- Final.mesa, modified by Sweet, July 5, 1978 9:28 AM

DIRECTORY
Code: FROM "code" USING [CodePassInconsistency],
CodeDefs: FROM "codedefs" USING [CCIndex, CCNul1, ChunkBase, JumpCCIndex, JumpCCNul1, JumpType, Label
**CCIndex, LabelCCNul1],
FopCodes: FROM "fopcodes",
InlineDefs: FROM "inlinedefs",
OpTableDefs: FROM "optabledefs" USING [instaligned, instlength],
P5ADefs: FROM "p5adefs" USING [deletecell],
P5BDefs: FROM "p5bdefs" USING [bindjump, codejump, Cpeephole],
TableDefs: FROM "tabledefs" USING [TableNotifier],
TreeDefs: FROM "treedefs" USING [treetype];

DEFINITIONS FROM CodeDefs;

Final: PROGRAM
IMPORTS CPtr: Code, OpTableDefs, P5ADefs, P5BDefs
EXPORTS CodeDefs, P5BDefs •
BEGIN
OPEN P5ADefs, P5BDefs;

cb: ChunkBase; -- code base (local copy)
CJump: ARRAY JumpType[JumpE .. ZJumpN] OF JumpType = [
JumpN, JumpE, JumpGE, JumpL, JumpLE, JumpG,
UJumpGE, UJumpL, UJumpLE, UJumpG, ZJumpN, ZJumpE];

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

DidSomething: BOOLEAN;
StartIndex, EndIndex: CCIndex;
SeenSwitch: BOOLEAN;
JumpCellCount: CARDINAL;

ThreadsValid: BOOLEAN;

AreThreadsValid: PUBLIC PROCEDURE RETURNS [BOOLEAN] •
BEGIN
RETURN[ThreadsValid]
END;

Cfixup: PUBLIC PROCEDURE [start: CCIndex] •
BEGIN -- a final pass over the code to fix up jumps
ThreadsValid + TRUE;
DidSomething + TRUE;
SeenSwitch + TRUE;
StartIndex + start;

--ComplementCursor[];
DO
-- pass 0: distinguish forward and backward jumps
Cpass0[];
IF ~DidSomething THEN EXIT;
DidSomething + FALSE;
SeenSwitch + SeenSwitch;
-- pass 1: convert conditional backward jumps to canonical form
--Cpass1[];
-- pass 2: eliminate multiple labels
Cpass2[];
-- pass 3: eliminate jump to jumps
Cpass3[];
-- pass 4: eliminate unreachable code
Cpass4[];
ENDLOOP; -- end of the meta-pass consisting of passes 0-4

-- pass 5: replace cj-j seq. with ccj
Cpass5[];
-- pass 6: do some peephole optimization: load-store, EXCH-commutative op.
Cpeephole[StartIndex];
-- jump threads no longer maintained, debug output take note
ThreadsValid + FALSE;
-- pass 7: set length and alignment (most already set)
CPass7[];
-- pass 8: resolve (most) jump instructions
CPass8[];
-- pass 9: resolve (remaining) jump instructions
CPass9[];
-- pass 10: set pad fields
CPass10[];
-- pass 11: code jumps
CPass11[];
-- ComplementCursor[];
RETURN
END;

-- ComplementCursor: PROCEDURE =
-- BEGIN
-- i: CARDINAL;
-- FOR i IN [431B..431B+16) DO MEMORY[i] + InlineDefs.BITNOT[MEMORY[i]] ENDLOOP;
-- RETURN
-- END;

CPass0: PROCEDURE =
BEGIN -- pass 0: distinguish forward and backward jumps
  c: CCIndex;
  JumpCellCount ~ 0;
  FOR c ~ cb[StartIndex].flink, cb[c].flink WHILE c ~ CCNull DO
    EndIndex + c;
    WITH cb[c] SELECT FROM
      label => labelseen + SeenSwitch;
      jump =>
        BEGIN
          forward ~ IF destlabel = LabelCCNull THEN TRUE
          ELSE -(cb[destlabel].labelseen + SeenSwitch);
          JumpCellCount + JumpCellCount + 1;
          END;
        ENDCASE;
        ENDOLOOP;
    RETURN
    END;

