```
-- file DIActionsHot.Mesa
-- Edited by:
--              Sandman, April 17, 1978  4:13 PM
--              Barbara, July 31, 1978  5:15 PM
--              Johnsson, August 29, 1978  9:52 AM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [wordlength],
  ControlDefs: FROM "controldefs" USING [
    FieldDescriptor, FrameHandle, GlobalFrameHandle],
  DebugContextDefs: FROM "debugcontextdefs" USING [IncorrectVersion],
  DebugData: FROM "debugdata" USING [gContext],
  DebuggerDefs: FROM "debuggerdefs" USING [
    fullbitaddress, fullsymaddress, InitSOP, LA, Lookup, LookupLocals,
    QualifyRecord, SA, SearchForBasicSym, SearchForModuleSym,
    SearchFrameForSym, SearchGFrameForSym, SOPointer, SymbolObject],
  DebugMiscDefs: FROM "debugmiscdefs" USING [
    DFreeString, DGetString, LookupFail],
  DebugSymbolDefs: FROM "debugsymboldefs" USING [
    DAcquireSymbolTable, DReleaseSymbolTable, SymbolsForGFrame],
  DebugUtilityDefs: FROM "debugutilitydefs" USING [
    CheckFrame, LongREAD, MREAD, ValidGlobalFrame],
  DIActionDefs: FROM "diactiondefs" USING [
    IncorrectType, InvalidExpression, litType, NotImplemented,
    ResetTypeStack],
  DIDefs: FROM "didefs" USING [
    ESPointer, EvalStackItem, hereESPointer, Operator, thereESPointer],
  DILitDefs: FROM "dilitdefs" USING [
    LiteralValue, LongLiteralValue, LTIndex, STIndex, StringLiteralValue],
  DITypeDefs: FROM "ditypedefs" USING [
    SeiLongInteger, SeiPType, TypeInteger, TypeIU, TypeIUP, TypeLong,
    TypePointer, TypeProcedure, TypeRecord, TypeUnspec],
  Mopcodes: FROM "mopcodes" USING [zRFS],
  StringDefs: FROM "stringdefs" USING [AppendSubString],
  SymbolTableDefs: FROM "symboltabledefs" USING [
    NoSymbolTable, SymbolTableBase],
  SymDefs: FROM "symdefs" USING [BitAddress, CSEIndex, ISENull, SENull],
  SystemDefs: FROM "systemdefs" USING [AllocateHeapNode, FreeHeapNode];

DIActionsHot: PROGRAM
  IMPORTS DebugContextDefs, DDptr: DebugData, DebuggerDefs, DebugMiscDefs,
    DebugSymbolDefs, DebugUtilityDefs, DIActionDefs, DILitDefs, DITypeDefs,
    StringDefs, SymbolTableDefs, SystemDefs
  EXPORTS DIActionDefs =
  BEGIN

--stack items
ESPointer: TYPE = DIDefs.ESPointer;
hereESPointer: TYPE = DIDefs.hereESPointer;
thereESPointer: TYPE = DIDefs.thereESPointer;
Operator: TYPE = DIDefs.Operator;
SOPointer: TYPE = DebuggerDefs.SOPointer;
currentST: SymbolTableDefs.SymbolTableBase ← NIL;

--stacks
MaxStackSize: CARDINAL = 10;
evalstack: ARRAY [1..MaxStackSize] OF ESPointer;
etop: CARDINAL ← 0;

EvalStackOverflow: PUBLIC SIGNAL = CODE;
EvalStackEmpty: PUBLIC SIGNAL = CODE;
NotOnEvalStack: PUBLIC SIGNAL = CODE;
NILesp: PUBLIC SIGNAL = CODE;

--eval stack manipulation
pushevalstack: PUBLIC PROCEDURE [esp: ESPointer] =
  BEGIN
  IF etop = MaxStackSize THEN SIGNAL EvalStackOverflow;
  etop ← etop + 1;
  evalstack[etop] ← esp;
  RETURN
  END;

popevalstack: PUBLIC PROCEDURE RETURNS [esp: ESPointer] =
  BEGIN
  IF etop = 0 THEN SIGNAL EvalStackEmpty;
  esp ← evalstack[etop];
  etop ← etop - 1;
  IF esp = NIL THEN SIGNAL NILesp;
  RETURN
  END;

popNevalstack: PUBLIC PROCEDURE [n: CARDINAL] RETURNS [esp: ESPointer] =
  BEGIN -- returns top-n from stack, adjusts stack
  i: CARDINAL;
  IF etop = n THEN SIGNAL NotOnEvalStack;
  esp ← evalstack[etop-n];
  IF esp = NIL THEN SIGNAL NILesp;
  FOR i DECREASING IN (0..n] DO
    evalstack[etop-i] ← evalstack[etop-i+1];
    ENDLOOP;
