```
-- file: DumpVars.Mesa
-- Edited by:
--             Johnsson, August 30, 1978  6:02 PM
--             Sandman, May 23, 1978  8:44 AM
--             Barbara, July 31, 1978  3:45 PM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [VMLimit, wordlength],
  ControlDefs: FROM "controldefs" USING [
    EPIndex, EPRange, GFT, GFTIndex, GFTNull, GlobalFrame, GlobalFrameHandle,
    NullFrame, NullGlobalFrame, ProcDesc, SignalDesc],
  DebugBreakptDefs: FROM "debugbreakptdefs" USING [EntryToBTI],
  DebugContextDefs: FROM "debugcontextdefs" USING [FrameToModuleName],
  DebuggerDefs: FROM "debuggerdefs" USING [
    addbitaddrs, AmIaRecord, DumpCtxList, EquivalentVersions, FieldContext,
    FormatRecord, FRPointer, fullbitaddress, fullsymaddress, GetValue,
    GetValueN, InitSOP, LA, Lookup, MainBTI, SA, SearchType, SeiHandle,
    SOPointer, SymbolObject, VersionStamp, WriteBlanks, WriteSeiHandle,
    WriteTransferName],
  DebugMiscDefs: FROM "debugmiscdefs" USING [
    CopyRead, DFreeString, DGetString, DWriteLongInteger, LookupFail,
    WriteCharZ, WriteEOL],
  DebugRealDefs: FROM "debugrealdefs" USING [AppendRealNumber],
  DebugSymbolDefs: FROM "debugsymboldefs" USING [
    DAcquireSymbolTable, DReleaseSymbolTable, HandleForBase,
    SymbolsForGFrame, TableForString],
  DebugUtilityDefs: FROM "debugutilitydefs" USING [
    Bound, LengthenPointer, LoadStateInvalid, LongREAD,
    MakeProcedureDescriptor, MREAD, ReadGlobalGFI, UserWriteLongSubString,
    UserWriteLongString, ValidGlobalFrame],
  IODefs: FROM "iodefs" USING [
    CR, DEL, LF, NUL, NumberFormat, SP, TAB, WriteChar,
    WriteDecimal, WriteNumber, WriteOctal, WriteString],
  LoadStateDefs: FROM "loadstatedefs" USING [
    InputLoadState, ReleaseLoadState],
  SegmentDefs: FROM "segmentdefs" USING [InsufficientVM],
  StringDefs: FROM "stringdefs" USING [
    AppendSubString, SubString, SubStringDescriptor],
  SymbolTableDefs: FROM "symboltabledefs" USING [
    NoSymbolTable, SymbolTableBase, SymbolTableHandle],
  SymDefs: FROM "symdefs" USING [
    BTIndex, BTNull, CBTIndex, codeBOOLEAN, codeCHARACTER, codeINTEGER,
    CSEIndex, CTXIndex, CTXNull, ISEIndex, ISENull, SEIndex, SENull,
    TransferMode];

DEFINITIONS FROM DebuggerDefs;

DumpVars: PROGRAM
  IMPORTS DebugBreakptDefs, DebugContextDefs, DebuggerDefs, DebugMiscDefs,
    DebugRealDefs, DebugSymbolDefs, DebugUtilityDefs, IODefs, LoadStateDefs,
    SegmentDefs, StringDefs, SymbolTableDefs
  EXPORTS DebuggerDefs =
BEGIN

UnsignedDecimal: IODefs.NumberFormat = [
  base: 10, zerofill: FALSE, unsigned: TRUE, columns: 1];
CTXIndex: TYPE = SymDefs.CTXIndex;
GlobalFrameHandle: TYPE = ControlDefs.GlobalFrameHandle;
ISENull: ISEIndex = SymDefs.ISENull;
ISEIndex: TYPE = SymDefs.ISEIndex;
SENull: SEIndex = SymDefs.SENull;
SEIndex: TYPE = SymDefs.SEIndex;

StringCTXIndex: CTXIndex = LOOPHOLE[6];

Display: PUBLIC PROCEDURE [sop: SOPointer, frp: FRPointer, doconst: BOOLEAN] =
  BEGIN OPEN sop.stbase;  -- dump variable described by sei in frame
  tsei: SymDefs.CSEIndex ← UnderType[sop.tsei];
  space: CARDINAL;

    IF sop.sei # ISENull AND (seb+sop.sei).constant AND ~doconst THEN RETURN;
    WITH (seb + tsei) SELECT FROM
      basic => DumpBasicType[sop];
      record =>
        BEGIN
        IF sop.space = 0 THEN
```

```
          BEGIN
          space ← IF sop.sei # ISENull THEN (seb+sop.sei).idinfo
            ELSE WordsForType[tsei]*AltoDefs.wordlength;
          IF length < space THEN sop.baddr.bd ← sop.baddr.bd + space-length;
          END;
        DumpRecord[sop,FALSE];
        END;
