```
-- BreakPt.Mesa
-- Edited by:
--           Barbara on August 1, 1978  3:31 PM
--           Johnsson on August 2, 1978  10:51 AM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [BYTE, PageSize, wordlength],
  ControlDefs: FROM "controldefs" USING [
    BytePC, FrameHandle, GlobalFrameHandle, NullFrame, WordPC],
  CoreSwapDefs: FROM "coreswapdefs" USING [
    BBHandle, SVPointer, UBBPointer, UserBreakBlock],
  DebugBreakptDefs: FROM "debugbreakptdefs" USING [
    BBPointer, BodyEntryPc, BodyExitPc, BreakBlock, BreakError, BTtype,
    CodeObject, EXOItype, GCtype, PrintLocation, SCtype, StringToFGTEntry,
    WriteOctalHerald],
  DebuggerDefs: FROM "debuggerdefs" USING [
    FrameRelBPC, fullbitaddress, fullsymaddress, LookupProc, LookupProg,
    MainBTI, PcToCBTI, SeiHandle, SA, SOPointer, SymbolObject, WriteSource,
    WriteTransferName],
  DebugMiscDefs: FROM "debugmiscdefs" USING [
    CopyRead, CopyWrite, InterpretString, LookupFail,
    StringExpressionToNumber, WriteEOL],
  DebugSymbolDefs: FROM "debugsymboldefs" USING [
    DAcquireSymbolTable, DReleaseSymbolTable, SymbolsForGFrame],
  DebugUtilityDefs: FROM "debugutilitydefs" USING [
    FreeOnDrum, GFtoCode, MREAD, ReadCodeByte, SREAD, SWRITE,
    ValidGlobalFrame, WriteCodeByte],
  DIActionDefs: FROM "diactiondefs" USING [CleanUp, espTosop],
  DIDefs: FROM "didefs" USING [ESPointer],
  FSPDefs: FROM "fspdefs" USING [
    AddToNewZone, FreeNode, MakeNewZone, MakeNode, NoRoomInZone, PruneZone,
    ZonePointer],
  IODefs: FROM "iodefs" USING [SP, WriteString],
  Mopcodes: FROM "mopcodes" USING [zBRK, zNOOP, zPORTI, zPUSHX, zRET],
  SDDefs: FROM "sddefs" USING [sBreakBlock, sBreakBlockSize, SD],
  StringDefs: FROM "stringdefs" USING [
    AppendString, AppendSubString, DeleteSubString, SubString,
    SubStringDescriptor],
  SymbolTableDefs: FROM "symboltabledefs" USING [
    NoSymbolTable, SymbolTableBase],
  SymDefs: FROM "symdefs" USING [BTIndex, BTNull, FGTEntry],
  SystemDefs: FROM "systemdefs" USING [
    AllocateHeapNode, AllocateHeapString, AllocateResidentPages, FreeHeapNode,
    FreeHeapString, FreePages];

BreakPt: PROGRAM
  IMPORTS DebugBreakptDefs, DebuggerDefs, DebugMiscDefs, DebugSymbolDefs,
    DebugUtilityDefs, DIActionDefs, FSPDefs, IODefs, StringDefs,
    SymbolTableDefs, SystemDefs
  EXPORTS DebugBreakptDefs =

  BEGIN  -- a collection of procedures to set and clear breakpoints
  BytePC: TYPE = ControlDefs.BytePC;
  WordPC: TYPE = ControlDefs.WordPC;
  BYTE: TYPE = AltoDefs.BYTE;
  GlobalFrameHandle: TYPE = ControlDefs.GlobalFrameHandle;
  CodeObject: TYPE = DebugBreakptDefs.CodeObject;
  BBPointer: TYPE = DebugBreakptDefs.BBPointer;
  BreakError: TYPE = DebugBreakptDefs.BreakError;
  UBBPointer: TYPE = CoreSwapDefs.UBBPointer;
  UserBreakBlock: TYPE = CoreSwapDefs.UserBreakBlock;

  SetBlocks: BBPointer ← BBnil;    --list of set blocks
  NewBlock: BBPointer ← BBnil;    --most recent breakblock
  bpError: SIGNAL = CODE;

  BBnil: BBPointer = NIL;

  BreakPointError: PUBLIC SIGNAL [error: BreakError] = CODE;

  DeleteBreakBlock: PROCEDURE [optr: BBPointer] RETURNS [last: BOOLEAN] =
    -- Delete optr from the list designated by SetBlocks and
    -- linked through the bpChain field.
