```
-- Expression.mesa, modified by Sweet, Aug 29, 1978 2:05 PM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [BYTE, BytesPerWord, wordlength],
  Code: FROM "code" USING [acstack, catchcount, CodeNotImplemented, curctxlvl, firstcaseselread, xtract
**ing, xtractlex],
  CodeDefs: FROM "codedefs" USING [BDOComponent, BDOIndex, ChunkBase, FullBitAddress, Lexeme, lTOS, Reg
**isterName, topostack, TosBDOComponent, WordZeroBDOComponent],
  ComData: FROM "comdata",
  ControlDefs: FROM "controldefs" USING [ControlLink, EPRange, GFTNull, Greg, Lreg, ProcDesc, SignalDes
**c],
  FOpCodes: FROM "fopcodes" USING [qADD, qAND, qBLTC, qBLTCL, qDADD, qDESCB, qDESCBS, qDIV, qDSUB, qEXC
**H, qGADRB, qLADRB, qLI, qLLK, qMUL, qNEG, qPOP, qPUSH, qRR, qRSTR, qRSTRL, qSDIV, qSHIFT, qSUB],
  InlineDefs: FROM "inlinedefs" USING [BITAND, BITSHIFT, DIVMOD],
  LitDefs: FROM "litdefs" USING [LTIndex, lttype, MasterString, MSTIndex, sttype],
  P5ADefs: FROM "p5adefs" USING [addfulladdrtobits, Ciout0, Ciout1, Cload, copyBDOItem, Csyscall, Csysc
**alln, genBDOItem, gentemplex, incrstack, loadaddress, loadlexaddress, loadtsonaddress, makeBDOItem, m
**akeretlex, maketempaddrBDOItem, makeTOSaddrBDOItem, makeTOSlex, maketsonBDOItem, markstack, operandty
**pe, P5Error, releaseBDOItem, RequireStack, rmakeBDOItem, treeliteral, treeliteralvalue, wordsforsei],
**
  P5BDefs: FROM "p5bdefs" USING [Ccasestmtexp, Cflowexp, movetocodeword, writecodeword],
  P5StmtExprDefs: FROM "p5stmtexprdefs" USING [Cassignx, Cbodyinit, Ccallexp, Cconstructx, Cdindex, Cfo
**rkexp, Cindex, Cjoinexp, Cnew, Cportinit, Crowconsx, Csigerrexp, Cstartexp, Cstringinit, Cvconstructx
**],
  SDDefs: FROM "sddefs" USING [sFADD, sFDIV, sFLOAT, sFMUL, sFSUB, sLongDiv, sLongMod, sLongMul],
  StringDefs: FROM "stringdefs" USING [StringHeaderSize],
  SymDefs: FROM "symdefs" USING [BitAddress, bodytype, BTIndex, BTNull, CBTIndex, ContextLevel, CSEInde
**x, CTXIndex, ctxtype, HTIndex, ISEIndex, lZ, SEIndex, setype],
  SymTabDefs: FROM "symtabdefs" USING [FnField, NormalType, UnderType, WordsForType, XferMode],
  TableDefs: FROM "tabledefs" USING [TableBase, TableNotifier],
  TreeDefs: FROM "treedefs" USING [empty, testtree, TreeIndex, TreeLink, treetype];

DEFINITIONS FROM FOpCodes, CodeDefs;

Expression: PROGRAM
    IMPORTS CPtr: Code, LitDefs, P5ADefs, P5BDefs, P5StmtExprDefs, SymTabDefs, TreeDefs
    EXPORTS CodeDefs, P5BDefs
    SHARES LitDefs, StringDefs =
  BEGIN
  OPEN P5ADefs, P5StmtExprDefs, P5BDefs;

  -- imported definitions

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

  StringHeaderSize: CARDINAL = StringDefs.StringHeaderSize;

  MSTIndex: TYPE = LitDefs.MSTIndex;

  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;
  lZ: ContextLevel = SymDefs.lZ;
  SEIndex: TYPE = SymDefs.SEIndex;
  LTIndex: TYPE = LitDefs.LTIndex;

  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)
```

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

Cexp: PUBLIC PROCEDURE [t: TreeLink] RETURNS [l: Lexeme] =
  BEGIN -- generates code for an expression
  sei: ISEIndex;
  node: TreeIndex;
  a: BitAddress;
  bti: CBTIndex;
  psize: CARDINAL;

