```
-- BcdDebug.Mesa  Edited by Johnsson on April 13, 1978  8:59 AM

DIRECTORY
  BcdControlDefs: FROM "bcdcontroldefs",
  BcdDefs: FROM "bcddefs",
  BcdTabDefs: FROM "bcdtabdefs",
  BcdTreeDefs: FROM "bcdtreedefs",
  IODefs: FROM "iodefs",
  StringDefs: FROM "stringdefs",
  TableDefs: FROM "tabledefs",
  TimeDefs: FROM "timedefs";

DEFINITIONS FROM IODefs, BcdTreeDefs, BcdDefs;

BcdDebug: PROGRAM
  IMPORTS BcdTabDefs, IODefs, TableDefs, TimeDefs
  EXPORTS BcdControlDefs =
  BEGIN

  tb, cxb, stb, mtb, etb, itb, ctb, sgb, ftb, ntb: TableDefs.TableBase;
  ssb: POINTER TO BcdDefs.PackedString;

  DebugNotify: TableDefs.TableNotifier =
    BEGIN
    tb ← base[treetype];
    cxb ← base[cxtype];
    stb ← base[sttype];
    mtb ← base[mttype];
    etb ← base[exptype];
    itb ← base[imptype];
    ctb ← base[cttype];
    sgb ← base[sgtype];
    ftb ← base[fttype];
    ntb ← base[nttype];
    ssb ← LOOPHOLE[base[sstype]];
    RETURN
    END;

  SubString: TYPE = StringDefs.SubString;

  desc: StringDefs.SubStringDescriptor;
  ss: SubString = @desc;

  -- Utility Writes

  WriteSubString: PROCEDURE [ss: SubString] =
    BEGIN i: CARDINAL;
    FOR i IN [ss.offset..ss.offset+ss.length) DO
      WriteChar[ss.base[i]]
      ENDLOOP;
    RETURN
    END;

  WriteName: PUBLIC PROCEDURE [n: NameRecord] =
    BEGIN
    ssd: StringDefs.SubStringDescriptor ← [
      base: @ssb.string, offset: n, length: ssb.size[n]];
    WriteSubString[@ssd];
    RETURN
    END;

  WriteTime: PUBLIC PROCEDURE [t: TimeDefs.PackedTime] =
    BEGIN OPEN TimeDefs;
    s: STRING ← [20];
    AppendDayTime[s,UnpackDT[t]];
    WriteString[s];
    RETURN
    END;

  WriteCR: PROCEDURE = BEGIN WriteChar[IODefs.CR] END;

  Indent: PROCEDURE [n: CARDINAL] =
    BEGIN
    THROUGH [1..n/8] DO WriteChar[IODefs.TAB] ENDLOOP;
    THROUGH [1..n MOD 8] DO WriteChar[' ] ENDLOOP;
    RETURN
    END;

  Tab: PROCEDURE [n: CARDINAL] =
    BEGIN WriteCR[];
    Indent[n];
    RETURN
    END;

  -- tree printing


  WriteNodeName: PROCEDURE[n: NodeName] =
    BEGIN
    NodePrintName: ARRAY NodeName OF STRING = [
      "list"L, "item"L, "source"L, "config"L,
      "assign"L, "plus"L, "then"L, "module"L];
    WriteString[NodePrintName[n]];
    RETURN
    END;

  printsubtree: PROCEDURE [t: TreeLink, Tabation: CARDINAL] =
    BEGIN
    node: TreeIndex;
    p: TreeXIndex;
    Tab[Tabation];
    WITH v: t SELECT FROM
      hash => printhti[v.index];
      symbol => BEGIN printhti[(stb+v.index).hti]; PrintIndex[v.index] END;
      string => PrintIndex[v.index];
      subtree =>
        BEGIN  node ← v.index;
        IF node = nullTreeIndex
          THEN WriteString["<empty>"L]
          ELSE
            BEGIN OPEN (tb+node);
            WriteNodeName[name];
            PrintIndex[node];
            WriteString[" codelinks: "L]; WriteChar[IF codelinks THEN 'T ELSE 'F];
            WriteString[", sourceindex: "L]; WriteOctal[sourceindex];
            p ← LOOPHOLE[node + TreeNodeSize];
            IF name = list AND nsons = 0 THEN
              UNTIL (tb+p).soni = endmark DO
                printsubtree[(tb+p).soni, Tabation+2];
                p ← p+1;
                ENDLOOP
            ELSE
              THROUGH [1..nsons] DO
                printsubtree[(tb+p).soni, Tabation+2];
                p ← p+1;
                ENDLOOP;
            END;
        END;
      ENDCASE;
    RETURN
    END;