    -- RETURN TRUE iff this is the last break in this code segment
    BEGIN
    p, q: BBPointer;
```

```
    IF (p ← SetBlocks) = BBnil THEN SIGNAL bpError;
    IF optr = SetBlocks THEN SetBlocks ← optr.chain
    ELSE
      DO
      IF p.chain = optr THEN
        BEGIN p.chain ← optr.chain; EXIT END;
      IF (p ← p.chain) = BBnil THEN SIGNAL bpError;
      ENDLOOP;
    last ← FALSE;
    FOR q ← SetBlocks, q.chain UNTIL q = BBnil DO
      IF q.code = optr.code THEN EXIT;
      REPEAT
      FINISHED => last ← TRUE;
      ENDLOOP;
    FreeBB[optr];
    RETURN
    END;

  BBzone: FSPDefs.ZonePointer ←
    FSPDefs.MakeNewZone[SystemDefs.AllocateResidentPages[1],
      AltoDefs.PageSize, SystemDefs.FreePages];

  FreeBB: PROCEDURE [bb: BBPointer] =
    BEGIN OPEN FSPDefs;
    FreeNode[BBzone, bb];
    [] ← PruneZone[BBzone];
    RETURN
    END;

  AllocBB: PROCEDURE RETURNS [bb: BBPointer] =
    BEGIN OPEN FSPDefs;
    bb ← MakeNode[BBzone, SIZE[DebugBreakptDefs.BreakBlock]
      ! NoRoomInZone =>
        IF z = BBzone THEN
          BEGIN
          AddToNewZone[BBzone, SystemDefs.AllocateResidentPages[1],
            AltoDefs.PageSize, SystemDefs.FreePages];
          RESUME
          END];
    RETURN
    END;

  InsertBreak: PUBLIC PROCEDURE [frame: GlobalFrameHandle, pc: BytePC,
    e: DebugBreakptDefs.EXOItype, b: DebugBreakptDefs.BTtype,
    condition: STRING] =
    BEGIN   --set breakpoint on the specified statement
    sp, p: BBPointer;
    op: BYTE;

    NewBlock ← BBnil;
    --allowed to insert same break again in order to change the condition
    IF (p ← FindBPRec[frame, pc]) # BBnil THEN
      BEGIN
      IF p.condition # NIL THEN
        BEGIN SystemDefs.FreeHeapString[p.condition]; p.condition ← NIL; END;
      IF condition # NIL AND condition.length # 0 THEN
        SetUpCondition[p, condition];
      RETURN
      END;
    op ← DebugUtilityDefs.ReadCodeByte[frame,pc];
    IF op = Mopcodes.zPORTI THEN ERROR BreakPointError[notAllowed];
    IF op = Mopcodes.zPUSHX OR op = Mopcodes.zNOOP THEN pc ← [pc + 1];
    IF e = exit AND op # Mopcodes.zRET AND op # Mopcodes.zBRK THEN
      ERROR BreakPointError[noReturn];
    NewBlock ← p ← AllocBB[];
    --set break point parameters
    p↑ ← DebugBreakptDefs.BreakBlock[faddr: frame, pc: pc, brkinst: ,
      chain: , code: , exo: e, bt: b, condition: NIL];
    IF (sp ← FindSegBPRec[frame, pc]) # BBnil THEN
      p.brkinst ← sp.brkinst
    ELSE
      BEGIN
      p.brkinst ← DebugUtilityDefs.ReadCodeByte[frame,pc];
      DebugUtilityDefs.WriteCodeByte[frame, pc, Mopcodes.zBRK];
      END;
```

```
    IF condition # NIL AND condition.length # 0 THEN
      SetUpCondition[p, condition];
    p.code ← DebugUtilityDefs.GFtoCode[p.faddr];
    --insert in breakblock in frame order
    IF SetBlocks # BBnil THEN
      BEGIN sp ← SetBlocks;
      DO
        IF sp.faddr = p.faddr OR sp.chain = BBnil THEN EXIT;
        sp ← sp.chain;
        ENDLOOP;
