```
-- FindSigs.Mesa  Edited by Sandman on October 14, 1977  2:06 PM

DIRECTORY
  AltoDefs: FROM "altodefs",
  AltoFileDefs: FROM "altofiledefs",
  BcdDefs: FROM "bcddefs",
  ControlDefs: FROM "controldefs",
  ImageDefs: FROM "imagedefs",
  InlineDefs: FROM "inlinedefs",
  IODefs: FROM "iodefs",
  OutputDefs: FROM "outputdefs",
  SegmentDefs: FROM "segmentdefs",
  StreamDefs: FROM "streamdefs",
  StringDefs: FROM "stringdefs",
  SymbolTableDefs: FROM "symboltabledefs",
  SymDefs: FROM "symdefs",
  SystemDefs: FROM "systemdefs";

DEFINITIONS FROM AltoDefs, AltoFileDefs, BcdDefs, SegmentDefs;

FindSigs: PROGRAM
  IMPORTS IODefs, OutputDefs, SegmentDefs, StreamDefs, StringDefs, SymbolTableDefs,
    SystemDefs =
  BEGIN

  file: FileHandle ← NIL;

  nsigs: CARDINAL;

  SigItem: TYPE = RECORD [name: STRING, desc: CARDINAL];
  sigdata: ARRAY [0..128) OF SigItem;

  debugging: BOOLEAN ← FALSE;

  PrintSignals: PROCEDURE [name:STRING, gframe, gfi: CARDINAL] RETURNS [ngfi: CARDINAL] =
    BEGIN OPEN SymbolTableDefs, StreamDefs, SymDefs;
    tname: STRING ← [60];
    modname: STRING ← [60];
    ss: StringDefs.SubStringDescriptor;
    symseg: FileSegmentHandle;
    symbols: SymbolTableBase;
    sei: ISEIndex;
    modout: BOOLEAN ← FALSE;
    t, desc: CARDINAL;
    gfimask: CARDINAL ← 0;

    GetName: PROCEDURE [s: STRING, hti: HTIndex] =
      BEGIN OPEN symbols;
      SubStringForHash[@ss, hti];
      s.length ← 0;
      StringDefs.AppendSubString[s, @ss];
      END;

    LOOPHOLE[gfimask,ControlDefs.ProcDesc].gftindex ← gfi;

    BEGIN
    file ← NewFile[name, Read, OldFileOnly !
      FileNameError =>
        IF ~ debugging THEN
        BEGIN OPEN OutputDefs;
        PutString[name]; PutString[" (cannot be opened)"]; PutCR[];
        ngfi ← 1; -- I guess
        GO TO openerror;
        END
        ];
    EXITS
      openerror => RETURN;
    END;

    [symseg, ngfi] ← GetModule[file];
    IF symseg = NIL THEN
      BEGIN OPEN OutputDefs;
      PutString[name]; PutString[" (cannot find symbols)"]; PutCR[];
      RETURN
      END;
    symbols ← AcquireSymbolTable[TableForSegment[symseg]];
```

```
    nsigs ← 0;
    GetName[modname, symbols.hashforse[(symbols.bb+ FIRST[BTIndex]).id]];
    FOR sei ← symbols.firstctxse[symbols.stHandle.outerCtx], symbols.nextse[sei] UNTIL sei = ISENull DO
**
      OPEN id: (symbols.seb+sei);
      IF id.writeonce THEN
        WITH (symbols.seb+symbols.undertype[id.idtype]) SELECT FROM
          transfer =>
            IF (mode = signal OR mode = error) AND (symbols.seb+sei).ctxnum = symbols.stHandle.outerCtx
** THEN
              BEGIN
              GetName[tname, (symbols.seb+sei).htptr];
              desc ← (symbols.seb+sei).idvalue;
              t ← nsigs;
              WHILE t > 0 DO
                IF sigdata[t-1].desc < desc THEN EXIT;
                sigdata[t] ← sigdata[t-1];
                t ← t-1;
                ENDLOOP;
              nsigs ← nsigs+1;
