```
-- file DIActionsMed.Mesa
-- Edited by:
--             Sandman, April 17, 1978  4:41 PM
--             Barbara, July 31, 1978  5:13 PM
--             Johnsson, August 29, 1978  11:19 AM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [wordlength],
  CommandDefs: FROM "commanddefs" USING [WriteErrorString],
  ControlDefs: FROM "controldefs" USING [
    FieldDescriptor, FrameHandle, GFT, GFTItem, GlobalFrameHandle,
    MaxParmsInStack, ProcDesc, StateVector],
  DebugData: FROM "debugdata" USING [worryentry],
  DebuggerDefs: FROM "debuggerdefs" USING [
    GetValue, GetValueN, InitSOP, LA, SA, SOPointer, SymbolObject],
  DebugMiscDefs: FROM "debugmiscdefs" USING [DebugAbort],
  DebugUtilityDefs: FROM "debugutilitydefs" USING [
    CheckFrame, LengthenPointer, LongREAD, LongWRITE, MREAD,
    MWRITE, ReadGlobalGFI, UserCall, ValidGlobalFrame],
  DIActionDefs: FROM "diactiondefs" USING [
    AllocateHereStackItem, AllocateThereStackItem, dereferenceItem, espTosop,
    FreeStackItem, GetCurrentST, IncorrectType, popevalstack, popNevalstack,
    pushevalstack, Transfer, TypesDontMatch],
  DIDefs: FROM "didefs" USING [ESPointer, hereESPointer, thereESPointer],
  DITypeDefs: FROM "ditypedefs" USING [
    AssignableTypes, ESPointer, SeiPType, TypeArray, TypeArrayDesc,
    TypeInteger, TypePointer, TypeProcedure, TypeString],
  InlineDefs: FROM "inlinedefs" USING [COPY],
  Mopcodes: FROM "mopcodes" USING [zRFS, zWFS],
  SDDefs: FROM "sddefs" USING [SD, sGFTLength],
  StringDefs: FROM "stringdefs" USING [SubString],
  SymDefs: FROM "symdefs" USING [
    CBTIndex, CSEIndex, ISEIndex, recordCSEIndex, SEIndex, SENull],
  SystemDefs: FROM "systemdefs" USING [AllocateHeapNode];

DIActionsMed: PROGRAM
  IMPORTS CommandDefs, DDptr: DebugData, DebuggerDefs, DebugMiscDefs,
    DebugUtilityDefs, DIActionDefs, DITypeDefs, SystemDefs
  EXPORTS DIActionDefs =
  BEGIN

--stack items
ESPointer: TYPE = DIDefs.ESPointer;
hereESPointer: TYPE = DIDefs.hereESPointer;
thereESPointer: TYPE = DIDefs.thereESPointer;
SOPointer: TYPE = DebuggerDefs.SOPointer;

--assignment statements
NotImplemented: PUBLIC SIGNAL = CODE;

assignvalue: PUBLIC PROCEDURE [rhs: ESPointer, lhs: thereESPointer] =
  BEGIN OPEN DIActionDefs;
  IF ~DITypeDefs.AssignableTypes[lhs, rhs] OR
    (DITypeDefs.TypeString[rhs] AND rhs.tag = here)
    THEN SIGNAL TypesDontMatch[lhs, rhs];
  Assign[lhs,rhs];
  FreeStackItem[rhs];
  FreeStackItem[lhs];
  RETURN
  END;

Assign: PROCEDURE [lhs: thereESPointer, right: ESPointer] =
  BEGIN OPEN DebugUtilityDefs;
  rhs: hereESPointer ← DIActionDefs.Transfer[right];
  i: CARDINAL;
  fd: ControlDefs.FieldDescriptor;
  word: UNSPECIFIED;
  IF rhs.wordlength = 1 THEN
    BEGIN
    WITH rhs.stbase.seb+rhs.stbase.UnderType[rhs.tsei] SELECT FROM
      subrange => IF origin # 0 THEN rhs.value ← rhs.value + origin;
      ENDCASE;
    WITH lhs.stbase.seb+lhs.stbase.UnderType[lhs.tsei] SELECT FROM
      subrange => IF origin # 0 THEN rhs.value ← rhs.value - origin;
      ENDCASE;
    END;
```

```
  IF lhs.bitsize <= AltoDefs.wordlength THEN
    BEGIN OPEN InlineDefs;
