```
-- Interpret.Mesa
-- Edited by:
--          Sandman, December 20, 1977  1:02 PM
--          Barbara, July 31, 1978  3:14 PM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [Address],
  ControlDefs: FROM "controldefs" USING [
    GlobalFrameHandle, Lreg, MaxParmsInStack, StateVector],
  CoreSwapDefs: FROM "coreswapdefs" USING [SVPointer],
  DebugBreakptDefs: FROM "debugbreakptdefs" USING [EntryToBTI],
  DebugData: FROM "debugdata" USING [StatePtr],
  DebuggerDefs: FROM "debuggerdefs" USING [
    addbitaddrs, Display, DumpValsFromState, FieldContext, FormatRecord,
    fullbitaddress, fullsymaddress, GetValue, InitSOP, Lookup, LookupProc,
    SA, SearchForModuleSym, SOPointer, SymbolObject],
  DebugInterpretDefs: FROM "debuginterpretdefs",
  DebugMiscDefs: FROM "debugmiscdefs" USING [
    DebugAbort, DFreeString, DGetString, DReadNumber, LookupFail,
    StringExpressionToNumber, WriteEOL],
  DebugSymbolDefs: FROM "debugsymboldefs" USING [
    DAcquireSymbolTable, DReleaseSymbolTable, SymbolsForGFrame],
  DebugUtilityDefs: FROM "debugutilitydefs" USING [
    MakeProcedureDescriptor, MREAD, UserCall, UserWriteSubString],
  IODefs: FROM "iodefs" USING [
    CR, LineOverflow, NUL, NumberFormat, ReadChar, ReadEditedString, ReadLine,
    SP, WriteChar, WriteDecimal, WriteLine, WriteNumber, WriteOctal,
    WriteString],
  StringDefs: FROM "stringdefs" USING [
    AppendString, AppendSubString, DeleteSubString, InvalidNumber,
    SubStringDescriptor],
  SymbolTableDefs: FROM "symboltabledefs" USING [SymbolTableBase],
  SymDefs: FROM "symdefs" USING [
    CSEIndex, CTXIndex, ISEIndex, ISENull, SEIndex, SENull];

DEFINITIONS FROM DebuggerDefs;

Interpret: PROGRAM
IMPORTS DebugBreakptDefs, DDptr: DebugData, DebuggerDefs, DebugMiscDefs,
  DebugSymbolDefs, DebugUtilityDefs, IODefs, StringDefs
EXPORTS DebugInterpretDefs, DebugMiscDefs =

BEGIN

Icall: PUBLIC PROCEDURE [proc: STRING] =
  BEGIN OPEN DebugSymbolDefs;
  state: ControlDefs.StateVector;
  symbase: SymbolTableDefs.SymbolTableBase;
  gframe: ControlDefs.GlobalFrameHandle;
  found: BOOLEAN;
  e: CARDINAL;
  psv: CoreSwapDefs.SVPointer;
  sei: SymDefs.ISEIndex;
  FR: FormatRecord ← FormatRecord[2, '=, IODefs.CR, IODefs.NUL, IODefs.NUL, TRUE, TRUE];

  BEGIN OPEN symbase;
  state.instbyte ← state.stkptr ← 0;
  [found, gframe, e] ← LookupProc[proc];
  IF ~found THEN ERROR DebugMiscDefs.LookupFail[proc];
  symbase ← DAcquireSymbolTable[DebugSymbolDefs.SymbolsForGFrame[gframe]];
  sei ← WITH b:(bb+DebugBreakptDefs.EntryToBTI[symbase, e]) SELECT FROM
    Callable => b.id,
    ENDCASE => SymDefs.ISENull;
  state.dest ← DebugUtilityDefs.MakeProcedureDescriptor[e, gframe];
  state.source ← REGISTER[ControlDefs.Lreg];
  getparams[sei, @state, symbase
        !UNWIND => DReleaseSymbolTable[symbase]];
  DebugMiscDefs.WriteEOL[];
  psv ← DebugUtilityDefs.UserCall[@state];
  DumpValsFromState[sei, @FR, symbase, psv];
  DReleaseSymbolTable[symbase];
  END;
  RETURN
  END;

getparams: PROCEDURE [bsei: SymDefs.ISEIndex, s: CoreSwapDefs.SVPointer, symbase: SymbolTableDefs.Symbo
```

```
**!TableBase] =
  BEGIN
