```
-- file: PeepholeQ.mesa, edited by Sweet on August 29, 1978  11:50 AM

DIRECTORY
  Code: FROM "code" USING [CodeNotImplemented, CodePassInconsistancy, codeptr],
  CodeDefs: FROM "codedefs" USING [CCIndex, CCNull, ChunkBase, CodeCCIndex, JumpCCIndex, JumpType, NULL
**fileindex],
  ComData: FROM "comdata" USING [dStar],
  ControlDefs: FROM "controldefs" USING [codebaseOffset],
  FOpCodes: FROM "fopcodes" USING [qADD, qAND, qBCAST, qBCASTL, qBLT, qBLTC, qBLTCL, qBLTL, qDADD, qDBL
**, qDEC, qDESCB, qDESCBS, qDUP, qDWDC, qEFC, qEXCH, qFDESCBS, qGADRB, qINC, qIWDC, qKFCB, qLADRB, qLG,
** qLGD, qLI, qLINKB, qLINT, qLL, qLLD, qLLK, qLP, qME, qMEL, qMRE, qMREL, qMUL, qMXD, qMXDL, qMXW, qMX
**WL, qNEG, qNOOP, qNOTIFY, qNOTIFYL, qOR, qPOP, qPUSH, qR, qRD, qRDL, qREQUEUE, qREQUEUEL, qRF, qRFL,
**qRFS, qRFSL, qRIG, qRIGL, qRIL, qRILF, qRILL, qRL, qRSTR, qRSTRL, qRXG, qRXGL, qRXL, qRXLL, qSDIV, qS
**FC, qSG, qSGD, qSHIFT, qSL, qSLD, qSUB, qW, qWD, qWDL, qWF, qWFL, qWIG, qWIGL, qWIL, qWILL, qWL, qWS,
** qWSD, qWSF, qWSTR, qWSTRL, qWXG, qWXGL, qWXL, qWXLL, qXOR],
  InlineDefs: FROM "inlinedefs" USING [BITAND, BITSHIFT],
  OpCodeParams: FROM "opcodeparams" USING [BYTE, GlobalHB, HB, LocalBase, LocalHB],
  P5ADefs: FROM "p5adefs" USING [deletecell, PopEffect, PushEffect],
  P5BDefs: FROM "p5bdefs" USING [C0, C1, C2, LoadConstant],
  PeepholeDefs: FROM "peepholedefs" USING [cpeepz, delete2, delete3, HalfByteLocal, InitJParametersBC,
**InitParametersABC, InitParametersBC, InitParametersC, JumpPeepState, LoadInst, MC0, PeepholeUNotify,
**PeepholeZNotify, PeepState, SetRealInst, SetSourceIndex, SlidePeepState1, SlidePeepState2, UnpackFD],
**
  SDDefs: FROM "sddefs" USING [sBLTE, sBLTEC, sBLTECL, sBLTEL, sBYTBLTE, sBYTBLTEC, sBYTBLTECL, sBYTBLT
**EL, sDivSS],
  TableDefs: FROM "tabledefs" USING [TableNotifier],
  TreeDefs: FROM "treedefs" USING [treetype];

PeepholeQ: PROGRAM
  IMPORTS CPtr: Code, MPtr: ComData, P5ADefs, P5BDefs, PeepholeDefs
  EXPORTS CodeDefs, P5BDefs =
  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)

  RJump: ARRAY JumpType[JumpE..UJumpLE] OF JumpType = [
        JumpE, JumpN, JumpG, JumpLE, JumpL, JumpGE,
        UJumpG, UJumpLE, UJumpL, UJumpGE];

  dummyProc: PROCEDURE =
    BEGIN -- every 2 minutes of compile time helps
    s: PeepState;
    js: JumpPeepState;
    IF FALSE THEN [] ← s;
    IF FALSE THEN [] ← js;
    END;

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


  start: CodeCCIndex;


  Cpeephole: PUBLIC PROCEDURE [s: CCIndex] =
    BEGIN
    start ← LOOPHOLE[s];
    SetRealInst[FALSE];
    IF ~MPtr.dStar THEN RemoveLongs[];
    cpeep0[];
    cpeep1[];
    cpeep2[];
    cpeep3[];
```

```
    cpeep4[];
    cpeep5[];
    cpeep6[];
    cpeep7[];
    SetRealInst[TRUE];
    cpeepz[start];
    SetSourceIndex[NULLfileindex];
    END;

  RemoveLongs: PROCEDURE =
    BEGIN -- remove long instructions
    OPEN FOpCodes;
    next: CodeCCIndex;
    state: PeepState;
    newinst: BYTE;

    BEGIN OPEN state;
     next ← start;
     UNTIL (c ← next) = CCNull DO
      next ← LOOPHOLE[cb[c].flink];
      newinst ← qNOOP;
      WITH cb[LOOPHOLE[c,CCIndex]] SELECT FROM
        code =>
          BEGIN
          InitParametersC[@state];
          SELECT cinst FROM
