```
DDDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUU         UUU     GGGGGGGGGGGG
DDDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUU         UUU     GGGGGGGGGGGG
DDDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBBBB    UUU         UUU     GGGGGGGGGGGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG
DDD         DDD  EEEEEEEEEEEE     BBBBBBBBBBBB     UUU         UUU  GGG
DDD         DDD  EEEEEEEEEEEE     BBBBBBBBBBBB     UUU         UUU  GGG
DDD         DDD  EEEEEEEEEEEE     BBBBBBBBBBBB     UUU         UUU  GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG   GGGGGGGGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG   GGGGGGGGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG   GGGGGGGGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG         GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG         GGG
DDD         DDD  EEE              BBB         BBB  UUU         UUU  GGG         GGG
DDDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUUUUUUUUUUUUUU     GGGGGGGGG
DDDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUUUUUUUUUUUUUU     GGGGGGGGG
DDDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUUUUUUUUUUUUUU     GGGGGGGGG
```

FILEID**DBGADDEXP


```
DDDDDDDD    BBBBBBBB      GGGGGGGG    AAAAAA    DDDDDDDD    DDDDDDDD    EEEEEEEEEE  XX      XX  PPPPPPPP
DDDDDDDD    BBBBBBBB      GGGGGGGG    AAAAAA    DDDDDDDD    DDDDDDDD    EEEEEEEEEE  XX      XX  PPPPPPPP
DD      DD  BB      BB  GG          AA      AA  DD      DD  DD      DD  EE          XX      XX  PP      PP
DD      DD  BB      BB  GG          AA      AA  DD      DD  DD      DD  EE          XX      XX  PP      PP
DD      DD  BB      BB  GG          AA      AA  DD      DD  DD      DD  EE            XX  XX    PP      PP
DD      DD  BB      BB  GG          AA      AA  DD      DD  DD      DD  EE            XX  XX    PP      PP
DD      DD  BBBBBBBB    GG          AA      AA  DD      DD  DD      DD  EEEEEEEE        XX      PPPPPPPP
DD      DD  BBBBBBBB    GG          AA      AA  DD      DD  DD      DD  EEEEEEEE        XX      PPPPPPPP
DD      DD  BB      BB  GG  GGGGGG  AAAAAAAAAA  DD      DD  DD      DD  EE            XX  XX    PP
DD      DD  BB      BB  GG  GGGGGG  AAAAAAAAAA  DD      DD  DD      DD  EE            XX  XX    PP
DD      DD  BB      BB  GG      GG  AA      AA  DD      DD  DD      DD  EE          XX      XX  PP          ....
DD      DD  BB      BB  GG      GG  AA      AA  DD      DD  DD      DD  EE          XX      XX  PP          ....
DDDDDDDD    BBBBBBBB      GGGGGG    AA      AA  DDDDDDDD    DDDDDDDD    EEEEEEEEEE  XX      XX  PP          ....
DDDDDDDD    BBBBBBBB      GGGGGG    AA      AA  DDDDDDDD    DDDDDDDD    EEEEEEEEEE  XX      XX  PP          ....

LL             IIIIII      SSSSSSSS
LL             IIIIII      SSSSSSSS
LL               II      SS
LL               II      SS
LL               II      SS
LL               II      SS
LL               II        SSSSSS
LL               II        SSSSSS
LL               II              SS
LL               II              SS
LL               II              SS
LL               II              SS
LLLLLLLLLL     IIIIII    SSSSSSSS
LLLLLLLLLL     IIIIII    SSSSSSSS
```

```
0 MODULE DBGADDEXP (IDENT = 'V04-000') =
0
1 BEGIN
1
1 !****************************************************************************
1 !*                                                                          *
1 !*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
1 !*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
1 !*  ALL RIGHTS RESERVED.                                                    *
1 !*                                                                          *
1 !*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
1 !*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
1 !*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
1 !*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
1 !*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
1 !*  TRANSFERRED.                                                            *
1 !*                                                                          *
1 !*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
1 !*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
1 !*  CORPORATION.                                                            *
1 !*                                                                          *
1 !*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
1 !*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
1 !*                                                                          *
1 !*                                                                          *
1 !****************************************************************************
1 !
1 ! WRITTEN BY
1 !       Rich Title      August, 1982
1 !
1 ! MODULE FUNCTION
1 !       This module contains the Address Expression Interpreter.
1 !
1
1 REQUIRE 'SRC$:DBGPROLOG.REQ';
1
1 FORWARD ROUTINE
1     DBG$EVAL_ADDR_OPERATOR,                ! Evaluate an Address Expr. operator
1     DBG$PRIM_TO_ADDR,                      ! Convert Primary Descriptor to
1                                            !     Value Descriptor containing
1                                            !     address of descriptor.
1     DETERMINE_TYPE: NOVALUE,               ! Determine type of inputs
1     GET_DEREFERENCE;                       ! Perform dereference operation
1
1 EXTERNAL
1     DBG$GL_DFLTTYP,                        ! Holds type from SET TYPE command
1     DBG$GW_DFLTLENG: WORD,                 ! Holds length from SET TYPE command
1     DBG$REG_VALUES: VECTOR[,LONG];         ! Register save area
1
1 EXTERNAL ROUTINE
1     DBG$BUILD_PRIMARY_SUBNODE: NOVALUE,    ! Build a Primary Subnode
1     DBG$CONV_TEXT_VALUE,                   ! Convert text string to value
1     DBG$INS_DECODE,                        ! Decode instruction
1     DBG$MAKE_VAL_DESC,                     ! Materialize value into Val Descr
1     DBG$MAKE_VMS_DESC,                     ! Convert Primary Descriptor to
1                                            !     VAX standard descriptor
1     DBG$MAKE_SKELETON_DESC,                ! Build skeleton descriptor
```

```
        DBG$NCOPY_DESC,                         ! Copy descriptors
        DBG$PRIM_TO_VAL,                        ! Convert Primary Descriptor to
                                                !    Value Descriptor.
        DBG$STA_SYMNAME : NOVALUE,              ! Obtain name of symbol from SYMID
        DBG$STA_SYM_IS_LITERAL,                 ! Determine whether a symid represents
                                                !    a literal value.
        DBG$STA_TYPEFCODE,                      ! Find fcode
        DBG$STA_TYP_TYPEDPTR: NOVALUE;          ! Look up typed pointer

    ! Define some codes for the kinds of addresses that a descriptor
    ! can represent.
    !
    LITERAL
        ADDR$K_UNKNOWN = -1,            ! Unknown type
        ADDR$K_MINTYPE = 1,             ! Minimum of known type codes below
        ADDR$K_LITERAL = 1,             ! Literal value
        ADDR$K_PRIMARY = 2,             ! Primary Descriptor
        ADDR$K_INST = 3,                ! Address of instruction
        ADDR$K_DATA = 4,                ! Address of data
        ADDR$K_BITFIELD = 5,            ! Bit field within address
        ADDR$K_MAXTYPE = 5;             ! Maximum of codes above
```

```
GLOBAL ROUTINE DBG$EVAL_ADDR_OPERATOR(OPERATOR, LEFT_ARG, RIGHT_ARG) =
!
! FUNCTION
!       This routine does the actual evaluation of a DEBUG Address Expression
!       operator.  It does a CASE on the operator code and does the appropriate
!       address computation or other operation for those operators allowed in
!       DEBUG Address Expressions.  For any operator not allowed in an Address
!       Expression, it signals an error message.
!
! INPUTS
!       OPERATOR - The Operator Token Entry for the operator to be evaluated.
!
!       LEFT_ARG - A pointer to the left argument Primary Descriptor or Value
!                  Descriptor.  If the operator is a unary operator, LEFT_ARG
!                  points to the operator's one argument.
!
!       RIGHT_ARG - A pointer to the right argument Primary Descriptor or Value
!                  Descriptor.  If the operator is a unary operator, RIGHT_ARG
!                  is not used.
!
! OUTPUTS
!       A pointer to the Value Descriptor or Primary Descriptor
!       which results from the evaluation of
!       the operator is returned as this routine's result.
!

    BEGIN

    MAP
        OPERATOR: REF TOKEN$ENTRY,         ! Token Entry for operator to perform
        LEFT_ARG: REF DBG$VALDESC,         ! Left operand Token Entry
        RIGHT_ARG: REF DBG$VALDESC;        ! Right operand Token Entry

    LOCAL
        BINARY_FLAG,                       ! TRUE if the Address Expression Operator
                                           !     is a binary operator
        LEFT_TYPE,                         ! Address type of left operand (one of
                                           !     the above codes
        OPCODE,                            ! The opcode of the Address Expression Operator
        RIGHT_TYPE;                        ! Address type of right operand (one
                                           !     of the above codes.


    ! The following macro containing the processing that is done on
    ! the ADD and SUBTRACT operators. This processing is the same
    ! except for occurences of either a "+" or a "-". For that reason,
    ! it was separated out as a macro.
    !
    MACRO PROCESS_ADD_OR_SUBTRACT (OP) =

            CASE .LEFT_TYPE FROM ADDR$K_MINTYPE TO ADDR$K_MAXTYPE OF
                SET

                ! Left argument is a Primary Descriptor.
                !
                [ADDR$K_PRIMARY] :
                    BEGIN
```

```
		! For all right args, we need to convert
		! the Primary Descriptor to Value Descriptor before we
		! do any operation.
		!
		DBG$PRIM_TO_VAL (.LEFT_ARG, DBG$K_V_VALUE_DESC, LEFT_ARG);
		RETURN DBG$EVAL_ADDR_OPERATOR (
					.OPERATOR,
					.LEFT_ARG,
					.RIGHT_ARG );
		END;


	[ADDR$K_LITERAL] :
		BEGIN

		IF .RIGHT_TYPE EQL ADDR$K_PRIMARY
		THEN
		    BEGIN
		    DBG$PRIM_TO_VAL (.RIGHT_ARG, DBG$K_V_VALUE_DESC, RIGHT_ARG);
		    RETURN DBG$EVAL_ADDR_OPERATOR (
				.OPERATOR,
				.LEFT_ARG,
				.RIGHT_ARG);
		    END;

		! For all other right operands, just add the literal
		! value to the right operand, and return the right
		! operand. Thus the result retains the type of the
		! right operand.
		!
		RIGHT_ARG [DBG$L_VALUE_POINTER] =
		    .LEFT_ARG [DBG$L_VALUE_POINTER] OP
		    .RIGHT_ARG [DBG$L_VALUE_POINTER];

		! Handle result type of instruction.
		!
		IF .RIGHT_TYPE EQL ADDR$K_INST
		THEN
		    RIGHT_ARG[DBG$W_VALUE_LENGTH] =
			DBG$INS_DECODE(.RIGHT_ARG[DBG$L_VALUE_POINTER],
					FALSE, FALSE) -
			.RIGHT_ARG[DBG$L_VALUE_POINTER];

		! Handle bitfields on the right.
		!
		IF .RIGHT_TYPE EQL ADDR$K_BITFIELD
		THEN
		    BEGIN
		    RIGHT_ARG [DBG$L_VALUE_POS] =
			.LEFT_ARG [DBG$L_VALUE_POS] OP
			.RIGHT_ARG [DBG$L_VALUE_POS];

		    ! Normalize the bit offset.
		    !
		    RIGHT_ARG [DBG$L_VALUE_POINTER] =
			.RIGHT_ARG [DBG$L_VALUE_POINTER] +
```

```
                        .RIGHT_ARG [DBG$L_VALUE_POS] / 8;
                    RIGHT_ARG [DBG$L_VALUE_POS] =
                        .RIGHT_ARG [DBG$L_VALUE_POS] MOD 8;
                    END;

                RETURN .RIGHT_ARG;
                END;

            [ADDR$K_INST, ADDR$K_DATA] :
                BEGIN

                ! Instruction OP Primary
                ! Convert the Primary and try again.
                !
                IF .RIGHT_TYPE EQL ADDR$K_PRIMARY
                THEN
                    BEGIN
                    DBG$PRIM_TO_VAL (.RIGHT_ARG, DBG$K_V_VALUE_DESC, RIGHT_ARG);
                    RETURN DBG$EVAL_ADDR_OPERATOR (
                                .OPERATOR,
                                .LEFT_ARG,
                                .RIGHT_ARG);
                    END;

                ! If the right arg is a bitfield, retain the type
                ! of the right arg.
                !
                IF .RIGHT_TYPE EQL ADDR$K_BITFIELD
                THEN
                    BEGIN
                    RIGHT_ARG [DBG$L_VALUE_POINTER] =
                        .LEFT_ARG  [DBG$L_VALUE_POINTER] OP
                        .RIGHT_ARG [DBG$L_VALUE_POINTER];
                    RIGHT_ARG [DBG$L_VALUE_POS] =
                        .LEFT_ARG [DBG$L_VALUE_POS] OP
                        .RIGHT_ARG [DBG$L_VALUE_POS];

                    ! Normalize the bit offset.
                    !
                    RIGHT_ARG [DBG$L_VALUE_POINTER] =
                        .RIGHT_ARG [DBG$L_VALUE_POINTER] +
                        .RIGHT_ARG [DBG$L_VALUE_POS] / 8;
                    RIGHT_ARG [DBG$L_VALUE_POS] =
                        .RIGHT_ARG [DBG$L_VALUE_POS] MOD 8;
                    RETURN .RIGHT_ARG;
                    END;

                ! Add the addresses.
                !
                LEFT_ARG [DBG$L_VALUE_POINTER] =
                    .LEFT_ARG  [DBG$L_VALUE_POINTER] OP
                    .RIGHT_ARG [DBG$L_VALUE_POINTER];

                ! If we are retaining instruction type then
                ! fill in the length correctly.
                !
                IF .RIGHT_TYPE EQL ADDR$K_LITERAL
```

```
                THEN
                    BEGIN
                    IF .LEFT_TYPE EQL ADDR$K_INST
                    THEN
                        LEFT_ARG[DBG$W_VALUE_LENGTH] =
                            DBG$INS_DECODE(.LEFT_ARG[DBG$L_VALUE_POINTER],
                                           FALSE, FALSE) -
                          .LEFT_ARG[DBG$L_VALUE_POINTER];
                    END

                ! If we are not adding a literal value, zero out
                ! the type information. This reflects the fact
                ! that in expressions such as EX INST OP DATA,
                ! the address being examined is of unknown type.
                !
