```
-- file DIActionsCold.Mesa
-- Edited by:
--          Johnsson, August 29, 1978  10:02 AM
--          Barbara, July 31, 1978  4:36 PM

DIRECTORY
  DebuggerDefs: FROM "debuggerdefs" USING [
    GetValue, InitSOP, LA, Lookup, SearchForModuleSym, SOPointer,
    SymbolObject, VariantRecord, WriteSubString],
  DebugInterpretDefs: FROM "debuginterpretdefs" USING [IarrayPtr],
  DebugMiscDefs: FROM "debugmiscdefs" USING [
    DFreeString, DGetString, DWriteLongPointer, LookupFail, WriteEOL],
  DebugUtilityDefs: FROM "debugutilitydefs" USING [
    LongREAD, MREAD, UserWriteSubString],
  DIActionDefs: FROM "diactiondefs" USING [
    ActualValue, AllocateHereStackItem, espTosop,
    FreeStackItem, GetCurrentST, LongValue, NotImplemented, pushevalstack,
    Transfer],
  DIDefs: FROM "didefs" USING [
    ESPointer, hereESPointer, MaxIndirections, Operator, predefinedType,
    thereESPointer, TIPointer, TypeItem],
  DILitDefs: FROM "dilitdefs" USING [STIndex, StringLiteralValue],
  DITypeDefs: FROM "ditypedefs" USING [
    SeiLongInteger, SeiPType, TypeArray, TypeArrayDesc, TypeIU, TypeIUP,
    TypeLong, TypePointer, TypeString, TypeUnspec],
  IODefs: FROM "iodefs" USING [NumberFormat, WriteChar, WriteNumber],
  StringDefs: FROM "stringdefs" USING [
    AppendSubString, SubString, SubStringDescriptor],
  SymDefs: FROM "symdefs" USING [SENull],
  SystemDefs: FROM "systemdefs" USING [AllocateHeapNode, FreeHeapNode];

DIActionsCold: PROGRAM
  IMPORTS DebuggerDefs, DebugInterpretDefs, DebugMiscDefs, DebugUtilityDefs,
    DIActionDefs, DILitDefs, DITypeDefs, IODefs, StringDefs, SystemDefs
  EXPORTS DIActionDefs =
  BEGIN

--using grammar version 7

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

--stack and index for stack
MaxStackSize: CARDINAL = 5;
typestack: ARRAY [1..MaxStackSize] OF TIPointer;
ttop: CARDINAL ← 0;

--type stack manipulation
TypeStackOverflow:  PUBLIC SIGNAL = CODE;
TypeStackEmpty:  PUBLIC SIGNAL = CODE;

pushtypestack: PUBLIC PROCEDURE [tip: TIPointer] =
  BEGIN
  IF ttop = MaxStackSize THEN SIGNAL TypeStackOverflow;
  ttop ← ttop + 1;
  typestack[ttop] ← tip;
  RETURN
  END;

poptypestack: PUBLIC PROCEDURE RETURNS [tip: TIPointer] =
  BEGIN
  IF ttop = 0 THEN SIGNAL TypeStackEmpty;
  tip ← typestack[ttop];
  ttop ← ttop - 1;
  RETURN
  END;

loopholeItem:  PUBLIC PROCEDURE [esp: ESPointer, tip: TIPointer]
  RETURNS [ESPointer] =
  BEGIN OPEN s1: esp.stbase, s2: tip.stbase;
  esp.stbase ← tip.stbase;
  esp.tsei ← tip.tsei;
  esp.indirection ← tip.indirection;
  FreeTypeItem[tip];
  RETURN[esp]
  END;

loopholeUnspecItem:  PUBLIC PROCEDURE [esp: ESPointer] RETURNS [ESPointer] =
  BEGIN
  esp.tsei ← DITypeDefs.SeiPType[unspecified, esp.stbase];
  esp.indirection ← 0; esp.intN ← esp.desc ← FALSE;
  RETURN[esp]