      union => BEGIN DumpVariant[sop,frp]; RETURN END;
      subrange => DumpSubRange[sop];
      enumerated => DumpEnumerated[sop];
      pointer =>
        IF FieldContext[sop.stbase, pointedtotype] = StringCTXIndex AND
          TypeForm[pointedtotype] = record THEN DumpString[sop, FALSE]
        ELSE DumpPtr[sop];
      array => DumpArray[sop,frp];
      arraydesc => DumpArrayD[sop,frp];
      transfer =>
        SELECT mode FROM
          procedure => DumpProcV[sop];
          signal, error => DumpSigVar[sop];
          port => DumpPortV[sop];
          process => DumpProcessV[sop];
          program => DumpProgV[sop];
          ENDCASE => BEGIN IODefs.WriteChar['?];DebugMiscDefs.WriteEOL[]; END;
      long => DumpLong[sop,frp];
      relative => DumpRelative[sop];
      real => DumpReal[sop];
      --ignore other types
      ENDCASE => BEGIN IODefs.WriteChar['?];DebugMiscDefs.WriteEOL[]; END;
    frp.firstsym ← FALSE;
    RETURN
  END;

DumpRecord: PROCEDURE [sop: SOPointer, recurring:  BOOLEAN] =
  BEGIN OPEN DebugSymbolDefs, sop.stbase;   -- type fields of record
  newFR: FormatRecord;
  a: fullbitaddress;
  linkso: SymbolObject;
  linksop: SOPointer ← @linkso;
  root:  SymDefs.CSEIndex;
  tag, i, count: CARDINAL ← 0;
  tt,link: SEIndex;
  s: ISEIndex;
  sh: SymbolTableDefs.SymbolTableHandle ← HandleForBase[sop.stbase];
  changedSymbols: BOOLEAN ← FALSE;

  newFR ← FormatRecord[indentation: 0, symid: TRUE, firstsym: TRUE,
    symdelim: ':, startchar: '[, termchar: '], intersym: ',];
  InitSOP[linksop];
  IF sop.sei # ISENull AND sop.there
    THEN a ← addbitaddrs[fullsymaddress[sop], sop.baddr]
  ELSE a ← sop.baddr;
  linksop↑ ← sop↑; link ← sop.tsei;
  DO
    root ← UnderType[link];
    WITH t:(seb+root) SELECT FROM
    record =>
      WITH t SELECT FROM
        linked =>
          IF linktype # SENull THEN
            BEGIN link ← linktype; count ← count + 1; END
          ELSE EXIT;
        ENDCASE => EXIT;
    ENDCASE => EXIT;
    ENDLOOP;
  root ← UnderType[sop.tsei];
  WITH t:(seb+root) SELECT FROM
    record =>
      WITH t SELECT FROM
        linked => IF linktype # SENull THEN
          BEGIN
          tt ← sop.tsei;
          DO
            WITH (seb+tt) SELECT FROM
              id =>  BEGIN tag ← idvalue; tt ← idinfo; END;
              ENDCASE => EXIT;
```

```
            ENDLOOP;
          linksop↑.sei ← WITH (seb+linktype) SELECT FROM
            id => LOOPHOLE[linktype, ISEIndex],
            ENDCASE => ISENull;
          linksop↑.tsei ← linktype;
          linksop↑.baddr ← a;
          DumpRecord[linksop, TRUE !
            EnteringVPart => BEGIN OPEN sb;
            sop.stbase ← sb;
            changedSymbols ← TRUE;
            FOR s ← FirstCtxSe[c], NextSe[s] UNTIL s = ISENull DO
              IF (seb+s).constant AND (seb+s).idvalue = tag THEN
                BEGIN sop.sei ← s; sop.tsei ← s;
                root ← UnderType[sop.tsei]; EXIT; END;
            REPEAT
              FINISHED => ERROR;
            ENDLOOP;
            CONTINUE;
            END];
          newFR ← FormatRecord[indentation: 0, symid: TRUE, firstsym: TRUE,
            symdelim: ':, startchar: '[, termchar: '], intersym: ',];
          END;
        ENDCASE;
    ENDCASE;
  WriteSeiHandle[[stbase: sop.stbase,
    sei: WITH (seb+sop.tsei) SELECT FROM
           id => LOOPHOLE[sop.tsei, ISEIndex],
           ENDCASE => ISENull]];
  DumpCtxList[FieldContext[sop.stbase, root], sop.stbase, sop.there, a, @newFR
        ! AmIaRecord => RESUME[TRUE];
         EnteringVPart => IF ~recurring THEN RESUME];
  FOR i IN [0..count) DO