      p.chain ← sp.chain;
      sp.chain ← p;
      END
    ELSE BEGIN p.chain ← SetBlocks; SetBlocks ← p; END;
    RETURN
    END;

  OctalBreakPoint: PUBLIC PROCEDURE [frame: GlobalFrameHandle, pc: BytePC,
    sc: DebugBreakptDefs.SCtype] =
    BEGIN
    ENABLE BreakPointError => BEGIN WriteBreakError[error]; CONTINUE; END;
    IF sc = set THEN InsertBreak[frame, pc, octal, break, NIL]
    ELSE RemoveBreak[frame, pc];
    RETURN
    END;

  RemoveBreak: PUBLIC PROCEDURE [frame: GlobalFrameHandle, pc: BytePC] =
    BEGIN   --remove breakpoint on the specified statement
    lastone: BOOLEAN;
    p: BBPointer;
    op: BYTE ← DebugUtilityDefs.ReadCodeByte[frame,pc];
    IF op = Mopcodes.zPUSHX OR op = Mopcodes.zNOOP THEN pc ← [pc + 1];
    IF (p ← FindBPRec[frame,pc]) = BBnil THEN ERROR BreakPointError[notFound];
    op ← p.brkinst;
    IF p.condition # NIL THEN
      BEGIN
      SystemDefs.FreeHeapString[p.condition];
      FreeUserBreakBlock[p.faddr, ByteToWordPC[p.pc]];
      END;
    lastone ← DeleteBreakBlock[p];
    IF lastone OR FindSegBPRec[frame,pc] = BBnil THEN
      BEGIN
      DebugUtilityDefs.WriteCodeByte[frame,pc,op];
      IF lastone THEN DebugUtilityDefs.FreeOnDrum[frame];
      END;
    RETURN
    END;

  FindBPRec: PUBLIC PROCEDURE [frame: GlobalFrameHandle, pc: BytePC]
    RETURNS [p: BBPointer] =
    BEGIN
    IF (p ← SetBlocks) = BBnil THEN RETURN;
    DO
      IF p.pc = pc AND p.faddr = frame THEN RETURN;
      IF (p ← p.chain) = BBnil THEN RETURN;
      ENDLOOP;
    RETURN
    END;

  FindSegBPRec: PUBLIC PROCEDURE [frame: GlobalFrameHandle, pc: BytePC]
    RETURNS [p: BBPointer] =
    BEGIN
    code: CodeObject ← DebugUtilityDefs.GFtoCode[frame];

    IF (p ← SetBlocks) = BBnil THEN RETURN;
    DO
      IF p.pc = pc AND p.code = code THEN RETURN;
      IF (p ← p.chain) = BBnil THEN RETURN;
      ENDLOOP;
    RETURN
    END;

  DeleteBreaks: PUBLIC PROCEDURE =
    BEGIN
    p: BBPointer;
    IF (p ← SetBlocks) = BBnil THEN RETURN;
```

```
    DO
      IF p.exo # octal
        THEN DebugUtilityDefs.WriteCodeByte[p.faddr, p.pc, p.brkinst];
      IF (p ← p.chain) = BBnil THEN EXIT;
      ENDLOOP;
    RETURN
    END;

  RestoreBreaks: PUBLIC PROCEDURE =
    BEGIN
    p: BBPointer;
    IF (p ← SetBlocks) = BBnil THEN RETURN;
    DO
      IF p.exo # octal
        THEN DebugUtilityDefs.WriteCodeByte[p.faddr, p.pc, Mopcodes.zBRK];
      IF (p ← p.chain) = BBnil THEN EXIT;
      ENDLOOP;
    RETURN
    END;

  CodeByteFromState: PROCEDURE [state: CoreSwapDefs.SVPointer]
    RETURNS [BYTE] =
    BEGIN OPEN DebugUtilityDefs;
    frame: ControlDefs.FrameHandle = MREAD[@state.dest];
    RETURN[ReadCodeByte[MREAD[@frame.accesslink],
      DebuggerDefs.FrameRelBPC[frame]]]
    END;

  ValidateBreakpoint: PUBLIC PROCEDURE [state: CoreSwapDefs.SVPointer]
    RETURNS [BOOLEAN] =
    BEGIN
    RETURN[state # NIL AND CodeByteFromState[state] = Mopcodes.zBRK]
    END;

