```
-- file: PeepholeZ.mesa, edited by Sweet on August 2, 1978  11:07 AM

DIRECTORY
  CodeDefs: FROM "codedefs" USING [CCIndex, CCNull, ChunkBase, CodeCCIndex, JumpCCIndex],
  ComData: FROM "comdata" USING [dStar],
  FOpCodes: FROM "fopcodes" USING [qADD, qALLOC, qAND, qBCAST, qBCASTL, qBITBLT, qBLT, qBLTC, qBLTCL, q
**BLTL, qBRK, qCATCH, qDADD, qDBL, qDCOMP, qDESCB, qDESCBS, qDIV, qDST, qDSUB, qDUP, qDWDC, qEFC, qEXCH
**, qFREE, qGADRB, qINC, qIWDC, qKFCB, qLADRB, qLDIV, qLFC, qLG, qLGD, qLI, qLINKB, qLL, qLLD, qLLK, qL
**P, qLST, qLSTF, qME, qMEL, qMRE, qMREL, qMUL, qMXD, qMXDL, qMXW, qMXWL, qNEG, qNOOP, qNOTIFY, qNOTIFY
**L, qOR, qPOP, qPORTI, qPORTO, qPS, qPSD, qPSF, qPUSH, qR, qRD, qRDL, qREQUEUE, qREQUEUEL, qRET, qRF,
**qRFC, qRFL, qRFS, qRFSL, qRIG, qRIGL, qRIL, qRILL, qRL, qRR, qRSTR, qRSTRL, qRXGL, qRXL, qRXLL, qSFC,
** qSG, qSGD, qSHIFT, qSL, qSLD, qSTARTIO, qSTOP, qSUB, qW, qWD, qWDL, qWF, qWFL, qWFS, qWFSL, qWIGL, q
**WIL, qWILL, qWL, qWR, qWS, qWSD, qWSF, qWSTR, qWSTRL, qWXGL, qWXL, qWXLL, qXOR],
  Mopcodes: FROM "mopcodes" USING [zADD, zADD01, zALLOC, zAND, zBCAST, zBITBLT, zBLT, zBLTC, zBLTCL, zB
**LTL, zBRK, zCATCH, zDADD, zDBL, zDCOMP, zDESCB, zDESCBS, zDIV, zDST, zDSUB, zDUP, zDWDC, zEFCB, zEXCH
**, zFREE, zGADRB, zINC, zIWDC, zKFCB, zLADRB, zLDIV, zLFCB, zLGB, zLGDB, zLI0, zLLB, zLLDB, zLLKB, zLP
**, zLST, zLSTF, zME, zMRE, zMUL, zMXD, zMXW, zNEG, zNOTIFY, zOR, zPOP, zPORTI, zPORTO, zPUSH, zPUSHX,
**zR0, zRB, zRBL, zRD0, zRDB, zRDBL, zREQUEUE, zRET, zRF, zRFC, zRFL, zRFS, zRFSL, zRIGP, zRIGPL, zRILP
**, zRILPL, zRR, zRSTR, zRSTRL, zRXGPL, zRXLP, zRXLPL, zSFC, zSGB, zSGDB, zSHIFT, zSLB, zSLDB, zSTARTIO
**, zSTOP, zSUB, zW0, zWB, zWBL, zWD0, zWDB, zWDBL, zWF, zWFL, zWFS, zWFSL, zWIGPL, zWILP, zWILPL, zWR,
** zWS0, zWSB, zWSDB, zWSF, zWSTR, zWSTRL, zWXGPL, zWXLP, zWXLPL, zXOR],
  OpCodeParams: FROM "opcodeparams" USING [BYTE, ExternalProcBase, ExternalProcSlots, GlobalBase, Globa
**lLoadSlots, GlobalStoreSlots, LocalBase, LocalLoadSlots, LocalProcBase, LocalProcSlots, LocalStoreSlo
**ts, ReadSlots, RILSlots, WriteSlots, zEFCn, zLFCn, zLGn, zLLn, zRILn, zRn, zSGn, zSLn, zWn],
  OpTableDefs: FROM "optabledefs" USING [instlength],
  P5ADefs: FROM "p5adefs" USING [deletecell, NumberOfParams, P5Error],
  P5BDefs: FROM "p5bdefs" USING [C0, C1, C2, LoadConstant],
  PeepholeDefs: FROM "peepholedefs" USING [InitParametersC, PackPair, PeepState],
  TableDefs: FROM "tabledefs" USING [TableNotifier],
  TreeDefs: FROM "treedefs" USING [treetype];

PeepholeZ: PROGRAM
  IMPORTS MPtr: ComData, OpTableDefs, P5ADefs, P5BDefs, PeepholeDefs
  EXPORTS CodeDefs, PeepholeDefs =
  BEGIN OPEN P5ADefs, P5BDefs, PeepholeDefs, OpCodeParams, CodeDefs;

  -- imported definitions

  BYTE: TYPE = OpCodeParams.BYTE;
  qNOOP: BYTE = FOpCodes.qNOOP;
  CodeCCIndex: TYPE = CodeDefs.CodeCCIndex;
