```
-- ListCode.mesa; edited by Johnsson; July 20, 1978  12:17 PM

DIRECTORY
    AltoDefs: FROM "altodefs",
    BcdDefs: FROM "bcddefs",
    CommanderDefs: FROM "commanderdefs",
    ControlDefs: FROM "controldefs",
    InlineDefs: FROM "inlinedefs",
    IODefs: FROM "iodefs",
    ListerDefs: FROM "listerdefs",
    Mopcodes: FROM "mopcodes",
    OpTableDefs: FROM "optabledefs",
    OutputDefs: FROM "outputdefs",
    SegmentDefs: FROM "segmentdefs",
    StreamDefs: FROM "streamdefs",
    StringDefs: FROM "stringdefs",
    SymDefs: FROM "symdefs",
    SymbolTableDefs: FROM "symboltabledefs",
    TimeDefs: FROM "timedefs";

DEFINITIONS FROM OutputDefs;

ListCode: PROGRAM
  IMPORTS CommanderDefs, IODefs, ListerDefs, OpTableDefs, OutputDefs, SegmentDefs, StreamDefs,
    StringDefs, SymbolTableDefs
  EXPORTS ListerDefs SHARES SymbolTableDefs =
  BEGIN

  BYTE: TYPE = AltoDefs.BYTE;
  FileSegmentHandle: TYPE = SegmentDefs.FileSegmentHandle;
  FrameHandle: TYPE = ControlDefs.FrameHandle;
  NumberFormat: TYPE = IODefs.NumberFormat;
  opcode: TYPE = BYTE;
  PageCount: TYPE = AltoDefs.PageCount;
  WordPC: TYPE = ControlDefs.WordPC;



  JumpOp: TYPE = [Mopcodes.zJ2..Mopcodes.zJIW];
  InstWord: TYPE = MACHINE DEPENDENT RECORD[oddbyte, evenbyte: BYTE];

  offset: CARDINAL;
  codebase: POINTER;
  codepages: PageCount;
  symbols: SymbolTableDefs.SymbolTableBase;
  Tinst, Tbytes, Pinst, Pbytes, Bbytes: CARDINAL;
  freqing: BOOLEAN ← FALSE;
  absolute: BOOLEAN ← FALSE;

  -- number formats
  decimal: NumberFormat = NumberFormat[base:10, columns:1, zerofill:FALSE, unsigned:TRUE];
  decimal3: NumberFormat = NumberFormat[base:10, columns:3, zerofill:FALSE, unsigned:TRUE];
  octal3: NumberFormat = NumberFormat[base:8, columns:3, zerofill:FALSE, unsigned:TRUE];
  octal3z: NumberFormat = NumberFormat[base:8, columns:3, zerofill:TRUE, unsigned:TRUE];
  octal5: NumberFormat = NumberFormat[base:8, columns:5, zerofill:FALSE, unsigned:TRUE];
  octal6: NumberFormat = NumberFormat[base:8, columns:6, zerofill:FALSE, unsigned:TRUE];
  octal1: NumberFormat = NumberFormat[base:8, columns:1, zerofill:FALSE, unsigned:TRUE];
```

```
-- generate list of opcodes

OpcodeList: PROCEDURE [root: STRING] =
  BEGIN OPEN OpTableDefs;
  op: STRING;
  length: [0..3];
  i: opcode;
  digit: STRING = "0123456789"L;
  OpenOutput[root,".list"L];
  PutString["-- Mesa Opcodes
-- Format: name octal(decimal)push,pop,length,aligned

"L];
  FOR i IN opcode DO
    op ← InstName[i];
    IF (length ← instlength[i]) = 0 THEN op.length ← 0;
    PutString[op];
    THROUGH (op.length..8] DO PutChar[' ] ENDLOOP;
    PutNumber[i,octal3];
    PutChar['(];
    PutNumber[i,decimal3];
    PutChar[')];
    PutChar[digit[pushstack[i]]];
    PutChar[','; PutChar[digit[popstack[i]]];
    PutChar[','; PutChar[digit[length]];
    PutChar[','; PutChar[IF instaligned[i] THEN 'T ELSE 'F];
    IF i MOD 4 = 3 THEN BEGIN PutChar[';]; PutCR[] END
    ELSE PutString[";  "L];
    ENDLOOP;
  CloseOutput[];
  END;
```

```
