```
-- file: PeepholeU.mesa, edited by Sweet on Jul 11, 1978 11:58 AM

DIRECTORY
  Code: FROM "code" USING [codeptr],
  CodeDefs: FROM "codedefs" USING [CCIndex, CCNull, ChunkBase, CodeCCIndex, CodeCCNull, JumpCCIndex],
  ControlDefs: FROM "controldefs" USING [FieldDescriptor],
  FOpCodes: FROM "fopcodes" USING [qGADRB, qLADRB, qLG, qLI, qLL, qNOOP, qRIG, qRIL, qRR],
  InlineDefs: FROM "inlinedefs" USING [BITAND, BITOR, BITSHIFT],
  Mopcodes: FROM "mopcodes" USING [zLIB, zLIN1, zLINB, zLIW],
  OpCodeParams: FROM "opcodeparams" USING [BYTE, LoadImmediateSlots, LocalHB, zLIn],
  OpTableDefs: FROM "optabledefs" USING [instaligned, instlength],
  P5ADefs: FROM "p5adefs" USING [AllocCodeCCItem, deletecell, NumberOfParams, P5Error, ParamCount],
  P5BDefs: FROM "p5bdefs",
  PeepholeDefs: FROM "peepholedefs" USING [JumpPeepState, PeepState],
  TableDefs: FROM "tabledefs" USING [TableNotifier],
  TreeDefs: FROM "treedefs" USING [treetype];

PeepholeU: PROGRAM
  IMPORTS CPtr: Code, OpTableDefs, P5ADefs
  EXPORTS CodeDefs, P5BDefs, PeepholeDefs =
  PUBLIC BEGIN OPEN P5ADefs, OpCodeParams, CodeDefs, PeepholeDefs;

  -- imported definitions

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


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

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

  sourceindex: CARDINAL;

  SetSourceIndex: PROCEDURE [c: CARDINAL] =
    BEGIN sourceindex ← c END;

  GenRealInst: BOOLEAN;

  SetRealInst: PROCEDURE [b: BOOLEAN] =
    BEGIN GenRealInst ← b END;


  HalfByteLocal: PROCEDURE [c: CCIndex] RETURNS [BOOLEAN] =
    BEGIN
    IF c = CCNull THEN RETURN [FALSE];
    WITH cb[c] SELECT FROM
      code => RETURN[inst = FOpCodes.qLL AND parameters[1] IN LocalHB];
      ENDCASE => RETURN [FALSE]
    END;

  LoadInst: PROCEDURE [c: CCIndex] RETURNS [BOOLEAN] =
    BEGIN OPEN FOpCodes;
    IF c = CCNull THEN RETURN[FALSE];
    WITH cb[c] SELECT FROM
      code => RETURN[~realinst AND (SELECT inst FROM
        qLI, qLL, qLG, qRIL, qRIG, qLADRB, qGADRB, qRR => TRUE,
        ENDCASE => FALSE)];
      ENDCASE => RETURN[FALSE]
    END;


  PackPair: PROCEDURE [l,r: [0..16)] RETURNS [w: WORD] =
    BEGIN OPEN InlineDefs;
    w ← BITSHIFT[l, 4];
    w ← BITOR[w, BITAND[r, 17B]];
    RETURN
    END;

  UnpackPair: PROCEDURE [w: WORD] RETURNS [l,r: [0..16)] =
```

```
    BEGIN OPEN InlineDefs;
    l ← BITAND[BITSHIFT[w, -4], 17B];
    r ← BITAND[w, 17B];
    RETURN
    END;

  UnpackFD: PROCEDURE [d: ControlDefs.FieldDescriptor] RETURNS [p,s: CARDINAL] =
    BEGIN
    [posn: p, size: s] ← d;
    RETURN
    END;

  abccount, bccount, sl1count, sl2count: CARDINAL ← 0;

  InitParametersABC: PROCEDURE [p: POINTER TO PeepState] =
    BEGIN -- p.c is initialized and p.c # CCNull and cb[p.c].cctag = code
    OPEN p;
    i: CARDINAL;

    abccount ← abccount+1;
    ainst ← binst ← cinst ← qNOOP;
    amin ← bmin ← cmin ← FALSE;
    a ← b ← CodeCCNull;
    IF ~(GenRealInst OR ~cb[c].realinst) THEN RETURN;
    FillInC[p];
    IF (b←LOOPHOLE[cb[c].blink, CodeCCIndex]) = CCNull THEN RETURN;
    WITH cb[LOOPHOLE[b, CCIndex]] SELECT FROM
      code =>
