```
-- file: ListCode.mesa
-- edited by Sandman September 21, 1977  1:27 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",
    OutputDefs: FROM "outputdefs",
    SegmentDefs: FROM "segmentdefs",
    StreamDefs: FROM "streamdefs",
    StringDefs: FROM "stringdefs",
    SymDefs: FROM "symdefs",
    SymbolTableDefs: FROM "symboltabledefs",
    TimeDefs: FROM "timedefs";

DEFINITIONS FROM ListerDefs, OutputDefs;

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

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



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

  offset: Address;
  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];
```

```
JItem: TYPE = RECORD [word: [0..7], byte: [0..1]];
JQuad: TYPE = RECORD [zero, one, two, three: JItem];

JTable: ARRAY [0..(LAST[JumpOp]-FIRST[JumpOp]+1)/4) OF JQuad = [
--       J1E      J2E      J3E      J4E       J1O      J2O      J3O      J4O
     [[1,0],  [2,0],  [3,0],  [4,0]],  [[1,1],  [2,1],  [3,1],  [4,1]],
--       JBE      JBO      JWE      JWO      NJBE     NJBO
     [[0,0],  [0,1],  [0,0],  [0,1]],  [[0,0],  [0,1],
--      JEQ1E    JEQ2E     JEQ3E    JEQ4E    JEQ1O    JEQ2O     JEQ3O    JEQ4O
       [1,0],   [2,0]],  [[3,0],   [4,0],   [1,1],   [2,1]],  [[3,1],   [4,1],
--      JEQBE    JEQBO
       [0,0],   [0,1]],
--      JNE1E    JNE2E    JNE3E    JNE4E     JNE1O    JNE2O    JNE3O    JNE4O
     [[1,0],  [2,0],  [3,0],  [4,0]],  [[1,1],  [2,1],  [3,1],  [4,1]],
--      JNEBE    JNEBO
     [[0,0],  [0,1],
--       JLBE     JLBO     JGEBE    JGEBO     JGBE     JGBO      JLEBE    JLEBO
       [0,0],   [0,1]],  [[0,0],  [0,1],   [0,0],   [0,1]],  [[0,0],   [0,1],
--      JULBE    JULBO    JUGEBE   JUGEBO    JUGBE    JUGBO     JULEBE   JULEBO
       [0,0],   [0,1]],  [[0,0],  [0,1],   [0,0],   [0,1]],  [[0,0],   [0,1],
--     JZEQBE   JZEQBO   JZNEBE   JZNEBO
       [0,0],  [0,1]],  [[0,0],   [0,1],
--     JDEQBE   JDEQBO    JDNEBE   JDNEBO
       [0,0],   [0,1]],  [[0,0],   [0,1],
--       JIB      JIW
       [0,0],  [0,0]]];


GetJItem: PROCEDURE [op: opcode] RETURNS [JItem] =
  BEGIN
  i: CARDINAL = (op-FIRST[JumpOp])/4;
  SELECT (op-FIRST[JumpOp]) MOD 4 FROM
    0 => RETURN[JTable[i].zero];
    1 => RETURN[JTable[i].one];
    2 => RETURN[JTable[i].two];
    3 => RETURN[JTable[i].three];
    ENDCASE;
  END;

Jword: PROCEDURE [op: opcode] RETURNS [[0..7]] =
  BEGIN
  RETURN[GetJItem[op].word]
  END;

Jbyte: PROCEDURE [op: opcode] RETURNS [[0..1]] =
  BEGIN
  RETURN[GetJItem[op].byte]
  END;

framevec: ARRAY [0..18] OF CARDINAL =
  [7,11,15,19,23,27,31,39,47,55,67,79,95,111,127,147,171,199,231];
```

```
-- generate list of opcodes

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

"];
  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]];
    IF i MOD 4 = 3 THEN PutCR[] ELSE PutString["  "];
    ENDLOOP;
  CloseOutput[];
  END;
```

```
-- source file procedures

SourceStream: StreamDefs.StreamHandle;
sourceavailable: BOOLEAN;

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

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

outcheck: PROCEDURE [xfirst: UNSIGNED, xlast: UNSIGNED] =
  BEGIN OPEN StreamDefs;
    control Z:  CHARACTER = 32C;
    nextchar:  CHARACTER;
    SetIndex[SourceStream,StreamIndex[0,xfirst]];
    WHILE xfirst # xlast DO
      IF SourceStream.endof[SourceStream] THEN RETURN;
      nextchar ← SourceStream.get[SourceStream];
      xfirst ← xfirst+1;
      IF nextchar = control Z THEN
        WHILE nextchar # IODefs.CR DO
          IF SourceStream.endof[SourceStream] THEN RETURN;
          nextchar ← SourceStream.get[SourceStream];
          xfirst ← xfirst+1;
          ENDLOOP;
        PutChar[nextchar];
        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;
  i: CARDINAL;

  IF sourceavailable OR
     (sei ← (bb+bti).id) = SENull OR
     (hti ← hashforse[sei]) = HTNull THEN RETURN;
  SubStringForHash[@ss, hti];
  FOR i IN [ss.offset..ss.offset+ss.length) DO PutChar[ss.base[i]] ENDLOOP;
  PutChar[':]; PutCR[];
  END;
```

```
EvenUp: PROCEDURE [n: CARDINAL] RETURNS [CARDINAL] =
  -- Round up to an even number
  BEGIN
  RETURN[n+InlineDefs.BITAND[n,1]];
  END;

getbyte: PROCEDURE [pc: Address] 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: Address] 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 [Address] =
  BEGIN -- given a jump operator and its argument, return
        -- its target address
  OPEN Mopcodes;
  jword, jbyte: CARDINAL;
  IF jop = zNJBE OR jop = zNJBO THEN arg←-arg;
  IF jop = zJIB OR jop = zJIW THEN
    BEGIN
    IF arg < 0 THEN BEGIN jbyte←1; jword ← -arg END
    ELSE BEGIN jbyte ← 0; jword ← arg END;
    END
  ELSE BEGIN jword ← Jword[jop] + arg; jbyte ← Jbyte[jop] END;
  RETURN [(offset/2 + jword)*2 + jbyte]
  END;
```

