```
-- Calls.mesa, modified by Sweet, July 5, 1978  9:23 AM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [BYTE, wordlength],
  Code: FROM "code" USING [acstack, actenable, catchcount, cfs, CodePassInconsistancy, codeptr],
  CodeDefs: FROM "codedefs" USING [BDOIndex, BDONull, ChunkBase, CodeCCIndex, LabelCCIndex, LabelCCNull
**, Lexeme, MaxParmsInStack, topostack],
  ComData: FROM "comdata" USING [stopping],
  ControlDefs: FROM "controldefs" USING [Lreg, returnOffset],
  ErrorDefs: FROM "errordefs" USING [error, Warning],
  FOpCodes: FROM "fopcodes" USING [qALLOC, qBLT, qCATCH, qDUP, qEFC, qFREE, qKFCB, qLFC, qLI, qLL, qLP,
** qMRE, qMREL, qMXD, qMXDL, qMXW, qMXWL, qPORTI, qPORTO, qPUSH, qR, qRL, qSFC],
  InlineDefs: FROM "inlinedefs",
  LitDefs: FROM "litdefs" USING [STIndex],
  OpTableDefs: FROM "optabledefs" USING [instlength],
  P5ADefs: FROM "p5adefs" USING [adjustacstack, AllocCodeCCItem, chkrandsonstack, Ciout0, Ciout1, compu
**teframesize, Coutjump, deletetomark, dumpstack, gentemplex, incrstack, insertlabel, labelalloc, loads
**eiaddress, loadtsonaddress, LongTreeAddress, makeTOSaddrBDOItem, makeTOSlex, markstack, operandtype,
**pop, P5Error, ReleaseLock, RequireStack, sCassign, transferconstruct, treeliteralvalue, wordsforsei],
**
  P5BDefs: FROM "p5bdefs" USING [Ccatchphrase, pushconst, pushlex, pushlitval, pushlnestedprocdesc, pus
**hrhs, sCcatchphrase],
  P5StmtExprDefs: FROM "p5stmtexprdefs",
  SDDefs: FROM "sddefs" USING [sCopy, sError, sErrorList, sFork, sJoin, sNew, sRestart, sReturnError, s
**ReturnErrorList, sSignal, sSignalList, sStart, sUnnamedError],
  SymDefs: FROM "symdefs" USING [BitAddress, bodytype, BTIndex, BTNull, CBTIndex, ContextLevel, CSEInde
**x, CTXIndex, ctxtype, HTIndex, ISEIndex, lG, MDIndex, SEIndex, SERecord, setype, TypeClass],
  SymSegDefs: FROM "symsegdefs" USING [FindExtension],
  SymTabDefs: FROM "symtabdefs" USING [UnderType, XferMode],
  TableDefs: FROM "tabledefs" USING [TableBase, TableNotifier],
  TreeDefs: FROM "treedefs" USING [empty, scanlist, TreeIndex, TreeLink, treetype];

DEFINITIONS FROM CodeDefs;

Calls: PROGRAM
    IMPORTS MPtr: ComData, CPtr: Code, ErrorDefs, OpTableDefs, P5ADefs, P5BDefs, SymSegDefs, SymTabDefs
**, TreeDefs
    EXPORTS CodeDefs, P5ADefs, P5StmtExprDefs =
  BEGIN
  OPEN P5ADefs, P5BDefs;

  -- imported definitions

  BYTE: TYPE = AltoDefs.BYTE;
  wordlength: CARDINAL = AltoDefs.wordlength;

  Lreg: INTEGER = ControlDefs.Lreg;
  returnOffset: CARDINAL = ControlDefs.returnOffset;
  sStart: BYTE = SDDefs.sStart;
  sRestart: BYTE = SDDefs.sRestart;
  sFork: BYTE = SDDefs.sFork;
  sJoin: BYTE = SDDefs.sJoin;

  STIndex: TYPE = LitDefs.STIndex;

  BitAddress: TYPE = SymDefs.BitAddress;
  BTIndex: TYPE = SymDefs.BTIndex;
  CBTIndex: TYPE = SymDefs.CBTIndex;
  BTNull: BTIndex = SymDefs.BTNull;
  ContextLevel: TYPE = SymDefs.ContextLevel;
  CSEIndex: TYPE = SymDefs.CSEIndex;
  CTXIndex: TYPE = SymDefs.CTXIndex;
  HTIndex: TYPE = SymDefs.HTIndex;
  ISEIndex: TYPE = SymDefs.ISEIndex;
  MDIndex: TYPE = SymDefs.MDIndex;
  lG: ContextLevel = SymDefs.lG;
  SEIndex: TYPE = SymDefs.SEIndex;
