```
DDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBB      UUU         UUU     GGGGGGGGGGG
DDDDDDDDDDDD    EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUU         UUU     GGGGGGGGGGGG
DDDDDDDDDDDD    EEEEEEEEEEEEEEE  BBBBBBBBBBBB     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 EEEEEEEEEEE      BBBBBBBBBBBB     UUU         UUU GGG
DDD         DDD EEEEEEEEEEE      BBBBBBBBBBBB     UUU         UUU GGG
DDD         DDD EEEEEEEEEEE      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      UUUUUUUUUUUUUUUUU     GGGGGGGGG
DDDDDDDDDDDD    EEEEEEEEEEEEEEE  BBBBBBBBBBBB      UUUUUUUUUUUUUUUUU     GGGGGGGGG
DDDDDDDDDDDD    EEEEEEEEEEEEEEE  BBBBBBBBBBBB      UUUUUUUUUUUUUUUUU     GGGGGGGGG
```

```
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
```

....
....
....
....

```
;     1          0001  0 MODULE DBGADDEXP (IDENT = 'V04-000') =
;     2          0002  0
;     3          0003  1 BEGIN
;     4          0004  1
;     5          0005  1 !**************************************************************************
;     6          0006  1 !*                                                                        *
;     7          0007  1 !*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                              *
;     8          0008  1 !*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                *
;     9          0009  1 !*  ALL RIGHTS RESERVED.                                                  *
;    10          0010  1 !*                                                                        *
;    11          0011  1 !*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED  *
;    12          0012  1 !*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE  *
;    13          0013  1 !*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
;    14          0014  1 !*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
;    15          0015  1 !*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
;    16          0016  1 !*  TRANSFERRED.                                                           *
;    17          0017  1 !*                                                                        *
;    18          0018  1 !*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
;    19          0019  1 !*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
;    20          0020  1 !*  CORPORATION.                                                           *
;    21          0021  1 !*                                                                        *
;    22          0022  1 !*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS  *
;    23          0023  1 !*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                *
;    24          0024  1 !*                                                                        *
;    25          0025  1 !*                                                                        *
;    26          0026  1 !**************************************************************************
;    27          0027  1 !
;    28          0028  1 ! WRITTEN BY
;    29          0029  1 !       Rich Title      August, 1982
;    30          0030  1 !
;    31          0031  1 ! MODULE FUNCTION
;    32          0032  1 !       This module contains the Address Expression Interpreter.
;    33          0033  1 !
;    34          0034  1
;    35          0035  1 REQUIRE 'SRC$:DBGPROLOG.REQ';
;    36          0169  1
;    37          0170  1 FORWARD ROUTINE
;    38          0171  1     DBG$EVAL_ADDR_OPERATOR,             ! Evaluate an Address Expr. operator
;    39          0172  1     DBG$PRIM_TO_ADDR,                   ! Convert Primary Descriptor to
;    40          0173  1                                         !    Value Descriptor containing
;    41          0174  1                                         !    address of descriptor.
;    42          0175  1     DETERMINE_TYPE: NOVALUE,            ! Determine type of inputs
;    43          0176  1     GET_DEREFERENCE;                    ! Perform dereference operation
;    44          0177  1
;    45          0178  1 EXTERNAL
;    46          0179  1     DBG$GL_DFLTTYP,                     ! Holds type from SET TYPE command
;    47          0180  1     DBG$GW_DFLTLENG: WORD,              ! Holds length from SET TYPE command
;    48          0181  1     DBG$REG_VALUES: VECTOR[,LONG];      ! Register save area
;    49          0182  1
;    50          0183  1 EXTERNAL ROUTINE
;    51          0184  1     DBG$BUILD_PRIMARY_SUBNODE: NOVALUE, ! Build a Primary Subnode
;    52          0185  1     DBG$CONV_TEXT_VALUE,                ! Convert text string to value
;    53          0186  1     DBG$INS_DECODE,                     ! Decode instruction
;    54          0187  1     DBG$MAKE_VAL_DESC,                  ! Materialize value into Val Descr
;    55          0188  1     DBG$MAKE_VMS_DESC,                  ! Convert Primary Descriptor to
;    56          0189  1                                         !    VAX standard descriptor
;    57          0190  1     DBG$MAKE_SKELETON_DESC,             ! Build skeleton descriptor
```

```
;    58          0191  1     DBG$NCOPY_DESC,                   ! Copy descriptors
;    59          0192  1     DBG$PRIM_TO_VAL,                  ! Convert Primary Descriptor to
;    60          0193  1                                       !    Value Descriptor.