    DebugMiscDefs.WriteCharZ[newFR.termchar];
    ENDLOOP;
  IF changedSymbols THEN
    BEGIN DReleaseSymbolTable[sop.stbase]; [] ← DAcquireSymbolTable[sh]; END;
  RETURN
  END;

EnteringVPart: SIGNAL [sb:SymbolTableDefs.SymbolTableBase, c:CTXIndex] = CODE;

DumpVariant: PROCEDURE [sop: SOPointer, frp: FRPointer] =
  BEGIN OPEN sop.stbase;   -- type fields of record
  vso: SymbolObject;
  vsop: SOPointer ← @vso;
  gsei: ISEIndex;
  tsei: SymDefs.CSEIndex;
  lb: INTEGER ← 0;
  frp.firstsym ← FALSE;
  WriteBlanks[frp.indentation];
  WITH (seb + UnderType[sop.tsei]) SELECT FROM
    union => SIGNAL EnteringVPart[sop.stbase, casectx];
    ENDCASE => ERROR;
  WITH (seb + UnderType[sop.tsei]) SELECT FROM
    union =>
      BEGIN
      IF ~controlled THEN
        BEGIN IODefs.WriteString["COMPUTED Variant[...]"L]; RETURN END;
      gsei ← tagsei;
      END;
    ENDCASE;
  InitSOP[vsop];
  vso ← SymbolObject[sei: gsei, tsei: (seb+gsei).idtype, baddr: sop.baddr,
    stbase: sop.stbase, space: 0, there: sop.there];
  tsei ← UnderType[vsop.tsei];
  WITH (seb + tsei) SELECT FROM
    subrange => lb ← origin;
    ENDCASE;
  WITH (seb + UnderType[sop.tsei]) SELECT FROM
    union => DumpVariantSubpart[sop.stbase, sop.baddr, sop.there, casectx, GetValue[vsop]+lb];
    ENDCASE => ERROR;
  IF frp.firstsym THEN DebugMiscDefs.WriteCharZ[frp.termchar];
  RETURN
  END;

DumpVariantSubpart: PROCEDURE [sbase: SymbolTableDefs.SymbolTableBase,
```

```
  baddr: fullbitaddress, there: BOOLEAN, c: CTXIndex, v: UNSPECIFIED] =
  BEGIN OPEN DebugSymbolDefs, sbase, StringDefs;
  newFR: FormatRecord;
  sei: ISEIndex;
  sh: SymbolTableDefs.SymbolTableHandle;
  filename: STRING ← [40];
  desc: SubStringDescriptor;
  filess: SubString ← @desc;
  newctx: CTXIndex;
  includedVersion: DebuggerDefs.VersionStamp;

  WITH (ctxb+c) SELECT FROM
    included =>
      BEGIN
      IF ~ctxcomplete THEN
        BEGIN
        sh ← HandleForBase[sbase];
        SubStringForHash[filess, (mdb+ctxmodule).mdhti];
        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;
        DumpVariantSubpart[sbase, baddr, there, newctx, v];
        DReleaseSymbolTable[sbase];
        [] ← DAcquireSymbolTable[sh];
        RETURN
        END;
      EXITS
        continue => sbase ← DAcquireSymbolTable[sh];
      END;
    ENDCASE;
  FOR sei ← FirstCtxSe[c], NextSe[sei] UNTIL sei = ISENull DO
    IF (seb+sei).constant AND (seb+sei).idvalue = v THEN EXIT;
    REPEAT
      FINISHED => BEGIN OPEN IODefs;
        WriteString["UnknownVariant["L];
        WriteOctal[v];
        WriteChar[']];
        RETURN
        END;
    ENDLOOP;
  WriteSeiHandle[[stbase: sbase, sei: sei]];
  newFR ← FormatRecord[indentation: 0, symid: TRUE, firstsym: TRUE,
    symdelim: ':, startchar: '[, termchar: '], intersym: ',];
  DumpCtxList[FieldContext[sbase, (seb+sei).idinfo], sbase, there, baddr, @newFR];
  RETURN
  END;

DumpEnumerated: PROCEDURE [sop: SOPointer] =
  BEGIN OPEN IODefs, sop.stbase;   -- type a enumerated value
  v: INTEGER;
  WITH (seb + UnderType[sop.tsei]) SELECT FROM
    enumerated =>
      IF ~WriteEnumeratedValue[sop.stbase, valuectx, v ← GetValue[sop]]
        THEN BEGIN WriteString["?["L]; WriteOctal[v]; WriteChar[']]; END;
    ENDCASE;
  RETURN
  END;

WriteEnumeratedValue: PROCEDURE [sbase: SymbolTableDefs.SymbolTableBase,
  c: CTXIndex, v: UNSPECIFIED] RETURNS [b: BOOLEAN] =
  BEGIN OPEN DebugSymbolDefs, StringDefs, sbase;
  sei: ISEIndex;
  sh: SymbolTableDefs.SymbolTableHandle;
  filename: STRING ← [40];
  desc: SubStringDescriptor;
  filess: SubString ← @desc;
  newctx: CTXIndex;