  END;

minusItem: PUBLIC PROCEDURE [esp: ESPointer] RETURNS [new: hereESPointer] =
  BEGIN OPEN DIActionDefs;
  IF ~DITypeDefs.TypeIU[esp] THEN SIGNAL IncorrectType[esp];
  new ← Transfer[esp];
  IF new.wordlength = 1 THEN
    BEGIN
    new.value ← -ActualValue[new];
    WITH new.stbase.seb+new.stbase.UnderType[new.tsei] SELECT FROM
      subrange =>
        IF origin # 0 THEN new.value ← new.value-origin;
      ENDCASE;
    END
  ELSE LOOPHOLE[new.ptr, POINTER TO LONG INTEGER]↑ ← - LongValue[new];
  RETURN
  END;

TooManyIndirections: PUBLIC SIGNAL = CODE;

addressofItem: PUBLIC PROCEDURE [tesp: thereESPointer]
  RETURNS [new: hereESPointer] =
  BEGIN
  IF tesp.indirection = DIDefs.MaxIndirections THEN
    BEGIN DIActionDefs.FreeStackItem[tesp]; SIGNAL TooManyIndirections; END;
  new ← DIActionDefs.AllocateHereStackItem[];
  WITH tesp SELECT FROM
    short => new.value ← shortAddr;
    long =>
      BEGIN
      new.ptr ← SystemDefs.AllocateHeapNode[new.wordlength ← 2];
      LOOPHOLE[new.ptr, POINTER TO DebuggerDefs.LA]↑ ← longAddr;
      END;
    ENDCASE;
  new.stbase ← tesp.stbase;
  new.tsei ← tesp.tsei;
  new.indirection ← tesp.indirection + 1;
  DIActionDefs.FreeStackItem[tesp];
  RETURN
  END;

IncorrectType: PUBLIC SIGNAL [esp: ESPointer] = CODE;

--built in calls
lengthItem: PUBLIC PROCEDURE [esp: ESPointer] RETURNS [new: hereESPointer] =
  BEGIN OPEN s: esp.stbase, DITypeDefs, DIActionDefs, DebugUtilityDefs;
  IF ~(TypeArray[esp] OR TypeArrayDesc[esp]) THEN SIGNAL IncorrectType[esp];
  new ← AllocateHereStackItem[];
  IF TypeArrayDesc[esp] THEN
    WITH e:esp SELECT FROM
      here => new.value ← (e.ptr+1)↑;
      there => WITH esp.stbase.seb+esp.stbase.UnderType[esp.tsei] SELECT FROM
        long => WITH e SELECT FROM
          short => new.value ← MREAD[shortAddr+2];
          long => new.value ← LongREAD[longAddr.lp+2];
          ENDCASE;
        arraydesc => WITH e SELECT FROM
          short => new.value ← MREAD[shortAddr+1];
          long => new.value ← LongREAD[longAddr.lp+1];
          ENDCASE;
        ENDCASE => ERROR;
      ENDCASE => ERROR
  ELSE
    WITH esp SELECT FROM
      there => WITH a: s.seb+s.UnderType[esp.tsei] SELECT FROM
        array => new.value ← s.Cardinality[a.indextype];
        ENDCASE => ERROR;
      ENDCASE => ERROR;
  new.tsei ← SeiPType[integer, DIActionDefs.GetCurrentST[]];
  FreeStackItem[esp];
  RETURN
  END;

baseItem: PUBLIC PROCEDURE [esp: ESPointer] RETURNS [new: hereESPointer] =
  BEGIN OPEN DITypeDefs, DIActionDefs, DebugUtilityDefs;
  IF ~(TypeArray[esp] OR TypeArrayDesc[esp]) THEN SIGNAL IncorrectType[esp];
  new ← AllocateHereStackItem[];
  IF TypeArrayDesc[esp] THEN
    WITH e:esp SELECT FROM
      here => new.value ← (e.ptr)↑;
      there => WITH e SELECT FROM
