```
-- Address.mesa, modified by Sweet, July 18, 1978  9:53 AM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [BYTE, charlength, wordlength],
  Code: FROM "code" USING [curctxlvl, firstcaseselread],
  CodeDefs: FROM "codedefs" USING [BDOComponent, BDOComponentNames, BDOIndex, BDOItem, BDONull, ChunkBa
**se, FullBitAddress, GetChunk, InUseThread, Lexeme, lTOS, topostack],
  ComData: FROM "comdata" USING [typeINTEGER],
  ControlDefs: FROM "controldefs" USING [FieldDescriptor, framelink, globalbase, localbase],
  FOpCodes: FROM "fopcodes" USING [qADD, qAND, qDADD, qGADRB, qLADRB, qLG, qLGD, qLI, qLL, qLLD, qPOP,
**qPUSH, qR, qRD, qRDL, qRF, qRFC, qRFL, qRIG, qRIGL, qRIL, qRILL, qRL, qRXGL, qRXL, qRXLL, qSG, qSGD,
**qSL, qSLD, qW, qWD, qWDL, qWF, qWFL, qWIG, qWIGL, qWIL, qWILL, qWL, qWXGL, qWXL, qWXLL],
  LitDefs: FROM "litdefs" USING [FindLiteral, LTIndex],
  OpCodeParams: FROM "opcodeparams" USING [HB, LocalHB, GlobalHB],
  P5ADefs: FROM "p5adefs" USING [bltnwordsfromstack, Ciout0, Ciout1, Ciout2, gentemplex, operandtype, P
**5Error, pop, RequireStack, sCassign, treeliteral, treeliteralvalue],
  P5BDefs: FROM "p5bdefs" USING [Cexp, Cregload, lpushlex, MWConstant, pushlex, pushlitval, pushrhs],
  P5StmtExprDefs: FROM "p5stmtexprdefs",
  SymDefs: FROM "symdefs" USING [BitAddress, ContextLevel, CSEIndex, CTXIndex, ctxtype, HTIndex, ISEInd
**ex, lG, lZ, SEIndex, setype],
  SymTabDefs: FROM "symtabdefs" USING [BitsForType, NormalType, UnderType, WordsForType],
  TableDefs: FROM "tabledefs" USING [TableBase, TableNotifier],
  TreeDefs: FROM "treedefs" USING [empty, freenode, mlpop, mlpush, pushlittree, pushtree, setattr, seti
**nfo, TreeIndex, TreeLink, treetype];

DEFINITIONS FROM CodeDefs;

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

  -- imported definitions

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

  framelink: CARDINAL = ControlDefs.framelink;
  globalbase: CARDINAL = ControlDefs.globalbase;
  localbase: CARDINAL = ControlDefs.localbase;

  BitAddress: TYPE = SymDefs.BitAddress;
  ContextLevel: TYPE = SymDefs.ContextLevel;
  CTXIndex: TYPE = SymDefs.CTXIndex;
  HTIndex: TYPE = SymDefs.HTIndex;
  ISEIndex: TYPE = SymDefs.ISEIndex;
  CSEIndex: TYPE = SymDefs.CSEIndex;
  lG: ContextLevel = SymDefs.lG;
  lZ: ContextLevel = SymDefs.lZ;
  SEIndex: TYPE = SymDefs.SEIndex;

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

  LTIndex: TYPE = LitDefs.LTIndex;

  InvalidBDOItemRelease: SIGNAL = CODE;
  AddressingError: SIGNAL = CODE;

  BDOItemlist: BDOIndex;

  WordZeroBDOComponent, TosBDOComponent: BDOComponent;


  tb: TableDefs.TableBase;              -- tree base (local copy)
  seb: TableDefs.TableBase;             -- semantic entry base (local copy)
  ctxb: TableDefs.TableBase;            -- context entry base (local copy)
  cb: ChunkBase;                -- code base (local copy)

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

  AddressError: PROCEDURE = BEGIN SIGNAL AddressingError; RETURN END;

  AddressInit: PUBLIC PROCEDURE =
    BEGIN -- called by Cmodule to init stuff in Addr
    BDOcount ← BDOcard ← 0;
    BDOItemlist ← BDONull;
    TosBDOComponent ←
        BDOComponent[level: lTOS, posn: FullBitAddress[0, 0], size: wordlength];
    WordZeroBDOComponent ←
        BDOComponent[level: lZ, posn: FullBitAddress[0, 0], size: wordlength];
    RETURN
    END;


  InvalidField: SIGNAL [RECORD[p,s: BYTE]] = CODE;

  FieldParam: PUBLIC PROCEDURE [r: BDOIndex] RETURNS [WORD] =
    BEGIN
    fd: ControlDefs.FieldDescriptor;
    p: CARDINAL ← cb[r].offset.posn.bd;
    s: CARDINAL ← cb[r].offset.size;
    fd ← [offset: 0,
     posn: p,
     size: s];
    IF p+s > wordlength THEN SIGNAL InvalidField[[p,s]];
    RETURN [LOOPHOLE[fd]];
    END;

  addfulladdrtobits: PUBLIC PROCEDURE [f: FullBitAddress, b: CARDINAL] RETURNS [rf: FullBitAddress] =
    BEGIN
    v: CARDINAL ← f.bd + b MOD wordlength;

    rf.bd ← v MOD wordlength;
    rf.wd ← f.wd + b/wordlength + v/wordlength;
    RETURN
    END;


  fulladdress: PROCEDURE [a: BitAddress] RETURNS [rf: FullBitAddress] =
    BEGIN
    rf.wd ← a.wd; rf.bd ← a.bd;
    RETURN
    END;


  rmakeBDOItem: PUBLIC PROCEDURE [l: Lexeme] RETURNS [BDOIndex] =
    BEGIN -- same as makeBDOItem, but returns BDOIndex
    RETURN [makeBDOItem[l].lexbdoi];
    END;


  makeBDOItem: PUBLIC PROCEDURE [l: Lexeme] RETURNS [bdo Lexeme] =