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

    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[" ("];
        PutNumber[jumpaddress[Mopcodes.zJIW,w],octal5];
        PutChar[')];
      ENDLOOP;
  END;


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

    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[" ("];
        PutNumber[jumpaddress[Mopcodes.zJIB,b],octal5];
        PutChar[')];
      ENDLOOP;
  END;
```

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

    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 BITAND[offset,1]=0 THEN ",E " ELSE ",O ")];
            END;
        PutNumber[offset,octal5];
        PutChar[':];

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

        PutTab[];

        PutString[instname[inst]];

        SELECT instlength[inst] FROM

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

         2=>BEGIN
            Pbytes←Pbytes+2;
            byte←getbyte[(offset←offset+1)];
            PutTab[];
            PutNumber[byte,octal6];
            IF inst=zLIB THEN lastconstant←byte
            ELSE IF inst IN [FIRST[JumpOp]..LAST[JumpOp]] THEN
                    BEGIN
                    PutString[" ("];
                    PutNumber[jumpaddress[inst,byte],octal1];
                    PutChar[')];
                    END;
            END;

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

            SELECT inst FROM
              zRF, zWF, zWSF =>
                BEGIN
                PutNumber[w.oddbyte,octal6];
                PutString[", ["];
                PutNumber[w.evenbyte/16,octal1]; PutChar[',];
                PutNumber[BITAND[w.evenbyte,17B],octal1];
                PutChar[']];
                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 [FIRST[JumpOp]..LAST[JumpOp]]=>
                    BEGIN
                    PutString[" ("];
                    PutNumber[jumpaddress[inst,v],octal1];
                    PutChar[')];
                    END;
                  ENDCASE;
                END;

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

```
BadBodyRecord: ERROR RETURNS [INTEGER];

listonebody: PROCEDURE [bti: SymDefs.BTIndex, octal: BOOLEAN]
  RETURNS [next: SymDefs.BTIndex] =
    BEGIN OPEN SymDefs, symbols;
    fgindex, fglast: CARDINAL;
    body: POINTER TO SymDefs.BodyRecord = bb+bti;
    cspp: POINTER TO ControlDefs.CsegPrefix = codebase;
    evi: POINTER TO ControlDefs.EntryVectorItem = @cspp.EntryVector[body.entryindex];
    endchunk: ByteIndex;
    procstart: Address = evi.initialpc*2;
    procend: Address;
    info: external bodyinfo;
    fsize: INTEGER ← evi.framesize;