        short => new.value ← MREAD[shortAddr];
        long => new.value ← LongREAD[longAddr.lp];
        ENDCASE;
      ENDCASE => ERROR
  ELSE
    WITH e:esp SELECT FROM
      there => WITH e SELECT FROM
        short => new.value ← shortAddr;
        long =>
          BEGIN
          new.ptr ← SystemDefs.AllocateHeapNode[new.wordlength ← 2];
          LOOPHOLE[new.ptr, POINTER TO DebuggerDefs.LA]↑ ← longAddr;
          END;
        ENDCASE;
      ENDCASE => ERROR;
  new.tsei ← SeiPType[unspecified, DIActionDefs.GetCurrentST[]];
  new.indirection ← 1;
  FreeStackItem[esp];
  RETURN
  END;

desc1Item: PUBLIC PROCEDURE [name: thereESPointer]
  RETURNS [new: hereESPointer] =
  BEGIN OPEN n: name.stbase, DIActionDefs;
  csize: CARDINAL;
  IF ~DITypeDefs.TypeArray[name] THEN SIGNAL IncorrectType[name];
  new ← AllocateHereStackItem[];
  new.desc ← TRUE;
  new.ptr ← SystemDefs.AllocateHeapNode[new.wordlength ← 2];
  WITH name SELECT FROM
    short => new.ptr↑ ← DebugUtilityDefs.MREAD[shortAddr];
    long => new.ptr↑ ← DebugUtilityDefs.LongREAD[longAddr.lp];
    ENDCASE;
  WITH (n.seb+n.UnderType[name.tsei]) SELECT FROM
    array => csize ← n.WordsForType[componenttype];
    ENDCASE => ERROR;
  (new.ptr+1)↑ ← (n.seb+name.sei).idinfo / csize;
  FreeStackItem[name];
  RETURN
  END;

desc2Item: PUBLIC PROCEDURE [length, base: ESPointer]
  RETURNS [new: hereESPointer] =
  BEGIN OPEN DITypeDefs, DIActionDefs;
  h1: hereESPointer;
  h2: hereESPointer;
  IF ~TypeIU[length] THEN SIGNAL IncorrectType[length];
  IF ~(TypePointer[base] OR TypeUnspec[base]) THEN SIGNAL IncorrectType[base];
  IF TypeLong[base] THEN SIGNAL DIActionDefs.NotImplemented;
  h1 ← Transfer[base]; h2 ← Transfer[length];
  new ← AllocateHereStackItem[];
  new.desc ← TRUE;
  new.ptr ← SystemDefs.AllocateHeapNode[new.wordlength ← 2];
  new.ptr↑ ← DIActionDefs.ActualValue[h1];
  (new.ptr+1)↑ ← DIActionDefs.ActualValue[h2];
  FreeStackItem[h1]; FreeStackItem[h2];
  RETURN
  END;

memItem: PUBLIC PROCEDURE [esp: ESPointer] RETURNS [new: hereESPointer] =
  BEGIN OPEN DIActionDefs;
  IF ~DITypeDefs.TypeIUP[esp] THEN SIGNAL IncorrectType[esp];
  new ← AllocateHereStackItem[];
  new ← Transfer[esp];
  new.tsei ← DITypeDefs.SeiPType[unspecified, DIActionDefs.GetCurrentST[]];
  new.value ← DebugUtilityDefs.LongREAD[LOOPHOLE[LongValue[new]]];
  RETURN
  END;

--type operations
typeOp: PUBLIC PROCEDURE [typeop: DIDefs.Operator, tip: TIPointer]
  RETURNS [new: hereESPointer] =
  BEGIN OPEN t:tip.stbase;
  new ← DIActionDefs.AllocateHereStackItem[];
  SELECT typeop FROM
    size =>
      BEGIN
      new.tsei ← DITypeDefs.SeiPType[integer,DIActionDefs.GetCurrentST[]];
      new.value ← IF tip.stbase # NIL THEN t.WordsForType[tip.tsei]
