```
-- Wart.Mesa  Edited by Sandman on May 12, 1978  3:34 PM

DIRECTORY
  AltoFileDefs: FROM "altofiledefs" USING [CFP, FP],
  BootDefs: FROM "bootdefs" USING [
    BootDataSegment, BootFile, BootFileSegment],
  ControlDefs: FROM "controldefs" USING [
    FrameHandle, GetReturnFrame, GlobalFrameHandle, Greg, StateVector, WDCreg,
    XTSreg],
  DirectoryDefs: FROM "directorydefs" USING [EnumerateDirectory],
  DiskDefs: FROM "diskdefs" USING [DA, VirtualDA],
  FrameDefs: FROM "framedefs" USING [Restart, Start, SwapOutCode],
  ImageDefs: FROM "imagedefs" USING [FileRequest, FirstImageDataPage],
  NovaOps: FROM "novaops" USING [NovaJSR, Stop],
  NucleusDefs: FROM "nucleusdefs" USING [
    DiskIO, DiskKD, Files, LoadState, NonResident, Process, Resident,
    Segments, Signaller, Swapper],
  SDDefs: FROM "sddefs" USING [sAddFileRequest, sBreak],
  SegmentDefs: FROM "segmentdefs" USING [
    AddressFromPage, DataSegmentHandle, DeleteDataSegment, DeleteFileSegment,
    EnumerateFileSegments, FileHandle, FileHint, FileSegmentHandle,
    InsertFile, NewDataSegment, PageFromAddress, PageNumber, Read, Unlock,
    VMtoDataSegment, Write],
  StringDefs: FROM "stringdefs" USING [
    EquivalentString, EquivalentSubStrings, SubStringDescriptor],
  TrapDefs: FROM "trapdefs" USING [TraceOff],
  WartDefs: FROM "wartdefs" USING [
    BootIndex, BootScriptEntry, BootScriptHeader, NullBootIndex];

DEFINITIONS FROM WartDefs, ControlDefs, SegmentDefs;

Wart: PROGRAM [h: POINTER TO BootScriptHeader] RETURNS [PROGRAM]
  IMPORTS BootDefs, DirectoryDefs, DiskDefs, FrameDefs, NucleusDefs,
    SegmentDefs, StringDefs
  EXPORTS NucleusDefs SHARES ControlDefs, DiskDefs, ImageDefs, SegmentDefs =
  BEGIN

  WartBreak: PROCEDURE  =
    BEGIN OPEN NovaOps;
    s: StateVector;
    f: FrameHandle;
    break: RECORD[a,b: WORD];
    s ← STATE;
    break ← [77400B, 1400B];
    f ← GetReturnFrame[];
    s.dest ← f;
    f.pc ← [IF f.pc < 0 THEN -f.pc ELSE (1-f.pc)];
    s.instbyte ← NovaJSR[JSR, @break, 0];
    RETURN WITH s;
    END;

  Nub: TYPE = PROGRAM [GlobalFrameHandle];

  ProcessWartList: PROCEDURE RETURNS [PROGRAM] =
    BEGIN
    eb: CARDINAL = h.tablebase;
    imagefile: SegmentDefs.FileHandle;
    oldp, p: BootIndex ← FIRST[WartDefs.BootIndex];
    OldBreak: PROCEDURE;
    MyBreak: PROCEDURE ← WartBreak;
    SD: POINTER TO ARRAY [0..1) OF UNSPECIFIED = h.sd;
    ptSegment: DataSegmentHandle ← NIL;
    pPageTable: POINTER TO PageTable = ppPageTable↑;
    vmaddr: POINTER;
    RequestHead: POINTER TO ImageDefs.FileRequest ← NIL;
    AddFileRequest: PROCEDURE [r: POINTER TO ImageDefs.FileRequest] =
      BEGIN
      r.link ← RequestHead;
      RequestHead ← r;
      END;
    ProcessFileRequests: PROCEDURE =
      BEGIN OPEN AltoFileDefs;
      checkone: PROCEDURE [fp: POINTER TO FP, dname: STRING] RETURNS [BOOLEAN] =
        BEGIN
        ss: StringDefs.SubStringDescriptor ← [dname,0,dname.length];
        r: POINTER TO ImageDefs.FileRequest;
```

```
      prev: POINTER TO ImageDefs.FileRequest ← NIL;
      FOR r ← RequestHead, r.link UNTIL r = NIL DO
        IF (WITH r SELECT FROM
              long => StringDefs.EquivalentSubStrings[@ss,@name],
              short => StringDefs.EquivalentString[dname,name],
              ENDCASE => FALSE) THEN
          BEGIN
          IF r.file = NIL THEN r.file ← InsertFile[fp,r.access]
          ELSE r.file.fp ← fp↑;
          IF prev = NIL THEN RequestHead ← r.link
          ELSE prev.link ← r.link;
          END
        ELSE prev ← r;
        ENDLOOP;
      RETURN[RequestHead = NIL]
      END;
    DirectoryDefs.EnumerateDirectory[checkone];
    END;

  REGISTER[WDCreg] ← 0;  -- interrupts on
  SD[SDDefs.sBreak] ← MyBreak;
  SD[SDDefs.sAddFileRequest] ← AddFileRequest;
  DO -- exited by a Stop BootScriptEntry
    WITH (eb+p) SELECT FROM
      Command =>
        BEGIN
        SELECT command FROM
          bc0 =>
            BEGIN
            FrameDefs.Start[LOOPHOLE[NucleusDefs.Resident]];
