROCEDURE;                                                   2A
   IF lstrkp >= $lstrkp THEN
      BEGIN
      setern(1229);
      GOTO qserr;
      END;
   [lstrkp] _ fcount;
   BUMP lstrkp;
   [lstrkp] _ srsize;
   BUMP lstrkp;
   RETURN END.
(lsrpop) PROCEDURE;                                                    2A
   IF lstrkp = $lststk THEN RETURN;
   BUMP DOWN lstrkp;
   srsize _ [lstrkp];
   BUMP DOWN lstrkp;
   fcount _ [lstrkp];
   SKIP RETURN END.
DECLARE acstk[200], acstkp=acstk;
(pushr) PROCEDURE;                                                     2A
   %Contents of the arrays accs and accvs are pushed on stack
   during production of code for logical test. This shows the
   contents of the accumulators if fall thru the test rather than
   branch.%
   accum _ $rgnum;
   IF acstkp + $regnum >= $acstkp THEN
      BEGIN
      setern(1009);
      GOTO qserr;
      END;
   WHILE accum < $a4next DO
      BEGIN
      [acstkp] _ accs[accum];
      BUMP acstkp;
      [acstkp] _ accvs[accum];
      BUMP acstkp,accum;
      END;
   RETURN END.
(pushb) PROCEDURE;                                                     2A
   %Contents of the arrays bregs and bregvs are pushed on stack
   during production of code for logical test. This shows the
   contents of the accumulators if branch rather than fall thru
   the test.%
```

```
    accum _ $rpnum;
    IF acstkp + $recnum >= $acstkp THEN
        BEGIN
        setern(1009);
        GOTO oserr;
        END;
    WHILE accum < $a4next DO
        BEGIN
        [acstkp] _ bregs[accum];
        BUMP acstkp;
        [acstkp] _ bregvs[accum];
        BUMP acstkp,accum;
        END;
    RETURN END.
(setb) PROCEDURE;                                                    2A
    %set bregs to accs when code for test is such that contents
    of registers the same whether fall thru or branch%
    accum _ $rpnum;
    WHILE accum < $a4next DO
        BEGIN
        bregs[accum] _ accs[accum];
        bregvs[accum] _ accvs[accum];
        BUMP accum;
        END;
    RETURN END.
(loadb) PROCEDURE;                                                   2A
    %set accs to bregs when reach place where branched to from
    logical test.%
    accum _ $rpnum;
    WHILE accum < $a4next DO
        BEGIN
        accs[accum] _ bregs[accum];
        accvs[accum] _ bregvs[accum];
        BUMP accum;
        END;
    RETURN END.
(poopr) PROCEDURE;                                                   2A
    %Restore values of accs and accvs from acstk, removing top
    set from acstk in the process%
    accum _ $a4next;
    WHILE (accum _ accum-1) >= $rpnum DO
        BEGIN
        acstkp _ acstkp - 1;
        accvs[accum] _ [acstkp];
        acstkp _ acstkp - 1;
        accs[accum] _ [acstkp];
        END;
    RETURN END.
(xchr) PROCEDURE;                                                    2A
    %Exchange accs and accvs with top set of values on acstk%
    accum _ $a4next;
    WHILE (accum _ accum-1) >= $rpnum DO
        BEGIN
        acstkp _ acstkp - 1;
        [acstkp] _ accvs[accum] := [acstkp];
        acstkp _ acstkp - 1;
```

```
      [acstkp] _ accs[accum] := [acstkp];
      END;
   acstkp _ acstkp + $regnum;
   RETURN END.
(compr) PROCEDURE;                                                    2A
   %Compare accs and accvs with top set of values on acstk.  Where
   a register is not the same set it to undefined (acmpty) in
   accs.  Top set of values removed from acstk in the process.%
   accum _ $a4next;
   WHILE (accum _ accum-1) >= $rpnum DO
      BEGIN
      acstkp _ acstkp - 2;
      IF [acstkp] # accs[accum] OR
         [acstkp+1] # accvs[accum] THEN
         BEGIN
         accs[accum] _ acmpty;
         accvs[accum] _ 0;
         END;
      END;
   RETURN END.
(compb) PROCEDURE;                                                    2A
   %Like compr except with bregs instead of accs.%
   accum _ $a4next;
   WHILE (accum _ accum-1) >= $rpnum DO
      BEGIN
      acstkp _ acstkp - 2;
      IF [acstkp] # bregs[accum] OR
         [acstkp+1] # bregvs[accum] THEN
         BEGIN
         bregs[accum] _ acmpty;
         bregvs[accum] _ 0;
         END;
      END;
   RETURN END.
DECLARE EXTERNAL regflg;
DECLARE rgstk[24], rgstkp=rgstk;
(rgstrt) PROCEDURE;                                                   2A
   %Want to evaluate an expression and have result end up in
   accumulator specified by acnum.  To do this set ac to acnum,
   but must also be able to call acpush and acpop during the
   production of code for the expression and be able to restore
   ac and acnum when finish.  Thus stack current value of ac,
   current accumulator which go to next when call acpush (which
   may not be ac if are already in midst of producing code for
   exp to go in some other special register), and save the value
   of acnum.%
   IF rgstkp+3 > $rgstkp THEN
      BEGIN
      setern(100);
      GOTO qserr;
      END;
   [rgstkp] _ ac; BUMP rgstkp;
   [rgstkp] _ IF ac >= $a1num THEN ac ELSE [rgstkp-3]; BUMP
   rgstkp;
   [rgstkp] _ ac _ acnum; BUMP rgstkp;
   BUMP regflg; RETURN END.
```