            qRL => BEGIN newinst ← qR; GOTO pop0 END;
            qRDL => BEGIN newinst ← qRD; GOTO pop0 END;
            qRFL => BEGIN newinst ← qRF; GOTO pop0 END;
            qWL => BEGIN newinst ← qW; GOTO pop0 END;
            qWFL => BEGIN newinst ← qWF; GOTO pop0 END;
            qRFSL => BEGIN newinst ← qRFS; GOTO pop1 END;
            qRSTRL => BEGIN newinst ← qRSTR; GOTO pop1 END;
            qWDL => BEGIN newinst ← qWD; GOTO pop0 END;
            qWSTRL => BEGIN newinst ← qWSTR; GOTO pop1 END;
            qRXLL => newinst ← qRXL;
            qWXLL => newinst ← qWXL;
            qRXGL => newinst ← qRXG;
            qWXGL => newinst ← qWXG;
            qRILL => newinst ← qRIL;
            qWILL => newinst ← qWIL;
            qRIGL => newinst ← qRIG;
            qWIGL => newinst ← qWIG;
            qBLTCL => BEGIN newinst ← qBLTC; GOTO pop0 END;
            qBLTL => BEGIN newinst ← qBLT; InsertPOP[0]; GOTO pop2 END;
            qMEL => BEGIN newinst ← qME; GOTO pop0 END;
            qMREL => BEGIN newinst ← qMRE; InsertPOP[0]; GOTO pop1 END;
            qMXWL => BEGIN newinst ← qMXW; InsertPOP[1]; GOTO pop2 END;
            qMXDL => BEGIN newinst ← qMXD; GOTO pop0 END;
            qNOTIFYL => BEGIN newinst ← qNOTIFY; GOTO pop0 END;
            qBCASTL => BEGIN newinst ← qBCAST; GOTO pop0 END;
            qREQUEUEL => BEGIN newinst ← qREQUEUE; InsertPOP[1]; GOTO pop2 END;
            qKFCB =>
              BEGIN OPEN SDDefs;
              newp1: WORD;
              SELECT cp[1] FROM
                sBLTEL => BEGIN newp1 ← sBLTE; InsertPOP[0] END;
                sBYTBLTEL => BEGIN newp1 ← sBYTBLTE; InsertPOP[0] END;
                sBLTECL => newp1 ← sBLTEC;
                sBYTBLTECL => newp1 ← sBYTBLTEC;
                ENDCASE => GO TO notspecial;
              cb[c].parameters[1] ← newp1;
              GO TO pop2;
              EXITS notspecial => NULL;
              END;
            ENDCASE;
          EXITS
            pop0 => InsertPOP[0];
            pop1 => InsertPOP[1];
            pop2 => InsertPOP[2];
          END;
        ENDCASE; -- of WITH
      IF newinst # qNOOP THEN cb[c].inst ← newinst;
      ENDLOOP;
    END; -- of OPEN
    RETURN
```

```
    END;

  BackupCP: PROCEDURE [n: INTEGER] RETURNS [INTEGER] =
    BEGIN OPEN FOpCodes; -- back up codeptr n stack positions
    cc: CCIndex ← CPtr.codeptr;
    neteffect: INTEGER;
    WHILE (cc ← cb[cc].blink) # CCNull AND n # 0 DO
      WITH cb[cc] SELECT FROM
        code =>
          BEGIN
          neteffect ← PushEffect[inst] - PopEffect[inst];
          IF n-neteffect < 0 THEN EXIT;
          n ← n - neteffect;
          END;
        ENDCASE => EXIT;
      ENDLOOP;
    CPtr.codeptr ← cc;
    RETURN[n]
    END;

  InsertPOP: PROCEDURE [n: INTEGER] =
    BEGIN OPEN FOpCodes; -- insert (or simulate) a POP of the word at tos-n
    savecodeptr: CCIndex ← CPtr.codeptr;
    n ← BackupCP[n];
    SELECT n FROM
      0 => C0[qPOP];
      1 => BEGIN C0[qEXCH]; C0[qPOP] END;
      2 => BEGIN C0[qPOP]; C0[qEXCH]; C0[qPUSH]; C0[qEXCH]; C0[qPOP] END;
      3 => BEGIN C0[qPOP]; C0[qPOP]; C0[qEXCH]; C0[qPUSH]; C0[qEXCH]; C0[qPUSH]; C0[qEXCH]; C0[qPOP] EN
**D;
      ENDCASE => SIGNAL CPtr.CodePassInconsistancy;