  SERecord: TYPE = SymDefs.SERecord;
  TypeClass: TYPE = SymDefs.TypeClass;

  empty: TreeLink = TreeDefs.empty;
  TreeIndex: TYPE = TreeDefs.TreeIndex;
  TreeLink: TYPE = TreeDefs.TreeLink;


  tb: TableDefs.TableBase;              -- tree base (local copy)
```

```
  seb: TableDefs.TableBase;              -- semantic entry base (local copy)
  ctxb: TableDefs.TableBase;             -- context entry base (local copy)
  bb: TableDefs.TableBase;               -- body entry base (local copy)
  cb: ChunkBase;                 -- code base (local copy)

  CallsNotify: PUBLIC TableDefs.TableNotifier =
    BEGIN  -- called by allocator whenever table area is repacked
    seb ← base[SymDefs.setype];
    ctxb ← base[SymDefs.ctxtype];
    bb ← base[SymDefs.bodytype];
    tb ← base[TreeDefs.treetype];
    cb ← LOOPHOLE[tb];
    RETURN
    END;

  Csyserror: PUBLIC PROCEDURE =
    BEGIN
    dumpstack[]; markstack[];
    Csyscall[SDDefs.sUnnamedError];
    RETURN
    END;


  Cnew: PUBLIC PROCEDURE [node: TreeIndex] RETURNS[Lexeme] =
    BEGIN -- generate code for NEW
    dumpstack[]; markstack[];
    IF (tb+node).attr1 THEN
      BEGIN
      pushrhs[(tb+node).son1];
      Csyscall[SDDefs.sCopy];
      END
    ELSE
      BEGIN
      pushconst[(tb+node).son1];
      Csyscall[SDDefs.sNew];
      END;
    CPtr.acstack ← 1; incrstack[1];
    RETURN[topostack]
    END;


  sCstart: PROCEDURE [node: TreeIndex] RETURNS [nrets: CARDINAL] =
    BEGIN
    OPEN FOpCodes;
    ptsei: CSEIndex ← operandtype[(tb+node).son1];
    dumpstack[]; markstack[];
    [] ← pushparms[(tb+node).son2, ptsei, FALSE];
    pushrhs[(tb+node).son1];
    Csyscall[sStart];
    CPtr.acstack ← 0;
    callcatch[IF (tb+node).nsons = 3 THEN (tb+node).son3 ELSE empty];
    WITH (seb+ptsei) SELECT FROM
      transfer => nrets ← wordsforsei[outrecord];
      ENDCASE;
    RETURN
    END;

  Cstart: PUBLIC PROCEDURE [node: TreeIndex] =
    BEGIN
    [] ← sCstart[node];
    RETURN
    END;

  Cstartexp: PUBLIC PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
    BEGIN
    nrets: CARDINAL ← sCstart[node];

    SELECT nrets FROM
      > MaxParmsInStack =>
        BEGIN
        CPtr.acstack ← 1; incrstack[1];
        RETURN [IndirectReturnRecord[node, nrets]]
        END;
      # 1 =>
        BEGIN
        CPtr.acstack ← nrets; incrstack[nrets];
```

```
      RETURN [makeTOSlex[nrets]];
      END;
    ENDCASE =>
      BEGIN
      CPtr.acstack ← 1; incrstack[1];
      RETURN [topostack];
      END
  END;

Crestart: PUBLIC PROCEDURE [node: TreeIndex] =
  BEGIN
  dumpstack[]; markstack[];
  pushrhs[(tb+node).son1];
  Csyscall[sRestart];
  callcatch[IF (tb+node).nsons = 3 THEN (tb+node).son3 ELSE empty];
  RETURN
  END;


Cstop: PUBLIC PROCEDURE [node: TreeIndex] =
  BEGIN OPEN FOpCodes;
  IF ~MPtr.stopping THEN SIGNAL CPtr.CodePassInconsistancy;
  Ciout1[qLL, returnOffset]; Ciout0[qSFC];
  callcatch[IF (tb+node).nsons = 1 THEN (tb+node).son1 ELSE empty];
  RETURN
  END;


callcatch: PROCEDURE [t: TreeLink] =
  BEGIN
  clabel: LabelCCIndex;

  WITH t SELECT FROM
    subtree =>
