```
-- File: DebugCache.Mesa
-- Last edited by
--                Johnsson; August 29, 1978  9:46 AM

  DIRECTORY
    AllocDefs: FROM "allocdefs" USING [MakeSwappedIn, SwappingProcedure],
    AltoDefs: FROM "altodefs" USING [PageSize],
    AltoFileDefs: FROM "altofiledefs" USING [fillinDA, vDA],
    BFSDefs: FROM "bfsdefs" USING [ActOnPages, GetNextDA],
    ControlDefs: FROM "controldefs" USING [ControlLink],
    CoreSwapDefs: FROM "coreswapdefs",
    DebugData: FROM "debugdata" USING [altoXM, debugPilot, mdsContext, onDO],
    DebugCacheDefs: FROM "debugcachedefs" USING [
      InitMapSeg, LookupMapEntry, NewPilot31Segment,
      ReleaseMap, WritePilot31Page],
    DebuggerDefs: FROM "debuggerdefs" USING [LA],
    DebugMiscDefs: FROM "debugmiscdefs",
    DebugUsefulDefs: FROM "debugusefuldefs",
    DebugUtilityDefs: FROM "debugutilitydefs",
    DebugXMDefs: FROM "debugxmdefs" USING [XMRead, XMWrite],
    DiskDefs: FROM "diskdefs" USING [DiskRequest],
    ImageDefs: FROM "imagedefs",
    InlineDefs: FROM "inlinedefs" USING [COPY, LongDivMod, LongMult],
    Mopcodes: FROM "mopcodes" USING [zMISC, zRBL, zWBL],
    SegmentDefs: FROM "segmentdefs" USING [
      DefaultBase, DeleteFileSegment, FileHandle, FileSegmentAddress,
      FileSegmentHandle, GetFileSegmentDA, InsertFile, LockFile,
      NewFileSegment, Read, SetFileSegmentDA, SwapIn, Unlock, Write],
    SystemDefs: FROM "systemdefs" USING [AllocatePages, FreePages],
    VMMapLog: FROM "vmmaplog" USING [Entry, PilotFID];

DebugCache: PROGRAM
  IMPORTS AllocDefs, BFSDefs, DebugCacheDefs, DDptr: DebugData,
    DebugXMDefs, SystemDefs, SegmentDefs
  EXPORTS DebugCacheDefs, DebugMiscDefs, DebugUsefulDefs, DebugUtilityDefs
  SHARES DiskDefs, SegmentDefs =

  BEGIN

  -- move this to a utility somewhere

  Bound: PUBLIC PROCEDURE [p: UNSPECIFIED] RETURNS [BOOLEAN] =
    BEGIN
    RETURN[LOOPHOLE[p, ControlDefs.ControlLink].tag # unbound]
    END;

  LA: TYPE = DebuggerDefs.LA;

  PageSize: CARDINAL = AltoDefs.PageSize;

  LAMult: PROCEDURE [CARDINAL, CARDINAL] RETURNS [LA] =
    LOOPHOLE[InlineDefs.LongMult];

  -- DO/Pilot Stuff

  MapFlags: TYPE = MACHINE DEPENDENT RECORD [
    LogSE, W, D, Ref: BOOLEAN];

  VacantFlags: MapFlags = [FALSE, TRUE, TRUE, FALSE];
  CleanFlags: MapFlags = [FALSE, FALSE, FALSE, FALSE];

  MapEntry: TYPE = MACHINE DEPENDENT RECORD [
    flags: MapFlags,
    realPage: [0..7777B]];

  Vacant: MapEntry = [VacantFlags,0];
  Clean: MapEntry = [CleanFlags,0];

  MapSwapBase: CARDINAL = 16384-256;

  SETF: PROCEDURE [CARDINAL, MapEntry] RETURNS [MapEntry] =
    MACHINE CODE BEGIN Mopcodes.zMISC, 1 END;

  RBL: PROCEDURE [LA] RETURNS [UNSPECIFIED] =