  etop ← etop - 1;
  RETURN
  END;

TypesDontMatch: PUBLIC SIGNAL [esp1, esp2: ESPointer] = CODE;

performAddOp: PUBLIC PROCEDURE [es2, es1: ESPointer, op: Operator]
  RETURNS [result: hereESPointer]=
  BEGIN OPEN DIActionDefs, DITypeDefs;
  left: hereESPointer ← Transfer[es1];
  right: hereESPointer ← Transfer[es2];
  leftptr: BOOLEAN ← TypePointer[left];
  rightptr: BOOLEAN ← TypePointer[right];
  leftLong: BOOLEAN ← TypeLong[left];
  rightLong: BOOLEAN ← TypeLong[right];
  SELECT op FROM
    plus =>
      BEGIN
      IF ~(TypeIUP[left] AND TypeIUP[right]) OR (rightptr AND leftptr)
        THEN SIGNAL TypesDontMatch[left,right];
      --subranges will get lost here
      SELECT TRUE FROM
        leftptr =>
          BEGIN --preserve pointer type
          IF rightLong AND ~leftLong THEN SIGNAL NotImplemented;
          result ← AllocateHereStackItem[];
          IF ~leftLong
            THEN result.value ← ActualValue[left] + ActualValue[right]
          ELSE BEGIN
            result.ptr ← SystemDefs.AllocateHeapNode[result.wordlength ← 2];
            LOOPHOLE[result.ptr, POINTER TO LONG INTEGER]↑
              ← LongValue[left] + LongValue[right];
            END;
          result.tsei ← left.tsei; result.indirection ← left.indirection;
          result.stbase ← left.stbase;
          END;
        rightptr =>
          BEGIN --preserve pointer type
          IF leftLong AND ~rightLong THEN SIGNAL NotImplemented;
          result ← AllocateHereStackItem[];
          IF ~rightLong
            THEN result.value ← ActualValue[left] + ActualValue[right]
          ELSE BEGIN
            result.ptr ← SystemDefs.AllocateHeapNode[result.wordlength ← 2];
            LOOPHOLE[result.ptr, POINTER TO LONG INTEGER]↑
              ← LongValue[left] + LongValue[right];
            END;
          result.tsei ← right.tsei; result.indirection ← right.indirection;
          result.stbase ← right.stbase;
          END;
        ENDCASE =>
          BEGIN
          result ← AllocateHereStackItem[];
          IF leftLong OR rightLong THEN
            BEGIN
            result.ptr ← SystemDefs.AllocateHeapNode[result.wordlength ← 2];
            LOOPHOLE[result.ptr, POINTER TO LONG INTEGER]↑
              ← LongValue[left] + LongValue[right];
            result.tsei ← SeiPType[longinteger, result.stbase ← NIL];
            END
          ELSE
          BEGIN
          result.value ← ActualValue[left] + ActualValue[right];
          result.tsei ← IF TypeInteger[left] OR TypeInteger[right]
            THEN SeiPType[integer,currentST]
          ELSE SeiPType[unspecified,currentST];
          END;
        END;
      END;
    minus =>
      BEGIN
      IF ~(TypeIUP[left] AND TypeIUP[right]) OR (rightptr AND ~leftptr)
        THEN SIGNAL TypesDontMatch[left,right];
      SELECT TRUE FROM
        (leftptr AND rightptr) =>
          BEGIN
          IF rightLong AND ~leftLong THEN SIGNAL NotImplemented;
          result ← AllocateHereStackItem[];
          IF ~leftLong THEN
            BEGIN
            result.value ← ActualValue[left] - ActualValue[right];
            result.tsei ← SeiPType[integer,currentST];
            END
          ELSE BEGIN
            result.ptr ← SystemDefs.AllocateHeapNode[result.wordlength ← 2];
            LOOPHOLE[result.ptr, POINTER TO LONG INTEGER]↑
              ← LongValue[left] - LongValue[right];