```
(rgend) PROCEDURE;                                                    2A
   %When finish producing code for expression to be loaded into
   special register call rgend to undo the work of rgstrt.%
   acnum _ [rgstkp-1]; rgstkp _ rgstkp - 3; ac _ [rgstkp];
   BUMP DOWN regflg; RETURN END.
(setern) PROC(ern);                                                   2A
   a1 _ ern;
   !MOVEM a1,40B;
   RETURN END.
DECLARE pstk[10], pstkp=pstk;
(pac) PROCEDURE;                                                      2A
   IF pstkp >= $pstkp THEN
      BEGIN
      setern(122B);
      GOTO qserr;
      END;
   [pstkp] _ ac; BUMP pstkp; RETURN END.
(ppac) PROCEDURE;                                                     2A
   BUMP DOWN pstkp;
   accum _ [pstkp];  RETURN END.
(acpush) PROCEDURE;                                                   2A
   %Change ac to specify a new accumulator to be used for
   evaluation of expressions.%
   IF regflg AND ac = [rgstkp-1] THEN
      BEGIN
      BUMP [rgstkp-2];
      ac _ [rgstkp-2];
      END
   ELSE BUMP ac;
   IF ac <= $a4num THEN RETURN;
   setern(103);
   GOTO qserr; END.
(acpop) PROCEDURE;                                                    2A
   %Restore ac to value prior to last call to acpush.%
   IF regflg AND ac = [rgstkp-2] THEN
      BEGIN
      BUMP DOWN [rgstkp-2];
      ac _ [rgstkp-1];
      END
   ELSE ac _ ac - 1;
   RETURN END.
DECLARE saveac[10], svacp=saveac;
(savreg) PROCEDURE;                                                   2A
   %Save any intermediate results which may be in accumulators
   when make procedure call by pushing them on the stack then
   set ac back to A1.%
   IF $svacp <= svacp THEN
      BEGIN
      setern(102);
      GOTO qserr;
      END;
   [svacp] _ ac; BUMP svacp;
   WHILE (ac _ ac-1) >= $a1num DO BEGIN
      %output PUSH S,AC%
      w _ 2615B + $sknum * 4B7 + ac;
      <LIBE,produce>() END;
```

```
    ac _ 0;
    WHILE (ac_ac+1) <= jsysrg DO BEGIN
        % output PUSH S,ac %
        w _ 26189 + $sknum*487 + ac;
        <LIBE,produce>() END;
    ac _ $a1num; %back to A1% RETURN END.
(resreg) PROCEDURE;                                                    2
    %Restore any registers which were saved by savreg.%
    BUMP DOWN svacp;
    IF (ac _ [svacp]) NOT= $a1num THEN BEGIN
        %output MOVE AC,A1%
        w _ 20089 + ac * 487 + $a1num; <LIBE,produce>() END;
    IF (accum_jsysrg) THEN
        WHILE accum>0 DO BEGIN
            % output POP S,ac %
            w _ 26289 + $sknum*487 + accum;
            <LIBE,produce>();
            accum _ accum-1 END;
    accum _ $a1num;
    WHILE ac > accum DO BEGIN
        %output POP S,ACCUM%
        w _ 26289 + $sknum * 487 + accum;
        <LIBE,produce>();
        BUMP accum END;
    RETURN END.
(jrstreg) PROCEDURE;                                                   2A
    % like resreg but to be called after producing JSYS %
    BUMP DOWN svacp;
    IF (ac _ [svacp]) NOT= 1 THEN BEGIN
        %output MOVE AC,1%
        w _ 20089 + ac * 487 + 1; <LIBE,produce>() END;
    IF (accum_jsysrg) THEN
        WHILE accum>0 DO BEGIN
            % output POP S,ac %
            w _ 26289 + $sknum*487 + accum;
            <LIBE,produce>();
            accum _ accum-1 END;
    accum _ $a1num;
    WHILE ac > accum DO BEGIN
        %output POP S,ACCUM%
        w _ 26289 + $sknum * 487 + accum;
        <LIBE,produce>();
        BUMP accum END;
    RETURN END.
DECLARE lpstck[200], lpptr=lpstck;
(lpush) PROC;                                                          2B
    %When enter a loop push gn1, gn2, and room for accs.  The accs
    on the stack (lpstck) will define the contents of the
    accumulators whenever exit the loop.  (The -2 signifies that
    there have been no exits produced yet.)%
    IF lpptr+$regnum+2 > $lpptr THEN
        BEGIN
        setern(102);
        GOTO qserr;
        END;
    [lpptr] _ gn1; BUMP lpptr; [lpptr] _ gn2; BUMP lpptr;
```

```
   [lpptr] _ -2; lpptr _ lpptr + $regnum;
   resetall(); RETURN END.
(lodxit) PROC;                                                    2E
   %Called when finish a DO statement%
   lplev _ 1; lpexit(); RETURN END.
(lpexit) PROC;                                                    2E
   %Accs reflect registers on exit from loop%
   xitype _ 0;
   lpx();
   RETURN END.
DECLARE EXTERNAL lplev;
(lpbxit) PROC;                                                    2E
   %Bregs reflect registers on exit from loop%
   BUMP xitype; lplev _ 1;
   lpx();
   RETURN END.
DECLARE xitype, lpt;
(lpx) PROC;                                                       2E
   %Called whenever produce code for exiting a loop.  Lplev
   determines which stacked set of accumulators is to be compared
   to either accs or bregs depending on which reflects the
   registers for this particular exit.  If -2 is still on the
   stack then this is the first exit, so just copy rather than
   compare.%
   accum _ $regnum;
   IF (lplev < 1) OR
      (lpt _ lpptr-lplev*($regnum+2)) < $lpstck THEN
         BEGIN
         setern(110);
         GOTO qserr;
         END;
   lpt _ lpt + 2;
   IF [lpt] = -2 THEN %copy%
      IF xitype = 0 THEN
         WHILE accum < $a4next DO
            BEGIN
            [lpt] _ accs[accum];
            BUMP lpt;
            [lpt] _ accvs[accum];
            BUMP lpt, accum;
            END
      ELSE
         WHILE accum < $a4next DO
            BEGIN
            [lpt] _ bregs[accum];
            BUMP lpt;
            [lpt] _ bregvs[accum];
            BUMP lpt, accum;
            END
   ELSE %compare%
      WHILE accum < $a4next DO
         BEGIN
         IF xitype = 0 AND
            ([lpt] # accs[accum] OR [lpt+1] # accvs[accum]) OR
            xitype # 0 AND
            ([lpt] # bregs[accum] OR [lpt+1] # bregvs[accum])
```