  JumpCCIndex: TYPE = CodeDefs.JumpCCIndex;


  cb: ChunkBase;                -- code base (local copy)

  PeepholeZNotify: PUBLIC TableDefs.TableNotifier =
    BEGIN  -- called by allocator whenever table area is repacked
    cb ← LOOPHOLE[base[TreeDefs.treetype]];
    RETURN
    END;

  dummy: PRIVATE PROCEDURE =
    BEGIN
    state: PeepState;
    IF FALSE THEN [] ← state;
    END;

  UnconvertedInstruction: SIGNAL [opcode: WORD] = CODE;

  cpeepz: PUBLIC PROCEDURE [start: CodeCCIndex] =
    BEGIN -- convert to real instructions (ie from qXXX to zXXX)
    OPEN Mopcodes, FOpCodes;
    next: CodeCCIndex;
    state: PeepState;

    next ← start;
    BEGIN OPEN state;
     UNTIL (c ← next) = CCNull DO
      next ← LOOPHOLE[cb[c].flink];
      WITH cb[LOOPHOLE[c,CCIndex]] SELECT FROM
        code =>
         IF ~cb[c].realinst THEN
          BEGIN
          InitParametersC[@state];
          SELECT cinst FROM
```

```
qLG =>
  BEGIN MoveVar[global, load, single, cp[1]]; deletecell[c] END;
qSG =>
  BEGIN MoveVar[global, store, single, cp[1]]; deletecell[c] END;
qLL =>
  BEGIN MoveVar[local, load, single, cp[1]]; deletecell[c] END;
qSL =>
  BEGIN MoveVar[local, store, single, cp[1]]; deletecell[c] END;
qLI => BEGIN LoadConstant[cp[1]]; deletecell[c] END;
qLGD =>
  BEGIN MoveVar[global, load, double, cp[1]]; deletecell[c] END;
qSGD =>
  BEGIN MoveVar[global, store, double, cp[1]]; deletecell[c] END;
qLLD =>
  BEGIN MoveVar[local, load, double, cp[1]]; deletecell[c] END;
qSLD =>
  BEGIN MoveVar[local, store, double, cp[1]]; deletecell[c] END;
qR => BEGIN Move[read, single, cp[1], 0]; deletecell[c] END;
qW => BEGIN Move[write, single, cp[1], 0]; deletecell[c] END;
qRL => BEGIN Move[readlong, single, cp[1], 0]; deletecell[c] END;
qWL => BEGIN Move[writelong, single, cp[1], 0]; deletecell[c] END;
qRF => BEGIN Move[read, partial, cp[1], cp[2]]; deletecell[c] END;
qWF => BEGIN Move[write, partial, cp[1], cp[2]]; deletecell[c] END;
qRFL => BEGIN Move[readlong, partial, cp[1], cp[2]]; deletecell[c] END;
qWFL => BEGIN Move[writelong, partial, cp[1], cp[2]]; deletecell[c] END;
qRFC => MakeReal[c, zRFC];
qRFS => MakeReal[c, zRFS];
qWFS => MakeReal[c, zWFS];
qRFSL => MakeReal[c, zRFSL];
qWFSL => MakeReal[c, zWFSL];
qRD => BEGIN Move[read, double, cp[1], 0]; deletecell[c] END;
qWD => BEGIN Move[write, double, cp[1], 0]; deletecell[c] END;
qRSTR => MakeReal[c, zRSTR];
qWSTR => MakeReal[c, zWSTR];
qRXL => MakeLPReal[c, zRXLP];
qWXL => MakeLPReal[c, zWXLP];
qRIG => MakeGPReal[c, zRIGP];
qRIL => IF cp[1] = LocalBase AND cp[2] IN RILSlots
  THEN BEGIN C0[zRILn+cp[2]]; deletecell[c] END
  ELSE MakeLPReal[c, zRILP];
qWIL => MakeLPReal[c, zWILP];
qRDL => BEGIN Move[readlong, double, cp[1], 0]; deletecell[c] END;
qWDL => BEGIN Move[writelong, double, cp[1], 0]; deletecell[c] END;