  WITH e: t SELECT FROM
    literal =>
      WITH e.info SELECT FROM
        word => RETURN[Lexeme[literal[word[index]]]];
        string => RETURN[Lexeme[literal[string[index]]]];
        ENDCASE;
    symbol =>
      BEGIN
      sei ← e.index;
      IF (seb+sei).linkSpace THEN
        BEGIN
        a ← (seb+sei).idvalue;
        Ciout1[FOpCodes.qLLK, a.wd];
        RETURN[topostack];
        END;
      IF (seb+sei).constant AND SymTabDefs.XferMode[(seb+sei).idtype] = procedure THEN
        BEGIN
        IF (seb+sei).extended THEN SIGNAL CPtr.CodeNotImplemented;
        bti ← (seb+sei).idinfo;
        IF bti = BTNull THEN pushlitval[(seb+sei).idvalue]
        ELSE pushlprocdesc[bti];
        RETURN[topostack];
        END;
      RETURN[Lexeme[se[sei]]];
      END;
    subtree =>
      BEGIN
      IF e = empty AND CPtr.xtracting THEN RETURN[CPtr.xtractlex];
      node ← e.index;
      SELECT (tb+node).name FROM
        caseexp =>
          BEGIN
          psize ← Ccasestmtexp[node, TRUE];
          l ← makeretlex[SymTabDefs.WordsForType[(tb+node).info], psize];
          END;
        assignx => l ← Cassignx[node];
        plus => l ← Cplus[node];
        minus => l ← Cminus[node];
        div => l ← Cdiv[node];
        mod => l ← Cmod[node];
        times => l ← Ctimes[node];
        dot, uparrow => l ← Cdotoruparrow[node];
        reloc => l ← Creloc[node, FALSE];
        dollar => l ← Cdollar[node];
        uminus => l ← Cuminus[node];
        addr => l ← Caddr[node];
        index => l ← Cindex[node];
        dindex => l ← Cdindex[node];
        constructx => l ← Cconstructx[node];
        vconstructx => l ← Cvconstructx[node];
        arraydesc => l ← Carraydesc[node];
        length => l ← Clength[node];
        base => l ← Cbase[node];
        portinit => l ← Cportinit[node];
        body => l ← Cbodyinit[node];
```

```
        rowconsx => 1 ← Crowconsx[node];
        stringinit => 1 ← Cstringinit[node];
        align => P5ADefs.P5Error[641];
        cast => 1 ← Cexp[(tb+node).son1];
        seqindex => 1 ← Cseqindex[(tb+node).son1,(tb+node).son2];
        register => 1 ← Cregister[node];
        memory =>
          BEGIN
          pushrhs[(tb+node).son1];
          1 ← Lexeme[bdo[makeTOSaddrBDOItem[wordlength]]];
          END;
        item => 1 ← Cexp[(tb+node).son2];
        temp => 1 ← gentemplex[SymTabDefs.WordsForType[(tb+node).info]];
        call, portcall => 1 ← Ccallexp[node];
        signal,error => 1 ← Csigerrexp[node];
        start => 1 ← Cstartexp[node];
        new => 1 ← Cnew[node];
        mwconst => 1 ← Cmwconst[node];
        signalinit => 1 ← Csignalinit[node];
        fork => 1 ← Cforkexp[node];
        join => 1 ← Cjoinexp[node];
        float => 1 ← Cfloat[node];
        ENDCASE => 1 ← Cflowexp[node];
      END;
    ENDCASE;
  RETURN
  END;


constoperand: PROCEDURE [t: TreeLink] RETURNS [BOOLEAN, INTEGER] =
  BEGIN -- if t is a literal node, return [TRUE,val(t)]
  IF treeliteral[t] THEN
    RETURN [TRUE, treeliteralvalue[t]]
  ELSE RETURN [FALSE, 0]
  END;

Dsyscall: PROCEDURE [op: BYTE] =
  BEGIN
  Csyscall1n[op,2];
  END;

Cplus: PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
  BEGIN -- generate code for +
  double: BOOLEAN = (tb+node).attr1;
  real: BOOLEAN;
  IF double THEN
    BEGIN
    RequireStack[0];
    IF (real ← (tb+node).attr2) THEN markstack[];
    END;
  pushrhs[(tb+node).son1];
  pushrhs[(tb+node).son2];
  IF double THEN
    BEGIN
    IF real THEN Dsyscall[SDDefs.sFADD]
    ELSE Ciout0[qDADD];
    RETURN[makeTOSlex[2]]
    END;
  Ciout0[qADD];
  RETURN[topostack]
  END;


Cminus: PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
  BEGIN -- generate code for -
  double: BOOLEAN = (tb+node).attr1;
  real: BOOLEAN;
  IF double THEN
    BEGIN
    RequireStack[0];
    IF (real ← (tb+node).attr2) THEN markstack[];
    END;
  pushrhs[(tb+node).son1];
  pushrhs[(tb+node).son2];
  IF double THEN
    BEGIN
```

```
    IF real THEN Dsyscall[SDDefs.sFSUB]
    ELSE Ciout0[qDSUB];
    RETURN[makeTOSlex[2]]
    END;
  Ciout0[qSUB];
  RETURN[topostack]
  END;