            result.tsei ← SeiPType[longinteger,result.stbase ← NIL];
            END;
          END;
        leftptr =>
          BEGIN --preserve pointer type
          IF ~leftLong AND rightLong THEN SIGNAL NotImplemented;
          result ← AllocateHereStackItem[];
          IF ~leftLong
            THEN result.value ← ActualValue[left] - ActualValue[right]
          ELSE BEGIN
            result.ptr ← SystemDefs.AllocateHeapNode[result.wordlength ← 2];
            LOOPHOLE[result.ptr, POINTER TO LONG INTEGER]↑
              ← LongValue[left] - LongValue[right];
            END;
          result.tsei ← left.tsei; result.indirection ← left.indirection;
          result.stbase ← left.stbase;
          END;
        ENDCASE =>
          BEGIN
          result ← AllocateHereStackItem[];
          IF leftLong THEN
            BEGIN
            result.ptr ← SystemDefs.AllocateHeapNode[result.wordlength ← 2];
            LOOPHOLE[result.ptr, POINTER TO LONG INTEGER]↑
              ← LongValue[left] - LongValue[right];
            result.tsei ← SeiPType[longinteger,result.stbase ← NIL];
            END
          ELSE
            BEGIN
            result.value ← ActualValue[left] - ActualValue[right];
            result.tsei ← IF TypeInteger[left] OR TypeInteger[right]
              THEN SeiPType[integer,currentST]
            ELSE SeiPType[unspecified,currentST];
            END;
          END;
      END;
    ENDCASE => ERROR;
  FreeStackItem[left]; FreeStackItem[right];
  RETURN
  END;

performMultOp:  PUBLIC PROCEDURE [es2, es1: ESPointer, op: Operator]
  RETURNS [result: hereESPointer]=
  BEGIN OPEN DITypeDefs;
  left: hereESPointer ← Transfer[es1];
  right: hereESPointer ← Transfer[es2];
  leftLong: BOOLEAN ← TypeLong[left];
  rightLong: BOOLEAN ← TypeLong[right];
  IF ~TypeIU[left] OR ~TypeIU[right] THEN SIGNAL TypesDontMatch[left, right];
  result ← AllocateHereStackItem[];
  SELECT op FROM
    times =>
      IF leftLong OR rightLong THEN
        BEGIN
        result.ptr ← SystemDefs.AllocateHeapNode[result.wordlength ← 2];
        LOOPHOLE[result.ptr, POINTER TO LONG INTEGER]↑
           ← LongValue[left] * LongValue[right];
        END
      ELSE result.value ← ActualValue[left] * ActualValue[right];
    div =>
      IF leftLong OR rightLong THEN
        BEGIN
        result.ptr ← SystemDefs.AllocateHeapNode[result.wordlength ← 2];
        LOOPHOLE[result.ptr, POINTER TO LONG INTEGER]↑
           ← LongValue[left] / LongValue[right];
        END
      ELSE result.value ← ActualValue[left] / ActualValue[right];
    mod =>
      IF leftLong OR rightLong THEN
        BEGIN
        result.ptr ← SystemDefs.AllocateHeapNode[result.wordlength ← 2];
        LOOPHOLE[result.ptr, POINTER TO LONG INTEGER]↑ ←
          LongValue[left] MOD LongValue[right];
        END
      ELSE result.value ← ActualValue[left] MOD ActualValue[right];
    ENDCASE => ERROR;
  result.tsei ← SELECT TRUE FROM
    (leftLong OR rightLong) => SeiPType[longinteger, result.stbase ← NIL],
    (TypeInteger[left] OR TypeInteger[right]) => SeiPType[integer,currentST],
    ENDCASE => SeiPType[unspecified,currentST];
  FreeStackItem[left]; FreeStackItem[right];
  RETURN
  END;

ActualValue: PUBLIC PROCEDURE [hesp: hereESPointer] RETURNS
  [value: UNSPECIFIED] =
  BEGIN
  IF hesp.stbase = NIL THEN RETURN[hesp.value];
  WITH hesp.stbase.seb+hesp.stbase.UnderType[hesp.tsei] SELECT FROM