qRSTRL => MakeReal[c, zRSTRL];
qWSTRL => MakeReal[c, zWSTRL];
qRXGL => MakeGPReal[c, zRXGPL];
qWXGL => MakeGPReal[c, zWXGPL];
qRXLL => MakeLPReal[c, zRXLPL];
qWXLL => MakeLPReal[c, zWXLPL];
qRIGL => MakeGPReal[c, zRIGPL];
qWIGL => MakeGPReal[c, zWIGPL];
qRILL => MakeLPReal[c, zRILPL];
qWILL => MakeLPReal[c, zWILPL];
qWS => BEGIN Move[swrite, single, cp[1], 0]; deletecell[c] END;
qWSF => BEGIN Move[swrite, partial, cp[1], cp[2]]; deletecell[c] END;
qWSD => BEGIN Move[swrite, double, cp[1], 0]; deletecell[c] END;
qPS => BEGIN Move[sput, single, cp[1], 0]; deletecell[c] END;
qPSF => BEGIN Move[sput, partial, cp[1], cp[2]]; deletecell[c] END;
qPSD => BEGIN Move[sput, double, cp[1], 0]; deletecell[c] END;
qADD => MakeRealFast[c:c, slow:zADD, fast:zADD01];
qSUB => MakeReal[c, zSUB];
qDADD => MakeReal[c, zDADD];
qDSUB => MakeReal[c, zDSUB];
qDCOMP => MakeReal[c, zDCOMP];
qMUL => MakeReal[c, zMUL];
qDIV => MakeReal[c, zDIV];
qLDIV => MakeReal[c, zLDIV];
qNEG => MakeReal[c, zNEG];
qAND => MakeReal[c, zAND];
qOR => MakeReal[c, zOR];
qXOR => MakeReal[c, zXOR];
qSHIFT => MakeReal[c, zSHIFT];
qPUSH => MakeReal[c, zPUSH];
qPOP => MakeReal[c, zPOP];
qEXCH => MakeReal[c, zEXCH];
qCATCH => MakeReal[c, zCATCH];
```

```
            qEFC =>
                IF cp[1] IN ExternalProcSlots THEN
                  BEGIN C0[zEFCn+cp[1]-ExternalProcBase]; deletecell[c] END
                ELSE MakeReal[c, zEFCB];
            qLLK => MakeReal[c, zLLKB];
            qLFC =>
                IF cp[1] IN LocalProcSlots THEN
                  BEGIN C0[zLFCn+cp[1]-LocalProcBase]; deletecell[c] END
                ELSE MakeReal[c, zLFCB];
            qSFC => MakeReal[c, zSFC];
            qRET => MakeReal[c, zRET];
            qPORTO => MakeReal[c, zPORTO];
            qPORTI => MakeReal[c, zPORTI];
            qKFCB => MakeReal[c, zKFCB];
            qBLT => MakeReal[c, zBLT];
            qBLTL => MakeReal[c, zBLTL];
            qBLTC => MakeReal[c, zBLTC];
            qBLTCL => MakeReal[c, zBLTCL];
            qALLOC => MakeReal[c, zALLOC];
            qFREE => MakeReal[c, zFREE];
            qSTOP => MakeReal[c, zSTOP];
            qBITBLT => MakeReal[c, zBITBLT];
            qSTARTIO => MakeReal[c, zSTARTIO];
            qDST => MakeReal[c, zDST];
            qLST => MakeReal[c, zLST];
            qLSTF => MakeReal[c, zLSTF];
            qWR => MakeReal[c, zWR];
            qRR => MakeReal[c, zRR];
            qBRK => MakeReal[c, zBRK];
            qLINKB => BEGIN C0[zPUSHX]; deletecell[c] END;
            qLADRB => MakeReal[c, zLADRB];
            qGADRB => MakeReal[c, zGADRB];
            qINC => MakeReal[c, zINC];
            qDUP => MakeReal[c, zDUP];
            qDBL => MakeReal[c, zDBL];
            qDWDC => MakeReal[c, zDWDC];
            qIWDC => MakeReal[c, zIWDC];
