```
-- file Pass4B.Mesa
-- last modified by Satterthwaite, July 17, 1978  10:34 AM

DIRECTORY
  AltoDefs: FROM "altodefs",
  BcdDefs: FROM "bcddefs",
  ComData: FROM "comdata",
  CompilerDefs: FROM "compilerdefs",
  CopierDefs: FROM "copierdefs",
  ErrorDefs: FROM "errordefs",
  P4Defs: FROM "p4defs",
  StreamDefs: FROM "streamdefs",
  StringDefs: FROM "stringdefs",
  SymDefs: FROM "symdefs",
  SymTabDefs: FROM "symtabdefs",
  SystemDefs: FROM "systemdefs",
  TableDefs: FROM "tabledefs",
  TreeDefs: FROM "treedefs",
  TypePackDefs: FROM "typepackdefs";

Pass4B: PROGRAM
    IMPORTS
        CompilerDefs, CopierDefs, ErrorDefs, P4Defs, StringDefs, SymTabDefs,
        SystemDefs, TreeDefs, TypePackDefs,
        dataPtr: ComData
    EXPORTS P4Defs =
  BEGIN
  OPEN SymTabDefs, SymDefs;

  tb: TableDefs.TableBase;      -- tree base address (local copy)
  seb: TableDefs.TableBase;     -- se table base address (local copy)
  ctxb: TableDefs.TableBase;    -- context table base address (local copy)
  mdb: TableDefs.TableBase;     -- body table base address (local copy)
  bb: TableDefs.TableBase;      -- body table base address (local copy)

  BCDNotify: PUBLIC TableDefs.TableNotifier =
    BEGIN  -- called by allocator whenever table area is repacked
    tb ← base[TreeDefs.treetype];
    seb ← base[setype];  ctxb ← base[ctxtype];  mdb ← base[mdtype];
    bb ← base[bodytype];  RETURN
    END;


 -- shared variables

  bcdHeader: POINTER TO BcdDefs.BCD;
  bcdOffset, mtOffset: CARDINAL;

  nString: BcdDefs.NameString;

  firstPorted: MDIndex = FIRST[MDIndex] + SIZE[MDRecord];
  lastPorted: MDIndex;  -- im/exported files in [firstPorted..lastPorted)

 -- service routines

  GFTIndex: TYPE = BcdDefs.GFTIndex;
  EPIndex: TYPE = BcdDefs.EPIndex;
  EPLimit: CARDINAL = LAST[EPIndex]+1;
  ControlLink: TYPE = BcdDefs.ControlLink;

  OwnGfi: GFTIndex = 1;

  GFSlots: PROCEDURE [epMax: EPIndex] RETURNS [nGfi: [1..4]] =
    BEGIN
    nGfi ← epMax/EPLimit + 1;  RETURN
    END;

  MakeEPLink: PUBLIC PROCEDURE [ep: CARDINAL, gfi: GFTIndex] RETURNS [ControlLink] =
    BEGIN
    RETURN [ControlLink[procedure[
        tag: procedure,
        ep: ep MOD EPLimit,
        gfi: gfi + ep/EPLimit]]]
    END;

  MakeFrameLink: PROCEDURE [ep: CARDINAL, gfi: GFTIndex] RETURNS [ControlLink] =
```

```
    BEGIN
    RETURN [ControlLink[procedure[
        tag: frame,
        ep: ep MOD EPLimit,
        gfi: gfi + ep/EPLimit]]]
    END;

  RelocateEPLink: PROCEDURE [link: ControlLink, offset: CARDINAL] RETURNS [ControlLink] =
    BEGIN
    IF link.gfi # BcdDefs.GFTNull THEN link.gfi ← link.gfi + offset;
    RETURN [link]
    END;


  PortedFile: PROCEDURE [ctx: CTXIndex] RETURNS [fti: BcdDefs.FTIndex] =
    BEGIN
    mdi: MDIndex;
    n: CARDINAL;
    mdi ← WITH c: (ctxb+ctx) SELECT FROM
      included => c.ctxmodule,
      imported => (ctxb+c.includeLink).ctxmodule,
      ENDCASE => OwnMdi;
    IF mdi = OwnMdi
      THEN  fti ← BcdDefs.FTSelf
      ELSE
        BEGIN