```
            THEN
               BEGIN
               [lpt] _ acmpty;
               [lpt+1] _ 0;
               END;
         lpt _ lpt + 2;
         BUMP accum;
         END;
   RETURN END.
(loop) PROC;                                                    2F
   %When reach end of a loop, set accs to what know the registers
   will hold based on their contents at the various exit points
   within the loop.%
   accum _ $a4next;
   WHILE (accum _ accum-1) >= $rpnum DO
      BEGIN
      BUMP DOWN lpptr;
      accvs[accum] _ [lpptr];
      BUMP DOWN lpptr;
      accs[accum] _ [lpptr];
      END;
   lpptr _ lpptr - 2;  RETURN END.
(lget1) PROCEDURE;                                              2E
   lget();
   gn1 _ [lpt];
   RETURN END.
(lget2) PROCEDURE;                                              2F
   lget();
   gn2 _ [lpt + 1];
   RETURN END.
(lget) PROC;                                                    2E
   IF (lplev < 1) OR
      (lpt _ lpptr-lplev*($regnum+2)) < $lpstck THEN
         BEGIN
         setern(0);
         GOTO gxerr;
         END;
   RETURN END.
DECLARE cstck[200], cpptr=cstck, castmp;
(cstrt) PROCEDURE;                                              2E
   %Called when enter the exp evaluation for a CASE statement.
   Saves gn1.%
   IF ¶cpptr <= cpptr THEN
      BEGIN
      setern(100);
      GOTO gserr;
      END;
   [cpptr] _ gn1;
   BUMP cpptr;
   resetall();
   RETURN END.
(cpusn) PROC;                                                   2F
   %Called when enter the tests for a CASE statement.  Saves gn1,
   gn2 and reserves room on cstck for accs.  The accs on stack
   will be updated to show what will be in registers when fall
   thru the case.%
```

```
   IF cpptr+$regnum+2 > $cpptr THEN
      BEGIN
      setern(100);
      GOTO cserr;
      END;
   [cpptr] _ pn1;
   BUMP cpptr;
   [cpptr] _ pn2;
   BUMP cpptr;
   [cpptr] _ -2;
   cpptr _ cpptr + $regnum;
   resetall();
   RETURN END.
(csx) PROCEDURE; %case exit%                                          2
   accum _ $rpnum;
   IF [lpt] = -2 THEN %copy%
      WHILE accum < $a4next DO
         BEGIN
         [lpt] _ accs[accum];
         BUMP lpt;
         [lpt] _ accvs[accum];
         BUMP lpt,accum;
         END
   ELSE %compare%
      WHILE accum < $a4next DO
         BEGIN
         IF [lpt] # accs[accum] OR
            [lpt+1] # accvs[accum] THEN
            BEGIN
            [lpt] _ acmpty;
            [lpt+1] _ 0;
            END;
         lpt _ lpt + 2;
         BUMP accum;
         END;
   RETURN END.
(caseck) PROC;                                                        2
   %Each time procuce a branch to the end of the case statement
   (i.e. at end of each case that does not end in an unconditional
   transfer) update info on cstck.%
   lpt _ cpptr-$regnum;
   csx();
   RETURN END.
(casxit) PROCEDURE;                                                   2
   %exiting from case.  level given by lplev.  adjust what know
   about accs%
   IF (lplev < 1) OR
      (lpt _ cpptr-lplev*($regnum+3)) < $cstck THEN
         BEGIN
         setern(0);
         GOTO qxerr;
         END;
   lpt _ lpt + 3;
   csx();
   RETURN END.
(cpop) PROC;                                                          2
```

```
   %When reach end of case statement set accs to what know then
   to be in the registers based on the contents at the various
   points where have branched after a case.  If the ENDCASE did
   not end in an unconditional transfer must call caseck again.%
   IF NOT ut THEN caseck();
   accum _ $a+next;
   WHILE (accum _ accum-1) >= $rgnum DO
      BEGIN
      BUMP DOWN cpptr;
      accvs[accum] _ [cpptr];
      BUMP DOWN cpptr;
      accs[accum] _ [cpptr];
      END;
   cpptr _ cpptr - 3;
   RETURN END.
(cgetsc) PROCEDURE;                                                    2E
   cget();
   gn1 _ [lpt];
   RETURN END.
(cgetst) PROCEDURE;                                                    2E
   cget();
   gn1 _ [lpt+1];
   RETURN END.
(cgetn2) PROCEDURE;                                                    2E
   cget();
   gn2 _ [lpt+2];
   RETURN END.
(cget) PROCEDURE;                                                      2E
   IF (lplev < 1) OR
      (lpt _ cpptr - lplev*($regnum+3)) < $cstck
      THEN
         BEGIN
         setern(0);
         GOTO cxerr;
         END;
   RETURN END.
(noddms)PROC;                                                          2E
   %Set noddt attr in macro definition string [name pointed to by idhash] i
   htax = 0%
   LOCAL svx;
   svx _ x;
   x _ idhash;
   macsth();
   IF .htax = 0 THEN .noddt _ 1;
   x _ svx;
   RETURN;
   END.
(macstr)PROC;                                                          2E
   %Top of kstack = hash number of id
      By here, we expect a string followed by '# %
   IF c = '# THEN
      entrc(c _ SP) %Emptry String...replace with one space%
   ELSE WHILE c # '# DO inc(entrc());
   entrs();
   %X points to string, 0[k] .A 18M is x for ic%
   macdef(zero[k] .A 18M);
```