  includedVersion: DebuggerDefs.VersionStamp;

  IF c = SymDefs.CTXNull THEN RETURN;
```

```
  WITH (ctxb+c) SELECT FROM
    included =>
      BEGIN
      IF ~ctxcomplete THEN
        BEGIN
        sh ← HandleForBase[sbase];
        SubStringForHash[filess, (mdb+ctxmodule).mdhti];
        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;
        b ← WriteEnumeratedValue[sbase, newctx, v];
        DReleaseSymbolTable[sbase];
        [] ← DAcquireSymbolTable[sh];
        RETURN
        END;
      EXITS
        continue => sbase ← DAcquireSymbolTable[sh];
      END;
    ENDCASE;
  FOR sei ← FirstCtxSe[c], NextSe[sei] UNTIL sei = ISENull DO
    IF (seb+sei).constant AND (seb+sei).idvalue = v THEN
      BEGIN
      WriteSeiHandle[SeiHandle[stbase: sbase, sei: sei]];
      RETURN[TRUE]
      END;
    ENDLOOP;
  RETURN[FALSE]
  END;

DumpSubRange: PROCEDURE [sop: SOPointer] =
  BEGIN OPEN IODefs, sop.stbase;   -- type a subrange value
  tsei: SymDefs.CSEIndex ← UnderType[sop.tsei];
  lb, v: INTEGER ← 0;
  size: CARDINAL ← AltoDefs.VMLimit;
  lbset:  BOOLEAN ← FALSE;
  WITH (seb+tsei) SELECT FROM
    subrange => size ← range;
    ENDCASE;
  DO
    WITH (seb+tsei) SELECT FROM
      subrange =>
        BEGIN
        IF ~lbset THEN lb ← origin;
        lbset ← TRUE;
        tsei ← UnderType[rangetype];
        END;
      enumerated =>
        BEGIN
        IF ~WriteEnumeratedValue[sop.stbase, valuectx, v ← (GetValue[sop]+lb)]
          THEN BEGIN WriteString["?["L]; WriteOctal[v]; WriteChar[']]; END;
        RETURN
        END;
      basic =>
        BEGIN
        v ← GetValue[sop] + lb;
        IF code = SymDefs.codeCHARACTER THEN
          BEGIN DumpCharacter[v]; RETURN END;
        IF size > 77777B THEN WriteOctal[v] ELSE WriteDecimal[v];
        RETURN
        END;
      pointer =>
        BEGIN
        v ← GetValue[sop] + lb;
        IF size > 77777B THEN WriteOctal[v] ELSE WriteDecimal[v];
        WriteChar['↑];
        RETURN
        END;
      ENDCASE =>
        BEGIN
        v ← GetValue[sop] + lb;
```

```
        IF size > 77777B THEN WriteOctal[v] ELSE WriteDecimal[v];
        RETURN
        END;
    ENDLOOP;
  RETURN
  END;

DumpCharacter: PUBLIC PROCEDURE [c: UNSPECIFIED] =
  BEGIN OPEN IODefs;
  SELECT c FROM
    NUL => WriteString["NUL"L];
    TAB => WriteString["TAB"L];
    LF => WriteString["LF"L];
    14C => WriteString["FF"L];
    CR => WriteString["CR"L];
    33C => WriteString["ESC"L];
    IN CHARACTER[NUL..SP) =>
      BEGIN WriteChar['↑]; WriteChar[LOOPHOLE[c+100B, CHARACTER]] END;
    SP => WriteString["BLANK"L];
    DEL => WriteString["RUBOUT"L];
    ENDCASE =>
      IF c ~IN CHARACTER[NUL..DEL] THEN WriteOctal[c]
      ELSE BEGIN WriteChar['']; WriteChar[c] END;
  RETURN
  END;

DumpString: PROCEDURE [sop: SOPointer, long: BOOLEAN] =
  BEGIN  OPEN DebugUtilityDefs, IODefs, StringDefs;  --type string at [[addr]]
  ls: LONG STRING;
  p: POINTER = @ls;
  l: CARDINAL;
  p↑ ← GetValue[sop];
  IF long THEN (p+1)↑ ← GetValueN[sop,1] ELSE ls ← LengthenPointer[p↑];
  IF ls = NIL THEN BEGIN WriteString["NIL"L]; RETURN; END;
  WriteChar['(]; WriteNumber[l ← LongREAD[@ls.length], UnsignedDecimal];
  WriteChar[',];
  -- the following address-arithmetic computes the location of ls.maxlength
  WriteNumber[LongREAD[ls+1], UnsignedDecimal]; WriteChar[')];
  WriteChar['"];
  IF l < 60 THEN UserWriteLongString[ls]
  ELSE
    BEGIN
    UserWriteLongSubString[ls: ls, length: 40, offset: 0];
    WriteString[" ... "L];
    UserWriteLongSubString[ls: ls, length: 10, offset: l-10];
    END;
  WriteChar['"];