        IF mdi IN [firstPorted .. lastPorted)
          THEN  n ← LOOPHOLE[mdi-firstPorted, CARDINAL]/SIZE[MDRecord]
          ELSE
            BEGIN
            n ← LOOPHOLE[lastPorted-firstPorted, CARDINAL]/SIZE[MDRecord];
            SwapMdi[mdi, lastPorted];
            lastPorted ← lastPorted + SIZE[MDRecord];
            END;
        fti ← LOOPHOLE[n*SIZE[BcdDefs.FTRecord]];
        END;
    RETURN
    END;

  SwapMdi: PROCEDURE [mdi1, mdi2: MDIndex] =
    BEGIN
    ctx: includedCTXIndex;
    t: MDRecord;
    IF mdi1 # mdi2
      THEN
        BEGIN
        FOR ctx ← (mdb+mdi1).mdctx, (ctxb+ctx).ctxchain UNTIL ctx = CTXNull
          DO  (ctxb+ctx).ctxmodule ← mdi2  ENDLOOP;
        FOR ctx ← (mdb+mdi2).mdctx, (ctxb+ctx).ctxchain UNTIL ctx = CTXNull
          DO  (ctxb+ctx).ctxmodule ← mdi1  ENDLOOP;
        t ← (mdb+mdi1)↑;  (mdb+mdi1)↑ ← (mdb+mdi2)↑;  (mdb+mdi2)↑ ← t;
        END;
    RETURN
    END;


  SubString: TYPE = StringDefs.SubString;
  SubStringDescriptor: TYPE = StringDefs.SubStringDescriptor;

  EnterId: PROCEDURE [id: SubString, ignoreCase: BOOLEAN] RETURNS [BcdDefs.NameRecord] =
    BEGIN
    i: CARDINAL;
    desc: SubStringDescriptor;
    s: SubString = @desc;
    t: BcdDefs.NameString;
    i ← 0;  s.base ← @nString.string;
    UNTIL i = nString.string.length
      DO
      s.offset ← i ← i+1;  s.length ← nString.size[i];
      IF (IF ignoreCase
            THEN StringDefs.EquivalentSubStrings
            ELSE StringDefs.EqualSubStrings)[id, s]
        THEN EXIT;
      i ← i + s.length;
      REPEAT
        FINISHED =>
```

```
        BEGIN
        IF nString.string.length + (id.length+1) > nString.string.maxlength
          THEN
            BEGIN     -- rewrite if nString is in table area
            t ← LOOPHOLE[SystemDefs.AllocateHeapString[
                  nString.string.maxlength + MAX[(id.length+1), 32]]];
            StringDefs.AppendString[@t.string, @nString.string];
            SystemDefs.FreeHeapString[@nString.string];  nString ← t;
            END;
        i ← nString.string.length ← nString.string.length + 1;
        nString.size[i] ← id.length;
        StringDefs.AppendSubString[@nString.string, id];
        END;
    ENDLOOP;
  RETURN [[i]]
  END;

EnterSymbolId: PROCEDURE [sei: ISEIndex] RETURNS [BcdDefs.NameRecord] =
  BEGIN
  s: SubStringDescriptor;
  SubStringForHash[@s, (seb+sei).htptr];
  RETURN [EnterId[@s, FALSE]]
  END;

EnterFileId: PROCEDURE [mdi: MDIndex] RETURNS [BcdDefs.NameRecord] =
  BEGIN
  s, dExt: SubStringDescriptor;
  bcd: STRING = ".bcd.";
  dBcd: SubStringDescriptor ← [base:bcd, offset:0, length:bcd.length];
  SubStringForHash[@s, (mdb+mdi).mdhti];
  dExt ← [base: s.base, offset: s.offset+s.length-bcd.length, length: bcd.length];
  IF StringDefs.EquivalentSubStrings[@dExt, @dBcd]
    THEN s.length ← s.length - bcd.length;
  RETURN [EnterId[@s, TRUE]]
  END;


TreeSymbol: PROCEDURE [t: TreeDefs.TreeLink] RETURNS [ISEIndex] =
  BEGIN
  WITH t SELECT FROM