    BEGIN -- forces l into lexeme-record format
    r: BDOIndex;

    WITH incomingl: l SELECT FROM
      bdo => RETURN[incomingl];
      other => WITH incomingl SELECT FROM
        register =>
          BEGIN Cregload[lexrn]; RETURN [makeBDOItem[topostack]] END;
        byte =>
          BEGIN pushlex[l]; RETURN [makeBDOItem[topostack]] END;
        ENDCASE => P5ADefs.P5Error[321];
      literal =>
        BEGIN pushlex[l]; RETURN [makeBDOItem[topostack]] END;
      se =>
        BEGIN
        r ← genBDOItem[];
        IF incomingl = topostack THEN cb[r].offset ← TosBDOComponent
        ELSE
          cb[r].offset ← BDOComponent[level: (ctxb+(seb+incomingl.lexsei).ctxnum).ctxlevel,
            posn: fulladdress[(seb+incomingl.lexsei).idvalue],
            size: (seb+incomingl.lexsei).idinfo];
      END;
    ENDCASE;
  cb[r].tag ← o;
  RETURN[Lexeme[bdo[r]]];
  END;


copyBDOItem: PUBLIC PROCEDURE [r: BDOIndex] RETURNS [rr: BDOIndex] =
  BEGIN -- returns rr as a copy of r
  rr ← genBDOItem[];
  cb[rr] ← cb[r];
  RETURN
  END;


maketsonBDOItem: PUBLIC PROCEDURE [t: TreeLink] RETURNS [bdo Lexeme] =
  BEGIN -- another interface to makeBDOItem
  RETURN[makeBDOItem[Cexp[t]]]
  END;


makeTOSaddrBDOItem: PUBLIC PROCEDURE [psize: CARDINAL] RETURNS [r: BDOIndex] =
  BEGIN -- makes a rec-lexeme for an address on TOS
  r ← genBDOItem[];
  cb[r].base ← TosBDOComponent;
  cb[r].base.size ← FullWordBits[psize];
  cb[r].offset ← WordZeroBDOComponent;
  cb[r].tag ← bo;
  RETURN
  END;


maketempaddrBDOItem: PUBLIC PROCEDURE [tlex: Lexeme] RETURNS [r: BDOIndex] =
  BEGIN -- makes a type-bo rec with temp (tlex) as pointer part
  r ← rmakeBDOItem[tlex];
  cb[r].base ← cb[r].offset;
  cb[r].offset ← WordZeroBDOComponent;
  cb[r].tag ← bo;
  RETURN
  END;


makeretlex: PUBLIC PROCEDURE [nwords, psize: CARDINAL] RETURNS [Lexeme] =
  BEGIN -- makes the appropriate TOS return of 1,2 or many values
  b: bdo Lexeme;
  SELECT nwords FROM
    1 => RETURN[topostack];
    2 => RETURN[makeTOSlex[2]];
    ENDCASE =>
      BEGIN
      b ← Lexeme[bdo[makeTOSaddrBDOItem[psize]]];
      cb[b.lexbdoi].offset.size ← nwords*wordlength;
      RETURN [b]
      END;
  END;


makeTOSlex: PUBLIC PROCEDURE [nwords: CARDINAL] RETURNS [bdo Lexeme] =
  BEGIN -- makes a record-type lexeme for nwords on stack
  r: BDOIndex ← genBDOItem[];

  cb[r].offset ← TosBDOComponent;
  cb[r].tag ← o;
  cb[r].offset.size ← nwords*wordlength;
  RETURN[Lexeme[bdo[r]]]
  END;


Cload: PUBLIC PROCEDURE [r: BDOIndex] =
  BEGIN -- generates code for rhs
  SELECT cb[r].tag FROM
    o => Cvarload[r];
    bo => Cptrload[r];
    bdo => Cindexedptrload[r];
```

```
      ENDCASE => BEGIN AddressError[]; releaseBDOItem[r]; END;
    RETURN
    END;


  loadlexaddress: PUBLIC PROCEDURE [l: Lexeme] RETURNS [CARDINAL] =
    BEGIN -- interfaces to loadaddress with lexeme parameter
    RETURN[loadaddress[rmakeBDOItem[l]]];
    END;


  loadseiaddress: PUBLIC PROCEDURE [sei: ISEIndex] RETURNS [CARDINAL] =
    BEGIN -- interfaces to loadaddress with sei parameter
    RETURN[loadlexaddress[Lexeme[se[lexsei: sei]]]];
    END;


  loadtsonaddress: PUBLIC PROCEDURE [t: TreeLink] RETURNS [CARDINAL] =
    BEGIN -- interfaces to loadaddress with tson parameter
    RETURN[loadaddress[rmakeBDOItem[Cexp[t]]]];
    END;


  loadaddress: PUBLIC PROCEDURE [r: BDOIndex] RETURNS [psize: CARDINAL] =
    BEGIN -- loads the address of the BDOItem's word zero onto stack
    tlex: se Lexeme;
    nwords: CARDINAL ← cb[r].offset.size/wordlength;
    delta: CARDINAL ← cb[r].offset.posn.wd;
    long: BOOLEAN ← FALSE;

    IF cb[r].tag = bdo THEN
      BEGIN
      loaddisp: PROCEDURE =
        BEGIN
        pushcomponent[dispcomponent, r];
        IF long AND cb[r].disp.size <= wordlength THEN
          Ciout1[FOpCodes.qLI, 0];
        END;
      loadbase: PROCEDURE =
        BEGIN
        pushcomponent[basecomponent, r];
        psize ← cb[r].base.size;
        END;
      baseOnStack, dispOnStack: BOOLEAN ← FALSE;
      IF cb[r].disp.size > wordlength THEN basedispcommute[r];
      IF cb[r].base.size > wordlength THEN
        BEGIN
        onstack: CARDINAL ← 0;
        IF cb[r].base.level = lTOS THEN
          BEGIN onstack ← onstack+(cb[r].base.size+wordlength-1)/wordlength;
          baseOnStack ← TRUE;
          END;
        IF cb[r].disp.level = lTOS THEN