  RETURN
  END;

InvalidGFTIndex: ERROR = CODE;

DumpProgV: PROCEDURE [sop: SOPointer] =
  BEGIN OPEN IODefs;  --type external representation of PROGRAM
  f: GlobalFrameHandle ← GetValue[sop];
  name: STRING ← DebugMiscDefs.DGetString[40];
  WriteString[" PROGRAM "L];
  IF DebugUtilityDefs.ValidGlobalFrame[f] THEN
    BEGIN
    [] ← LoadStateDefs.InputLoadState[];
    DebugContextDefs.FrameToModuleName[f, name];
    WriteString[name];
    WriteString[", Frame: "L];
    WriteOctal[f];
    LoadStateDefs.ReleaseLoadState[];
    END
  ELSE BEGIN WriteString["?["L]; WriteOctal[f]; WriteChar[']]; END;
  DebugMiscDefs.DFreeString[name];
  RETURN
  END;

DumpProcV: PROCEDURE [sop: SOPointer] =
  BEGIN  --type external representation of PROCV
  OPEN sop.stbase, ControlDefs, DebugUtilityDefs, IODefs;
  cl: ProcDesc;
  cp: BOOLEAN ← IF sop.sei # ISENull THEN (seb+sop.sei).constant ELSE FALSE;
```

```
  sh: SeiHandle;

  BEGIN
  WriteString[" PROCEDURE "L];
  IF ~cp THEN cl ← GetTransferValue[sop !LinksInCode => GOTO notImplemented]
  ELSE BEGIN
    sa: DebuggerDefs.SA;
    WITH sop.baddr SELECT FROM
      short => sa ← shortAddr;
      ENDCASE => ERROR;
    cl ← MakeProcedureDescriptor[(LOOPHOLE[(seb+sop.sei).idinfo, SymDefs.CBTIndex]+bb).entryIndex,
    LOOPHOLE[sa, GlobalFrameHandle]];
    END;
  DO
    SELECT cl.tag FROM
      procedure => EXIT;
      indirect => cl ← MREAD[cl];
      ENDCASE => GOTO end;
    ENDLOOP;
  IF cl.gfi # GFTNull THEN
    BEGIN
    sh ← ProcSeiHandle[cl.gfi, cl.ep !
      SymbolTableDefs.NoSymbolTable => GOTO noSym;
       SegmentDefs.InsufficientVM, InvalidGFTIndex => GOTO end];
    WriteTransferName[sh, TRUE, ControlDefs.NullFrame,
      MREAD[@ControlDefs.GFT[cl.gfi].frame]];
    DebugSymbolDefs.DReleaseSymbolTable[sh.stbase];
    END
  ELSE GOTO end;
  EXITS
    end => BEGIN WriteString["?["L]; WriteOctal[cl]; WriteChar[']]; END;
    notImplemented => WriteString["?[?]"L];
    noSym => WriteModuleNoSym[cl, MREAD[@GFT[cl.gfi].frame]];
  END;
  RETURN
  END;

ProcSeiHandle: PROCEDURE [gfti: ControlDefs.GFTIndex,
  epn: ControlDefs.EPIndex] RETURNS [sh: SeiHandle] =
  -- finds module and sei for proc whose descriptor is [gfti,epn]
  BEGIN OPEN DebugUtilityDefs, DebugSymbolDefs, ControlDefs, sh.stbase;
  gf: GlobalFrameHandle ← MREAD[@GFT[gfti].frame];
  gepn: EPIndex ← MREAD[@GFT[gfti].epbase];
  bti: SymDefs.BTIndex;
  IF gf = NullGlobalFrame OR ~ValidGlobalFrame[gf] OR (gepn MOD EPRange) # 0
    THEN ERROR InvalidGFTIndex;
  sh.stbase ← DAcquireSymbolTable[SymbolsForGFrame[gf]];
  epn ← epn + gepn;
  bti ← DebugBreakptDefs.EntryToBTI[sh.stbase, epn];
  sh.sei ← IF bti = SymDefs.BTNull THEN ISENull
  ELSE WITH b:(bb+bti) SELECT FROM
         Callable => b.id,
         ENDCASE => ISENull;
  RETURN
  END;

DumpSigVar: PROCEDURE [sop: SOPointer] =
  BEGIN  --type external representation of SIGV
  OPEN IODefs, ControlDefs, DebugUtilityDefs, DebugSymbolDefs, sop.stbase;
  sigv: ControlDefs.SignalDesc;
  cp: BOOLEAN ← IF sop.sei # ISENull THEN (seb+sop.sei).constant ELSE FALSE;
  sh: SeiHandle;

  BEGIN
  WriteString[IF XferMode[sop.tsei] = error THEN " ERROR "L ELSE " SIGNAL "L];
  IF ~cp THEN sigv ← LOOPHOLE[GetTransferValue[sop
    !LinksInCode => GOTO notImplemented]]
  ELSE BEGIN
    sa: DebuggerDefs.SA;
    WITH sop.baddr SELECT FROM
      short => sa ← shortAddr;
      ENDCASE => ERROR;
    sigv ← LOOPHOLE[(seb+sop.sei).idvalue+ReadGlobalGFI[LOOPHOLE[sa, GlobalFrameHandle]], SignalDesc];
    END;
  IF sigv.gfi # ControlDefs.GFTNull THEN
    BEGIN
```

```
    sh ← SigSeiHandle[sigv !