    CPtr.codeptr ← savecodeptr;
    RETURN
    END;

-- from hybrid compiler
--  ConvertBLTC: PROCEDURE [longdest: BOOLEAN] =
--    BEGIN OPEN FOpCodes;
--    savecodeptr: CCIndex ← CPtr.codeptr;
--    IF BackupCP[IF longdest THEN 3 ELSE 2]#0 THEN
--      SIGNAL CPtr.CodePassInconsistancy;
--    IF longdest THEN LoadConstant[0];
--    C0[qIWDC];
--    C1[qLG, ControlDefs.codebaseOffset];
--    IF longdest THEN C0[qLP];
--    MC0[(IF longdest THEN qDADD ELSE qADD), TRUE];
--    CPtr.codeptr ← savecodeptr;
--    C0[qDWDC];
--    RETURN
--    END;

  ConvertBLTC: PROCEDURE [longdest: BOOLEAN] =
    BEGIN OPEN FOpCodes;
    savecodeptr: CCIndex ← CPtr.codeptr;
    IF BackupCP[IF longdest THEN 3 ELSE 2]#0 THEN
      SIGNAL CPtr.CodePassInconsistancy;
    IF MPtr.dStar THEN
      BEGIN
      LoadConstant[0];
      C1[qLGD, ControlDefs.codebaseOffset];
      C0[qDADD];
      END
    ELSE
      BEGIN
      C0[qIWDC];
      C1[qLG, ControlDefs.codebaseOffset];
      MC0[qADD, TRUE];
      END;
    IF MPtr.dStar AND ~longdest THEN
      BEGIN
      CPtr.codeptr ← savecodeptr;
      [] ← BackupCP[0];
      C0[qLP];
      END;
    CPtr.codeptr ← savecodeptr;
    IF ~MPtr.dStar THEN C0[qDWDC];
```

```
    RETURN
    END;

  cpeep0: PROCEDURE =
    BEGIN -- undo doubles, undo BLTC
    OPEN 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 =>
          BEGIN
          InitParametersC[@state];
          SELECT cinst FROM
            qLGD =>
              BEGIN inst ← qLG; C1[qLG, cp[1]+1]; END;
            qLLD =>
              BEGIN inst ← qLL; C1[qLL, cp[1]+1]; END;
            qBLTCL =>
              BEGIN inst ← qBLTL; ConvertBLTC[TRUE]; END;
            ENDCASE;
          END;
        ENDCASE; -- of WITH
      ENDLOOP;
    END; -- of OPEN state
    RETURN
    END;

  cpeep1: PROCEDURE =
    BEGIN -- remove POPs by modifying previous instruction
    OPEN FOpCodes;
    next, ci: CCIndex;
    state: PeepState;
    didsomething: BOOLEAN ← TRUE;
    canSlide: BOOLEAN ← FALSE;

    WHILE didsomething DO
     OPEN state;
     next ← start;
     didsomething ← FALSE;
     UNTIL (ci ← next) = CCNull DO
      next ← cb[ci].flink;
      WITH cb[ci] SELECT FROM
        code =>
          BEGIN
          didThisTime: BOOLEAN ← FALSE;
          IF canSlide THEN SlidePeepState2[@state, LOOPHOLE[ci]]
          ELSE
            BEGIN
            c ← LOOPHOLE[ci];
            InitParametersABC[@state];
            END;
          canSlide ← FALSE;
          SELECT cinst FROM
            qPOP =>
              IF POPable[binst] THEN
                BEGIN
                deletecell[b];
                deletecell[c];
                didThisTime ← TRUE;
                END
              ELSE
                SELECT binst FROM
                  qR, qRF, qRXL, qNEG, qDESCBS, qINC, qDEC =>
                    BEGIN
                    deletecell[b];
                    didThisTime ← TRUE;
                    END;
                  qDADD =>
                    IF POPable[ainst] THEN
                      BEGIN
                      delete2[a,b];
```

```
                    InsertPOP[1];
                    MC0[qADD, bmin];
                    deletecell[c];
                    didThisTime ← TRUE;
                    END;
