```
-- Store.mesa, modified by Sweet, August 2, 1978  11:10 AM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [Address, BYTE, charlength, wordlength],
  Code: FROM "code" USING [acstack, CodeNotImplemented, CodePassInconsistancy, curctxlvl, fileindex, xt
**racting, xtractlex, xtractsei],
  CodeDefs: FROM "codedefs" USING [BDOIndex, BDONull, ChunkBase, FullBitAddress, Lexeme, lTOS, MaxParms
**InStack, RegisterName, topostack],
  ComData: FROM "comdata" USING [tC0],
  FOpCodes: FROM "fopcodes" USING [qADD, qBLT, qBLTL, qDUP, qFREE, qGADRB, qLADRB, qLI, qLP, qPOP, qPS,
** qPSD, qPSF, qPUSH, qRD, qRDL, qSL, qW, qWD, qWDL, qWL, qWR, qWS, qWSD, qWSF, qWSTR, qWSTRL],
  InlineDefs: FROM "inlinedefs" USING [BITSHIFT, DIVMOD],
  LitDefs: FROM "litdefs" USING [LTIndex, lttype, MasterString, MSTIndex, sttype],
  P5ADefs: FROM "p5adefs" USING [addfulladdrtobits, bitsforoperand, bitsfortype, Ciout0, Ciout1, Ciout2
**, Cload, copyBDOItem, Cstore, Csyscall, dumpstack, FieldParam, freetempsei, genstringbodylex, gentemp
**lex, loadaddress, loadlexaddress, loadseiaddress, loadtsonaddress, LogHeapFree, makeBDOItem, makeretl
**ex, maketempaddrBDOItem, makeTOSlex, maketreeliteral, maketsonBDOItem, markstack, nextvar, operandtyp
**e, P5Error, prevvar, pushcomponent, releaseBDOItem, RequireStack, rmakeBDOItem, treeliteralvalue, wor
**dsforoperand, wordsforsei],
  P5BDefs: FROM "p5bdefs" USING [Cexp, movetocodeword, MWConstant, pushlex, pushlitval, pushlnonnestedp
**rocdesc, pushlprocdesc, pushrhs, writecodeword],
  P5StmtExprDefs: FROM "p5stmtexprdefs",
  SDDefs: FROM "sddefs" USING [sStringInit],
  SymDefs: FROM "symdefs" USING [BitAddress, bodytype, BTIndex, CBTIndex, ContextLevel, CSEIndex, CTXIn
**dex, ctxtype, HTIndex, ISEIndex, ISENull, lG, lZ, recordCSEIndex, SEIndex, setype, TypeClass],
  SymTabDefs: FROM "symtabdefs" USING [BitsForType, Cardinality, ContextVariant, FnField, NextSe, Recor
**dRoot, TypeLink, TypeRoot, UnderType, WordsForType],
  TableDefs: FROM "tabledefs" USING [TableBase, TableNotifier],
  TreeDefs: FROM "treedefs" USING [empty, listhead, nullTreeIndex, reversescanlist, scanlist, testtree,
** TreeIndex, TreeLink, treetype];

DEFINITIONS FROM CodeDefs;

Store: PROGRAM
    IMPORTS MPtr: ComData, CPtr: Code, LitDefs, P5ADefs, P5BDefs, SymTabDefs, TreeDefs
    EXPORTS CodeDefs, P5ADefs, P5StmtExprDefs
    SHARES LitDefs =
  BEGIN
  OPEN SymTabDefs, P5ADefs, P5BDefs;

  -- imported definitions

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

  BitAddress: TYPE = SymDefs.BitAddress;
  BTIndex: TYPE = SymDefs.BTIndex;
  CBTIndex: TYPE = SymDefs.CBTIndex;
  ContextLevel: TYPE = SymDefs.ContextLevel;
  CSEIndex: TYPE = SymDefs.CSEIndex;
  CTXIndex: TYPE = SymDefs.CTXIndex;
  HTIndex: TYPE = SymDefs.HTIndex;
  ISEIndex: TYPE = SymDefs.ISEIndex;
  ISENull: ISEIndex = SymDefs.ISENull;
  lZ: ContextLevel = SymDefs.lZ;
  lG: ContextLevel = SymDefs.lG;
  recordCSEIndex: TYPE = SymDefs.recordCSEIndex;
  SEIndex: TYPE = SymDefs.SEIndex;
  TypeClass: TYPE = SymDefs.TypeClass;

  MSTIndex: TYPE = LitDefs.MSTIndex;

  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)
  stb: TableDefs.TableBase;                -- string base (local copy)
  ltb: TableDefs.TableBase;                -- literal base (local copy)
```

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

-- state data and code for construction

DUPcount: CARDINAL;
AddressOnStack, BuildingInStack: BOOLEAN;

ConstructionState: TYPE = RECORD[
      savDUPcount: CARDINAL,
      savAddressOnStack, savBuildingInStack: BOOLEAN];

ConstructionError: SIGNAL = CODE;

SaveConstructionState: PROCEDURE [p: POINTER TO ConstructionState] =
  BEGIN
  p↑ ← ConstructionState[DUPcount, AddressOnStack, BuildingInStack];
  RETURN
  END;

RestoreConstructionState: PROCEDURE [p: POINTER TO ConstructionState] =
  BEGIN
  [DUPcount, AddressOnStack, BuildingInStack] ← p↑;
  RETURN
  END;

InitConstructState: PROCEDURE [node: TreeIndex] =
  BEGIN
  DUPcount ← 0; AddressOnStack ← BuildingInStack ← FALSE;
  IF ((tb+node).name = rowcons OR (tb+node).name = rowconsx)
   AND (tb+node).attr1 THEN
    BEGIN DUPcount ← 1; RETURN END;
  TreeDefs.scanlist[(tb+node).son2, countDUPs];
  RETURN
  END;

countDUPs: PROCEDURE [t: TreeLink] =
  BEGIN
  node: TreeIndex;

  IF t = empty THEN RETURN;
  WITH t SELECT FROM
    subtree =>
      BEGIN
      node ← index;
      SELECT (tb+node).name FROM
        rowconsx, constructx =>
          WITH (tb+node).son1 SELECT FROM
            subtree =>
              IF (tb+index).name = temp
               AND ~((tb+node).name = rowconsx AND (tb+node).attr1) THEN
                BEGIN TreeDefs.scanlist[(tb+node).son2, countDUPs]; RETURN END;
            ENDCASE;
        unionx =>
          BEGIN
          IF (tb+node).attr2 THEN DUPcount ← DUPcount+1;
          TreeDefs.scanlist[(tb+node).son2, countDUPs];
          RETURN
          END;
        cast, align => BEGIN TreeDefs.scanlist[(tb+node).son1, countDUPs]; RETURN END;
        ENDCASE;
      END;
    ENDCASE;
  DUPcount ← DUPcount+1;
  RETURN
  END;
```

```
ConstructCountDown: PROCEDURE =
  BEGIN
  IF LOOPHOLE[(DUPcount ← DUPcount-1),INTEGER] < 0 THEN
    SIGNAL ConstructionError;
  RETURN
  END;