      IF t # empty THEN Ccatchphrase[index]
      ELSE
        IF CPtr.actenable # LabelCCNull THEN
          BEGIN
          clabel ← labelalloc[];
          Ciout1[FOpCodes.qCATCH, CPtr.cfs];
          Coutjump[JumpA, clabel];
          Coutjump[Jump, CPtr.actenable];
          insertlabel[clabel];
          END;
    ENDCASE;
  RETURN
  END;


sCcall: PROCEDURE [node: TreeIndex] RETURNS [nrets: CARDINAL] =
  BEGIN -- generates code for procedure call statement
  OPEN FOpCodes;
  ptsei: CSEIndex ← operandtype[(tb+node).son1];
  portcall: BOOLEAN ← SymTabDefs.XferMode[ptsei] = port;
  computedtarget: BOOLEAN;
  inlineTree: TreeLink;
  nparms: CARDINAL;
  savestacksize: INTEGER;
  sei: ISEIndex;
  bti: CBTIndex;
  a: BitAddress;
  inlineCall: BOOLEAN;

  WITH (tb+node).son1 SELECT FROM
    symbol =>
      BEGIN
      sei ← index;
      inlineCall ← (seb+sei).constant AND (seb+sei).extended;
      computedtarget ← (ctxb+(seb+sei).ctxnum).ctxlevel # 1G;
      END;
    ENDCASE =>
      BEGIN
      inlineCall ← FALSE;
      computedtarget ← TRUE;
      END;
  IF ~inlineCall THEN dumpstack[];
```

```
    markstack[];
    nparms ← pushparms[(tb+node).son2, ptsei, FALSE];
    IF inlineCall THEN
      BEGIN
      deletetomark[];
      inlineTree ← SymSegDefs.FindExtension[sei];
      WITH inlineTree SELECT FROM
        subtree => TreeDefs.scanlist[(tb+index).son1, CodeInline];
        ENDCASE => P5ADefs.P5Error[513];
      END
    ELSE IF computedtarget THEN
      IF portcall THEN
        BEGIN
        [] ← loadtsonaddress[(tb+node).son1];
        deletetomark[]; incrstack[1];
        Ciout0[qPORTO]; Ciout0[qPORTI];
        END
      ELSE
        BEGIN
        pushrhs[(tb+node).son1];
        deletetomark[]; incrstack[1];
        Ciout0[qSFC];
        END
    ELSE
      BEGIN
      deletetomark[];
      IF (seb+sei).constant THEN
        BEGIN
        bti ← (seb+sei).idinfo;
        IF bti = BTNull THEN
          BEGIN pushlitval[(seb+sei).idvalue]; Ciout0[qSFC] END
        ELSE
          IF (bb+bti).nesting = Inner THEN
            BEGIN pushlnestedprocdesc[bti]; Ciout0[qSFC]; END
          ELSE Ciout1[qLFC, (bb+bti).entryIndex];
        END
      ELSE IF portcall THEN
        BEGIN [] ← loadseiaddress[sei]; Ciout0[qPORTO]; Ciout0[qPORTI]; END
      ELSE IF (seb+sei).linkSpace THEN
        BEGIN a ← (seb+sei).idvalue; Ciout1[qEFC, a.wd]; END
      ELSE BEGIN pushlex[[se[sei]]]; Ciout0[qSFC] END;
      END;
    WITH (seb+ptsei) SELECT FROM
      transfer => nrets ← wordsforsei[outrecord];
      ENDCASE => P5ADefs.P5Error[514];
    IF inlineCall THEN
      BEGIN
      IF (tb+node).nsons = 3 THEN
        BEGIN
        savestacksize ← CPtr.acstack;
        CPtr.acstack ← 0;
        WITH (tb+node).son3 SELECT FROM
          subtree => Ccatchphrase[index];
          ENDCASE;
        CPtr.acstack ← savestacksize;
        END;
      adjustacstack[nrets-nparms];
      incrstack[nrets];
      END
    ELSE
      BEGIN
      CPtr.acstack ← 0;
      callcatch[IF (tb+node).nsons = 3 THEN (tb+node).son3 ELSE empty];
      incrstack[CPtr.acstack ← IF nrets > MaxParmsInStack THEN 1 ELSE nrets];