            qDESCB => MakeReal[c, zDESCB];
            qDESCBS => MakeReal[c, zDESCBS];
            qLP => MakeReal[c, IF MPtr.dStar THEN zLP ELSE zLIO];
            qME, qMEL => MakeReal[c, zME];
            qMRE, qMREL => MakeReal[c, zMRE];
            qMXW, qMXWL => MakeReal[c, zMXW];
            qMXD, qMXDL => MakeReal[c, zMXD];
            qNOTIFY, qNOTIFYL => MakeReal[c, zNOTIFY];
            qBCAST, qBCASTL => MakeReal[c, zBCAST];
            qREQUEUE, qREQUEUEL => MakeReal[c, zREQUEUE];
            ENDCASE =>
              BEGIN SIGNAL UnconvertedInstruction[cinst]; deletecell[c] END;
          END;
        ENDCASE; -- of WITH
      ENDLOOP;
    END; -- of OPEN state
    RETURN
    END;

  MakeReal: PROCEDURE [c: CodeCCIndex, i: BYTE] =
    BEGIN
    IF cb[c].realinst OR NumberOfParams[cb[c].inst] # OpTableDefs.instlength[i]-1 THEN P5ADefs.P5Error[
**1025];
    cb[c].inst ← i;
    cb[c].realinst ← TRUE;
    RETURN
    END;

  MakeRealFast: PROCEDURE [c: CodeCCIndex, slow, fast: BYTE] =
    BEGIN
    IF cb[c].realinst OR NumberOfParams[cb[c].inst] # OpTableDefs.instlength[slow]-1 THEN P5ADefs.P5Err
**or[1026];
    cb[c].inst ← IF cb[c].minimalStack THEN fast ELSE slow;
    cb[c].realinst ← TRUE;
    RETURN
    END;

  MakeLPReal: PROCEDURE [c: CodeCCIndex, i: BYTE] =
    BEGIN
```

```
    IF cb[c].realinst OR NumberOfParams[cb[c].inst] # OpTableDefs.instlength[i]-1+1 THEN P5ADefs.P5Erro
**r[1027];
    C1[i, PackPair[cb[c].parameters[1]-LocalBase, cb[c].parameters[2]]];
    deletecell[c];
    RETURN
    END;

  MakeGPReal: PROCEDURE [c: CodeCCIndex, i: BYTE] =
    BEGIN
    IF cb[c].realinst OR NumberOfParams[cb[c].inst] # OpTableDefs.instlength[i]-1+1 THEN P5ADefs.P5Erro
**r[1028];
    C1[i, PackPair[cb[c].parameters[1]-GlobalBase, cb[c].parameters[2]]];
    deletecell[c];
    RETURN
    END;


  cpeep9: PROCEDURE =
    BEGIN -- find 2-instruction sequences
    RETURN
    END;

  cpeep10: PROCEDURE =
    BEGIN -- find bit-testing jumps
    RETURN
    END;

  Mdirection: TYPE = {read, write, swrite, sput, readlong, writelong};
  Mtype: TYPE = {single, double, partial};
  MVdirection: TYPE = {load, store, put};
  MVtype: TYPE = {single, double};
  MVclass: TYPE = {global, local};

  MoveB: ARRAY MVclass OF ARRAY MVtype OF
    PACKED ARRAY MVdirection[load..store] OF BYTE ← [
      [[Mopcodes.zLGB, Mopcodes.zSGB], [Mopcodes.zLGDB, Mopcodes.zSGDB]],
      [[Mopcodes.zLLB, Mopcodes.zSLB], [Mopcodes.zLLDB, Mopcodes.zSLDB]]];

  MoveVar: PROCEDURE [c: MVclass, d: MVdirection, t: MVtype, offset: WORD] =
    BEGIN -- handles LG, SG, LL, SL, LGD, SGD, LLD, SLD, PL class instructions
    OPEN Mopcodes;
    IF t = single THEN
      IF c = local THEN
        SELECT d FROM
          load => IF offset IN LocalLoadSlots THEN
            BEGIN C0[zLLn+offset-LocalBase]; RETURN END;