--
Crowcons: PUBLIC PROCEDURE [node: TreeIndex] =
  BEGIN -- array initialization
  r: BDOIndex;
  asize: CARDINAL;

  InitConstructState[node];
  IF DUPcount = 0 THEN RETURN;
  r ← maketsonBDOItem[(tb+node).son1].lexbdoi;
  asize ← cb[r].offset.size;
  IF ~easilyaddressed[r] THEN
    BEGIN
    IF loadaddress[copyBDOItem[r]] # wordlength THEN
      SIGNAL CPtr.CodeNotImplemented;
    addrtostack[r];
    AddressOnStack ← TRUE;
    END;
  Carrayinit[r, node, operandtype[(tb+node).son1]];
  IF DUPcount # 0 THEN SIGNAL ConstructionError;
  RETURN
  END;


Crowconsx: PUBLIC PROCEDURE [node: TreeIndex]  RETURNS[bdo Lexeme] =
  BEGIN -- array (expression)initialization
  r: BDOIndex;
  asei: CSEIndex ← (tb+node).info;
  awords: CARDINAL ← WordsForType[asei];
  savCS: ConstructionState;

  SaveConstructionState[@savCS];
  InitConstructState[node];
  r ← maketsonBDOItem[(tb+node).son1].lexbdoi;
  IF ~easilyaddressed[r] THEN
    BEGIN
    IF loadaddress[copyBDOItem[r]] # wordlength THEN
      SIGNAL CPtr.CodeNotImplemented;
    addrtostack[r];
    AddressOnStack ← TRUE;
    END;
  DUPcount ← DUPcount+1;
  IF DUPcount # 1 THEN Carrayinit[copyBDOItem[r], node, asei];
  IF DUPcount # 1 THEN SIGNAL ConstructionError;
  cb[r].offset.size ← awords*wordlength;
  RestoreConstructionState[@savCS];
  RETURN[Lexeme[bdo[r]]]
  END;


Carrayinit: PROCEDURE [r: BDOIndex, node: TreeIndex, asei: CSEIndex] =
  BEGIN
  -- called for ARRAY initialization by rowcons and rowconsx
  w: CARDINAL;
  Crow[r, node, asei
      !PutAddressOnStack => RESUME;
       GetFloorWD =>
        BEGIN
        IF loadaddress[copyBDOItem[r]] # wordlength THEN
          SIGNAL CPtr.CodeNotImplemented;
        w ← cb[r].offset.posn.wd;
        addrtostack[r];
        RESUME[w]
        END];
  RETURN
  END;


Crow: PROCEDURE [r: BDOIndex, node: TreeIndex, asei: CSEIndex] =
```

```
    BEGIN
    -- handles ARRAY initialization
    n: CARDINAL;
    nbits: CARDINAL;
    csei: CSEIndex;
    c: Address;
    filled: BOOLEAN ← FALSE;
    localstrconst, globalstrconst: BOOLEAN;
    constrow: PROCEDURE [t: TreeLink] =
      BEGIN -- outputs a row of constants
      scr: PROCEDURE [t: TreeLink] =
        BEGIN
        msti: MSTIndex;
        WITH e:t SELECT FROM
          literal =>
            WITH e.info SELECT FROM
              string =>
                BEGIN
                msti ← LitDefs.MasterString[index];
                IF (stb+msti).local THEN localstrconst ← TRUE
                ELSE globalstrconst ← TRUE;
                writecodeword[(stb+msti).info];
                END;
            ENDCASE => P5ADefs.P5Error[577];
        ENDCASE => P5ADefs.P5Error[578];
        n ← n+1;
        RETURN
        END;

      n ← 0;
      TreeDefs.scanlist[t, scr];
      RETURN
      END; -- of constrow

    scrow: PROCEDURE [t: TreeLink] =
      BEGIN
      sr: BDOIndex;
      node: TreeIndex;

      IF t # empty THEN IF BuildingInStack THEN ERROR ELSE
      BEGIN
      sr ← copyBDOItem[r];
      WITH t SELECT FROM
        subtree =>
          SELECT (tb+index).name FROM
           align =>
            BEGIN
            t ← (tb+index).son1; -- note the variant may change here
            cb[sr].offset.size ← cb[sr].offset.size -
                        (bitsfortype[csei] - bitsforoperand[t]);
            END;
           cast => t ← (tb+index).son1;
           ENDCASE;
        ENDCASE;
      WITH t SELECT FROM
        subtree =>
          BEGIN
          node ← index;
          SELECT (tb+node).name FROM
            rowconsx =>
              IF TreeDefs.testtree[(tb+node).son1, temp] THEN
                BEGIN
                Crow[sr, node, csei
                        !PutAddressOnStack => partialaddrtostack[sr,WDdecr]];
                cb[r].offset.posn ← addfulladdrtobits[cb[r].offset.posn, nbits];
                RETURN
                END;
            vconstructx, constructx =>
              IF TreeDefs.testtree[(tb+node).son1, temp] THEN
                BEGIN
                IF (tb+node).name = vconstructx THEN P5ADefs.P5Error[579];
                checkalignment[sr, TypeRoot[csei]];
                mainconstruct[sr, (tb+node).son2, operandtype[t], fieldaddress
                        !PutAddressOnStack => partialaddrtostack[sr,WDdecr]];
                cb[r].offset.posn ← addfulladdrtobits[cb[r].offset.posn, nbits];
                RETURN
```

```
            END;
          ENDCASE;
        END;
      ENDCASE;
    ConstructCountDown[];
    IF AddressOnStack THEN CRassign[sr, t, ISENull, DUPcount#0]
    ELSE
      BEGIN
      mwc: BOOLEAN ← FALSE;
      l: Lexeme;
      l ← Cexp[t ! MWConstant => BEGIN mwc ← TRUE; RESUME[[bdo[sr]]] END];
      IF mwc THEN releaseBDOItem[sr]
      ELSE tllCassign[empty, [bdo[sr]], l, FALSE, nbits];
      END;
    END;
    cb[r].offset.posn ← addfulladdrtobits[cb[r].offset.posn, nbits];
    RETURN
    END; -- of scrow

  w: CARDINAL;
  tlex: se Lexeme;

