```
-- file: DumpUtils.Mesa
-- Edited by:
--            Sandman on May 23, 1978  8:32 AM
--            Barbara on July 31, 1978  4:44 PM
--            Johnsson on August 29, 1978  11:05 AM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [wordlength],
  ControlDefs: FROM "controldefs" USING [
    BytePC, FieldDescriptor, FrameHandle, GlobalFrameHandle, MaxParmsInStack,
    NullFrame, WordPC],
  CoreSwapDefs: FROM "coreswapdefs" USING [SVPointer],
  DebugBreakptDefs: FROM "debugbreakptdefs" USING [
    CodeToSourceIndex, PrintLocation, SourceFileMissing],
  DebuggerDefs: FROM "debuggerdefs" USING [
    Display, FieldContext, FRPointer, fullbitaddress, LA, MainBTI, SA,
    SeiHandle, SOPointer, SymbolObject, SymFrameHandle, VersionStamp],
  DebugMiscDefs: FROM "debugmiscdefs" USING [WriteCharZ, WriteEOL],
  DebugSymbolDefs: FROM "debugsymboldefs" USING [
    DAcquireSymbolTable, DReleaseSymbolTable, HandleForBase,
    TableForString],
  DebugUtilityDefs: FROM "debugutilitydefs" USING [
    LongREAD, MREAD, ReadCodeByte, SREAD],
  IODefs: FROM "iodefs" USING [WriteChar, WriteOctal, WriteString],
  Mopcodes: FROM "mopcodes" USING [zBRK, zNOOP, zRFS],
  StringDefs: FROM "stringdefs" USING [
    AppendSubString, SubString, SubStringDescriptor],
  SymbolTableDefs: FROM "symboltabledefs" USING [
    NoSymbolTable, SymbolTableBase, SymbolTableHandle],
  SymDefs: FROM "symdefs" USING [
    BitAddress, BodyRecord, BTIndex, BTNull, CBTIndex, CBTNull, CTXIndex,
    CTXNull, HTIndex, HTNull, ISEIndex, ISENull, SEIndex, SENull,
    TransferMode];

DumpUtils: PROGRAM
IMPORTS DebugBreakptDefs, DebuggerDefs, DebugMiscDefs, DebugSymbolDefs,
  DebugUtilityDefs, IODefs, StringDefs, SymbolTableDefs
EXPORTS DebuggerDefs =

BEGIN

SeiHandle: TYPE = DebuggerDefs.SeiHandle;
SymFrameHandle: TYPE = DebuggerDefs.SymFrameHandle;
SOPointer: TYPE = DebuggerDefs.SOPointer;
FRPointer: TYPE = DebuggerDefs.FRPointer;
SymbolTableBase: TYPE = SymbolTableDefs.SymbolTableBase;

WriteBlanks: PUBLIC PROCEDURE [n: CARDINAL] =
  BEGIN
  THROUGH [0..n) DO IODefs.WriteChar[' ']; ENDLOOP;
  RETURN
  END;

WriteBodyName: PUBLIC PROCEDURE [f: SymFrameHandle, printFrames: BOOLEAN] =
  BEGIN  -- type id of body associated with frame
  sei: SymDefs.ISEIndex ← BodySei[f];
  WriteTransferName[SeiHandle[stbase: f.stbase, sei: sei], printFrames,
    IF printFrames THEN f.faddr ELSE ControlDefs.NullFrame,
    DebugUtilityDefs.MREAD[@f.faddr.accesslink]];
  RETURN
  END;

BodySei: PUBLIC PROCEDURE [f: SymFrameHandle] RETURNS [SymDefs.ISEIndex] =
  -- finds sei for body corresponding to frame f
  BEGIN OPEN SymDefs, f.stbase;
  cbti: CBTIndex ← PcToCBTI[f.stbase, FrameRelBPC[f.faddr]];
  RETURN[IF cbti # CBTNull THEN (bb+cbti).id ELSE ISENull]
  END;

PcToBTI: PUBLIC PROCEDURE [stbase: SymbolTableBase, pc: ControlDefs.BytePC]
  RETURNS [SymDefs.BTIndex] =
  BEGIN OPEN stbase, SymDefs; -- maps pc into deepest body table index
  btlimit: BTIndex = LOOPHOLE[stHandle.bodyBlock.size];