Cuminus: PROCEDURE [node: TreeIndex] RETURNS [l: Lexeme] =
  BEGIN -- generate code for unary minus
  tt: TreeLink ← (tb+node).son1;
  double: BOOLEAN = (tb+node).attr1;
  real: BOOLEAN;

  l ← IF double THEN makeTOSlex[2] ELSE topostack;
  WITH tt SELECT FROM
    subtree =>
      IF (tb+index).name = uminus THEN
        BEGIN pushrhs[(tb+index).son1]; RETURN END;
    ENDCASE;
  IF double THEN
    BEGIN
    RequireStack[0];
    IF (real ← (tb+node).attr2) THEN BEGIN markstack[]; markstack[]; END;
    pushlitval[0]; pushlitval[0];
    IF real THEN Dsyscall[SDDefs.sFLOAT];
    END;
  pushrhs[tt];
  IF double THEN
    IF real THEN Dsyscall[SDDefs.sFSUB]
    ELSE Ciout0[qDSUB]
  ELSE Ciout0[qNEG];
  RETURN
  END;


Ctimes: PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
  BEGIN -- generates code for multiply
  double: BOOLEAN = (tb+node).attr1;
  IF double THEN BEGIN RequireStack[0]; markstack[] END;
  pushrhs[(tb+node).son1];
  pushrhs[(tb+node).son2];
  IF double THEN
    BEGIN
    Dsyscall[IF (tb+node).attr2 THEN SDDefs.sFMUL
      ELSE SDDefs.sLongMul];
    RETURN[makeTOSlex[2]];
    END;
  Ciout0[qMUL];
  RETURN[topostack]
  END;


log2: PROCEDURE [i: INTEGER] RETURNS [BOOLEAN, [0..16]] =
  BEGIN OPEN InlineDefs;
  shift: [0..16];

  IF i = 0 THEN RETURN [FALSE, 0];
  i ← ABS[i];
  IF BITAND[i, i-1] # 0 THEN RETURN [FALSE, 0];
  FOR shift IN [0..16) DO
    IF BITAND[i,1] = 1 THEN RETURN[TRUE, shift];
    i ← BITSHIFT[i, -1];
    ENDLOOP
  END;


Cdiv: PROCEDURE [node: TreeIndex] RETURNS [l: Lexeme] =
  BEGIN -- generate code for divide
  double: BOOLEAN = (tb+node).attr1;
  rand2lit, powerof2: BOOLEAN;
  rand2val: INTEGER;
  shift: [0..16];

  l ← IF double THEN makeTOSlex[2] ELSE topostack;
```

```
    IF double THEN BEGIN RequireStack[0]; markstack[] END;
    pushrhs[(tb+node).son1];
    IF ~double AND (tb+node).attr2 THEN
      BEGIN
      [rand2lit, rand2val] ← constoperand[(tb+node).son2];
      IF rand2lit AND rand2val > 0 THEN
        BEGIN
        [powerof2, shift] ← log2[rand2val];
        IF powerof2 THEN
          BEGIN pushlitval[-shift]; Ciout0[qSHIFT]; RETURN END;
        END;
      END;
    pushrhs[(tb+node).son2];
    IF double THEN
      BEGIN
      Dsyscall[IF (tb+node).attr2 THEN SDDefs.sFDIV
        ELSE SDDefs.sLongDiv];
      RETURN[makeTOSlex[2]];
      END;
    IF (tb+node).attr2 THEN Ciout0[qDIV]
    ELSE Ciout0[qSDIV];
    RETURN
    END;


  Cmod: PROCEDURE [node: TreeIndex] RETURNS [l: Lexeme] =
    BEGIN -- generate code for MOD
    double: BOOLEAN = (tb+node).attr1;
    rand2lit, powerof2: BOOLEAN;
    rand2val: INTEGER;

    l ← IF double THEN makeTOSlex[2] ELSE topostack;
    IF double THEN
      BEGIN
      IF (tb+node).attr2 THEN SIGNAL CPtr.CodeNotImplemented;
      RequireStack[0]; markstack[]
      END;
    pushrhs[(tb+node).son1];
    IF ~double AND (tb+node).attr2 THEN
      BEGIN
      [rand2lit, rand2val] ← constoperand[(tb+node).son2];
      IF rand2lit AND rand2val > 0 THEN
        BEGIN
        [powerof2, ] ← log2[rand2val];
        IF powerof2 THEN
          BEGIN pushlitval[rand2val-1]; Ciout0[qAND]; RETURN END;
        END;
      END;
    pushrhs[(tb+node).son2];
    IF double THEN
      BEGIN
      Csyscall[SDDefs.sLongMod];
      CPtr.acstack ← 2;
      incrstack[2];
      RETURN
      END;
    IF (tb+node).attr2 THEN Ciout0[qDIV]
    ELSE Ciout0[qSDIV];
    Ciout0[qPUSH];
    Ciout0[qEXCH];
    Ciout0[qPOP];
    RETURN
    END;