  FGFrame: TYPE = RECORD [fge: SymDefs.FGTEntry, frame: GlobalFrameHandle];

  FindProcString: PROCEDURE [proc, text: STRING] RETURNS [FGFrame] =
    BEGIN -- convert [proc,text] to FGTEntry
    found: BOOLEAN;
    frame: GlobalFrameHandle;
    entryindex: CARDINAL;
    [found, frame, entryindex] ← DebuggerDefs.LookupProc[proc];
    IF ~found THEN ERROR DebugMiscDefs.LookupFail[proc];
    RETURN [FGFrame[DebugBreakptDefs.StringToFGTEntry[frame, entryindex, text], frame]]
    END;

  FindProgString: PROCEDURE [prog, text: STRING] RETURNS [FGFrame] =
    BEGIN -- convert [prog,text] to FGTEntry
    stbase: SymbolTableDefs.SymbolTableBase;
    found: BOOLEAN;
    gframe: GlobalFrameHandle;
    IF prog[0] ~IN ['0..'9] THEN
      BEGIN
      [found, gframe, stbase] ← DebuggerDefs.LookupProg[prog];
      IF ~found THEN ERROR DebugMiscDefs.LookupFail[prog];
      DebugSymbolDefs.DReleaseSymbolTable[stbase];
      END
    ELSE BEGIN
      gframe ← LOOPHOLE[DebugMiscDefs.StringExpressionToNumber[prog, 8], GlobalFrameHandle];
      IF ~DebugUtilityDefs.ValidGlobalFrame[gframe]
        THEN ERROR DebugMiscDefs.LookupFail[prog];
      END;
    RETURN[FGFrame[DebugBreakptDefs.StringToFGTEntry[gframe, 0, text],gframe]]
    END;

  TextBreakPoint: PUBLIC PROCEDURE [body, source, condition: STRING,
    bt: DebugBreakptDefs.BTtype, gc: DebugBreakptDefs.GCtype] =
    BEGIN
    fgf: FGFrame;
    fgf ← IF gc = prog THEN FindProgString[body, source]
      ELSE FindProcString[body, source];
    InsertBreak[fgf.frame, [fgf.fge.cindex], in, bt, condition !
      BreakPointError => BEGIN WriteBreakError[error]; GOTO exit; END];
    IF NewBlock # BBnil THEN
      DebugBreakptDefs.PrintLocation[fgf.frame, fgf.fge.findex, FALSE];
    RETURN
```

```
    EXITS exit => RETURN;
    END;

  ClearTextBreakPoint: PUBLIC PROCEDURE [body, source: STRING,
    gc: DebugBreakptDefs.GCtype] =
    BEGIN
    fgf: FGFrame;
    fgf ← IF gc = prog THEN FindProgString[body, source]
      ELSE FindProcString[body, source];
    RemoveBreak[fgf.frame, [fgf.fge.cindex] !
      BreakPointError => BEGIN WriteBreakError[error]; GOTO exit; END];
    IF NewBlock # BBnil THEN
      DebugBreakptDefs.PrintLocation[fgf.frame, fgf.fge.findex, FALSE];
    RETURN
    EXITS exit => RETURN;
    END;

  BreakPoint: PUBLIC PROCEDURE [body, condition: STRING,
    bt: DebugBreakptDefs.BTtype, sc: DebugBreakptDefs.SCtype,
    ex: DebugBreakptDefs.EXOItype] =
    BEGIN OPEN DebugBreakptDefs;
    noSymPc, pc: BytePC;
    frame: GlobalFrameHandle;
    found: BOOLEAN;
    entryindex: CARDINAL;
    [found, frame, entryindex] ← DebuggerDefs.LookupProc[body];
    IF ~found THEN SIGNAL DebugMiscDefs.LookupFail[body];
    pc ← [IF ex = entry THEN BodyEntryPc[frame, entryindex, FALSE]
      ELSE BodyExitPc[frame, entryindex]];
    IF sc = set THEN InsertBreak[frame, pc, ex, bt, condition !
      BreakPointError => BEGIN WriteBreakError[error]; GOTO exit; END]
    ELSE RemoveBreak[frame, pc !