      END;
    RETURN
    END;


  sCsigerr: PROCEDURE [node: TreeIndex] RETURNS [nrets: CARDINAL] =
    BEGIN -- generates code for signal/error
    ptsei: CSEIndex ← operandtype[(tb+node).son1];
    nparms: CARDINAL;

    dumpstack[]; markstack[];
    IF (tb+node).son1 = empty THEN pushlitval[-1]
```

```
    ELSE pushrhs[(tb+node).son1];
    nparms ← pushparms[(tb+node).son2, ptsei, TRUE];
    IF (tb+node).name = signal THEN
      Csyscall[IF nparms > 1 THEN SDDefs.sSignalList ELSE SDDefs.sSignal]
    ELSE
      Csyscall[IF nparms > 1 THEN SDDefs.sErrorList ELSE SDDefs.sError];
    WITH (seb+ptsei) SELECT FROM
      transfer => nrets ← wordsforsei[outrecord];
      ENDCASE;
    CPtr.acstack ← 0;
    callcatch[IF (tb+node).nsons = 3 THEN (tb+node).son3 ELSE empty];
    IF (tb+node).name = error THEN Coutjump[JumpRet,LabelCCNull];
    RETURN
    END;

  Cxerror: PUBLIC PROCEDURE [node: TreeIndex] =
    BEGIN -- generates code for RETURN WITH error
    ptsei: CSEIndex ← operandtype[(tb+node).son1];
    nparms: CARDINAL;
    monitored: BOOLEAN ← (tb+node).attr1;

    IF monitored AND (tb+node).attr2 THEN
      BEGIN ReleaseLock[]; monitored ← FALSE END;
    dumpstack[]; markstack[];
    IF (tb+node).son1 = empty THEN pushlitval[-1]
    ELSE pushrhs[(tb+node).son1];
    nparms ← pushparms[(tb+node).son2, ptsei, TRUE];
    IF monitored THEN
      BEGIN
      RequireStack[0];
      ReleaseLock[];
      chkrandsonstack[2];
      END;
    Csyscall[IF nparms > 1 THEN SDDefs.sReturnErrorList
                ELSE SDDefs.sReturnError];
    Coutjump[JumpRet,LabelCCNull];
    RETURN
    END;

  CodeInline: PROCEDURE [t: TreeLink] =
    BEGIN
    opByte: ARRAY [0..3] OF BYTE;
    iLength: CARDINAL ← 0;
    tLength: CARDINAL;
    i: CARDINAL;
    c: CodeCCIndex;

    PickUpByte: PROCEDURE [t: TreeLink] =
      BEGIN
      opByte[iLength] ← WITH t SELECT FROM
        symbol => (seb+index).idvalue,
        ENDCASE => treeliteralvalue[t];
      IF (iLength ← iLength+1) > 3 THEN ErrorDefs.error[instLength];
      END;

    TreeDefs.scanlist[t, PickUpByte];
    IF iLength = 0 THEN RETURN;
    tLength ← OpTableDefs.instlength[opByte[0]];
    IF tLength # 0 AND iLength # tLength THEN
      ErrorDefs.Warning[instLength];
    c ← AllocCodeCCItem[iLength-1];
    cb[c].realinst ← TRUE;
    cb[c].inst ← opByte[0];
    cb[c].isize ← iLength;
    FOR i IN [1..iLength) DO cb[c].parameters[i] ← opByte[i]; ENDLOOP;
    END;


  pushparms: PROCEDURE [t: TreeLink, ptsei: CSEIndex, sigerr: BOOLEAN] RETURNS [nparms: CARDINAL] =
    BEGIN
    rsei: CSEIndex;

    WITH (seb+ptsei) SELECT FROM
      transfer =>
        BEGIN nparms ← wordsforsei[inrecord]; rsei ← SymTabDefs.UnderType[inrecord] END;
      ENDCASE => P5ADefs.P5Error[515];
```

```
    IF nparms > MaxParmsInStack OR (sigerr AND nparms > 1) THEN
      BEGIN
      pushlitval[computeframesize[nparms]];
      Ciout0[FOpCodes.qALLOC];
      transferconstruct[makeTOSaddrBDOItem[wordlength], t, rsei];
      END
    ELSE IF sigerr AND nparms = 0 THEN pushlitval[-1]
    ELSE transferconstruct[BDONull, t, rsei];
    RETURN