  IF (tb+node).attr1 THEN
    BEGIN
    c ← movetocodeword[];
    localstrconst ← globalstrconst ← FALSE;
    constrow[(tb+node).son2];
    ConstructCountDown[];
    IF AddressOnStack THEN
      BEGIN
      tlex ← gentemplex[1];
      sCassign[tlex.lexsei];
      END;
    dumpstack[]; markstack[];
    pushlitval[c];
    pushlitval[n];
    IF localstrconst AND globalstrconst THEN SIGNAL CPtr.CodeNotImplemented;
    Ciout1[IF localstrconst THEN FOpCodes.qLADRB ELSE FOpCodes.qGADRB,0];
    IF AddressOnStack THEN pushlex[tlex]; -- since r thinks it is already there
    IF loadaddress[r] # wordlength THEN
      SIGNAL CPtr.CodeNotImplemented;
    Csyscall[SDDefs.sStringInit];
    IF AddressOnStack AND DUPcount # 0 THEN pushlex[tlex];
    RETURN
    END;
  WITH (seb+asei) SELECT FROM
    array =>
      BEGIN
      csei ← UnderType[componenttype];
      IF packed AND SymTabDefs.BitsForType[componenttype] <= 8 THEN
        BEGIN
        nbits ← charlength;
        w ← SIGNAL GetFloorWD;
        SIGNAL PutAddressOnStack[w];
        AddressOnStack ← TRUE;
        IF Cardinality[UnderType[indextype]] MOD 2 # 0 THEN
          BEGIN DUPcount ← DUPcount+1; filled ← TRUE END;
        END
      ELSE nbits ← wordsforsei[csei]*wordlength;
      END;
    ENDCASE => P5ADefs.P5Error[580];
  cb[r].offset.size ← nbits;
  TreeDefs.scanlist[(tb+node).son2, scrow];
  IF filled THEN scrow[MPtr.tC0];
  releaseBDOItem[r];
  RETURN
  END;


CRassign: PUBLIC PROCEDURE [r: BDOIndex, t: TreeLink, sei: ISEIndex, usePut: BOOLEAN] =
  BEGIN -- does a reverse-store for contructors
  OPEN FOpCodes;
  tlex: se Lexeme;
  s, v: CARDINAL;
  mwconst: BOOLEAN ← FALSE;
  l: Lexeme;
```

```
    IF cb[r].tag # bo OR cb[r].base.size # wordlength THEN
      SIGNAL CPtr.CodePassInconsistancy;
    v ← cb[r].offset.posn.wd; s ← cb[r].offset.size;
    IF cb[r].base.level # lTOS THEN
      BEGIN
      pushcomponent[basecomponent,r];
      END;
    IF s > wordlength*2 THEN
      BEGIN
      psize: CARDINAL;
      cb[r].offset.posn.wd ← 0;
      [] ← loadaddress[r];
      tlex ← gentemplex[1];
      sCassign[tlex.lexsei];
      RequireStack[0];
      IF t # empty THEN
        BEGIN
        l ← Cexp[t
        !MWConstant =>
          BEGIN
          r ← maketempaddrBDOItem[tlex];
          cb[r].offset.posn.wd ← v;
          mwconst ← TRUE;
          RESUME[ Lexeme[bdo[r]]];
          END];
        IF mwconst THEN GO TO stored;
        psize ← loadlexaddress[l];
        END
      ELSE psize ← loadseiaddress[sei];
      pushlitval[s/wordlength];
      pushlex[tlex];
      IF v # 0 THEN BEGIN pushlitval[v]; Ciout0[qADD] END;
      IF psize > wordlength THEN
        BEGIN Ciout0[qLP]; Ciout0[qBLTL] END
      ELSE Ciout0[qBLT];
      GO TO stored;
      EXITS
        stored => BEGIN IF usePut THEN pushlex[tlex]; RETURN END;
      END;
    IF t # empty THEN pushrhs[t] ELSE pushlex[Lexeme[se[sei]]];
    IF s = wordlength THEN Ciout1[IF usePut THEN qPS ELSE qWS, v] ELSE
    IF s = 2*wordlength THEN Ciout1[IF usePut THEN qPSD ELSE qWSD, v]
    ELSE Ciout2[IF usePut THEN qPSF ELSE qWSF, v, FieldParam[r]];
    releaseBDOItem[r];
    RETURN
    END;


  transferconstruct: PUBLIC PROCEDURE [r: BDOIndex, t: TreeLink, tsei: CSEIndex] =
    BEGIN -- subroutine for parameter/RETURN records built in memory
    savCS: ConstructionState;

    SaveConstructionState[@savCS];
    DUPcount ← 0; BuildingInStack ← r = BDONull;
    TreeDefs.scanlist[t, countDUPs];
    IF DUPcount # 0 THEN
      BEGIN
      AddressOnStack ← ~BuildingInStack;
      IF AddressOnStack THEN DUPcount ← DUPcount+1;
      mainconstruct[r, t, tsei, FnField];
      IF DUPcount # (IF AddressOnStack THEN 1 ELSE 0) THEN SIGNAL ConstructionError;
      END ELSE IF r # BDONull THEN releaseBDOItem[r];
    RestoreConstructionState[@savCS];
    RETURN
    END;


  fieldaddress: PROCEDURE [sei: ISEIndex] RETURNS [BitAddress, CARDINAL] =
    BEGIN
    RETURN [(seb+sei).idvalue, (seb+sei).idinfo]
    END;

  mainconstruct: PROCEDURE [r: BDOIndex, maint: TreeLink, rsei: CSEIndex,
        fa: PROCEDURE[ISEIndex] RETURNS [BitAddress, CARDINAL]] =
    BEGIN -- workhorse subroutine for construction in memory
```

```
    fieldsei: ISEIndex;
    temp: TreeLink;
    more: BOOLEAN ← TRUE;
    savDC, pad: CARDINAL;
    constctx: CTXIndex;
    rcsei: recordCSEIndex;
    ssmc: PROCEDURE [root: TreeLink] =
      BEGIN
      rep: BitAddress;
      res: CARDINAL;
      copyr: BDOIndex ← BDONull;
      node: TreeIndex;
      tsei: ISEIndex;
      tagvalue, fillsize: CARDINAL;
      iscontrolled: BOOLEAN ← FALSE;
      delta: CARDINAL ← 0;
      nw, nb: CARDINAL;
      w: CARDINAL;
      bis: PROCEDURE =
          BEGIN
          lti: LitDefs.LTIndex;
          i: CARDINAL;
          mwconst: BOOLEAN ← FALSE;
          WITH root SELECT FROM
            subtree => SELECT (tb+index).name FROM
              mwconst =>
                BEGIN
                WITH (tb+index).son1 SELECT FROM
                  literal => WITH info SELECT FROM
                    word => lti ← index;
                    ENDCASE => P5ADefs.P5Error[581];
                  ENDCASE => P5ADefs.P5Error[582];
                WITH ll:(ltb+lti) SELECT FROM
                  short => pushlitval[ll.value];
                  long => FOR i IN [0..ll.length) DO
                    pushlitval[ll.value[i]];
                    ENDLOOP;
                  ENDCASE;
                mwconst ← TRUE;
                END;
              ENDCASE;
            ENDCASE;
          IF ~mwconst THEN pushrhs[root];
          IF delta # 0 THEN
            BEGIN
            [nw, nb] ← InlineDefs.DIVMOD[delta, wordlength];
            IF nb # 0 THEN SIGNAL CPtr.CodePassInconsistancy;
            THROUGH [1..nw] DO pushlitval[0] ENDLOOP;
            END;
          RETURN
          END;