                ELSE
                    BEGIN
                    LEFT_ARG [DBG$B_DHDR_KIND] = RST$K_DATA;
                    LEFT_ARG [DBG$B_DHDR_FCODE] = RST$K_TYPE_DESCR;
                    LEFT_ARG [DBG$V_DHDR_OVERRIDE] = TRUE;
                    LEFT_ARG [DBG$B_VALUE_CLASS] = 0;
                    LEFT_ARG [DBG$B_VALUE_DTYPE] = .DBG$GL_DFLTTYP;
                    IF .DBG$GL_DFLTTYP EQL DSC$K_DTYPE_ZI
                    THEN
                        LEFT_ARG[DBG$W_VALUE_LENGTH] =
                            DBG$INS_DECODE(.LEFT_ARG[DBG$L_VALUE_POINTER],
                                           FALSE, FALSE) -
                            .LEFT_ARG[DBG$L_VALUE_POINTER]
                    ELSE
                        LEFT_ARG [DBG$W_VALUE_LENGTH] = .DBG$GW_DFLTLENG;
                    END;

                ! Return the left argument.
                !
                RETURN .LEFT_ARG;
                END;


            [ADDR$K_BITFIELD] :
                BEGIN

                ! Bitfield OP Primary.
                ! Convert the Primary and try again.
                !
                IF .RIGHT_TYPE EQL ADDR$K_PRIMARY
                THEN
                    BEGIN
                    DBG$PRIM_TO_VAL (.RIGHT_ARG, DBG$K_V_VALUE_DESC, RIGHT_ARG);
                    RETURN DBG$EVAL_ADDR_OPERATOR (
                            .OPERATOR,
                            .LEFT_ARG,
                            .RIGHT_ARG);
                    END;

                ! In other cases, add the addresses.
                !
                LEFT_ARG [DBG$L_VALUE_POINTER] =
```

```
                        .LEFT_ARG  [DBG$L_VALUE_POINTER] OP
                        .RIGHT_ARG [DBG$L_VALUE_POINTER];
                    LEFT_ARG [DBG$L_VALUE_POS] =
                        .LEFT_ARG [DBG$L_VALUE_POS] OP
                        .RIGHT_ARG [DBG$L_VALUE_POS];

                    ! Normalize the bit offset.
                    !
                    LEFT_ARG [DBG$L_VALUE_POINTER] =
                            .LEFT_ARG [DBG$L_VALUE_POINTER] +
                            .LEFT_ARG [DBG$L_VALUE_POS] / 8;
                    LEFT_ARG [DBG$L_VALUE_POS] =
                            .LEFT_ARG [DBG$L_VALUE_POS] MOD 8;

                    RETURN .LEFT_ARG;
                    END;

                TES %;

    ! Multiply and divide are similar to add and subtract except that
    ! we never try to retain a Primary for symbolization purposes.
    !
    MACRO PROCESS_MULTIPLY_OR_DIVIDE (OP) =
        BEGIN

        ! Handle the case where the left arg is a Primary.
        !
        IF .LEFT_TYPE EQL ADDR$K_PRIMARY
        THEN
            BEGIN

            ! For all right args, we need to convert
            ! the Primary Descriptor to Value Descriptor before we
            ! do any operation.
            !
            DBG$PRIM_TO_VAL (.LEFT_ARG, DBG$K_V_VALUE_DESC, LEFT_ARG);
            RETURN DBG$EVAL_ADDR_OPERATOR (
                                        .OPERATOR,
                                        .LEFT_ARG,
                                        .RIGHT_ARG );
            END;


        ! Handle the case where the right arg is a Primary.
        !
        IF .RIGHT_TYPE EQL ADDR$K_PRIMARY
        THEN
            BEGIN

            ! Convert the Primary and try again.
            !
            DBG$PRIM_TO_VAL (.RIGHT_ARG, DBG$K_V_VALUE_DESC, RIGHT_ARG);
            RETURN DBG$EVAL_ADDR_OPERATOR (
                                .OPERATOR,
                                .LEFT_ARG,
                                .RIGHT_ARG);
```

```
                END;

            ! Neither arg is a Primary.
            ! Multiply the addresses. Zero out the bit offset.
            !
            LEFT_ARG [DBG$L_VALUE_POINTER] =
                    .LEFT_ARG  [DBG$L_VALUE_POINTER] OP
                    .RIGHT_ARG [DBG$L_VALUE_POINTER];

            ! Zero out the bit offset. Change the type to
            ! unknown and return the left arg.
            !
            LEFT_ARG [DBG$V_DHDR_LITERAL] = FALSE;
            LEFT_ARG [DBG$L_VALUE_POS] = 0;
            LEFT_ARG [DBG$B_DHDR_KIND] = RST$K_DATA;
            LEFT_ARG [DBG$B_DHDR_FCODE] = RST$K_TYPE_DESCR;
            LEFT_ARG [DBG$V_DHDR_OVERRIDE] = TRUE;
            LEFT_ARG [DBG$B_VALUE_CLASS] = 0;
            LEFT_ARG [DBG$B_VALUE_DTYPE] = .DBG$GL_DFLTTYP;
            IF .DBG$GL_DFLTTYP EQL DSC$K_DTYPE_ZI
            THEN
                LEFT_ARG[DBG$W_VALUE_LENGTH] =
                    DBG$INS_DECODE(.LEFT_ARG[DBG$L_VALUE_POINTER],
                                   FALSE, FALSE) -
                    .LEFT_ARG[DBG$L_VALUE_POINTER]
            ELSE
                LEFT_ARG [DBG$W_VALUE_LENGTH] = .DBG$GW_DFLTLENG;
            RETURN .LEFT_ARG;
            END %;


        ! Get the opcode and set the flag saying whether we are processing
        ! a binary operator. Also initialize the left and right type codes
        ! to unknown.
        !
        OPCODE = .OPERATOR [TOKEN$W_CODE];
        BINARY_FLAG =   .OPCODE EQL TOKEN$K_ADD
                     OR .OPCODE EQL TOKEN$K_SUBTRACT
                     OR .OPCODE EQL TOKEN$K_MULTIPLY
                     OR .OPCODE EQL TOKEN$K_DIVIDE;


        ! Call the routine which fills in the codes for LEFT_TYPE and
        ! RIGHT_TYPE. If an unconverted value needs to be converted,
        ! this routine constructs a new descriptor and the LEFT_ARG and
        ! RIGHT_ARG pointers may be modified to point to these new
        ! descriptors.
        !
        DETERMINE_TYPE (.LEFT_ARG, LEFT_ARG, LEFT_TYPE);
        IF .BINARY_FLAG
        THEN
            DETERMINE_TYPE (.RIGHT_ARG, RIGHT_ARG, RIGHT_TYPE);

        ! Select the operation to perform based on the operation code.
        !
        CASE .OPERATOR [TOKEN$W_CODE] FROM TOKEN$K_MIN_OPERATOR TO TOKEN$K_MAX_OPERATOR OF
            SET
```

```
    ! Do the identity operation.
    !
    [TOKEN$K_IDENTITY]:
        BEGIN
        RETURN .LEFT_ARG;
        END;

    ! Do the indirection (dereferencing) operation.
    !
    [TOKEN$K_INDIRECT]:
        BEGIN

        BUILTIN
            PROBER;

        LOCAL
            ADDRESS,                    ! Address which is value of left arg
            LENGTH;                     ! Bit length given in left arg.

        ! Primary Descriptors.
        !
        IF .LEFT_TYPE EQL ADDR$K_PRIMARY
        THEN
            BEGIN
            LOCAL
                LANG,
                VMS_DESC: DBG$STG_DESC;

            ! If we can dereference a typed pointer, and return
            ! a Primary representing the pointed-to object,
            ! then do so.
            !
            IF GET_DEREFERENCE(.LEFT_ARG)
            THEN
                RETURN .LEFT_ARG;

            ! Use MAKE_VMS_DESC and MAKE_VAL_DESC to do the fetch.
            ! Preserve the language code.
            !
            LANG = .LEFT_ARG[DBG$B_DHDR_LANG];
            DBG$MAKE_VMS_DESC (.LEFT_ARG, VMS_DESC);
            LEFT_ARG = DBG$MAKE_VAL_DESC (VMS_DESC, DBG$K_VALUE_DESC);
            LEFT_ARG[DBG$B_DHDR_LANG] = .LANG;

            ! Change the descriptor back to a Volatile Value Descriptor.
            !
            LEFT_ARG[DBG$B_DHDR_TYPE] = DBG$K_V_VALUE_DESC;
            LEFT_ARG[DBG$L_VALUE_POINTER] = .LEFT_ARG[DBG$L_VALUE_VALUE0];
            IF .LEFT_ARG[DBG$B_VALUE_CLASS] EQL DSC$K_CLASS_UBS
            THEN
                LEFT_ARG[DBG$L_VALUE_POS] = .LEFT_ARG[DBG$L_VALUE_VALUE1];

            ! Do a special case check for ".PC"
            !
            IF .VMS_DESC[DSC$A_POINTER] EQLA DBG$REG_VALUES[15]
            THEN
```

```
5                       BEGIN
5
5                       ! Change dtype to instruction.
5                       !
5                       LEFT_ARG[DBG$B_VALUE_CLASS] = DSC$K_CLASS_S;
5                       LEFT_ARG[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_ZI;
5                       LEFT_ARG[DBG$W_VALUE_LENGTH] =
5                           DBG$INS_DECODE(.LEFT_ARG[DBG$L_VALUE_POINTER],
5                                          FALSE, FALSE) -
5                           .LEFT_ARG[DBG$L_VALUE_POINTER];
4                       END;
4                   RETURN .LEFT_ARG;
3                   END;
3
3               ! Check that we have read access to the address
3               ! given in the value descriptor.
3               !
3               IF NOT PROBER (%REF(0), %REF(1), .LEFT_ARG[DBG$L_VALUE_POINTER])
3               THEN
3                   SIGNAL (DBG$_NOACCESSR, 1, .LEFT_ARG[DBG$L_VALUE_POINTER]);
3
3               ! For bitfields, do the bit selection.
3               !
3               IF .LEFT_TYPE EQL ADDR$K_BITFIELD
3               THEN
4                   BEGIN
4
4                   ! Do additional checking for length <= 32 bits.
4                   !
4                   LENGTH = .LEFT_ARG [DBG$W_VALUE_LENGTH];
4                   IF .LENGTH LSS 0
4                   OR .LENGTH GTR 32
4                   THEN
4                       SIGNAL (DBG$_ILLSIZFLD, 1, .LENGTH);
4
4                   ! Also check for read access to top of range.
4                   !
4                   ADDRESS = .LEFT_ARG[DBG$L_VALUE_POINTER] +
4                             (.LEFT_ARG[DBG$L_VALUE_POS] + .LENGTH - 1) / 8;
4                   IF NOT PROBER (%REF(0), %REF(1), .ADDRESS)
4                   THEN
4                       SIGNAL (DBG$_NOACCESSR, 1, .ADDRESS);
4
4                   ! Do the bit selection.
4                   !
4                   LEFT_ARG [DBG$L_VALUE_POINTER] =
5                       (IF .LEFT_ARG [DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_SV
5                        OR .LEFT_ARG [DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_SVU
5                        THEN
5                           .(.LEFT_ARG [DBG$L_VALUE_POINTER]) <
5                                 .LEFT_ARG [DBG$L_VALUE_POS],
5                                 .LEFT_ARG [DBG$W_VALUE_LENGTH],
5                                 1>
5                        ELSE
5                           .(.LEFT_ARG [DBG$L_VALUE_POINTER]) <
5                                 .LEFT_ARG [DBG$L_VALUE_POS],
5                                 .LEFT_ARG [DBG$W_VALUE_LENGTH],
```

```
4                              0>));
4                  END
4
3              ELSE
3
3                  ! For other left args, just do the indirection.
3                  !
3                  LEFT_ARG [DBG$L_VALUE_POINTER] = ..LEFT_ARG [DBG$L_VALUE_POINTER];
3
3              ! Change the type to unknown and return the argument.
3              !
3              LEFT_ARG [DBG$V_DHDR_LITERAL] = FALSE;
3              LEFT_ARG [DBG$B_DHDR_KIND] = RST$K_DATA;
3              LEFT_ARG [DBG$B_DHDR_FCODE] = RST$K_TYPE_DESCR;
3              LEFT_ARG [DBG$V_DHDR_OVERRIDE] = TRUE;
3              LEFT_ARG [DBG$B_VALUE_CLASS] = 0;
3              LEFT_ARG [DBG$B_VALUE_DTYPE] = .DBG$GL_DFLTTYP;
3              IF .DBG$GL_DFLTTYP EQL DSC$K_DTYPE_ZI
3              THEN
3                  LEFT_ARG[DBG$W_VALUE_LENGTH] =
3                      DBG$INS_DECODE(.LEFT_ARG[DBG$L_VALUE_POINTER],
3                                     FALSE, FALSE) -
3                      .LEFT_ARG[DBG$L_VALUE_POINTER]
3              ELSE
3                  LEFT_ARG [DBG$W_VALUE_LENGTH] = .DBG$GW_DFLTLENG;
3              LEFT_ARG [DBG$L_VALUE_POS] = 0;
3              RETURN .LEFT_ARG;
2              END;
2
2
2          ! Do the add operation.
2          !
2          [TOKEN$K_ADD]:
2              PROCESS_ADD_OR_SUBTRACT (+);
2
2
2          ! Do the subtract operation.
2          !
2          [TOKEN$K_SUBTRACT]:
2              PROCESS_ADD_OR_SUBTRACT (-);
2
2
2          ! Do the unary plus operation.
2          !
2          [TOKEN$K_UNARY_PLUS]:
2              RETURN .LEFT_ARG;
2
2
2          ! Do the unary minus operation (i.e., negation).
2          !
2          [TOKEN$K_UNARY_MINUS]:
3              BEGIN
3
3              ! First convert Primary Descriptors to Value Descriptors.
3              !
3              IF .LEFT_TYPE EQL ADDR$K_PRIMARY
3              THEN
```

```
                    DBG$PRIM_TO_VAL (.LEFT_ARG, DBG$K_V_VALUE_DESC, LEFT_ARG);

                ! Negate the address.
                !
                LEFT_ARG [DBG$L_VALUE_POINTER] = - .LEFT_ARG [DBG$L_VALUE_POINTER];
                LEFT_ARG [DBG$L_VALUE_POS] = - .LEFT_ARG [DBG$L_VALUE_POS];

                ! Normalize the address.