       IF GenRealInst OR ~realinst THEN
        BEGIN
        binst ← inst;
        bmin ← minimalStack;
        bp ← [0,0,0];
        FOR i IN [1..ParamCount[b]] DO bp[i] ← parameters[i] ENDLOOP;
        END;
      ENDCASE;
    IF (a←LOOPHOLE[cb[b].blink, CodeCCIndex]) = CCNull THEN RETURN;
    WITH cb[LOOPHOLE[a, CCIndex]] SELECT FROM
      code =>
       IF GenRealInst OR ~realinst THEN
        BEGIN
        ainst ← inst;
        amin ← minimalStack;
        ap ← [0,0,0];
        FOR i IN [1..ParamCount[a]] DO ap[i] ← parameters[i] ENDLOOP;
        END;
      ENDCASE;
    END;

  InitParametersBC: PROCEDURE [p: POINTER TO PeepState] =
    BEGIN -- p.c is initialized and p.c # CCNull and cb[p.c].cctag = code
    OPEN p;
    i: CARDINAL;

    bccount ← bccount+1;
    ainst ← binst ← cinst ← qNOOP;
    amin ← bmin ← cmin ← FALSE;
    a ← b ← CodeCCNull;
    IF ~(GenRealInst OR ~cb[c].realinst) THEN RETURN;
    FillInC[p];
    IF (b←LOOPHOLE[cb[c].blink, CodeCCIndex]) = CCNull THEN RETURN;
    WITH cb[LOOPHOLE[b, CCIndex]] SELECT FROM
      code =>
       IF GenRealInst OR ~realinst THEN
        BEGIN
        binst ← inst;
        bmin ← minimalStack;
        bp ← [0,0,0];
        FOR i IN [1..ParamCount[b]] DO bp[i] ← parameters[i] ENDLOOP;
        END;
      ENDCASE;
    END;

  InitParametersC: PROCEDURE [p: POINTER TO PeepState] =
    BEGIN -- p.c is initialized and p.c # CCNull and cb[p.c].cctag = code
    OPEN p;
```

```
    ainst ← binst ← cinst ← qNOOP;
    amin ← bmin ← cmin ← FALSE;
    a ← b ← CodeCCNull;
    IF ~(GenRealInst OR ~cb[c].realinst) THEN RETURN;
    FillInC[p];
    END;

  CondFillInC: PRIVATE PROCEDURE [p: POINTER TO PeepState] =
    BEGIN OPEN p;
    IF GenRealInst OR ~cb[c].realinst THEN FillInC[p]
    ELSE
      BEGIN
      ainst ← binst ← cinst ← qNOOP;
      amin ← bmin ← cmin ← FALSE;
      a ← b ← CodeCCNull;
      END;
    END;

  FillInC: PRIVATE PROCEDURE [p: POINTER TO PeepState] =
    BEGIN -- p.c is initialized and p.c # CCNull and cb[p.c].cctag = code
    OPEN p;
    i: CARDINAL;

    CPtr.codeptr ← c;
    sourceindex ← cb[c].sourcefileindex;
    cinst ← cb[c].inst;
    cmin ← cb[c].minimalStack;
    cp ← [0,0,0];
    FOR i IN [1..ParamCount[c]] DO cp[i] ← cb[c].parameters[i] ENDLOOP;
    END;

  InitJParametersBC: PROCEDURE [p: POINTER TO JumpPeepState] =
    BEGIN -- p.c is initialized and p.c # CCNull and cb[p.c].cctag = jump
    OPEN p;
    i: CARDINAL;

    binst ← cinst ← qNOOP;
    bmin ← cmin ← FALSE;
    b ← CodeCCNull;
    IF (b←LOOPHOLE[cb[c].blink, CodeCCIndex]) = CCNull THEN RETURN;
    WITH cb[LOOPHOLE[b, CCIndex]] SELECT FROM
      code =>
        BEGIN
        bp ← [0,0,0];
        IF ~(GenRealInst OR ~cb[b].realinst) THEN
          BEGIN
          binst ← qNOOP;
          bmin ← FALSE;
          b ← CodeCCNull;
          RETURN;
          END;
        binst ← inst;
        bmin ← minimalStack;
        FOR i IN [1..ParamCount[b]] DO bp[i] ← parameters[i] ENDLOOP;
        END;
      ENDCASE;
    END;

  SlidePeepState2: PROCEDURE [p: POINTER TO PeepState, ci: CodeCCIndex] =
    BEGIN OPEN p;
    sl2count ← sl2count+1;