  Cfloat: PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
    BEGIN
    RequireStack[0];
    markstack[];
    pushrhs[(tb+node).son1];
    Dsyscall[SDDefs.sFLOAT];
    RETURN[makeTOSlex[2]];
    END;

  Caddr: PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
    BEGIN -- generates code for "@"
    psize: CARDINAL = loadtsonaddress[(tb+node).son1];
```

```
    RETURN[IF psize > wordlength THEN makeTOSlex[2] ELSE topostack]
    END;


  Cregister: PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
    BEGIN -- creates lexeme for (some) r-register
    RETURN [Lexeme[other[register[treeliteralvalue[(tb+node).son1]]]]];
    END;


  Cregload: PUBLIC PROCEDURE [v: RegisterName] =
    BEGIN -- pushes value of (some) r-register on stack
    OPEN ControlDefs;
    SELECT v FROM
      Lreg => Ciout1[qLADRB, 0];
      Greg => Ciout1[qGADRB, 0];
      ENDCASE => IF v < 100B THEN BEGIN Ciout1[qRR, v] END
        ELSE SIGNAL CPtr.CodeNotImplemented;
    RETURN
    END;

  Cseqindex: PROCEDURE [string, index: TreeLink] RETURNS [Lexeme] =
    BEGIN
    psize: CARDINAL;
    psize ← spushrhs[string];
    pushrhs[index];
    RETURN [Lexeme[other[byte[StringHeaderSize*BytesPerWord, psize>wordlength]]]]
    END;


  Carraydesc: PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
    BEGIN -- pushes two components of an array descriptor onto stack
    size: CARDINAL;
    WITH (tb+node).son1 SELECT FROM
      subtree =>
        BEGIN
        size ← spushrhs[(tb+index).son1];
        size ← spushrhs[(tb+index).son2] + size;
        END;
      ENDCASE;
    RETURN[makeTOSlex[size/wordlength]]
    END;


  Clength: PROCEDURE [node: TreeIndex] RETURNS [l: bdo Lexeme] =
    BEGIN -- generates code to extract length from array descriptor
    r: BDOIndex;
    IF TreeDefs.testtree[(tb+node).son1, reloc] THEN
      SIGNAL CPtr.CodeNotImplemented;
    r ← (l ← maketsonBDOItem[(tb+node).son1]).lexbdoi;
    cb[r].offset.posn.wd ←
      cb[r].offset.posn.wd+(cb[r].offset.size-wordlength)/wordlength;
    cb[r].offset.size ← wordlength;
    RETURN
    END;


  Cbase: PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
    BEGIN -- generates code to extract base from array descriptor
    l: bdo Lexeme;
    psize: CARDINAL;
    t1: TreeLink ← (tb+node).son1;
    IF TreeDefs.testtree[t1, reloc] THEN
      BEGIN
      psize ← loadlexaddress[Creloc[LOOPHOLE[t1, subtree TreeLink].index, TRUE]];
      RETURN[IF psize = wordlength THEN topostack ELSE makeTOSlex[2]];
      END
    ELSE
      BEGIN
      l ← maketsonBDOItem[t1];
      cb[l.lexbdoi].offset.size ← cb[l.lexbdoi].offset.size - wordlength;
      END;
    RETURN[l]
    END;
```

```
  Cdotoruparrow: PROCEDURE [mainnode: TreeIndex]  RETURNS [Lexeme] =
    BEGIN
    -- generate code for "exp.field"
    t1: TreeLink ← (tb+mainnode).son1;
    sei: ISEIndex;
    node: TreeIndex;
    csei: CSEIndex;
    rr, r: BDOIndex;
    l, lr: Lexeme;
    duasym: PROCEDURE [tsei: ISEIndex] RETURNS [l: Lexeme] =
      BEGIN
      a: BitAddress;
      IF (seb+tsei).linkSpace THEN
        BEGIN
        pushlex[[se[tsei]]];
        RETURN[[bdo[r ← makeTOSaddrBDOItem[wordlength]]]];
        END;
      l ← Lexeme[bdo[r ← genBDOItem[]]];
      cb[r].tag ← bo;
      a ← (seb+tsei).idvalue;
      cb[r].base ←
            BDOComponent[posn: FullBitAddress[bd: a.bd, wd: a.wd],
                size: (seb+tsei).idinfo,
                level: (ctxb+(seb+tsei).ctxnum).ctxlevel];
      RETURN
      END;