    IF rhs.wordlength > 1 THEN SIGNAL DIActionDefs.TypesDontMatch[lhs, rhs];
    fd ← [offset: 0, posn: lhs.bitoffset, size: lhs.bitsize];
    WITH lhs SELECT FROM
      short => word ← MREAD[shortAddr];
      long => word ← LongREAD[longAddr.lp];
      ENDCASE;
    WriteField[rhs.value, @word, fd];
    WITH lhs SELECT FROM
      short => MWRITE[shortAddr, word];
      long => LongWRITE[longAddr.lp, word];
      ENDCASE;
    END
  ELSE
    BEGIN OPEN DebuggerDefs;
    IF lhs.bitsize MOD AltoDefs.wordlength # 0 OR lhs.bitoffset # 0
      THEN ERROR;
    IF rhs.wordlength = 1 THEN LengthenHESP[lhs, rhs];
    FOR i IN [0..lhs.bitsize/AltoDefs.wordlength) DO
      WITH lhs SELECT FROM
        short => MWRITE[shortAddr+i, (rhs.ptr+i)↑];
        long => LongWRITE[longAddr.lp+i, (rhs.ptr+i)↑];
        ENDCASE;
      ENDLOOP;
    END;
  DIActionDefs.FreeStackItem[rhs];
  RETURN
  END;

LengthenHESP: PROCEDURE [lesp: thereESPointer, esp: hereESPointer] =
  BEGIN OPEN DebuggerDefs, DITypeDefs;
  la: LA;
  SELECT TRUE FROM
    TypePointer[lesp] => la.lp ← DebugUtilityDefs.LengthenPointer[esp.value];
    TypeInteger[lesp] =>
      BEGIN OPEN e:esp.stbase;
      WITH e.seb + e.UnderType[esp.tsei] SELECT FROM
        subrange => la.li ← CARDINAL[esp.value];
        basic => la.li ← INTEGER[esp.value];
        ENDCASE => ERROR;
      END;
    ENDCASE => ERROR;
  esp.ptr ← SystemDefs.AllocateHeapNode[esp.wordlength ← 2];
  LOOPHOLE[esp.ptr, POINTER TO LA]↑ ← la;
  RETURN
  END;

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

WriteField: PROCEDURE [UNSPECIFIED, POINTER, ControlDefs.FieldDescriptor] =
  MACHINE CODE BEGIN Mopcodes.zWFS END;

--expression lists
GetArrayElement: PROCEDURE [esp: ESPointer] RETURNS [new: ESPointer] =
  BEGIN OPEN DIActionDefs, s: esp.stbase;
  temp: ESPointer;
  isei, csei: SymDefs.SEIndex;
  packed: BOOLEAN;
  i: INTEGER;
  csize: CARDINAL;
  tnew: thereESPointer;
  temp ← popevalstack[];  -- get temp.value(th) element
  WITH s.seb+s.UnderType[esp.tsei] SELECT FROM
    long => esp.tsei ← rangetype;
    ENDCASE;
  WITH a: (s.seb+s.UnderType[esp.tsei]) SELECT FROM
    array =>
       BEGIN
       packed ← a.packed;
       csize ← s.WordsForType[s.UnderType[csei ← a.componenttype]];
       isei ← a.indextype;
       END;
    arraydesc =>
       WITH aa: (s.seb+s.UnderType[a.describedType]) SELECT FROM
```

```
        array =>
          BEGIN
          packed ← aa.packed;
          csize ← s.WordsForType[s.UnderType[csei ← aa.componenttype]];
          isei ← aa.indextype;
          END;
        ENDCASE => ERROR;
    ENDCASE => ERROR;
  i ← GetTheValue[temp];
  WITH s.seb+s.UnderType[isei] SELECT FROM
    subrange => i ← i - origin;
    ENDCASE;
  WITH e: esp SELECT FROM
    there =>
      BEGIN OPEN DebuggerDefs, DebugUtilityDefs;
      wordoffset: CARDINAL =
        IF packed THEN CARDINAL[i]/2 ELSE CARDINAL[i]*csize;
      tnew ← AllocateThereStackItem[];
      tnew.stbase ← esp.stbase;
      tnew.tsei ← csei;
      tnew.bitoffset ← IF packed AND i MOD 2 = 1 THEN 8 ELSE 0;
      tnew.bitsize ← IF packed THEN 8 ELSE 16*csize;
      WITH e SELECT FROM
        short => tnew.addr ←