      BreakPointError => IF error # notFound OR ex # entry
        THEN BEGIN WriteBreakError[error]; GOTO exit; END
      ELSE BEGIN
        noSymPc ← [BodyEntryPc[frame, entryindex, TRUE]];
        IF noSymPc = pc THEN BEGIN WriteBreakError[error]; GOTO exit; END
        ELSE RemoveBreak[frame, noSymPc  !BreakPointError =>
          BEGIN WriteBreakError[error]; GOTO exit; END];
        GOTO exit;
        END];
    RETURN
    EXITS
      exit => RETURN;
    END;

  TraceAll: PUBLIC PROCEDURE [body: STRING, sc: DebugBreakptDefs.SCtype, ex: DebugBreakptDefs.EXOItype]
** =
    BEGIN OPEN DebugSymbolDefs, DebugBreakptDefs, SymDefs;
    found: BOOLEAN;
    gframe: GlobalFrameHandle;
    stbase: SymbolTableDefs.SymbolTableBase;
    bti: BTIndex;
    s: STRING ← [30];
    desc: StringDefs.SubStringDescriptor;
    ss: StringDefs.SubString ← @desc;
    noSymPc, pc: BytePC;

    IF body[0] ~IN ['0..'9] THEN
      BEGIN
      [found, gframe, stbase] ← DebuggerDefs.LookupProg[body];
      IF ~found THEN ERROR DebugMiscDefs.LookupFail[body];
      END
    ELSE BEGIN
      gframe ← LOOPHOLE[DebugMiscDefs.StringExpressionToNumber[body, 8], GlobalFrameHandle];
      IF ~DebugUtilityDefs.ValidGlobalFrame[gframe] THEN GOTO notfound;
      stbase ← DAcquireSymbolTable[SymbolsForGFrame[gframe
         !SymbolTableDefs.NoSymbolTable => GOTO notfound]
           !SymbolTableDefs.NoSymbolTable => GOTO notfound];
      EXITS
        notfound => ERROR DebugMiscDefs.LookupFail[body];
      END;
    BEGIN OPEN stbase;
    IF (bti ← (bb+DebuggerDefs.MainBTI).firstSon) = BTNull THEN RETURN;
    DO
      WITH b:(bb+bti) SELECT FROM
```

```
      Callable =>
        BEGIN
        SubStringForHash[ss, (b.id+seb).htptr];
        s.length ← 0;
        StringDefs.AppendSubString[s, ss];
        pc ← [IF ex = entry THEN BodyEntryPc[gframe, b.entryIndex, FALSE]
        ELSE BodyExitPc[gframe, b.entryIndex]];
        SELECT sc FROM
          set => InsertBreak[gframe, pc, ex, trace, NIL !
            BreakPointError => BEGIN WriteBreakError[error]; GOTO exit;END];
          clear => RemoveBreak[gframe, pc !
            BreakPointError => IF error # notFound OR ex # entry
              THEN BEGIN WriteBreakError[error]; GOTO exit; END
            ELSE BEGIN
              noSymPc ← [BodyEntryPc[gframe, b.entryIndex, TRUE]];
              IF noSymPc = pc
                THEN BEGIN WriteBreakError[error]; GOTO exit; END
              ELSE RemoveBreak[gframe, noSymPc !BreakPointError =>
                BEGIN WriteBreakError[error]; GOTO exit; END];
              GOTO exit;
              END];
          ENDCASE;
        EXITS
          exit => NULL;
        END;
      ENDCASE;
    IF (bb+bti).link.which = parent THEN EXIT
    ELSE bti ← (bb+bti).link.index;
    ENDLOOP;
  DReleaseSymbolTable[stbase];
  END;
  RETURN
  END;

ClearAllBT: PUBLIC PROCEDURE [bt: DebugBreakptDefs.BTtype] =
  BEGIN   --clear all trace/break points
  p, q: BBPointer;
  IF (p ← SetBlocks) = BBnil THEN RETURN;
  DO
    q ← p.chain;
    IF p.bt = bt THEN RemoveBreak[p.faddr, p.pc !