    symbol => RETURN [index];
    ENDCASE => ERROR;
  END;


-- relocating imported control links

RelocateImports: PROCEDURE [ctx: CTXIndex, gfi: GFTIndex] RETURNS [epMax: EPIndex] =
  BEGIN
  sei: ISEIndex;
  epN: CARDINAL;
  epMax ← 0;
  IF ctx = CTXNull THEN RETURN;
  FOR sei ← (ctxb+ctx).selist, NextSe[sei] UNTIL sei = SENull
    DO
    epN ← (seb+sei).idvalue;
    epMax ← MAX[epMax, epN];
    (seb+sei).idvalue ← SELECT XferMode[(seb+sei).idtype] FROM
      procedure, signal, error => MakeEPLink[epN, gfi],
      program => MakeFrameLink[epN, gfi],
      ENDCASE => BcdDefs.UnboundLink;
    ENDLOOP;
  RETURN
  END;

AssignImports: PUBLIC TreeDefs.TreeScan =
  BEGIN
  gfi: GFTIndex;
  saveIndex: CARDINAL = dataPtr.textIndex;

  ImportItem: TreeDefs.TreeScan =
    BEGIN
    node: TreeDefs.TreeIndex = TreeDefs.GetNode[t];
    sei: ISEIndex = TreeSymbol[(tb+node).son1];
    type: CSEIndex = UnderType[(seb+sei).idtype];
    epMax: EPIndex;
```

```
      dataPtr.textIndex ← (tb+node).info;
      WITH (seb+type) SELECT FROM
        definition =>
          BEGIN
          IF (ctxb+defCtx).selist = SENull
            THEN
              WITH c: (ctxb+defCtx) SELECT FROM
                imported =>
                  IF (tb+node).attr1
                   OR ~(mdb+(ctxb+c.includeLink).ctxmodule).mdExported
                    THEN  ErrorDefs.WarningSei[unusedImport, sei];
                ENDCASE;
          epMax ← RelocateImports[defCtx, gfi];
          (seb+sei).idvalue ← gfi;
          gfi ← gfi + ((seb+sei).idinfo ← nGfi);
          -- consider adding GFSlots[epMax] instead --
          END;
        pointer =>
          BEGIN
          (seb+sei).idvalue ← MakeFrameLink[ep:0, gfi:gfi];
          gfi ← gfi + 1;
          END;
        ENDCASE;
      (seb+sei).mark4 ← TRUE;
      RETURN
      END;

    dataPtr.mtRoot.gfi ← OwnGfi;
    dataPtr.mtRoot.ngfi ← GFSlots[MAX[dataPtr.nBodies, dataPtr.nSigCodes]-1];
    gfi ← bcdHeader.firstdummy ← OwnGfi + dataPtr.mtRoot.ngfi;
    TreeDefs.scanlist[t, ImportItem];
    bcdHeader.nDummies ← gfi - bcdHeader.firstdummy;
    dataPtr.textIndex ← saveIndex;  RETURN
    END;


-- writing import records

  ProcessImports: PUBLIC TreeDefs.TreeScan =
    BEGIN
    -- N.B. nextGfi must be regenerated to match AssignImports
    nImports: CARDINAL;
    impi: BcdDefs.IMPIndex;
    nextGfi: GFTIndex;
    anyNamed: BOOLEAN;

    ImportItem: TreeDefs.TreeScan =
      BEGIN
      node: TreeDefs.TreeIndex = TreeDefs.GetNode[t];
      sei1: ISEIndex = TreeSymbol[(tb+node).son1];
      sei2: ISEIndex = TreeSymbol[(tb+node).son2];
      type: CSEIndex;
      rType: recordCSEIndex;
      entry: BcdDefs.IMPRecord;
      entry ← [
                name: EnterSymbolId[sei2],
                port: interface,
                namedinstance: (seb+sei1).htptr # (seb+sei2).htptr,
                file: ,
                gfi: ,
                ngfi: ];
      type ← UnderType[(seb+sei1).idtype];
      WITH (seb+type) SELECT FROM
        definition =>