  PrintTree: PUBLIC PROCEDURE [t: TreeLink] =
    BEGIN
    TableDefs.AddNotify[DebugNotify];
    printsubtree[t, 0];  WriteCR[];
    TableDefs.DropNotify[DebugNotify];
    RETURN
    END;

  printhti: PROCEDURE [hti: BcdTabDefs.HTIndex] =
    BEGIN
    desc: StringDefs.SubStringDescriptor;
    s: SubString = @desc;
    IF hti = BcdTabDefs.HTNull THEN WriteString["(anonymous)"L]
    ELSE
      BEGIN
      BcdTabDefs.SubStringForHash[s, hti];
      WriteSubString[s];
      END;
    RETURN
    END;
```

```
   contexts and semantic entries

  printContext: PROCEDURE [cxi: BcdTabDefs.CXIndex] =
    BEGIN OPEN BcdDefs, BcdTabDefs;
    sti: STIndex;
    TableDefs.AddNotify[DebugNotify];
    WriteString["Context: "L]; WriteDecimal[LOOPHOLE[cxi]];
    FOR sti ← (cxb+cxi).link, (stb+sti).link UNTIL sti = STNull DO
      OPEN stb+sti;
      Tab[2];
      printhti[hti]; PrintIndex[sti];
      IF filename THEN WriteString[", filename"L];
      IF assigned THEN WriteString[", assigned"L];
      IF imported THEN
        BEGIN
        WriteString[", imported"L];
        IF impi # IMPNull THEN PrintIndex[impi];
        END;
      IF exported THEN WriteString[", exported"L];
      WITH s:stb+sti SELECT FROM
        external =>
          BEGIN
          WriteString[", external["L];
          WITH m:s.map SELECT FROM
            config =>
              BEGIN WriteString["config"L]; PrintIndex[m.cti] END;
            module =>
              BEGIN WriteString["module"L]; PrintIndex[m.mti] END;
            interface =>
              BEGIN WriteString["interface"L]; PrintIndex[m.expi] END;
            ENDCASE;
          WITH p:s SELECT FROM
            file =>
              BEGIN WriteString[", file"L]; PrintIndex[p.fti] END;
            instance =>
              BEGIN WriteString[", instance"L]; PrintIndex[p.sti] END;
            ENDCASE;
          WriteChar[']];
          END;
        local =>
          BEGIN
          WriteString[", local"L];
          PrintIndex[s.info];
          WriteString[" context"L]; PrintIndex[s.context];
          END;
        ENDCASE;
      ENDLOOP;
    TableDefs.DropNotify[DebugNotify];
    WriteCR[];
    RETURN
    END;

  PrintSemanticEntries: PUBLIC PROCEDURE =
    BEGIN OPEN TableDefs, BcdTabDefs;
    cxi, cxLimit: CXIndex;
    cxLimit ← LOOPHOLE[TableDefs.TableBounds[cxtype].size];
    FOR cxi ← FIRST[CXIndex], cxi+SIZE[CXRecord] UNTIL cxi = cxLimit DO
      printContext[cxi];  WriteCR[];
      ENDLOOP;
    RETURN
    END;


-- various bcd tables

  PrintBcd: PUBLIC PROCEDURE =
    BEGIN
    PrintConfigs[];
    PrintImports[];  PrintExports[];
    PrintModules[];  PrintFiles[];
    RETURN
    END;

  PrintConfigs: PUBLIC PROCEDURE =
    BEGIN
    cti: CTIndex ← FIRST[CTIndex];
    ctLimit: CTIndex = LOOPHOLE[TableDefs.TableBounds[cttype].size];
    WriteCR[];
    WriteString["Configurations:"L];
    WriteCR[];
    UNTIL cti = ctLimit DO
      PrintConfig[cti];
      cti ← cti + SIZE[CTRecord];
      ENDLOOP;
    WriteCR[];
    RETURN
    END;

  PrintConfig: PUBLIC PROCEDURE [cti: CTIndex] =
    BEGIN OPEN ctb+cti;
     TableDefs.AddNotify[DebugNotify];
    Tab[2];
    WriteName[name];  PrintIndex[cti];
    IF namedinstance THEN
      BEGIN
      WriteString[", instance: "L];
      WriteNameFromTable[[config[cti]]];
      END;
    WriteString[", file: "L];
    PrintFileName[file];  PrintIndex[file];
    IF config # CTNull THEN
      BEGIN WriteString[", parent: "L];