          store => IF offset IN LocalStoreSlots THEN
            BEGIN C0[zSLn+offset-LocalBase]; RETURN END;
          ENDCASE
      ELSE
        SELECT d FROM
          load => IF offset IN GlobalLoadSlots THEN
            BEGIN C0[zLGn+offset-GlobalBase]; RETURN END;
          store => IF offset IN GlobalStoreSlots THEN
            BEGIN C0[zSGn+offset-GlobalBase]; RETURN END;
          ENDCASE;
    IF offset ~IN BYTE THEN
      BEGIN
      C1[IF c = global THEN zGADRB ELSE zLADRB, LAST[BYTE]];
      LoadConstant[offset - LAST[BYTE]]; C0[zADD];
      IF t = single THEN IF d = load THEN C0[zR0] ELSE C0[zW0]
      ELSE IF d = load THEN C0[zRD0] ELSE C0[zWD0];
      RETURN
      END;
    C1[MoveB[c][t][d], offset];
    RETURN
    END;


  Move: PROCEDURE [d: Mdirection, t: Mtype, offset, field: WORD] =
    BEGIN -- handles R, W, RF, WF, WS, WSF, PS, PSF class instructions
    OPEN Mopcodes;

    IF d = read AND t = single AND offset IN ReadSlots THEN
      BEGIN C0[zRn+offset]; RETURN END;
    IF d = write AND t = single AND offset IN WriteSlots THEN
```

```
      BEGIN C0[zWn+offset]; RETURN END;
    IF offset ~IN BYTE THEN
      BEGIN
      LoadConstant[offset];
      IF d >= readlong THEN
        BEGIN LoadConstant[0]; C0[zDADD] END
      ELSE C0[zADD];
      offset ← 0;
      END;
    IF offset = 0 AND d < readlong THEN
      SELECT d FROM
        read => SELECT t FROM
          single => C0[zR0];
          double => C0[zRD0];
          partial => C2[zRF, 0, field];
          ENDCASE;
        write => SELECT t FROM
          single => C0[zW0];
          double => C0[zWD0];
          partial => C2[zWF, 0, field];
          ENDCASE;
        swrite, sput => SELECT t FROM
          single => C0[zWS0];
          double => C1[zWSDB, 0];
          partial => C2[zWSF, 0, field];
          ENDCASE;
        ENDCASE
    ELSE
      SELECT d FROM
        read => SELECT t FROM
          single => C1[zRB, offset];
          double => C1[zRDB, offset];
          partial => C2[zRF, offset, field];
          ENDCASE;
        write => SELECT t FROM
          single => C1[zWB, offset];
          double => C1[zWDB, offset];
          partial => C2[zWF, offset, field];
          ENDCASE;
        swrite, sput => SELECT t FROM
          single => C1[zWSB, offset];
          double => C1[zWSDB, offset];
          partial => C2[zWSF, offset, field];
          ENDCASE;
        readlong => SELECT t FROM
          single => C1[zRBL, offset];
          double => C1[zRDBL, offset];
          partial => C2[zRFL, offset, field];
          ENDCASE;
        writelong => SELECT t FROM
          single => C1[zWBL, offset];
          double => C1[zWDBL, offset];
          partial => C2[zWFL, offset, field];
          ENDCASE;
        ENDCASE;
    IF d = sput THEN C0[zPUSH];
    RETURN
    END;


  END...
```