                !
                LEFT_ARG [DBG$L_VALUE_POINTER] =
                    .LEFT_ARG [DBG$L_VALUE_POINTER] +
                    .LEFT_ARG [DBG$L_VALUE_POS] / 8;
                LEFT_ARG [DBG$L_VALUE_POS] =
                    .LEFT_ARG [DBG$L_VALUE_POS] MOD 8;

                ! Handle result type of instruction.
                !
                IF .LEFT_TYPE EQL ADDR$K_INST
                THEN
                    LEFT_ARG[DBG$W_VALUE_LENGTH] =
                                DBG$INS_DECODE(.LEFT_ARG[DBG$L_VALUE_POINTER],
                                                FALSE, FALSE) =
                                .LEFT_ARG[DBG$L_VALUE_POINTER];

                ! Return the result.
                RETURN .LEFT_ARG;
                END;


            ! Do the multiply operation.
            !
            [TOKEN$K_MULTIPLY]:
                PROCESS_MULTIPLY_OR_DIVIDE (*);


            ! Do the divide operation.
            !
            [TOKEN$K_DIVIDE]:
                PROCESS_MULTIPLY_OR_DIVIDE (/);


            ! Do the bit-selection operation, i.e. X<pos,size,ext>.
            !
            [TOKEN$K_BITSELECT]:
                BEGIN
                LOCAL
                    ADDR_INCREMENT,         ! Increment to be added to byte address
                    BIT_OFFSET,             ! New bit offset from byte address
                    DTYPE,                  ! New dtype
                    POS;

                ! Convert Primaries to Value Descriptors
                !
                IF .LEFT_ARG [DBG$B_DHDR_TYPE] EQL DBG$K_PRIMARY_DESC
                THEN
                    DBG$PRIM_TO_VAL (.LEFT_ARG, DBG$K_V_VALUE_DESC, LEFT_ARG);
```

```
3               !++
3               ! Handle value descriptors.
3               !--
3
3               ! Clear the literal flag - the result is of type bitfield.
3               !
3               LEFT_ARG [DBG$V_DHDR_LITERAL] = FALSE;
3
3               ! Add the bit offsets.
3               !
3               IF .LEFT_ARG[DBG$B_VALUE_CLASS] EQL DSC$K_CLASS_UBS
3               THEN
3                   POS = .LEFT_ARG [DBG$L_VALUE_POS]
3               ELSE
3                   POS = 0;
3               BIT_OFFSET = .POS + .OPERATOR [TOKEN$W_BIT_OFFSET];
3
3               ! Compute the new byte address.
3               !
3               LEFT_ARG [DBG$L_VALUE_POINTER] = .LEFT_ARG [DBG$L_VALUE_POINTER] +
3                                                .BIT_OFFSET / 8;
3
3               ! Compute the bit offset. From it and the sign extension bit,
3               ! determine the new class and dtype.
3               !
3               BIT_OFFSET = .BIT_OFFSET MOD 8;
3               IF .BIT_OFFSET EQL 0
3               THEN
4                   BEGIN
4                   IF .OPERATOR [TOKEN$V_SGNEXT]
4                   THEN
4                       DTYPE = DSC$K_DTYPE_SV
4                   ELSE
4                       DTYPE = DSC$K_DTYPE_V;
4                   LEFT_ARG [DBG$B_VALUE_CLASS] = DSC$K_CLASS_S;
4                   END
3               ELSE
4                   BEGIN
4                   IF .OPERATOR [TOKEN$V_SGNEXT]
4                   THEN
4                       DTYPE = DSC$K_DTYPE_SVU
4                   ELSE
4                       DTYPE = DSC$K_DTYPE_VU;
4                   LEFT_ARG [DBG$B_VALUE_CLASS] = DSC$K_CLASS_UBS;
3                   END;
3
3               ! Fill in the new dtype, bit offset, and length.
3               !
3               LEFT_ARG [DBG$B_VALUE_DTYPE] = .DTYPE;
3               LEFT_ARG [DBG$W_VALUE_LENGTH] = .OPERATOR [TOKEN$W_BIT_LENGTH];
3               LEFT_ARG [DBG$L_VALUE_POS] = .BIT_OFFSET;
3               RETURN .LEFT_ARG;
2               END;
2
2
2           ! Any other code constitutes an error and is signalled as such.
2           ! (The operator is not allowed in Address Expressions.)
```

```
2              !
2              [INRANGE, OUTRANGE]:
3                  BEGIN
3                  SIGNAL(DBG$_INVOPADDR, 1, OPERATOR[TOKEN$B_OPLEN]);
3
3                  ! We never get here but throw in a return to keep the BLISS
3                  ! compiler happy.
3                  !
3                  RETURN 0;
2                  END;
2
2              TES;
1          END;


                                                .TITLE  DBGADDEXP
                                                .IDENT  \V04-000\

                                                .EXTRN  DBG$GL_DFLTTYP, DBG$GW_DFLTLENG
                                                .EXTRN  DBG$REG_VALUES, DBG$BUILD_PRIMARY_SUBNODE
                                                .EXTRN  DBG$CONV_TEXT_VALUE
                                                .EXTRN  DBG$INS_DECODE, DBG$MAKE_VAL_DESC
                                                .EXTRN  DBG$MAKE_VMS_DESC
                                                .EXTRN  DBG$MAKE_SKELETON_DESC
                                                .EXTRN  DBG$NCOPY_DESC, DBG$PRIM_TO_VAL
                                                .EXTRN  DBG$STA_SYMNAME
                                                .EXTRN  DBG$STA_SYM_IS_LITERAL
                                                .EXTRN  DBG$STA_TYPEFCODE
                                                .EXTRN  DBG$STA_TYP_TYPEDPTR

                                                .PSECT  DBG$CODE,NOWRT,  SHR,  PIC,0

                     03FC 00000                 .ENTRY  DBG$EVAL_ADDR_OPERATOR, Save R2,R3,R4,R5,-   ; 0213
                                                        R6,R7,R8,R9                                  ;
59 00000000G   00   9E 00002                    MOVAB   DBG$PRIM_TO_VAL, R9                          ;
58 00000000G   00   9E 00009                    MOVAB   DBG$GW_DFLTLENG, R8                          ;
57 00000000G   00   9E 00010                    MOVAB   DBG$GL_DFLTTYP, R7                           ;
56 00000000G   00   9E 00017                    MOVAB   LIB$SIGNAL, R6                               ;
55 00000000G   00   9E 0001E                    MOVAB   DBG$INS_DECODE, R5                           ;
5E             14   C2 00025                    SUBL2   #20, SP                                      ;
54        04   AC   D0 00028                    MOVL    OPERATOR, R4                                 ; 0533
52        02   A4   3C 0002C                    MOVZWL  2(R4), OPCODE                                ;
               51   D4 00030                    CLRL    R1                                           ; 0534
06             52   D1 00032                    CMPL    OPCODE, #6                                   ;
               02   12 00035                    BNEQ    1$                                           ;
               51   D6 00037                    INCL    R1                                           ;
               50   D4 00039 1$:                CLRL    R0                                           ; 0535
07             52   D1 0003B                    CMPL    OPCODE, #7                                   ;
               02   12 0003E                    BNEQ    2$                                           ;
               50   D6 00040                    INCL    R0                                           ;
50             51   C8 00042 2$:                BISL2   R1, R0                                       ;
               51   D4 00045                    CLRL    R1                                           ; 0536
08             52   D1 00047                    CMPL    OPCODE, #8                                   ;
               02   12 0004A                    BNEQ    3$                                           ;
               51   D6 0004C                    INCL    R1                                           ;
51             50   C8 0004E 3$:                BISL2   R0, R1                                       ;
               50   D4 00051                    CLRL    R0                                           ; 0537
```

```
                          09          52  D1 00053       CMPL    OPCODE, #9                    ;
                                      02  12 00056       BNEQ    4$                            ;
                                      50  D6 00058       INCL    R0                            ;
                52        50          51  C9 0005A 4$:   BISL3   R1, R0, BINARY_FLAG           ;
                                      5E  DD 0005E       PUSHL   SP                            ; 0546
                                08    AC  9F 00060       PUSHAB  LEFT_ARG                      ;
                                08    AC  DD 00063       PUSHL   LEFT_ARG                      ;
                     0000V  CF        03  FB 00066       CALLS   #3, DETERMINE_TYPE            ;
                            0E        52  E9 0006B       BLBC    BINARY_FLAG, 5$               ; 0547
                                04    AE  9F 0006E       PUSHAB  RIGHT_TYPE                    ; 0549
                                0C    AC  9F 00071       PUSHAB  RIGHT_ARG                     ;
                                0C    AC  DD 00074       PUSHL   RIGHT_ARG                     ;
                     0000V  CF        03  FB 00077       CALLS   #3, DETERMINE_TYPE            ;
           0042       8F    01  02    A4  AF 0007C 5$:   CASEW   2(R4), #1, #66                ; 0770
   039C    0097     0086      0086       00083 6$:       .WORD   7$-6$,-                       ;
   03DA    02AE     01B9      03A1       0008B                   7$-6$,-                       ;
   0086    0086     0086      0400       00093                   8$-6$,-                       ;
   0086    0086     0086      0086       0009B                   54$-6$,-                      ;
   0086    0086     0086      0086       000A3                   55$-6$,-                      ;
   0086    0086     0086      0086       000AB                   25$-6$,-                      ;
   0086    0086     0086      0086       000B3                   39$-6$,-                      ;
   0086    0086     0086      0086       000BB                   57$-6$,-                      ;
   0086    0086     0086      0086       000C3                   61$-6$,-                      ;
   0086    0086     0086      0086       000CB                   7$-6$,-                       ;
   0086    0086     0086      0086       000D3                   7$-6$,-                       ;
   0086    0086     0086      0086       000DB                   7$-6$,-                       ;
   0086    0086     0086      0468       000E3                   7$-6$,-                       ;
   039C    0086     0086      0086       000EB                   7$-6$,-                       ;
   0086    0086     0086      0086       000F3                   7$-6$,-                       ;
   0086    0086     0086      0086       000FB                   7$-6$,-                       ;
           0086     0086      0086       00103                   7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
                                                                 7$-6$,-                       ;
```

```
                                                              7$-6$,-                                  ;
                                                              7$-6$,-                                  ;
                                                              7$-6$,-                                  ;
                                                              7$-6$,-                                  ;
                                                              7$-6$,-                                  ;
                                                              70$-6$,-                                 ;
                                                              7$-6$,-                                  ;
                                                              7$-6$,-                                  ;
                                                              7$-6$,-                                  ;
                                                              7$-6$,-                                  ;
                                                              7$-6$,-                                  ;
                                                              7$-6$,-                                  ;
                                                              54$-6$,-                                 ;
                                                              7$-6$,-                                  ;
                                                              7$-6$,-                                  ;
                                                              7$-6$,-                                  ;
                                                              7$-6$,-                                  ;
                                                              7$-6$,-                                  ;
                                                              7$-6$,-                                  ;
                                                              7$-6$,-                                  ;
                                                              7$-6$,-                                  ;
                                                              7$-6$,-                                  ;
                                                              7$-6$,-                                  ;
                                                              7$-6$                                    ;
                        0C    A4    9F 00109 7$:     PUSHAB   12(R4)                                   ; 0843
                              01    DD 0010C         PUSHL    #1                                       ;
                  0002899A    8F    DD 0010E         PUSHL    #166298                                  ;
               66             03    FB 00114         CALLS    #3, LIB$SIGNAL                           ;
                              50    D4 00117         CLRL     R0                                       ; 0848
                                    04 00119         RET                                               ;
               02             6E    D1 0011A 8$:     CMPL     LEFT_TYPE, #2                            ; 0577
                              69    12 0011D         BNEQ     12$                                      ;
                        08    AC    DD 0011F         PUSHL    LEFT_ARG                                 ; 0588
         0000V CF             01    FB 00122         CALLS    #1, GET_DEREFERENCE                      ;
               03             50    E9 00127         BLBC     R0, 9$                                   ;
                            02F2    31 0012A         BRW      54$                                      ;
53     08   BC          08    18    EF 0012D 9$:     EXTZV    #24, #8, @LEFT_ARG, LANG                 ; 0595
                        08    AE    9F 00133         PUSHAB   VMS_DESC                                 ; 0596
                        08    AC    DD 00136         PUSHL    LEFT_ARG                                 ;
       00000000G  00              02    FB 00139         CALLS    #2, DBG$MAKE_VMS_DESC                    ;
               7E       7A    8F    9A 00140         MOVZBL   #122, -(SP)                              ; 0597
                        0C    AE    9F 00144         PUSHAB   VMS_DESC                                 ;
       00000000G  00              02    FB 00147         CALLS    #2, DBG$MAKE_VAL_DESC                    ;
          08   AC             50    D0 0014E         MOVL     R0, LEFT_ARG                             ;
               52       08    AC    D0 00152         MOVL     LEFT_ARG, R2                             ; 0598
          03   A2             53    90 00156         MOVB     LANG, 3(R2)                              ;
          02   A2       83    8F    90 0015A         MOVB     #-125, 2(R2)                             ; 0602
          18   A2       20    A2    D0 0015F         MOVL     32(R2), 24(R2)                           ; 0603
               0D       17    A2    91 00164         CMPB     23(R2), #13                              ; 0604
                              05    12 00168         BNEQ     10$                                      ;
          1C   A2       24    A2    D0 0016A         MOVL     36(R2), 28(R2)                           ; 0606
               50 00000000G   00    9E 0016F 10$:    MOVAB    DBG$REG_VALUES+60, R0                    ; 0610
               50       0C    AE    D1 00176         CMPL     VMS_DESC+4, R0                           ;
                              03    13 0017A         BEQL     11$                                      ;
                            0368    31 0017C         BRW      69$                                      ;
          16   A2     0116    8F    B0 0017F 11$:    MOVW     #278, 22(R2)                             ; 0617