  i, nwords: CARDINAL;
  typein, typeout: SymDefs.SEIndex;
  termchar: CHARACTER;

  BEGIN OPEN symbase;
  [typein, typeout] ← TransferTypes[(seb+bsei).idtype];
  IF typein = SymDefs.SENull THEN RETURN;
  nwords ← WordsForType[typein];
  IF nwords > ControlDefs.MaxParmsInStack THEN
    BEGIN
    IODefs.WriteString[ "Too many parameters"L];
    SIGNAL DebugMiscDefs.DebugAbort
    END;
  s.stkptr ← nwords;
  i ← 0;
  UNTIL i >= nwords DO
    BEGIN OPEN IODefs;
    WriteChar[' ]; WriteOctal[i]; WriteChar[':]; WriteChar[' ];
    parmstr ← DebugMiscDefs.DGetString[40];
    termchar ← ReadEditedString[parmstr, paramfound, TRUE
      !LineOverflow =>
         BEGIN WriteString[" String too long!"L]; GOTO badparam END];
    SELECT parmstr[0] FROM
      '" =>
        BEGIN
        DebugMiscDefs.DFreeString[parmstr];
        WriteString[" -- STRING parameters not supported in the debugger!"L];
        SIGNAL DebugMiscDefs.DebugAbort
        END;
      ENDCASE =>
        BEGIN
        s.stk[i] ← DebugMiscDefs.StringExpressionToNumber[parmstr,10
                !StringDefs.InvalidNumber =>
             BEGIN WriteChar['?]; GOTO badparam END];
        END;
    i ← i+1;
    EXITS
      badparam => DebugMiscDefs.DFreeString[parmstr];
    END;
    ENDLOOP;
  END;
  RETURN
  END;

parmstr: STRING;

paramfound: PROCEDURE [c: CHARACTER] RETURNS [BOOLEAN] =
  BEGIN
  SELECT c FROM
    IODefs.SP, IODefs.CR =>
      IF parmstr[0] # '" OR parmstr[parmstr.length-1] = '" THEN RETURN [TRUE];
    ENDCASE;
  RETURN [FALSE]
  END;

outbase: TYPE = {octal, decimal};

DisplayIVal: PROCEDURE [v: UNSPECIFIED, b: outbase] =
  BEGIN OPEN IODefs;
  WriteString[" -- "L];
  SELECT b FROM
    octal => WriteOctal[v];
    decimal => WriteDecimal[v];
    ENDCASE;
  RETURN
  END;

Iaddress: PUBLIC PROCEDURE [var: STRING] =
  BEGIN
  so: SymbolObject;
  sop: SOPointer ← @so;
  sym: DebuggerDefs.fullbitaddress;
  BEGIN OPEN sop.stbase;
  InitSOP[sop];
```

```
  IF ~Lookup[var, FALSE, sop, FALSE, mod] THEN
    SIGNAL DebugMiscDefs.LookupFail[var];
  IF (seb+sop.sei).constant THEN
    BEGIN
    IODefs.WriteString[" not addressable"L];
    SIGNAL DebugMiscDefs.DebugAbort
    END;
  sym ← addbitaddrs[fullsymaddress[sop], sop.baddr];
  WITH sym SELECT FROM
    short => DisplayIVal[shortAddr, octal];
    ENDCASE;
  END;
  RETURN
  END;

modulestr: STRING;
typestr: STRING;
VariantType: SIGNAL = CODE;

Ipointer: PUBLIC PROCEDURE [a: AltoDefs.Address, type: STRING] =
  BEGIN
  so: SymbolObject;
  sop: SOPointer ← @so;
  FR: FormatRecord ← FormatRecord[2, '=, IODefs.CR, IODefs.NUL, IODefs.NUL, TRUE, TRUE];

  BEGIN OPEN DebugMiscDefs, sop.stbase;
  InitSOP[sop];
  modulestr ← DGetString[40];
  typestr ← DGetString[40];
  StringDefs.AppendString[to: typestr, from: type];
  parsetype[typestr, modulestr    !VariantType =>
    BEGIN
    IODefs.WriteString["VariantTypes not yet implemented!"L];
    DFreeString[typestr]; DFreeString[modulestr];
    SIGNAL DebugAbort
    END];
  IF modulestr.length = 0 THEN
    BEGIN
    IF ~Lookup[typestr, FALSE, sop, TRUE, mod] THEN SIGNAL LookupFail[typestr]