          BEGIN onstack ← onstack+(cb[r].disp.size+wordlength-1)/wordlength;
          dispOnStack ← TRUE;
          END;
        long ← TRUE;
        RequireStack[onstack]
        END;
      IF dispOnStack AND ~baseOnStack THEN
        BEGIN loaddisp[]; loadbase[] END
      ELSE BEGIN loadbase[]; loaddisp[] END;
      IF long THEN
        BEGIN
        Ciout0[FOpCodes.qDADD];
        END
      ELSE Ciout0[FOpCodes.qADD];
      cb[r].tag ← bo;
      END;
    IF cb[r].tag = bo THEN
      BEGIN
      IF cb[r].base.size > wordlength THEN
        BEGIN long ← TRUE; IF delta # 0 THEN RequireStack[0]; END;
      pushcomponent[basecomponent, r];
      psize ← cb[r].base.size;
      IF delta # 0 THEN
```

```
        BEGIN pushlitval[cb[r].offset.posn.wd];
        IF long THEN
          BEGIN Ciout1[FOpCodes.qLI, 0]; Ciout0[FOpCodes.qDADD] END
        ELSE Ciout0[FOpCodes.qADD];
        END;
      releaseBDOItem[r];
      RETURN
      END;
    psize ← wordlength;
    IF cb[r].offset.level = lTOS THEN
      BEGIN
      tlex ← bltnwordsfromstack[nwords];
      THROUGH [0..nwords) DO pop[] ENDLOOP;
      releaseBDOItem[r];
      [] ← loadlexaddress[tlex];
      RETURN
      END;
    IF cb[r].offset.level # CPtr.curctxlvl AND cb[r].offset.level # lG THEN
      BEGIN
      GetFrame[r];
      [] ← loadaddress[r];
      RETURN
      END;
    IF cb[r].offset.level = lG THEN
      Ciout1[FOpCodes.qGADRB, cb[r].offset.posn.wd]
    ELSE Ciout1[FOpCodes.qLADRB, cb[r].offset.posn.wd];
    releaseBDOItem[r];
    RETURN
    END;


  loadaddr: PROCEDURE [r: BDOIndex] =
    BEGIN -- load the address in r (type o) onto stack and adjust offset of r
    rr: BDOIndex ← genBDOItem[];

    cb[rr].offset ← cb[r].offset;
    cb[rr].tag ← o;
    [] ← loadaddress[rr];
    cb[r].offset.posn.wd ← 0;
    cb[r].offset.level ← lZ;
    cb[r].base ← TosBDOComponent;
    cb[r].tag ← bo;
    RETURN
    END;


  loadlex: PROCEDURE [l: ContextLevel, wordoffset, nwords: INTEGER] =
    BEGIN -- loads 1 or 2 words at level l, offset wordoffset, onto stack
    rr: BDOIndex ← genBDOItem[];

    cb[rr].tag ← o;
    cb[rr].offset ←
        BDOComponent[level: l, posn: FullBitAddress[wd: wordoffset, bd: 0], size: nwords*wordlength];
    Cvarload[rr];
    RETURN
    END;


  Cvarload: PROCEDURE [r: BDOIndex] =
    BEGIN -- loads a type-o BDOItem onto stack
    OPEN FOpCodes;
    l: ContextLevel ← cb[r].offset.level;
    v: CARDINAL ← cb[r].offset.posn.wd;
    s: CARDINAL;
    tlex: se Lexeme;
    g: BOOLEAN ← l=lG;
    rr: BDOIndex;

    IF l = lTOS THEN
      BEGIN
      IF cb[r].offset.posn = FullBitAddress[0,0] AND cb[r].offset.size >= wordlength THEN
        BEGIN releaseBDOItem[r]; RETURN END;
      tlex ← gentemplex[(cb[r].offset.size+wordlength-1)/wordlength];
      sCassign[tlex.lexsei];
      THROUGH [1..cb[r].offset.posn.wd] DO Ciout0[FOpCodes.qPOP] ENDLOOP;
      rr ← rmakeBDOItem[tlex];
      cb[rr].offset.posn.bd ← cb[r].offset.posn.bd;
      cb[rr].offset.size ← cb[r].offset.size;
      releaseBDOItem[r];
      Cvarload[rr];
      RETURN
      END;
    IF ~g AND l # CPtr.curctxlvl THEN
      BEGIN
      GetFrame[r];
      Cload[r];
      RETURN
      END;
    IF cb[r].offset.size = 2*wordlength THEN
      BEGIN
      IF g THEN Ciout1[qLGD, v] ELSE Ciout1[qLLD, v];
      releaseBDOItem[r];
      RETURN
      END;
    IF cb[r].offset.size > wordlength THEN
      BEGIN
      s ← cb[r].offset.size/wordlength;
      v ← cb[r].offset.posn.wd;
      WHILE s >= 2 DO loadlex[l, v, 2]; v ← v+2; s ← s-2; ENDLOOP;
      IF s # 0 THEN loadlex[l, v, 1];
      releaseBDOItem[r];
      RETURN
      END;
    IF cb[r].offset.size < wordlength THEN
      BEGIN loadaddr[r]; Cptrload[r]; RETURN END;
    IF g THEN Ciout1[qLG, v] ELSE Ciout1[qLL, v];
    releaseBDOItem[r];
    RETURN
    END;


  OperandSize: TYPE = {single,double,field};
  PtrLength: TYPE = [1..2];
  ReadOp: ARRAY OperandSize OF PACKED ARRAY PtrLength OF BYTE =
    [[FOpCodes.qR, FOpCodes.qRL], [FOpCodes.qRD, FOpCodes.qRDL], [FOpCodes.qRF, FOpCodes.qRFL]];
  WriteOp: ARRAY OperandSize OF PACKED ARRAY PtrLength OF BYTE =
    [[FOpCodes.qW, FOpCodes.qWL], [FOpCodes.qWD, FOpCodes.qWDL], [FOpCodes.qWF, FOpCodes.qWFL]];
  RilOp: ARRAY PtrLength OF PACKED ARRAY BOOLEAN OF BYTE =
    [[FOpCodes.qRIL, FOpCodes.qRIG],[FOpCodes.qRILL, FOpCodes.qRIGL]];