  cbti: BTIndex;
  bti: SymDefs.BTIndex ← DebuggerDefs.MainBTI;
  IF pc = ControlDefs.BytePC[0] THEN
```

```
    RETURN[LOOPHOLE[bti+SIZE[Other BodyRecord]]]; -- main body
  UNTIL bti = btlimit DO
    WITH body: (bb+bti).info SELECT FROM
      External => IF pc IN [body.origin..body.origin+body.bytes) THEN
        BEGIN
        DO
          cbti ← (bb+bti).firstSon;
          IF cbti = BTNull THEN RETURN[bti];
          DO
            WITH (bb+cbti).info SELECT FROM
              External => IF pc IN [origin..origin+bytes) THEN
                BEGIN bti ← cbti; EXIT END;
              ENDCASE => ERROR;
            IF (bb+cbti).link.which = parent THEN RETURN[bti];
            cbti ← (bb+cbti).link.index;
            ENDLOOP;
          ENDLOOP;
        END;
      ENDCASE => ERROR;
    bti ← bti+ (WITH b: (bb+bti) SELECT FROM
      Callable => (WITH b SELECT FROM
        Inner => SIZE[Inner Callable BodyRecord],
        ENDCASE => SIZE[Outer Callable BodyRecord]),
      ENDCASE => SIZE[Other BodyRecord]);
    ENDLOOP;
  RETURN [BTNull];
  END;

PcToCBTI: PUBLIC PROCEDURE [stbase: SymbolTableBase, pc: ControlDefs.BytePC]
  RETURNS [SymDefs.CBTIndex] =
  BEGIN OPEN stbase, SymDefs; -- maps pc into first callable bti
  bti: BTIndex ← PcToBTI[stbase, pc];
  DO
    IF bti = BTNull THEN EXIT;
    WITH b: (bb+bti) SELECT FROM
      Callable => RETURN[LOOPHOLE[bti]];
      ENDCASE =>
        BEGIN
        UNTIL (bb+bti).link.which = parent DO
          bti ← (bb+bti).link.index;
          ENDLOOP;
        bti ← (bb+bti).link.index;
        END;
    ENDLOOP;
  RETURN[CBTNull];
  END;

AmIaRecord:  PUBLIC SIGNAL RETURNS [BOOLEAN] = CODE;

DumpCtxList: PUBLIC PROCEDURE [c: SymDefs.CTXIndex, sbase: SymbolTableDefs.SymbolTableBase, there: BOOL
**EAN, faddr: fullbitaddress, frp: FRPointer] =
  BEGIN OPEN DebugSymbolDefs, sbase;
  sh: SymbolTableDefs.SymbolTableHandle;
  sei: SymDefs.ISEIndex;
  SO: DebuggerDefs.SymbolObject;
  sop: SOPointer ← @SO;
  isrecord: BOOLEAN ← SIGNAL AmIaRecord;
  hasname: BOOLEAN;
  t: SymDefs.TransferMode;
  filename: STRING ← [40];
  desc: StringDefs.SubStringDescriptor;
  filess: StringDefs.SubString ← @desc;
  newctx: SymDefs.CTXIndex;
  includedVersion: DebuggerDefs.VersionStamp;

  IF c = SymDefs.CTXNull THEN RETURN;
  DebuggerDefs.InitSOP[sop];
  WITH (ctxb+c) SELECT FROM
    included =>
      BEGIN
      IF ~ctxcomplete THEN
        BEGIN
        sh ← HandleForBase[sbase];
        SubStringForHash[filess, (mdb+ctxmodule).mdhti];
        StringDefs.AppendSubString[filename, filess];
        newctx ← ctxmap;
```

```
        includedVersion ← (mdb+ctxmodule).mdStamp;
        DReleaseSymbolTable[sbase];
        sbase ← DAcquireSymbolTable[TableForString[filename
             !SymbolTableDefs.NoSymbolTable => GOTO continue]
          !SymbolTableDefs.NoSymbolTable => GOTO continue];
        IF ~EquivalentVersions[sbase.stHandle.version, includedVersion] THEN