-- source file procedures

SourceStream: StreamDefs.StreamHandle;
sourceavailable: BOOLEAN;

printsource: PROCEDURE [index: SymDefs.ByteIndex] =
  BEGIN
    OPEN symbols;
    j: SymDefs.ByteIndex;
    firstx, lastx: CARDINAL;

    IF ~sourceavailable THEN RETURN;
    firstx←fgTable[index].findex;
    lastx←LAST[CARDINAL];
    FOR j IN [0..LENGTH[fgTable]) DO
        IF j#index THEN
            IF fgTable[j].findex <= lastx AND
               fgTable[j].findex >= firstx THEN
                lastx ← fgTable[j].findex;
        ENDLOOP;
  outcheck[firstx, lastx];
  END;

outcheck: PROCEDURE [xfirst: CARDINAL, xlast: CARDINAL] =
  BEGIN OPEN StreamDefs;
    nextchar:  CHARACTER;
    lastcr: CARDINAL;
    FOR lastcr ← xfirst, lastcr-1 UNTIL lastcr = 0 DO
      SetIndex[SourceStream, [0,lastcr]];
      IF SourceStream.get[SourceStream] = IODefs.CR THEN EXIT;
      ENDLOOP;
    THROUGH (lastcr..xfirst) DO PutChar[IODefs.SP] ENDLOOP;
    SetIndex[SourceStream,StreamIndex[0,xfirst]];
    WHILE xfirst # xlast DO
      nextchar ← SourceStream.get[SourceStream ! StreamError => GOTO eof];
      xfirst ← xfirst+1;
      IF nextchar = IODefs.ControlZ THEN
        WHILE nextchar # IODefs.CR DO
          nextchar ← SourceStream.get[SourceStream ! StreamError => GOTO eof];
          xfirst ← xfirst+1;
          ENDLOOP;
      PutChar[nextchar];
      REPEAT eof => NULL;
      ENDLOOP;
  IF nextchar # IODefs.CR THEN PutChar[IODefs.CR];
  END;

setupsource: PROCEDURE =
  BEGIN OPEN SegmentDefs;
  sourceavailable ← TRUE;
  SourceStream ← StreamDefs.CreateByteStream[
    NewFile[symbols.sourceFile,Read,DefaultVersion
      ! FileNameError => BEGIN sourceavailable ← FALSE; CONTINUE END], Read];
  END;

closesource: PROCEDURE =
  BEGIN
  IF sourceavailable THEN SourceStream.destroy[SourceStream]
  END;

PrintBodyName: PROCEDURE [bti: SymDefs.BTIndex] =
  BEGIN OPEN StringDefs, SymDefs, symbols;
  sei: ISEIndex;
  hti: HTIndex;
  ss: SubStringDescriptor;

  IF sourceavailable THEN RETURN;
  WITH (bb+bti) SELECT FROM
    Callable =>
      IF (sei ← id) = SENull OR (hti ← (seb+sei).htptr) = HTNull THEN RETURN;
    ENDCASE => RETURN;
  SubStringForHash[@ss, hti];
  PutSubString[@ss];
  PutChar[':]; PutCR[];
  END;
```

```
EvenUp: PROCEDURE [n: CARDINAL] RETURNS [CARDINAL] =
  -- Round up to an even number
  BEGIN
  RETURN[n + n MOD 2];
  END;

getbyte: PROCEDURE [pc: CARDINAL] RETURNS [b: BYTE] =
  -- pc is a byte address
  BEGIN OPEN InlineDefs;
    w: POINTER TO InstWord;

    IF absolute THEN
        BEGIN
        w←LOOPHOLE[pc/2];
        b←IF BITAND[pc,1] = 0 THEN w.evenbyte ELSE w.oddbyte;
        END
    ELSE
        BEGIN
        w←codebase+pc/2;
        b←IF BITAND[pc,1] = 0 THEN w.evenbyte ELSE w.oddbyte;
        END;
  END;

getword: PROCEDURE [pc: CARDINAL] RETURNS [WORD] =
  -- pc is a word address
  BEGIN
    IF absolute THEN RETURN [MEMORY[pc]];
    RETURN[(codebase+pc)↑];
  END;


jumpaddress: PROCEDURE [jop: opcode, arg: INTEGER] RETURNS [CARDINAL] =