      SymbolTableDefs.NoSymbolTable => GOTO noSym;
      SegmentDefs.InsufficientVM, InvalidGFTIndex => GOTO end];
    WriteTransferName[sh, TRUE, ControlDefs.NullFrame,
      MREAD[@ControlDefs.GFT[sigv.gfi].frame]];
    DReleaseSymbolTable[sh.stbase];
    END
  ELSE GOTO end;
  EXITS
    end => BEGIN WriteString["?["L]; WriteOctal[sigv]; WriteChar[']]; END;
    notImplemented => WriteString["?[?]"L];
    noSym => WriteModuleNoSym[sigv, MREAD[@GFT[sigv.gfi].frame]];
  END;
  RETURN
  END;

SigSeiHandle: PUBLIC PROCEDURE [sigv: ControlDefs.SignalDesc] RETURNS [sh: SeiHandle] =
  BEGIN -- finds module and sei for sig
  OPEN DebugUtilityDefs, DebugSymbolDefs, ControlDefs, sh.stbase;
  t: SymDefs.TransferMode;
  sei: ISEIndex;
  gf: GlobalFrameHandle ← MREAD[@GFT[sigv.gfi].frame];
  gepn: EPIndex ← MREAD[@GFT[sigv.gfi].epbase];
  sh.sei ← ISENull;
  IF gf = NullGlobalFrame OR ~ValidGlobalFrame[gf] OR (gepn MOD EPRange) # 0
    THEN ERROR InvalidGFTIndex;
  sh.stbase ← DAcquireSymbolTable[SymbolsForGFrame[gf]];
  sigv.gfi ← gepn/EPRange; --strip off the gfi
  FOR sei ← FirstCtxSe[stHandle.outerCtx], NextSe[sei]
  UNTIL sei = ISENull DO
    t ← XferMode[(seb+sei).idtype];
    IF (t = signal OR t = error) AND (seb+sei).constant
      AND (seb+sei).idvalue = sigv
      THEN BEGIN sh.sei ← sei; RETURN END;
    ENDLOOP;
  RETURN
  END;

WriteModuleNoSym: PROCEDURE [v: UNSPECIFIED, gf: GlobalFrameHandle] =
  BEGIN OPEN IODefs;
  module: STRING ← DebugMiscDefs.DGetString[40];
  WriteString["?["L];
  WriteOctal[v];
  WriteString["], ("L];
  BEGIN ENABLE DebugUtilityDefs.LoadStateInvalid => GOTO end;
  [] ← LoadStateDefs.InputLoadState[];
  DebugContextDefs.FrameToModuleName[gf, module];
  WriteString["in "L];
  WriteString[module];
  DebugMiscDefs.DFreeString[module];
  LoadStateDefs.ReleaseLoadState[];
  WriteString[", "L];
  EXITS
   end => NULL;
  END;
  WriteString["G: "L];
  WriteOctal[gf];
  WriteChar[')];
  RETURN
  END;

LinksInCode: SIGNAL = CODE;

GetTransferValue: PROCEDURE [sop: SOPointer] RETURNS [ControlDefs.ProcDesc] =
  BEGIN OPEN ControlDefs, sop.stbase;
  localFrame, gf: GlobalFrameHandle;
  IF sop.sei = ISENull OR ~(seb+sop.sei).linkSpace
    THEN RETURN [GetValue[sop]];
  WITH sop.baddr SELECT FROM
    short => gf ← LOOPHOLE[shortAddr, GlobalFrameHandle];
    ENDCASE => ERROR;
  DebugMiscDefs.CopyRead[from: gf, to: @localFrame,
    nwords: SIZE[GlobalFrame]];
  IF ~localFrame.codelinks THEN --links in frame
    BEGIN
    addr: UNSPECIFIED ← gf - (seb+sop.sei).idvalue - 1;
```

```
    RETURN[DebugUtilityDefs.MREAD[addr]];
    END
  ELSE SIGNAL LinksInCode;
  END;

DumpPortV: PROCEDURE [sop: SOPointer] =
  BEGIN OPEN IODefs;
  WriteString["PORT ["L];
  WriteOctal[GetValue[sop]];
  WriteString[", "L];
  WriteOctal[GetValueN[sop,1]];
  WriteChar[']];
  RETURN
  END;

DumpProcessV: PROCEDURE [sop: SOPointer] =
  BEGIN OPEN IODefs;
  WriteString["PROCESS ["L];
  WriteOctal[GetValue[sop]];
  WriteChar[']];
  RETURN
  END;

DumpBasicType: PROCEDURE [sop: SOPointer] =
  BEGIN OPEN SymDefs, IODefs, sop.stbase;
  v: UNSPECIFIED ← GetValue[sop];
  WITH (seb+UnderType[sop.tsei]) SELECT FROM
    basic => SELECT code FROM
      codeINTEGER => WriteDecimal[v];
      codeBOOLEAN =>