                  qRD =>
                    BEGIN
                    cb[b].inst ← qR;
                    deletecell[c];
                    didThisTime ← TRUE;
                    END;
                  qIWDC, qDWDC =>
                    BEGIN
                    CommuteCells[b,c];
                    didThisTime ← TRUE;
                    END;
                  ENDCASE;
            ENDCASE;
          didsomething ← didsomething OR didThisTime;
          canSlide ← ~didThisTime;
          END;
        ENDCASE => canSlide ← FALSE; -- of WITH
      ENDLOOP;
    ENDLOOP;
    RETURN
    END;

  POPable: PROCEDURE [inst: BYTE] RETURNS [BOOLEAN] =
    BEGIN OPEN FOpCodes;
    RETURN[inst#qNOOP AND
      (PopEffect[inst]=0 AND PushEffect[inst]=1 OR inst = qLP OR inst = qDUP)]
    END;

  cpeep2: PROCEDURE =
    BEGIN -- expand families
    OPEN FOpCodes;
    next, ci: CCIndex;
    state: PeepState;
    canSlide: BOOLEAN ← FALSE;

    next ← start;
    BEGIN OPEN state;
     UNTIL (ci ← next) = CCNull DO
      next ← cb[ci].flink;
      WITH cb[ci] SELECT FROM
        code =>
          BEGIN
          IF canSlide THEN SlidePeepState2[@state, LOOPHOLE[ci]]
          ELSE
            BEGIN
            c ← LOOPHOLE[ci];
            InitParametersABC[@state];
            END;
          canSlide ← FALSE;
          SELECT cinst FROM
            -- expand out-of-range families
            qRIG =>
              IF cp[1] ~IN GlobalHB OR cp[2] ~IN HB THEN
                BEGIN C1[qLG, cp[1]]; C1[qR, cp[2]]; deletecell[c]; END;
            qRIL =>
              IF cp[1] ~IN LocalHB OR cp[2] ~IN HB THEN
                BEGIN C1[qLL, cp[1]]; C1[qR, cp[2]]; deletecell[c]; END;
            qRXL =>
              IF cp[1] ~IN LocalHB OR cp[2] ~IN HB THEN
                BEGIN C1[qLL, cp[1]]; MC0[qADD, cmin]; C1[qR, cp[2]]; deletecell[c]; END;
            qWXL =>
              IF cp[1] ~IN LocalHB OR cp[2] ~IN HB THEN
                BEGIN C1[qLL, cp[1]]; C0[qADD]; C1[qW, cp[2]]; deletecell[c]; END;
            qWIL =>
              IF cp[1] ~IN LocalHB OR cp[2] ~IN HB THEN
                BEGIN C1[qLL, cp[1]]; C1[qW, cp[2]]; deletecell[c]; END;
            qRXG =>
              IF TRUE THEN
                BEGIN C1[qLG, cp[1]]; MC0[qADD, cmin]; C1[qR, cp[2]]; deletecell[c]; END;
            qWXG =>
              IF TRUE THEN
```

```
    BEGIN C1[qLG, cp[1]]; C0[qADD]; C1[qW, cp[2]]; deletecell[c]; END;
qWIG =>
  IF TRUE THEN
    BEGIN C1[qLG, cp[1]]; C1[qW, cp[2]]; deletecell[c]; END;
qRIGL =>
  IF cp[1] ~IN GlobalHB OR cp[2] ~IN HB THEN
    BEGIN C1[qLGD, cp[1]]; C1[qRL, cp[2]]; deletecell[c]; END;
qRILL =>
  IF cp[1] ~IN LocalHB OR cp[2] ~IN HB THEN
    BEGIN C1[qLLD, cp[1]]; C1[qRL, cp[2]]; deletecell[c]; END;
qRXLL =>
  IF cp[1] ~IN LocalHB OR cp[2] ~IN HB THEN
    BEGIN LoadConstant[0]; C1[qLLD, cp[1]]; C0[qDADD]; C1[qRL, cp[2]]; deletecell[c]; END;
qWXLL =>
  IF cp[1] ~IN LocalHB OR cp[2] ~IN HB THEN
    BEGIN LoadConstant[0]; C1[qLLD, cp[1]]; C0[qDADD]; C1[qWL, cp[2]]; deletecell[c]; END;
qWILL =>
  IF cp[1] ~IN LocalHB OR cp[2] ~IN HB THEN
    BEGIN C1[qLLD, cp[1]]; C1[qWL, cp[2]]; deletecell[c]; END;
qRXGL =>
  IF cp[1] ~IN GlobalHB OR cp[2] ~IN HB THEN