                            034B    31 00185         BRW      67$                                      ; 0619
```

```
                      52        08  AC  D0 00188 12$:   MOVL    LEFT_ARG, R2                        ; 0629
                      54        18  A2  D0 0018C        MOVL    24(R2), R4                          ;
              64      01            00  0C 00190        PROBER  #0, #1, (R4)                        ;
                                    0D  12 00194        BNEQ    13$                                 ;
                                    54  DD 00196        PUSHL   R4                                  ; 0631
                                    01  DD 00198        PUSHL   #1                                  ;
                         00028228   8F  DD 0019A        PUSHL   #164392                             ;
                      66            03  FB 001A0        CALLS   #3, LIB$SIGNAL                      ;
                      05            6E  D1 001A3 13$:   CMPL    LEFT_TYPE, #5                       ; 0635
                                    5A  12 001A6        BNEQ    20$                                 ;
                      53        14  A2  3C 001A8        MOVZWL  20(R2), LENGTH                      ; 0641
                                    05  19 001AC        BLSS    14$                                 ; 0642
                      20            53  D1 001AE        CMPL    LENGTH, #32                         ; 0643
                                    0D  15 001B1        BLEQ    15$                                 ;
                                    53  DD 001B3 14$:   PUSHL   LENGTH                              ; 0645
                                    01  DD 001B5        PUSHL   #1                                  ;
                         00028ED0   8F  DD 001B7        PUSHL   #167632                             ;
                      66            03  FB 001BD        CALLS   #3, LIB$SIGNAL                      ;
              50      53        1C  A2  C1 001C0 15$:   ADDL3   28(R2), LENGTH, R0                  ; 0650
                                    50  D7 001C5        DECL    R0                                  ;
                      50            08  C6 001C7        DIVL2   #8, R0                              ;
                      50            54  C0 001CA        ADDL2   R4, ADDRESS                         ;
              60      01            00  0C 001CD        PROBER  #0, #1, (ADDRESS)                   ; 0651
                                    0D  12 001D1        BNEQ    16$                                 ;
                                    50  DD 001D3        PUSHL   ADDRESS                             ; 0653
                                    01  DD 001D5        PUSHL   #1                                  ;
                         00028228   8F  DD 001D7        PUSHL   #164392                             ;
                      66            03  FB 001DD        CALLS   #3, LIB$SIGNAL                      ;
                      29        16  A2  91 001E0 16$:   CMPB    22(R2), #41                         ; 0658
                                    06  13 001E4        BEQL    17$                                 ;
                      2A        16  A2  91 001E6        CMPB    22(R2), #42                         ; 0659
                                    09  12 001EA        BNEQ    18$                                 ;
  50          64      14  A2    1C  A2  EE 001EC 17$:   EXTV    28(R2), 20(R2), (R4), R0            ; 0663
                                    07  11 001F3        BRB     19$                                 ; 0661
  50          64      14  A2    1C  A2  EF 001F5 18$:   EXTZV   28(R2), 20(R2), (R4), R0            ; 0668
                      18  A2            50  D0 001FC 19$:   MOVL    R0, 24(R2)                      ; 0658
                                    04  11 00200        BRB     21$                                 ; 0635
                      18  A2            64  D0 00202 20$:   MOVL    (R4), 24(R2)                    ; 0676
                      04  A2        40  8F  8A 00206 21$:   BICB2   #64, 4(R2)                      ; 0680
                      06  A2      0603  8F  B0 0020B        MOVW    #1539, 6(R2)                    ; 0682
                      04  A2        80  8F  88 00211        BISB2   #128, 4(R2)                     ; 0683
                          50            67  D0 00216        MOVL    DBG$GL_DFLTTYP, R0              ; 0685
                  16  A2            50  9B 00219        MOVZBW  R0, 22(R2)                          ;
                          16            50  D1 0021D        CMPL    R0, #22                         ; 0686
                                    10  12 00220        BNEQ    22$                                 ;
                                    7E  7C 00222        CLRQ    -(SP)                               ; 0689
                                18  A2  DD 00224        PUSHL   24(R2)                              ;
                      65            03  FB 00227        CALLS   #3, DBG$INS_DECODE                  ;
          14  A2      50        18  A2  A3 0022A        SUBW3   24(R2), R0, 20(R2)                  ; 0691
                                    04  11 00230        BRB     23$                                 ; 0688
                  14  A2            68  B0 00232 22$:   MOVW    DBG$GW_DFLTLENG, 20(R2)             ; 0693
                                1C  A2  D4 00236 23$:   CLRL    28(R2)                              ; 0694
                                  02AB  31 00239 24$:   BRW     69$                                 ; 0770
                      53        08  AC  D0 0023C 25$:   MOVL    LEFT_ARG, R3                        ; 0702
              04      01            6E  CF 00240        CASEL   LEFT_TYPE, #1, #4                   ;
  0047      0047    000A          0016     00244 26$:   .WORD   28$-26$,-                           ;
                                  00BF     0024C                27$-26$,-                           ;
```

```
                                                                  30$-26$,-
                                                                  30$-26$,-
                                                                  36$-26$
                                08    AC  9F 0024E 27$:  PUSHAB   LEFT_ARG
                          7E    83    8F  9A 00251       MOVZBL   #131, -(SP)
                                      53  DD 00255       PUSHL    R3
                                    023E  31 00257       BRW      63$
                          02    04    AE  D1 0025A 28$:  CMPL     RIGHT_TYPE, #2
                                      2F  13 0025E       BEQL     31$
                          52    0C    AC  D0 00260       MOVL     RIGHT_ARG, R2
                    13    A2    18    A3  C0 00264       ADDL2    24(R3), 24(R2)
                          03    04    AE  D1 00269       CMPL     RIGHT_TYPE, #3
                                      0E  12 0026D       BNEQ     29$
                                      7E  7C 0026F       CLRQ     -(SP)
                                18    A2  DD 00271       PUSHL    24(R2)
                          65          03  FB 00274       CALLS    #3, DBG$INS_DECODE
        14    A2          50    18    A2  A3 00277       SUBW3    24(R2), R0, 20(R2)
                          05    04    AE  D1 0027D 29$:  CMPL     RIGHT_TYPE, #5
                                      B6  12 00281       BNEQ     24$
                    1C    A2    1C    A3  C0 00283       ADDL2    28(R3), 28(R2)
                                    0179  31 00288       BRW      52$
                          02    04    AE  D1 0028B 30$:  CMPL     RIGHT_TYPE, #2
                                      76  13 0028F 31$:  BEQL     37$
                          05    04    AE  D1 00291       CMPL     RIGHT_TYPE, #5
                                      29  12 00295       BNEQ     32$
                          50    0C    AC  D0 00297       MOVL     RIGHT_ARG, R0
                          51    08    AC  D0 0029B       MOVL     LEFT_ARG, R1
                    18    A0    18    A1  C0 0029F       ADDL2    24(R1), 24(R0)
                    1C    A0    1C    A1  C0 002A4       ADDL2    28(R1), 28(R0)
              51    1C    A0          08  C7 002A9       DIVL3    #8, 28(R0), R1
                    18    A0          51  C0 002AE       ADDL2    R1, 24(R0)
  7E          00    1C    A0          01  7A 002B2       EMUL     #1, 28(R0), #0, -(SP)
  51          51          8E          08  7B 002B8       EDIV     #8, (SP)+, R1, R1
                                    02A6  31 002BD       BRW      80$
                          50    0C    AC  D0 002C0 32$:  MOVL     RIGHT_ARG, R0
                    18    A3    18    A0  C0 002C4       ADDL2    24(R0), 24(R3)
                          01    04    AE  D1 002C9       CMPL     RIGHT_TYPE, #1
                                      07  12 002CD       BNEQ     33$
                          03          6E  D1 002CF       CMPL     LEFT_TYPE, #3
                                      5B  12 002D2       BNEQ     38$
                                      17  11 002D4       BRB      34$
                    06    A3  0603    8F  B0 002D6 33$:  MOVW     #1539, 6(R3)
                    04    A3    80    8F  88 002DC       BISB2    #128, 4(R3)
                          50          67  D0 002E1       MOVL     DBG$GL_DFLTTYP, R0
                    16    A3          50  9B 002E4       MOVZBW   R0, 22(R3)
                          16          50  D1 002E8       CMPL     R0, #22
                                      10  12 002EB       BNEQ     35$
                                      7E  7C 002ED 34$:  CLRQ     -(SP)
                                18    A3  DD 002EF       PUSHL    24(R3)
                          65          03  FB 002F2       CALLS    #3, DBG$INS_DECODE
        14    A3          50    18    A3  A3 002F5       SUBW3    24(R3), R0, 20(R3)
                                      32  11 002FB       BRB      38$
                    14    A3          68  B0 002FD 35$:  MOVW     DBG$GW_DFLTLENG, 20(R3)
                                      2C  11 00301       BRB      38$
                          02    04    AE  D1 00303 36$:  CMPL     RIGHT_TYPE, #2
                                      4A  13 00307 37$:  BEQL     43$
                          50    0C    AC  D0 00309       MOVL     RIGHT_ARG, R0
```

```
                     18  A3      18  A0  C0 0030D          ADDL2   24(R0), 24(R3)              :
                     1C  A3      1C  A0  C0 00312          ADDL2   28(R0), 28(R3)              :
             50      1C  A3      08  C7 00317              DIVL3   #8, 28(R3), R0              :
                     18  A3      50  C0 0031C              ADDL2   R0, 24(R3)                  :
      7E     00      1C  A3      01  7A 00320              EMUL    #1, 28(R3), #0, -(SP)       :
      50     50          8E      08  7B 00326              EDIV    #8, (SP)+, R0, R0           :
                     1C  A3      50  D0 0032B              MOVL    R0, 28(R3)                  :
                             67  11 0032F 38$:             BRB     45$                         :
                         52  08  AC  D0 00331 39$:         MOVL    LEFT_ARG, R2                : 0708
             04          01      6E  CF 00335              CASEL   LEFT_TYPE, #1, #4           :
    0063   0063        000A    0016     00339 40$:         .WORD   42$-40$,-                   :
                               00B7     00341                      41$-40$,-                   :
                                                                   46$-40$,-                   :
                                                                   46$-40$,-                   :
                                                                   50$-40$                     :
                         08  AC  9F 00343 41$:             PUSHAB  LEFT_ARG                    :
                     7E  83  8F  9A 00346                  MOVZBL  #131, -(SP)                 :
                             52  DD 0034A                  PUSHL   R2                          :
                           0149  31 0034C                  BRW     63$                         :
                     02  04  AE  D1 0034F 42$:             CMPL    RIGHT_TYPE, #2              :
                             4B  13 00353 43$:             BEQL    47$                         :
                         53  0C  AC  D0 00355              MOVL    RIGHT_ARG, R3               :
         18  A3      18  A2  18  A3  C3 00359              SUBL3   24(R3), 24(R2), 24(R3)      :
                     03  04  AE  D1 00360                  CMPL    RIGHT_TYPE, #3              :
                             0E  12 00364                  BNEQ    44$                         :
                             7E  7C 00366                  CLRQ    -(SP)                       :
                         18  A3  DD 00368                  PUSHL   24(R3)                      :
                     65  03  FB 0036B                      CALLS   #3, DBG$INS_DECODE          :
         14  A3      50  18  A3  A3 0036E                  SUBW3   24(R3), R0, 20(R3)          :
                     05  04  AE  D1 00374 44$:             CMPL    RIGHT_TYPE, #5              :
                             1E  12 00378                  BNEQ    45$                         :
                     50  1C  A3  9E 0037A                  MOVAB   28(R3), R0                  :
             60      1C  A2      60  C3 0037E              SUBL3   (R0), 28(R2), (R0)          :
             51          60      08  C7 00383              DIVL3   #8, (R0), R1                :
                     18  A3      51  C0 00387              ADDL2   R1, 24(R3)                  :
      7E     00          60      01  7A 0038B              EMUL    #1, (R0), #0, -(SP)         :
      51     51          8E      08  7B 00390              EDIV    #8, (SP)+, R1, R1           :
                         60      51  D0 00395              MOVL    R1, (R0)                    :
                         50      53  D0 00398 45$:         MOVL    R3, R0                      :
                                 04 0039B                  RET                                 :
                     02  04  AE  D1 0039C 46$:             CMPL    RIGHT_TYPE, #2              :
                             52  13 003A0 47$:             BEQL    51$                         :
                     05  04  AE  D1 003A2                  CMPL    RIGHT_TYPE, #5              :
                             2E  12 003A6                  BNEQ    48$                         :
                     50  0C  AC  D0 003A8                  MOVL    RIGHT_ARG, R0               :
                     51  08  AC  D0 003AC                  MOVL    LEFT_ARG, R1                :
         18  A0      18  A1  18  A0  C3 003B0              SUBL3   24(R0), 24(R1), 24(R0)      :
                     53  1C  A0  9E 003B7                  MOVAB   28(R0), R3                  :
             63      1C  A1      63  C3 003BB              SUBL3   (R3), 28(R1), (R3)          :
             51          63      08  C7 003C0              DIVL3   #8, (R3), R1                :
                     18  A0      51  C0 003C4              ADDL2   R1, 24(R0)                  :
      7E     00          63      01  7A 003C8              EMUL    #1, (R3), #0, -(SP)         :
      51     51          8E      08  7B 003CD              EDIV    #8, (SP)+, R1, R1           :
                         63      51  D0 003D2              MOVL    R1, (R3)                    :
                                 04 003D5                  RET                                 :
                     50  0C  AC  D0 003D6 48$:             MOVL    RIGHT_ARG, R0               :
                 18  A2  18  A0  C2 003DA                  SUBL2   24(R0), 24(R2)              :
```

```
                          01    04    AE  D1 003DF        CMPL    RIGHT_TYPE, #1                      ;
                                      03  13 003E3        BEQL    49$                                 ;
                                    00D4  31 003E5        BRW     66$                                 ;
                          03          6E  D1 003E8  49$:  CMPL    LEFT_TYPE, #3                       ;
                                      2F  12 003EB        BNEQ    53$                                 ;
                                    00E3  31 003ED        BRW     67$                                 ;
                          02    04    AE  D1 003F0  50$:  CMPL    RIGHT_TYPE, #2                      ;
                                      7C  13 003F4  51$:  BEQL    60$                                 ;
                          50    0C    AC  D0 003F6        MOVL    RIGHT_ARG, R0                       ;