    subrange =>
      IF origin # 0 THEN RETURN[hesp.value+origin];
    ENDCASE;
  RETURN[hesp.value];
  END;

LongValue: PUBLIC PROCEDURE [hesp: hereESPointer] RETURNS [LONG INTEGER] =
  BEGIN
  IF hesp.wordlength = 1 THEN RETURN[LONG[CARDINAL[hesp.value]]];
  RETURN[LOOPHOLE[hesp.ptr, POINTER TO LONG INTEGER]↑]
  END;

--perform an action on an eval stack item
qualifyItem: PUBLIC PROCEDURE [esp: ESPointer, id: DILitDefs.STIndex,
  locals: BOOLEAN] RETURNS [ESPointer] =
  BEGIN OPEN DebuggerDefs;
  so: SymbolObject;
  sop: SOPointer ← @so;
  bitaddr: SymDefs.BitAddress;
  val: UNSPECIFIED;
  local: BOOLEAN ← FALSE;
  fd: ControlDefs.FieldDescriptor;
  i, lengthOfFieldInRecord, sizeOfItemWithinField: CARDINAL;
  IF DITypeDefs.TypePointer[esp] THEN esp ← dereferenceItem[esp];
  espTosop[esp,sop];
  SELECT TRUE FROM
    DITypeDefs.TypeRecord[esp] =>
      IF ~QualifyRecord[sop, DILitDefs.StringLiteralValue[id]]
        THEN SIGNAL DIActionDefs.InvalidExpression;
    (locals AND DITypeDefs.TypeProcedure[esp]) =>
      IF ~LookupLocals[sop, DILitDefs.StringLiteralValue[id]]
        THEN SIGNAL DIActionDefs.InvalidExpression
      ELSE local ← TRUE;
    ENDCASE =>
      SIGNAL DIActionDefs.IncorrectType[esp];
  BEGIN OPEN t: esp.stbase, s: sop.stbase;
    bitaddr ← (s.seb+sop.sei).idvalue;
    lengthOfFieldInRecord ← (s.seb+sop.sei).idinfo;
    sizeOfItemWithinField ← s.BitsForType[sop.tsei];
    WITH e: esp SELECT FROM
      there =>
        BEGIN
        WITH e SELECT FROM
          short => IF local THEN e.addr ← short[shortAddr:[bitaddr.wd]]
            ELSE e.addr ← short[shortAddr: [shortAddr+bitaddr.wd]];
          long => e.addr ← long[longAddr: LA[LI[li:longAddr.li+bitaddr.wd]]];
          ENDCASE;
        e.bitoffset ← e.bitoffset + bitaddr.bd +
          lengthOfFieldInRecord - sizeOfItemWithinField;
        e.bitsize ← sizeOfItemWithinField;
        END;
      here =>
        BEGIN OPEN AltoDefs;
        SELECT sizeOfItemWithinField FROM
          < wordlength =>
            BEGIN
            fd.offset ← bitaddr.wd;
            fd.size ← sizeOfItemWithinField;
            fd.posn ← bitaddr.bd +
              lengthOfFieldInRecord - sizeOfItemWithinField;
            val ← ReadField[IF e.wordlength = 1 THEN @e.value ELSE e.ptr, fd];
            IF e.wordlength # 1 THEN
              BEGIN
              SystemDefs.FreeHeapNode[e.ptr];
              e.wordlength ← 1;
              END;
            e.value ← val
            END;
          = wordlength =>
            IF e.wordlength # 1 THEN
              BEGIN
              val ← (e.ptr + bitaddr.wd)↑;
              SystemDefs.FreeHeapNode[e.ptr];
              e.wordlength ← 1;
              e.value ← val
              END;
          ENDCASE =>
            BEGIN
            e.wordlength ← sizeOfItemWithinField/wordlength;
            val ← SystemDefs.AllocateHeapNode[e.wordlength];
            FOR i IN [0..e.wordlength) DO
              LOOPHOLE[val+i, POINTER]↑ ← (e.ptr + bitaddr.wd + i)↑;
              ENDLOOP;
            SystemDefs.FreeHeapNode[e.ptr];
            e.ptr ← val;
            END;
        END;
      ENDCASE => ERROR;
    esp.stbase ← sop.stbase; esp.tsei ← sop.tsei;
    --necessary for correct field extraction on records
    esp.sei ← IF ~local THEN SymDefs.ISENull ELSE sop.sei;
    END;
  RETURN[esp]
  END;