CPass2: PROCEDURE =
BEGIN -- pass 2: eliminate multiple labels
  duplabel, unreferencedlabel: LabelCCIndex;
  nextc, c: CCIndex;
  FOR c ~ cb[StartIndex].flink, nextc WHILE c ~ CCNull DO
    WITH cc:cb[c] SELECT FROM
      label =>
        IF cc.jumplist ~ JumpCCNull THEN
          BEGIN
            unreferencedlabel + LOOPHOLE[c, LabelCCIndex]; nextc + cc.flink;
            DidSomething + TRUE; deletecell[unreferencedlabel];
          END
        ELSE
          BEGIN
            duplabel + LOOPHOLE[c, LabelCCIndex]; nextc + cc.flink;
            IF cc.flink = CCNull THEN RETURN;
            WITH cb[cc.flink] SELECT FROM
              label =>
                BEGIN
                  deleteLabel[duplabel, LOOPHOLE[cc.flink, LabelCCIndex]];
                  DidSomething + TRUE;
                  END;
                ENDCASE;
            END;
          ENDCASE
          => nextc + cc.flink;
        ENDLOOP;
    RETURN
    END;

CPass3: PROCEDURE =
BEGIN -- pass 3: eliminate jump to jumps

c, cc, oldc: CCIndex;
jc: JumpCCIndex;
jtojexists: BOOLEAN;
jclabel, formerLabel: LabelCCIndex;
jcount: CARDINAL;

FOR c = cb[StartIndex].flink, cb[c].flink WHILE c # CCNull DO
  WITH cb[c] SELECT FROM
    jump =>
    IF destLabel # LabelCCNull THEN
      BEGIN
        jtojexists + FALSE;
        jcount + 0;
        jc + LOOPHOLE[c, JumpCCIndex];
      DO
        jclabel + cb[jc].destLabel;
        IF (cc + cb[jclabel].flink) = CCNull THEN EXIT;
        IF ~ UCjump[cc] THEN EXIT;
        jc + LOOPHOLE[cc, JumpCCIndex];
        IF jc = c THEN BEGIN jtojexists = FALSE; EXIT END;
        jcount + jcount + 1;
        IF jcount > JumpCellCount THEN
          BEGIN jtojexists = FALSE; EXIT END;
        IF jtype = JumpC AND ~cb[jc].forward THEN EXIT;
        jtojexists = TRUE;
      ENDLOOP;
      IF jtojexists THEN
        BEGIN
          DidSomething = TRUE;
          formerLabel = destLabel;
          unthreadJump[LOOPHOLE[c, JumpCCIndex]];
        IF jtype = JumpC AND cb[formerLabel].jumplist = JumpCCNull THEN
          BEGIN
            cc = cb[formerLabel].flink;
            deleteCell[formerLabel];
          DO
            oldc = cc;
            cc = cb[cc].flink;
            WITH cb[oldc] SELECT FROM
              label => EXIT;
              jump => unthreadJump[LOOPHOLE[oldc, JumpCCIndex]];
            ENDCASE;
            deleteCell[oldc];
          ENDLOOP;
        END;
      ENDCASE;
    END;
  ENDLOOP;
RETURN
END

CPass4: PROCEDURE =
BEGIN -- pass 4: eliminate unreachable code
  c, cc, oldc: CCIndex;

  FOR c = cb[StartIndex].flink, cb[c].flink WHILE c # CCNull DO
    WITH cb[c] SELECT FROM
      jump =>
      IF UCjump[c] OR jtype = JumpRet THEN
        BEGIN
          cc + flink;
        DO
          IF (oldc + cc) = CCNull THEN RETURN;
          cc + cb[cc].flink;
          WITH cb[oldc] SELECT FROM
            label => IF jumplist # JumpCCNull THEN EXIT;
            jump => unthreadJump[LOOPHOLE[oldc, JumpCCIndex]];
          other => EXIT;
          ENDCASE;
          deleteCell[oldc];
DidSomething \rightarrow \text{TRUE};
ENDLOOP;
END;
ENDCASE;
ENDLOOP;
RETURN
END;