                    18    A2    18    A0  C2 003FA        SUBL2   24(R0), 24(R2)                      ;
                    1C    A2    1C    A0  C2 003FF        SUBL2   28(R0), 28(R2)                      ;
              50    1C    A2          08  C7 00404  52$:  DIVL3   #8, 28(R2), R0                      ;
                    18    A2          50  C0 00409        ADDL2   R0, 24(R2)                          ;
        7E    00    1C    A2          01  7A 0040D        EMUL    #1, 28(R2), #0, -(SP)               ;
        50    50          8E          08  7B 00413        EDIV    #8, (SP)+, R0, R0                   ;
                    1C    A2          50  D0 00418        MOVL    R0, 28(R2)                          ;
                                    00C8  31 0041C  53$:  BRW     69$                                 ;
                          50    08    AC  D0 0041F  54$:  MOVL    LEFT_ARG, R0                        ; 0770
                                          04 00423        RET                                         ;
                          02          6E  D1 00424  55$:  CMPL    LEFT_TYPE, #2                       ; 0724
                                      0D  12 00427        BNEQ    56$                                 ;
                                08    AC  9F 00429        PUSHAB  LEFT_ARG                            ; 0726
                          7E    83    8F  9A 0042C        MOVZBL  #131, -(SP)                         ;
                                08    AC  DD 00430        PUSHL   LEFT_ARG                            ;
                          69          03  FB 00433        CALLS   #3, DBG$PRIM_TO_VAL                 ;
                          52    08    AC  D0 00436  56$:  MOVL    LEFT_ARG, R2                        ; 0730
                    18    A2    18    CE     0043A        MNEGL   24(R2), 24(R2)                      ;
                          50    1C    A2  9E 0043F        MOVAB   28(R2), R0                          ; 0731
                          60          60  CE 00443        MNEGL   (R0), (R0)                          ;
              51          60          08  C7 00446        DIVL3   #8, (R0), R1                        ; 0737
                    18    A2          51  C0 0044A        ADDL2   R1, 24(R2)                          ;
        7E    00          60          01  7A 0044E        EMUL    #1, (R0), #0, -(SP)                 ; 0739
        51    51          8E          08  7B 00453        EDIV    #8, (SP)+, R1, R1                   ;
                          60          51  D0 00458        MOVL    R1, (R0)                            ;
                                      8B  11 0045B        BRB     49$                                 ; 0743
                          02          6E  D1 0045D  57$:  CMPL    LEFT_TYPE, #2                       ; 0758
                                      0C  12 00460        BNEQ    59$                                 ;
                                08    AC  9F 00462  58$:  PUSHAB  LEFT_ARG                            ;
                          7E    83    8F  9A 00465        MOVZBL  #131, -(SP)                         ;
                                08    AC  DD 00469        PUSHL   LEFT_ARG                            ;
                                      2A  11 0046C        BRB     63$                                 ;
                          02    04    AE  D1 0046E  59$:  CMPL    RIGHT_TYPE, #2                      ;
                                      1A  13 00472  60$:  BEQL    62$                                 ;
                          52    08    AC  D0 00474        MOVL    LEFT_ARG, R2                        ;
                          50    0C    AC  D0 00478        MOVL    RIGHT_ARG, R0                       ;
                    18    A2    18    A0  C4 0047C        MULL2   24(R0), 24(R2)                      ;
                                      31  11 00481        BRB     65$                                 ;
                          02          6E  D1 00483  61$:  CMPL    LEFT_TYPE, #2                       ; 0764
                                      DA  13 00486        BEQL    58$                                 ;
                          02    04    AE  D1 00488        CMPL    RIGHT_TYPE, #2                      ;
                                      19  12 0048C        BNEQ    64$                                 ;
                                0C    AC  9F 0048E  62$:  PUSHAB  RIGHT_ARG                           ;
                          7E    83    8F  9A 00491        MOVZBL  #131, -(SP)                         ;
                                0C    AC  DD 00495        PUSHL   RIGHT_ARG                           ;
                          69          03  FB 00498  63$:  CALLS   #3, DBG$PRIM_TO_VAL                 ;
                          7E    08    AC  7D 0049B        MOVQ    LEFT_ARG, -(SP)                     ;
                                      54  DD 0049F        PUSHL   R4                                  ;
```

```
                             FB5A   CF         03  FB 004A1        CALLS   #3, DBG$EVAL_ADDR_OPERATOR              ;
                                                   04 004A6        RET                                             ;
                                    52   08    AC  D0 004A7 64$:   MOVL    LEFT_ARG, R2                            ;
                                    50   0C    AC  D0 004AB        MOVL    RIGHT_ARG, R0                           ;
                               18   A2   18    A0  C6 004AF        DIVL2   24(R0), 24(R2)                          ;
                               04   A2   40    8F  8A 004B4 65$:   BICB2   #64, 4(R2)                              ;
                                         1C    A2  D4 004B9        CLRL    28(R2)                                  ;
                               06   A2 0603    8F  B0 004BC 66$:   MOVW    #1539, 6(R2)                            ;
                               04   A2   80    8F  88 004C2        BISB2   #128, 4(R2)                             ;
                                    50         67  D0 004C7        MOVL    DBG$GL_DFLTTYP, R0                      ;
                               16   A2         50  9B 004CA        MOVZBW  R0, 22(R2)                              ;
                                    16         50  D1 004CE        CMPL    R0, #22                                 ;
                                               10  12 004D1        BNEQ    68$                                     ;
                                               7E  7C 004D3 67$:   CLRQ    -(SP)                                   ;
                                         18    A2  DD 004D5        PUSHL   24(R2)                                  ;
                                    65         03  FB 004D8        CALLS   #3, DBG$INS_DECODE                      ;
               14   A2              50   18    A2  A3 004DB        SUBW3   24(R2), R0, 20(R2)                      ;
                                               04  11 004E1        BRB     69$                                     ;
                               14   A2         68  B0 004E3 68$:   MOVW    DBG$GW_DFLTLENG, 20(R2)                 ;
                                    50         52  D0 004E7 69$:   MOVL    R2, R0                                  ; 0770
                                                   04 004EA        RET                                             ;
00000079   8F  08   BC              08         10  ED 004EB 70$:   CMPZV   #16, #8, @LEFT_ARG, #121                ; 0779
                                               0D  12 004F5        BNEQ    71$                                     ;
                                         08    AC  9F 004F7        PUSHAB  LEFT_ARG                                ; 0781
                                    7E   83    8F  9A 004FA        MOVZBL  #131, -(SP)                             ;
                                         08    AC  DD 004FE        PUSHL   LEFT_ARG                                ;
                                    69         03  FB 00501        CALLS   #3, DBG$PRIM_TO_VAL                     ;
                                    50   08    AC  D0 00504 71$:   MOVL    LEFT_ARG, R0                            ; 0789
                               04   A0   40    8F  8A 00508        BICB2   #64, 4(R0)                              ;
                                    53   14    A0  9E 0050D        MOVAB   20(R0), R3                              ; 0793
                                    0D   03    A3  91 00511        CMPB    3(R3), #13                              ;
                                               06  12 00515        BNEQ    72$                                     ;
                                    51   1C    A0  D0 00517        MOVL    28(R0), POS                             ; 0795
                                               02  11 0051B        BRB     73$                                     ;
                                               51  D4 0051D 72$:   CLRL    POS                                     ; 0797
                                    52   08    A4  3C 0051F 73$:   MOVZWL  8(R4), R2                               ; 0798
                                    51         52  C0 00523        ADDL2   R2, BIT_OFFSET                          ;
                    52              51         08  C7 00526        DIVL3   #8, BIT_OFFSET, R2                      ; 0803
                               18   A0         52  C0 0052A        ADDL2   R2, 24(R0)                              ;
           7E       00              51         01  7A 0052E        EMUL    #1, BIT_OFFSET, #0, -(SP)               ; 0808
           51       51              8E         08  7B 00533        EDIV    #8, (SP)+, BIT_OFFSET, BIT_OFFSET       ;
                                               51  D5 00538        TSTL    BIT_OFFSET                              ; 0809
                                               12  12 0053A        BNEQ    76$                                     ;
                    05              64         0A  E1 0053C        BBC     #10, (R4), 74$                          ; 0812
                                    52         29  D0 00540        MOVL    #41, DTYPE                              ; 0814
                                               03  11 00543        BRB     75$                                     ;
                                    52         01  D0 00545 74$:   MOVL    #1, DTYPE                               ; 0816
                               03   A3         01  90 00548 75$:   MOVB    #1, 3(R3)                               ; 0817
                                               10  11 0054C        BRB     79$                                     ; 0809
                    05              64         0A  E1 0054E 76$:   BBC     #10, (R4), 77$                          ; 0821
                                    52         2A  D0 00552        MOVL    #42, DTYPE                              ; 0823
                                               03  11 00555        BRB     78$                                     ;
                                    52         22  D0 00557 77$:   MOVL    #34, DTYPE                              ; 0825
                               03   A3         0D  90 0055A 78$:   MOVB    #13, 3(R3)                              ; 0826
                               02   A3         52  90 0055E 79$:   MOVB    DTYPE, 2(R3)                            ; 0831
                                    63   0A    A4  B0 00562        MOVW    10(R4), (R3)                            ; 0832
                               1C   A0         51  D0 00566 80$:   MOVL    BIT_OFFSET, 28(R0)                      ; 0833
```

```
                                04 0056A        RET                                           ; 0852

; Routine Size:  1387 bytes,    Routine Base:  DBG$CODE + 0000
```

```
1 GLOBAL ROUTINE DBG$PRIM_TO_ADDR (DESC1, DTYPE, DESC2) =
1 !
1 ! FUNCTION
1 !       Converts a Primary Descriptor into a value descriptor containing
1 !       the address of the primary.
1 !
1 !       For Primary Descriptors representing data or code, the address is stored
1 !       as a (byte address, bit offset) pair in the value field of the
1 !       descriptor. If the given dtype is zero, the dtype is left alone. If the
1 !       user supplied a dtype that he wants the result to be, this
1 !       new dtype is stuffed in and the class and length fields are fixed
1 !       up accordingly.
1 !
1 !       For Primary Descriptors representing literals, the literal value
1 !       is stored in the value field of the resulting descriptor. The
1 !       dtype information is left alone in this case - it presumably
1 !       describes the type of the literal constant.
1 !
1 !       Note that BLISS fields are a special kind of literal. The four
1 !       field values are stored in four longwords in the value field
1 !       of the resulting value descriptor.
1 !
1 !
1 ! INPUTS
1 !       DESC1   - The address of the Primary Descriptor to be converted.
1 !       DTYPE   - The desired Dtype code of the resulting value descriptor
1 !       DESC2   - An address in which to leave a pointer to the new value
1 !                 descriptor which is constructed.
1 !
1 ! OUTPUTS
1 !       A Value Descriptor is allocated and filled in, and a pointer
1 !       to it is left in DESC2. A status code is returned, which is
1 !       one of:
1 !       STS$K_SUCCESS   - Success.
1 !       STS$K_SEVERE    - Failure.
1 !
2   BEGIN
2
2   MAP
2       DESC1: REF DBG$PRIMARY;          ! Pointer to input primary descriptor
2
2   LOCAL
2       COUNT,                           ! For BLISS fields, count of fields
2       DSTPTR: REF DST$RECORD,          ! Pointer to DST record for the primary
2       PTR,                             ! Scratch pointer
2       RSTPTR: REF RST$ENTRY,           ! Pointer to RST entry for the primary
2       SUBVECTOR: REF VECTOR[],         ! Pointer to vector of field values
2       TEMP_DESC: REF DBG$VALDESC;      ! Pointer to a value descriptor
2
2
2   ! Check for a volatile value descriptor coming in. In this case,
2   ! turn it into an ordinary value descriptor with the "address" in
2   ! the V. Value desc becoming the "value" in the ordinary value desc.
2   !
2   IF .DESC1[DBG$B_DHDR_TYPE] EQL DBG$K_V_VALUE_DESC
2   THEN
3       BEGIN
```

```
3              MAP
3                  DESC2: REF DBG$VALDESC;
3              LOCAL
3                  DUMMY;
3
3              ! Slightly kludgy solution to an obscure problem:
3              ! If you examine a string of length > 256 and language is
3              ! set to BLISS or MACRO then you get into this routine
3              ! with a volatile value descriptor representing the string.
3              ! In this case we just want to return the descriptor unchanged:
3              ! do not add the level of indirection.
3              !
3              IF .DESC1[DBG$W_VALUE_LENGTH] GTR 256
3              THEN
4                  BEGIN
4                  .DESC2 = .DESC1;
4                  RETURN STS$K_SUCCESS;
3                  END;
3
3              ! Otherwise, copy the descriptor and make it into an ordinary
3              ! value descriptor.
3              !
3              DBG$NCOPY_DESC(.DESC1, TEMP_DESC, DUMMY, FALSE);
3              TEMP_DESC[DBG$B_DHDR_TYPE] = DBG$K_VALUE_DESC;
3              TEMP_DESC[DBG$L_VALUE_VALUE0] = .TEMP_DESC[DBG$L_VALUE_POINTER];
3              IF .TEMP_DESC[DBG$B_VALUE_CLASS] EQL DSC$K_CLASS_UBS
3              THEN
3                  TEMP_DESC[DBG$L_VALUE_VALUE1] = .TEMP_DESC[DBG$L_VALUE_POS];
3              TEMP_DESC[DBG$L_VALUE_POINTER] = TEMP_DESC[DBG$L_VALUE_VALUE0];
3              .DESC2 = .TEMP_DESC;
3              RETURN STS$K_SUCCESS;
2              END;
2
2
2          ! Check for BLISS field. This is a hack needed to support BLISS structure
2          ! references X[fieldname]. Basically, when we parse the "fieldname"
2          ! primary, we are expecting either an offset as in X[0,0,32,0] or a
2          ! fieldname as in X[fieldname].
2          ! If we get a fieldname, we indicate it by building a special
2          ! kind of value descriptor whose fcode is
2          ! RST$K_TYPE_BLIFLD, and in the value field we have the four integers
2          ! that the fieldname translates into.
2          !
2          IF .DESC1 [DBG$B_DHDR_FCODE] EQL RST$K_TYPE_BLIFLD
2          THEN
3              BEGIN
3              RSTPTR = .DESC1 [DBG$L_DHDR_SYMID0];
3              DSTPTR = .RSTPTR [RST$L_DSTPTR];
3
3              ! Set up a pointer to the BLISS field components,
3              ! and copy those four components into the value descriptor.
3              !
3              COUNT = .DSTPTR[DST$L_BLIFLD_COMPS];
3              PTR = 1 + DSTPTR[DST$B_NAME] + .DSTPTR[DST$B_NAME];
3
3              ! Allocate the result descriptor now that we know how big
3              ! it needs to be.
```

```
3              !