  WilOp: ARRAY PtrLength OF PACKED ARRAY BOOLEAN OF BYTE =
    [[FOpCodes.qWIL, FOpCodes.qWIG],[FOpCodes.qWILL, FOpCodes.qWIGL]];

  Cptrload: PROCEDURE [r: BDOIndex] =
    BEGIN -- loads a type-bo BDOItem onto the stack
    OPEN FOpCodes;
    s,v, bv: CARDINAL;
    pl: CARDINAL;
    tlex: se Lexeme;
    rr: BDOIndex;
    l: ContextLevel ← cb[r].base.level;
    nb: CARDINAL;

    pl ← cb[r].base.size/wordlength;
    v ← cb[r].offset.posn.wd; s ← cb[r].offset.size;
    bv ← cb[r].base.posn.wd;
    IF v IN OpCodeParams.HB
        AND s = wordlength
        AND ((l = lG AND bv IN OpCodeParams.GlobalHB)
          OR (l = CPtr.curctxlvl AND bv IN OpCodeParams.LocalHB))
        AND pl IN [1..2] THEN
      BEGIN
      Ciout2[RilOp[pl][l=lG], bv, v];
      RETURN;
      END;

    pl ← MAX[pl,1];
    IF l # lTOS THEN
      BEGIN
      pushcomponent[basecomponent, r];
      END;
    IF s = 2*wordlength THEN
      BEGIN Ciout1[ReadOp[double][pl], v]; END
    ELSE IF s > wordlength THEN
      BEGIN
      tlex ← gentemplex[pl];
      sCassign[tlex.lexsei];
      UNTIL s=0 DO
        rr ← maketempaddrBDOItem[lpushlex[tlex]];
        nb ← MIN[s,2*wordlength];
        cb[rr].offset ←
          BDOComponent[level: ,posn: FullBitAddress[wd: v, bd: 0], size: nb];
        Cptrload[rr];
        v ← v+2; s ← s-nb;
        ENDLOOP;
      END
    ELSE IF s = wordlength THEN Ciout1[ReadOp[single][pl], v]
    ELSE  Ciout2[ReadOp[field][pl], v, FieldParam[r]];
    releaseBDOItem[r];
    RETURN
    END;

  Cindexedptrload: PROCEDURE [r: BDOIndex] =
    BEGIN OPEN FOpCodes;
    Cindexedptrmove[r,[qRXL,qRXLL,qRXGL],Cptrload];
    END;

  Cindexedptrmove: PROCEDURE [r: BDOIndex, imoveop: PACKED ARRAY {local, locallong, globallong} OF BYTE
**, cptrmove: PROCEDURE[BDOIndex]] =
    BEGIN -- loads a type-bdo BDOItem onto the stack
    OPEN FOpCodes;
    s: CARDINAL ← cb[r].offset.size;
    v: CARDINAL ← cb[r].offset.posn.wd;
    l: ContextLevel ← cb[r].base.level;
    bv: CARDINAL;
    baseOnStack, dispOnStack: BOOLEAN;
    onstack: CARDINAL ← 0;

    IF cb[r].disp.size > wordlength THEN basedispcommute[r];
    dispOnStack ← cb[r].disp.level = lTOS;
    IF dispOnStack THEN
      onstack ← onstack+(cb[r].disp.size+wordlength-1)/wordlength;
    l ← cb[r].base.level;
    baseOnStack ← l = lTOS;
    IF baseOnStack THEN
      onstack ← onstack+(cb[r].base.size+wordlength-1)/wordlength;
    bv ← cb[r].base.posn.wd;
    IF cb[r].base.size > wordlength THEN
      BEGIN -- base long, disp unknown
      IF cb[r].disp.size > wordlength THEN
        BEGIN
        RequireStack[onstack]; -- DADD is minimal stack
        pushcomponent[basecomponent, r];
        pushcomponent[dispcomponent, r];
        Ciout0[qDADD];
        END
      ELSE IF cb[r].disp.size < wordlength THEN
        BEGIN
        RequireStack[onstack]; -- DADD is minimal stack
        IF dispOnStack AND ~baseOnStack THEN
          BEGIN
          pushcomponent[dispcomponent, r];
          Ciout1[qLI, 0];
          pushcomponent[basecomponent, r];
          END
        ELSE
          BEGIN
          pushcomponent[basecomponent, r];
          pushcomponent[dispcomponent, r];
          Ciout1[qLI, 0];
          END;
        Ciout0[qDADD];
        END
      ELSE
        BEGIN -- long base, one word disp
        pushcomponent[dispcomponent, r];
        IF ~dispOnStack THEN
          onstack ← onstack+(cb[r].disp.size+wordlength-1)/wordlength;
        IF s = wordlength AND v IN OpCodeParams.HB THEN
        IF l = lG AND bv IN OpCodeParams.GlobalHB THEN
          BEGIN
          Ciout2[imoveop[globallong], bv, v];
          releaseBDOItem[r];
          RETURN;
          END
        ELSE IF l = CPtr.curctxlvl AND bv IN OpCodeParams.LocalHB THEN
          BEGIN
          Ciout2[imoveop[locallong], bv, v];
          releaseBDOItem[r];
          RETURN;
          END;
      RequireStack[onstack]; -- DADD is minimal stack
      Ciout1[qLI, 0]; -- or however we're supposed to lengthen it
      pushcomponent[basecomponent, r];
      Ciout0[qDADD];
      END
    END
  ELSE
    BEGIN --base and disp both short
    IF cb[r].base.level # CPtr.curctxlvl OR cb[r].base.level = lG
OR cb[r].base.size # wordlength
      THEN
        BEGIN
        basedispcommute[r];
        l ← cb[r].base.level;
        bv ← cb[r].base.posn.wd;
        END;
    IF v IN OpCodeParams.HB AND l = CPtr.curctxlvl
