```
-- RunImage.mesa; edited by Sandman; July 17, 1978  12:02 PM

DIRECTORY
  AllocDefs: FROM "allocdefs" USING [AllocInfo, MakeDataSegment],
  AltoDefs: FROM "altodefs" USING [PageSize],
  AltoFileDefs: FROM "altofiledefs" USING [CFP],
  BFSDefs: FROM "bfsdefs" USING [MakeCFP],
  BinaryDefs: FROM "binarydefs" USING [MesaBootLoader],
  ControlDefs: FROM "controldefs" USING [
    FieldDescriptor, StateVector, SVPointer],
  CoreSwapDefs: FROM "coreswapdefs" USING [PuntInfo],
  DiskDefs: FROM "diskdefs" USING [DA, RealDA],
  ImageDefs: FROM "imagedefs" USING [ImageHeader, MapItem, VersionID],
  InlineDefs: FROM "inlinedefs" USING [BITSHIFT],
  MiscDefs: FROM "miscdefs" USING [DestroyFakeModule, Zero],
  Mopcodes: FROM "mopcodes" USING [zLI4, zRFS, zSHIFT, zWFS],
  NovaOps: FROM "novaops" USING [NovaJSR],
  ProcessDefs: FROM "processdefs" USING [
    ActiveWord, ConditionVector, CV, DIW, SwatLevel],
  SegmentDefs: FROM "segmentdefs" USING [
    CopyFileToDataSegment, DataSegmentAddress, DataSegmentHandle,
    DeleteDataSegment, DeleteFileSegment, FileHandle, FileSegmentAddress,
    FileSegmentHandle, GetFileSegmentDA, NewFileSegment, Read, SwapIn,
    VMnotFree];

RunImage: PROGRAM
  IMPORTS AllocDefs, BFSDefs, BinaryDefs, DiskDefs, MiscDefs, SegmentDefs
  EXPORTS ImageDefs
  SHARES DiskDefs, ImageDefs, ProcessDefs, SegmentDefs =
  BEGIN OPEN SegmentDefs, ImageDefs;

  PageSize: CARDINAL = AltoDefs.PageSize;

  BD: PROCEDURE [CARDINAL] RETURNS [ControlDefs.FieldDescriptor] =
    MACHINE CODE BEGIN Mopcodes.zLI4; Mopcodes.zSHIFT END;
  SetBit: PROCEDURE [[0..1], POINTER, ControlDefs.FieldDescriptor] =
    MACHINE CODE BEGIN Mopcodes.zWFS END;
  GetBit: PROCEDURE [POINTER, ControlDefs.FieldDescriptor] RETURNS [[0..1]] =
    MACHINE CODE BEGIN Mopcodes.zRFS END;

  BootData: TYPE = RECORD [
    pageMap: POINTER TO PageTable,
    firstDa: DiskDefs.DA,
    initialState: ControlDefs.SVPointer,
    terminator: WORD]; -- = 0

  PageTableHeader: TYPE = RECORD [
    cfp: AltoFileDefs.CFP,
    firstpage: CARDINAL];

  PageTable: TYPE = RECORD [
    header: PageTableHeader,
    address: ARRAY [0..0) OF UNSPECIFIED];

  ptPointer: POINTER TO POINTER TO PageTable = LOOPHOLE[24B];

  InvalidImage: PUBLIC SIGNAL = CODE;

  NoRoomForLoader: PUBLIC SIGNAL = CODE;

  RunImage: PUBLIC PROCEDURE [headerseg: FileSegmentHandle] =
    BEGIN
    bootdata: BootData;
    map: ARRAY [0..16) OF UNSPECIFIED;
    ptwords: CARDINAL ← SIZE[PageTableHeader]+1;
    pt: POINTER TO PageTable;
    j: CARDINAL;
    mapi: POINTER TO MapItem;
    loader, pagetable: DataSegmentHandle;
    image: POINTER TO ImageHeader;
    state: ControlDefs.SVPointer =
      LOOPHOLE[175000B-SIZE[ControlDefs.StateVector]];
    ifile: FileHandle = headerseg.file;
    SwapIn[headerseg];
    image ← FileSegmentAddress[headerseg];
    IF image.prefix.versionident # ImageDefs.VersionID OR
```

```
    image.prefix.options # 0 THEN SIGNAL InvalidImage;