CPass5: \text{PROCEDURE \rightarrow BEGIN -- pass 5: replace cj-j seq. with ccj}
c, nextc: \text{CCIndex};

\text{FOR c \rightarrow cb[\text{StartIndex}].flink, cb[c].flink WHILE c \neq \text{CCNull DO WITH oldc:cb[c] SELECT FROM jump =>}
  \text{IF oldc.jtype = JumpRet THEN BEGIN}
    nextc \rightarrow oldc.flink;
    deletecell[c];
    c \rightarrow nextc;
  \text{END ELSE IF -UCjump[c] THEN BEGIN}
    nextc \rightarrow oldc.flink;
    IF nextc = \text{CCNull THEN RETURN;}
    \text{WITH cb[nextc] SELECT FROM jump =>}
    \text{IF UCjump[nextc] AND (oldc.forward = forward) AND (cb[oldc.destlabel].blink = nextc) THEN BEGIN}
      unthreadjump[\text{LOophole[nextc, JumpCCIndex]}];
      unthreadjump[\text{LOophole[c, JumpCCIndex]}];
      oldc.destlabel \rightarrow destlabel;
      oldc.thread \rightarrow cb[oldc.destlabel].jumplist;
      cb[oldc.destlabel].jumplist \rightarrow \text{LOophole[c, JumpCCIndex]};
      oldc.jtype \rightarrow CJump[oldc.jtype];
      deletecell[nextc];
    END; ENDCASE; ENDLOOP; RETURN END;
\text{END; END; ENDCASE; ENDLOOP; RETURN END;)

CPass7: \text{PROCEDURE \rightarrow BEGIN -- pass 7: set length and alignment (most already set) c: \text{CCIndex;}}

\text{FOR c \rightarrow cb[\text{StartIndex}].flink, cb[c].flink WHILE c \neq \text{CCNull DO WITH cb[c] SELECT FROM code =>}
  \text{BEGIN}
    \text{IF isize = 0 THEN isize \rightarrow OpTableDefs.instlength[inst];}
    \text{aligned \rightarrow isize \neq 3 OR OpTableDefs.instaligned[inst];}
  \text{END; ENDCASE; ENDLOOP;}
\text{RETURN}
\text{END; END; ENDCASE; ENDLOOP; RETURN END;}

CPass8: \text{PROCEDURE \rightarrow BEGIN -- pass 8: resolve (most) jump instructions c, prev: \text{CCIndex; min, max: CARDINAL;}}

\text{FOR c \rightarrow cb[\text{StartIndex}].flink, cb[c].flink WHILE c \neq \text{CCNull DO EndIndex \rightarrow c; ENDLOOP;}}

\text{DidSomething \rightarrow \text{TRUE; WHILE DidSomething DO}
  \text{DidSomething \rightarrow \text{FALSE; FOR c \rightarrow EndIndex, prev WHILE c \neq \text{CCNull DO prev \rightarrow cb[c].blink; WITH cb[c] SELECT FROM jump =>}}}

IF ~fixedup THEN
BEGIN
[min, max] + EstimateJumpDistance[LOOPHOLE[c, JumpCCIndex]]; IF max ± 0 THEN
BEGIN
IF RemoveableJump[c] AND forward THEN
BEGIN
DidSomething ± TRUE;
END
ELSE
BEGIN
If bindjump[0, 0, LOOPHOLE[c, JumpCCIndex]] THEN
DidSomething ± TRUE;
END
ELSE
BEGIN
If bindjump[min, max, LOOPHOLE[c, JumpCCIndex]] THEN
DidSomething ± TRUE;
END
ENDCASE;
ENDLOOP;
RETURN
END;

CPass9: PROCEDURE
BEGIN -- pass 9: resolve (remaining) jump instructions
c, prev: CCIndex;
nbytes: CARDINAL;
FOR c ± EndIndex, prev WHILE c ± CCNull DO
prev ± cb[c].blink;
WITH cb[c] SELECT FROM
jump =>
IF ~fixedup THEN
BEGIN
nbytes ± EstimateJumpDistance[LOOPHOLE[c, JumpCCIndex]].max;
[] ± bindjump[nbytes, nbytes, LOOPHOLE[c, JumpCCIndex]];
END;
ENDCASE;
ENDLOOP;
RETURN
END;