        AND cb[r].base.size = wordlength AND s = wordlength THEN
      BEGIN
      IF cb[r].disp.level = CPtr.curctxlvl
        AND cb[r].disp.size = wordlength
          AND cb[r].disp.posn.wd < bv THEN
            BEGIN
            basedispcommute[r];
            bv ← cb[r].base.posn.wd;
            END;
      IF bv IN OpCodeParams.LocalHB THEN
        BEGIN
        pushcomponent[dispcomponent, r];
        Ciout2[imoveop[local], cb[r].base.posn.wd, cb[r].offset.posn.wd];
        releaseBDOItem[r];
        RETURN;
        END;
      END;
    pushcomponent[basecomponent, r];
    pushcomponent[dispcomponent, r];
    Ciout0[ qADD];
    END;
  cb[r].tag ← bo;
  cptrmove[r];
  RETURN
  END;


 Cstore: PUBLIC PROCEDURE [r: BDOIndex] =
  BEGIN -- generates code for lhs
  SELECT cb[r].tag FROM
    o => Cvarstore[r];
    bo => Cptrstore[r];
    bdo => Cindexedptrstore[r];
    ENDCASE => BEGIN AddressError[];releaseBDOItem[r];
          END;
  RETURN
  END;


 storelex: PUBLIC PROCEDURE [l: ContextLevel, wordoffset, nwords: CARDINAL] =
  BEGIN -- stores 1 or 2 words at lvl l, offset bitoffset, onto stack
  rr: BDOIndex ← genBDOItem[];

  cb[rr].tag ← o;
  cb[rr].offset ←
     BDOComponent[level: l, posn: FullBitAddress[wd: wordoffset, bd: 0], size: nwords*wordlength];
  Cvarstore[rr];
```

```
    RETURN
    END;


  Cvarstore: PROCEDURE [r: BDOIndex] =
    BEGIN -- stores a type-o BDOItem from stack
    OPEN FOpCodes;
    l: ContextLevel ← cb[r].offset.level;
    v: CARDINAL ← cb[r].offset.posn.wd;
    s: CARDINAL ← cb[r].offset.size;
    g: BOOLEAN ← l=lG;

    IF l = lTOS THEN BEGIN AddressError[]; releaseBDOItem[r]; RETURN END;
    IF ~g AND l # CPtr.curctxlvl THEN
      BEGIN GetFrame[r]; Cstore[r]; RETURN END;
    IF s = 2*wordlength THEN
      BEGIN
      IF g THEN Ciout1[qSGD, v] ELSE Ciout1[qSLD, v];
      releaseBDOItem[r];
      RETURN
      END;
    IF s > wordlength THEN
      BEGIN
      s ← s/wordlength;
      v ← cb[r].offset.posn.wd + s;
      THROUGH [1..s/2] DO
        v ← v-2; storelex[l, v, 2]; s ← s-2; ENDLOOP;
      IF s # 0 THEN storelex[l, v-1, 1];
      releaseBDOItem[r];
      RETURN
      END;
    IF s < wordlength THEN
      BEGIN loadaddr[r]; Cptrstore[r]; RETURN END;
    IF g THEN Ciout1[qSG, v] ELSE Ciout1[qSL, v];
    releaseBDOItem[r];
    RETURN
    END;



  Cptrstore: PROCEDURE [r: BDOIndex] =
    BEGIN -- stores a type-bo BDOItem from the stack
    OPEN FOpCodes;
    s,v, bv: CARDINAL;
    pl: CARDINAL;
    tlex: se Lexeme;
    rr: BDOIndex;
    l: ContextLevel ← cb[r].base.level;
    nb: CARDINAL;

    pl ← cb[r].base.size/wordlength;
    v ← cb[r].offset.posn.wd; s ← cb[r].offset.size;
    bv ← cb[r].base.posn.wd;

    IF v IN OpCodeParams.HB
        AND s = wordlength
        AND ((l = lG AND bv IN OpCodeParams.GlobalHB)
          OR (l = CPtr.curctxlvl AND bv IN OpCodeParams.LocalHB))
        AND pl IN [1..2] THEN
      BEGIN
      Ciout2[WilOp[pl][l=lG], bv, v];
      RETURN;
      END;

    pl ← MAX[pl,1];
    IF l # lTOS THEN
      BEGIN
      pushcomponent[basecomponent, r];
      END;
    IF s = 2*wordlength THEN
      BEGIN Ciout1[WriteOp[double][pl], v]; END
    ELSE IF s > wordlength THEN
      BEGIN
      tlex ← gentemplex[pl];
      v ← v+s/wordlength;
      sCassign[tlex.lexsei];
      UNTIL s=0 DO
        rr ← maketempaddrBDOItem[lpushlex[tlex]];
        nb ← MIN[s,2*wordlength];
        v ← v-nb/wordlength;
        cb[rr].offset ←
          BDOComponent[level: ,posn: FullBitAddress[wd: v, bd: 0], size: nb];
        Cptrstore[rr];
        s ← s-nb;
        ENDLOOP;
      END
    ELSE IF s = wordlength THEN Ciout1[WriteOp[single][pl], v]
    ELSE  Ciout2[WriteOp[field][pl], v, FieldParam[r]];
    releaseBDOItem[r];
    RETURN
    END;


  Cindexedptrstore: PROCEDURE [r: BDOIndex] =
    BEGIN OPEN FOpCodes;
    Cindexedptrmove[r,[qWXL,qWXLL,qWXGL],Cptrstore];
    END;


  Cindex: PUBLIC PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
    BEGIN -- generates code for array indexing
    r, rr: BDOIndex;
    s: CARDINAL ← wordlength * SymTabDefs.WordsForType[(tb+node).info];
    offset: BDOComponent;
    la: bdo Lexeme;
    alpha: INTEGER;
    onstack, simple: BOOLEAN;
    arraytype: CSEIndex;
    treeinserted, suminserted, packed: BOOLEAN ← FALSE;
    t2: TreeLink;
    mwcOffset, psize: CARDINAL;