  BEGIN -- given a jump operator and its argument, return
        -- its target address
  OPEN Mopcodes;
  SELECT OpTableDefs.instlength[jop] FROM
    1 =>
      SELECT jop FROM
        IN [zJ2..zJ9] => arg ← jop - zJ2 + 2;
        IN [zJEQ2..zJEQ9] => arg ← jop - zJEQ2 + 2;
        IN [zJNE2..zJNE9] => arg ← jop - zJNE2 + 2;
        ENDCASE => ERROR;
    2 =>
      IF arg > 177B THEN arg ← InlineDefs.BITOR[arg,177400B];
    ENDCASE;
  RETURN[INTEGER[offset]+arg]
  END;
```

```
outwjtab: PROCEDURE [tabstart, tablength: CARDINAL, octal: BOOLEAN] =
  BEGIN
    w: INTEGER;
    pc: CARDINAL;

    Pbytes←Pbytes+tablength*2;
    FOR pc IN [tabstart..tabstart+tablength) DO
        w←getword[pc];
        PutCR[]; PutTab[]; PutTab[];
        IF octal THEN BEGIN PutTab[]; PutTab[]; END;
        PutString[" ("L];
        PutNumber[jumpaddress[Mopcodes.zJIW,w],octal5];
        PutChar[')];
      ENDLOOP;
  END;


outbjtab: PROCEDURE [tabstart, tablength: CARDINAL, octal: BOOLEAN] =
  BEGIN
    b: BYTE;
    pc: CARDINAL;

    Pbytes←Pbytes+EvenUp[tablength];
    FOR pc IN [tabstart*2..tabstart*2+tablength) DO
        b←getbyte[InlineDefs.BITXOR[pc,1]];  -- bytes "backwards"
        IF b >= 200B THEN b ← b + 177400B;  -- sign extend
        PutCR[]; PutTab[]; PutTab[];
        IF octal THEN BEGIN PutTab[]; PutTab[]; END;
        PutString[" ("L];
        PutNumber[jumpaddress[Mopcodes.zJIB,b],octal5];
        PutChar[')];
      ENDLOOP;
  END;
```

```
PutPair: PROCEDURE [byte: CARDINAL] =
  BEGIN
  a: CARDINAL = byte/16;
  b: CARDINAL = byte MOD 16;
  IF a<8 AND b<8 THEN PutChar[IODefs.SP];
  PutChar['[];
  PutNumber[a,octal1];
  PutChar[',];
  PutNumber[b, octal1];
  PutChar[']];
  RETURN
  END;

printcode: PROCEDURE [startcode, endcode: CARDINAL, octal: BOOLEAN] =
  BEGIN       -- list opcodes for indicated segment of code
  OPEN InlineDefs, Mopcodes;
  w: InstWord;
  inst, byte: BYTE;
  lastconstant, v: INTEGER;
  il: [0..3];

  FOR offset IN [startcode..endcode) DO
      inst←getbyte[offset];
      -- loginst[inst];
      Pinst←Pinst+1;
      PutTab[];
      IF octal THEN
          BEGIN
          PutNumber[offset/2,octal5];
          PutString[(IF offset MOD 2 = 0 THEN ",E " ELSE ",O ")];
          END;
      PutNumber[offset,octal5];
      PutChar[':];

      IF octal THEN
          BEGIN
          PutTab[];
          PutChar['[]; PutNumber[inst,octal3z]; PutChar[']];
          END;

      PutTab[];

      PutString[OpTableDefs.InstName[inst]];

      il ← OpTableDefs.instlength[inst];
      IF OpTableDefs.instaligned[inst] AND (offset + il) MOD 2 # 0 THEN
        BEGIN
        byte ← getbyte[offset ← offset + 1];
        IF byte = 377B THEN PutChar['*]
        ELSE
          BEGIN
          PutString[" <"L];
          PutNumber[byte,octal3];
          PutChar['>];
          END;
        Pbytes ← Pbytes + 1;
        END;
      SELECT il FROM

       0,1=>BEGIN
          Pbytes←Pbytes+1;
          IF inst IN [zLI0..zLI6] THEN
              lastconstant←inst-zLI0