            FrameDefs.Start[LOOPHOLE[NucleusDefs.NonResident]];
            OldBreak ← SD[SDDefs.sBreak];
            SD[SDDefs.sBreak] ← MyBreak;
            START NucleusDefs.DiskIO;
            START NucleusDefs.Swapper[h.ffvmp, h.lfvmp];
            ptSegment ← NewDataSegment[PageFromAddress[pPageTable], 1];
            START NucleusDefs.Process;
            START NucleusDefs.Signaller;
            START NucleusDefs.Segments;
            START NucleusDefs.Files;
            imagefile ← BootDefs.BootFile[Read+Write];
            END;
          bc1 =>
            BEGIN
            START NucleusDefs.DiskKD;
            START NucleusDefs.LoadState[(eb+h.loadState).handle,
              (eb+h.initLoadState).handle, (eb+h.bcd).handle];
            SegmentDefs.Unlock[(eb+h.bcd).handle];
            SegmentDefs.DeleteFileSegment[(eb+h.bcd).handle];
            START LOOPHOLE[h.nub, Nub][h.user];
            END;
          bc2 =>
            BEGIN
            RESTART NucleusDefs.Resident;
            END;
          ENDCASE;
        p ← p + SIZE [Command BootScriptEntry];
        END;
      SwapOutCode =>
        BEGIN
        FrameDefs.SwapOutCode[frame];
        p ← p + SIZE [SwapOutCode BootScriptEntry];
        END;
      OpenFile =>
        BEGIN
        ProcessFileRequests[];
        p ← p + SIZE [OpenFile BootScriptEntry];
        END;
      Segment =>
        BEGIN OPEN BootDefs;
        vmaddr ← IF vmpage = 0 THEN NIL ELSE AddressFromPage[vmpage];
        handle ← IF data
          THEN LOOPHOLE[BootDataSegment[base,pages]]
          ELSE BootFileSegment[imagefile, base, pages, access, vmaddr];
        p ← p + SIZE [Segment BootScriptEntry];
        END;
```

```
        CodeLink =>
          BEGIN
          frame.codesegment ← (eb+codeseg).handle;
          (eb+codeseg).handle.class ← code;
          IF frame = LOOPHOLE[REGISTER[Greg], GlobalFrameHandle] THEN
            BEGIN
            BootPageTable[imagefile, pPageTable];
            DeleteDataSegment[ptSegment];
            END;
          p ← p + SIZE [CodeLink BootScriptEntry];
          END;
        Unlock =>
          BEGIN
          IF seg#NullBootIndex THEN Unlock[(eb+seg).handle];
          p ← p + SIZE [Unlock BootScriptEntry];
          END;
        Stop =>
          EXIT;
        ENDCASE => NovaOps.Stop[Punt];
      IF p=oldp THEN NovaOps.Stop[Punt];
      oldp ← p;
      ENDLOOP;

    DeleteDataSegment[VMtoDataSegment[LOOPHOLE[eb]]];
    SD[SDDefs.sAddFileRequest] ← 0;
    SD[SDDefs.sBreak] ← OldBreak;
    GetReturnFrame[].returnlink ← LOOPHOLE[FrameDefs.Restart];
    RETURN[LOOPHOLE[h.nub]];
    END;


-- page table

  PageTable: TYPE = MACHINE DEPENDENT RECORD [
    fp: AltoFileDefs.CFP,
    firstpage: CARDINAL,
    table: ARRAY [0..1) OF DiskDefs.DA];
  ppPageTable: POINTER TO POINTER TO PageTable = LOOPHOLE[24B];

  BootPageTable: PROCEDURE [file:FileHandle, pt:POINTER TO PageTable] =
    BEGIN OPEN AltoFileDefs, DiskDefs;
    lastpage: PageNumber;
    pageInc: PageNumber = pt.firstpage - ImageDefs.FirstImageDataPage;
    PlugHint: PROCEDURE [seg:FileSegmentHandle] RETURNS [BOOLEAN] =
      BEGIN
      IF seg.file = file THEN
        BEGIN
        seg.base ← seg.base + pageInc;
        IF seg.base IN [pt.firstpage..lastpage] THEN
          WITH s: seg SELECT FROM
            disk => s.hint ← FileHint[
              page: s.base,
              da: DiskDefs.VirtualDA[pt.table[s.base-pt.firstpage]]];
            ENDCASE;
        END;
      RETURN[FALSE]
      END;
    file.open ← TRUE;
    file.fp ← FP[serial: pt.fp.serial, leaderDA: pt.fp.leaderDA];
    FOR lastpage ← 0, lastpage+1
    UNTIL pt.table[lastpage] = DA[0,0,0,0,0]
      DO NULL ENDLOOP;
    IF lastpage = 0 THEN RETURN;
    lastpage ← lastpage+pt.firstpage-1;
    [] ← EnumerateFileSegments[PlugHint];
    RETURN
    END;

  REGISTER[ControlDefs.XTSreg] ← TrapDefs.TraceOff;
  RETURN[ProcessWartList[]];

  END..
```