    freet2: PROCEDURE =
      BEGIN
      WITH t2 SELECT FROM
        subtree =>
          BEGIN (tb+index).son1 ← TreeDefs.empty;
          TreeDefs.freenode[index];
          END;
      ENDCASE => P5ADefs.P5Error[322];
      END;

    t2 ← (tb+node).son2;
    arraytype ← operandtype[(tb+node).son1];
    WITH a:(seb+arraytype) SELECT FROM
      array =>
        IF a.packed AND
          SymTabDefs.BitsForType[a.componenttype] <= 8 THEN
            packed ← TRUE;
      ENDCASE;
    BEGIN
    la ← makeBDOItem[Cexp[(tb+node).son1
      !MWConstant--[cOffset]-- =>
        IF packed OR s # wordlength THEN
          RESUME[gentemplex[SymTabDefs.WordsForType[arraytype]]]
        ELSE BEGIN mwcOffset ← cOffset; GO TO useRFC END]];
    EXITS useRFC =>
      BEGIN -- can't get here if store or t2 constant
      r ← makeTOSaddrBDOItem[wordlength];  -- ignoring the base
      cb[r].offset.posn.wd ← mwcOffset;
      [t2, treeinserted] ← checkadditivity[t2, r];
      pushrhs[t2];
      mwcOffset ← cb[r].offset.posn.wd;
      IF mwcOffset > LAST[BYTE] THEN
        BEGIN
        pushlitval[mwcOffset-LAST[BYTE]];
        mwcOffset ← LAST[BYTE];
        Ciout0[FOpCodes.qADD];
        END;
      Ciout2[FOpCodes.qRFC, mwcOffset,
        LOOPHOLE[ControlDefs.FieldDescriptor[offset:0, posn:0, size:wordlength]]];
      IF treeinserted THEN freet2[];
      RETURN[topostack]
      END;
    END;
    r ← la.lexbdoi;
    onstack ← cb[r].tag = o AND cb[r].offset.level = lTOS;

    IF packed THEN
      BEGIN
      IF cb[r].tag = o THEN alpha ← 0
      ELSE
        BEGIN
        alpha ← 2 * cb[r].offset.posn.wd;
        cb[r].offset ← WordZeroBDOComponent;
        END;
      psize ← loadaddress[r];
      RETURN[packedarrayelement[t2, alpha, psize>wordlength]];
      END;

    IF treeliteral[t2] THEN
      BEGIN
      IF ~ onstack THEN
        BEGIN
        cb[r].offset.size ← s;
        cb[r].offset.posn.wd ←
                cb[r].offset.posn.wd + treeliteralvalue[t2];
        RETURN [la]
        END;
      END
    ELSE
      [t2, treeinserted] ← checkadditivity[t2, r];
    BEGIN
    SELECT cb[r].tag FROM
      o => simple ← TRUE;
      bo => GO TO alreadybo;
      bdo => simple ← FALSE;
      ENDCASE;
    IF ~simple THEN
      BEGIN offset ← cb[r].offset; cb[r].offset ← WordZeroBDOComponent; END;
    psize ← loadaddress[r];
    la.lexbdoi ← r ← makeTOSaddrBDOItem[psize];
    IF ~simple THEN cb[r].offset ← offset;
    EXITS
      alreadybo => NULL;
    END;
    cb[r].offset.size ← s;
    IF onstack AND treeliteral[t2] THEN -- i.e. didn't get caught above
      BEGIN
      cb[r].offset.posn.wd ←
                cb[r].offset.posn.wd + treeliteralvalue[t2];
      RETURN [la]
      END;
    cb[r].tag ← bdo;
    rr ← rmakeBDOItem[Cexp[t2]];
    IF cb[rr].tag = o THEN
      BEGIN cb[r].disp ← cb[rr].offset; releaseBDOItem[rr] END
    ELSE BEGIN Cload[rr]; cb[r].disp ← TosBDOComponent END;
    IF treeinserted THEN freet2[];
    RETURN [la]
    END;


  checkadditivity: PROCEDURE [t: TreeLink, r: BDOIndex] RETURNS [rt: TreeLink, insertedtree: BOOLEAN] =
**
    BEGIN OPEN TreeDefs;
    node: TreeIndex;
    p: BOOLEAN;

    insertedtree ← FALSE;
    rt ← t;
    WITH t SELECT FROM
      subtree =>
        BEGIN node ← index;
        IF (p ← (tb+node).name = plus) OR (tb+node).name = minus THEN
          IF treeliteral[(tb+node).son1] THEN
            BEGIN
            cb[r].offset.posn.wd ←
                cb[r].offset.posn.wd + treeliteralvalue[(tb+node).son1];
          IF ~p THEN
            BEGIN
            mlpush[(tb+node).son2]; pushtree[uminus, 1];
            setinfo[MPtr.typeINTEGER];
            setattr[1, FALSE];  rt ← mlpop[];
            insertedtree ← TRUE;
            END
          ELSE rt ← (tb+node).son2;
          END ELSE
        IF treeliteral[(tb+node).son2]
            AND (p OR treeliteralvalue[(tb+node).son2] <= cb[r].offset.posn.wd) THEN
          BEGIN
          cb[r].offset.posn.wd ← IF p THEN
              cb[r].offset.posn.wd+treeliteralvalue[(tb+node).son2]
            ELSE cb[r].offset.posn.wd-treeliteralvalue[(tb+node).son2];
          rt ← (tb+node).son1;
          END;
      END;
    ENDCASE;
  RETURN
  END;


Cdindex: PUBLIC PROCEDURE [node: TreeIndex] RETURNS [Lexeme] =
  BEGIN -- generates code for indexing from an array descriptor
  ld: bdo Lexeme;
  r, rr: BDOIndex;