      BEGIN
      [rep, res] ← fa[fieldsei];
      WITH (seb+UnderType[(seb+fieldsei).idtype]) SELECT FROM
        union =>
          BEGIN
          iscontrolled ← controlled;
          IF iscontrolled THEN
            BEGIN fieldsei ← tagsei; res ← (seb+fieldsei).idinfo END
          ELSE BEGIN fieldsei ← ISENull; res ← 0 END;
          END;
        ENDCASE;
      IF r # BDONull THEN
        BEGIN
        copyr ← copyBDOItem[r];
        cb[copyr].offset.size ← res;
        cb[copyr].offset.posn ←
            addfulladdrtobits[cb[copyr].offset.posn, rep.wd * wordlength + rep.bd];
        END;
      IF pad # 0 THEN
        BEGIN
        cb[copyr].offset.size ← (res ← res+pad);
        cb[r].offset.posn ← addfulladdrtobits[cb[r].offset.posn, pad];
        pad ← 0;
        END;
```

```
IF root # empty THEN
  BEGIN
  WITH root SELECT FROM
   subtree =>
    SELECT (tb+index).name FROM
     align =>
      BEGIN
      root ← (tb+index).son1; -- note the variant may change here
      delta ← bitsfortype[(seb+fieldsei).idtype] - bitsforoperand[root];
      IF r # BDONull THEN
        res ← (cb[copyr].offset.size ← cb[copyr].offset.size - delta);
      END;
     cast => root ← (tb+index).son1;
     ENDCASE;
   ENDCASE;
  WITH root SELECT FROM
    subtree =>
      BEGIN
      node ← index;
      IF (tb+node).name = constructx
       AND TreeDefs.testtree[(tb+node).son1, temp] THEN
        BEGIN
        checkalignment[copyr, TypeRoot[operandtype[root]]
          !BuildInTemp =>
            BEGIN
            savDC ← DUPcount;
            DUPcount ← 0;
            TreeDefs.scanlist[(tb+node).son2, countDUPs];
            DUPcount ← savDC - DUPcount;
            bis[];
            GOTO done
            END];
        mainconstruct[copyr, (tb+node).son2, operandtype[root], fieldaddress
          !PutAddressOnStack => partialaddrtostack[copyr, WDdecr]];
        GOTO done
        END;
      IF (tb+node).name = unionx THEN
        BEGIN
        WITH (tb+node).son1 SELECT FROM
          symbol => tsei ← index;
          ENDCASE => P5ADefs.P5Error[583];
        tagvalue ← (seb+tsei).idvalue;
        more ← TRUE;
        maint ← (tb+node).son2;
        rcsei ← LOOPHOLE[UnderType[tsei], recordCSEIndex];
        constctx ← (seb+rcsei).fieldctx;
        fieldsei ← nextvar[(ctxb+constctx).selist];
        IF iscontrolled THEN
          BEGIN
          IF fieldsei # ISENull AND (seb+fieldsei).ctxnum # constctx THEN
            BEGIN -- a dummy fill field
            fillsize ← (seb+fieldsei).idinfo;
            tagvalue ← InlineDefs.BITSHIFT[tagvalue, fillsize];
            cb[copyr].offset.size ←
               cb[copyr].offset.size+fillsize;
            fieldsei ← nextvar[NextSe[fieldsei]];
            END;
          ConstructCountDown[];
          IF ~AddressOnStack THEN P5ADefs.P5Error[584];
          CRassign[copyr, maketreeliteral[tagvalue], ISENull, DUPcount#0];
          END;
        RETURN
        END;
      IF (tb+node).name = rowconsx AND ~BuildingInStack
       AND TreeDefs.testtree[(tb+node).son1, temp] THEN
        BEGIN
        Crow[copyr, node, UnderType[(seb+fieldsei).idtype]!
          GetFloorWD =>
            BEGIN
            [] ← loadaddress[copyBDOItem[copyr]];
            w ← cb[copyr].offset.posn.wd;
            addrtostack[copyr];
            RESUME[w]
            END;
          PutAddressOnStack => partialaddrtostack[r,WDdecr]];
        GOTO done
```

```
            END;
          END;
        ENDCASE;
      ConstructCountDown[];
      IF BuildingInStack THEN BEGIN bis[]; GOTO done END;
      IF AddressOnStack THEN CRassign[copyr, root, ISENull, DUPcount#0]
      ELSE
        BEGIN
        mwc: BOOLEAN ← FALSE;
        1: Lexeme;
        1 ← Cexp[root ! MWConstant => BEGIN mwc ← TRUE; RESUME[[bdo[copyr]]] END];
        IF mwc THEN releaseBDOItem[copyr]
        ELSE t1lCassign[empty, [bdo[copyr]], 1, FALSE, res];
        END;
      END
    ELSE
      IF BuildingInStack THEN
        THROUGH [0..(res+wordlength-1)/wordlength) DO pushlitval[0] ENDLOOP;
    EXITS
      done => NULL;
    END;
    fieldsei ← nextvar[NextSe[fieldsei]];
    RETURN
    END;

  WITH (seb+rsei) SELECT FROM
    record =>
      BEGIN
      rcsei ← LOOPHOLE[rsei, recordCSEIndex];
      IF fa # FnField AND (seb+rcsei).length < wordlength THEN
        pad ← cb[r].offset.size - (seb+rcsei).length
      ELSE pad ← 0;
      rcsei ← RecordRoot[rcsei];
      constctx ← (seb+rcsei).fieldctx;
      fieldsei ← nextvar[(ctxb+constctx).selist];
      END;
    union =>
      BEGIN
      pad ← 0;  temp ← TreeDefs.listhead[maint];
      WITH temp SELECT FROM
        subtree =>
          BEGIN
          IF (tb+index).name # unionx THEN P5ADefs.P5Error[585];
          WITH (tb+index).son1 SELECT FROM
            symbol => rsei ← UnderType[TypeLink[index]];
            ENDCASE => P5ADefs.P5Error[586];
          END;
        ENDCASE => P5ADefs.P5Error[587];
      WITH (seb+rsei) SELECT FROM
        record => fieldsei ← ContextVariant[fieldctx];
        ENDCASE => P5ADefs.P5Error[588];
      constctx ← (seb+fieldsei).ctxnum; -- pass the first test
      END;
    ENDCASE => P5ADefs.P5Error[589];
  WHILE more DO
    more ← FALSE;
    IF fieldsei # ISENull AND (seb+fieldsei).ctxnum # constctx THEN
      BEGIN
      DUPcount ← DUPcount+1;
      ssmc[maketreeliteral[0]];
      END;
    TreeDefs.scanlist[maint, ssmc];
    ENDLOOP;
  IF ~BuildingInStack THEN releaseBDOItem[r];
  RETURN
  END;


Cconstructx: PUBLIC PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
  BEGIN
  RETURN [sconstructx[node, FALSE]]
  END;


Cvconstructx: PUBLIC PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
  BEGIN
```