    BEGIN LoadConstant[0]; C1[qLGD, cp[1]]; C0[qDADD]; C1[qRL, cp[2]]; deletecell[c]; END;
qWXGL =>
  IF cp[1] ~IN GlobalHB OR cp[2] ~IN HB THEN
    BEGIN LoadConstant[0]; C1[qLGD, cp[1]]; C0[qDADD]; C1[qWL, cp[2]]; deletecell[c]; END;
qWIGL =>
  IF cp[1] ~IN GlobalHB OR cp[2] ~IN HB THEN
    BEGIN C1[qLGD, cp[1]]; C1[qWL, cp[2]]; deletecell[c]; END;
qRILF =>
  IF TRUE THEN
    BEGIN C1[qLL, cp[1]]; C2[qRF, cp[2], cp[3]]; deletecell[c]; END;
qEFC, qLLK =>
  IF cp[1] ~IN BYTE THEN
    SIGNAL CPtr.CodeNotImplemented;
qLINKB =>
  IF TRUE THEN
    BEGIN
    LoadConstant[cp[1]];
    C0[qSUB]; C1[qSL, LocalBase];
    END;
qDESCBS, qDESCB, qFDESCBS =>
    BEGIN
    IF cp[1]/2 ~IN BYTE OR cp[1] MOD 2 = 0 THEN
      SIGNAL CPtr.CodeNotImplemented;
    parameters[1] ← cp[1]/2;
    IF cinst = qFDESCBS THEN
      BEGIN inst ← qDESCBS; C0[qSFC]; END;
    END;
qSDIV =>
  BEGIN
  C1[qKFCB, SDDefs.sDivSS];
  deletecell[c];
  END;
qDEC => IF cmin THEN
    BEGIN LoadConstant[-1]; MC0[qADD, TRUE]; deletecell[c] END
  ELSE BEGIN LoadConstant[1]; C0[qSUB]; deletecell[c] END;
qLINT =>
  BEGIN
  C0[qDUP];
  LoadConstant[-15];
  C0[qSHIFT];
  C0[qNEG];
  deletecell[c];
  END;
qGADRB, qLADRB =>
  IF cp[1] ~IN BYTE THEN
    BEGIN
    parameters[1] ← LAST[BYTE];
    LoadConstant[cp[1]-LAST[BYTE]]; MC0[qADD, cmin];
    END;
-- discover family members from sequences
qR =>
  IF cp[1] IN HB THEN
    SELECT binst FROM
      qADD =>
        IF HalfByteLocal[a] THEN
```

```
                    BEGIN C2[qRXL, ap[1], cp[1]]; delete3[a,b,c]; END;
                qLL =>
                  IF bp[1] IN LocalHB THEN
                    BEGIN C2[qRIL, bp[1], cp[1]]; delete2[b,c]; END;
                qLG =>
                  IF bp[1] IN GlobalHB THEN
                    BEGIN C2[qRIG, bp[1], cp[1]]; delete2[b,c]; END;
                ENDCASE;
            qW =>
              IF cp[1] IN HB THEN
                SELECT binst FROM
                  qADD =>
                    IF HalfByteLocal[a] THEN
                      BEGIN C2[qWXL, ap[1], cp[1]]; delete3[a,b,c]; END;
                  qLL =>
                    IF bp[1] IN LocalHB THEN
                      BEGIN C2[qWIL, bp[1], cp[1]]; delete2[b,c]; END;
                  ENDCASE;
            ENDCASE => canSlide ← TRUE;
          END;
        ENDCASE => canSlide ← FALSE; -- of WITH
      ENDLOOP;
    END; -- of OPEN state
    RETURN
    END;

  cpeep3: PROCEDURE =
    BEGIN -- sprinkle DUPs
    OPEN FOpCodes;
    next, ci: CCIndex;
    state: PeepState;
    canSlide: BOOLEAN ← FALSE;

    next ← start;
    BEGIN OPEN state;
     UNTIL (ci ← next) = CCNull DO
      next ← cb[ci].flink;
      WITH cb[ci] SELECT FROM
        code =>
          BEGIN
          IF canSlide THEN SlidePeepState2[@state, LOOPHOLE[ci]]
          ELSE
            BEGIN
            c ← LOOPHOLE[ci];
            InitParametersABC[@state];
            END;
          canSlide ← FALSE;
          SELECT cinst FROM
            -- replace load,load with load,DUP
            qLL, qLG, qLI =>
              IF binst = cinst AND cp[1] = bp[1] THEN
                BEGIN C0[qDUP]; deletecell[c] END;
            qRIL, qRIG, qRILL, qRIGL =>
              IF binst = cinst AND cp[1] = bp[1] AND cp[2] = bp[2] THEN