```
      inc(); delb();
      SKIP RETURN;
      END.
   FINISH of library
FILE libe109 %  FOR L109 (arcsubsys,l109,) to (l10,libe109,) %
   UNO-OK
   ALLOW!
   SET rgnum=9, a1num=10, a4num=13, a4next=14, sknum=15;
   DECLARE EXTERNAL
      pgleft, mode, fcount, ac=a1num, actype, acval, acnum, accon=1,
      acadcr=2, acvar=3, acmpty=-1, srsize, accs[16], accvs[16],
      bregs[16], bregvs[16], accum, ptrklg, jsysrg=0;
          % jsysrg is the highest register with partial results for a jsys cal
         -- see savreg and rstreg, jrstreg %
   SET regnum=10, zero=0;
   SET sf=400001B; REF sf;
   DECLARE STRING sfstr[50];
   DECLARE pkac=a4next;
   REGISTER c=2, r1=1, x=1, r2=2, r3=3, w=4, k=6, a1=12 , a2=13, a3=14, a4=15;
   (sfilstr) PROCEDURE; % produce source file string %
      LOCAL sfilp, sfiln, savr3;
      savr3 _ R3;
      IF &sf # 0 THEN
         BEGIN
         w _ sf.L*186 + sf.L; produce();
         sfilp _ &sf+1; sfiln _ (sf.L+4)/5;
         WHILE (sfiln_sfiln-1)>=0 DO
            BEGIN
            w _ [sfilp];
            BUMP sfilp;
            produce();
            END;
         END
      ELSE IF frmnls THEN
         BEGIN
         w _ 0; produce();
         END
      ELSE % not from NLS, get file name %
         BEGIN
         !jfns($sfstr+440788+1, fnumi, 0);
         r1 _ r1 .A 18M;
         sfstr.L _ (r1-$sfstr)*5;
         &sf _ $sfstr;
         sfilstr();
         % put out some nulls at end of string... %
         END;
      &sf _ 0; % in case compiler run again %
      R3 _ savr3;
      RETURN;
      END.
   %ddt support routines%
      DECLARE ddtsr1, ddtsr2, ddtsr3, ddtv, ddtadr, ddtval;
      DECLARE symloc=116B, global=1;
      ADDRESS loadlimit = 1208;
      (word) RECORD
         rh[18], % right half of word %
```

```
    lh[18]; % left half of word %
(ddtdef) PROCEDURE;                                                    3K
    LOCAL str, value;
    str _ ddtgsr();
    x _ idhash;  value _ IF .reloca THEN relocc ELSE 0;
    IF ddtenter(str,value + htvl[idhash],global) THEN RETURN;
    % put out an error message %
    tcrlf(): wstr($"FAILURE TO DEFINE ",-1);
    wstrh(idhash): BUMP errorn; RETURN END.
(ddtgsr) PROCEDURE;                                                    3K
    x _ idhash;
    ddtsr1.lh _ .htlx; % fake a dummy NLS string header %
    ddtsr1.rh _ .htlx;
    ddtsr2 _ ss[.htpx+1];
    IF .htlx < 5 THEN ddtsr3 _ 0
    ELSE ddtsr3 _ ss[.htpx+2];
    RETURN($ddtsr1) END.
(ddtget) PROCEDURE;                                                    3K
    IF ddtlookup( ddtgsr()) THEN
        BEGIN
        ddtv _ ddtval;
        x _ idhash; .reloca _ FALSE; define(ddtv);
        RETURN END;
    RETURN[FALSE] END.
(ddtlookup) PROCEDURE(astr);                                           3K
    LOCAL i,j,v,blockptr;
    REF astr, symloc;
    % convert symbol to a radix 50 value %
    v _ ddtname(&astr, global);
    % now search down through the symbol table %
        % this code was removed for the L10 vesion %


    % now search the whole damn table %
    j _ symloc.rh; % lowest symbtol table address %
    blockptr _ (j - symloc.lh) .A 18M - 1; % highest sym table address %
    WHILE blockptr > j DO
        BEGIN
        IF ddtsrchblk( blockptr, v ) THEN
            RETURN( TRUE);
        blockptr _ (blockptr + [blockptr].lh) .A 18M;
        END;
    RETURN(FALSE);
    END.
(ddtname) PROCEDURE( astr, type);                                      3K
    % this routine returns a radix 50 interpretation of the string and
    incorporates the type information with it to produce a ddtsymbol %
    LOCAL lvalue, astptr, i ,j;
    REF astr;
    % make radix 50 value from symbol %
    lvalue _ 0; astptr _ &astr .V 440768 + 1;
    FOR i _ 1 UP UNTIL > MIN(astr.L,6) DO
        BEGIN
        j _ IF (j _ |astptr-47) > 10
            THEN IF j > 49 THEN j-39 %lower case% ELSE  j -7
            ELSE j;
```

```
        lvalue _ 40 * lvalue + j;
        END;
    % set appropriate type mask %
    lvalue _ lvalue .V (IF type = global THEN 4B10 ELSE 10B10);
    RETURN (lvalue);
    END.
(ddtenter) PROCEDURE (astr, lvalue, type);                                  3K1
    % this routine enters a symbol and value into the ddt symbol table
    the current range of the symbol table is contained in "symloc"
    the highest load address is contained in "loadlimit.LH". %
    LOCAL ptr, i, blockptr;
    REF symloc ,astr, blockptr;
    % find end of current block -- look for a word of all zeroes %
    &blockptr _ symloc.rh + 2;
    UNTIL blockptr = 0 DO
        &blockptr _ &blockptr + 2; % try next entry %
    &blockptr _ &blockptr - 1;
    % lh of [blockptr] containes -l where l is the length of the block%
    ptr _ (&blockptr + blockptr.lh -1) .A 18M;
    % check for table overflow %
    IF ptr < loadlimit.LH THEN RETURN( FALSE ); % out of space %
    % update symloc to reflect new extent of table %
    symloc _ symloc - 2000002B;
    % update the first header word %
    blockptr _ blockptr - 2B6;
    % make new header for block
      first word is zeroes
      second word contains the highest address in the block %
    [ptr] _ 0;
    BUMP ptr;
    [ptr] _ MAX([ptr+2],lvalue);
    % add new entry to block %
    BUMP ptr;
    [ptr] _ ddtname( &astr, type);
    BUMP ptr;
    [ptr] _ lvalue;
    RETURN( TRUE );
    END.
(ddtsrchblk) PROCEDURE( blockptr, symbolhash);                              3K1
    LOCAL lowaddr, highaddr;
    lowaddr _ (blockptr + [blockptr].lh + 3) .A 18M;
    highaddr _ blockptr - 3;
    WHILE lowaddr <= highaddr DO
        IF [lowaddr] = symbolhash
            THEN
                BEGIN
                ddtadr _ lowaddr+1;
                ddtval _ [ddtadr];
                RETURN( TRUE)
                END
            ELSE lowaddr _ lowaddr + 2;
    RETURN (FALSE);
    END.
(srwords) PROCEDURE;
    %number of words needed for a string of srsize.%
    srsize _ (srsize + 5) / 5;
```