    IF (tb+mainnode).name = uparrow THEN sei ← (tb+mainnode).info
    ELSE
      WITH (tb+mainnode).son2 SELECT FROM
        symbol => sei ← index;
        ENDCASE;
    WITH t1 SELECT FROM -- produces better code if LOOPHOLE is present
      subtree => IF (tb+index).name = cast THEN t1 ← (tb+index).son1;
      ENDCASE;
    WITH t1 SELECT FROM
      symbol => l ← duasym[index];
      subtree =>
        BEGIN
        node ← index;
        SELECT (tb+node).name FROM
          plus =>
            BEGIN
            r ← genBDOItem[];
            cb[r].tag ← bdo;
            l ← Cexp[(tb+node).son1];
            WITH l SELECT FROM
              se => l ← makeBDOItem[l];
              bdo => NULL;
              ENDCASE => BEGIN l ← lpushlex[l]; END;
            WITH l SELECT FROM
              bdo => IF cb[lexbdoi].tag # o THEN l ← lpushlex[l];
              ENDCASE;
            WITH l SELECT FROM
              se => cb[r].base ← [level: lTOS, posn: FullBitAddress[0, 0], size: wordlength]; -- topost
**ack if here
              bdo =>
                BEGIN
                rr ← lexbdoi;
                cb[r].base ← cb[rr].offset;
                releaseBDOItem[rr];
                END;
              ENDCASE;
            lr ← Cexp[(tb+node).son2];
            WITH lr SELECT FROM
              se => lr ← makeBDOItem[lr];
              bdo => NULL;
              ENDCASE => lr ← lpushlex[lr];
            WITH lr SELECT FROM
              bdo => IF cb[lexbdoi].tag # o THEN lr ← lpushlex[lr];
              ENDCASE;
            WITH lr SELECT FROM
              se => cb[r].disp ← [level: lTOS, posn: FullBitAddress[0, 0], size: wordlength]; -- topost
**ack if here
              bdo =>
                BEGIN
```

```
            rr ← lexbdoi;
            cb[r].disp ← cb[rr].offset;
            releaseBDOItem[rr];
            END;
          ENDCASE;
        1 ← [bdo[r]];
        END;
      ENDCASE =>
        BEGIN
        1 ← [bdo[r ← maketempaddrBDOItem[lpushrhs[t1]]]];
        END;
    END;
  literal =>
    BEGIN
    pushconst[t1];
    1 ← Lexeme[bdo[r ← makeTOSaddrBDOItem[wordlength]]];
    END;
  ENDCASE;
cb[r].offset.level ← 1Z;
IF (tb+mainnode).name = uparrow THEN
  BEGIN
  cb[r].offset.size ← wordlength*wordsforsei[sei];
  cb[r].offset.posn ← FullBitAddress[0,0];
  END ELSE
IF (seb+sei).constant THEN
  BEGIN
  ConstantField[r, sei];
  RETURN[topostack]
  END
ELSE
  BEGIN
  WITH (seb+SymTabDefs.NormalType[operandtype[(tb+mainnode).son1]]) SELECT FROM
    pointer =>
      BEGIN OPEN SymTabDefs;
      cb[r].offset.posn ← FullBitAddress[0,0];
      csei ← UnderType[pointedtotype];
      cb[r].offset.size ← adjustbdoitem[r, csei, sei, WordsForType[csei]*wordlength];
      addBitAddresstooffset[r, (seb+sei).idvalue];
      END;
    ENDCASE => P5ADefs.P5Error[642];
  END;
RETURN[1]
END;


Creloc: PUBLIC PROCEDURE [node: TreeIndex, allowdescriptor: BOOLEAN]
  RETURNS [Lexeme] =
  BEGIN -- generates code for "baseptr[relptr]"
  psize: CARDINAL;
  rb, rd, rr: BDOIndex;
  rr ← genBDOItem[];
  cb[rr].offset ← WordZeroBDOComponent;
  cb[rr].offset.size ← wordlength*SymTabDefs.WordsForType[(tb+node).info];
  cb[rr].tag ← bdo;

  rb ← rmakeBDOItem[Cexp[(tb+node).son1]];
  psize ← cb[rb].offset.size;
  IF cb[rb].tag = o THEN
    BEGIN
    cb[rr].base ← cb[rb].offset;
    releaseBDOItem[rb];
    END
  ELSE
    BEGIN
    Cload[rb];
    cb[rr].base ← TosBDOComponent;
    cb[rr].base.size ← MAX[wordlength, psize];
    END;

  rd ← rmakeBDOItem[Cexp[(tb+node).son2]];
  IF (tb+node).attr2 THEN
    BEGIN
    IF cb[rd].tag = o AND cb[rd].offset.level = lTOS THEN Ciout0[qPOP];
    cb[rd].offset.size ← cb[rd].offset.size-wordlength;
    END;
  psize ← cb[rd].offset.size;
```