    END;

  Ccall: PUBLIC PROCEDURE [node: TreeIndex] =
    BEGIN
    [] ← sCcall[node];
    RETURN
    END;

  Csigerr: PUBLIC PROCEDURE [node: TreeIndex] =
    BEGIN
    [] ← sCsigerr[node];
    RETURN
    END;

  Ccallexp: PUBLIC PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
    BEGIN
    nrets: CARDINAL ← sCcall[node];

    SELECT nrets FROM
      > MaxParmsInStack =>
        BEGIN
        RETURN [IndirectReturnRecord[node, nrets]]
        END;
      # 1 =>
        BEGIN
        RETURN [makeTOSlex[nrets]];
        END;
      ENDCASE =>
        BEGIN
        RETURN [topostack];
        END
    END;

  LogHeapFree: PUBLIC SIGNAL [calltree: TreeLink] RETURNS [BOOLEAN, se Lexeme] = CODE;

  IndirectReturnRecord: PROCEDURE [node: TreeIndex, nrets: CARDINAL] RETURNS[Lexeme] =
    BEGIN
    OPEN FOpCodes;
    tlex, hlex: se Lexeme;
    logged: BOOLEAN;
    r: BDOIndex;

    [logged, hlex] ← SIGNAL LogHeapFree[TreeLink[subtree[node]]];
    IF ~logged THEN
      BEGIN
      tlex ← gentemplex[1];
      sCassign[tlex.lexsei];
      Ciout0[qPUSH];
      hlex ← gentemplex[nrets];
      pushlitval[nrets];
      [] ← loadseiaddress[hlex.lexsei];
      Ciout0[qBLT];
      pushlex[tlex];
      Ciout0[qFREE];
      RETURN [hlex]
      END;
    IF hlex # topostack THEN
      BEGIN sCassign[hlex.lexsei]; Ciout0[qPUSH]; END;
    r ← makeTOSaddrBDOItem[wordlength];
    cb[r].offset.size ← nrets*wordlength;
    RETURN [Lexeme[bdo[r]]]
    END;

  Csigerrexp: PUBLIC PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
    BEGIN
    OPEN FOpCodes;
    nrets: CARDINAL ← sCsigerr[node];
```

```
    CPtr.acstack ← 1; incrstack[1];
    SELECT nrets FROM
      > 1 => RETURN [IndirectReturnRecord[node, nrets]];
      ENDCASE => RETURN [topostack]
    END;

  Csyscall: PUBLIC PROCEDURE [alpha: BYTE] =
    BEGIN -- puts out call via system transfer vector
    deletetomark[];
    Ciout1[FOpCodes.qKFCB, alpha];
    CPtr.acstack ← 0;
    RETURN
    END;

  Csyscalln: PUBLIC PROCEDURE [alpha: BYTE, n: CARDINAL] =
    BEGIN -- puts out call via system transfer vector
    deletetomark[];
    Ciout1[FOpCodes.qKFCB, alpha];
    CPtr.acstack ← n;
    incrstack[n];
    RETURN
    END;

  Cwait: PUBLIC PROCEDURE [node: TreeIndex] =
    BEGIN OPEN FOpCodes;
    retry: LabelCCIndex;
    t1addrsize, t2addrsize: CARDINAL;
    longWait: BOOLEAN;
    t1addrsize ← loadtsonaddress[(tb+node).son1];
    IF t1addrsize = wordlength AND LongTreeAddress[(tb+node).son2] THEN
      Ciout0[qLP];
    t2addrsize ← loadtsonaddress[(tb+node).son2];
    longWait ← t1addrsize > wordlength OR t2addrsize > wordlength;
    IF ~longWait THEN
      BEGIN
      Ciout0[qDUP]; Ciout1[qR,1];  -- load timeout
      Ciout0[qMXW];
      END
    ELSE
      BEGIN
      IF t2addrsize = wordlength THEN Ciout0[qLP];
      [] ← loadtsonaddress[(tb+node).son2];
      IF t2addrsize = wordlength THEN Ciout0[qLP];
      Ciout1[qRL, 1];
      Ciout0[qMXWL];
      END;
    retry ← labelalloc[];
    insertlabel[retry];
    [] ← loadtsonaddress[(tb+node).son1];
    IF longWait AND t1addrsize = wordlength THEN Ciout0[qLP];