```
   RETURN END.
(pgsz) PROCEDURE;
   IF (pgleft _ (lc + pgleft) .A 777B) # 0 THEN
      pgleft _ 512 - pgleft;
   RETURN END.
(erreset) PROCEDURE;
   ac _ $a1num;
   pkac _ $a4next;
   acstkp _ $acstk;
   rgstkp _ $rgstk;
   pstkp _ $pstk;
   svacp _ $saveac;
   lpptr _ $lpstck;
   cpptr _ $cstck;
   RETURN END.
(modechange) PROCEDURE;
   CASE mode OF
      =2, =6: NULL
      ENDCASE mode _ 8 - mode;
   RETURN END.
(pickac) PROCEDURE;
   %Called when there is a choice as to which accumulator to use
   next.  The variable acnum is set to the acc which will be used.
   The quantity specified by actype and acval is the target.
   If find in any acc, then use that one.  Else if find in any
   of the acc's in the set on the top of the acstack, then use
   that one.  Else just pick arbitrarily.%
   %set accum to lowest number for available accumulator%
   LOCAL f;
   accum _ IF ac < $a1num THEN [rgstkp-2] %see rgstrt% ELSE ac;
   isinac(:[f]);
   IF f AND acnum >= accum THEN RETURN;
   % search for correspondence with stacked accums %
   IF acstkp > $acstk THEN
      BEGIN %see if can find on stack%
      acnum _ $a4next;
      lpt _ acstkp;
      WHILE (acnum _ acnum-1) >= accum DO
         BEGIN
         lpt _ lpt - 2;
         IF [lpt] = actype AND [lpt+1] = acval THEN RETURN;
         END;
      END;
   % look for an empty acc %
   acnum _ $a4rext;
   WHILE (acnum _ acnum-1) >= accum DO
      IF accs[acnum] = acmpty THEN RETURN;
   %none empty%
   IF (pkac _ pkac-1) < accum THEN pkac _ $a4num;
   acnum _ pkac;
   RETURN END.
(isin) PROCEDURE;
   %If (actype,acval) is in accumulator acnum according to the
   information in accs and accvs, then return true, else fail.%
   IF acnum NOT IN [$rgnum,$a4num] OR
      accs[acnum] # actype OR
```

```
      accvs[acnum] # acval THEN RETURN[FALSE];
   RETURN END.
(isinac) PROCEDURE;
   %True if the quantity specified by actype and acval is in any
   of the accumulators from RP to A4.  If it is found then acnum
   is left pointing to it.%
   acnum _ $a4num;
   DO IF accs[acnum] = actype AND
      accvs[acnum] = acval THEN RETURN
   UNTIL (acnum _ acnum-1) < $rpnum;
   RETURN[FALSE] END.
(setac) PROCEDURE;
   %quantity specified by actype and acval is now known to be
   in the accumulator acnum.%
   IF acnum IN [$rpnum,$a4num] THEN
      BEGIN
      accs[acnum] _ actype;
      accvs[acnum] _ acval;
      END;
   RETURN END.
(stoac) PROCEDURE;
   %Have stored accumulator acnum into the identifier acval.%
   actype _ acvar;
   modsvs();
   setac();
   RETURN END.
(movac) PROCEDURE;
   %Have moved accumulator ac to accumulator acnum.%
   IF acnum IN [$rpnum,$a4num] THEN
      IF ac IN [$rpnum,$a4num] THEN
         BEGIN
         accs[acnum] _ accs[ac];
         accvs[acnum] _ accvs[ac];
         END
      ELSE
         BEGIN
         accs[acnum] _ acmpty;
         accvs[acnum] _ 0;
         END;
   RETURN END.
(modsvs) PROCEDURE;
   %Identifier specified by acval has been changed.  Any
   accumulator that was holding it is set to undefined.%
   accum _ $rpnum;
   WHILE accum < $a4next DO BEGIN
      IF accvs[accum] = acval AND
         accs[accum] = acvar THEN
         BEGIN
         accs[accum] _ acmpty;
         accvs[accum] _ 0;
         END;
      BUMP accum END;
   RETURN END.
(resetall) PROCEDURE;
   %Contents of all accumulators set to undefined.%
   accum _ $rpnum;
```

```
    WHILE accum < $a4next DO
        BEGIN
        accs[accum] _ acmpty;
        accvs[accum] _ 0;
        BUMP accum;
        END;
    RETURN END.
(rstac) PROCEDURE;                                                  3
    %Accumulator acnum set to undefined.%
    IF acnum IN [$rpnum,$a4num] THEN
        BEGIN
        accs[acnum] _ acmpty;
        accvs[acnum] _ 0;
        END;
    RETURN END.
DECLARE EXTERNAL
    wbitc, %number of bits used in current word%
    rwrdc, %number of words used in current record%
    fbitc; %number of bits used in current field%
(setfld) PROCEDURE;                                                 3
    %Checks to see is next field will overflow the current word
    of the record that is being declared.  If that is the case,
    then moves to the next word.%
    IF fbitc + wbitc > 36 THEN BEGIN wbitc _ 0; BUMP rwrdc END;
    RETURN END.
DECLARE lststk[30], lstrkp;
(lcrst) PROCEDURE;                                                  3A
    fcount _ 1;
    lstrkp _ $lststk;
    RETURN END.
(lsrpush) PROCEDURE;                                                3A
    IF lstrkp >= $lstrkp THEN
        BEGIN
        setern(1229);
        GOTO qserr;
        END;
    [lstrkp] _ fcount;
    BUMP lstrkp;
    [lstrkp] _ srsize;
    BUMP lstrkp;
    RETURN END.
(lsrpop) PROCEDURE;                                                 3A
    IF lstrkp = $lststk THEN RETURN[FALSE];
    BUMP DOWN lstrkp;
    srsize _ [lstrkp];
    BUMP DOWN lstrkp;
    fcount _ [lstrkp];
    RETURN END.
DECLARE acstk[200], acstkp=$acstk;
(pushr) PROCEDURE;                                                  3A
    %Contents of the arrays accs and accvs are pushed on stack
    during production of code for logical test. This shows the
    contents of the accumulators if fall thru the test rather than
    branch.%
    accum _ $rpnum;
    IF acstkp + $regnum >= $acstkp THEN
```