dereferenceItem: PUBLIC PROCEDURE [esp: ESPointer] RETURNS [tesp: thereESPointer] =
  BEGIN OPEN s:esp.stbase, DITypeDefs, DebugUtilityDefs;
  type: SymDefs.CSEIndex;
  long: BOOLEAN ← FALSE;
  IF TypeUnspec[esp] THEN esp.indirection ← 1;
  IF ~(TypePointer[esp] OR esp.indirection # 0)
    THEN SIGNAL DIActionDefs.IncorrectType[esp];
  tesp ← AllocateThereStackItem[];
  IF esp.indirection > 0 THEN
    BEGIN
    WITH e:esp SELECT FROM
      here =>
        BEGIN
        tesp↑ ← [next:, stbase: e.stbase, sei: SymDefs.ISENull, tsei: e.tsei,
          desc: e.desc, intN: e.intN, indirection: e.indirection-1,
          body: there[bitoffset: 0, bitsize: AltoDefs.wordlength,
          addr: short[shortAddr: e.value]]];
        IF e.stbase # NIL THEN tesp.bitsize ← e.stbase.BitsForType[e.tsei]
        ELSE IF e.tsei = SeiLongInteger THEN
          tesp.bitsize ← 2 * AltoDefs.wordlength;
        END;
      ENDCASE => SIGNAL DIActionDefs.InvalidExpression;
    RETURN
    END;
  type ← s.UnderType[esp.tsei];
  DO
    WITH s.seb+type SELECT FROM
      subrange => type ← s.UnderType[rangetype];
      long => BEGIN long ← TRUE; type ← s.UnderType[rangetype]; END;
      pointer => BEGIN esp.tsei ← pointedtotype; EXIT END;
      ENDCASE => ERROR;
    ENDLOOP;
  tesp↑ ← [next:, stbase: esp.stbase, sei: SymDefs.ISENull, tsei: esp.tsei,
    desc: esp.desc, intN: esp.intN, indirection: 0, body: there[bitoffset: 0,
    addr:, bitsize: esp.stbase.BitsForType[esp.tsei]]];
  WITH e:esp SELECT FROM
    here => tesp.addr ← short[shortAddr:
      IF e.wordlength = 1 THEN e.value ELSE e.ptr↑];
    there => WITH e SELECT FROM
      short => IF ~long THEN
        tesp.addr ← short[shortAddr:MREAD[shortAddr]]
        ELSE BEGIN
          la: LA DebuggerDefs.LA;
          la.low ←  MREAD[shortAddr];
          la.high ← MREAD[shortAddr+1];
          tesp.addr ← long[longAddr:la];
          END;
      long => IF ~long THEN
        tesp.addr ← short[shortAddr:LongREAD[longAddr.lp]]
        ELSE BEGIN
          la: DebuggerDefs.LA;
          la.low ←  LongREAD[longAddr.lp];
          la.high ← LongREAD[longAddr.lp+1];
          tesp.addr ← long[longAddr:la];
          END;
      ENDCASE;
    ENDCASE;
  FreeStackItem[esp];
  RETURN
  END;

--handle literals
getLiteral: PUBLIC PROCEDURE [type: DIActionDefs.litType, value: DILitDefs.LTIndex]
  RETURNS [new: hereESPointer] =
  BEGIN
  new ← AllocateHereStackItem[];
  new.value ← DILitDefs.LiteralValue[value];
  new.tsei ← SELECT type FROM
    num => DITypeDefs.SeiPType[integer,currentST],
    ENDCASE => DITypeDefs.SeiPType[character,currentST];
  RETURN
  END;

getLongLiteral: PUBLIC PROCEDURE [value: DILitDefs.LTIndex]
  RETURNS [new: hereESPointer] =
  BEGIN
  new ← AllocateHereStackItem[];
  new.ptr ← SystemDefs.AllocateHeapNode[new.wordlength ← 2];
  LOOPHOLE[new.ptr, POINTER TO LONG INTEGER]↑ ←
    DILitDefs.LongLiteralValue[value];
  new.tsei ← DITypeDefs.SeiPType[longinteger,new.stbase ← NIL];
  RETURN
  END;

getStringLiteral: PUBLIC PROCEDURE [value: DILitDefs.STIndex]
  RETURNS [new: hereESPointer] =
  BEGIN
  new ← AllocateHereStackItem[];
  new.value ← DILitDefs.StringLiteralValue[value];
  new.tsei ← DITypeDefs.SeiPType[string, currentST];
  RETURN
  END;

--symboltable manipulation
```

```
LookupId: PUBLIC PROCEDURE [id: DILitDefs.STIndex] RETURNS [ESPointer] =
  BEGIN OPEN DebuggerDefs;