                BEGIN C0[qDUP]; deletecell[c] END;
            ENDCASE => canSlide ← TRUE;
          END;
        ENDCASE => canSlide ← FALSE; -- of WITH
      ENDLOOP;
    END; -- of OPEN state
    RETURN
    END;

  cpeep4: PROCEDURE =
    BEGIN -- PUTs and PUSHs, RF and WF to RSTR and WSTR
    OPEN FOpCodes;
    next, ci: CCIndex;
    state: PeepState;
    pos, size: [0..16);
    canSlide: BOOLEAN ← FALSE;

    next ← start;
    BEGIN OPEN state;
     UNTIL (ci ← next) = CCNull DO
      next ← cb[ci].flink;
      WITH cb[ci] SELECT FROM
```

```
      code =>
        BEGIN
        IF canSlide THEN SlidePeepState2[@state, LOOPHOLE[ci]]
        ELSE
          BEGIN
          c ← LOOPHOLE[ci];
          InitParametersABC[@state];
          END;
        canSlide ← FALSE;
        SELECT cinst FROM
          qLL =>
            IF binst = qSL AND cp[1] = bp[1] THEN
              BEGIN C0[qPUSH]; deletecell[c]; END
            ELSE GO TO Slide;
          qLG =>
            IF binst = qSG AND cp[1] = bp[1] THEN
              BEGIN C0[qPUSH]; deletecell[c]; END
            ELSE GO TO Slide;
          qRIL =>
            IF binst = qWIL AND cp[1] = bp[1] AND cp[2] = bp[2] THEN
              BEGIN C0[qPUSH]; deletecell[c] END
            ELSE GO TO Slide;
          qRILL =>
            IF binst = qWILL AND cp[1] = bp[1] AND cp[2] = bp[2] THEN
              BEGIN C0[qPUSH]; deletecell[c] END
            ELSE GO TO Slide;
          qRIGL =>
            IF binst = qWIGL AND cp[1] = bp[1] AND cp[2] = bp[2] THEN
              BEGIN C0[qPUSH]; deletecell[c] END
            ELSE GO TO Slide;
          qRF, qWF, qRFL, qWFL =>
            BEGIN
            [pos, size] ← UnpackFD[LOOPHOLE[cp[2]]];
            IF size = 8 AND cp[1] <= LAST[BYTE]/2 THEN
              SELECT pos FROM
                0, 8 =>
                  BEGIN
                  LoadConstant[0];
                  C1[(SELECT cinst FROM
                    qRF => qRSTR,
                    qWF => qWSTR,
                    qRFL => qRSTRL,
                    ENDCASE => qWSTRL), cp[1]*2+pos/8];
                  deletecell[c];
                  END;
                ENDCASE => GO TO Slide
            ELSE GO TO Slide;
            END;
          ENDCASE => GO TO Slide;
        EXITS
          Slide => canSlide ← TRUE;
        END;
      ENDCASE => canSlide ← FALSE; -- of WITH
    ENDLOOP;
  END; -- of OPEN state
  RETURN
  END;

NonWS: ARRAY [FOpCodes.qWS..FOpCodes.qWSD] OF BYTE =
                    [FOpCodes.qW, FOpCodes.qWF, FOpCodes.qWD];

cpeep5: PROCEDURE =
  BEGIN -- put doubles back, eliminate EXCH preceding commutative operator
  OPEN FOpCodes;
  next, ci: CCIndex;
  state: PeepState;
  canSlide: BOOLEAN ← FALSE;

  next ← start;
  BEGIN OPEN state;
   UNTIL (ci ← next) = CCNull DO
    next ← cb[ci].flink;
    WITH cc:cb[ci] SELECT FROM
      code =>
        BEGIN
        IF canSlide THEN SlidePeepState2[@state, LOOPHOLE[ci]]
```

```
        ELSE
          BEGIN
          c ← LOOPHOLE[ci];
          InitParametersABC[@state];
          END;
        canSlide ← FALSE;
        SELECT cinst FROM
          qLL =>
            IF binst = qLL AND cp[1] = bp[1]+1 THEN
              BEGIN cb[b].inst ← qLLD; deletecell[c]; END
            ELSE GO TO Slide;