  treeinserted, suminserted: BOOLEAN ← FALSE;
  arraytype, arraydtype: CSEIndex;
  t1, t2: TreeLink;
  psize: CARDINAL;

  t1 ← (tb+node).son1;
  t2 ← (tb+node).son2;
  arraydtype ← SymTabDefs.NormalType[operandtype[t1]];
  ld ← makeBDOItem[Cexp[t1]];
  r ← ld.lexbdoi;
  IF cb[r].tag = o AND cb[r].offset.level = lTOS THEN
    CioutO[FOpCodes.qPOP];
  cb[r].offset.size ← cb[r].offset.size-wordlength;
  psize ← cb[r].offset.size;
  WITH (seb+arraydtype) SELECT FROM
    arraydesc =>
      BEGIN
      arraytype ← SymTabDefs.UnderType[describedType];
      WITH (seb+arraytype) SELECT FROM
        array => IF ~packed OR SymTabDefs.BitsForType[componenttype] > 8 THEN
          GO TO notpacked;
        ENDCASE;
      Cload[r];
      RETURN[packedarrayelement[t2, 0, psize>wordlength]];
      EXITS
        notpacked => NULL;
      END;
    ENDCASE;
  IF cb[r].tag = o THEN
   IF cb[r].offset.level = lTOS THEN
    BEGIN
    Cvarload[r];
    ld.lexbdoi ← r ← makeTOSaddrBDOItem[psize];
    END
   ELSE
    BEGIN
    cb[r].base ← cb[r].offset;
    cb[r].tag ← bo;
    cb[r].offset ← WordZeroBDOComponent;
    END
  ELSE
    BEGIN
    pushlex[ld];
    ld.lexbdoi ← r ← makeTOSaddrBDOItem[psize];
    END;
  cb[r].offset.size ← wordlength*SymTabDefs.WordsForType[(tb+node).info];
  IF treeliteral[t2] THEN
    BEGIN cb[r].offset.posn.wd ← treeliteralvalue[t2]; RETURN [ld] END
  ELSE [t2, treeinserted] ← checkadditivity[t2, r];
```

```
    rr ← rmakeBDOItem[Cexp[t2]];
    cb[r].tag ← bdo;
    IF cb[rr].tag = o THEN
      BEGIN cb[r].disp ← cb[rr].offset; releaseBDOItem[rr] END
    ELSE BEGIN Cload[rr]; cb[r].disp ← TosBDOComponent END;
    IF treeinserted THEN WITH t2 SELECT FROM
      subtree =>
        BEGIN (tb+index).son1 ← TreeDefs.empty; TreeDefs.freenode[index]; END;
      ENDCASE => P5ADefs.P5Error[323];
    RETURN [ld]
    END;


  packedarrayelement: PROCEDURE [t2: TreeLink, alpha: INTEGER, long: BOOLEAN] RETURNS [Lexeme] =
    BEGIN -- @a[0] is on stack, eval[t2]+alpha is index
    constindex: BOOLEAN;
    treeinserted, suminserted: BOOLEAN ← FALSE;
    addend: INTEGER;
    addback: INTEGER ← 0;

    constindex ← treeliteral[t2];
    IF constindex THEN
      addend ← treeliteralvalue[t2]
    ELSE [addend, t2, treeinserted] ← extractconstant[t2];
    alpha ← alpha + addend;
    IF constindex THEN
      BEGIN
      SELECT alpha FROM
        < 0 =>
          BEGIN
          pushlitval[alpha];
          alpha ← 0;
          END;
        IN BYTE => pushlitval[0];
        ENDCASE =>
          BEGIN
          addback ← alpha-LAST[BYTE];
          alpha ← LAST[BYTE];
          pushlitval[addback];
          END;
      RETURN [Lexeme[other[byte[lexalpha:alpha, long:long]]]];
      END;
    SELECT alpha FROM
      < 0 =>
        BEGIN addback ← alpha;
        alpha ← 0;
        END;
      IN BYTE => NULL;
      ENDCASE =>
        BEGIN addback ← alpha-LAST[BYTE];
        alpha ← LAST[BYTE];
        END;

    IF addback # 0 THEN
      BEGIN t2 ← putbackconstant[t2, addback]; suminserted ← TRUE; END;
    Cload[rmakeBDOItem[Cexp[t2]]];
    IF suminserted OR treeinserted THEN WITH t2 SELECT FROM
      subtree =>
        BEGIN (tb+index).son1 ← TreeDefs.empty;
        IF suminserted THEN (tb+index).son2 ← TreeDefs.empty;
        TreeDefs.freenode[index];
        END;
      ENDCASE => P5ADefs.P5Error[324];
    RETURN [Lexeme[other[byte[lexalpha:alpha, long:long]]]];
    END;

  extractconstant: PROCEDURE [t: TreeLink] RETURNS [val: INTEGER, rt: TreeLink, insertedtree: BOOLEAN]
**=
    BEGIN OPEN TreeDefs;
    node: TreeIndex;
    p: BOOLEAN;

    insertedtree ← FALSE;
    val ← 0;
    rt ← t;
    WITH t SELECT FROM
```

```
      subtree =>
        BEGIN node ← index;
        IF (p ← (tb+node).name = plus) OR (tb+node).name = minus THEN
          IF treeliteral[(tb+node).son1] THEN
            BEGIN
            val ← treeliteralvalue[(tb+node).son1];
            IF ~p THEN
              BEGIN
              mlpush[(tb+node).son2]; pushtree[uminus, 1];
              setinfo[MPtr.typeINTEGER];
              setattr[1, FALSE]; rt ← mlpop[];
              insertedtree ← TRUE;
              END
            ELSE rt ← (tb+node).son2;
            END ELSE
          IF treeliteral[(tb+node).son2] THEN
            BEGIN
            val ← treeliteralvalue[(tb+node).son2];
            IF ~p THEN val ← -val;