          short[shortAddr: [WITH a:(s.seb+s.UnderType[tsei]) SELECT FROM
            arraydesc => MREAD[shortAddr] + wordoffset,
            ENDCASE => shortAddr + wordoffset]];
        long => WITH a:(s.seb+s.UnderType[tsei]) SELECT FROM
          arraydesc => tnew.addr ← short[
            shortAddr: [LongREAD[longAddr.lp] + wordoffset]];
          ENDCASE => tnew.addr ← long[
            longAddr:LA[LI[longAddr.li + wordoffset]]];
        ENDCASE;
      new ← tnew;
      END;
    here =>
      WITH a: (s.seb+s.UnderType[e.tsei]) SELECT FROM
         arraydesc =>
          BEGIN
          tnew ← AllocateThereStackItem[];
          tnew.stbase ← esp.stbase;
          tnew.tsei ← csei;
          tnew.bitoffset ← IF packed AND i MOD 2 = 1 THEN 8 ELSE 0;
          tnew.addr ← short[LOOPHOLE[e.ptr↑ +(IF packed THEN CARDINAL[i]/2
            ELSE CARDINAL[i]*csize), DebuggerDefs.SA]];
          tnew.bitsize ← IF packed THEN 8 ELSE 16*csize;
          new ← tnew;
          END;
        ENDCASE => ERROR; -- what about here arrays ??
    ENDCASE => ERROR;
  FreeStackItem[temp];
  FreeStackItem[esp];
  RETURN
  END;

--indexing strings
GetStringElement: PROCEDURE [esp: ESPointer] RETURNS [new: ESPointer] =
  BEGIN OPEN DIActionDefs, s: esp.stbase;
  i: CARDINAL;
  hnew: hereESPointer;
  tnew: thereESPointer;
  i ← GetTheValue[popevalstack[]];  -- get i(th) character
  WITH e:esp SELECT FROM
     here =>
       BEGIN OPEN ss: LOOPHOLE[e.value, StringDefs.SubString];
       hnew ← AllocateHereStackItem[];
       hnew.value ← ss.base[ss.offset+i];
       new ← hnew;
       END;
     there =>
       BEGIN OPEN DebugUtilityDefs;
       tnew ← AllocateThereStackItem[];
       WITH e SELECT FROM
         short => tnew.addr ← short[shortAddr: [MREAD[shortAddr]+2+i/2]];
         long => tnew.addr ← short[shortAddr: [LongREAD[longAddr.lp]+2+i/2]];
         ENDCASE;
```

```
        tnew.bitsize ← 8;
        tnew.bitoffset ← IF i MOD 2 = 1 THEN 8 ELSE 0;
        new ← tnew;
        END;
      ENDCASE => ERROR;
  new.stbase ← DIActionDefs.GetCurrentST[];
  new.tsei ← DITypeDefs.SeiPType[character, DIActionDefs.GetCurrentST[]];
  FreeStackItem[esp];
  RETURN
  END;

InvalidExpression: PUBLIC SIGNAL = CODE;

--calling procedures
ProcedureCall: PROCEDURE [esp: ESPointer, nparams: CARDINAL]
  RETURNS [results: hereESPointer] =
  BEGIN OPEN DIActionDefs, s: esp.stbase;
  found: BOOLEAN;
  so: DebuggerDefs.SymbolObject;
  sop: SOPointer ← @so;
  state: ControlDefs.StateVector;
  procdesc: ControlDefs.ProcDesc;
  param: ESPointer;
  i,n: CARDINAL ← 0;
  typein, typeout: SymDefs.recordCSEIndex;
  sei: SymDefs.ISEIndex;
  IF DDptr.worryentry THEN
    BEGIN
    CommandDefs.WriteErrorString[naworry];
    SIGNAL DebugMiscDefs.DebugAbort;
    END;
  WITH (s.seb+LOOPHOLE[esp.tsei, SymDefs.CSEIndex]) SELECT FROM
    transfer =>
      BEGIN typein ← inrecord; typeout ← outrecord; END;
    ENDCASE => ERROR;
  IF (state.stkptr ← s.WordsForType[typein]) > ControlDefs.MaxParmsInStack
    OR s.WordsForType[typeout] > ControlDefs.MaxParmsInStack
    THEN SIGNAL InvalidExpression;