          qSL =>
            IF binst = qSL AND cp[1] = bp[1]-1 THEN
              BEGIN cb[c].inst ← qSLD; deletecell[b]; END
            ELSE GO TO Slide;
          qLG =>
            IF binst = qLG AND cp[1] = bp[1]+1 THEN
              BEGIN cb[b].inst ← qLGD; deletecell[c]; END
            ELSE GO TO Slide;
          qSG =>
            IF binst = qSG AND cp[1] = bp[1]-1 THEN
              BEGIN cb[c].inst ← qSGD; deletecell[b]; END
            ELSE GO TO Slide;
          qADD, qMUL, qAND, qOR, qXOR =>
            IF binst = qEXCH THEN deletecell[b]
            ELSE GO TO Slide;
          qWS, qWSF, qWSD =>
            IF binst = qEXCH THEN
              BEGIN deletecell[b]; cc.inst ← NonWS[cinst]; END
            ELSE GO TO Slide;
          qEXCH =>
            IF binst = qEXCH THEN delete2[b,c]
            ELSE IF LoadInst[b] AND LoadInst[a] THEN
              BEGIN
              deletecell[c];
              CommuteCells[a,b];
              cb[a].minimalStack ← bmin;
              cb[b].minimalStack ← amin;
              END
            ELSE GO TO Slide;
          ENDCASE => GO TO Slide;
        EXITS
          Slide => canSlide ← TRUE;
        END;
      jump =>
        BEGIN
        canSlide ← FALSE;
        IF cc.jtype IN [JumpE..UJumpLE] THEN
          WITH cb[cc.blink] SELECT FROM
            code => IF ~realinst AND inst = qEXCH AND
                       ~PushFollows[LOOPHOLE[ci,JumpCCIndex]] THEN
              BEGIN deletecell[cc.blink]; cc.jtype ← RJump[cc.jtype]; END;
            ENDCASE;
        END;
      ENDCASE => canSlide ← FALSE; -- of WITH
    ENDLOOP;
  END; -- of OPEN state
  RETURN
  END;

PushFollows: PROCEDURE [c: JumpCCIndex] RETURNS [BOOLEAN] =
  BEGIN -- c is conditional jump; TRUE if PUSH follows on either branch
  next: CCIndex;
  FOR next ← cb[c].flink, cb[next].flink WHILE next # CCNull DO
    WITH cb[next] SELECT FROM
      code => IF ~realinst AND inst = FOpCodes.qPUSH THEN RETURN[TRUE]
              ELSE EXIT;
      label => NULL;
      ENDCASE => EXIT;
    ENDLOOP;
  IF (next←cb[cb[c].destlabel].flink) # CCNull THEN
    WITH cb[next] SELECT FROM
      code => IF ~realinst AND inst = FOpCodes.qPUSH THEN RETURN[TRUE];
      ENDCASE;
  RETURN[FALSE]
  END;
```

```
CommuteCells: PROCEDURE [a, b: CCIndex] =
  BEGIN
  prev, next: CCIndex;
  prev ← cb[a].blink; -- never Null
  next ← cb[b].flink;
  cb[prev].flink ← b;
  cb[b].blink ← prev;
  cb[b].flink ← a;
  cb[a].blink ← b;
  cb[a].flink ← next;
  IF next # CCNull THEN cb[next].blink ← a;
  RETURN
  END;

cpeep6: PROCEDURE =
  BEGIN -- store double/load double, INC and DEC, MUL to SHIFT etc
  OPEN FOpCodes;
  next, ci: CCIndex;
  canSlide: BOOLEAN ← FALSE;
  state: PeepState;
  negate, powerof2: BOOLEAN;
  log: CARDINAL;
  d2: PROCEDURE =
      BEGIN
      delete2[state.b, state.c];
      IF negate THEN CO[qNEG];
      RETURN
      END;

  next ← start;
  BEGIN OPEN state;
   UNTIL (ci ← next) = CCNull DO
    next ← cb[ci].flink;
    WITH cb[ci] SELECT FROM
      code =>
        BEGIN
        IF canSlide THEN SlidePeepState1[@state, LOOPHOLE[ci]]
        ELSE
          BEGIN
          c ← LOOPHOLE[ci];
          InitParametersBC[@state];
          END;
        canSlide ← FALSE;