CPass10: PROCEDURE
BEGIN -- pass 10: set pad field of chunks
c: CCIndex;
parity: [0..2] ± 0;
cpad: [0..1];
aligned: BOOLEAN;
t: CARDINAL;
FOR c ± cb[StartIndex].flink, cb[c].flink WHILE c ± CCNull DO
WITH cc:cb[c] SELECT FROM
code =>
BEGIN
END;
other => WITH cc SELECT FROM
table =>
BEGIN
END;
jump =>
IF cc.completed THEN BEGIN t ± 0; aligned ± FALSE END
ELSE
BEGIN
  t + cc.jsize;
  aligned + t > 1;
END;

label =>
BEGIN
  t = 0;
  aligned + FALSE;
END;
ENDCASE;
parity + (parity + t) MOD 2;
IF aligned AND parity # 0 THEN
BEGIN
  cpad + 1;
  parity + 0;
END
ELSE cpad + 0;
  cb[c].pad + cpad;
ENDLOOP:

CPassl1: PROCEDURE
BEGIN -- pass 11: code jumps
c: CCIndex:
FOR c = cb[StartIndex].flink, cb[c].flink WHILE c # CCNull DO
  WITH cb[c] SELECT FROM
    jump =>
  BEGIN
    IF ~fixedup THEN SIGNAL CPtr.CodePassInconsistency
    ELSE codejump[ComputeJumpDistance[LOOPHOLE[c, JumpCCIndex]], LOOPHOLE[c, JumpCCIndex]];
  END;
ENDCASE;
ENDLOOP;
RETURN
END;

deletelabel: PROCEDURE [oldc, c: LabelCCIndex] =
BEGIN -- removes extra label from code stream
  lq, q: JumpCCIndex;
  IF cb[c].jumplist = JumpCCNull THEN cb[c].jumplist = cb[oldc].jumplist
  ELSE
    BEGIN
      q + cb[c].jumplist;
      UNTIL q = JumpCCNull DO
        lq + q;
        q + cb[q].thread;
        ENDOLOOP;
        cb[lq].thread + cb[oldc].jumplist;
      END;
      FOR q = cb[oldc].jumplist, cb[q].thread UNTIL q = JumpCCNull DO
        cb[q].destlabel + c ENDLOOP;
        deletecell[oldc];
      RETURN
    END;
END;

unthreadjump: PROCEDURE [c: JumpCCIndex] =
BEGIN -- pull jump cell out of thread from label
  1: LabelCCIndex + cb[c].destlabel;
  jc: JumpCCIndex;
  IF 1 = LabelCCNull THEN RETURN;
  jc + cb[1].jumplist;
  IF jc = c THEN cb[1].jumplist + cb[jc].thread
  ELSE
    BEGIN
      UNTIL cb[jc].thread + c DO jc + cb[jc].thread ENDLOOP;
      cb[jc].thread + cb[c].thread;
    END;
  RETURN
END;
UCjump: PROCEDURE [c: CCIndex] RETURNS [BOOLEAN] =
BEGIN -- predicate testing if c is an unconditional jump
WITH cb[c] SELECT FROM
jump -> RETURN[jtype = Jump];
ENDCASE -> RETURN[FALSE]
END;

Removeablejump: PROCEDURE [c: CCIndex] RETURNS [BOOLEAN] =
BEGIN -- predicate testing if c is an unconditional jump
WITH cb[c] SELECT FROM
jump -> RETURN[(jtype = Jump OR jtype = JumpA OR jtype = JumpCA)];
ENDCASE -> RETURN[FALSE]
END;

dMinMax: ARRAY {unconditional, equal, relational} OF
ARRAY [0 .. 2] OF PACKED ARRAY BOOLEAN OF RECORD [min, max: [0 .. 15]]
[[(2,4), [1,4]], -- unconditional, parity 0 (backward, forward)
[(3,3), [1,3]], -- unconditional, parity 1 (backward, forward)
[(2,4), [1,4]], -- unconditional, parity 2 (backward, forward)
[(3,4), [1,4]], -- equal, parity 0 (backward, forward)
[(2,5), [1,5]], -- equal, parity 1 (backward, forward)
[(2,6), [2,6]], -- equal, parity 2 (backward, forward)
[(3,7), [3,7]], -- equal, parity 1 (backward, forward)
[(2,7), [2,7]]]; -- equal, parity 2 (backward, forward)