      BreakPointError => BEGIN WriteBreakError[error]; CONTINUE; END];
    IF (p ← q) = BBnil THEN EXIT;
    ENDLOOP;
  RETURN
  END;

ClearBreakBlocks: PUBLIC PROCEDURE =
  BEGIN   -- initialize trace/break points
  p, q: BBPointer;
  FOR p ← SetBlocks, q UNTIL p = BBnil DO
    q ← p.chain;
    FreeBB[p];
    ENDLOOP;
  SetBlocks ← BBnil;
  RETURN
  END;

ListAll: PUBLIC PROCEDURE [bt: DebugBreakptDefs.BTtype]=
  BEGIN
  found: BOOLEAN ← FALSE;
  p: BBPointer ← SetBlocks;
  UNTIL p = BBnil DO
    IF p.bt = bt THEN
      BEGIN found ← TRUE; PrintTextLine[p]; END;
    p ← p.chain;
    ENDLOOP;
  IF ~found THEN SELECT bt FROM
    break => WriteBreakError[noBreaksSet];
    trace => WriteBreakError[noTracesSet];
    ENDCASE;
  RETURN
  END;

PrintTextLine: PROCEDURE [p: BBPointer]=
  BEGIN OPEN DebuggerDefs, DebugSymbolDefs;
```

```
    stbase: SymbolTableDefs.SymbolTableBase;
    IF p.exo = octal THEN DebugBreakptDefs.WriteOctalHerald[p.faddr, p.pc]
    ELSE BEGIN
      IODefs.WriteString[SELECT p.exo FROM
        entry => "Entry to "L,
        exit =>  "Exit from "L,
        ENDCASE => "In "L];
      stbase ← DAcquireSymbolTable[SymbolsForGFrame[p.faddr
         !SymbolTableDefs.NoSymbolTable => GOTO nosym]
           !SymbolTableDefs.NoSymbolTable => GOTO nosym];
      WriteTransferName[SeiHandle[stbase: stbase,
        sei: (stbase.bb+PcToCBTI[stbase, p.pc]).id], TRUE,
        ControlDefs.NullFrame, p.faddr];
      IF p.condition # NIL THEN
        BEGIN
        IODefs.WriteString[" Condition: "L];
        IODefs.WriteString[p.condition];
        END;
      IF p.exo = in THEN WriteSource[p.faddr, p.pc, FALSE];
      DReleaseSymbolTable[stbase];
      EXITS
        nosym => WriteBreakError[noSym];
      END;
    IF p.exo # in THEN DebugMiscDefs.WriteEOL[];
    RETURN
    END;

  InvalidCondition: SIGNAL = CODE;

  SetUpCondition: PROCEDURE [bb: BBPointer, condition: STRING] =
    BEGIN
    userBB: UBBPointer;
    BEGIN
    userBB ← SystemDefs.AllocateHeapNode[SIZE[UserBreakBlock]];
    userBB↑ ← [frame: bb.faddr, pc: ByteToWordPC[bb.pc], ptrL: NIL, ptrR: NIL,
      posnL: 0, posnR: 0, sizeL: 16, sizeR: 16, inst:bb.brkinst, relation:,
      immediateR: FALSE, counterL: FALSE, localL: FALSE, localR: FALSE];
    ParseConditional[condition, userBB
      ! InvalidCondition => GOTO invalid];
    bb.condition ← SystemDefs.AllocateHeapString[condition.length];
    StringDefs.AppendString[bb.condition, condition];
    WriteUserBreakBlock[userBB];
    EXITS
      invalid => WriteBreakError[badCondition];
    END;
    SystemDefs.FreeHeapNode[userBB];
    RETURN
    END;

  ParseConditional: PROCEDURE [c: STRING, p: UBBPointer] =
    BEGIN
    i: CARDINAL;
    localcopy: STRING ← [100];
    var: STRING ← [40];
    CopyString[from: c, to: localcopy];
    ParseForID[localcopy, var];
    FOR i IN [0..var.length-1) DO
      IF var[i] ~IN['0..'9] THEN EXIT;
      REPEAT
         FINISHED =>
           BEGIN
           SetupForCounter[var, p];
           FOR i IN [0..localcopy.length) DO
             IF localcopy[i] # IODefs.SP THEN SIGNAL InvalidCondition;
             ENDLOOP;
           RETURN;
           END;
      ENDLOOP;
    SetupForID[var, TRUE, p];
    SetupForRelation[localcopy, p];