  IF typein # SymDefs.SENull THEN -- no input params
    FOR sei ← s.FirstCtxSe[(s.seb+typein).fieldctx], s.NextSe[sei]
    UNTIL sei = SymDefs.SENull DO
      IF nparams = 0 THEN SIGNAL InvalidExpression;
      param ← popNevalstack[nparams-1];
      IF ~ArgumentType[param, esp, (s.seb+sei).idtype]
        THEN SIGNAL InvalidExpression
      ELSE BEGIN
        DebuggerDefs.InitSOP[sop];
        espTosop[param,sop];
        FOR i IN [0..param.stbase.WordsForType[param.tsei]) DO
          state.stk[n] ← DebuggerDefs.GetValueN[sop,i];
          WITH sop.stbase.seb+sop.stbase.UnderType[sop.tsei] SELECT FROM
            subrange =>
              IF origin # 0 THEN state.stk[n] ← state.stk[n] + origin;
            ENDCASE;
          n ← n+1;
          ENDLOOP;
        END;
      nparams ← nparams - 1;
      FreeStackItem[param];
      ENDLOOP;
  IF nparams # 0 THEN SIGNAL InvalidExpression;
  DebuggerDefs.InitSOP[sop];
  espTosop[esp,sop];
  [procdesc,found] ← decodeproc[sop];
  IF ~found THEN SIGNAL InvalidExpression[];
  state.dest ← procdesc;
  state.source ← 0; state.instbyte ← 0; state.fill ← 0; --not used
  [] ← DebugUtilityDefs.UserCall[@state];
  IF typeout # SymDefs.SENull THEN
    BEGIN
    results ← AllocateHereStackItem[];
    results.stbase ← esp.stbase;
    IF (results.stbase.seb+typeout).unifield THEN
      BEGIN OPEN rs: results.stbase, r: (rs.seb+typeout);
      results.tsei ← (rs.FirstCtxSe[r.fieldctx]+rs.seb).idtype;
      END
```

```
    ELSE results.tsei ← typeout;
    results.wordlength ← s.WordsForType[typeout];
    IF results.wordlength = 1 THEN results.value ← state.stk[0]
    ELSE BEGIN
      results.ptr ← SystemDefs.AllocateHeapNode[results.wordlength];
      InlineDefs.COPY[from: @state.stk[0], to: results.ptr, nwords: results.wordlength];
      END;
    END
  ELSE results ← NIL;
  FreeStackItem[esp];
  RETURN
  END;

decodeproc: PROCEDURE [sop: SOPointer]
  RETURNS[pd: ControlDefs.ProcDesc, found: BOOLEAN] =
  BEGIN OPEN DebugUtilityDefs, ControlDefs, SDDefs, sop.stbase;
  f: FrameHandle;
  frame: GlobalFrameHandle;
  e: CARDINAL;
  gfti: CARDINAL;
  bti: SymDefs.CBTIndex;
  gft: DESCRIPTOR FOR ARRAY OF GFTItem ←
     DESCRIPTOR[GFT, MREAD[SD+sGFTLength]];
  WITH sop.baddr SELECT FROM
    short => f ← LOOPHOLE[shortAddr, FrameHandle];
    ENDCASE => ERROR;
  frame ← IF CheckFrame[f] THEN MREAD[@f.accesslink]
    ELSE LOOPHOLE[f, GlobalFrameHandle];
  IF (seb+sop.sei).constant THEN
    BEGIN
    frame ← IF CheckFrame[f] THEN MREAD[@f.accesslink]
      ELSE LOOPHOLE[f, GlobalFrameHandle];
    bti ← (seb+sop.sei).idinfo;
    e ← (bb+bti).entryIndex;
    WITH (bb+bti) SELECT FROM
      Outer => pd.tag ← procedure;
      ENDCASE => pd.tag ← unbound;
    gfti ← ReadGlobalGFI[frame];
    pd.gfi ← gfti + e/32;
    pd.ep ← e MOD 32;
    END
  ELSE
    BEGIN
    pd ← DebuggerDefs.GetValue[sop];
    gfti ← pd.gfi;
    f ← MREAD[@gft[gfti].frame];
    frame ← MREAD[@f.accesslink];
    END;
  IF gfti ~ IN[0..LENGTH[gft]) OR MREAD[@gft[gfti].epbase] MOD 32 # 0 OR
    ~ValidGlobalFrame[frame] OR pd.tag # procedure THEN
    RETURN[pd, FALSE];
  RETURN[pd, TRUE];