        SELECT cinst FROM
          qLLD =>
            IF binst = qSLD AND cp[1] = bp[1] THEN
              BEGIN CO[qPUSH]; CO[qPUSH]; deletecell[c] END
            ELSE GO TO Slide;
          qLGD =>
            IF binst = qSGD AND cp[1] = bp[1] THEN
              BEGIN CO[qPUSH]; CO[qPUSH]; deletecell[c] END
            ELSE GO TO Slide;
          qADD, qSUB =>
            IF binst = qLI THEN
              BEGIN
              SELECT LOOPHOLE[bp[1], INTEGER] FROM
                0 => delete2[b,c];
                1 => IF cinst = qADD THEN
                  BEGIN cb[c].inst ← qINC; deletecell[b]; END;
                -1 => IF cinst = qSUB THEN
                  BEGIN cb[c].inst ← qINC; deletecell[b]; END;
              ENDCASE => GO TO Slide;
              END
            ELSE IF binst = qNEG THEN
              BEGIN
              cb[c].inst ← IF cinst = qADD THEN qSUB ELSE qADD;
              deletecell[b];
              END
            ELSE GO TO Slide;
          qSHIFT =>
            IF binst = qLI THEN
              SELECT bp[1] FROM
                1 => BEGIN cb[c].inst ← qDBL; deletecell[b] END;
                0 => delete2[b,c];
                ENDCASE => GO TO Slide
```

```
            ELSE GO TO Slide;
          qMUL =>
            IF binst = qLI THEN
              BEGIN
              negate ← FALSE;
              IF LOOPHOLE[bp[1], INTEGER] < 0 THEN
                BEGIN negate ← TRUE; bp[1] ← -LOOPHOLE[bp[1],INTEGER]; END;
              SELECT bp[1] FROM
                1 => d2[];
                2 => BEGIN C0[qDBL]; d2[]; END;
                3 => BEGIN C0[qDUP]; C0[qDBL]; MC0[qADD, cmin]; d2[]; END;
                4 => BEGIN C0[qDBL]; C0[qDBL]; d2[]; END;
                5 => BEGIN C0[qDUP]; C0[qDBL]; C0[qDBL]; MC0[qADD, cmin]; d2[]; END;
                6 => BEGIN C0[qDBL]; C0[qDUP]; C0[qDBL]; MC0[qADD, cmin]; d2[]; END;
                ENDCASE =>
                  BEGIN
                  [powerof2, log] ← log2[LOOPHOLE[bp[1]]];
                  IF powerof2 THEN
                    BEGIN LoadConstant[log]; C0[qSHIFT]; d2[]; END
                  ELSE GO TO Slide;
                  END;
              END;
          ENDCASE => GO TO Slide;
        EXITS
          Slide => canSlide ← TRUE;
        END;
      ENDCASE => canSlide ← FALSE; -- of WITH
    ENDLOOP;
  END; -- of OPEN state
  RETURN
  END;

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

  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;


cpeep7: PROCEDURE =
  BEGIN -- find special jumps
  OPEN FOpCodes;
  next: JumpCCIndex;
  jstate: JumpPeepState;

  next ← LOOPHOLE[start];
  BEGIN OPEN jstate;
   UNTIL (c ← next) = CCNull DO
    next ← LOOPHOLE[cb[c].flink];
    WITH cb[LOOPHOLE[c,CCIndex]] SELECT FROM
      jump =>
        BEGIN
        InitJParametersBC[@jstate];
        CPtr.codeptr ← c;
        SELECT jtype FROM
          JumpE =>
            IF binst = qLI THEN
              IF bp[1] = 0 THEN BEGIN jtype ← ZJumpE; deletecell[b] END;
          JumpN =>
            IF binst = qLI THEN
              IF bp[1] = 0 THEN BEGIN jtype ← ZJumpN; deletecell[b] END;
          ENDCASE;
        END;
      ENDCASE; -- of WITH
    ENDLOOP;
  END; -- of OPEN state
  RETURN
  END;
```

```
END...
```