```
    RETURN [sconstructx[node, TRUE]]
    END;


  sconstructx: PROCEDURE [node: TreeIndex, variant: BOOLEAN] RETURNS [l: Lexeme] =
    BEGIN
    -- generate code for constructor expression
    r: BDOIndex;
    tsei: recordCSEIndex;
    wa: BOOLEAN ← FALSE;
    node1: TreeIndex;
    csei: CSEIndex;
    savCS: ConstructionState;
    nbits, w: CARDINAL;
    t1: TreeLink ← (tb+node).son1;

    SaveConstructionState[@savCS];
    InitConstructState[node];
    IF variant THEN
      WITH t1 SELECT FROM
        subtree => t1 ← (tb+index).son1;
        ENDCASE => P5ADefs.P5Error[590];
    tsei ← LOOPHOLE[operandtype[t1], recordCSEIndex];
    IF ~variant THEN
      BEGIN
      wa ← wordaligned[LOOPHOLE[tsei, recordCSEIndex]];
      WITH t1 SELECT FROM
        subtree =>
          BEGIN
          node1 ← index;
          IF (tb+node1).name = temp AND wa AND WordsForType[tsei] <= MaxParmsInStack THEN
            BuildingInStack ← TRUE;
          END;
        ENDCASE;
      END;
    IF variant THEN
      WITH (tb+node).son1 SELECT FROM
        subtree =>
          WITH (tb+index).son2 SELECT FROM
            symbol =>
              IF nextvar[(ctxb+(seb+RecordRoot[tsei]).fieldctx).selist] = index THEN
                BEGIN csei ← tsei; variant ← FALSE END
              ELSE csei ← UnderType[(seb+index).idtype];
            ENDCASE => P5ADefs.P5Error[591];
        ENDCASE
    ELSE csei ← tsei;
    IF ~BuildingInStack THEN
      BEGIN
      r ← maketsonBDOItem[t1].lexbdoi;
      IF ~wa OR ~easilyaddressed[r] THEN
        BEGIN
        IF loadaddress[copyBDOItem[r]] # wordlength THEN
          SIGNAL CPtr.CodeNotImplemented;
        addrtostack[r];
        IF variant AND (nbits ← bitsforoperand[t1]) < wordlength THEN
          BEGIN
          cb[r].offset.posn ←
            addfulladdrtobits[cb[r].offset.posn, cb[r].offset.size - nbits];
          cb[r].offset.size ← nbits;
          END;
        AddressOnStack ← TRUE;
        END;
      END;
    DUPcount ← DUPcount + 1;
    mainconstruct[IF ~BuildingInStack THEN copyBDOItem[r] ELSE BDONull,
      (tb+node).son2, csei, fieldaddress
        !PutAddressOnStack => RESUME;
         GetFloorWD =>
          BEGIN
          [] ← loadaddress[copyBDOItem[r]];
          w ← cb[r].offset.posn.wd;
          addrtostack[r];
          RESUME[w]
          END];
    IF DUPcount # 1 THEN SIGNAL ConstructionError;
    IF BuildingInStack THEN l ← makeTOSlex[WordsForType[tsei]] ELSE l ← Lexeme[bdo[r]];
```

```
    RestoreConstructionState[@savCS];
    RETURN [1]
    END;


  wordaligned: PROCEDURE [tsei: recordCSEIndex] RETURNS [BOOLEAN] =
    BEGIN -- sees if a word-aligned record (never TRUE for a variant record)
    sei: ISEIndex;
    wa: INTEGER ← 0;
    a: BitAddress;

    tsei ← RecordRoot[tsei];
    IF (seb+tsei).variant THEN RETURN[FALSE];
    sei ← nextvar[(ctxb+(seb+tsei).fieldctx).selist];
    UNTIL sei = ISENull DO
      a ← (seb+sei).idvalue;
      IF a.bd # 0 THEN RETURN[FALSE];
      IF a.wd < wa THEN RETURN [FALSE];
      wa ← a.wd;
      sei ← nextvar[NextSe[sei]];
      ENDLOOP;
    RETURN[TRUE]
    END;


  GetFloorWD: SIGNAL RETURNS [CARDINAL] = CODE;
  PutAddressOnStack: SIGNAL [WDdecr: CARDINAL] = CODE;
  BuildInTemp: SIGNAL = CODE;

  checkalignment: PROCEDURE [r: BDOIndex, rsei: CSEIndex] =
    BEGIN
    w: CARDINAL;

    IF AddressOnStack THEN RETURN;
    IF ~wordaligned[LOOPHOLE[UnderType[rsei], recordCSEIndex]] THEN
      BEGIN
      IF BuildingInStack THEN SIGNAL BuildInTemp;
      w ← SIGNAL GetFloorWD;
      SIGNAL PutAddressOnStack[w];
      partialaddrtostack[r, w];
      AddressOnStack ← TRUE;
      END;
    RETURN
    END;

  easilyaddressed: PUBLIC PROCEDURE [r: BDOIndex] RETURNS [BOOLEAN] =
    BEGIN
    l: ContextLevel;
    IF cb[r].tag # o THEN RETURN [FALSE];
    l ← cb[r].offset.level;
    RETURN [l = CPtr.curctxlvl OR l = lG]
    END;

  addrtostack: PROCEDURE [r: BDOIndex] =
    BEGIN
    partialaddrtostack[r, cb[r].offset.posn.wd];
    RETURN
    END;

  partialaddrtostack: PROCEDURE [r: BDOIndex, w: CARDINAL] =
    BEGIN
    psize: CARDINAL = IF cb[r].tag = bo THEN cb[r].base.size
        ELSE wordlength;
    cb[r].base ← [level: lTOS, posn: FullBitAddress[0, 0], size: psize];
    cb[r].tag ← bo;
    cb[r].offset.posn.wd ← cb[r].offset.posn.wd - w;
    cb[r].offset.level ← lZ;
    RETURN
    END;

  Cconstruct: PUBLIC PROCEDURE [node: TreeIndex] =
    BEGIN
    sconstruct[node, FALSE];
    RETURN
    END;
```

```
Cvconstruct: PUBLIC PROCEDURE [node: TreeIndex] =
  BEGIN
  sconstruct[node, TRUE];
  RETURN
  END;


sconstruct: PROCEDURE [node: TreeIndex, variant: BOOLEAN] =
  BEGIN -- generate code for construct statement
  tsei: recordCSEIndex;
  wa: BOOLEAN ← FALSE;
  r: BDOIndex;
  csei: CSEIndex;
  nbits, w: CARDINAL;
  t1: TreeLink ← (tb+node).son1;