          BEGIN
          entry.file ← PortedFile[defCtx];
          entry.gfi ← (seb+sei1).idvalue;
          entry.ngfi ← (seb+sei1).idinfo;
          nextGfi ← (seb+sei1).idvalue + (seb+sei1).idinfo;
          END;
        pointer =>
          BEGIN
          entry.port ← module;
          rType ← LOOPHOLE[UnderType[pointedtotype]];
          entry.file ← PortedFile[(seb+rType).fieldctx];
          entry.gfi ← nextGfi;  entry.ngfi ← 1;
```

```
        nextGfi ← nextGfi + 1;
        END;
      ENDCASE;
    nImports ← nImports + 1;
    IF entry.namedinstance THEN anyNamed ← TRUE;
    CompilerDefs.AppendBCDWords[@entry, SIZE[BcdDefs.IMPRecord]];
    RETURN
    END;

  NameItem: TreeDefs.TreeScan =
    BEGIN
    node: TreeDefs.TreeIndex = TreeDefs.GetNode[t];
    sei1: ISEIndex = TreeSymbol[(tb+node).son1];
    sei2: ISEIndex = TreeSymbol[(tb+node).son2];
    entry: BcdDefs.NTRecord;
    IF  (seb+sei1).htptr # (seb+sei2).htptr
      THEN
        BEGIN
        entry ← [
              name: EnterSymbolId[sei1],
              item: BcdDefs.Namee[import[impi]]];
        CompilerDefs.AppendBCDWords[@entry, SIZE[BcdDefs.NTRecord]];
        END;
    impi ← impi + SIZE[BcdDefs.IMPRecord];
    RETURN
    END;

  offset: CARDINAL;
  bcdHeader.impOffset ← offset ← CompilerDefs.ReadBCDOffset[];
  nImports ← 0;  impi ← FIRST[BcdDefs.IMPIndex];
  nextGfi ← bcdHeader.firstdummy;  anyNamed ← FALSE;
  TreeDefs.scanlist[t, ImportItem];
  bcdHeader.nImports ← nImports;
  bcdHeader.impLimit ← LOOPHOLE[CompilerDefs.ReadBCDOffset[]-offset];
  bcdHeader.ntOffset ← offset ← CompilerDefs.ReadBCDOffset[];
  IF anyNamed THEN TreeDefs.scanlist[t, NameItem];
  bcdHeader.ntLimit ← LOOPHOLE[CompilerDefs.ReadBCDOffset[]-offset];
  RETURN
  END;


-- writing export records

 ExportId: TreeDefs.TreeMap =
  BEGIN
  type: CSEIndex = P4Defs.OperandType[t];
  ctx: includedCTXIndex;
  iBase: TypePackDefs.SymbolTableBase;
  id, sei, iSei: ISEIndex;
  ss: StringDefs.SubStringDescriptor;
  hti: HTIndex;
  link: ControlLink;
  header: BcdDefs.EXPRecord;
  nItems: CARDINAL;
  id ← TreeSymbol[t];
  WITH v: (seb+type) SELECT FROM
    definition =>
      BEGIN  ctx ← LOOPHOLE[v.defCtx];
      iBase ← CopierDefs.GetSymbolTable[(ctxb+ctx).ctxmodule];
      IF iBase # NIL
        THEN
          BEGIN
          header ← [
              name: EnterSymbolId[id],
              size: 0,
              port: interface,
              namedinstance: FALSE,
              file: PortedFile[v.defCtx],
              links: ];
          FOR iSei ← iBase.FirstCtxSe[(ctxb+ctx).ctxmap],
           iBase.NextSe[iSei] UNTIL iSei = SENull
            DO
            IF ~(iBase.seb+iSei).constant
              THEN  header.size ← header.size + 1;
            ENDLOOP;
          CompilerDefs.AppendBCDWords[@header, SIZE[BcdDefs.EXPRecord]];
```

```
            nItems ← 0;
            FOR iSei ← iBase.FirstCtxSe[(ctxb+ctx).ctxmap],
             iBase.NextSe[iSei] UNTIL iSei = SENull
              DO