3              TEMP_DESC = DBG$MAKE_SKELETON_DESC (DBG$K_VALUE_DESC, 4*(.COUNT+1));
3              TEMP_DESC [DBG$B_DHDR_LANG]   = .DESC1 [DBG$B_DHDR_LANG];
3              TEMP_DESC [DBG$B_DHDR_KIND]   = .DESC1 [DBG$B_DHDR_KIND];
3              TEMP_DESC [DBG$B_DHDR_FCODE]  = .DESC1 [DBG$B_DHDR_FCODE];
3              TEMP_DESC [DBG$L_DHDR_TYPEID] = .DESC1 [DBG$L_DHDR_TYPEID];
3
3              SUBVECTOR = TEMP_DESC [DBG$A_VALUE_ADDRESS];
3              SUBVECTOR[0] = .COUNT;
3              INCR I FROM 1 TO .COUNT DO
4                  BEGIN
4                  SUBVECTOR [.I] = ..PTR;
4                  PTR = .PTR + 4;
3                  END;
3
3              ! Fix up the pointer field, fill in the output parameter, and
3              ! return success.
3              !
3              TEMP_DESC [DBG$L_VALUE_POINTER] = TEMP_DESC [DBG$A_VALUE_ADDRESS];
3              .DESC2 = .TEMP_DESC;
3              RETURN STS$K_SUCCESS;
2              END;
2
2          ! Check for literal constants.
2          !
2          IF DBG$STA_SYM_IS_LITERAL (.DESC1 [DBG$L_DHDR_SYMID0])
2          THEN
3              BEGIN
3
3              ! PRIM_TO_VAL does what we want with literals, so we just
3              ! call that routine.
3              !
3              DBG$PRIM_TO_VAL (.DESC1, DBG$K_VALUE_DESC, .DESC2);
3              RETURN STS$K_SUCCESS;
2              END;
2
2          ! Build a value descriptor of the desired type.
2          ! Fill in the header fields of this
2          ! value descriptor.
2          !
2          TEMP_DESC = DBG$MAKE_SKELETON_DESC (DBG$K_VALUE_DESC, 4);
2          TEMP_DESC [DBG$B_DHDR_LANG]   = .DESC1 [DBG$B_DHDR_LANG];
2          TEMP_DESC [DBG$B_DHDR_KIND]   = .DESC1 [DBG$B_DHDR_KIND];
2          TEMP_DESC [DBG$B_DHDR_FCODE]  = .DESC1 [DBG$B_DHDR_FCODE];
2          TEMP_DESC [DBG$L_DHDR_TYPEID] = .DESC1 [DBG$L_DHDR_TYPEID];
2
2          ! Set up for a call to DBG$MAKE_VMS_DESC. This routine (in DBGVALUES)
2          ! converts Primary Descriptors to VMS standard descriptors. Since the
2          ! value descriptor has a VMS descriptor inside it, we pass this
2          ! to DBG$MAKE_VMS_DESC.
2
2          IF NOT DBG$MAKE_VMS_DESC (.DESC1, TEMP_DESC[DBG$A_VALUE_VMSDESC])
2          THEN
2              RETURN STS$K_SEVERE;
2
2          ! Normalize the pointer to include the bit offset.
2          !
```

```
;  2      IF .TEMP_DESC [DBG$B_VALUE_CLASS] EQL DSC$K_CLASS_UBS
;  2      THEN
;  3          BEGIN
;  3          TEMP_DESC [DBG$L_VALUE_POINTER] = .TEMP_DESC [DBG$L_VALUE_POINTER] +
;  3                  .TEMP_DESC [DBG$L_VALUE_POS] / %BPUNIT;
;  3          TEMP_DESC [DBG$L_VALUE_POS] = .TEMP_DESC [DBG$L_VALUE_POS] MOD %BPUNIT;
;  2          END;
;  2
;  2      ! If this routine was passed a nonzero DTYPE, fill in the desired DTYPE.
;  2      ! Also fill in all the related fields.
;  2      ! Otherwise, leave it the way we got it from MAKE_VMS_DESC.
;  2      !
;  2      IF .DTYPE NEQ 0
;  2      THEN
;  2
;  2          ! The only case where we dummy in a different dtype is for address
;  2          ! arithmetic in BLISS. E.g., EVAL F+F, without dots, we want to
;  2          ! treat the address of F as a longword integer.
;  2          !
;  2          ! In this case, the dtype we pass in is L.
;  2          !
;  2          IF .DTYPE EQL DSC$K_DTYPE_L
;  2          THEN
;  3              BEGIN
;  3              TEMP_DESC [DBG$B_DHDR_KIND] = RST$K_DATA;
;  3              TEMP_DESC [DBG$B_DHDR_FCODE] = RST$K_TYPE_DESCR;
;  3              TEMP_DESC [DBG$B_VALUE_CLASS] = DSC$K_CLASS_S;
;  3              TEMP_DESC [DBG$B_VALUE_DTYPE]  = DSC$K_DTYPE_L;
;  3              TEMP_DESC [DBG$W_VALUE_LENGTH] = 4;
;  3              END
;  3
;  3          ! We do not currently handle other override dtypes here.
;  3          !
;  2          ELSE
;  2              $DBG_ERROR ('DBGADDEXP\DBG$PRIM_TO_ADDR unknown dtype');
;  2
;  2      ! Extract the desired information from the value descriptor.
;  2      !
;  2      TEMP_DESC[DBG$L_VALUE_VALUE0] = .TEMP_DESC[DBG$L_VALUE_POINTER];
;  2      TEMP_DESC[DBG$L_VALUE_VALUE1] = .TEMP_DESC[DBG$L_VALUE_POS];
;  2
;  2      ! Fix up the pointer field, fill in the output parameter, and
;  2      ! return success.
;  2      !
;  2      TEMP_DESC [DBG$L_VALUE_POINTER] = TEMP_DESC [DBG$A_VALUE_ADDRESS];
;  2      .DESC2 = .TEMP_DESC;
;  2      RETURN STS$K_SUCCESS
;  1      END;
```

```
                                                                       .PSECT  DBG$PLIT,NOWRT,  SHR,  PIC,0

24 47 42 44 5C 50 58 45 44 44 41 47 42 44 28 00000 P.AAA:  .ASCII  \(DBGADDEXP\<92>\DBG$PRIM_TO_ADDR unknow\  ;
6E 75 20 52 44 44 41 5F 4F 54 5F 4D 49 52 50 0000F                                                           ;
                                  77 6F 6E 6B 0001E                                                           ;
                      65 70 79 74 64 20 6E 00022          .ASCII  \n dtype\                                  ;
```

```
                                                            .PSECT  DBG$CODE,NOWRT,  SHR,  PIC,0

                                    00FC 00000              .ENTRY  DBG$PRIM_TO_ADDR, Save R2,R3,R4,R5,R6,R7      ; 0853
                     57 00000000G  00   9E 00002              MOVAB   DBG$MAKE_SKELETON_DESC, R7                    ;
                                   5E   08   C2 00009         SUBL2   #8, SP                                        ;
                          53   04   AC   D0 0000C             MOVL    DESC1, R3                                     ; 0907
              83   8F   02   A3   91 00010                    CMPB    2(R3), #131                                   ;
                                  40   12 00015               BNEQ    3$                                            ;
           0100   8F   14   A3   B1 00017                     CMPW    20(R3), #256                                  ; 0922
                                  07   1B 0001D               BLEQU   1$                                            ;
              0C   BC        53   D0 0001F                    MOVL    R3, @DESC2                                    ; 0925
                                013C   31 00023               BRW     14$                                           ; 0926
                                  7E   D4 00026  1$:          CLRL    -(SP)                                         ; 0932
                          04   AE   9F 00028                  PUSHAB  DUMMY                                         ;
                          0C   AE   9F 0002B                  PUSHAB  TEMP_DESC                                     ;
                                  53   DD 0002E               PUSHL   R3                                            ;
      00000000G   00         04   FB 00030                    CALLS   #4, DBG$NCOPY_DESC                            ;
                   50   04   AE   D0 00037                    MOVL    TEMP_DESC, R0                                 ; 0933
              02   A0   7A   8F   90 0003B                    MOVB    #122, 2(R0)                                   ;
              20   A0   18   A0   D0 00040                    MOVL    24(R0), 32(R0)                                ; 0934
                   0D   17   A0   91 00045                    CMPB    23(R0), #13                                   ; 0935
                                  05   12 00049               BNEQ    2$                                            ;
              24   A0   1C   A0   D0 0004B                    MOVL    28(R0), 36(R0)                                ; 0937
              18   A0   20   A0   9E 00050  2$:               MOVAB   32(R0), 24(R0)                                ; 0938
                                  56   11 00055               BRB     6$                                            ; 0939
                   0E   06   A3   91 00057  3$:               CMPB    6(R3), #14                                    ; 0953
                                  56   12 0005B               BNEQ    7$                                            ;
                   50   0C   A3   D0 0005D                    MOVL    12(R3), RSTPTR                                ; 0956
                   50   0C   A0   D0 00061                    MOVL    12(RSTPTR), DSTPTR                            ; 0957
                   52   03   A0   D0 00065                    MOVL    3(DSTPTR), COUNT                              ; 0962
                   51   07   A0   9A 00069                    MOVZBL  7(DSTPTR), R1                                 ; 0963
                   56   08 A140   9E 0006D                    MOVAB   8(R1)[DSTPTR], PTR                            ;
      7E           52        02   78 00072                    ASHL    #2, COUNT, -(SP)                              ; 0968
                   6E        04   C0 00076                    ADDL2   #4, (SP)                                      ;
                   7E   7A   8F   9A 00079                    MOVZBL  #122, -(SP)                                   ;
                   67        02   FB 0007D                    CALLS   #2, DBG$MAKE_SKELETON_DESC                    ;
              04   AE        50   D0 00080                    MOVL    R0, TEMP_DESC                                 ;
              03   A0   03   A3   90 00084                    MOVB    3(R3), 3(R0)                                  ; 0969
              06   A0   06   A3   B0 00089                    MOVW    6(R3), 6(R0)                                  ; 0971
              08   A0   08   A3   D0 0008E                    MOVL    8(R3), 8(R0)                                  ; 0972
                   55   20   A0   9E 00093                    MOVAB   32(R0), R5                                    ; 0974
                   54        55   D0 00097                    MOVL    R5, SUBVECTOR                                 ;
                   64        52   D0 0009A                    MOVL    COUNT, (SUBVECTOR)                            ; 0975
                             51   D4 0009D                    CLRL    I                                             ; 0976
                             04   11 0009F                    BRB     5$                                            ;
                6441         86   D0 000A1  4$:               MOVL    (PTR)+, (SUBVECTOR)[I]                        ; 0978
      F8           51        52   F3 000A5  5$:               AOBLEQ  COUNT, I, 4$                                  ; 0976
              18   A0        55   D0 000A9                    MOVL    R5, 24(R0)                                    ; 0985
              0C   BC        50   D0 000AD  6$:               MOVL    R0, @DESC2                                    ; 0986
                             1D   11 000B1                    BRB     8$                                            ; 0987
                        0C   A3   DD 000B3  7$:               PUSHL   12(R3)                                        ; 0992
      00000000G   00         01   FB 000B6                    CALLS   #1, DBG$STA_SYM_IS_LITERAL                    ;
                   13        50   E9 000BD                    BLBC    R0, 9$                                        ;
                        0C   AC   DD 000C0                    PUSHL   DESC2                                         ; 0999
```

```
               7E        7A   8F   9A 000C3          MOVZBL  #122, -(SP)                            ;
                              53   DD 000C7          PUSHL   R3                                     ;
        00000000G  00         03   FB 000C9          CALLS   #3, DBG$PRIM_TO_VAL                    ;
                            008F   31 000D0 8$:      BRW     14$                                    ; 1000
                              04   DD 000D3 9$:      PUSHL   #4                                     ; 1007
               7E        7A   8F   9A 000D5          MOVZBL  #122, -(SP)                            ;
               67             02   FB 000D9          CALLS   #2, DBG$MAKE_SKELETON_DESC             ;
          04   AE             50   D0 000DC          MOVL    R0, TEMP_DESC                          ;
               52        04   AE   D0 000E0          MOVL    TEMP_DESC, R2                          ; 1008
          03   A2        03   A3   90 000E4          MOVB    3(R3), 3(R2)                           ;
          06   A2        06   A3   B0 000E9          MOVW    6(R3), 6(R2)                           ; 1010
          08   A2        08   A3   D0 000EE          MOVL    8(R3), 8(R2)                           ; 1011
               54        14   A2   9E 000F3          MOVAB   20(R2), R4                             ; 1018
                              18   BB 000F7          PUSHR   #^M<R3,R4>                             ;
        00000000G  00         02   FB 000F9          CALLS   #2, DBG$MAKE_VMS_DESC                  ;
                   04         50   E8 00100          BLBS    R0, 10$                                ;
                   50         04   D0 00103          MOVL    #4, R0                                 ; 1020
                                   04 00106          RET                                            ;
                   0D    03   A4   91 00107 10$:     CMPB    3(R4), #13                             ; 1024
                              18   12 0010B          BNEQ    11$                                    ;
     50        1C  A2             08   C7 0010D      DIVL3   #8, 28(R2), R0                         ; 1028
               18  A2             50   C0 00112      ADDL2   R0, 24(R2)                             ;
7E   00        1C  A2             01   7A 00116      EMUL    #1, 28(R2), #0, -(SP)                  ; 1029
50   50        8E                 08   7B 0011C      EDIV    #8, (SP)+, R0, R0                      ;
          1C   A2             50   D0 00121          MOVL    R0, 28(R2)                             ;
                         08   AC   D5 00125 11$:     TSTL    DTYPE                                  ; 1036
                              2A   13 00128          BEQL    13$                                    ;
               08        08   AC   D1 0012A          CMPL    DTYPE, #8                              ; 1045
                              0F   12 0012E          BNEQ    12$                                    ;
          06   A2      0603   8F   B0 00130          MOVW    #1539, 6(R2)                           ; 1049
               64  01080004   8F   D0 00136          MOVL    #17301508, (R4)                        ; 1052
                              15   11 0013D          BRB     13$                                    ; 1045
                  00000000'   EF   9F 0013F 12$:     PUSHAB  P.AAA                                  ; 1058
                              01   DD 00145          PUSHL   #1                                     ;
                  00028362    8F   DD 00147          PUSHL   #164706                                ;
        00000000G  00         03   FB 0014D          CALLS   #3, LIB$SIGNAL                         ;
          20   A2        18   A2   7D 00154 13$:     MOVQ    24(R2), 32(R2)                         ; 1062
          18   A2        20   A2   9E 00159          MOVAB   32(R2), 24(R2)                         ; 1068
          0C   BC             52   D0 0015E          MOVL    R2, @DESC2                             ; 1069
               50             01   D0 00162 14$:     MOVL    #1, R0                                 ; 1070
                                   04 00165          RET                                            ; 1071

; Routine Size:  358 bytes,    Routine Base:  DBG$CODE + 056B
```

```
1 ROUTINE DETERMINE_TYPE (ARG, NEW_ARG, TYPE) : NOVALUE =
1 !