          BEGIN DReleaseSymbolTable[sbase]; GOTO continue; END;
        DumpCtxList[newctx, sbase, there, faddr, frp];
        DReleaseSymbolTable[sbase];
        [] ← DAcquireSymbolTable[sh];
        RETURN
        END;
      EXITS
        continue =>
          BEGIN IODefs.WriteChar['?]; sbase ← DAcquireSymbolTable[sh]; END;
      END;
    ENDCASE;
  IF (sei ← FirstCtxSe[c]) = SymDefs.ISENull THEN RETURN;
  IF (seb+sei).ctxnum # c THEN sei ← NextSe[sei];
  DebugMiscDefs.WriteCharZ[frp.startchar];
  FOR sei ← sei, NextSe[sei] UNTIL sei = SymDefs.ISENull DO
    hasname ← (seb+sei).htptr # SymDefs.HTNull;
    IF ~(seb+sei).constant AND (isrecord OR hasname) AND ~(seb+sei).linkSpace THEN
      BEGIN
      SO ← DebuggerDefs.SymbolObject[stbase: sbase, baddr: faddr, sei: sei,
        tsei: (seb+sei).idtype, space: 0, there: there];
      t ← XferMode[sop.tsei];
      IF (seb+sei).writeonce AND (t = signal OR t = error) THEN GOTO skipit;
      IF ~isrecord OR (TypeForm[sop.tsei] # union AND hasname) THEN
        WriteSymDef[sop, frp]
        ELSE IF ~frp.firstsym THEN DebugMiscDefs.WriteCharZ[frp.intersym];
      DebuggerDefs.Display[sop, frp, FALSE];
      EXITS
        skipit => NULL;
      END;
    ENDLOOP;
  IF ~frp.firstsym THEN DebugMiscDefs.WriteCharZ[frp.termchar];
  RETURN
  END;


DumpValsFromState: PUBLIC PROCEDURE [bsei: SymDefs.ISEIndex, frp: FRPointer,
  symbase: SymbolTableDefs.SymbolTableBase, ps: CoreSwapDefs.SVPointer] =
  BEGIN -- if the bsei is a SIGNAL or ERROR then dumps input context
  OPEN symbase;
  typein, typeout: SymDefs.SEIndex;
  tsei: SymDefs.SEIndex ← (seb+bsei).idtype;
  t: SymDefs.TransferMode;
  [typein, typeout] ← TransferTypes[tsei];
  t ← XferMode[tsei];
  tsei ← IF (t = signal OR t = error) THEN typein ELSE typeout;
  IF tsei = SymDefs.SENull THEN RETURN;
  DebugMiscDefs.WriteEOL[];
  DumpCtxFromState[tsei, frp, symbase, ps, (t = signal OR t = error)];
  RETURN
  END;

DumpCtxFromState: PUBLIC PROCEDURE [tsei: SymDefs.SEIndex, frp: FRPointer,
  symbase: SymbolTableDefs.SymbolTableBase, ps: CoreSwapDefs.SVPointer,
  sigerr: BOOLEAN] =
  BEGIN OPEN DebuggerDefs, symbase; -- tsei is a record sei
  sei: SymDefs.ISEIndex;
  SO: SymbolObject;
  sop: SOPointer ← @SO;
  nwords: CARDINAL;
  IF tsei = SymDefs.SENull THEN RETURN;
  InitSOP[sop];
  sop.stbase ← symbase;
  nwords ← WordsForType[tsei];
  sop.baddr ← fullbitaddress[bd: 0, wd: short[shortAddr:LOOPHOLE[
    IF nwords > (IF sigerr THEN 1 ELSE ControlDefs.MaxParmsInStack)
      THEN DebugUtilityDefs.SREAD[@ps.stk[0]] ELSE @ps.stk[0]]]];
  FOR sei ← FirstCtxSe[FieldContext[sop.stbase,tsei]], NextSe[sei]
  UNTIL sei = SymDefs.ISENull DO
    IF ~(seb+sei).constant THEN
      BEGIN
```

```
      sop.sei ← sei;
      sop.tsei ← (seb+sei).idtype;
      WriteSymDef[sop, frp];
      sop.sei ← SymDefs.ISENull;
      Display[sop, frp, FALSE];
      WITH s:sop.baddr SELECT FROM
        short => s.wd ← short[shortAddr: [s.shortAddr + WordsForType[sop.tsei]]];