              IF ~(iBase.seb+iSei).constant
                THEN
                  BEGIN
                  link ← BcdDefs.NullLink;
                  iBase.SubStringForHash[@ss, (iBase.seb+iSei).htptr];
                  hti ← FindString[@ss];
                  IF hti = HTNull
                    THEN  sei ← ISENull
                    ELSE
                      BEGIN
                      sei ← SearchContext[hti, dataPtr.mainCtx];
                      IF sei = SENull
                        THEN sei ← SearchContext[hti, dataPtr.moduleCtx];
                      END;
                  IF sei # SENull AND (seb+sei).public
                    THEN
                      BEGIN
                      IF ~(seb+sei).constant OR (seb+sei).extended
                        THEN ErrorDefs.errorsei[varExport, sei];
                      IF ~TypePackDefs.AssignableTypes[
                       [iBase, iBase.UnderType[(iBase.seb+iSei).idtype]],
                       [dataPtr.ownSymbols, UnderType[(seb+sei).idtype]]]
                        THEN ErrorDefs.errorsei[exportClash, sei];
                      link ← RelocateEPLink[
                        IF XferMode[(seb+sei).idtype] = program
                          THEN MakeFrameLink[ep:0, gfi:0]
                          ELSE (seb+sei).idvalue,
                        OwnGfi];
                      nItems ← nItems + 1;
                      END
                    ELSE
                      IF sei # SENull AND (seb+sei).ctxnum = dataPtr.mainCtx
                       AND TypePackDefs.AssignableTypes[
                        [iBase, iBase.UnderType[(iBase.seb+iSei).idtype]],
                        [dataPtr.ownSymbols, UnderType[(seb+sei).idtype]]]
                          THEN ErrorDefs.WarningSei[privateExport, sei];
                  CompilerDefs.AppendBCDWord[link];
                  END;
              ENDLOOP;
            CopierDefs.FreeSymbolTable[iBase];
            IF nItems = 0 THEN ErrorDefs.WarningSei[unusedExport, id];
            END;
        END;
      ENDCASE;
    RETURN [t]
    END;

  ProcessExports: PUBLIC TreeDefs.TreeMap =
    BEGIN
    offset: CARDINAL = CompilerDefs.ReadBCDOffset[];
    bcdHeader.nExports ← TreeDefs.listlength[t];
    bcdHeader.expOffset ← offset;
    v ← TreeDefs.updatelist[t, ExportId];
    bcdHeader.expLimit ← LOOPHOLE[CompilerDefs.ReadBCDOffset[]-offset];
    RETURN
    END;

  -- initialization/finalization

  ProcessFiles: PROCEDURE =
    BEGIN
    ftEntry: BcdDefs.FTRecord;
    mdi: MDIndex;
    offset: CARDINAL = CompilerDefs.ReadBCDOffset[];
    bcdHeader.ftOffset ← offset;
    FOR mdi ← firstPorted, mdi+SIZE[MDRecord] UNTIL mdi = lastPorted
      DO
      ftEntry ← [name: EnterFileId[mdi], version: (mdb+mdi).mdStamp];
      CompilerDefs.AppendBCDWords[@ftEntry, SIZE[BcdDefs.FTRecord]];
      ENDLOOP;
    bcdHeader.ftLimit ← LOOPHOLE[CompilerDefs.ReadBCDOffset[] - offset];
    RETURN
```

```
    END;