    END
  ELSE
    IF ~SearchForModuleSym[modulestr, typestr, FALSE, sop, TRUE]
      THEN SIGNAL LookupFail[typestr];
  sop.baddr.wd ← short[shortAddr:[a]];
  sop.tsei ← sop.sei;
  sop.sei ← SymDefs.ISENull;
  WriteEOL[];
  Display[sop, @FR, FALSE];
  DFreeString[modulestr];
  DFreeString[typestr];
  RETURN
  END;
  END;

parsetype: PROCEDURE [type: STRING, module: STRING]=
  BEGIN
  i: CARDINAL;
  ss: StringDefs.SubStringDescriptor;
  FOR i IN [0..type.length) DO
    IF type[i] = '. THEN
      BEGIN
      ss ← [type, 0, i+1];
      StringDefs.AppendSubString[module, @ss];
      StringDefs.AppendString[module, "bcd"L];
      StringDefs.DeleteSubString[@ss];
      END;
    IF type[i] = IODefs.SP AND i # (type.length-1)
      THEN SIGNAL VariantType;
    ENDLOOP;
  RETURN
  END;

Iarray: PUBLIC PROCEDURE [a: STRING, i, n: CARDINAL] =
  BEGIN
  so: SymbolObject;
  sop: SOPointer ← @so;
```

```
  InitSOP[sop];
  IF ~DebuggerDefs.Lookup[a, FALSE, sop, FALSE, mod]
    THEN SIGNAL DebugMiscDefs.LookupFail[a];
  IarrayPtr[sop, i, n];
  END;

IarrayPtr: PUBLIC PROCEDURE [sop: SOPointer, i, n: CARDINAL] =
  BEGIN OPEN DebuggerDefs;
  csei: SymDefs.SEIndex;
  isei: SymDefs.SEIndex;
  rsei: SymDefs.CSEIndex;
  FR: FormatRecord ← FormatRecord[2, IODefs.NUL, IODefs.SP, IODefs.NUL, IODefs.NUL, FALSE, FALSE];
  j: CARDINAL;
  csize: CARDINAL;
  packed:  BOOLEAN;
  sym: fullbitaddress ← fullsymaddress[sop];
  sa: SA;
  WITH sym SELECT FROM
    short => sa ← shortAddr;
    ENDCASE => ERROR;
  BEGIN OPEN DebugMiscDefs, sop.stbase;
  SELECT TypeForm[sop.tsei] FROM
    array => WITH s:sop.baddr SELECT FROM
      short => s.shortAddr ← [s.shortAddr + sa];
      long => s.longAddr.li ←  s.longAddr.li + sa;
      ENDCASE;
    arraydesc => sop.baddr.wd ← short[GetValue[sop]];
    ENDCASE =>
      BEGIN IODefs.WriteString[" not an ARRAY"L]; SIGNAL DebugAbort END;
  rsei ← UnderType[sop.tsei];
  WITH a: (seb + rsei) SELECT FROM
    array =>
      BEGIN
      packed ← a.packed;
      csize ← WordsForType[UnderType[csei ← a.componenttype]];
      isei ← a.indextype;
      END;
    arraydesc =>
      WITH aa: (seb+UnderType[a.describedType]) SELECT FROM
        array =>
          BEGIN
          packed ← aa.packed;
          csize ← WordsForType[UnderType[csei ← aa.componenttype]];
          isei ← aa.indextype;
          END;
        ENDCASE;
    ENDCASE;
  WITH (seb + UnderType[isei]) SELECT FROM
    subrange => i ← i - CARDINAL[origin];
    ENDCASE;
  IF packed THEN
    BEGIN
    sop.space ← 8;
    WITH s:sop.baddr SELECT FROM
      short => s.shortAddr ← [s.shortAddr + i/2];
      long => s.longAddr.li ← s.longAddr.li + i/2;
      ENDCASE;
    sop.baddr.bd ← IF i MOD 2 = 0 THEN 0 ELSE 8;
    END
  ELSE WITH s:sop.baddr SELECT FROM
      short => s.shortAddr ← [s.shortAddr + i*csize];
      long => s.longAddr.li ← s.longAddr.li + i*csize;
      ENDCASE;
  sop.tsei ← csei;
  sop.sei ← SymDefs.ISENull;
  FOR j IN [0..n) DO
    Display[sop, @FR, FALSE];
    IF ~packed THEN WITH s:sop.baddr SELECT FROM
      short => s.shortAddr ← [s.shortAddr + csize];