           IF v = 0 THEN WriteString["FALSE"L]
           ELSE WriteString["TRUE"L];
      codeCHARACTER => DumpCharacter[v];
      ENDCASE => WriteOctal[v];
    ENDCASE;
  RETURN
  END;

DumpPtr: PROCEDURE [sop: SOPointer] =
  BEGIN OPEN IODefs;
  v: UNSPECIFIED ← GetValue[sop];
  IF v = NIL THEN
    BEGIN WriteString["NIL"L]; RETURN END;
  WriteOctal[v]; WriteChar['↑];
  RETURN
  END;

DumpLongPtr: PROCEDURE [sop: SOPointer] =
  BEGIN OPEN IODefs;
  lp: LONG POINTER;
  p: POINTER ← @lp;
  p↑ ← GetValue[sop];
  (p+1)↑ ← GetValueN[sop, 1];
  IF lp = NIL THEN BEGIN WriteString["NIL"L]; RETURN END;
  DebugMiscDefs.DWriteLongInteger[LOOPHOLE[lp], 8];
  WriteChar['↑];
  RETURN
  END;

DumpRelative: PROCEDURE [sop: SOPointer] =
  BEGIN OPEN IODefs;
  v: UNSPECIFIED ← GetValue[sop];
  WriteNumber[v, UnsignedDecimal]; WriteString["↑R"L];
  RETURN
  END;

sDumpArray: PROCEDURE [nelements, elementsize: CARDINAL,
  sop: SOPointer, frp: FRPointer, rsei: SEIndex] =
  BEGIN OPEN IODefs;  --dump array elements
  nminus1, k: CARDINAL;
  char: CHARACTER ← '[;
  all:  BOOLEAN ← FALSE;
  aSO:  SymbolObject ← sop↑;
  asop:  SOPointer ← @aSO;
  aFR :  FormatRecord ← frp↑;
  afrp:  FRPointer ← @aFR;
```

```
  WriteChar['('); WriteDecimal[nelements]; WriteChar[')];
  IF nelements = 0 THEN RETURN;
  nminus1 ← nelements - 1;
  IF nelements < 7 OR nelements*elementsize < 7
    OR (sop.space=8 AND nelements<14)
    THEN BEGIN all ← TRUE; k ← nelements; END
  ELSE k ← MIN[2, nminus1];
  afrp.symid ← FALSE;
  asop.tsei ← rsei;
  asop.sei ← ISENull;
  IF k = 0 THEN WriteChar['[]
  ELSE THROUGH [1..k] DO
    WriteChar[char];  WriteChar[' ];
    Display[asop, afrp, FALSE];
    char ← ',;
    IF asop.space = 0
      THEN WITH a:asop.baddr SELECT FROM
        short => a.shortAddr ← [a.shortAddr + elementsize];
        long => a.longAddr.li ←a.longAddr.li+elementsize;
        ENDCASE
    ELSE asop.baddr ← addbitaddrs[asop.baddr, fullbitaddress[wd:short[[0]],bd:8]];
    ENDLOOP;
  IF ~all THEN
    BEGIN
    WriteString[", ... , "L];
    IF asop.space = 0
      THEN WITH sop.baddr SELECT FROM
        short => asop.baddr.wd ←
          short[shortAddr: [shortAddr + elementsize*nminus1]];
        long => asop.baddr.wd ← long[longAddr: LA[LI[li:longAddr.li+elementsize*nminus1]]];
        ENDCASE
    ELSE asop.baddr ← addbitaddrs[sop.baddr,
      fullbitaddress[wd:short[[nminus1/2]],bd:(nminus1 MOD 2)*8]];
    Display[asop, afrp, FALSE];
    END;
  WriteChar[']];
  RETURN
  END;

DumpArray: PROCEDURE [sop: SOPointer, frp: FRPointer] =
  BEGIN OPEN sop.stbase;
  rsei: SymDefs.CSEIndex ← UnderType[sop.tsei];
  csei: SEIndex;
  nelements: CARDINAL;
  sym: fullbitaddress ← fullsymaddress[sop];
  sa: DebuggerDefs.SA;
  aSO:  SymbolObject ← sop↑;
  asop:  SOPointer ← @aSO;

  WITH sym SELECT FROM
    short => sa ← shortAddr;
    ENDCASE => ERROR;
  WITH sop.baddr SELECT FROM
    short => asop.baddr.wd ← short[[shortAddr + sa]];
    long => asop.baddr.wd ← long[longAddr:LA[LI[li:longAddr.li+sa]]];
    ENDCASE;
  WITH a: (seb + rsei) SELECT FROM
    array =>
      BEGIN
      IF a.packed THEN asop.space ← 8;
      nelements ← Cardinality[a.indextype];
      rsei ← UnderType[csei ← a.componenttype];
      END;
    ENDCASE => ERROR;