    MACHINE CODE BEGIN Mopcodes.zRBL, 0 END;
```

```
  WBL: PROCEDURE [UNSPECIFIED, LA] =
    MACHINE CODE BEGIN Mopcodes.zWBL, 0 END;

  -- Cache of user memory pages

  CoreSegmentObject: TYPE = RECORD [
    address: POINTER,
    lastused: CARDINAL,
    mempage: CARDINAL,
    inuse: BOOLEAN,
    dirty: BOOLEAN,
    body:
      SELECT type: * FROM
        alto => [
          segment: SegmentDefs.FileSegmentHandle],
        pilot31 => [
          vda: CARDINAL,
          fid: VMMapLog.PilotFID,
          filepage: CARDINAL],
        ENDCASE];

  CoreSegment: TYPE = POINTER TO CoreSegmentObject;

  maxsegments: CARDINAL = 8;        -- number of pages to keep in core

  CS: ARRAY [0..maxsegments) OF CoreSegmentObject;

  CurrentUseValue: CARDINAL;
  CoreFile: SegmentDefs.FileHandle;
  DAs: ARRAY [-1..256] OF AltoFileDefs.vDA;
  DiskAddresses: DESCRIPTOR FOR ARRAY OF AltoFileDefs.vDA ←
    DESCRIPTOR[@DAs[0],256];
  InitCoreCache: PROCEDURE [file: SegmentDefs.FileHandle] =
    BEGIN
    i: CARDINAL;
    CurrentUseValue ← 0;
    FOR i IN [0..maxsegments) DO CS[i].inuse ← FALSE ENDLOOP;
    CoreFile ← file;
    SegmentDefs.LockFile[CoreFile];
    END;


  FlushCoreCache: PUBLIC AllocDefs.SwappingProcedure =
    BEGIN OPEN SegmentDefs;
    did: BOOLEAN ← FALSE;
    i: CARDINAL ← 0;
    cs: CoreSegment;
--    CacheSwap.proc ← AllocDefs.CantSwap;
    FOR i IN [0..maxsegments) DO
      cs←@CS[i];
      IF cs.inuse THEN
        BEGIN
        FlushCS[cs];
        did ← TRUE;
        END;
      ENDLOOP;
    CurrentUseValue←0;
    RETURN[did]
    END;


  FlushCS: PROCEDURE [cs: CoreSegment] =
    BEGIN OPEN SegmentDefs;
    WITH cs SELECT FROM
      alto =>
        BEGIN
        IF dirty THEN segment.write ← TRUE;
        Unlock[segment];
        DeleteFileSegment[segment];
        END;
      pilot31 =>
        BEGIN
        IF dirty THEN
          DebugCacheDefs.WritePilot31Page[address, vda, fid, filepage];
        SystemDefs.FreePages[address];
        END;
```

```
        ENDCASE;
    cs.inuse ← FALSE;
    END;


  NewSwateeSegment: PROCEDURE [mempage: CARDINAL, cs: CoreSegment] =
    BEGIN OPEN SegmentDefs;
    filepage: CARDINAL = SELECT mempage FROM
      IN[2..253] => mempage, 1 => 254, ENDCASE => 255;
    seg: FileSegmentHandle = NewFileSegment[CoreFile, filepage, 1, Read];
    SetFileSegmentDA[seg, DiskAddresses[filepage]];
    AllocDefs.MakeSwappedIn[seg, DefaultBase,
      [0,hard,bottomup,initial,other,TRUE,FALSE]];
    DiskAddresses[filepage] ← GetFileSegmentDA[seg];
    cs↑ ← [
      address: FileSegmentAddress[seg],
      inuse: TRUE,
      dirty: FALSE,
      lastused:,
      mempage: mempage,
      body: alto[seg]];