      WriteName[(ctb+config).name];
      PrintIndex[config];
      END;
    IF control # MTNull THEN
      BEGIN  WriteString[", control: "L];
      WriteName[(mtb+control).name];
      PrintIndex[control];
      END;
    WriteCR[];
    TableDefs.DropNotify[DebugNotify];
    RETURN
    END;

  PrintImports: PUBLIC PROCEDURE =
    BEGIN
    iti: IMPIndex ← FIRST[IMPIndex];
    impLimit: IMPIndex = LOOPHOLE[TableDefs.TableBounds[imptype].size];
    WriteCR[];
    WriteString["Imports:"L];
    WriteCR[];
    UNTIL iti = impLimit DO
      PrintImport[iti];
      iti ← iti + SIZE[IMPRecord];
      ENDLOOP;
    WriteCR[];
    RETURN
    END;

  PrintImport: PUBLIC PROCEDURE [iti: IMPIndex] =
    BEGIN OPEN itb+iti;
     TableDefs.AddNotify[DebugNotify];
    Tab[2];
    WriteName[name];  PrintIndex[iti];
    SELECT port FROM
      module => WriteString[" (module)"L];
      interface => WriteString[" (interface)"L];
      ENDCASE;
    IF namedinstance THEN
      BEGIN
      WriteString[", instance: "L];
      WriteNameFromTable[[import[iti]]];
      END;
    WriteString[", file: "L];
    PrintFileName[file];  PrintIndex[file];
    WriteString[", gfi: "L];
    WriteDecimal[gfi];
    WriteString[", ngfi: "L];
    WriteDecimal[ngfi];
    WriteCR[];
    TableDefs.DropNotify[DebugNotify];
    RETURN
    END;

  PrintExports: PUBLIC PROCEDURE =
    BEGIN
    eti: EXPIndex ← FIRST[EXPIndex];
    expLimit: EXPIndex = LOOPHOLE[TableDefs.TableBounds[exptype].size];
    WriteString["Exports:"L];
    WriteCR[];
    UNTIL eti = expLimit DO
      PrintExport[eti];
      eti ← eti + (etb+eti).size + SIZE[EXPRecord];
      ENDLOOP;
    WriteCR[];
    RETURN
    END;

  PrintExport: PUBLIC PROCEDURE [eti: EXPIndex] =
    BEGIN OPEN etb+eti;
    i: CARDINAL;
     TableDefs.AddNotify[DebugNotify];
    Tab[2];
    WriteName[name];  PrintIndex[eti];
    IF port = module THEN WriteString[" [module]"L];
    IF namedinstance THEN
      BEGIN
      WriteString[", instance: "L];
      WriteNameFromTable[[export[eti]]];
      END;
    WriteString[", file: "L];
    PrintFileName[file];  PrintIndex[file];
    WriteString[", size: "L];
    WriteDecimal[size];
    WriteString[", links:"L];
    FOR i IN [0..size) DO
      IF i MOD 8 = 0 THEN Tab[4] ELSE WriteChar[' ];
      PrintControlLink[links[i]];
      IF i+1 # size THEN WriteChar[',];
      ENDLOOP;
    WriteCR[];
    TableDefs.DropNotify[DebugNotify];
    RETURN
    END;

  PrintModules: PUBLIC PROCEDURE =
    BEGIN
    mti: MTIndex ← FIRST[MTIndex];
    mtLimit: MTIndex = LOOPHOLE[TableDefs.TableBounds[mttype].size];
    WriteCR[];
    WriteString["Modules:"L];
    WriteCR[];
    UNTIL mti = mtLimit DO
      PrintModule[mti];
      mti ← mti + SIZE[MTRecord]+(mtb+mti).frame.length;
      ENDLOOP;
    WriteCR[];
    RETURN
    END;

  PrintModule: PUBLIC PROCEDURE [mti: MTIndex] =
    BEGIN OPEN mtb+mti;
    i: CARDINAL;
     TableDefs.AddNotify[DebugNotify];
    Tab[2];
    WriteName[name];  PrintIndex[mti];
    IF namedinstance THEN
      BEGIN
      WriteString["instance: "L];
      WriteNameFromTable[[module[mti]]];
      END;
    WriteString[", file: "L];
    PrintFileName[file];  PrintIndex[file];
    WriteString[", links: "L];
    WriteString[IF links=frame THEN "frame" ELSE "code"L];
    IF config # CTNull THEN
      BEGIN
      WriteString[", config: "L];
      WriteName[(ctb+config).name];