  InitConstructState[node];
  IF DUPcount = 0 THEN RETURN;
  IF variant THEN
    WITH t1 SELECT FROM
      subtree => t1 ← (tb+index).son1;
      ENDCASE => P5ADefs.P5Error[592];
  tsei ← LOOPHOLE[operandtype[t1], recordCSEIndex];
  IF ~variant THEN wa ← wordaligned[LOOPHOLE[tsei, recordCSEIndex]];
  IF variant THEN
    WITH (tb+node).son1 SELECT FROM
      subtree =>
        WITH (tb+index).son2 SELECT FROM
          symbol =>
            IF nextvar[(ctxb+(seb+RecordRoot[tsei]).fieldctx).selist] = index THEN
              BEGIN csei ← tsei; variant ← FALSE END
            ELSE csei ← UnderType[(seb+index).idtype];
          ENDCASE => P5ADefs.P5Error[593];
      ENDCASE
  ELSE csei ← tsei;
  r ← maketsonBDOItem[t1].lexbdoi;
  IF ~wa OR cb[r].tag # o THEN
    BEGIN
    IF loadaddress[copyBDOItem[r]] # wordlength THEN
      SIGNAL CPtr.CodeNotImplemented;
    addrtostack[r];
    IF variant AND (nbits ← bitsforoperand[t1]) < wordlength THEN
      BEGIN
      cb[r].offset.posn ←
        addfulladdrtobits[cb[r].offset.posn, cb[r].offset.size - nbits];
      cb[r].offset.size ← nbits;
      END;
    AddressOnStack ← TRUE;
    END;
  mainconstruct[r, (tb+node).son2, csei, fieldaddress
      !PutAddressOnStack => RESUME;
       GetFloorWD =>
        BEGIN
        [] ← loadaddress[copyBDOItem[r]];
        w ← cb[r].offset.posn.wd;
        addrtostack[r];
        RESUME[w]
        END];
  IF DUPcount # 0 THEN SIGNAL ConstructionError;
  RETURN
  END;


Cextract: PUBLIC PROCEDURE [node: TreeIndex] =
  BEGIN
  tsei: recordCSEIndex ← LOOPHOLE[operandtype[(tb+node).son1], recordCSEIndex];
  fa: PROCEDURE[ISEIndex] RETURNS [BitAddress, CARDINAL] ←
      IF (seb+tsei).argument THEN FnField ELSE fieldaddress;
  startsei: ISEIndex ← (ctxb+(seb+tsei).fieldctx).selist;
  sei: ISEIndex ← startsei;
  isei: ISEIndex ← startsei;
  soncount: CARDINAL ← 0;
  r: BDOIndex;
  transferrec, instk, ExAddressOnStack, addrintemp, wa: BOOLEAN ← FALSE;
  tlex: se Lexeme;
```

```
    psize: CARDINAL;

    xcount: PROCEDURE [t: TreeLink] =
      BEGIN
      IF t # empty THEN
        BEGIN
        soncount ← soncount+1;
        WITH t SELECT FROM
          subtree => IF wordsforoperand[(tb+index).son1] > 2 THEN
            addrintemp ← TRUE;
          ENDCASE;
        END;
      RETURN
      END;
    sextract: PROCEDURE [t: TreeLink] =
      BEGIN
      rep: BitAddress;
      res: CARDINAL;
      rr: BDOIndex;

      [rep, res] ← fa[sei];
      IF t # empty THEN
        BEGIN
        soncount ← soncount-1;
        IF addrintemp THEN pushlex[tlex]
        ELSE IF transferrec OR (ExAddressOnStack AND soncount > 0) THEN
          Ciout0[FOpCodes.qDUP];
        CPtr.xtractlex ← Lexeme[bdo[rr ← copyBDOItem[r]]];
        CPtr.xtractsei ← sei;
        IF instk THEN cb[rr].offset.posn ← FullBitAddress[0,0]
        ELSE cb[rr].offset.posn ← addfulladdrtobits[cb[rr].offset.posn, rep.wd*wordlength+rep.bd];
        cb[rr].offset.size ← res;
        WITH t SELECT FROM
          subtree => Cassign[index];
          ENDCASE => P5ADefs.P5Error[594];
        END
      ELSE IF instk THEN THROUGH [1..res/wordlength] DO Ciout0[FOpCodes.qPOP] ENDLOOP;
      sei ← prevvar[startsei, sei];
      RETURN
      END; -- of sextract

    IF (seb+tsei).argument THEN fa ← FnField
    ELSE fa ← fieldaddress;

    UNTIL (isei ← NextSe[sei]) = ISENull DO
      isei ← nextvar[isei];
      IF isei = ISENull THEN EXIT;
      sei ← isei;
      ENDLOOP;
    r ← maketsonBDOItem[(tb+node).son2
        !LogHeapFree =>
          IF calltree = (tb+node).son2 THEN
            BEGIN transferrec ← TRUE; RESUME[TRUE, topostack] END;
         MWConstant => SIGNAL CPtr.CodeNotImplemented].lexbdoi;
    instk ← cb[r].tag = o AND cb[r].offset.level = lTOS;
    IF fa # FnField AND cb[r].offset.size > bitsfortype[tsei] THEN
      BEGIN -- padding in record, value shifted to the right
      cb[r].offset.posn ← addfulladdrtobits[cb[r].offset.posn,
        cb[r].offset.size - bitsfortype[tsei]];
      END;
    wa ← wordaligned[LOOPHOLE[operandtype[(tb+node).son2], recordCSEIndex]];
    TreeDefs.scanlist[(tb+node).son1, xcount];
    IF addrintemp AND instk AND wa THEN addrintemp ← FALSE;
    IF addrintemp OR
      (fa # FnField AND (~wa OR (~instk AND ~easilyaddressed[r]))) THEN
      BEGIN
      psize ← loadaddress[copyBDOItem[r]];
      IF psize > wordlength THEN addrintemp ← TRUE;
      cb[r].base.size ← psize;
      addrtostack[r];
      IF addrintemp THEN
        BEGIN
        tlex ← gentemplex[psize/wordlength];
        sCassign[tlex.lexsei];
        END
      ELSE ExAddressOnStack ← TRUE;
```

```
    instk ← FALSE;
    END;
  IF soncount = 0 THEN
    BEGIN
    releaseBDOItem[r];
    IF transferrec THEN
      BEGIN
      IF addrintemp THEN pushlex[tlex];
      Ciout0[FOpCodes.qFREE];
      END;
    UNTIL CPtr.acstack = 0 DO Ciout0[FOpCodes.qPOP] ENDLOOP;
    RETURN
    END;
  CPtr.xtracting ← TRUE;
  TreeDefs.reversescanlist[(tb+node).son1, sextract];
  releaseBDOItem[r];
  IF transferrec THEN
    BEGIN
    IF addrintemp THEN pushlex[tlex];
    Ciout0[FOpCodes.qFREE];
    END;
  UNTIL CPtr.acstack = 0 DO Ciout0[FOpCodes.qPOP] ENDLOOP;
  CPtr.xtracting ← FALSE;
  RETURN
  END;


sCassign: PUBLIC PROCEDURE [sei: ISEIndex] =
  BEGIN -- assigns to a simple variable from the stack
  Cstore[rmakeBDOItem[Lexeme[se[sei]]]];
  RETURN
  END;


Cassign: PUBLIC PROCEDURE [node: TreeIndex] =
  BEGIN -- generates code for assignment statement
  IF (tb+node).attr1 THEN
    BEGIN SIGNAL CPtr.CodeNotImplemented; RETURN END;
  comassign[(tb+node).son1, (tb+node).son2, FALSE];
  RETURN
  END;