  MTRootSize: CARDINAL = SIZE[BcdDefs.MTRecord]-SIZE[BcdDefs.FrameFrag];

  InitBCD: PUBLIC PROCEDURE =
    BEGIN  OPEN BcdDefs;
    lastPorted ← firstPorted;
    nString ← LOOPHOLE[SystemDefs.AllocateHeapString[64]];
    -- allocate the null name
      nString.string.length ← BcdDefs.NullName;
      nString.size[BcdDefs.NullName] ← 0;
    CompilerDefs.StartBCD[];
    bcdHeader ← SystemDefs.AllocateHeapNode[SIZE[BCD]];
      bcdHeader.versionident ← VersionID;
      bcdHeader.version ← dataPtr.objectVersion;
      bcdHeader.sourceVersion ← dataPtr.sourceVersion;
      bcdHeader.creator ← dataPtr.compilerVersion;
      bcdHeader.bound ← VersionStamp[zapped:FALSE, net:0, host:0, time:[0,0]];
      bcdHeader.nConfigs ← 0;
      bcdHeader.nModules ← 1;
      bcdHeader.nImports ← bcdHeader.nExports ← 0;
      bcdHeader.definitions ← dataPtr.definitionsOnly;
      bcdHeader.ctOffset ← 0;  bcdHeader.ctLimit ← LOOPHOLE[0];
      nString.string.length ← nString.string.length + 1;
      bcdHeader.source ← NameRecord[nString.string.length];
        nString.size[bcdHeader.source] ← dataPtr.sourceFile.length;
        StringDefs.AppendString[@nString.string, dataPtr.sourceFile];
    bcdOffset ← CompilerDefs.ReadBCDOffset[];
    CompilerDefs.AppendBCDWords[bcdHeader, SIZE[BCD]];
    dataPtr.fixupLoc ← CompilerDefs.ReadBCDIndex[];
    bcdHeader.sgOffset ← CompilerDefs.ReadBCDOffset[];
    CompilerDefs.AppendBCDWords[@dataPtr.codeSeg, SIZE[SGRecord]];
    CompilerDefs.AppendBCDWords[@dataPtr.symSeg, SIZE[SGRecord]];
    bcdHeader.mtOffset ← mtOffset ← CompilerDefs.ReadBCDOffset[];
    bcdHeader.sgLimit ← LOOPHOLE[mtOffset - bcdHeader.sgOffset];
    CompilerDefs.AppendBCDWords[@dataPtr.mtRoot, MTRootSize];
    RETURN
    END;

  FinishBCD: PUBLIC PROCEDURE =
    BEGIN  OPEN BcdDefs;
    PageSize: CARDINAL = AltoDefs.PageSize;
    bcdSize: CARDINAL;
    -- fill MTRecord
      dataPtr.mtRoot.name ← EnterSymbolId[(bb+dataPtr.mainBody).id];
      dataPtr.mtRoot.namedinstance ← FALSE;
      dataPtr.mtRoot.initial ← FALSE;
      dataPtr.mtRoot.file ← dataPtr.codeSeg.file ← dataPtr.symSeg.file ←
        PortedFile[dataPtr.mainCtx];
      dataPtr.mtRoot.links ← frame;
      dataPtr.mtRoot.config ← CTNull;
      dataPtr.mtRoot.code ←
        CodeDesc[
            sgi: FIRST[SGIndex],
            packed: FALSE, linkspace: FALSE,
            offset: 0, length: ];
      dataPtr.mtRoot.sseg ← FIRST[SGIndex] + SIZE[SGRecord];
    bcdHeader.mtLimit ← LOOPHOLE[bcdHeader.impOffset-bcdHeader.mtOffset];
    ProcessFiles[];
    bcdHeader.ssOffset ← CompilerDefs.ReadBCDOffset[];
    CompilerDefs.AppendBCDString[@nString.string];
    bcdSize ← CompilerDefs.ReadBCDOffset[];
    bcdHeader.ssLimit ← LOOPHOLE[bcdSize-bcdHeader.ssOffset];
    bcdHeader.nPages ← (bcdSize + (PageSize-1))/PageSize;
    CompilerDefs.UpdateBCDWords[bcdOffset, bcdHeader, SIZE[BCD]];
    CompilerDefs.EndBCD[];
    SystemDefs.FreeHeapString[@nString.string];
    SystemDefs.FreeHeapNode[bcdHeader];
    RETURN
    END;

  END.
```