1 ! FUNCTION
1 !
1 !       Given a pointer to a descriptor, this routine does two things:
1 !       1 - Determines which of the address types the descriptor represents
1 !       2 - If necessary, calls CONV_TEXT_VALUE and returns a converted arg
1 !           in NEW_ARG.
1 !
1 ! INPUTS
1 !
1 !       ARG     - Points to either a Primary Descriptor or a Value Descriptor
1 !                 or a Volatile Value Descriptor
1 !       NEW_ARG - The address in which to leave a pointer to the converted
1 !                 descriptor.
1 !       TYPE    - The address in which to leave a type code
1 !
1 ! OUTPUTS
1 !
1 !       In TYPE, returns one of the five type codes ADDR$K_LITERAL, ADDR$K_INST,
1 !       ADDR$K_DATA, ADDR$K_BITFIELD, ADDR$K_PRIMARY, saying what kind of
1 !       address the descriptor represents.
1 !
1 !       In NEW_ARG, returns the address of the original argument if no conversion
1 !       was done on the descriptor. Otherwise, returns the address of the
1 !       new descriptor.
1 !
2    BEGIN
2
2    MAP
2        ARG : REF DBG$VALDESC;
2
2    LOCAL
2        DTYPE,                          ! Holds a dtype code
2        NAME: REF VECTOR[,BYTE],        ! Holds name of a primary
2        SYMID;                          ! Pointer to a SYMID
2
2    ! Turn value descriptors into volatile value descriptors.
2    !
2    IF .ARG [DBG$B_DHDR_TYPE] EQL DBG$K_VALUE_DESC
2    THEN
3        BEGIN
3
3        ! Check for "unconverted" value descriptors as input. These arise from
3        ! constants, e.g. on EXAMINE 1000 we get a value descriptor with
3        ! the string "1000".
3        !
3        IF .ARG [DBG$V_DHDR_UNCVT]
3        THEN
4            BEGIN
4
4            ! Check for the correct dtype.
4            !
4            SELECTONE .ARG [DBG$B_VALUE_DTYPE] OF
4                SET
4                [DSC$K_DTYPE_L]:
4                    ARG = DBG$CONV_TEXT_VALUE (.ARG, .ARG, 0);
```

```
                        ! The number scanner may pick up the constant as pack decimal
                        ! for some languages, take in pack decimal with no '.' and
                        ! treat it as long.
                        !
                        [DSC$K_DTYPE_P]:
                            BEGIN
                            IF NOT CH$FIND_CH(.ARG [DBG$W_VALUE_LENGTH],
                                              .ARG [DBG$L_VALUE_POINTER], %C'.')
                            THEN
                                BEGIN
                                ARG [DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_L;
                                ARG [DBG$W_VALUE_TOKENCODE] = TOKEN$K_INTEGER;
                                ARG = DBG$CONV_TEXT_VALUE (.ARG, .ARG, 0);
                                END
                            ELSE
                                SIGNAL (DBG$_ILLADDCON, 2, .ARG [DBG$W_VALUE_LENGTH],
                                        .ARG [DBG$L_VALUE_POINTER]);

                            END;

                        [OTHERWISE]:
                            SIGNAL (DBG$_ILLADDCON, 2, .ARG [DBG$W_VALUE_LENGTH],
                                    .ARG [DBG$L_VALUE_POINTER]);
                        TES;

                    ! Fill in correct dtype and length information.
                    !
                    ARG [DBG$V_DHDR_LITERAL] = TRUE;
                    ARG [DBG$V_DHDR_OVERRIDE] = TRUE;
                    ARG [DBG$B_VALUE_CLASS] = 0;
                    ARG [DBG$B_VALUE_DTYPE] = .DBG$GL_DFLTTYP;
                    IF .DBG$GL_DFLTTYP EQL DSC$K_DTYPE_ZI
                    THEN
                        ARG[DBG$W_VALUE_LENGTH] =
                            DBG$INS_DECODE(..ARG[DBG$L_VALUE_POINTER],
                                           FALSE, FALSE) -
                          ..ARG[DBG$L_VALUE_POINTER]
                    ELSE
                        ARG [DBG$W_VALUE_LENGTH] = .DBG$GW_DFLTLENG;
                    END;

                ! Make the value descriptor into a volatile value descriptor.
                !
                ARG [DBG$B_DHDR_TYPE] = DBG$K_V_VALUE_DESC;
                ARG [DBG$L_VALUE_POINTER] = .ARG [DBG$L_VALUE_VALUE0];
                IF .ARG [DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_V
                OR .ARG [DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_SV
                OR .ARG [DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_VU
                OR .ARG [DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_SVU
                THEN
                    ARG [DBG$L_VALUE_POS] = .ARG [DBG$L_VALUE_VALUE1];
                END;

            ! Case on the kind of descriptor.
            !
```

```
2        SELECTONE .ARG[DBG$B_DHDR_TYPE] OF
2            SET
2
2            ! Primary Descriptors.
2            !
2            [DBG$K_PRIMARY_DESC] :
3                BEGIN
3
3                ! Check for Primary representing a literal.
3                !   Example: In PASCAL,
3                !   CONST
3                !       X = 512;
3                !       Y = 1.1;
3                !
3                !   On EXAMINE X, we will construct a Value Descriptor with 512 in
3                !   the value field. On EXAMINE Y, we will signal an error.
3                !
3                IF DBG$STA_SYM_IS_LITERAL (.ARG [DBG$L_DHDR_SYMID0])
3                THEN
4                    BEGIN
4
4                    ! Save the symid
4                    !
4                    SYMID = .ARG [DBG$L_DHDR_SYMID0];
4
4                    ! Convert the Primary into a Value Descriptor and obtain the dtype.
4                    !
4                    DBG$PRIM_TO_VAL (.ARG, DBG$K_VALUE_DESC, ARG);
4                    DTYPE = .ARG[DBG$B_VALUE_DTYPE];
4
4                    ! Check for integer dtype. These are the only kinds of
4                    ! literals we can do address arithmetic on.
4                    !
4                    IF  .DTYPE NEQ DSC$K_DTYPE_Z
4                    AND .DTYPE NEQ DSC$K_DTYPE_L
4                    AND .DTYPE NEQ DSC$K_DTYPE_LU
4                    AND .DTYPE NEQ DSC$K_DTYPE_W
4                    AND .DTYPE NEQ DSC$K_DTYPE_WU
4                    AND .DTYPE NEQ DSC$K_DTYPE_B
4                    AND .DTYPE NEQ DSC$K_DTYPE_BU
4                    THEN
5                        BEGIN
5
5                        ! Call the routine that turns a symid into a name, in order
5                        ! to be able to signal the error.
5                        !
5                        DBG$STA_SYMNAME (.SYMID, NAME);
5                        SIGNAL (DBG$_ILLADDCON, 2, .NAME[0], NAME[1]);
5                        END;
4
4                    ! Make the value descriptor into a volatile value descriptor.
4                    !
4                    ARG [DBG$B_DHDR_TYPE] = DBG$K_V_VALUE_DESC;
4                    ARG [DBG$L_VALUE_POINTER] = .ARG [DBG$L_VALUE_VALUE0];
4                    IF .ARG [DBG$B_VALUE_CLASS] EQL DSC$K_CLASS_UBS
4                    THEN
4                        ARG [DBG$L_VALUE_POS] = .ARG [DBG$L_VALUE_VALUE1];
```

```
4
4                       ! Put in the flag for literal.
4                       !
4                       ARG [DBG$V_DHDR_LITERAL] = TRUE;
4                       .TYPE = ADDR$K_LITERAL;
4                       END
4
3                   ELSE
3
3                       ! The value is not a literal. Set the type code to Primary
3                       !
3                       .TYPE = ADDR$K_PRIMARY;
3
2                   END;
2
2               ! Volatile Value Descriptors.
2               !
2               [DBG$K_V_VALUE_DESC] :
3                   BEGIN
3
3                   ! If the descriptor came from a literal constant, fill in
3                   ! dtype of literal.
3                   !
3                   IF .ARG [DBG$V_DHDR_LITERAL]
3                   THEN
3                       .TYPE = ADDR$K_LITERAL
3
3                   ! Else look at the dtype field.
3                   !
3                   ELSE
3                       SELECTONE .ARG [DBG$B_VALUE_DTYPE] OF
3                           SET
3
3                           [DSC$K_DTYPE_ZI, DSC$K_DTYPE_ZEM] :
4                               BEGIN
4
4                               ! Change type from entry mask to instruction if
4                               ! any operations are to be performed.
4                               !
4                               ARG [DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_ZI;
4                              .TYPE = ADDR$K_INST;
3                               END;
3
3                           [DSC$K_DTYPE_VU, DSC$K_DTYPE_V,
3                            DSC$K_DTYPE_SVU, DSC$K_DTYPE_SV] :
3                               .TYPE = ADDR$K_BITFIELD;
3
3                           [OTHERWISE] :
3                               .TYPE = ADDR$K_DATA;
3
3                           TES;
3
3                   ! For non-bitfield types, make sure the POS field is zero.
3                   !
3                   IF ..TYPE NEQ ADDR$K_BITFIELD
3                   THEN
3                       ARG [DBG$L_VALUE_POS] = 0;
```

```
; 1170          1300  3                END;
; 1171          1301  2
; 1172          1302  2            ! We do not expect any other kind of descriptor.
; 1173          1303  2            !
; 1174          1304  2            [OTHERWISE] :
; 1175          1305  2                $DBG_ERROR ('DBGADDEXP\DETERMINE_TYPE unknown arg type');
; 1176          1306  2
; 1177          1307  2            TES;
; 1178          1308  2
; 1179          1309  2        ! Fill in the output parameter and return.
; 1180          1310  2        !