  MiscDefs.Zero[@map, 16];
  mapi ← @image.map[0];
  DO
    IF mapi.count = 0 THEN EXIT;
    ptwords ← ptwords + mapi.count;
    FOR j IN [mapi.page..mapi.page+mapi.count] DO
      SetBit[1, @map, BD[j]] ENDLOOP;
    WITH mapi SELECT FROM
      change =>
        BEGIN
        ptwords ← ptwords + 2;
        mapi ← mapi + SIZE[change MapItem];
        END;
      normal => mapi ← mapi + SIZE[normal MapItem];
      ENDCASE => ERROR;
    ENDLOOP;
  loader ← Alloc[@map, 1];
  IF loader = NIL THEN ERROR NoRoomForLoader;
  pagetable ← Alloc[@map, (ptwords+PageSize-1)/PageSize];
  IF pagetable = NIL THEN
    BEGIN DeleteDataSegment[loader]; ERROR NoRoomForLoader END;
  SetUpBootMap[ptPointer↑ ← pt ← DataSegmentAddress[pagetable], image];
  BFSDefs.MakeCFP[cfp:@pt.header.cfp, fp:@ifile.fp];
  pt.header.firstpage ← headerseg.base+1;
  state↑ ← image.prefix.state;
  bootdata ← [
    pageMap: pt,
    firstDa: FindDa[ifile, pt.header.firstpage],
    initialState: state,
    terminator: 0];
  CopyFileToDataSegment[
    MiscDefs.DestroyFakeModule[LOOPHOLE[BinaryDefs.MesaBootLoader]].seg,
    loader];
  BEGIN OPEN ProcessDefs;
  ActiveWord↑ ← DIW↑ ← InlineDefs.BITSHIFT[1,SwatLevel];
  MiscDefs.Zero[CV,SIZE[ConditionVector]];
  CoreSwapDefs.PuntInfo↑ ← NIL;
  END;
  [] ← NovaOps.NovaJSR[JSR, DataSegmentAddress[loader], @bootdata];
  END;

Alloc: PROCEDURE [map: POINTER, npages: CARDINAL]
  RETURNS [d: DataSegmentHandle] =
  BEGIN
  i, j: CARDINAL;
  d ← NIL;
  FOR i IN [4..249] DO
    IF GetBit[map,BD[i]] = 0 THEN
      BEGIN ENABLE VMnotFree => BEGIN SetBit[1, map, BD[i]]; CONTINUE END;
      info: AllocDefs.AllocInfo = [0,hard,topdown,initial,other,TRUE,FALSE];
      d ← AllocDefs.MakeDataSegment[i, npages, info];
      FOR j IN [0..npages) DO SetBit[1, map, BD[i+j]] ENDLOOP;
      IF d # NIL THEN RETURN;
      i ← i + npages-1;
      END
    ENDLOOP;
  RETURN
  END;

SetUpBootMap: PROCEDURE [pt: POINTER TO PageTable, image: POINTER TO ImageHeader] =
  BEGIN
  j: CARDINAL ← 0;
  memaddress, memcount: CARDINAL;
  mapi: POINTER TO ImageDefs.MapItem ← @image.map[0];
  DO
    IF (memcount ← mapi.count) = 0 THEN EXIT;
    memaddress ← mapi.page*PageSize;
    WITH mapi SELECT FROM
      change =>
        BEGIN
        pt.address[j] ← base*2+1;
        pt.address[j+1] ← da;
        j ← j + 2;
        mapi ← mapi + SIZE[change ImageDefs.MapItem];
        END;
```

```
        normal => mapi ← mapi + SIZE[normal ImageDefs.MapItem];
        ENDCASE;
      THROUGH [0..memcount) DO
        pt.address[j] ← memaddress;
        memaddress ← memaddress + PageSize;
        j ← j + 1;
        ENDLOOP;
      ENDLOOP;
    pt.address[j] ← 0;
    END;

  FindDa: PROCEDURE [file: FileHandle, page: CARDINAL] RETURNS [da: DiskDefs.DA] =
    BEGIN
    seg: FileSegmentHandle ← NewFileSegment[file, page, 1, Read];
    da ← DiskDefs.RealDA[SegmentDefs.GetFileSegmentDA[seg]];
    DeleteFileSegment[seg];
    RETURN
    END;

  END...
```