    RETURN
    END;


  NonExistentMemoryPage: PUBLIC SIGNAL [page: CARDINAL] = CODE;
  InvalidAddress: PUBLIC SIGNAL [a: LA] = CODE;

  NewVMSegment: PROCEDURE [mempage: CARDINAL, cs: CoreSegment] =
    BEGIN ENABLE UNWIND => DebugCacheDefs.ReleaseMap[];
    entry: POINTER TO VMMapLog.Entry = DebugCacheDefs.LookupMapEntry[mempage];
    offset: CARDINAL;
    IF entry = NIL THEN GOTO notThere;
    offset ← mempage-entry.page;
    BEGIN ENABLE UNWIND => DebugCacheDefs.ReleaseMap[];
    WITH e:entry SELECT FROM
      alto31 =>
        BEGIN OPEN SegmentDefs;
        seg: FileSegmentHandle = NewFileSegment[
          InsertFile[LOOPHOLE[@e.fp],Read+Write],
          e.filePage+offset, 1, Read];
        cs.body ← alto[seg];
        SwapIn[seg];
        cs.address ← FileSegmentAddress[seg];
        END;
      pilot31 =>
        BEGIN
        cs.body ← pilot31[
          vda: e.vda+offset,
          fid: e.fid,
          filepage: e.filePage+offset];
        cs.address ← DebugCacheDefs.NewPilot31Segment[
          e.vda+offset, e.fid, e.filePage+offset];
        END;
      ENDCASE => GOTO notThere;
    END; -- ELBANE
    DebugCacheDefs.ReleaseMap[];
    cs.mempage ← mempage;
    cs.inuse ← TRUE;
    cs.dirty ← FALSE;
    EXITS notThere =>
      BEGIN
      DebugCacheDefs.ReleaseMap[];
      ERROR NonExistentMemoryPage[mempage];
      END;
    END;


  GetCS: PROCEDURE [mempage: CARDINAL] RETURNS [sp: CoreSegment] =
    BEGIN
    minUseVal: CARDINAL ← CurrentUseValue;
    minUseIndex: CARDINAL ← 0;
    i: CARDINAL;

    BEGIN
    FOR i IN [0..maxsegments) DO
```

```
      sp ← @CS[i];
      IF ~sp.inuse THEN GO TO newseg;
      IF sp.mempage = mempage THEN EXIT;
      IF sp.lastused < minUseVal THEN
        BEGIN minUseVal←sp.lastused; minUseIndex←i END;
      REPEAT FINISHED =>
        BEGIN
        FOR i IN [0..maxsegments) DO
          CS[i].lastused ← CS[i].lastused - minUseVal;
          ENDLOOP;
        CurrentUseValue ← CurrentUseValue - minUseVal;
        FlushCS[sp ← @CS[minUseIndex]];
        GO TO newseg;
        END
      ENDLOOP;
    EXITS newseg =>
      BEGIN
      cso: CoreSegmentObject;
      IF mempage IN [0..253] THEN NewSwateeSegment[mempage, @cso]
      ELSE NewVMSegment[mempage, @cso];
      FOR i IN [0..maxsegments) DO
        sp ← @CS[i];
        IF ~sp.inuse THEN BEGIN sp↑ ← cso; EXIT END;
        REPEAT FINISHED => ERROR
        ENDLOOP;
      END;
    END;
    sp.lastused ← CurrentUseValue ← CurrentUseValue+1;
--    CacheSwap.proc ← FlushCoreCache;
    RETURN[sp];
    END;

  LongREAD: PUBLIC PROCEDURE [a: LONG POINTER] RETURNS [UNSPECIFIED] =
    BEGIN
    mempage, offset: CARDINAL;