  s: STRING ← DebugMiscDefs.DGetString[30];
  so: SymbolObject;
  sop: SOPointer ← @so;
  tesp: thereESPointer;
  hesp: hereESPointer;
  found, constant, transfer: BOOLEAN;
  InitSOP[sop];
  StringDefs.AppendSubString[s, DILitDefs.StringLiteralValue[id]];
  IF (found ← Lookup[s, FALSE, sop, FALSE, mod]) THEN
    BEGIN
    constant ← (sop.stbase.seb+sop.sei).constant;
    transfer ← WITH sop.stbase.seb+sop.stbase.UnderType[sop.tsei] SELECT FROM
      transfer => TRUE,
      ENDCASE => FALSE;
    IF ~constant OR (constant AND transfer) THEN
      BEGIN
      tesp ← AllocateThereStackItem[];
      sopToesp[sop,tesp];
      IF ~constant AND ~transfer THEN tesp.sei ← SymDefs.ISENull;
      DebugMiscDefs.DFreeString[s];
      RETURN[tesp];
      END;
    END;
  IF (found AND constant) OR SearchForBasicSym[s, sop] THEN
    BEGIN
    hesp ← AllocateHereStackItem[];
    hesp.stbase ← sop.stbase;
    hesp.sei ← sop.sei;
    hesp.tsei ← sop.tsei;
    DebugMiscDefs.DFreeString[s];
    IF ~(sop.stbase.seb+sop.sei).extended THEN
      BEGIN
      hesp.wordlength ← 1;
      hesp.value ← (sop.stbase.seb+sop.sei).idvalue;
      END
    ELSE SIGNAL DIActionDefs.NotImplemented; --multiword constants
    RETURN[hesp];
    END;
  SIGNAL DebugMiscDefs.LookupFail[s];
  END;

SearchFrameForId: PUBLIC PROCEDURE [num: DILitDefs.LTIndex, id: DILitDefs.STIndex]
  RETURNS [ESPointer] =
  BEGIN OPEN DebuggerDefs;
  gframe: ControlDefs.GlobalFrameHandle
    ← LOOPHOLE[DILitDefs.LiteralValue[num], ControlDefs.GlobalFrameHandle];
  sym: STRING ← DebugMiscDefs.DGetString[30];
  frame: ControlDefs.FrameHandle ← LOOPHOLE[DILitDefs.LiteralValue[num]];
  so: SymbolObject;
  sop: SOPointer ← @so;
  InitSOP[sop];
  StringDefs.AppendSubString[sym, DILitDefs.StringLiteralValue[id]];
  IF DebugUtilityDefs.ValidGlobalFrame[gframe] THEN
    BEGIN
    IF ~SearchGFrameForSym[gframe, sym,  FALSE, sop, FALSE] THEN
      SIGNAL DebugMiscDefs.LookupFail[sym]
    END
  ELSE IF DebugUtilityDefs.CheckFrame[frame] THEN
    BEGIN
    IF ~SearchFrameForSym[frame, sym,  FALSE, sop, FALSE] THEN
      SIGNAL DebugMiscDefs.LookupFail[sym]
    END
  ELSE SIGNAL DIActionDefs.InvalidExpression;
  DebugMiscDefs.DFreeString[sym];
  RETURN[SetUpId[sop]]
  END;

SetUpId: PROCEDURE [sop: DebuggerDefs.SOPointer] RETURNS [ESPointer] =
  BEGIN
  tesp: thereESPointer;
  hesp: hereESPointer;
  constant, transfer: BOOLEAN ← FALSE;
  constant ← (sop.stbase.seb+sop.sei).constant;
  WITH sop.stbase.seb+sop.stbase.UnderType[sop.tsei] SELECT FROM
```

```
    transfer => transfer ← TRUE;
    ENDCASE;
  IF ~constant OR (constant AND transfer) THEN
    BEGIN
    tesp ← AllocateThereStackItem[];
    sopToesp[sop,tesp];
    tesp.sei ← SymDefs.ISENull;
    RETURN[tesp];
    END;
  hesp ← AllocateHereStackItem[];
  hesp.stbase ← sop.stbase;
  hesp.sei ← sop.sei;
  hesp.tsei ← sop.tsei;
  hesp.wordlength ← 1;
  hesp.value ← (sop.stbase.seb+sop.sei).idvalue;
  RETURN[hesp];
  END;

SearchFileForId: PUBLIC PROCEDURE [file, id: DILitDefs.STIndex]
  RETURNS [ESPointer] =
  BEGIN OPEN DebugMiscDefs, DebuggerDefs;
  mod: STRING ← DGetString[30];
  type: STRING ← DGetString[30];
  so: SymbolObject;
  sop: SOPointer ← @so;
  InitSOP[sop];
  StringDefs.AppendSubString[mod, DILitDefs.StringLiteralValue[file]];
  StringDefs.AppendSubString[type, DILitDefs.StringLiteralValue[id]];
  IF ~SearchForModuleSym[mod, type, FALSE, sop, FALSE] THEN