        ENDCASE => ERROR;
      END;
    ENDLOOP;
  IF ~frp.firstsym THEN DebugMiscDefs.WriteCharZ[frp.termchar];
  RETURN
  END;

WriteSymDef: PUBLIC PROCEDURE [sop: SOPointer, frp: FRPointer] =
  BEGIN -- types symbol definiton prefix
  IF ~frp.symid THEN RETURN;
  IF ~frp.firstsym THEN DebugMiscDefs.WriteCharZ[frp.intersym];
  THROUGH [0..frp.indentation) DO IODefs.WriteChar[' ] ENDLOOP;
  WriteSeiHandle[SeiHandle[stbase: sop.stbase, sei: sop.sei]];
  DebugMiscDefs.WriteCharZ[frp.symdelim];
  RETURN
  END;

FrameRelBPC: PUBLIC PROCEDURE [f: ControlDefs.FrameHandle]
  RETURNS [ControlDefs.BytePC] =
  BEGIN -- returns cseg-relative pc from frame
  wpc: ControlDefs.WordPC ← DebugUtilityDefs.MREAD[@f.pc];
  RETURN [[ABS[wpc]*2 + (IF wpc < 0 THEN 1 ELSE 0)]]
  END;

WriteFrameLocus: PUBLIC PROCEDURE [f: SymFrameHandle, psource: BOOLEAN] =
  BEGIN
  WriteBodyName[f, TRUE];
  IF psource THEN WriteSource[
    DebugUtilityDefs.MREAD[@f.faddr.accesslink], FrameRelBPC[f.faddr], TRUE];
  RETURN
  END;

WriteSource: PUBLIC PROCEDURE [f: ControlDefs.GlobalFrameHandle,
  pc: ControlDefs.BytePC, scroll: BOOLEAN] =
  BEGIN OPEN DebugBreakptDefs;
  na: STRING = " not available"L;
  index: CARDINAL;
  source: BOOLEAN ← TRUE;
  DebugMiscDefs.WriteEOL[];
  IODefs.WriteString[" Source: "L];
  IF pc # 0 THEN
    BEGIN OPEN DebugUtilityDefs;
    IF ReadCodeByte[f, pc] # Mopcodes.zBRK THEN pc ← LOOPHOLE[pc-1];
    IF ReadCodeByte[f, pc] = Mopcodes.zNOOP THEN pc ← LOOPHOLE[pc-1];
    END;
  index ← CodeToSourceIndex[f, pc
    !SourceFileMissing =>
       BEGIN OPEN IODefs;
       WriteString[sourcename];
       WriteString[na];
       source ← FALSE;
       CONTINUE;
       END];
  IF source THEN PrintLocation[f, index, scroll];
  RETURN
  END;

WriteSubString: PUBLIC PROCEDURE [s: StringDefs.SubString] =
  BEGIN -- type substring
  i: CARDINAL;
  FOR i IN [0..s.length)
    DO IODefs.WriteChar[s.base[s.offset + i]] ENDLOOP;
  RETURN
  END;

WriteSeiHandle: PUBLIC PROCEDURE [sh: SeiHandle] =
  BEGIN -- type id of sei
  OPEN sh.stbase;
  desc: StringDefs.SubStringDescriptor;
```

```
  ss: StringDefs.SubString ← @desc;
  hti: SymDefs.HTIndex;
  IF sh.sei = SymDefs.ISENull THEN RETURN;
  IF (hti ← (seb+sh.sei).htptr) = SymDefs.HTNull THEN
    BEGIN IODefs.WriteString["(anon)"L]; RETURN END;
  SubStringForHash[ss, hti];
  WriteSubString[ss];
  RETURN
  END;

WriteTransferName: PUBLIC PROCEDURE[sh: SeiHandle, fullname: BOOLEAN,
  frame: ControlDefs.FrameHandle, gframe: ControlDefs.GlobalFrameHandle] =
  BEGIN OPEN IODefs, sh.stbase;
  IF sh.sei = SymDefs.ISENull
    THEN BEGIN WriteString[" (priv in "L]; fullname ← TRUE; END
  ELSE
    BEGIN
    WriteSeiHandle[sh];
    IF frame # ControlDefs.NullFrame THEN
      BEGIN WriteString[", L: "L]; WriteOctal[frame]; END;
    IF fullname THEN WriteString[" (in "L];
    END;
  IF fullname THEN
    BEGIN
    WriteSeiHandle[[stbase: sh.stbase, sei: (bb+DebuggerDefs.MainBTI).id]];