```
    IF cb[rd].tag = o THEN
      BEGIN
      cb[rr].disp ← cb[rd].offset;
      releaseBDOItem[rd];
      END
    ELSE
      BEGIN
      Cload[rd];
      cb[rr].disp ← TosBDOComponent;
      cb[rr].disp.size ← MAX[wordlength, psize];
      END;

    RETURN[[bdo[rr]]];
    END;


  ConstantField: PROCEDURE [r: BDOIndex, sei: ISEIndex] =
    BEGIN
    p: ControlDefs.ProcDesc;
    bti: CBTIndex;
    cb[r].offset.size ← wordlength;
    cb[r].offset.posn ← [0,0];
    SELECT SymTabDefs.XferMode[(seb+sei).idtype] FROM
      procedure =>
        BEGIN
        IF (seb+sei).extended THEN SIGNAL CPtr.CodeNotImplemented;
        bti ← (seb+sei).idinfo;
        IF bti = BTNull THEN
          BEGIN pushlitval[(seb+sei).idvalue]; RETURN END;
        WITH (bb+bti) SELECT FROM
          Inner =>
            BEGIN
            cb[r].offset.posn.wd ← frameOffset;
            IF loadaddress[r] # wordlength THEN
              SIGNAL CPtr.CodeNotImplemented;
            END;
          Outer =>
            BEGIN OPEN ControlDefs;
            IF loadaddress[r] # wordlength THEN
              SIGNAL CPtr.CodeNotImplemented;
            p.gfi ← entryIndex/EPRange;
            p.ep ← entryIndex MOD EPRange;
            p.tag ← procedure;
            Ciout1[qDESCBS, LOOPHOLE[p]];
            END;
          ENDCASE;
        END;
      signal, error =>
        BEGIN
        IF loadaddress[r] # wordlength THEN
          SIGNAL CPtr.CodeNotImplemented;
        Ciout1[qDESCBS, (seb+sei).idvalue];
        END;
      ENDCASE => P5ADefs.P5Error[643];
    END;

  Cdollar: PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
    BEGIN -- generates code for "exp$field"
    sei: ISEIndex;
    res: CARDINAL;
    l: bdo Lexeme;
    recsei: CSEIndex ← operandtype[(tb+node).son1];
    functionCall: BOOLEAN;
    rep: BitAddress;

    WITH (seb+recsei) SELECT FROM
      record => functionCall ← argument;
      ENDCASE => P5ADefs.P5Error[644];

    l ← makeBDOItem[Cexp[(tb+node).son1]];
    WITH (tb+node).son2 SELECT FROM
      symbol =>
        BEGIN
        sei ← index;
        IF (seb+sei).constant THEN
          BEGIN
```

```
        WITH (tb+node).son1 SELECT FROM
          subtree => IF (tb+index).name # uparrow THEN P5ADefs.P5Error[645];
          ENDCASE => P5ADefs.P5Error[646];
        ConstantField[l.lexbdoi, sei];
        RETURN [topostack];
        END;
      IF functionCall THEN
        [rep,res] ← SymTabDefs.FnField[sei]
      ELSE
        BEGIN
        res ← adjustbdoitem[l.lexbdoi, recsei, sei, cb[l.lexbdoi].offset.size];
        rep ← (seb+sei).idvalue;
        END;
      sdollar[l, rep, res];
      RETURN [l];
      END;
    ENDCASE
  END;


adjustbdoitem: PROCEDURE [r: BDOIndex, rsei: CSEIndex, fieldsei: ISEIndex, destsize: CARDINAL]
      RETURNS [fieldsize: CARDINAL] =
  BEGIN
  pad: CARDINAL;

  fieldsize ← (seb+fieldsei).idinfo;
  WITH (seb+rsei) SELECT FROM
    record =>
      BEGIN
      IF length < wordlength AND length < destsize THEN
        BEGIN
        pad ← destsize - length;
        IF (seb+fieldsei).idvalue = 0 THEN
          fieldsize ← fieldsize + pad
        ELSE cb[r].offset.posn ← addfulladdrtobits[cb[r].offset.posn, pad];
        END;
      RETURN
      END;
    ENDCASE => ERROR
  END;

sdollar: PROCEDURE [l: bdo Lexeme, rep: BitAddress, res: CARDINAL] =
  BEGIN -- main subroutine for Cdollar and Cfdollar
  OPEN AltoDefs;
  r: BDOIndex ← l.lexbdoi;
  ss: CARDINAL;