    BEGIN
    DFreeString[mod];
    SIGNAL DebugMiscDefs.LookupFail[type];
    END;
  DFreeString[mod];
  DFreeString[type];
  RETURN[SetUpId[sop]]
  END;

--conversion utilities
espTosop: PUBLIC PROCEDURE [esp: ESPointer, sop: SOPointer] =
  BEGIN OPEN DebuggerDefs;
  sym: fullbitaddress;
  sa: SA;
  InitSOP[sop];
  sop.stbase ← esp.stbase;
  sop.sei ← esp.sei;
  sop.tsei ← esp.tsei;
  sym ← fullsymaddress[sop];
  WITH sym SELECT FROM
    short => sa ← shortAddr;
    ENDCASE => ERROR;
  WITH e: esp SELECT FROM
    here =>
      BEGIN
      sop.baddr.wd ← short[shortAddr: [LOOPHOLE[
        (IF e.wordlength = 1 THEN @e.value ELSE e.value), SA] - sa]];
      sop.there ← FALSE;
      END;
    there =>
      BEGIN
      WITH e SELECT FROM
        short => sop.baddr.wd ← short[shortAddr:
          [shortAddr-sa]];
        long => sop.baddr.wd ← long[longAddr: LA[LI[li:longAddr.li-sa]]];
        ENDCASE;
      sop.baddr.bd ← e.bitoffset;
      sop.space ← e.bitsize MOD 16;
      END;
    ENDCASE => ERROR;
  RETURN
  END;

sopToesp: PUBLIC PROCEDURE [sop: SOPointer, tesp: thereESPointer] =
  BEGIN OPEN DebuggerDefs, sop.stbase;
  sa: SA;
  sym: fullbitaddress ← fullsymaddress[sop];
  WITH sym SELECT FROM
    short => sa ← shortAddr;
    ENDCASE => ERROR;
  tesp.stbase ← sop.stbase;
  tesp.sei ← sop.sei;
  tesp.tsei ← sop.tsei;
  tesp.bitsize ← BitsForType[sop.tsei];
  tesp.bitoffset ← IF tesp.bitsize < AltoDefs.wordlength
    THEN (AltoDefs.wordlength - tesp.bitsize) ELSE 0;
  WITH sop.baddr SELECT FROM
    short => tesp.addr ← short[shortAddr: [shortAddr+sa]];
    long => tesp.addr ← long[longAddr: LA[LI[li:longAddr.li+sa]]];
    ENDCASE;
  RETURN
  END;

Transfer: PUBLIC PROCEDURE [esp: ESPointer] RETURNS [newesp: hereESPointer] =
    BEGIN OPEN DebugUtilityDefs, DIDefs;
    i: CARDINAL;
    fd: ControlDefs.FieldDescriptor;
    WITH e:esp SELECT FROM
      here => RETURN[@e];
      there =>
        BEGIN
        newesp ← AllocateHereStackItem[];
        newesp↑ ← EvalStackItem[next:,stbase: e.stbase, sei: SymDefs.ISENull,
          tsei: e.tsei, desc: e.desc, intN: e.intN,
          indirection: e.indirection, body: here[wordlength:, data:]];
        IF e.bitsize <= AltoDefs.wordlength THEN
          BEGIN
          newesp.wordlength ← 1;
          WITH e SELECT FROM
            short => i ← MREAD[shortAddr];
            long => i ← LongREAD[longAddr.lp];
            ENDCASE;
          fd ← [offset: 0, posn: e.bitoffset, size: e.bitsize];
          newesp.value ← ReadField[@i, fd];
          END
        ELSE
          BEGIN
          IF e.bitsize MOD AltoDefs.wordlength # 0 OR e.bitoffset # 0
            THEN ERROR;
          newesp.wordlength ← e.bitsize/AltoDefs.wordlength;
          newesp.ptr ← SystemDefs.AllocateHeapNode[newesp.wordlength];
          FOR i IN [0..newesp.wordlength) DO -- use val for loop counter
            WITH e SELECT FROM
              short => (newesp.ptr+i)↑ ← MREAD[shortAddr+i];
              long => (newesp.ptr+i)↑ ← LongREAD[longAddr.lp+i];
              ENDCASE;
            ENDLOOP;
          END;
        END;
      ENDCASE;
    FreeStackItem[esp];
    RETURN[newesp]
    END;

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

LA: TYPE = DebuggerDefs.LA;

--initialization and reset
GetSetUp: PUBLIC PROCEDURE =
  BEGIN OPEN DebugSymbolDefs;
  BEGIN --only valid HERE !!!