          ELSE IF inst IN JumpOp THEN
                  BEGIN
                  PutTab[]; PutString["       ("L];
                  PutNumber[jumpaddress[inst,0],octal1];
                  PutChar[')];
                  END;
          END;

       2=>BEGIN
          Pbytes←Pbytes+2;
          byte←getbyte[(offset←offset+1)];
          PutTab[];
```

```
            SELECT inst FROM
              zRILP, zWILP, zRXLP, zWXLP, zRIGP,
              zRXLPL, zWXLPL, zRXGPL, zWXGPL,
              zRILPL, zWILPL, zRIGPL, zWIGPL => PutPair[byte];
              ENDCASE => PutNumber[byte,octal6];
            IF inst=zLIB THEN lastconstant←byte
            ELSE IF inst IN JumpOp THEN
                    BEGIN
                    PutString[" ("L];
                    PutNumber[jumpaddress[inst,byte],octal1];
                    PutChar[')];
                    END;
            END;

          3=>BEGIN
            Pbytes←Pbytes+3;
            w.evenbyte←getbyte[(offset←offset+1)];
            w.oddbyte←getbyte[(offset←offset+1)];
            PutTab[];

            SELECT inst FROM
              zRF, zWF, zWSF, zRFC, zRFL, zWFL =>
                BEGIN
                PutNumber[w.oddbyte,octal6];
                PutString[", "L];
                PutPair[w.evenbyte];
                END;
              ENDCASE =>
                BEGIN
                PutNumber[(v←w.oddbyte*256+w.evenbyte),octal6];
                SELECT inst FROM
                  zJIB=> outbjtab[v,lastconstant,octal];
                  zJIW=> outwjtab[v,lastconstant,octal];
                  zLIW=> lastconstant←v;
                  IN JumpOp =>
                    BEGIN
                    PutString[" ("L];
                    PutNumber[jumpaddress[inst,v],octal1];
                    PutChar[')];
                    END;
                  ENDCASE;
                END;

            END;
          ENDCASE;
        PutCR[];
        ENDLOOP;
    END;
```

```
listonebody: PROCEDURE [bti: SymDefs.CBTIndex, octal: BOOLEAN]
  RETURNS [next: SymDefs.BTIndex] =
    BEGIN OPEN SymDefs, symbols;
    fgindex, fglast: CARDINAL;
    body: POINTER TO Callable SymDefs.BodyRecord = bb+bti;
    cspp: POINTER TO ControlDefs.CSegPrefix = codebase;
    evi: POINTER TO ControlDefs.EntryVectorItem = @cspp.entry[body.entryIndex];
    endchunk: ByteIndex;
    procstart: CARDINAL = evi.initialpc*2;
    procend: CARDINAL;
    info: External BodyInfo;
    fsize: INTEGER ← evi.framesize;

    Pinst ← Pbytes ← 0;
    next ← bti +
      (WITH body SELECT FROM
          Inner => SIZE[Inner Callable BodyRecord],
          ENDCASE => SIZE[Outer Callable BodyRecord]);
    IF fsize < ControlDefs.MaxAllocSlot THEN fsize←ControlDefs.FrameVec[fsize]
    ELSE
      BEGIN
      Pbytes←Pbytes+2;
      fsize←getword[procstart/2-1];
      END;

    PutCR[];


    WITH i:body.info SELECT FROM
      External => info ← i;
      ENDCASE => ERROR;
    procend ← procstart + info.bytes;
    Bbytes ← info.bytes;
    FOR fgindex IN [info.startIndex..