  IF cb[r].tag = o AND cb[r].offset.level = lTOS
        AND (ss ← cb[r].offset.size) > wordlength THEN
    BEGIN
    THROUGH [rep.wd + (res+wordlength-1)/wordlength .. ss/wordlength) DO
      Cioutθ[qPOP];
      cb[r].offset.size ← cb[r].offset.size - wordlength;
      ENDLOOP;
    IF res <= wordlength THEN
      UNTIL rep.wd = 0 DO
        Cioutθ[qEXCH]; Cioutθ[qPOP];
        cb[r].offset.size ← cb[r].offset.size - wordlength;
        rep.wd ← rep.wd -1;
        ENDLOOP;
    END;
  addBitAddresstooffset[r, rep];
  cb[r].offset.size ← res;
  RETURN
  END;


addBitAddresstooffset: PROCEDURE[r: BDOIndex, rep: BitAddress] =
  BEGIN
  w: CARDINAL;

  [w, cb[r].offset.posn.bd] ← InlineDefs.DIVMOD[cb[r].offset.posn.bd + rep.bd, wordlength];
  cb[r].offset.posn.wd ← cb[r].offset.posn.wd + (rep.wd + w);
  RETURN
  END;
```

```
CopyLex: PROCEDURE [l: Lexeme] RETURNS [Lexeme] =
  BEGIN
  WITH l SELECT FROM
    bdo => RETURN [[bdo[copyBDOItem[lexbdoi]]]];
    ENDCASE => RETURN[l];
  END;

MWConstant: PUBLIC SIGNAL [cOffseg: CARDINAL] RETURNS [Lexeme] = CODE;

Cmwconst: PROCEDURE [node: TreeIndex] RETURNS [l: Lexeme] =
  BEGIN -- puts multi-word constant out to code stream and puts address on TOS
  cOffset, destpsize: CARDINAL;
  lti: LTIndex;
  nwords: CARDINAL;
  i: CARDINAL;

  WITH (tb+node).son1 SELECT FROM
    literal => WITH info SELECT FROM
      word => lti ← index;
      ENDCASE => P5ADefs.P5Error[647];
    ENDCASE => P5ADefs.P5Error[648];
  WITH ll:(ltb+lti) SELECT FROM
    short => RETURN [[literal[word[lti]]]];
    long =>
      BEGIN
      SELECT ll.length FROM
        0 => P5ADefs.P5Error[649];
        1 =>
          BEGIN pushlitval[ll.value[0]]; RETURN[topostack] END;
        2 =>
          BEGIN
          pushlitval[ll.value[0]];
          pushlitval[ll.value[1]];
          l ← makeTOSlex[2];
          RETURN
          END;
        ENDCASE;
      nwords ← ll.length;
      IF ll.codeIndex = 0 THEN
        BEGIN
        ll.codeIndex ← movetocodeword[];
        FOR i IN [0..nwords) DO writecodeword[ll.value[i]]; ENDLOOP;
        END;
      cOffset ← ll.codeIndex;
      END;
    ENDCASE;
  l ← SIGNAL MWConstant[cOffset];
  RequireStack[0];
  pushlitval[cOffset];
  pushlitval[nwords];
  destpsize ← loadlexaddress[CopyLex[l]];
  Ciout0[IF destpsize = wordlength THEN qBLTC ELSE qBLTCL];
  RETURN
  END;

lpushrhs: PUBLIC PROCEDURE [t: TreeLink] RETURNS [Lexeme] =
  BEGIN -- forces a value onto the stack
  size: CARDINAL ← spushrhs[t];
  RETURN [IF size <= wordlength THEN topostack
      ELSE makeTOSlex[size/wordlength]];
  END;


pushrhs: PUBLIC PROCEDURE [t: TreeLink] =
  BEGIN -- forces a value onto the stack
  [] ← spushrhs[t];
  RETURN
  END;


spushrhs: PROCEDURE [t: TreeLink] RETURNS [size: CARDINAL] =
  BEGIN -- forces a value onto the stack
  size ← wordlength;
  IF t = empty THEN
    BEGIN
    IF CPtr.xtracting THEN RETURN[spushlex[CPtr.xtractlex]];
```

```
    IF CPtr.firstcaseselread THEN CPtr.firstcaseselread ← FALSE
    ELSE Ciout0[qPUSH];
    END
  ELSE RETURN[spushlex[Cexp[t]]];
  RETURN
  END;