  IF DDptr.gContext # NIL THEN
    currentST ← DAcquireSymbolTable[SymbolsForGFrame[DDptr.gContext
       ! SymbolTableDefs.NoSymbolTable => GOTO nosym;
         DebugContextDefs.IncorrectVersion => RESUME]
     ! SymbolTableDefs.NoSymbolTable => GOTO nosym]
  ELSE currentST ← NIL;
  EXITS
  --this is a problem - what if no symboltable - try alittle harder ??
    nosym => currentST ← NIL;
  END;
  RETURN
  END;

GetCurrentST: PUBLIC PROCEDURE RETURNS [SymbolTableDefs.SymbolTableBase] =
  BEGIN
  RETURN[currentST]
  END;

CleanUp: PUBLIC PROCEDURE =
  BEGIN
  IF currentST # NIL THEN
    BEGIN DebugSymbolDefs.DReleaseSymbolTable[currentST]; currentST ← NIL; END;
  ResetStacks[];
  RETURN
  END;

ResetStacks: PUBLIC PROCEDURE =
  BEGIN
  esp: ESPointer ← EvalStackList;
  nesp: ESPointer;
  UNTIL esp = NIL DO
    nesp ← esp.next;
    WITH e: esp SELECT FROM
      here => IF e.wordlength > 1 AND e.ptr # NIL
        THEN SystemDefs.FreeHeapNode[e.ptr];
      ENDCASE;
    SystemDefs.FreeHeapNode[esp];
    esp ← nesp;
    ENDLOOP;
  EvalStackList ← NIL; etop ← 0;
  DIActionDefs.ResetTypeStack[];
  RETURN
  END;

EvalStackList: ESPointer ← NIL;

AllocateHereStackItem: PUBLIC PROCEDURE RETURNS [hesp: hereESPointer] =
  BEGIN OPEN DIDefs;
  hesp ← SystemDefs.AllocateHeapNode[SIZE[here EvalStackItem]];
  hesp↑ ← EvalStackItem[next: EvalStackList, stbase: currentST,
      sei: SymDefs.ISENull, tsei: SymDefs.SENull, desc: FALSE, intN: FALSE,
      indirection: 0, body: here[wordlength:1, data:]];
  EvalStackList ← hesp;
  RETURN
  END;

AllocateThereStackItem: PUBLIC PROCEDURE RETURNS [tesp: thereESPointer] =
  BEGIN OPEN DIDefs;
  tesp ← SystemDefs.AllocateHeapNode[SIZE[there EvalStackItem]];
  tesp↑ ← EvalStackItem[next: EvalStackList, stbase: currentST,
      sei: SymDefs.ISENull, tsei: SymDefs.SENull, desc: FALSE, intN: FALSE,
      indirection: 0, body: there[bitoffset:0, addr: short[shortAddr:[0]],
      bitsize: 0]];
  EvalStackList ← tesp;
  RETURN
  END;

FreeStackItem: PUBLIC PROCEDURE [esp: ESPointer] =
  BEGIN
  dl: ESPointer ← EvalStackList;
  pdl: ESPointer ← NIL;
  UNTIL dl = NIL DO
    IF dl = esp THEN
      BEGIN
      IF pdl = NIL THEN EvalStackList ← dl.next ELSE pdl.next ← dl.next;
      WITH e: esp SELECT FROM
        here =>
          IF e.wordlength > 1 AND e.ptr # NIL THEN SystemDefs.FreeHeapNode[e.ptr];
        ENDCASE;
      SystemDefs.FreeHeapNode[esp];
      RETURN
      END;
    pdl ← dl; dl ← dl.next;
    ENDLOOP;
  RETURN
  END;
```

```
END..
```