      PrintIndex[config];
      END;
    WriteString[", fsi: "L];  WriteDecimal[fsi];
    WriteString[", framesize: "L];  WriteDecimal[framesize];
    WriteString[", gfi: "L];  WriteDecimal[gfi];
    WriteString[", ngfi: "L];  WriteDecimal[ngfi];
    Tab[4];
    WriteString["code: "L];  PrintSegment[code.sgi];
    WriteString[", offset: "L];  WriteOctal[code.offset];
    WriteString[", length: "L];  WriteOctal[code.length];
    IF code.linkspace THEN WriteString [", space available for links"L];
    Tab[4];
    WriteString["symbols: "L];  PrintSegment[sseg];
    BEGIN OPEN frame;  Tab[4];
      WriteString[", frame length: "L];  WriteDecimal[length];
      WriteString[", control links:"L];
      FOR i IN [0..length) DO
        IF i MOD 8 = 0 THEN Tab[6] ELSE WriteChar[' ];
        PrintControlLink[frag[i]];
        IF i+1 # length THEN WriteChar[',];
        ENDLOOP;
      END;
    WriteCR[];
    TableDefs.DropNotify[DebugNotify];
    RETURN
    END;

  PrintSegment: PUBLIC PROCEDURE [sgi: SGIndex] =
    BEGIN OPEN sd: sgb+sgi;
    PrintFileName[sd.file]; WriteString[" [base: "L];
    WriteDecimal[sd.base]; WriteString[", pages: "L];
    WriteDecimal[sd.pages];
    IF sd.extraPages # 0 THEN
      BEGIN WriteChar['+]; WriteDecimal[sd.extraPages] END;
    WriteChar[']];
    RETURN
    END;

  PrintFiles: PUBLIC PROCEDURE =
    BEGIN
    fti: FTIndex ← FIRST[FTIndex];
    ftLimit: FTIndex = LOOPHOLE[TableDefs.TableBounds[fttype].size];
    WriteCR[];
    WriteString["Files:"L];
    WriteCR[];
    UNTIL fti = ftLimit DO
      PrintFile[fti];
      fti ← fti + SIZE[FTRecord];
      ENDLOOP;
    WriteCR[];
    RETURN
    END;

  PrintFile: PUBLIC PROCEDURE [fti: FTIndex] =
    BEGIN OPEN ftb+fti;
     TableDefs.AddNotify[DebugNotify];
    Tab[2];
    WriteName[name];  PrintIndex[fti];
    IF version.time = [0,0] THEN WriteString ["(Null Version)"L]
    ELSE
      BEGIN
      WriteString[", time: "L];
      WriteTime[version.time];
      WriteString[", processor: "L];
      PrintMachine[version];
      END;
    WriteCR[];
    TableDefs.DropNotify[DebugNotify];
    RETURN
    END;
```

```
-- Utility Prints

PrintControlLink: PROCEDURE [link: ControlLink] =
  BEGIN
  map: ARRAY ControlLinkTag OF CHARACTER = ['0,'1,'2,'3];
  WriteChar['[]; WriteDecimal[link.gfi];
  WriteChar[',]; WriteDecimal[link.ep];
  WriteChar[',]; WriteChar[map[link.tag]];
  WriteChar[']]; RETURN
  END;

PrintMachine: PROCEDURE [stamp: BcdDefs.VersionStamp] =
  BEGIN
  octal: NumberFormat = [8,FALSE,FALSE,1];
  WriteNumber[stamp.net, octal];
  WriteChar['#];
  WriteNumber[stamp.host, octal];
  WriteChar['#];
  IF stamp.zapped THEN WriteString[" zapped!"L];
  RETURN
  END;

PrintFileName: PROCEDURE [fti: FTIndex] =
  BEGIN
  SELECT fti FROM
    FTNull => WriteString["(null)"L];
    FTSelf => WriteString["(self)"L];
    ENDCASE => WriteName[(ftb+fti).name];
  RETURN
  END;

PrintIndex: PROCEDURE [index: UNSPECIFIED] =
  BEGIN
  WriteChar['[];
  IF index = TableDefs.TableLimit-1 THEN WriteString["Null"L]
  ELSE WriteDecimal[index];
  WriteChar[']];
  RETURN
  END;

WriteNameFromTable: PROCEDURE [n: Namee] =
  BEGIN OPEN TableDefs;
  nti: NTIndex;
  ntLimit: NTIndex = LOOPHOLE[TableBounds[nttype].size];
  FOR nti ← FIRST[NTIndex], nti + SIZE[NTRecord] UNTIL nti = ntLimit DO
    IF (ntb+nti).item = n THEN
      BEGIN WriteName[(ntb+nti).name]; RETURN END;
    ENDLOOP;
  RETURN
  END;


END.
```