    var.length ← 0;
    ParseForID[localcopy, var];
    IF var[0] ~IN ['0..'9] THEN SetupForID[var, FALSE, p]
    ELSE SetupForLiteral[var, p];
    RETURN
    END;
```

```
  ParseForID: PROCEDURE [local, var: STRING] =
    BEGIN OPEN StringDefs;
    i, count: CARDINAL ← 0;
    ssd: SubStringDescriptor ← [base: local, offset: 0, length: 0];
    ss: SubString ← @ssd;
    IF local[0] = IODefs.SP THEN DropLeadingSpaces[local];
    FOR i IN [0..local.length) DO
      count ← i;
      SELECT local[i] FROM
        IN ['<..'>], =IODefs.SP, ='# => GOTO done;
        ENDCASE => NULL;
      REPEAT
        done => ssd.length ← count;
        FINISHED => ssd.length ← count+1;
      ENDLOOP;
    AppendSubString[var, ss];
    DeleteSubString[ss];
    RETURN
    END;

  SetupForID: PROCEDURE [var: STRING, left: BOOLEAN, p: UBBPointer] =
    BEGIN
    SetUpBreakBlock: PROCEDURE [esp: DIDefs.ESPointer] =
      BEGIN OPEN DebuggerDefs;
      sym: fullbitaddress;
      sopWd, sa: SA;
      so: SymbolObject;
      sop: SOPointer ← @so;
      DIActionDefs.espTosop[esp, sop];
      WITH b:sop.baddr SELECT FROM
        short => sopWd ← b.shortAddr;
        ENDCASE => SIGNAL InvalidCondition;
      sym ← fullsymaddress[sop];
      WITH sym SELECT FROM
        short => sa ← shortAddr;
        ENDCASE => ERROR;
      IF left THEN
        BEGIN OPEN sop.stbase;
        p.ptrL ← LOOPHOLE[sopWd+sa];
        p.sizeL ← BitsForType[sop.tsei];
        p.posnL ← IF p.sizeL < AltoDefs.wordlength THEN
          (AltoDefs.wordlength - p.sizeL) ELSE 0;
        p.localL ← IF sopWd=0 THEN TRUE ELSE FALSE;
        END
      ELSE
        BEGIN OPEN sop.stbase;
        p.ptrR ← LOOPHOLE[sopWd+sa];
        p.sizeR ← BitsForType[sop.tsei];
        p.posnR ← IF p.sizeR < AltoDefs.wordlength THEN
          (AltoDefs.wordlength - p.sizeR) ELSE 0;
        p.localR ← IF sopWd=0 THEN TRUE ELSE FALSE;
        END;
      RETURN
      END;
    DebugMiscDefs.InterpretString[var, SetUpBreakBlock, TRUE
      ! ANY => BEGIN DIActionDefs.CleanUp[]; SIGNAL InvalidCondition END];
    RETURN
    END;

  SetupForLiteral: PROCEDURE [var: STRING, p: UBBPointer] =
    BEGIN
    p.immediateR ← TRUE;
    p.ptrR ← DebugMiscDefs.StringExpressionToNumber[var, 10];
    RETURN
    END;

  SetupForCounter: PROCEDURE [var: STRING, p: UBBPointer] =
    BEGIN
    p.counterL ← TRUE;
    p.relation ← eq;
    p.ptrL ← LOOPHOLE[0];
    p.ptrR ← DebugMiscDefs.StringExpressionToNumber[var, 10];
    RETURN
    END;
```

```
DropLeadingSpaces: PROCEDURE [local: STRING] =
  BEGIN OPEN StringDefs;
  i: CARDINAL;
  ssd: SubStringDescriptor ← [base: local, offset:0, length: 0];
  ss: SubString ← @ssd;
  IF local[0] # IODefs.SP THEN RETURN;
  FOR i IN [0..local.length) DO
    IF local[i] # IODefs.SP THEN
      BEGIN ssd.length ← i; EXIT; END;
    REPEAT
      FINISHED => SIGNAL InvalidCondition;
    ENDLOOP;
  DeleteSubString[ss];
  RETURN
  END;

SetupForRelation: PROCEDURE [condition: STRING, p: UBBPointer] =
  BEGIN OPEN StringDefs;
  i, count: CARDINAL ← 0;
  ssd: SubStringDescriptor ← [base: condition, offset:0, length: 1];
  ss: SubString ← @ssd;
  IF condition[0] = IODefs.SP THEN DropLeadingSpaces[condition];
  SELECT condition[1] FROM