        ELSE IF tip.tsei = DITypeDefs.SeiLongInteger THEN 2 ELSE 1;
      END;
    ENDCASE => ERROR;
  FreeTypeItem[tip];
  RETURN
  END;

setPredefined: PUBLIC PROCEDURE [type: DIDefs.predefinedType]
  RETURNS [tip: TIPointer] =
  BEGIN
  tip ← AllocateTypeItem[];
  tip.tsei ← DITypeDefs.SeiPType[type, DIActionDefs.GetCurrentST[]];
  RETURN
  END;

SearchFileForType: PUBLIC PROCEDURE [file, id: DILitDefs.STIndex]
  RETURNS [tip: TIPointer] =
  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, TRUE] THEN
    BEGIN
    DFreeString[mod];
    SIGNAL DebugMiscDefs.LookupFail[type];
    END;
  tip ← AllocateTypeItem[];
  tip.stbase ← sop.stbase;
  tip.tsei ← sop.sei;
  DFreeString[mod];
  DFreeString[type];
  RETURN
  END;

SearchForType: PUBLIC PROCEDURE [id: DILitDefs.STIndex]
  RETURNS [tip: TIPointer] =
  BEGIN OPEN DebuggerDefs;
  s: STRING ← DebugMiscDefs.DGetString[30];
  so: SymbolObject;
  sop: SOPointer ← @so;
  InitSOP[sop];
  StringDefs.AppendSubString[s, DILitDefs.StringLiteralValue[id]];
  IF ~Lookup[s, FALSE, sop, TRUE, mod]
    THEN SIGNAL DebugMiscDefs.LookupFail[s];
  tip ← AllocateTypeItem[];
  tip.stbase ← sop.stbase;
  tip.tsei ← sop.sei;
  DebugMiscDefs.DFreeString[s];
  RETURN
  END;

InvalidType: PUBLIC SIGNAL [tip: TIPointer] = CODE;

SearchForVariantType: PUBLIC PROCEDURE [var: DILitDefs.STIndex, tip: TIPointer]
```

```
  RETURNS [TIPointer] =
  BEGIN OPEN DebuggerDefs;
  so: SymbolObject;
  sop: SOPointer ← @so;
  InitSOP[sop];
  sop.stbase ← tip.stbase; sop.tsei ← tip.tsei;
  IF sop.stbase = NIL OR ~VariantRecord[sop,DILitDefs.StringLiteralValue[var]]
    THEN SIGNAL InvalidType[tip]
  ELSE BEGIN tip.stbase ← sop.stbase; tip.tsei ← sop.sei; END;
  RETURN[tip]
  END;

pointertoType: PUBLIC PROCEDURE [tip: TIPointer] RETURNS [TIPointer] =
  BEGIN
  IF tip.indirection < DIDefs.MaxIndirections
    THEN tip.indirection ← tip.indirection + 1
  ELSE BEGIN
    FreeTypeItem[tip];
    SIGNAL TooManyIndirections;
    END;
  RETURN[tip]
  END;

--new interval notation
setIntervalBit: PUBLIC PROCEDURE [esp: ESPointer] RETURNS [ESPointer] =
  BEGIN
  esp.intN ← TRUE;
  RETURN[esp]
  END;

InvalidLongInterval: PUBLIC SIGNAL [left, right: LONG INTEGER] = CODE;

printOctal: PUBLIC PROCEDURE [n, start: ESPointer] =
  BEGIN OPEN IODefs, DITypeDefs, DIActionDefs;
  i: INTEGER ← -1;
  count: INTEGER;
  j, end: LONG INTEGER;
  leftSide: LONG POINTER;
  side2: hereESPointer;
  IF ~TypeIU[n] THEN SIGNAL IncorrectType[n];
  IF ~TypeIUP[start] THEN SIGNAL IncorrectType[start];
  leftSide ← LOOPHOLE[LongValue[Transfer[start]]];
  side2 ← Transfer[n];
  IF side2.intN THEN end ← LongValue[side2]
  ELSE end ← LongValue[side2] - LOOPHOLE[leftSide, LONG INTEGER] +1;
  --IF leftSide > rightSide
    --THEN SIGNAL InvalidLongInterval[leftSide, rightSide];