      (fglast←info.startIndex+info.indexLength-1)] DO
        -- find end of this piece of code
        IF fgindex = fglast THEN endchunk ← procend
        ELSE endchunk ← fgTable[fgindex+1].cindex;

        printsource[fgindex];
        IF fgindex = info.startIndex THEN
            BEGIN
            PrintBodyName[bti];
            IF octal THEN PutTab[];
            PutString["   Frame size:  "L];
            PutNumber[fsize,decimal]; PutCR[];
            END;
        printcode[fgTable[fgindex].cindex, endchunk, octal];
        PutCR[];
        ENDLOOP;

    IF octal THEN PutTab[];
    PutString["Instructions: "L]; PutNumber[Pinst,decimal];
    PutString[", Bytes: "L]; PutNumber[Pbytes ← EvenUp[Pbytes],decimal];
    PutCR[]; PutCR[];
    Tinst ← Tinst + Pinst; Tbytes ← Tbytes + Pbytes;
  END;
```

```
ListFile: PROCEDURE [root: STRING, octal: BOOLEAN] =
  BEGIN OPEN StringDefs, SegmentDefs, symbols;
  i: CARDINAL;
  cseg,sseg: FileSegmentHandle;
  mintextindex: SymDefs.ByteIndex ← 77777B;
  bti: SymDefs.BTIndex;
  bcdFile: STRING ← [40];

  AppendString[bcdFile,root];
  FOR i IN [0..root.length) DO
    IF root[i] = '. THEN EXIT;
    REPEAT FINISHED => AppendString[bcdFile,".bcd"L];
    ENDLOOP;

  [cseg,sseg]←ListerDefs.Load[bcdFile];
  SwapIn[cseg];
  codebase ← FileSegmentAddress[cseg];
  codepages ← cseg.pages;
  symbols←SymbolTableDefs.AcquireSymbolTable[
    SymbolTableDefs.TableForSegment[sseg]];
  ListerDefs.SetRoutineSymbols[symbols];
  setupsource[];
  OpenOutput[root,".cl"L];
  ListerDefs.WriteFileID[];
  IF sourceavailable THEN
    BEGIN
    FOR i IN [0..LENGTH[fgTable]) DO
      IF fgTable[i].findex < mintextindex THEN
        mintextindex ← fgTable[i].findex;
      ENDLOOP;
    IF mintextindex # 0 THEN outcheck[0,mintextindex];
    END;

  Tbytes←Tinst←0;
  bti ← LOOPHOLE[0];
  UNTIL bti = LOOPHOLE[stHandle.bodyBlock.size, SymDefs.BTIndex] DO
    WITH (symbols.bb+bti) SELECT FROM
      Callable => bti ← listonebody[LOOPHOLE[bti], octal];
      ENDCASE => bti ← bti + SIZE[Other SymDefs.BodyRecord];
    ENDLOOP;

  SymbolTableDefs.ReleaseSymbolTable[symbols];
  DeleteFileSegment[sseg];
  Unlock[cseg]; DeleteFileSegment[cseg];
  closesource[];
  PutCR[]; IF octal THEN PutTab[];
  PutString["Total instructions: "L]; PutNumber[Tinst,decimal];
  PutString[", Bytes: "L]; PutNumber[Tbytes,decimal];
  PutCR[];
  CloseOutput[];
  END;

LCode: PROCEDURE[name: STRING, octal: BOOLEAN] =
  BEGIN OPEN ListerDefs;
  ListFile[name,octal
    !NoCode,NoFGT,NoSymbols,IncorrectVersion =>
      BEGIN IODefs.WriteString["Bad format"L]; CONTINUE END;
    SegmentDefs.FileNameError =>
      BEGIN IODefs.WriteString["File not found"L]; CONTINUE END
    ];
  END;

Code: PROCEDURE[name: STRING] =
  BEGIN
  LCode[name,FALSE];
  END;

OctalCode: PROCEDURE[name: STRING] =
  BEGIN
  LCode[name,TRUE];
  END;

Init: PROCEDURE =
  BEGIN
  command: CommanderDefs.CommandBlockHandle;
```

```
  command ← CommanderDefs.AddCommand["OpcodeList",LOOPHOLE[OpcodeList],1];
  command.params[0] ← [type: string, prompt: "Filename"];

  command ← CommanderDefs.AddCommand["OctalCode",LOOPHOLE[OctalCode],1];
  command.params[0] ← [type: string, prompt: "Filename"];

  command ← CommanderDefs.AddCommand["Code",LOOPHOLE[Code],1];
  command.params[0] ← [type: string, prompt: "Filename"];
  END;

Init[];
END. of listcode
```