  sDumpArray[nelements, WordsForType[rsei], asop, frp, csei];
  RETURN
  END;

DumpArrayD: PROCEDURE [sop: SOPointer, frp: FRPointer] =
  BEGIN --dump array descriptor
  arrayD[sop, frp, GetValueN[sop,1]];
  RETURN
  END;

DumpLongArrayD: PROCEDURE [sop: SOPointer, frp: FRPointer] =
```

```
  BEGIN OPEN sop.stbase; --dump long array descriptor
  WITH (seb+UnderType[sop.tsei]) SELECT FROM
    long => sop.tsei ← rangetype;
    ENDCASE => ERROR;
  arrayD[sop, frp, GetValueN[sop,2]];
  RETURN
  END;

arrayD: PROCEDURE [sop: SOPointer, frp: FRPointer, nelements: CARDINAL] =
  BEGIN OPEN sop.stbase; --dump array descriptor
  rsei: SymDefs.CSEIndex ← UnderType[sop.tsei];
  csei: SEIndex;
  aSO: SymbolObject ← sop↑;
  asop: SOPointer ← @aSO;

  asop.baddr.wd ← short[GetValue[sop]];
  WITH (seb + rsei) SELECT FROM
    arraydesc =>
      BEGIN
      WITH a:(seb+UnderType[describedType]) SELECT FROM
        array =>
          BEGIN
          IF a.packed THEN asop.space ← 8;
          rsei ← UnderType[csei ← a.componenttype];
          END;
        ENDCASE => ERROR;
      END;
    ENDCASE => ERROR;
  sDumpArray[nelements, WordsForType[rsei], asop, frp, csei];
  RETURN
  END;

DumpLong: PROCEDURE [sop: SOPointer, frp: FRPointer] =
  BEGIN OPEN sop.stbase;
  WITH (seb+UnderType[sop.tsei]) SELECT FROM
    long =>
      WITH (seb+UnderType[rangetype]) SELECT FROM
        arraydesc => DumpLongArrayD[sop,frp];
        pointer =>
          IF FieldContext[sop.stbase, pointedtotype] = StringCTXIndex AND
            TypeForm[pointedtotype] = record THEN DumpString[sop, TRUE]
          ELSE DumpLongPtr[sop];
        basic =>
          SELECT code FROM
            SymDefs.codeINTEGER => DumpLongInteger[sop];
            ENDCASE => ERROR;
        ENDCASE => ERROR;
    ENDCASE => ERROR;
  RETURN
  END;

DumpLongInteger: PROCEDURE [sop: SOPointer] =
  BEGIN
  num: LONG INTEGER;
  p: POINTER ← @num;
  p↑ ← GetValue[sop];
  (p+1)↑ ← GetValueN[sop,1];
  DebugMiscDefs.DWriteLongInteger[num, 10];
  RETURN
  END;


DumpReal: PROCEDURE [sop: SOPointer] =
  BEGIN
  s: STRING ← [30];
  r: REAL;
  p: POINTER ← @r;
  p↑ ← GetValue[sop];
  (p+1)↑ ← GetValueN[sop,1];
  IF DebugUtilityDefs.Bound[DebugRealDefs.AppendRealNumber] THEN
    BEGIN
    DebugRealDefs.AppendRealNumber[s, r];
    IODefs.WriteString[s];
    END
  ELSE
    BEGIN OPEN IODefs;
```

```
    WriteString["REAL["L];
    WriteOctal[p↑]; WriteChar[',]; WriteOctal[(p+1)↑];
    WriteChar[']];
    END;
  END;

DumpVar: PUBLIC PROCEDURE [s: STRING, where: SearchType] =
  BEGIN
  so: SymbolObject;
  sop: SOPointer = @so;
  FR: FormatRecord;
  frp: FRPointer = @FR;
  InitSOP[sop];
  IF ~Lookup[s, FALSE, sop, FALSE, where]
    THEN SIGNAL DebugMiscDefs.LookupFail[s];
  FR ← FormatRecord[indentation:1, symid:FALSE, firstsym:TRUE, symdelim: '=,
    startchar: IODefs.NUL, termchar: IODefs.NUL, intersym: IODefs.CR];
  Display[sop, frp, TRUE
    !AmIaRecord => RESUME[FALSE]];
  IF where = config THEN
    BEGIN OPEN sop.stbase;
    IODefs.WriteString[" (in "L];
    WriteSeiHandle[[stbase: sop.stbase, sei: (bb+MainBTI).id]];
    IODefs.WriteChar[')];
    END;
  DebugMiscDefs.WriteEOL[];
  DebugSymbolDefs.DReleaseSymbolTable[sop.stbase];
  RETURN
  END;


END ...
```