EstimateJumpDistance: PROCEDURE [c: JumpCCIndex] RETURNS [min, max: CARDINAL] =
BEGIN -- counts the number of bytes between a jump and its label.
label: CCIndex = cb[c].destlabel;
start, end, k: CCIndex;
t: CARDINAL;
parity: [0 .. 2] = 2;
dMin, dMax: [0 .. 15];

min + max + 0;
IF cb[c].forward THEN BEGIN start = c; end = label END
ELSE BEGIN start = label; end = c END;
FOR k = cb[start].flink, cb[k].flink UNTIL k = end DO
WITH cc:cb[k] SELECT FROM
code =>
BEGIN
  t = cc.isize;
  IF cc.isigned THEN
    BEGIN
      IF parity = 2 THEN max = max + 1
      ELSE IF (parity+t) MOD 2 # 0 THEN t = t + 1;
        parity = 0;
      END
    ELSE IF parity # 2 THEN parity = (parity+t) MOD 2;
      min = min + t;
      max = max + t;
    END;
  jump => IF cc.jtype # JumpC THEN
    BEGIN
      IF ~cc.fixedup THEN
        BEGIN
          [dMin, dMax] = dMinMax[SELECT cc.jtype FROM
            Jump, JumpA, JumpCA => unconditional,
            JumpE, JumpN => equal,
            ENDCASE => relational][parity][cc.forward];
          min = min + dMin; max = max + dMax;
        IF cc.jtype = (SELECT cc.jtype FROM
            Jump, JumpA, JumpCA => 2,
            JumpE, JumpN => IF cc.forward AND parity # 1 THEN 2 ELSE 0,
            ENDCASE => 0);
        END
      ELSE IF ~cc.completed THEN
        BEGIN
          t = cc.jsize;
          IF t = 1 THEN
            BEGIN
              IF parity # 2 THEN parity = (parity+1) MOD 2 END
            ELSE
              BEGIN
                dMin = 1;
                dMax = 0;
              END;
            IF parity = 1 THEN
              BEGIN
                dMin = 0;
                dMax = 1;
              END;
            ELSE
              BEGIN
                dMin = 1;
                dMax = 1;
              END;
            END;
          ELSE
            BEGIN
              dMin = 0;
              dMax = 1;
            END;
          END;
        END;
      END;
    END;
END.
IF parity = 2 THEN max + max + 1
ELSE IF (parity+t) MOD 2 # 0 THEN t + t + 1;
parity + 0;
END;
min + min + t;
max + max + t;
END;
END;
other => WITH cc SELECT FROM
table =>
BEGIN
t + tablecodebytes;
IF parity = 2 THEN max + max + 1
ELSE IF (parity+t) MOD 2 # 0 THEN t + t + 1;
parity + 0;
min + min + t;
max + max + t;
END;
ENDCASE;
ENDCASE;
ENDLOOP;
RETURN
END;

ComputeJumpDistance: PROCEDURE [c: JumpCCIndex] RETURNS [nbytes: CARDINAL] =
BEGIN -- counts the number of bytes between a jump and its label.
-- all jump lengths have been resolved and pad values set
label: CCIndex + cb[c].destLabel;
start,end,k: CCIndex;
nbytes + 0;
IF cb[c].forward THEN BEGIN start + c; end + label END
ELSE BEGIN start + label; end + c END;
FOR k = cb[start].flink, cb[k].flink UNTIL k = end DO
  nbytes + nbytes+cb[k].pad + (WITH cc:cb[k] SELECT FROM
code => cb[k].isize,
jump => IF cc.completed THEN 0 ELSE cc.jsize,
other => (WITH cc SELECT FROM
table => tablecodebytes,
ENDCASE => 0).
ENDCASE => 0);
ENDLOOP;
RETURN
END;

END...