```
        BEGIN
        setern(1009);
        GOTO qserr;
        END;
    WHILE accum < $a4next DO
        BEGIN
        [acstkp] _ accs[accum];
        BUMP acstkp;
        [acstkp] _ accvs[accum];
        BUMP acstkp,accum;
        END;
    RETURN END.
(pushb) PROCEDURE;                                              3A
    %Contents of the arrays bregs and bregvs are pushed on stack
    during production of code for logical test. This shows the
    contents of the accumulators if branch rather than fall thru
    the test.%
    accum _ $rpnum;
    IF acstkp + $regnum >= $acstkp THEN
        BEGIN
        setern(1009);
        GOTO qserr;
        END;
    WHILE accum < $a4next DO
        BEGIN
        [acstkp] _ bregs[accum];
        BUMP acstkp;
        [acstkp] _ bregvs[accum];
        BUMP acstkp,accum;
        END;
    RETURN END.
(setb) PROCEDURE;                                               3A
    %set bregs to accs when code for test is such that contents
    of registers the same whether fall thru or branch%
    accum _ $rpnum;
    WHILE accum < $a4next DO
        BEGIN
        bregs[accum] _ accs[accum];
        bregvs[accum] _ accvs[accum];
        BUMP accum;
        END;
    RETURN END.
(loadb) PROCEDURE;                                              3A
    %set accs to bregs when reach place where branched to from
    logical test.%
    accum _ $rpnum;
    WHILE accum < $a4next DO
        BEGIN
        accs[accum] _ bregs[accum];
        accvs[accum] _ bregvs[accum];
        BUMP accum;
        END;
    RETURN END.
(popr) PROCEDURE;                                               3A
    %Restore values of accs and accvs from acstk, removing top
    set from acstk in the process%
```

```
   accum _ $a4next;
   WHILE (accum _ accum-1) >= $rpnum DO
      BEGIN
      acstkp _ acstkp - 1;
      accvs[accum] _ [acstkp];
      acstkp _ acstkp - 1;
      accs[accum] _ [acstkp];
      END;
   RETURN END.
(xchr) PROCEDURE;                                                    3/
   %Exchange accs and accvs with top set of values on acstk%
   accum _ $a4next;
   WHILE (accum _ accum-1) >= $rpnum DO
      BEGIN
      acstkp _ acstkp - 1;
      [acstkp] _ accvs[accum] := [acstkp];
      acstkp _ acstkp - 1;
      [acstkp] _ accs[accum] := [acstkp];
      END;
   acstkp _ acstkp + $regnum;
   RETURN END.
(compr) PROCEDURE;                                                   3/
   %Compare accs and accvs with top set of values on acstk.  Where
   a register is not the same set it to undefined (acmpty) in
   accs.  Top set of values removed from acstk in the process.%
   accum _ $a4next;
   WHILE (accum _ accum-1) >= $rpnum DO
      BEGIN
      acstkp _ acstkp - 2;
      IF [acstkp] # accs[accum] OR
         [acstkp+1] # accvs[accum] THEN
         BEGIN
         accs[accum] _ acmpty;
         accvs[accum] _ 0;
         END;
      END;
   RETURN END.
(compb) PROCEDURE;                                                   3/
   %Like compr except with bregs instead of accs.%
   accum _ $a4next;
   WHILE (accum _ accum-1) >= $rpnum DO
      BEGIN
      acstkp _ acstkp - 2;
      IF [acstkp] # bregs[accum] OR
         [acstkp+1] # bregvs[accum] THEN
         BEGIN
         bregs[accum] _ acmpty;
         bregvs[accum] _ 0;
         END;
      END;
   RETURN END.
DECLARE EXTERNAL regflg;
DECLARE rgstk[24], rgstkp=$rgstk;
(rgstrt) PROCEDURE;                                                  3/
   %Want to evaluate an expression and have result end up in
   accumulator specified by acnum.  To do this set ac to acnum,
```

```
   but must also be able to call acpush and acpop during the
   production of code for the expression and be able to restore
   ac and acnum when finish.  Thus stack current value of ac,
   current accumulator which go to next when call acpush (which
   may not be ac if are already in midst of producing code for
   exp to go in some other special register), and save the value
   of acnum.%
   IF rgstkp+3 > $rgstkp THEN
      BEGIN
      setern(100);
      GOTO qserr;
      END;
   [rgstkp] _ ac; BUMP rgstkp;
   [rgstkp] _ IF ac >= $a1num THEN ac ELSE [rgstkp-3]; BUMP
   rgstkp;
   [rgstkp] _ ac _ acnum; BUMP rgstkp;
   BUMP regflg; RETURN END.
(rgend) PROCEDURE;                                              3A
   %When finish producing code for expression to be loaded into
   special register call rgend to undo the work of rgstrt.%
   acnum _ [rgstkp-1]; rgstkp _ rgstkp - 3; ac _ [rgstkp];
   BUMP DOWN regflg; RETURN END.
(setern) PROC(ern);                                             3A
   a1 _ ern;
   !MOVEM a1,40B;
   RETURN END.
DECLARE pstk[10], pstkp=$pstk;
(pac) PROCEDURE;                                                3A
   IF pstkp >= $pstkp THEN
      BEGIN
      setern(122B);
      GOTO qserr;
      END;
   [pstkp] _ ac; BUMP pstkp; RETURN END.
(ppac) PROCEDURE;                                               3A
   BUMP DOWN pstkp;
   accum _ [pstkp];  RETURN END.
(acpush) PROCEDURE;                                             3A
   %Change ac to specify a new accumulator to be used for
   evaluation of expressions.%
   IF regflg AND ac = [rgstkp-1] THEN
      BEGIN
      BUMP [rgstkp-2];
      ac _ [rgstkp-2];
      END
   ELSE BUMP ac;
   IF ac <= $a4num THEN RETURN;
   setern(103);
   GOTO qserr; END.
(acpop) PROCEDURE;                                              3A
   %Restore ac to value prior to last call to acpush.%
   IF regflg AND ac = [rgstkp-2] THEN
      BEGIN
      BUMP DOWN [rgstkp-2];
      ac _ [rgstkp-1];
      END
```