    a ← b; amin ← bmin; ap ← bp; ainst ← binst;
    b ← c; bmin ← cmin; bp ← cp; binst ← cinst;
    c ← ci; CondFillInC[p];
    END;

  SlidePeepState1: PROCEDURE [p: POINTER TO PeepState, ci: CodeCCIndex] =
    BEGIN OPEN p;
    sl1count ← sl1count+1;
    b ← c; bmin ← cmin; bp ← cp; binst ← cinst;
    c ← ci; CondFillInC[p];
    END;

  LoadConstant: PROCEDURE [c: WORD] =
    BEGIN
```

```
    OPEN Mopcodes;
    ic: INTEGER;
    IF ~GenRealInst THEN
      BEGIN C1[FOpCodes.qLI, c]; RETURN END;
    ic ← LOOPHOLE[c];
    SELECT ic FROM
      IN LoadImmediateSlots => C0[zLIn+ic];
      -1 => C0[zLIN1];
      IN BYTE => C1[zLIB, ic];
      ENDCASE =>
        IF -ic IN BYTE THEN
          C1[zLINB, InlineDefs.BITAND[ic,377B]]
        ELSE C1W[zLIW, ic];
    RETURN
    END;

  MC0: PROCEDURE [i: BYTE, minimal: BOOLEAN] =
    BEGIN -- outputs an parameter-less instruction
    c: CodeCCIndex;

    IF InstParamCount[i] # 0 THEN P5ADefs.P5Error[961];
    c ← PeepAllocCodeCCItem[i,0];
    cb[c].inst ← i;
    cb[c].minimalStack ← minimal;
    RETURN
    END;


  C0: PROCEDURE [i: BYTE] =
    BEGIN -- outputs an parameter-less instruction
    c: CodeCCIndex;

    IF InstParamCount[i] # 0 THEN P5ADefs.P5Error[962];
    c ← PeepAllocCodeCCItem[i,0];
    cb[c].inst ← i;
    RETURN
    END;


  C1: PROCEDURE [i: BYTE, p1: WORD] =
    BEGIN -- outputs a one-parameter instruction
    c: CodeCCIndex;

    c ← PeepAllocCodeCCItem[i,1];
    cb[c].inst ← i;
    cb[c].parameters[1] ← p1;
    RETURN
    END;


  C1W: PROCEDURE [i: BYTE, p1: WORD] =
    BEGIN -- outputs a one-parameter(two-byte-param) instruction
    c: CodeCCIndex;

    c ← PeepAllocCodeCCItem[i,2];
    cb[c].inst ← i;
    cb[c].parameters[1] ← InlineDefs.BITSHIFT[p1, -8];
    cb[c].parameters[2] ← InlineDefs.BITAND[p1, 377B];
    RETURN
    END;


  C2: PROCEDURE [i: BYTE, p1, p2: WORD] =
    BEGIN -- outputs a two-parameter instruction
    c: CodeCCIndex;

    c ← PeepAllocCodeCCItem[i,2];
    cb[c].inst ← i;
    cb[c].parameters[1] ← p1;
    cb[c].parameters[2] ← p2;
    RETURN
    END;


  C3: PROCEDURE [i: BYTE, p1, p2, p3: WORD] =
    BEGIN -- outputs a three-parameter instruction
```

```
    c: CodeCCIndex;

    c ← PeepAllocCodeCCItem[i,3];
    cb[c].inst ← i;
    cb[c].parameters[1] ← p1;
    cb[c].parameters[2] ← p2;
    cb[c].parameters[3] ← p3;
    RETURN
    END;


  InstParamCount: PROCEDURE [i: BYTE] RETURNS [CARDINAL] =
    BEGIN
    RETURN[IF GenRealInst THEN OpTableDefs.instlength[i]-1 ELSE NumberOfParams[i]]
    END;

  PeepAllocCodeCCItem: PROCEDURE [i: BYTE, n: [0..3]] RETURNS [c: CodeCCIndex] =
    BEGIN
    IF InstParamCount[i] # n THEN P5ADefs.P5Error[963];
    c ← AllocCodeCCItem[n];
    cb[c].realinst ← GenRealInst;
    cb[c].sourcefileindex ← sourceindex;
    IF GenRealInst THEN
      BEGIN
      cb[c].isize ← n+1;
      cb[c].aligned ← OpTableDefs.instaligned[i];
      END;
    RETURN
    END;

  delete2: PROCEDURE [a,b: CCIndex] =
    BEGIN deletecell[a]; deletecell[b]; RETURN END;

  delete3: PROCEDURE [a,b,c: CCIndex] =
    BEGIN deletecell[a]; deletecell[b]; deletecell[c]; RETURN END;


  END...
```