ValsOnStack: SIGNAL = CODE;
AddrOnStack: SIGNAL [addrsize: CARDINAL] = CODE;

Cassignx: PUBLIC PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
  BEGIN -- generates code for assignment expression
  OPEN CPtr;
  nwords: CARDINAL;
  push: BOOLEAN ← TRUE;
  valshere: BOOLEAN ← FALSE;
  psize: CARDINAL ← wordlength;

  IF (tb+node).attr1 THEN ERROR CPtr.CodeNotImplemented;
  nwords ← wordsforoperand[(tb+node).son1];
  comassign[(tb+node).son1, (tb+node).son2, TRUE
              !AddrOnStack =>
                BEGIN push ← FALSE; psize ← addrsize; RESUME END;
              ValsOnStack => BEGIN valshere ← TRUE; RESUME END];
  IF ~valshere THEN
    BEGIN
    IF push THEN Ciout0[FOpCodes.qPUSH];
    IF nwords = 2 THEN Ciout0[FOpCodes.qPUSH];
    END;
  RETURN[makeretlex[nwords, psize]];
  END;



comassign: PROCEDURE [t1,t2: TreeLink, isexp: BOOLEAN] =
  BEGIN
  nbits: CARDINAL;
  node: TreeIndex;
  leftr, rightr: BDOIndex;
  rightl: Lexeme;
```

```
    leftl: bdo Lexeme;
    psize: CARDINAL;
    mwconst: BOOLEAN ← FALSE;

    diddleleft: PROCEDURE =
      BEGIN
      leftl ← makeBDOItem[Cexp[(tb+node).son1]];
      leftr ← leftl.lexbdoi;
      cb[leftr].offset.size ←
        cb[leftr].offset.size - (bitsforoperand[(tb+node).son1] - nbits);
      END;

    WITH t1 SELECT FROM
      subtree =>
        BEGIN
        node ← index;
        IF (tb+node).name = align THEN
          BEGIN
          nbits ← bitsforoperand[t2];
          IF nbits <= wordlength*2 THEN
            BEGIN
            pushrhs[t2];
            rightr ← rmakeBDOItem[makeTOSlex[(nbits+wordlength-1)/2]];
            END
          ELSE
            BEGIN
            rightl ← Cexp[t2
              !MWConstant =>
                BEGIN
                mwconst ← TRUE;
                diddleleft[];
                RESUME[leftl];
                END];
            IF mwconst THEN
              BEGIN
              IF isexp THEN
                BEGIN psize ← loadlexaddress[leftl];
                SIGNAL AddrOnStack[psize];
                END
              ELSE ReleaseLex[leftl];
              RETURN;
              END;
            rightr ← rmakeBDOItem[rightl];
            psize ← loadaddress[copyBDOItem[rightr]];
            cb[rightr].base.size ← psize;
            addrtostack[rightr];
            END;
          diddleleft[];
          tllCassign[empty, leftl, [bdo[rightr]], isexp, nbits];
          RETURN
          END;
        END;
      ENDCASE;
    nbits ← bitsforoperand[t1];
    IF t2 # empty THEN WITH t2 SELECT FROM
      subtree =>
        BEGIN
        node ← index;
        SELECT (tb+node).name FROM
          cast => t2 ← (tb+node).son1;
          ENDCASE;
        END;
      ENDCASE;
    IF t2 # empty THEN WITH t2 SELECT FROM
      subtree =>
        BEGIN
        node ← index;
        SELECT (tb+node).name FROM
          align =>
            BEGIN
            rightr ← rmakeBDOItem[Cexp[(tb+node).son1
              !MWConstant =>
                RESUME[gentemplex[ (nbits+wordlength-1)/ wordlength]]]];
            psize ← loadaddress[copyBDOItem[rightr]];
            cb[rightr].base.size ← psize;
            addrtostack[rightr];
```

```
          cb[rightr].offset.size ←
            cb[rightr].offset.size - (bitsforoperand[(tb+node).son1] - nbits);
          t1lCassign[t1, topostack, [bdo[rightr]], isexp, nbits];
          RETURN
          END;
        mwconst =>
          BEGIN
          right1 ← Cexp[t2 !MWConstant => GO TO mwconst];
          t1lCassign[t1, topostack, right1, isexp, nbits];
          RETURN;
          EXITS mwconst =>
            BEGIN CassignMwconst[t1, t2, isexp]; RETURN END;
          END;
        ENDCASE;
      END;
    ENDCASE;
  t1lCassign[t1, topostack, Cexp[t2], isexp, nbits];
  RETURN
  END;

ComplicatedAddr: PROCEDURE [node: TreeIndex] RETURNS [BOOLEAN] =
  BEGIN -- TRUE if arithmetic needed to generate address of t
  -- FALSE e.g. if t = p↑
  IF (tb+node).name # uparrow THEN RETURN[TRUE];
  WITH (tb+node).son1 SELECT FROM
    symbol => RETURN[FALSE];
    ENDCASE => RETURN[TRUE];
  END;

CassignMwconst: PROCEDURE [t1, t2: TreeLink, isexp: BOOLEAN] =
  BEGIN OPEN FOpCodes; -- t2 is mwconst of length > 2
  psize: CARDINAL;
  destaddrlex: se Lexeme;
  destlex: Lexeme;
  BEGIN
  WITH t1 SELECT FROM
    subtree =>
      IF ComplicatedAddr[index] THEN
        BEGIN
        psize ← loadtsonaddress[t1];
        destaddrlex ← gentemplex[psize/wordlength];
        sCassign[destaddrlex.lexsei];
        destlex ← [bdo[maketempaddrBDOItem[destaddrlex]]];
        GO TO complicated;
        END;
    ENDCASE;
  destlex ← Cexp[t1];
  EXITS complicated => NULL;
  END;
  destlex ← Cexp[t2 !MWConstant => RESUME[destlex]];
  IF isexp THEN
    BEGIN
    psize ← loadlexaddress[destlex];
    SIGNAL AddrOnStack[psize];
    END
  ELSE ReleaseLex[destlex];
  END;

ReleaseLex: PROCEDURE[l: Lexeme] =
  BEGIN
  WITH l SELECT FROM
    bdo => releaseBDOItem[lexbdoi];
    ENDCASE;
  RETURN
  END;

Cregstore: PROCEDURE[v: RegisterName] =
  BEGIN
  SELECT v FROM
    < 100B =>
      BEGIN Ciout1[FOpCodes.qWR, v] END;
    ENDCASE => SIGNAL CPtr.CodeNotImplemented;
  RETURN
  END;
```

```
tllCassign: PROCEDURE [leftson: TreeLink, leftlex, l: Lexeme, exp: BOOLEAN, nbits: CARDINAL] =
  BEGIN -- main subroutine for doing assignment statements and expressions
  OPEN FOpCodes;
  tlex: se Lexeme;
  r: BDOIndex;
  nwords: CARDINAL ← (nbits+wordlength-1)/wordlength;
  v, ss, psize, dsize: CARDINAL;