```
   ELSE ac _ ac - 1;
   RETURN END.
DECLARE saveac[10], svacp=$saveac;
(savreg) PROCEDURE;                                                    3A
   %Save any intermediate results which may be in accumulators
   when make procedure call by pushing them on the stack then
   set ac back to A1.%
   IF $svacp <= svacp THEN
      BEGIN
      setern(102);
      GOTO qserr;
      END;
   [svacp] _ ac; BUMP svacp;
   WHILE (ac _ ac-1) >= $a1num DO BEGIN
      %output PUSH S,AC%
      w _ 261B9 + $sknum * 4B7 + ac;
      <LIBE,produce>() END;
   ac _ 0;
   WHILE (ac_ac+1) <= jsysrg DO BEGIN
      % output PUSH S,ac %
      w _ 261B9 + $sknum*4B7 + ac;
      <LIBE,produce>() END;
   ac _ $a1num; %back to A1% RETURN END.
(resreg) PROCEDURE;                                                    3A
   %Restore any registers which were saved by savreg.%
   BUMP DOWN svacp;
   IF (ac _ [svacp]) NOT= $a1num THEN BEGIN
      %output MOVE AC,A1%
      w _ 200B9 + ac * 4B7 + $a1num; <LIBE,produce>() END;
   IF (accum_jsysrg) THEN
      WHILE accum>0 DO BEGIN
         % output POP S,ac %
         w _ 262B9 + $sknum*4B7 + accum;
         <LIBE,produce>();
         accum _ accum-1 END;
   accum _ $a1num;
   WHILE ac > accum DO BEGIN
      %output POP S,ACCUM%
      w _ 262B9 + $sknum * 4B7 + accum;
      <LIBE,produce>();
      BUMP accum END;
   RETURN END.
(jrstreg) PROCEDURE;                                                   3F
   % like resreg but to be called after producing JSYS %
   BUMP DOWN svacp;
   IF (ac _ [svacp]) NOT= 1 THEN BEGIN
      %output MOVE AC,1%
      w _ 200B9 + ac * 4B7 + 1; <LIBE,produce>() END;
   IF (accum_jsysrg) THEN
      WHILE accum>0 DO BEGIN
         % output POP S,ac %
         w _ 262B9 + $sknum*4B7 + accum;
         <LIBE,produce>();
         accum _ accum-1 END;
   accum _ $a1num;
   WHILE ac > accum DO BEGIN
```

```
      %output POP S,ACCUM%
      w _ 26259 + $sknum * 487 + accum;
      <LIBE,produce>();
      BUMP accum END;
   RETURN END.
DECLARE lpstck[200], lpptr=$lpstck;
(lpush) PROC;                                                    35
   %When enter a loop push gn1, gn2, and room for accs.  The accs
   on the stack (lpstck) will define the contents of the
   accumulators whenever exit the loop.  (The -2 signifies that
   there have been no exits produced yet.)%
   IF lpptr+$regnum+2 > $lpptr THEN
      BEGIN
      setern(102);
      GOTO qserr;
      END;
   [lpptr] _ gn1; BUMP lpptr; [lpptr] _ gn2; BUMP lpptr;
   [lpptr] _ -2; lpptr _ lpptr + $regnum;
   resetall(); RETURN END.
(lpdxit) PROC;                                                   3E
   %Called when finish a DO statement%
   lplev _ 1; lpexit(); RETURN END.
(lpexit) PROC;                                                   3E
   %Accs reflect registers on exit from loop%
   xitype _ 0;
   lpx();
   RETURN END.
DECLARE EXTERNAL lplev;
(lpbxit) PROC;                                                   3E
   %Bregs reflect registers on exit from loop%
   BUMP xitype; lplev _ 1;
   lpx();
   RETURN END.
DECLARE xitype, lpt;
(lpx) PROC;                                                      3E
   %Called whenever produce code for exiting a loop.  Lplev
   determines which stacked set of accumulators is to be compared
   to either accs or bregs depending on which reflects the
   registers for this particular exit.  If -2 is still on the
   stack then this is the first exit, so just copy rather than
   compare.%
   accum _ $rpnum;
   IF (lplev < 1) OR
      (lpt _ lpptr-lplev*($regnum+2)) < $lpstck THEN
         BEGIN
         setern(110);
         GOTO qserr;
         END;
   lpt _ lpt + 2;
   IF [lpt] = -2 THEN %copy%
      IF xitype = 0 THEN
         WHILE accum < $a4next DO
            BEGIN
            [lpt] _ accs[accum];
            BUMP lpt;
            [lpt] _ accvs[accum];
```

```
            BUMP lpt, accum;
            END
         ELSE
            WHILE accum < $a4next DO
               BEGIN
               [lpt] _ bregs[accum];
               BUMP lpt;
               [lpt] _ bregvs[accum];
               BUMP lpt, accum;
               END
   ELSE %compare%
      WHILE accum < $a4next DO
         BEGIN
         IF xitype = 0 AND
            ([lpt] # accs[accum] OR [lpt+1] # accvs[accum]) OR
            xitype # 0 AND
            ([lpt] # bregs[accum] OR [lpt+1] # bregvs[accum])
            THEN
               BEGIN
               [lpt] _ acmpty;
               [lpt+1] _ 0;
               END;
         lpt _ lpt + 2;
         BUMP accum;
         END;
   RETURN END.
(lpop) PROC;                                                    3F
   %When reach end of a loop, set accs to what know the registers
   will hold based on their contents at the various exit points
   within the loop.%
   accum _ $a4next;
   WHILE (accum _ accum-1) >= $rpnum DO
      BEGIN
      BUMP DOWN lpptr;
      accvs[accum] _ [lpptr];
      BUMP DOWN lpptr;
      accs[accum] _ [lpptr];
      END;
   lpptr _ lpptr - 2;  RETURN END.
(lgetg1) PROCEDURE;                                             3F
   lget();
   gn1 _ [lpt];
   RETURN END.
(lgetg2) PROCEDURE;                                             3F
   lget();
   gn2 _ [lpt + 1];
   RETURN END.
(lget) PROC;                                                    3F
   IF (lplev < 1) OR
      (lpt _ lpptr-lplev*($regnum+2)) < $lpstck THEN
         BEGIN
         setern(0);
         GOTO qxerr;
         END;
   RETURN END.
DECLARE cstck[200], cpptr=$cstck, castmp;
```