              sigdata[t]← [SystemDefs.AllocateHeapString[tname.length], desc];
              StringDefs.AppendString[sigdata[t].name,tname];
              END;
          ENDCASE;
      ENDLOOP;

    IODefs.WriteNumber[gframe, [8,FALSE,TRUE,6]];
    IODefs.WriteString["B "];
    IODefs.WriteLine[modname];

    IF nsigs > 0 THEN
      BEGIN OPEN OutputDefs;
      PutCR[];
      PutNumber[gframe, [8,FALSE,TRUE,6]];
      PutString["B "];
      PutString[modname]; PutCR[];
      FOR t IN [0..nsigs) DO
        PutString["  "];
        PutNumber[sigdata[t].desc+gfimask, [8,FALSE,TRUE,6]];
        PutString["B  "];
        PutString[sigdata[t].name]; PutCR[];
        ENDLOOP;
      END;
    ReleaseSymbolTable[symbols];
    DeleteFileSegment[symseg];
    RETURN
    END;

  GetModule: PROCEDURE [file:FileHandle] RETURNS [
   symseg:FileSegmentHandle, ngfi: CARDINAL] =
    BEGIN
    pages: PageCount;
    bcd: POINTER TO BcdDefs.BCD;
    mtb: CARDINAL;
    mti: BcdDefs.MTIndex = FIRST[BcdDefs.MTIndex];
    bcdseg: FileSegmentHandle ← NewFileSegment[file,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;
    IF bcd.versionident # BcdDefs.VersionID THEN
      BEGIN
      OutputDefs.PutString["  bad version ID "];
      OutputDefs.PutDecimal[bcd.versionident];
      Unlock[bcdseg];
      DeleteFileSegment[bcdseg];
      RETURN[NIL, 0]
      END;
    IF bcd.nModules # 1 THEN
      BEGIN
```

```
      OutputDefs.PutString["  too many modules: "];
      OutputDefs.PutDecimal[bcd.nModules];
      Unlock[bcdseg];
      DeleteFileSegment[bcdseg];
      RETURN[NIL, 0]
      END;
    mtb ← LOOPHOLE[bcd,CARDINAL]+bcd.mtOffset;
    ngfi ← (mtb+mti).ngfi;
    symseg ← FindSegment[bcdseg, (mtb+mti).sseg, FALSE];
    IF symseg # NIL THEN symseg.class ← symbols;
    Unlock[bcdseg];
    DeleteFileSegment[bcdseg];
    RETURN
    END;

  FindSegment: PROCEDURE [seg: FileSegmentHandle, segdesc: BcdDefs.SegDesc, long: BOOLEAN]
    RETURNS [FileSegmentHandle] =
    BEGIN
    ss: StringDefs.SubStringDescriptor;
    file: SegmentDefs.FileHandle;
    name: STRING;
    bcd: POINTER TO BcdDefs.BCD ← FileSegmentAddress[seg];
    IF segdesc.file = BcdDefs.FTNull THEN RETURN[NIL]
    ELSE IF segdesc.file = BcdDefs.FTSelf THEN file ← seg.file
    ELSE
      BEGIN OPEN f: LOOPHOLE[bcd+bcd.ftOffset, CARDINAL]+segdesc.file;
      name ← SystemDefs.AllocateHeapString[f.name.length+4];
      ss ← [LOOPHOLE[bcd+bcd.ssOffset, STRING], f.name.offset, f.name.length];
      StringDefs.AppendSubString[name, @ss];
      CheckForExtension[name, ".bcd"];
      file ← NewFile[name, DefaultAccess, DefaultVersion];
      SystemDefs.FreeHeapString[name];
      END;
    RETURN[NewFileSegment[file, segdesc.base,
      segdesc.pages + (IF long THEN segdesc.extraPages ELSE 0), Read]];
    END;

  CheckForExtension: PROCEDURE [name, ext: STRING] =
    BEGIN
    i: CARDINAL;
    FOR i IN [0..name.length) DO
      IF name[i] = '. THEN RETURN;