    Pinst ← Pbytes ← 0;
    next ← bti +
      (WITH body SELECT FROM
        inner => SIZE[inner BodyRecord],
        outer => SIZE[outer BodyRecord],
        ENDCASE => ERROR BadBodyRecord);
    IF fsize < ControlDefs.maxallocslot THEN fsize←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 ← fgt[fgindex+1].cindex;

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

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

```
IncorrectVersion: PUBLIC SIGNAL = CODE;
NoFGT: PUBLIC SIGNAL = CODE;
NoCode: PUBLIC SIGNAL = CODE;
NoSymbols: PUBLIC SIGNAL = CODE;
version, creator: BcdDefs.VersionStamp;

Load: PUBLIC PROCEDURE [bcdFile: STRING] RETURNS [codeseg, symbolseg: FileSegmentHandle] =
  BEGIN OPEN SegmentDefs;
  bcdseg: FileSegmentHandle;
  bcd: POINTER TO BcdDefs.BCD;
  pages: AltoDefs.PageCount;
  codefile: FileHandle;
  sd: BcdDefs.SegDesc;

  codefile ← NewFile[bcdFile, Read, DefaultVersion];
  bcdseg ← NewFileSegment[codefile, 1, 1, Read];
  SwapIn[bcdseg];
  bcd ← FileSegmentAddress[bcdseg];
  IF (pages ← bcd.nPages) # 1 THEN BEGIN
    Unlock[bcdseg];
    MoveFileSegment[bcdseg, 1, pages];
    SwapIn[bcdseg];
    bcd ← FileSegmentAddress[bcdseg];
    END;
  BEGIN
  ENABLE
    UNWIND => BEGIN Unlock[bcdseg]; DeleteFileSegment[bcdseg] END;
  IF bcd.versionident # BcdDefs.VersionID THEN SIGNAL IncorrectVersion;
  IF bcd.definitions OR bcd.nConfigs # 0 OR bcd.nModules # 1 THEN SIGNAL NoCode;
  version ← bcd.version;
  creator ← bcd.creator;
  sd ← (LOOPHOLE[bcd, CARDINAL]+bcd.mtOffset+FIRST[BcdDefs.MTIndex]).cseg;
  codeseg ← NewFileSegment[codefile, sd.base, sd.pages, Read];
  codeseg.class ← code;
  sd ← (LOOPHOLE[bcd, CARDINAL]+bcd.mtOffset+FIRST[BcdDefs.MTIndex]).sseg;
  IF sd.pages = 0 THEN SIGNAL NoSymbols;
  IF sd.extraPages = 0 THEN SIGNAL NoFGT;
  symbolseg ← NewFileSegment[codefile, sd.base, sd.pages+sd.extraPages, Read];
  symbolseg.class ← symbols;
  END;
  Unlock[bcdseg];  DeleteFileSegment[bcdseg];
  RETURN
  END;

WriteFileID: PUBLIC PROCEDURE [name: STRING] =
  BEGIN
  PutString[name];
  PutString[" compiled "];
  PutTime[version.time];
  PutString[" by "];
  PutNumber[version.net,octal3];
  PutChar['#];
  PutNumber[version.host,octal3];
  PutChar['#];
  IF version.zapped THEN PutString[" zapped!!!"];
  PutCR[];
  PutString["  Creator "];
  PutTime[creator.time];
  PutString[" "];
  PutNumber[creator.net,octal3];
  PutChar['#];
  PutNumber[creator.host,octal3];
  PutChar['#];
  IF creator.zapped THEN PutString[" zapped!!!"];
  PutCR[]; PutCR[];
  RETURN
  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"];
    ENDLOOP;

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

  Tbytes←Tinst←0;
  bti ← LOOPHOLE[0];
  UNTIL bti = LOOPHOLE[stHandle.bodySize, SymDefs.BTIndex] DO
    bti ← listonebody[bti, octal];
    ENDLOOP;

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

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

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

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

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. of listcode
```