    [] ← loadtsonaddress[(tb+node).son2];
    IF longWait AND t2addrsize = wordlength THEN Ciout0[qLP];
    Ciout0[IF longWait THEN qMREL ELSE qMRE];
    CPtr.acstack ← 0; pop[];
    callcatch[IF (tb+node).nsons = 3 THEN (tb+node).son3 ELSE empty];
    CPtr.acstack ← 1; incrstack[1];
    Ciout1[FOpCodes.qLI, 0];
    Coutjump[JumpE, retry];
    END;

  Cforkexp: PUBLIC PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
    BEGIN
    ptsei: CSEIndex ← operandtype[(tb+node).son1];
    dumpstack[]; markstack[];
    [] ← pushparms[(tb+node).son2, ptsei, FALSE];
    pushrhs[(tb+node).son1];
    Csyscall[sFork];
    callcatch[IF (tb+node).nsons = 3 THEN (tb+node).son3 ELSE empty];
    CPtr.acstack ← 1; incrstack[1];
    RETURN [topostack];
    END;

  sCjoin: PUBLIC PROCEDURE [node: TreeIndex] RETURNS [nrets: CARDINAL] =
    BEGIN
    localcatch: BOOLEAN ← (tb+node).nsons = 3;
```

```
    ptsei: CSEIndex ← operandtype[(tb+node).son1];
    aroundlabel, firstcatch: LabelCCIndex;
    savcfs: CARDINAL ← CPtr.cfs;
    r: CodeCCIndex;
    dumpstack[]; markstack[];
    pushrhs[(tb+node).son1];
    Csyscall[sJoin];
    IF localcatch THEN
      BEGIN
      aroundlabel ← labelalloc[];
      firstcatch ← labelalloc[];
      CPtr.catchcount ← CPtr.catchcount + 1;
      Ciout1[FOpCodes.qCATCH, 0];
      r ← LOOPHOLE[CPtr.codeptr, CodeCCIndex];
      Coutjump[JumpA, aroundlabel];
      insertlabel[firstcatch];
      WITH (tb+node).son3 SELECT FROM
        subtree => sCcatchphrase[index];
        ENDCASE => P5ADefs.P5Error[516];
      cb[r].parameters[1] ← CPtr.cfs;
      insertlabel[aroundlabel];
      CPtr.catchcount ← CPtr.catchcount - 1;
      END
    ELSE callcatch[empty];
    CPtr.acstack ← 1; incrstack[1];
    Ciout0[FOpCodes.qSFC];
    IF localcatch THEN
      BEGIN
      aroundlabel ← labelalloc[];
      Ciout1[FOpCodes.qCATCH, CPtr.cfs];
      Coutjump[JumpA, aroundlabel];
      Coutjump[Jump, firstcatch];
      insertlabel[aroundlabel];
      CPtr.cfs ← savcfs;
      END
    ELSE callcatch[empty];
    WITH (seb+ptsei) SELECT FROM
      transfer => nrets ← wordsforsei[outrecord];
      ENDCASE;
    END;

  Cjoinexp: PUBLIC PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
    BEGIN
    nrets: CARDINAL ← sCjoin[node];
    SELECT nrets FROM
      > MaxParmsInStack =>
        BEGIN
        CPtr.acstack ← 1; incrstack[1];
        RETURN [IndirectReturnRecord[node, nrets]]
        END;
      # 1 =>
        BEGIN
        CPtr.acstack ← nrets; incrstack[nrets];
        RETURN [makeTOSlex[nrets]];
        END;
      ENDCASE =>
        BEGIN
        CPtr.acstack ← 1; incrstack[1];
        RETURN [topostack];
        END
    END;

  Cjoin: PUBLIC PROCEDURE [node: TreeIndex] =
    BEGIN
    [] ← sCjoin[node];
    END;

  Cunlock: PUBLIC PROCEDURE [node: TreeIndex] =
    BEGIN
    mlock: TreeLink ← (tb+node).son1;
    lockaddrsize: CARDINAL;
    IF mlock # empty THEN
      BEGIN lockaddrsize ← loadtsonaddress[mlock];
      Ciout0[IF lockaddrsize = wordlength THEN FOpCodes.qMXD
        ELSE FOpCodes.qMXDL];
      END;
```

```
    RETURN
    END;

  END...
```