; 1181          1311  2        .NEW_ARG = .ARG;
; 1182          1312  2        RETURN;
; 1183          1313  2        END;
; 1184          1314  1


                                                               .PSECT  DBG$PLIT,NOWRT,  SHR,  PIC,0

45 54 45 44 5C 50 58 45 44 44 41 47 42 44 29 00029 P.AAB:     .ASCII  \)DBGADDEXP\<92>\DETERMINE_TYPE unknown \    ;
6E 6B 6E 75 20 45 50 59 54 5F 45 4E 49 4D 52 00038                                                                  ;
                                     20 6E 77 6F 00047                                                               ;
                         65 70 79 74 20 67 72 61 0004B         .ASCII  \arg type\                                    ;


                                                               .PSECT  DBG$CODE,NOWRT,  SHR,  PIC,0

                                          001C 00000 DETERMINE_TYPE:
                                                               .WORD   Save R2,R3,R4                              ; 1072
                    54 00000000G     00    9E 00002            MOVAB   LIB$SIGNAL, R4                             ;
                    5E               04    C2 00009            SUBL2   #4, SP                                     ;
                    52         04    AC    D0 0000C            MOVL    ARG, R2                                    ; 1111
             7A     8F         02    A2    91 00010            CMPB    2(R2), #122                                ;
                                     03    13 00015            BEQL    1$                                         ;
                                   00AD    31 00017            BRW     9$                                         ;
        7D          04         A2    05    E1 0001A 1$:        BBC     #5, 4(R2), 7$                              ; 1119
                    50         16    A2    9A 0001F            MOVZBL  22(R2), R0                                 ; 1125
                    08               50    91 00023            CMPB    R0, #8                                     ; 1127
                                     1A    13 00026            BEQL    3$                                         ;
                    15               50    91 00028            CMPB    R0, #21                                    ; 1134
                                     28    12 0002B            BNEQ    4$                                         ;
    18     B2       14         A2    2E    3A 0002D            LOCC    #46, 20(R2), @24(R2)                       ; 1136
                                     02    12 00033            BNEQ    2$                                         ;
                                     51    D4 00035            CLRL    R1                                         ;
                    1B               51    E8 00037 2$:        BLBS    R1, 4$                                     ;
             16     A2               08    90 0003A            MOVB    #8, 22(R2)                                 ; 1140
             10     A2               04    B0 0003E            MOVW    #4, 16(R2)                                 ; 1141
                                     7E    D4 00042 3$:        CLRL    -(SP)                                      ; 1142
                                     52    DD 00044            PUSHL   R2                                         ;
                                     52    DD 00046            PUSHL   R2                                         ;
        00000000G   00               03    FB 00048            CALLS   #3, DBG$CONV_TEXT_VALUE                    ;
             04     AC               50    D0 0004F            MOVL    R0, ARG                                    ;
                                     12    11 00053            BRB     5$                                         ; 1136
                               18    A2    DD 00055 4$:        PUSHL   24(R2)                                     ; 1152
                    7E         14    A2    3C 00058            MOVZWL  20(R2), -(SP)                              ; 1151
```

```
                                  02  DD 0005C         PUSHL   #2                                ;
                      00028EF8    8F  DD 0005E         PUSHL   #167672                           ;
                 64               04  FB 00064         CALLS   #4, LIB$SIGNAL                    ;
                 52         04    AC  D0 00067  5$:    MOVL    ARG, R2                           ; 1157
            04   A2         C0    8F  88 0006B         BISB2   #192, 4(R2)                       ; 1158
                 50  00000000G    00  D0 00070         MOVL    DBG$GL_DFLTTYP, R0                ; 1160
            16   A2               50  9B 00077         MOVZBW  R0, 22(R2)                        ;
                 16               50  D1 0007B         CMPL    R0, #22                           ; 1161
                                  14  12 0007E         BNEQ    6$                                ;
                                  7E  7C 00080         CLRQ    -(SP)                             ; 1164
                            18    B2  DD 00082         PUSHL   @24(R2)                           ;
       00000000G   00             03  FB 00085         CALLS   #3, DBG$INS_DECODE                ;
 14   A2           50       18    B2  A3 0008C         SUBW3   @24(R2), R0, 20(R2)               ; 1166
                                  08  11 00092         BRB     7$                                ; 1163
            14   A2  00000000G    00  B0 00094  6$:    MOVW    DBG$GW_DFLTLENG, 20(R2)           ; 1168
                 50         04    AC  D0 0009C  7$:    MOVL    ARG, R0                           ; 1173
            02   A0         83    8F  90 000A0         MOVB    #-125, 2(R0)                      ;
            18   A0         20    A0  D0 000A5         MOVL    32(R0), 24(R0)                    ; 1174
                 01         16    A0  91 000AA         CMPB    22(R0), #1                        ; 1175
                                  12  13 000AE         BEQL    8$                                ;
                 29         16    A0  91 000B0         CMPB    22(R0), #41                       ; 1176
                                  0C  13 000B4         BEQL    8$                                ;
                 22         16    A0  91 000B6         CMPB    22(R0), #34                       ; 1177
                                  06  13 000BA         BEQL    8$                                ;
                 2A         16    A0  91 000BC         CMPB    22(R0), #42                       ; 1178
                                  05  12 000C0         BNEQ    9$                                ;
            1C   A0         24    A0  D0 000C2  8$:    MOVL    36(R0), 28(R0)                    ; 1180
                 52         04    AC  D0 000C7  9$:    MOVL    ARG, R2                           ; 1186
            79   8F         02    A2  91 000CB         CMPB    2(R2), #121                       ; 1191
                                  03  13 000D0         BEQL    10$                               ;
                                008D  31 000D2         BRW     14$                               ;
                            0C    A2  DD 000D5  10$:   PUSHL   12(R2)                            ; 1203
       00000000G   00             01  FB 000D8         CALLS   #1, DBG$STA_SYM_IS_LITERAL        ;
                   7A             50  E9 000DF         BLBC    R0, 13$                           ;
                   53       0C    A2  D0 000E2         MOVL    12(R2), SYMID                     ; 1209
                            04    AC  9F 000E6         PUSHAB  ARG                               ; 1213
                   7E       7A    8F  9A 000E9         MOVZBL  #122, -(SP)                       ;
                                  52  DD 000ED         PUSHL   R2                                ;
       00000000G   00             03  FB 000EF         CALLS   #3, DBG$PRIM_TO_VAL               ;
                   52       04    AC  D0 000F6         MOVL    ARG, R2                           ; 1214
                   50       16    A2  9A 000FA         MOVZBL  22(R2), DTYPE                     ;
                                  3C  13 000FE         BEQL    11$                               ; 1219
                   08             50  D1 00100         CMPL    DTYPE, #8                         ; 1220
                                  37  13 00103         BEQL    11$                               ;
                   04             50  D1 00105         CMPL    DTYPE, #4                         ; 1221
                                  32  13 00108         BEQL    11$                               ;
                   07             50  D1 0010A         CMPL    DTYPE, #7                         ; 1222
                                  2D  13 0010D         BEQL    11$                               ;
                   03             50  D1 0010F         CMPL    DTYPE, #3                         ; 1223
                                  28  13 00112         BEQL    11$                               ;
                   06             50  D1 00114         CMPL    DTYPE, #6                         ; 1224
                                  23  13 00117         BEQL    11$                               ;
                   02             50  D1 00119         CMPL    DTYPE, #2                         ; 1225
                                  1E  13 0011C         BEQL    11$                               ;
                          4008    8F  BB 0011E         PUSHR   #^M<R3,SP>                        ; 1232
       00000000G   00             02  FB 00122         CALLS   #2, DBG$STA_SYMNAME               ;
      7E           6E             01  C1 00129         ADDL3   #1, NAME, -(SP)                   ; 1233
```

```
          7E        04   BE  9A 0012D          MOVZBL  @NAME, -(SP)                        ;
                         02  DD 00131          PUSHL   #2                                  ;
              00028EF8   8F  DD 00133          PUSHL   #167672                             ;
          64             04  FB 00139          CALLS   #4, LIB$SIGNAL                      ;
     02   A2        83   8F  90 0013C  11$:    MOVB    #-125, 2(R2)                        ; 1238
     18   A2        20   A2  D0 00141          MOVL    32(R2), 24(R2)                      ; 1239
     0D             17   A2  91 00146          CMPB    23(R2), #13                         ; 1240
                         05  12 0014A          BNEQ    12$                                 ;
     1C   A2        24   A2  D0 0014C          MOVL    36(R2), 28(R2)                      ; 1242
     04   A2        40   8F  88 00151  12$:    BISB2   #64, 4(R2)                          ; 1246
     0C   BC             01  D0 00156          MOVL    #1, @TYPE                           ; 1247
                         6A  11 0015A          BRB     21$                                 ; 1203
     0C   BC             02  D0 0015C  13$:    MOVL    #2, @TYPE                           ; 1254
                         64  11 00160          BRB     21$                                 ; 1186
     83   8F        02   A2  91 00162  14$:    CMPB    2(R2), #131                         ; 1260
                         4C  12 00167          BNEQ    20$                                 ;
06   04   A2             06  E1 00169          BBC     #6, 4(R2), 15$                      ; 1266
     0C   BC             01  D0 0016E          MOVL    #1, @TYPE                           ; 1268
                         36  11 00172          BRB     19$                                 ;
          50        16   A2  9A 00174  15$:    MOVZBL  22(R2), R0                          ; 1273
          16             50  91 00178          CMPB    R0, #22                             ; 1276
                         0F  1F 0017B          BLSSU   16$                                 ;
          17             50  91 0017D          CMPB    R0, #23                             ;
                         0A  1A 00180          BGTRU   16$                                 ;
     16   A2             16  90 00182          MOVB    #22, 22(R2)                         ; 1282
     0C   BC             03  D0 00186          MOVL    #3, @TYPE                           ; 1283
                         1E  11 0018A          BRB     19$                                 ; 1273
          01             50  91 0018C  16$:    CMPB    R0, #1                              ; 1286
                         0F  13 0018F          BEQL    17$                                 ;
          22             50  91 00191          CMPB    R0, #34                             ;
                         0A  13 00194          BEQL    17$                                 ;
          29             50  91 00196          CMPB    R0, #41                             ;
                         0B  1F 00199          BLSSU   18$                                 ;
          2A             50  91 0019B          CMPB    R0, #42                             ;
                         06  1A 0019E          BGTRU   18$                                 ;
     0C   BC             05  D0 001A0  17$:    MOVL    #5, @TYPE                           ; 1288
                         04  11 001A4          BRB     19$                                 ;
     0C   BC             04  D0 001A6  18$:    MOVL    #4, @TYPE                           ; 1291
          05        0C   BC  D1 001AA  19$:    CMPL    @TYPE, #5                           ; 1297
                         16  13 001AE          BEQL    21$                                 ;
                    1C   A2  D4 001B0          CLRL    28(R2)                              ; 1299
                         11  11 001B3          BRB     21$                                 ; 1186
              00000000'  EF  9F 001B5  20$:    PUSHAB  P.AAB                               ; 1306
                         01  DD 001BB          PUSHL   #1                                  ;
              00028362   8F  DD 001BD          PUSHL   #164706                             ;
          64             03  FB 001C3          CALLS   #3, LIB$SIGNAL                      ;
     08   BC        04   AC  D0 001C6  21$:    MOVL    ARG, @NEW_ARG                       ; 1312
                             04 001CB          RET                                         ; 1314
```

; Routine Size: 460 bytes,   Routine Base: DBG$CODE + 06D1

```
1 ROUTINE GET_DEREFERENCE (PRIMPTR) =
1 !
1 ! FUNCTION
1 !       This routine is called upon seeing the dereference operator
1 !       in an address expression, when the operand is a primary.
1 !       E.g., "EXAM *PTR" in language C.
1 !
1 !       If the object being dereferenced is a typed pointer then
1 !       this routine dereferences it by modifying the Primary
1 !       Descriptor to refer to the pointed-to object. The value
1 !       "TRUE" is then returned indicating that the routine
1 !       was successful.
1 !
1 !       This routine is similar to the GET_DEREFERENCE routine
1 !       in DBGPARSER.
1 !
1 ! INPUTS
1 !       PRIMPTR         - A pointer to the Primary Descriptor
1 !                         being dereferenced.
1 !
1 ! OUTPUTS
1 !       If the Primary represents a typed pointer, then
1 !       it is modified and TRUE is returned.
1 !
1 !       If the Primary does not represent a typed pointer then
1 !       it is not modified and FALSE is returned.
1 !
2     BEGIN
2     MAP
2         PRIMPTR: REF DBG$PRIMARY;
2
2     LOCAL
2         FCODE,                          ! Local variable holding fcode info
2         JUNK,                           ! Dummy output parameter
2         NODEPTR: REF DBG$PRIM_NODE,     ! Points to a Primary Sub-node
2         TYPEID;                         ! Pointer to a RST type entry
2
2
2     ! Check that the object being dereferenced is actually a pointer.
2     !
2     IF  .PRIMPTR[DBG$B_DHDR_FCODE] NEQ RST$K_TYPE_TPTR
2     THEN
2         RETURN FALSE;
2
2
2     ! Obtain a pointer to the bottom level sub-node by following th
2     ! back-pointer. Light the EVAL bit in this subnode,
2     ! which indicates that pointer dereferencing is
2     ! taking place.
2     ! Then, obtain the pointer to the RST type entry for the object
2     ! dereferenced.
2     !
2     NODEPTR = .PRIMPTR [DBG$L_PRIM_BLINK];
2     NODEPTR [DBG$V_PNODE_EVAL] = TRUE;
2     TYPEID = .NODEPTR [DBG$L_PNODE_TYPEID];
2
2
```

```
; 1243          1372  2      ! From this typeid, get the typeid for the object being pointed to.
; 1244          1373  2      ! For pointer variables, use the routine that extracts the typeid
; 1245          1374  2      ! of the pointed-to object.
; 1246          1375  2      ! Then obtain the fcode from the typeid.
; 1247          1376  2      !
; 1248          1377  2      DBG$STA_TYP_TYPEDPTR (.TYPEID, TYPEID);
; 1249          1378  2      FCODE = DBG$STA_TYPEFCODE (.TYPEID);
; 1250          1379  2      DBG$BUILD_PRIMARY_SUBNODE (.PRIMPTR, RST$K_DATA, 0, .FCODE, .TYPEID, 0);
; 1251          1380  2      RETURN TRUE;
; 1252          1381  1      END;



                                        0004 00000 GET_DEREFERENCE:
                                                         .WORD   Save R2                                  ; 1315
                      5E            04   C2 00002        SUBL2   #4, SP
                      52       04   AC   D0 00005        MOVL    PRIMPTR, R2                              ; 1355
                      06       06   A2   91 00009        CMPB    6(R2), #6                                ;
                                    38   12 0000D        BNEQ    1$
                      50       18   A2   D0 0000F        MOVL    24(R2), NODEPTR                          ; 1367
               0A     A0            01   88 00013        BISB2   #1, 10(NODEPTR)                          ; 1368
                      6E       0C   A0   D0 00017        MOVL    12(NODEPTR), TYPEID                      ; 1369
                                    5E   DD 0001B        PUSHL   SP                                       ; 1377
                               04   AE   DD 0001D        PUSHL   TYPEID                                   ;
        00000000G     00            02   FB 00020        CALLS   #2, DBG$STA_TYP_TYPEDPTR                 ;
                                    6E   DD 00027        PUSHL   TYPEID                                   ; 1378
        00000000G     00            01   FB 00029        CALLS   #1, DBG$STA_TYPEFCODE                    ;
                                    7E   D4 00030        CLRL    -(SP)                                    ; 1379
                               04   AE   DD 00032        PUSHL   TYPEID                                   ;
                                    50   DD 00035        PUSHL   FCODE                                    ;
                      7E            06   7D 00037        MOVQ    #6, -(SP)                                ;
                                    52   DD 0003A        PUSHL   R2                                       ;
        00000000G     00            06   FB 0003C        CALLS   #6, DBG$BUILD_PRIMARY_SUBNODE            ;
                      50            01   D0 00043        MOVL    #1, R0                                   ; 1380
                                         04 00046        RET                                              ;
                                    50   D4 00047 1$:    CLRL    R0                                       ; 1381
                                         04 00049        RET                                              ;

; Routine Size:  74 bytes,    Routine Base:  DBG$CODE + 089D


; 1253          1382  1 END
; 1254          1383  0 ELUDOM



                                                         .EXTRN  LIB$SIGNAL

;                               PSECT SUMMARY
;
;       Name                  Bytes                    Attributes
;
;   DBG$CODE                   2279  NOVEC,NOWRT,  RD ,  EXE,  SHR,  LCL,  REL,  CON,  PIC,ALIGN(0)
;   DBG$PLIT                     83  NOVEC,NOWRT,  RD ,  EXE,  SHR,  LCL,  REL,  CON,  PIC,ALIGN(0)
```

```
;                          Library Statistics
;
;                                          --------- Symbols --------      Pages       Processing
;       File                               Total    Loaded    Percent      Mapped      Time
;
;  _$255$DUA28:[SYSLIB]LIB.L32;1           18619       17          0        1000        00:01.8
;  _$255$DUA28:[DEBUG.OBJ]STRUCDEF.L32;1      32        0          0           7        00:00.1
;  _$255$DUA28:[DEBUG.OBJ]DBGLIB.L32;1      1545      173         11          97        00:01.9
;  _$255$DUA28:[DEBUG.OBJ]DSTRECRDS.L32;1
;                                            418      105         25          31        00:00.4
;  _$255$DUA28:[DEBUG.OBJ]DBGMSG.L32;1       386        5          1          22        00:00.3



;                                  COMMAND QUALIFIERS

;       BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS$:DBGADDEXP/OBJ=OBJ$:DBGADDEXP MSRC$:DBGADDEXP/UPDATE=(ENH$:DBGADDEXP)

; Size:          2279 code + 83 data bytes
; Run Time:         00:47.7
; Elapsed Time:     02:41.5
; Lines/CPU Min:     1738
; Lexemes/CPU-Min: 17509
; Memory Used:  515 pages
; Compilation Complete
```

0077 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

STRUCDEF
REQ

TEMPREQ
REQ

DBGADDEXP
LIS

DBGATSIGN
LIS

SSIDEF
REQ