    WriteString[", G:"L];
    WriteOctal[gframe];
    WriteChar[')];
    END;
  RETURN
  END;

fullbitaddress: TYPE = DebuggerDefs.fullbitaddress;
LA: TYPE = DebuggerDefs.LA;
SA: TYPE = DebuggerDefs.SA;

GetValue: PUBLIC PROCEDURE [sop: SOPointer] RETURNS [UNSPECIFIED] =
  BEGIN RETURN [GetValueN[sop,0]] END;

GetValueN: PUBLIC PROCEDURE [sop: SOPointer, n: CARDINAL]
  RETURNS [UNSPECIFIED]=
  BEGIN OPEN sop.stbase;
  a: fullbitaddress;
  fd: ControlDefs.FieldDescriptor;
  v: UNSPECIFIED;

  IF sop.sei # SymDefs.SENull THEN
    BEGIN
    IF (seb+sop.sei).constant THEN RETURN [(seb+sop.sei).idvalue];
    --space # 0 means already adjusted to exact size
    fd.size ← IF sop.space # 0 THEN sop.space
      ELSE (seb+sop.sei).idinfo MOD AltoDefs.wordlength;
    a ← addbitaddrs[sop.baddr, fullsymaddress[sop]];
    END
  ELSE
    BEGIN
    a ← sop.baddr; fd.size ← sop.space;
    END;
  fd.offset ← 0;
  fd.posn ← a.bd;
  WITH a SELECT FROM
    short => IF sop.there THEN v ← DebugUtilityDefs.MREAD[shortAddr + n]
        ELSE v ← (LOOPHOLE[shortAddr, POINTER]+n)↑;
    long => IF sop.there THEN v ← DebugUtilityDefs.LongREAD[longAddr.lp+n];
    ENDCASE => ERROR;
  RETURN[READFIELD[v, fd]]
  END;

READFIELD: PROCEDURE [v: UNSPECIFIED, fd: ControlDefs.FieldDescriptor] RETURNS [UNSPECIFIED] =
  BEGIN
  ReadField: PROCEDURE [POINTER, ControlDefs.FieldDescriptor] RETURNS [UNSPECIFIED] =
    MACHINE CODE BEGIN Mopcodes.zRFS END;
  RETURN[ReadField[@v, fd]]
  END;

fullsymaddress: PUBLIC PROCEDURE [sop: SOPointer] RETURNS [fullbitaddress] =
```

```
  BEGIN OPEN sop.stbase;
  b: SymDefs.BitAddress ←
    IF sop.sei # SymDefs.SENull AND ~(seb+sop.sei).constant
      THEN (seb+sop.sei).idvalue
    ELSE SymDefs.BitAddress[0,0];
  RETURN [fullbitaddress[bd: b.bd, wd: short[shortAddr: [b.wd]]]]
  END;

addbitaddrs: PUBLIC PROCEDURE [a,b: fullbitaddress]
  RETURNS [c: fullbitaddress] =
  BEGIN
  t: CARDINAL = a.bd + b.bd;
  tDiv: CARDINAL = t/AltoDefs.wordlength;
  aSA: SA;
  aLA: LA;
  WITH a SELECT FROM
    short =>
      BEGIN
      aSA ← shortAddr;
      WITH b SELECT FROM
        short => c.wd ← short[shortAddr: [shortAddr + tDiv + aSA]];
        long => c.wd ← long[longAddr: LA[LI[li:longAddr.li+tDiv + aSA]]];
        ENDCASE;
      END;
    long =>
      BEGIN
      aLA ← longAddr;
      WITH b SELECT FROM
        short => c.wd ← long[longAddr: LA[LI[li:aLA.li+ shortAddr+tDiv]]];
        long => c.wd ← long[longAddr: LA[LI[li:aLA.li+longAddr.li+ tDiv]]];
        ENDCASE;
      END;
    ENDCASE;
  c.bd ← t MOD AltoDefs.wordlength;
  RETURN
  END;

EquivalentVersions: PUBLIC PROCEDURE [v1, v2: DebuggerDefs.VersionStamp]
  RETURNS [BOOLEAN] =
  BEGIN
  RETURN[v1.zapped OR v2.zapped OR v1 = v2]
  END;

InitSOP: PUBLIC PROCEDURE [sop: SOPointer]=
  BEGIN
  sop↑ ← [stbase: NIL, sei: SymDefs.ISENull, tsei: SymDefs.SENull,
    there: TRUE, space: 0, baddr: fullbitaddress[bd: 0,
    wd: short[shortAddr:[0]]]];
  END;


END ...
```