  FOR j ← 0, j + 1 UNTIL j = end DO
    IF (i ← i+ 1) MOD 8 = 0 THEN
      BEGIN
      DebugMiscDefs.WriteEOL[];
      DebugMiscDefs.DWriteLongPointer[leftSide + j, 8];
      WriteChar['/]
      END;
    WriteChar[' ];
    WriteNumber[count ← DebugUtilityDefs.LongREAD[LOOPHOLE[leftSide + j]], NumberFormat[8,FALSE,TRUE,6]
**];
    WriteChar[IF count ~IN[0..7] THEN 'B ELSE ' ];
    ENDLOOP;
  --note to stop after interval
  pushevalstack[setIntervalBit[side2]];
  RETURN
  END;

InvalidInterval: PUBLIC SIGNAL [b1, b2: UNSPECIFIED] = CODE;

printInterval: PUBLIC PROCEDURE [n, start, exp: ESPointer] =
  BEGIN OPEN DITypeDefs, DIActionDefs;
  so: DebuggerDefs.SymbolObject;
  sop: SOPointer ← @so;
  ss: StringDefs.SubStringDescriptor;
  h1: hereESPointer;
  h2: hereESPointer;
  IF ~TypeIU[n] THEN SIGNAL IncorrectType[n];
  IF ~TypeIU[start] THEN SIGNAL IncorrectType[start];
  IF ~(TypeArray[exp] OR TypeArrayDesc[exp] OR TypeString[exp])
```

```
    THEN SIGNAL IncorrectType[exp];
  h1 ← Transfer[start]; h2 ← Transfer[n];
  h1.value ← ActualValue[h1]; h2.value ← ActualValue[h2];
  IF ~h2.intN THEN h2.value ← h2.value - h1.value + 1;
  IF h2.value < 1
    THEN SIGNAL InvalidInterval[h1.value, h2.value];
  espTosop[exp,sop];
  IF TypeString[exp] THEN
    WITH exp SELECT FROM
      there =>
        BEGIN
        ss ←[LOOPHOLE[DebuggerDefs.GetValue[sop],STRING], h1.value, h2.value];
        DebugUtilityDefs.UserWriteSubString[@ss];
        END;
      here =>
        BEGIN
        ss ← [LOOPHOLE[value,StringDefs.SubString].base, h1.value, h2.value];
        DebuggerDefs.WriteSubString[@ss];
        END;
      ENDCASE => ERROR
  ELSE DebugInterpretDefs.IarrayPtr[sop, h1.value, h2.value];
  --note to stop after interval
  pushevalstack[setIntervalBit[h2]];
  FreeStackItem[h1]; FreeStackItem[exp];
  RETURN
  END;

TypeStackList: TIPointer ← NIL;

AllocateTypeItem: PROCEDURE RETURNS [tip: TIPointer] =
  BEGIN OPEN DIDefs;
  tip ← SystemDefs.AllocateHeapNode[SIZE[TypeItem]];
  tip↑ ← TypeItem[next: TypeStackList, stbase: DIActionDefs.GetCurrentST[],
     tsei: SymDefs.SENull, indirection: 0];
  TypeStackList ← tip;
  RETURN
  END;

FreeTypeItem: PROCEDURE [tip: TIPointer] =
  BEGIN
  dl: TIPointer ← TypeStackList;
  pdl: TIPointer ← NIL;
  UNTIL dl = NIL DO
    IF dl = tip THEN
      BEGIN
      IF pdl = NIL THEN TypeStackList ← dl.next ELSE pdl.next ← dl.next;
      SystemDefs.FreeHeapNode[tip];
      RETURN
      END;
    pdl ← dl; dl ← dl.next;
    ENDLOOP;
  RETURN
  END;

ResetTypeStack: PUBLIC PROCEDURE =
  BEGIN
  tip: TIPointer ← TypeStackList;
  ntip: TIPointer;
  UNTIL tip = NIL DO
    ntip ← tip.next;
    SystemDefs.FreeHeapNode[tip];
    tip ← ntip;
    ENDLOOP;
  TypeStackList ← NIL; ttop ← 0;
  RETURN
  END;

END..
```