  END;

ArgumentType: PROCEDURE[param, esp: ESPointer, tsei: SymDefs.SEIndex]
  RETURNS[ok: BOOLEAN] =
  BEGIN OPEN DIActionDefs;
  temp: ESPointer;
  IF DITypeDefs.TypeString[param] AND param.tag = here THEN RETURN[FALSE];
  temp ← AllocateHereStackItem[];
  temp.stbase ← esp.stbase; temp.tsei ← tsei;
  ok ← DITypeDefs.AssignableTypes[param, temp];
  FreeStackItem[temp];
  END;

GetRelativePointer: PROCEDURE [esp: ESPointer] RETURNS[tnew: thereESPointer] =
  BEGIN OPEN DIActionDefs;
  rptr: ESPointer;
  rptr ← popevalstack[];  -- of the form esp[rptr]
  WITH p:(esp.stbase.seb+esp.stbase.UnderType[esp.tsei]) SELECT FROM
    pointer => IF ~p.basing THEN SIGNAL IncorrectType[esp];
    ENDCASE => SIGNAL IncorrectType[esp];
  WITH r:(rptr.stbase.seb+rptr.stbase.UnderType[rptr.tsei]) SELECT FROM
    relative =>
      BEGIN
      hnew: hereESPointer;
```

```
      hnew ← AllocateHereStackItem[];
      hnew.stbase ← esp.stbase;
      hnew.tsei ← r.resultType;
      hnew.value ← GetTheValue[esp] + GetTheValue[rptr];
      tnew ← AllocateThereStackItem[];
      tnew ← dereferenceItem[hnew !ANY => SIGNAL IncorrectType[hnew]];
      END;
    ENDCASE => SIGNAL IncorrectType[rptr];
  FreeStackItem[esp];
  RETURN
  END;

evaluateExpList: PUBLIC PROCEDURE RETURNS [ESPointer] =
  BEGIN OPEN DIActionDefs, DebugUtilityDefs, DITypeDefs;
  size: hereESPointer ← LOOPHOLE[popevalstack[], hereESPointer];
  listsize: CARDINAL ← size.value;
  esp: ESPointer ← popNevalstack[listsize];
  testesp: ESPointer;
  FreeStackItem[size];
  IF TypeProcedure[esp] THEN RETURN[ProcedureCall[esp, listsize]];
  IF listsize # 1 THEN SIGNAL DIActionDefs.IncorrectType[esp];
  SELECT TRUE FROM
    TypeString[esp] => RETURN[GetStringElement[esp]];
    TypePointer[esp] =>
      BEGIN
      WITH p:(esp.stbase.seb+esp.stbase.UnderType[esp.tsei]) SELECT FROM
        pointer => IF p.basing THEN RETURN[GetRelativePointer[esp]];
        ENDCASE;
      testesp ← dereferenceItem[esp  ! ANY => GOTO null];
      RETURN[IF (TypeArrayDesc[testesp] OR TypeArray[testesp])
         THEN GetArrayElement[testesp] ELSE GetRelativePointer[testesp]];
      EXITS
        null => RETURN[GetRelativePointer[esp]];
      END;
    (TypeArrayDesc[esp] OR TypeArray[esp]) => RETURN[GetArrayElement[esp]];
    ENDCASE => SIGNAL DIActionDefs.IncorrectType[esp];
  END;

GetTheValue: PROCEDURE [esp: ESPointer] RETURNS [value: UNSPECIFIED] =
  BEGIN OPEN DIActionDefs;
  so: DebuggerDefs.SymbolObject;
  sop: SOPointer ← @so;
  WITH esp SELECT FROM
    here => RETURN[value];
    there =>
      BEGIN
      espTosop[esp,sop];
      RETURN[DebuggerDefs.GetValue[sop]];
      END;
    ENDCASE => ERROR;
  END;

startList: PUBLIC PROCEDURE [size: CARDINAL] RETURNS [new: hereESPointer] =
  BEGIN OPEN DIActionDefs;
  new ← AllocateHereStackItem[];
  new.tsei ← DITypeDefs.SeiPType[integer,DIActionDefs.GetCurrentST[]];
  new.value ← size;
  RETURN
  END;

incrementList: PUBLIC PROCEDURE =
  BEGIN OPEN DIActionDefs;
  new: hereESPointer ← LOOPHOLE[popNevalstack[1], hereESPointer];
  new.value ← new.value + 1;
  pushevalstack[new];
  RETURN
  END;

END..
```