```
(cstrt) PROCEDURE;                                                    3E
   %Called when enter the exp evaluation for a CASE statement.
   Saves gn1.%
   IF $cpptr <= cpptr THEN
      BEGIN
      setern(100);
      GOTO qserr;
      END;
   [cpptr] _ gn1;
   BUMP cpptr;
   resetall();
   RETURN END.
(cpush) PROC;                                                         3E
   %Called when enter the tests for a CASE statement.  Saves gn1,
   gn2 and reserves room on cstck for accs.  The accs on stack
   will be updated to show what will be in registers when fall
   thru the case.%
   IF cpptr+$regnum+2 > $cpptr THEN
      BEGIN
      setern(100);
      GOTO qserr;
      END;
   [cpptr] _ gn1;
   BUMP cpptr;
   [cpptr] _ gn2;
   BUMP cpptr;
   [cpptr] _ -2;
   cpptr _ cpptr + $regnum;
   resetall();
   RETURN END.
(csx) PROCEDURE; %case exit%                                          3E
   accum _ $rpnum;
   IF [lpt] = -2 THEN %copy%
      WHILE accum < $a4next DO
         BEGIN
         [lpt] _ accs[accum];
         BUMP lpt;
         [lpt] _ accvs[accum];
         BUMP lpt,accum;
         END
   ELSE %compare%
      WHILE accum < $a4next DO
         BEGIN
         IF [lpt] # accs[accum] OR
            [lpt+1] # accvs[accum] THEN
            BEGIN
            [lpt] _ acmpty;
            [lpt+1] _ 0;
            END;
         lpt _ lpt + 2;
         BUMP accum;
         END;
   RETURN END.
(caseck) PROC;                                                        3E
   %Each time produce a branch to the end of the case statement
   (i.e. at end of each case that does not end in an unconditional
```

```
   transfer) update info on cstck.%
   lpt _ cpptr-$regnum;
   csx();
   RETURN END.
(casxit) PROCEDURE;                                                    3E
   %exiting from case.  level given by lplev.  adjust what know
   about accs%
   IF (lplev < 1) OR
      (lpt _ cpptr-lplev*($regnum+3)) < $cstck THEN
         BEGIN
         setern(0);
         GOTO qxerr;
         END;
   lpt _ lpt + 3;
   csx();
   RETURN END.
(cpop) PROC;                                                           3E
   %When reach end of case statement set accs to what know then
   to be in the registers based on the contents at the various
   points where have branched after a case.  If the ENDCASE did
   not end in an unconditional transfer must call caseck again.%
   IF NOT ut THEN caseck();
   accum _ $a4next;
   WHILE (accum _ accum-1) >= $rgnum DO
      BEGIN
      BUMP DOWN cpptr;
      accvs[accum] _ [cpptr];
      BUMP DOWN cpptr;
      accs[accum] _ [cpptr];
      END;
   cpptr _ cpptr - 3;
   RETURN END.
(cgetsc) PROCEDURE;                                                    3F
   cget();
   gn1 _ [lpt];
   RETURN END.
(cgetst) PROCEDURE;                                                    3F
   cget();
   gn1 _ [lpt+1];
   RETURN END.
(cget92) PROCEDURE;                                                    3F
   cget();
   gn2 _ [lpt+2];
   RETURN END.
(cget) PROCEDURE;                                                      3F
   IF (lplev < 1) OR
      (lpt _ cpptr - lplev*($regnum+3)) < $cstck
      THEN
         BEGIN
         setern(0);
         GOTO qxerr;
         END;
   RETURN END.
(noddms)PROC;                                                          3I
   %Set noddt attr in macro definition string [name pointed to by idhash] i
   htax = 0%
```

```
      qserr(2000);
      END.
   (macstr)PROC;                                                    3E
      qserr(2000);
      END.
   FINISH of library
;RUNFIL to load XL10 on a TOPS-20 (l10,l108.run,)
   conn l10
   <arcsubsys>l10ldr
   /s
   /m
   /4000100
   libe
   l108
   libe10
   sys:monsym
   <ESC>
   ddt
   initll<ESC>qops.txt
   <ESC>
   st
   l10syms.txt
   <ESC>
   sav <l10>xl10.sav 400 554
   del ops.rel.*
   del l10syms.rel.*
   ;
;RUNFIL to load L109 on a TOPS-20 (l10,l109.run,)
   conn l10
   <arcsubsys>l10ldr
   /s
   /m
   /4000100
   libe9
   l10
   libe109
   minil10data
   minil10runtime
   sys:monsym
   <ESC>
   ddt
   initll<ESC>qops.txt
   <ESC>
   st
   l10syms.txt
   <ESC>
   sav <l10>l109.exe 400 554
   del ops.rel.*
   del l10syms.rel.*
   ;
;RUNFIL to load L109 on a TENEX (l10,l109.run,)
   conn l10
   <arcsubsys>l10ldr
   /s
   /m
   /4000100
```

```
SLF,   8-Sep-79 13:07          < COMPSRC, L10.NLS;3, >  14

libe9
l10
libe109
minil10data
minil10runtime
<subsys>stenex
<ESC>
dot
initll<ESC>ops.txt

<ESC>
st
l10syms.txt

<ESC>
ssav 400 554 <l10>l109.sav

del ops.rel;*
del l10syms.rel;*
;
```

CML