    [mempage, offset] ← InlineDefs.LongDivMod[LOOPHOLE[a],PageSize];
    DO
      IF mempage > 255 THEN
        BEGIN
        real: BOOLEAN;
        [real, mempage] ← RealMemory[mempage];
        IF real THEN RETURN[ReadVM[mempage, offset]];
        EXIT;
        END
      ELSE
        BEGIN
        IF DDptr.debugPilot AND DDptr.onDO THEN
          BEGIN
          mempage ← RemapLowPage[mempage];
          IF mempage > 255 THEN LOOP;
          END;
        IF mempage > 253 THEN RETURN[MEMORY[mempage*PageSize+offset]];
        EXIT;
        END;
      ENDLOOP;
    RETURN [(GetCS[mempage].address + offset)↑];
    END;


  LongWriteCommon: PROCEDURE [a: LONG POINTER, v: UNSPECIFIED, setdbit: BOOLEAN] =
    BEGIN
    mempage, offset: CARDINAL;
    s: CoreSegment;
    [mempage, offset] ← InlineDefs.LongDivMod[LOOPHOLE[a],PageSize];
    IF DDptr.debugPilot AND DDptr.onDO AND setdbit THEN SetDirty[mempage];
    DO
      IF mempage > 255 THEN
        BEGIN
        real: BOOLEAN;
        [real, mempage] ← RealMemory[mempage];
        IF real THEN BEGIN WriteVM[mempage, offset, v]; RETURN END;
        EXIT;
        END
      ELSE
        BEGIN
        IF DDptr.debugPilot AND DDptr.onDO THEN
```

```
        BEGIN
        mempage ← RemapLowPage[mempage];
        IF mempage > 255 THEN LOOP;
        END;
      IF mempage > 253 THEN
        BEGIN MEMORY[mempage*PageSize+offset] ← v; RETURN END;
      EXIT;
      END;
    ENDLOOP;
  ((s←GetCS[mempage]).address + offset)↑ ← v;
  s.dirty ← TRUE;
  RETURN
  END;

LongWRITE: PUBLIC PROCEDURE [a: LONG POINTER, v: UNSPECIFIED] =
  BEGIN
  LongWriteCommon[a, v, TRUE];
  RETURN
  END;

LongWRITEClean: PUBLIC PROCEDURE [a: LONG POINTER, v: UNSPECIFIED] =
  BEGIN
  LongWriteCommon[a, v, FALSE];
  RETURN
  END;

RemapLowPage: PROCEDURE [mempage: CARDINAL] RETURNS [CARDINAL] =
  BEGIN
  m: MapEntry = ReadMap[MapSwapBase+mempage];
  rp: CARDINAL = m.realPage;
  vp: CARDINAL;
  IF m.flags = VacantFlags THEN RETURN[MapSwapBase+mempage]
  ELSE IF ReadMap[mempage].m.realPage = rp THEN RETURN[mempage];
  FOR vp IN [0..253] DO
    IF ReadMap[vp].m.realPage = rp THEN RETURN[vp];
    ENDLOOP;
  RETURN[MapSwapBase+mempage];
  END;

AREAD, MREAD, SREAD: PUBLIC PROCEDURE [a: UNSPECIFIED] RETURNS [UNSPECIFIED] =
  BEGIN
  mds: LA = LAMult[DDptr.mdsContext, PageSize];
  RETURN[LongREAD[mds.lp+CARDINAL[a]]]
  END;


AWRITE, MWRITE, SWRITE: PUBLIC PROCEDURE [a: UNSPECIFIED, v: UNSPECIFIED] =
  BEGIN
  mds: LA = LAMult[DDptr.mdsContext, PageSize];
  LongWRITE[mds.lp+CARDINAL[a], v];
  RETURN
  END;

RealMemory: PROCEDURE [mempage: CARDINAL] RETURNS [BOOLEAN, CARDINAL] =
  BEGIN
  IF DDptr.onDO THEN
    BEGIN
    flags: MapFlags;