            rt ← (tb+node).son1;
            END;
        END;
      ENDCASE;
    RETURN
    END;

  putbackconstant: PROCEDURE [t: TreeLink, val: INTEGER] RETURNS [rt: TreeLink] =
    BEGIN OPEN TreeDefs;
    node: TreeIndex;
    lti: LTIndex;
    p: BOOLEAN ← TRUE;
    m: BOOLEAN ← val<0;
    rt ← t;
    WITH t SELECT FROM
      subtree =>
        BEGIN node ← index;
        IF (tb+node).name = uminus THEN
          BEGIN p ← FALSE;
          rt ← (tb+node).son1;
          (tb+node).son1 ← empty;
          freenode[index];
          END;
        END;
      ENDCASE;
    IF p THEN
      BEGIN
      lti ← LitDefs.FindLiteral[ABS[val]];
      mlpush[rt]; pushlittree[lti];
      pushtree[IF m THEN minus ELSE plus, 2];
      END
    ELSE
      BEGIN
      lti ← LitDefs.FindLiteral[val];
      pushlittree[lti]; mlpush[rt];
      pushtree[minus, 2];
      END;
    setinfo[MPtr.typeINTEGER];
    setattr[1, FALSE]; rt ← mlpop[];
    RETURN
    END;



  GetFrame: PUBLIC PROCEDURE [r: BDOIndex] =
    BEGIN -- gets back to frame at level 1
    l: ContextLevel ← cb[r].offset.level;
    rr: BDOIndex;
    FLoffsetFromL: BDOComponent ←
        [size: wordlength, level: CPtr.curctxlvl, posn: FullBitAddress[bd: 0, wd: framelink]];
    FLoffset: BDOComponent ←
        [size: wordlength, level: 1Z, posn: FullBitAddress[bd: 0, wd: framelink-localbase]];

    IF cb[r].tag # o THEN P5ADefs.P5Error[325];
    IF CPtr.curctxlvl = 1 THEN RETURN;
    cb[r].offset.level ← 1Z;
    cb[r].offset.posn.wd ← cb[r].offset.posn.wd-localbase;
    cb[r].tag ← bo;
    IF CPtr.curctxlvl = l+1 THEN
      BEGIN cb[r].base ← FLoffsetFromL; RETURN END;
    rr ← genBDOItem[];
    cb[rr].tag ← bo;
    cb[rr].offset ← FLoffset;
    cb[rr].base ← FLoffsetFromL;
    Cptrload[rr];
    THROUGH (1..CPtr.curctxlvl-l) DO
      rr ← genBDOItem[];
      cb[rr].tag ← bo;
      cb[rr].offset ← FLoffset;
      cb[rr].base ← TosBDOComponent;
      Cptrload[rr];
      ENDLOOP;
    cb[r].base ← TosBDOComponent;
    RETURN
    END;


  FullWordBits: PUBLIC PROCEDURE [bits: CARDINAL] RETURNS [CARDINAL] =
    BEGIN
    RETURN[((bits+wordlength-1)/wordlength) * wordlength]
    END;


  pushcomponent: PUBLIC PROCEDURE [t: BDOComponentNames, r: BDOIndex] =
    BEGIN -- pushes base, disp, or offset from lrrecord onto stack
    rr: BDOIndex ← genBDOItem[];
    tos: BDOComponent;

    SELECT t FROM
      basecomponent => cb[rr].offset ← cb[r].base;
      dispcomponent => cb[rr].offset ← cb[r].disp;
      offsetcomponent => cb[rr].offset ← cb[r].offset;
      ENDCASE;
    cb[rr].tag ← o;
    tos ← [level: lTOS, posn: FullBitAddress[0, 0], size: FullWordBits[cb[rr].offset.size]];
    Cload[rr];
    SELECT t FROM
      basecomponent => cb[r].base ← tos;
      dispcomponent => cb[r].disp ← tos;
      offsetcomponent => cb[r].offset ← tos;
      ENDCASE;
    RETURN
    END;


  basedispcommute: PROCEDURE [r: BDOIndex] =
    BEGIN -- commutes base and disp components
    rr: BDOComponent;

    rr ← cb[r].base;
    cb[r].base ← cb[r].disp;
    cb[r].disp ← rr;
    RETURN
    END;


  loadtsonchars: PUBLIC PROCEDURE [t: TreeLink, nchars: CARDINAL] =
    BEGIN -- t is an expression of type packed array, load
        -- nchars <= 4 onto stack
        -- called from Cfrel and Crel
    IF t = empty THEN
      BEGIN
      IF ~CPtr.firstcaseselread THEN
        THROUGH [1..(nchars+1)/2] DO Ciout0[FOpCodes.qPUSH]; ENDLOOP
      ELSE CPtr.firstcaseselread ← FALSE;
      RETURN;
      END;
    pushrhs[t]; -- load full words in this case;
    IF nchars MOD 2 = 1 THEN
      BEGIN
      pushlitval[177400B];
      Ciout0[FOpCodes.qAND];
      END;
    RETURN
    END;


  BDOcount: PUBLIC INTEGER;
  BDOcard: PUBLIC INTEGER;

  genBDOItem: PUBLIC PROCEDURE RETURNS [r: BDOIndex] =
    BEGIN -- returns the cb-relative index of a lrrecord
    BDOcount ← BDOcount + 1;
    r ← BDOItemlist;
    IF r # BDONull THEN BDOItemlist ← cb[r].thread
    ELSE
      BEGIN r ← CodeDefs.GetChunk[SIZE[BDOItem]]; BDOcard ← BDOcard + 1 END;
    cb[r].thread ← InUseThread;
    RETURN
    END;


  releaseBDOItem: PUBLIC PROCEDURE [r: BDOIndex] =
    BEGIN -- returns lrrecord to free pool
    BDOcount ← BDOcount - 1;
    IF cb[r].thread # InUseThread THEN
      BEGIN SIGNAL InvalidBDOItemRelease; RETURN END;
    cb[r].thread ← BDOItemlist;
    BDOItemlist ← r;
    RETURN
    END;

END...
```