  IF nwords = 1 THEN
    BEGIN
    pushlex[l];
    IF leftson # empty THEN leftlex ← Cexp[leftson];
    WITH e: leftlex SELECT FROM
      other => WITH e SELECT FROM
        byte =>
          BEGIN
          Ciout1[(IF long THEN qWSTRL ELSE qWSTR), lexalpha];
          RETURN
          END;
        register => Cregstore[lexrn];
        ENDCASE;
      ENDCASE => Cstore[rmakeBDOItem[leftlex]];
    RETURN
    END;
  WITH l SELECT FROM
    bdo =>
      BEGIN
      r ← lexbdoi;
      IF cb[r].tag = o AND cb[r].offset.level = lTOS THEN
        BEGIN
        ss ← cb[r].offset.posn.wd;
        cb[r].offset.posn.wd ← 0;
        Cload[r];
        r ← IF leftson = empty THEN rmakeBDOItem[leftlex]
            ELSE maketsonBDOItem[leftson].lexbdoi;
        IF ~exp OR (nwords <= 2 AND ss=0) THEN
          BEGIN
          Cstore[r];
          THROUGH [1..ss] DO Ciout0[qPOP] ENDLOOP;
          RETURN
          END;
        psize ← loadaddress[r];
        tlex ← gentemplex[psize/wordlength];
        sCassign[tlex.lexsei];
        v ← nwords;
        THROUGH [1..v/2] DO
          pushlex[tlex]; v ← v-2;
          Ciout1[IF psize > wordlength THEN qWDL ELSE qWD, v];
          ENDLOOP;
        IF v # 0 THEN
          BEGIN
          pushlex[tlex];
          Ciout1[IF psize > wordlength THEN qWL ELSE qW, v-1];
          END;
        THROUGH [1..ss] DO Ciout0[qPOP] ENDLOOP;
        IF exp THEN
          IF nwords > 2 THEN
              BEGIN
              pushlex[tlex];
              SIGNAL AddrOnStack[psize];
              END
          ELSE
              BEGIN
              pushlex[tlex];
              Ciout1[IF psize > wordlength THEN qRDL ELSE qRD, 0];
              SIGNAL ValsOnStack;
              END;
        RETURN
        END
      END;
    ENDCASE;
  IF nwords > 2 THEN
    BEGIN
    psize ← loadaddress[rmakeBDOItem[l]];
    RequireStack[psize/wordlength];
    IF exp THEN
```

```
      BEGIN
      tlex ← gentemplex[psize/wordlength];
      sCassign[tlex.lexsei];
      Ciout0[qPUSH];
      IF psize > wordlength THEN Ciout0[qPUSH];
      END;
    IF psize = wordlength AND
      (IF leftson = empty THEN LongLexAddress[leftlex]
       ELSE LongTreeAddress[leftson]) THEN Ciout0[qLP];
    pushlitval[nwords];
    dsize ← loadlexaddress[IF leftson = empty THEN leftlex ELSE Cexp[leftson]];
    IF psize > dsize THEN BEGIN Ciout0[qLP]; dsize ← psize END;
    Ciout0[IF dsize > wordlength THEN qBLTL ELSE qBLT];
    IF exp THEN BEGIN pushlex[tlex]; SIGNAL AddrOnStack[psize]; END;
    RETURN
    END;
  Cload[rmakeBDOItem[l]];
  Cstore[IF leftson = empty THEN rmakeBDOItem[leftlex] ELSE maketsonBDOItem[leftson].lexbdoi];
  RETURN
  END;


LongTreeAddress: PUBLIC PROCEDURE [t: TreeLink] RETURNS [long: BOOLEAN] =
  BEGIN
  node: TreeIndex;
  WITH t SELECT FROM
    subtree =>
      BEGIN  node ← index;
      IF node = TreeDefs.nullTreeIndex
        THEN  long ← FALSE
        ELSE  SELECT (tb+node).name FROM
          loophole, cast, openexp, align =>
            long ← LongTreeAddress[(tb+node).son1];
          dot, uparrow, dindex, seqindex, dollar, index, reloc =>
            long ← (tb+node).attr1;
          assignx => WITH (tb+node).son2 SELECT FROM
            subtree => IF (tb+index).name = mwconst THEN
                long ← LongTreeAddress[(tb+node).son1]
              ELSE long ← LongTreeAddress[(tb+node).son2];
            ENDCASE => long ← LongTreeAddress[(tb+node).son2];
          ENDCASE => long ← FALSE;
      END;
    ENDCASE => long ← FALSE;
  RETURN
  END;


LongLexAddress: PUBLIC PROCEDURE [l: Lexeme] RETURNS [long: BOOLEAN] =
  BEGIN
  WITH l SELECT FROM
    bdo =>
      SELECT cb[lexbdoi].tag FROM
        o => long ← FALSE;
        bo => long ← cb[lexbdoi].base.size > wordlength;
        ENDCASE =>
          long ← cb[lexbdoi].base.size > wordlength OR
          cb[lexbdoi].disp.size > wordlength;
    ENDCASE =>  long ← FALSE;
  RETURN
  END;


slCassign: PUBLIC PROCEDURE [sei: ISEIndex, l: Lexeme, exp: BOOLEAN, nwords: CARDINAL] =
  BEGIN -- sei-lexeme interface to tllCassign
  tllCassign[empty, Lexeme[se[sei]], l, exp, nwords*wordlength];
  RETURN
  END;


Cportinit: PUBLIC PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
  BEGIN
  Ciout1[FOpCodes.qLI, 0];
  Ciout1[FOpCodes.qLI, 0];
  RETURN[makeTOSlex[2]]
  END;
```

```
Cbodyinit: PUBLIC PROCEDURE [node: TreeIndex] RETURNS [se Lexeme] =
  BEGIN -- assigns proc. desc for proc. variable
  bti: CBTIndex ← (tb+node).info;

  WITH (bb+bti).info SELECT FROM
    Internal => CPtr.fileindex ← sourceIndex;
    ENDCASE;
  pushlprocdesc[bti];
  RETURN [topostack]
  END;


Cstringinit: PUBLIC PROCEDURE [node: TreeIndex] RETURNS [se Lexeme] =
  BEGIN -- inits string storage and pushes pointer on stack
  nchars: CARDINAL;
  l: se Lexeme;

  nchars ← treeliteralvalue[(tb+node).son2];
  l ← genstringbodylex[nchars];
  [] ← loadlexaddress[l];
  freetempsei[l.lexsei];
  pushlitval[0];
  pushlitval[nchars];
  Ciout1[FOpCodes.qPSD, 0];
  RETURN [topostack]
  END;


Cprocinit: PUBLIC PROCEDURE [node: TreeIndex] =
  BEGIN
  bti: CBTIndex ← (tb+node).info;

  WITH (bb+bti) SELECT FROM
    Inner =>
      BEGIN
      pushlnonnestedprocdesc[entryIndex];
      Ciout1[FOpCodes.qSL, frameOffset];
      END;
    ENDCASE;
  RETURN
  END;

END...
```