    rp, vp: CARDINAL;
    [flags,rp] ← ReadMap[mempage].m;
    IF flags = VacantFlags THEN RETURN[FALSE, mempage];
    FOR vp IN [0..253] DO
      IF ReadMap[vp].m.realPage = rp THEN RETURN[FALSE, vp];
      ENDLOOP;
    RETURN[TRUE, mempage]
    END;
  IF DDptr.altoXM THEN RETURN[mempage < 1024, mempage];
  ERROR NonExistentMemoryPage[mempage];
  END;

SwappedIn: PUBLIC PROCEDURE [mempage: CARDINAL] RETURNS [BOOLEAN] =
  BEGIN
  IF ~DDptr.onDO THEN RETURN[FALSE];
  IF mempage <= 255 THEN mempage ← mempage + MapSwapBase;
  RETURN[ReadMap[mempage].m.flags # VacantFlags]
  END;
```

```
ReadVM: PROCEDURE [mempage, offset: CARDINAL] RETURNS [UNSPECIFIED] =
  BEGIN
  p: LA = [LP[LAMult[mempage,PageSize].lp+offset]];
  IF DDptr.onDO THEN RETURN[RBL[p]];
  IF DDptr.altoXM AND Bound[DebugXMDefs.XMRead] THEN
    RETURN[DebugXMDefs.XMRead[LOOPHOLE[p]]];
  ERROR InvalidAddress[p];
  END;

WriteVM: PROCEDURE [mempage, offset: CARDINAL, v: UNSPECIFIED] =
  BEGIN
  p: LA = [LP[LAMult[mempage,PageSize].lp+offset]];
  IF DDptr.onDO THEN
    BEGIN
    oldm: MapEntry ← SETF[mempage, Vacant];
    [] ← SETF[mempage, Clean];
    WBL[v, p];
    oldm.flags.Ref ← TRUE;
    IF ~oldm.flags.W THEN oldm.flags.D ← TRUE;  -- wait for hardware change
    [] ← SETF[mempage, oldm];
    RETURN
    END;
  IF DDptr.altoXM AND Bound[DebugXMDefs.XMWrite] THEN
    BEGIN DebugXMDefs.XMWrite[LOOPHOLE[p], v]; RETURN END;
  ERROR InvalidAddress[p];
  END;

ReadMap: PROCEDURE [mempage: CARDINAL] RETURNS [m: MapEntry] =
  BEGIN
  m ← SETF[mempage, Clean];
  [] ← SETF[mempage, m];
  RETURN
  END;

SetDirty: PROCEDURE [mempage: CARDINAL] =
  BEGIN
  m: MapEntry;
  IF mempage <= 255 THEN mempage ← mempage + MapSwapBase;
  m ← SETF[mempage, Clean];
  m.flags.D ← m.flags.Ref ← TRUE;
  m.flags.W ← FALSE;  -- wait for hardware change
  [] ← SETF[mempage, m];
  RETURN
  END;


-- initialization

InitializeDebuggerFiles: PUBLIC PROCEDURE [debuggee: SegmentDefs.FileHandle] =
  BEGIN OPEN AltoFileDefs, DiskDefs, SegmentDefs;
  p: POINTER;
  diskrequest: DiskRequest;
  diskrequest ← DiskRequest [
    ca: SystemDefs.AllocatePages[1],
    fixedCA: TRUE,
    da: @DiskAddresses[0],
    fp: @debuggee.fp,
    firstPage: 0,
    lastPage: 255,
    action: ReadD,
    lastAction: ReadD,
    signalCheckError: FALSE,
    option: update[BFSDefs.GetNextDA]];
  p ← LOOPHOLE[@DiskAddresses[0]-1];
  p↑ ← fillinDA;
  InlineDefs.COPY[from: p, to: p+1, nwords: 257];
  DiskAddresses[0] ← debuggee.fp.leaderDA;
  [] ← BFSDefs.ActOnPages[LOOPHOLE[@diskrequest]];
  SystemDefs.FreePages[diskrequest.ca];
  DebugCacheDefs.InitMapSeg[debuggee];
  InitCoreCache[debuggee];
  END;
```

```
END...
```