```
-- file Jumps.mesa
-- last modified by Sweet, July 26, 1978  3:09 PM

DIRECTORY
    AltoDefs: FROM "altodefs" USING [BYTE],
    Code: FROM "code" USING [CodePassInconsistancy, codeptr],
    CodeDefs: FROM "codedefs" USING [ChunkBase, JumpCCIndex, JumpType],
    InlineDefs: FROM "inlinedefs",
    Mopcodes: FROM "mopcodes" USING [zJB, zJEQ4, zJEQB, zJGB, zJGEB, zJLB, zJLEB, zJNE4, zJNEB, zJUGB,
**zJUGEB, zJULB, zJULEB,
    zJW, zJZEQB, zJZNEB],
    OpCodeParams: FROM "opcodeparams" USING [zJEQn, zJn, zJNEn],
    P5ADefs: FROM "p5adefs",
    P5BDefs: FROM "p5bdefs" USING [C0, C1, C1W],
    TableDefs: FROM "tabledefs" USING [TableNotifier],
    TreeDefs: FROM "treedefs" USING [treetype];

DEFINITIONS FROM OpCodeParams, Mopcodes, CodeDefs;

Jumps: PROGRAM
        IMPORTS CPtr: Code, P5BDefs
        EXPORTS CodeDefs, P5BDefs =
    BEGIN
    OPEN P5ADefs, P5BDefs;


    -- imported definitions

    BYTE: TYPE = AltoDefs.BYTE;


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

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

    CJump: ARRAY JumpType[JumpL..ZJumpN] OF JumpType = [
                JumpGE, JumpL, JumpLE, JumpG,
                UJumpGE, UJumpL, UJumpLE, UJumpG, ZJumpN, ZJumpE];

    RelJumpOps: ARRAY JumpType[JumpL..ZJumpN] OF BYTE = [
                zJLB, zJGEB, zJGB, zJLEB, zJULB, zJUGEB, zJUGB, zJULEB,
                zJZEQB, zJZNEB];

    bindjump: PUBLIC PROCEDURE [min, max: INTEGER, c: JumpCCIndex]  RETURNS [bindable: BOOLEAN] =
        BEGIN -- compute size of jump instruction(s)
        -- max and min are counts of the number of bytes between the
        -- jump and the label.  in particular max does not allow for
        -- a possible pad on this jump (if backward).
        t: JumpType;
        js: [0..7];

        bindable ← TRUE;
        t ← cb[c].jtype;
        IF t = JumpC THEN
            BEGIN
            IF ~cb[c].forward THEN SIGNAL CPtr.CodePassInconsistancy;
            cb[c].fixedup ← TRUE;
            cb[c].completed ← TRUE;
            RETURN
            END;
        IF ~cb[c].forward THEN BEGIN max ← -max; min ← -min END;
        SELECT t FROM
                    Jump, JumpA =>
                        IF max IN [1..8] THEN js ← 1
                        ELSE IF max IN (8..127) AND min > 8 THEN js ← 2
                        ELSE IF max IN (-126..0] THEN js ← 2
                        ELSE IF min ~IN (-126..127) THEN js ← 3
                        ELSE bindable ← FALSE;
                    JumpE, JumpN =>
                        IF max IN [1..8] THEN js ← 1
                        ELSE IF max IN (8..127) AND min > 8 THEN js ← 2
                        ELSE IF max IN (-126..0] THEN js ← 2
```

```
                    ELSE IF min ~IN (-126..127) THEN js ← 4
                    ELSE bindable ← FALSE;
                JumpCA =>
                    IF max IN (-126..127) THEN js ← 2
                    ELSE IF min ~IN (-126..127) THEN js ← 3
                    ELSE bindable ← FALSE;
                ENDCASE =>
                    IF max IN (-126..127) THEN js ← 2
                    ELSE IF min ~IN (-126..127) THEN js ← 6
                    ELSE bindable ← FALSE;
        IF bindable THEN BEGIN cb[c].fixedup ← TRUE; cb[c].jsize ← js END;
        RETURN
        END;


    codejump: PUBLIC PROCEDURE [nbytes: INTEGER, c: JumpCCIndex] =
        BEGIN -- code all jump instruction(s)
        forward: BOOLEAN;
        pad: [0..1];
        t: JumpType;
        l: [0..7];

        t ← cb[c].jtype;
        CPtr.codeptr ← c;
        l ← cb[c].jsize;
        forward ← cb[c].forward;
        pad ← cb[c].pad;
        -- this statement copes with the fact that the parameter to a jump
        -- instruction is added to the byte pc of the last byte of the instruction
        -- nbytes is the number of bytes between the jump and its label
        IF ~forward THEN nbytes ← 1-nbytes-l-pad
        ELSE nbytes ← nbytes+1;
        SELECT t FROM
            Jump, JumpA, JumpCA =>
                SELECT l FROM
                    1 =>
                        BEGIN
                        IF nbytes ~IN [2..9] THEN SIGNAL CPtr.CodePassInconsistancy;
                        C0[zJn+nbytes-2];
                        END;
                    2 =>
                        BEGIN
                        IF nbytes ~IN [-128..128) THEN SIGNAL CPtr.CodePassInconsistancy;
                        C1[zJB, nbytes];
                        cb[CPtr.codeptr].pad ← pad;
                        END;
                    ENDCASE =>
                        BEGIN
                        C1W[zJW, nbytes];
                        cb[CPtr.codeptr].pad ← pad;
                        END;
            JumpE, JumpN =>
                SELECT l FROM
                    1 =>
                        BEGIN
                        IF nbytes ~IN [2..9] THEN SIGNAL CPtr.CodePassInconsistancy;
                        C0[(IF t=JumpE THEN zJEQn ELSE zJNEn)+nbytes-2];
                        END;
                    2 =>
                        BEGIN
                        IF nbytes ~IN [-128..128) THEN SIGNAL CPtr.CodePassInconsistancy;
                        C1[(IF t = JumpE THEN zJEQB ELSE zJNEB), nbytes];
                        cb[CPtr.codeptr].pad ← pad;
                        END;
                    ENDCASE =>
                        BEGIN
                        C0[(IF t = JumpE THEN zJNE4 ELSE zJEQ4)+pad];
                        C1W[zJW, nbytes]; cb[CPtr.codeptr].pad ← pad;
                        END;
            JumpC => cb[c].jbytes ← nbytes;
            ENDCASE =>
                SELECT l FROM
                    2 =>
                        BEGIN
                        IF nbytes ~IN [-128..128) THEN SIGNAL CPtr.CodePassInconsistancy;
```

```
                    C1[RelJumpOps[t], nbytes];
                    cb[CPtr.codeptr].pad ← pad;
                    END;
                ENDCASE =>
                    BEGIN
                    C1[RelJumpOps[CJump[t]], 5]; cb[CPtr.codeptr].pad ← pad;
                    C1W[zJW, nbytes]; cb[CPtr.codeptr].pad ← 1;
                    END;
        cb[c].completed ← TRUE;
        cb[c].pad ← 0; -- so it doesn't have to be ignored in ComputeJumpDistance
        cb[c].jsize ← 0; -- so it doesn't have to be ignored in ComputeJumpDistance
        RETURN
        END;


END..
```