spushlex: PROCEDURE [l: Lexeme] RETURNS [size: CARDINAL] =
  BEGIN -- forces a lexeme onto the stack
  a: BitAddress;
  bti: CBTIndex;
  r: BDOIndex;
  size ← wordlength;
  IF l = topostack THEN RETURN;
  WITH e: l SELECT FROM
    literal =>
      WITH e SELECT FROM
        word => pushconst[TreeLink[literal[[word[lexlti]]]]];
        string => pushconst[TreeLink[literal[[string[lexsti]]]]];
        ENDCASE;
    se =>
      BEGIN
      IF (seb+e.lexsei).linkSpace THEN
        BEGIN a ← (seb+e.lexsei).idvalue; Ciout1[qLLK, a.wd] END
      ELSE IF (seb+e.lexsei).constant THEN
            SELECT SymTabDefs.XferMode[(seb+e.lexsei).idtype] FROM
              procedure =>
                BEGIN
                bti ← (seb+e.lexsei).idinfo;
                IF bti = BTNull THEN pushlitval[(seb+e.lexsei).idvalue]
                ELSE pushlprocdesc[bti];
                END;
              signal, error => pushlsigdesc[(seb+e.lexsei).idvalue];
              ENDCASE => ERROR
      ELSE
        BEGIN
        r ← rmakeBDOItem[e];
        size ← cb[r].offset.size;
        Cload[r];
        END;
      END;
    bdo => BEGIN size ← cb[e.lexbdoi].offset.size; Cload[e.lexbdoi]; END;
    other => WITH e SELECT FROM
      register => Cregload[lexrn];
      byte =>
        BEGIN
        Ciout1[(IF long THEN qRSTRL ELSE qRSTR), lexalpha];
        RETURN
        END;
      ENDCASE;
    ENDCASE;
  RETURN
  END;


pushlex: PUBLIC PROCEDURE [l: Lexeme] =
  BEGIN
  [] ← spushlex[l];
  END;


lpushlex: PUBLIC PROCEDURE [l: Lexeme] RETURNS [Lexeme] =
  BEGIN
  size: CARDINAL ← spushlex[l];
  RETURN [IF size <= wordlength THEN topostack
      ELSE makeTOSlex[size/wordlength]];
  END;


pushconst: PUBLIC PROCEDURE [t: TreeLink] =
  BEGIN -- forces a 16-bit constant onto the stack
  msti: MSTIndex;
  IF treeliteral[t] THEN
    BEGIN pushlitval[treeliteralvalue[t]]; RETURN END;
  WITH e: t SELECT FROM
```

```
      literal =>
        WITH e.info SELECT FROM
          string =>
            BEGIN
            msti ← LitDefs.MasterString[index];
            IF ~(stb+msti).local THEN Ciout1[qGADRB, (stb+msti).info]
            ELSE
              BEGIN
              r: BDOIndex ← genBDOItem[];
              cb[r].tag ← o;
              cb[r].offset ← [
                posn: [wd: (stb+msti).info, bd: 0],
                size: wordlength,
                level: CPtr.curctxlvl - CPtr.catchcount];
              [] ← loadaddress[r];
              END;
            END;
          ENDCASE;
      ENDCASE => P5ADefs.P5Error[650];
    RETURN
    END;


  pushlitval: PUBLIC PROCEDURE [v: WORD] =
    BEGIN -- forces a constant onto the stack
    Ciout1[qLI, v];
    RETURN
    END;


  pushlprocdesc: PUBLIC PROCEDURE [bti: CBTIndex] =
    BEGIN -- pushes a descriptor for local procedure on stack
    WITH (bb+bti) SELECT FROM
      Inner => pushlnestedprocdesc[bti];
      Outer => pushlnonnestedprocdesc[entryIndex];
      ENDCASE;
    RETURN
    END;

  pushlnestedprocdesc: PUBLIC PROCEDURE [bti: CBTIndex] =
    BEGIN -- pushes a descriptor for nested local procedure on stack
    v: ContextLevel ← (bb+bti).level - 1;
    r: BDOIndex;

    WITH (bb+bti) SELECT FROM
      Inner =>
        BEGIN
        r ← genBDOItem[];
        cb[r].tag ← o;
        cb[r].offset ← [level: v, posn: [wd: frameOffset, bd: 0], size: wordlength];
        [] ← loadaddress[r];
        RETURN
        END;
      ENDCASE
    END;

  pushlnonnestedprocdesc: PUBLIC PROCEDURE [n: CARDINAL] =
    BEGIN -- pushes a descriptor for local procedure n on stack
    OPEN ControlDefs;
    p: ProcDesc;

    p.gfi ← n/EPRange;
    p.ep ← n MOD EPRange;
    p.tag ← procedure;
    Ciout1[qDESCB, LOOPHOLE[p]];
    RETURN
    END;

  pushlsigdesc: PROCEDURE [desc: ControlDefs.SignalDesc] =
    BEGIN
    IF desc.gfi # ControlDefs.GFTNull THEN Ciout1[qDESCB, LOOPHOLE[desc]]
    ELSE pushlitval[LOOPHOLE[desc]];
    RETURN
    END;

  Csignalinit: PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
```

```
    BEGIN  OPEN ControlDefs;
    v: CARDINAL ← (tb+node).info;

    Ciout1[qDESCB, LOOPHOLE[ControlLink[procedure[
        gfi: v/EPRange,
        ep: v MOD EPRange,
        tag: procedure]]]];
    RETURN [topostack]
    END;


  END...
```