    '= =>
      BEGIN
      SELECT condition[0] FROM
        '< => p.relation ← le;
        '> => p.relation ← ge;
        ENDCASE => SIGNAL InvalidCondition;
      ssd.length ← 2;
      END;
    IODefs.SP, IN['0..'9], IN['a..'z], IN['A..'Z] =>
      SELECT condition[0] FROM
        '< => p.relation ← lt;
        '> => p.relation ← gt;
        '= => p.relation ← eq;
        '# => p.relation ← ne;
        ENDCASE => SIGNAL InvalidCondition;
    ENDCASE => SIGNAL InvalidCondition;
  DeleteSubString[ss];
  RETURN
  END;

CopyString: PROCEDURE [to: STRING, from: STRING] =
  BEGIN
  to.length ← 0;
  StringDefs.AppendString[to, from];
  RETURN
  END;

ByteToWordPC: PROCEDURE [bpc: BytePC] RETURNS [wpc: WordPC] =
  BEGIN
  wpc ← WordPC[bpc/2];
  RETURN[IF bpc MOD 2 = 1 THEN WordPC[-wpc] ELSE wpc]
  END;

WriteUserBreakBlock: PROCEDURE [userBB: UBBPointer] =
  BEGIN OPEN SDDefs, DebugUtilityDefs;
  i: CARDINAL;
  bbHandle: CoreSwapDefs.BBHandle ← SREAD[@SD[sBreakBlock]];
  BEGIN
  i ← GetUserBreakBlock[userBB.frame, userBB.pc
    ! TooManyConditions => GOTO ignore];
  DebugMiscDefs.CopyWrite[from: userBB, to: @bbHandle.blocks[i],
      nwords: SIZE[UserBreakBlock]];
  EXITS
    ignore => WriteBreakError[tooManyConditions];
  END;
  RETURN
  END;

TooManyConditions: SIGNAL = CODE;

GetUserBreakBlock: PROCEDURE [frame: GlobalFrameHandle, pc: WordPC]
  RETURNS [i: CARDINAL] =
  BEGIN OPEN SDDefs, DebugUtilityDefs;
```

```
    bbHandle: CoreSwapDefs.BBHandle ← SREAD[@SD[sBreakBlock]];
    n: CARDINAL ← SREAD[@bbHandle.length];
    FOR i IN [0..n) DO
      IF SREAD[@bbHandle.blocks[i].frame] = frame
      AND SREAD[@bbHandle.blocks[i].pc] = pc
        THEN RETURN[i];
      ENDLOOP;
    IF n >= SREAD[@SD[sBreakBlockSize]] / SIZE[UserBreakBlock]
      THEN SIGNAL TooManyConditions;
    SWRITE[@bbHandle.length,n+1];
    RETURN[n]
    END;

  FreeUserBreakBlock: PROCEDURE [frame: GlobalFrameHandle, pc: WordPC] =
    BEGIN OPEN SDDefs, DebugUtilityDefs;
    i, j: CARDINAL;
    bbHandle: CoreSwapDefs.BBHandle ← SREAD[@SD[sBreakBlock]];
    n: CARDINAL ← SREAD[@bbHandle.length];
    local: UserBreakBlock;
    FOR i IN [0..n) DO
      IF SREAD[@bbHandle.blocks[i].frame] = frame
      AND SREAD[@bbHandle.blocks[i].pc] = pc THEN
        BEGIN
        FOR j IN [i+1..n) DO
          DebugMiscDefs.CopyRead[from: @bbHandle.blocks[j], to: @local,
            nwords: SIZE[UserBreakBlock]];
          DebugMiscDefs.CopyWrite[from: @local, to: @bbHandle.blocks[j-1],
            nwords: SIZE[UserBreakBlock]];
          ENDLOOP;
        SWRITE[@bbHandle.length,n-1];
        END;
      ENDLOOP; --what to do if not found?
    RETURN
    END;

  WriteBreakError: PROCEDURE [error: BreakError] =
    BEGIN
    IODefs.WriteString[SELECT error FROM
      notFound => " breakpoint not found!"L,
      notAllowed => " not allowed here!"L,
      noReturn => " does not return!"L,
      noBreaksSet => "--No breaks have been set!--"L,
      noTracesSet => "--No traces have been set!--"L,
      badCondition => " condition failed"L,
      tooManyConditions => " too many conditions"L,
      noSym => "--symboltable missing - looks mysterious!--"L,
      ENDCASE => "??"L];
    RETURN
    END;

END ...
```