      long => s.longAddr.li ← s.longAddr.li + csize;
      ENDCASE
    ELSE sop.baddr ← addbitaddrs[sop.baddr, fullbitaddress[wd:short[[0]],bd:8]];
    IF j # n-1 THEN IODefs.WriteString[", "L];
    ENDLOOP;
  RETURN
  END;
```

```
  END;

Istring: PUBLIC PROCEDURE [s: STRING, i, n: CARDINAL] =
  BEGIN
  so: SymbolObject;
  sop: SOPointer ← @so;
  ss:  StringDefs.SubStringDescriptor;
  StringCTXIndex: SymDefs.CTXIndex ← LOOPHOLE[6];

  BEGIN OPEN DebugMiscDefs, sop.stbase;
  InitSOP[sop];
  IF ~Lookup[s, FALSE, sop, FALSE, mod] THEN SIGNAL LookupFail[s];
  SELECT TypeForm[sop.tsei] FROM
    pointer => sop.baddr.wd ← short[GetValue[sop]];
    ENDCASE => GOTO notastring;
  WITH (seb + UnderType[sop.tsei]) SELECT FROM
    pointer => IF TypeForm[pointedtotype] # record OR
      FieldContext[sop.stbase, pointedtotype] # StringCTXIndex
      THEN GOTO notastring;
    ENDCASE => GOTO notastring;
  WITH sop.baddr SELECT FROM
    short => ss ← [LOOPHOLE[shortAddr, STRING], i, n];
    ENDCASE => ERROR;
  DebugUtilityDefs.UserWriteSubString[@ss];
  RETURN
  EXITS
    notastring =>
      BEGIN IODefs.WriteString[" not a STRING"L]; SIGNAL DebugAbort END;
  END;
  END;

Ideref: PUBLIC PROCEDURE [var: STRING] =
  BEGIN
  so: SymbolObject;
  sop: SOPointer ← @so;
  tsei: SymDefs.CSEIndex;
  FR: FormatRecord ← FormatRecord[2, '=, IODefs.CR, IODefs.NUL, IODefs.NUL, TRUE, TRUE];

  BEGIN OPEN DebugMiscDefs, sop.stbase;
  InitSOP[sop];
  IF ~Lookup[var, FALSE, sop, FALSE, mod] THEN SIGNAL LookupFail[var];
  sop.baddr.wd ← short[GetValue[sop]];
  tsei ← UnderType[sop.tsei];
  WITH (seb + tsei) SELECT FROM
    pointer => sop.tsei ← pointedtotype;
    ENDCASE =>
      BEGIN IODefs.WriteString[" not type POINTER"L]; SIGNAL DebugAbort END;
  sop.sei ← SymDefs.ISENull;
  WriteEOL[];
  Display[sop, @FR, FALSE];
  RETURN
  END;
  END;

lastexpression: INTEGER;

Iexpression: PUBLIC PROCEDURE =
  BEGIN OPEN IODefs;
  v: INTEGER = DebugMiscDefs.DReadNumber[lastexpression,10];
  WriteString[" = "L];
  WriteNumber[v,NumberFormat[10,FALSE,FALSE,1]]; WriteChar['D];
  WriteString[" = "L];
  WriteNumber[v,NumberFormat[8,FALSE,TRUE,1]]; WriteChar['B];
  IF v < 0 THEN
    BEGIN
    WriteString[" = "L];
    WriteNumber[v,NumberFormat[10,FALSE,TRUE,1]];
    WriteChar['D];
    END;
  lastexpression ← v
  END;

--miscellaneous commands

IgnoreComment: PUBLIC PROCEDURE =
  BEGIN OPEN IODefs;
```

```
  s: STRING ← [100];
  IF ReadChar[] # '- THEN BEGIN WriteLine[" ?"L]; RETURN; END
  ELSE WriteChar['-];
  ReadLine[s];
  RETURN
  END;

DisplayEvalStack: PUBLIC PROCEDURE =
  BEGIN OPEN IODefs;
  i: CARDINAL;
  state: ControlDefs.StateVector;
  Dstate: DESCRIPTOR FOR ARRAY OF UNSPECIFIED ← DESCRIPTOR[@state, SIZE[ControlDefs.StateVector]];

  IF DDptr.StatePtr = NIL THEN
    BEGIN WriteString["empty!"L]; RETURN END;
  FOR i IN [0..LENGTH[Dstate]) DO
    Dstate[i] ← DebugUtilityDefs.MREAD[DDptr.StatePtr+i]
    ENDLOOP;
  IF state.stkptr = 0 THEN
    BEGIN WriteString["empty!"L]; RETURN END;
  FOR i IN [0..state.stkptr)
    DO WriteChar[' ]; WriteOctal[state.stk[i]] ENDLOOP;
  RETURN
  END;

END...
```