      ENDLOOP;
    StringDefs.AppendString[name, ext];
    RETURN
    END;

  parseloadmapline: PROCEDURE [line: STRING, file: STRING] RETURNS [gframe: CARDINAL] =
    BEGIN
    i: CARDINAL← 0;
    c: CARDINAL;
    gframe ← 0; file.length ← 0;
    WHILE i < line.length AND line[i] ~ IN ['0..'9] DO
      i ← i+1;
      ENDLOOP;
    WHILE i < line.length AND line[i] IN ['0..'9] DO
      c ← LOOPHOLE[line[i],CARDINAL]-LOOPHOLE['0,CARDINAL];
      gframe ← InlineDefs.BITSHIFT[gframe,3] + c;
      i ← i+1;
      ENDLOOP;
    WHILE i < line.length AND line[i] ~IN ['a..'z] AND line[i] ~IN ['A..'Z] DO
      i ← i+1;
      ENDLOOP;
    WHILE i < line.length AND line[i] # CR AND line[i] # '  DO
      StringDefs.AppendChar[file, line[i]];
      i ← i+1;
      ENDLOOP;
    RETURN;
    END;

  instring: STRING ← [200];
  CR: CHARACTER = IODefs.CR;

Done: SIGNAL = CODE;

ProcessLoadmap: PROCEDURE =
```

```
BEGIN OPEN StreamDefs;
ch: CHARACTER;
EndOfFile, firstline: BOOLEAN;
infile: STRING ← [40];
root: STRING ← [40];
instream: StreamHandle;
i, gframe: CARDINAL;
gfi: CARDINAL← 1;
ngfi: CARDINAL;

GetToken[infile];
IF infile.length = 0 THEN SIGNAL Done;
CheckForExtension[infile, ".loadmap"];

instream ← NewByteStream[infile, Read];

FOR i IN [0..infile.length) DO
  IF infile[i] = '. THEN EXIT;
  StringDefs.AppendChar[root, infile[i]];
  ENDLOOP;

OutputDefs.OpenOutput[root, ".signals."];

firstline ← TRUE;
EndOfFile ← FALSE;
UNTIL EndOfFile DO
  instring.length ← 0;
  DO
      BEGIN
      ch ← instream.get[instream !StreamError =>
        IF error = StreamAccess THEN GO TO fileend];
      EXITS
        fileend =>
          BEGIN
          EndOfFile ← TRUE;
          IF instring.length = 0 THEN GO TO nomorestrings;
          END;
      END;
    StringDefs.AppendChar[instring, ch];
    IF ch = CR OR EndOfFile THEN EXIT;
    ENDLOOP;

  IF firstline AND instring[0] # 'N THEN
    BEGIN
    OutputDefs.PutString[instring];  OutputDefs.PutCR[];
    IODefs.WriteLine[instring];
    END
  ELSE
    BEGIN
    firstline ← FALSE;
    IF instring[0] # 'N THEN GO TO nomorestrings;
    gframe ← parseloadmapline[instring, name];
    CheckForExtension[name, ".bcd"];
    ngfi ← PrintSignals[name, gframe, gfi];
    gfi ← gfi + ngfi;
    END;

  REPEAT
    nomorestrings => instream.destroy[instream];
  ENDLOOP;

OutputDefs.CloseOutput[];
END;

GetToken: PROCEDURE [token: STRING] =
  BEGIN
  c: CHARACTER;
  token.length ← 0;
  UNTIL comstr.endof[comstr] DO
    SELECT c ← comstr.get[comstr] FROM
      IODefs.SP, CR => IF token.length # 0 THEN RETURN;
      ENDCASE => StringDefs.AppendChar[token, c];
    ENDLOOP;
  RETURN
  END;
```

```
  -- Main Body

  name: STRING ← [80];
  comstr: StreamDefs.StreamHandle ← StreamDefs.NewByteStream["Com.Cm.", Read];
  START IODefs.StreamIO[NIL, NIL];
  GetToken[name];
  GetToken[name];
  DO ENABLE Done => EXIT;
    ProcessLoadmap[];
    ENDLOOP;
  ImageDefs.StopMesa[];

  END...
```