;    61          0194  1     DBG$STA_SYMNAME : NOVALUE,        ! Obtain name of symbol from SYMID
;    62          0195  1     DBG$STA_SYM_IS_LITERAL,           ! Determine whether a symid represents
;    63          0196  1                                       !    a literal value.
;    64          0197  1     DBG$STA_TYPEFCODE,                ! Find fcode
;    65          0198  1     DBG$STA_TYP_TYPEDPTR: NOVALUE;    ! Look up typed pointer
;    66          0199  1
;    67          0200  1 ! Define some codes for the kinds of addresses that a descriptor
;    68          0201  1 ! can represent.
;    69          0202  1 !
;    70          0203  1 LITERAL
;    71          0204  1     ADDR$K_UNKNOWN = -1,          ! Unknown type
;    72          0205  1     ADDR$K_MINTYPE = 1,           ! Minimum of known type codes below
;    73          0206  1     ADDR$K_LITERAL = 1,           ! Literal value
;    74          0207  1     ADDR$K_PRIMARY = 2,           ! Primary Descriptor
;    75          0208  1     ADDR$K_INST = 3,              ! Address of instruction
;    76          0209  1     ADDR$K_DATA = 4,              ! Address of data
;    77          0210  1     ADDR$K_BITFIELD = 5,          ! Bit field within address
;    78          0211  1     ADDR$K_MAXTYPE = 5;           ! Maximum of codes above
;    79          0212  1
```

```
;    81          0213  1 GLOBAL ROUTINE DBG$EVAL_ADDR_OPERATOR(OPERATOR, LEFT_ARG, RIGHT_ARG) =
;    82          0214  1 !
;    83          0215  1 ! FUNCTION
;    84          0216  1 !       This routine does the actual evaluation of a DEBUG Address Expression
;    85          0217  1 !       operator.  It does a CASE on the operator code and does the appropriate
;    86          0218  1 !       address computation or other operation for those operators allowed in
;    87          0219  1 !       DEBUG Address Expressions.  For any operator not allowed in an Address
;    88          0220  1 !       Expression, it signals an error message.
;    89          0221  1 !
;    90          0222  1 ! INPUTS
;    91          0223  1 !       OPERATOR - The Operator Token Entry for the operator to be evaluated.
;    92          0224  1 !
;    93          0225  1 !       LEFT_ARG - A pointer to the left argument Primary Descriptor or Value
;    94          0226  1 !                  Descriptor.  If the operator is a unary operator, LEFT_ARG
;    95          0227  1 !                  points to the operator's one argument.
;    96          0228  1 !
;    97          0229  1 !       RIGHT_ARG - A pointer to the right argument Primary Descriptor or Value
;    98          0230  1 !                  Descriptor.  If the operator is a unary operator, RIGHT_ARG
;    99          0231  1 !                  is not used.
;   100          0232  1 !
;   101          0233  1 ! OUTPUTS
;   102          0234  1 !       A pointer to the Value Descriptor or Primary Descriptor
;   103          0235  1 !       which results from the evaluation of
;   104          0236  1 !       the operator is returned as this routine's result.
;   105          0237  1 !
;   106          0238  1
;   107          0239  2     BEGIN
;   108          0240  2
;   109          0241  2     MAP
;   110          0242  2         OPERATOR: REF TOKEN$ENTRY,      ! Token Entry for operator to perform
;   111          0243  2         LEFT_ARG: REF DBG$VALDESC,      ! Left operand Token Entry
;   112          0244  2         RIGHT_ARG: REF DBG$VALDESC;     ! Right operand Token Entry
;   113          0245  2
;   114          0246  2     LOCAL
;   115          0247  2         BINARY_FLAG,                    ! TRUE if the Address Expression Operator
;   116          0248  2                                         !    is a binary operator
;   117          0249  2         LEFT_TYPE,                      ! Address type of left operand (one of
;   118          0250  2                                         !    the above codes
;   119          0251  2         OPCODE,                         ! The opcode of the Address Expression Operator
;   120          0252  2         RIGHT_TYPE;                     ! Address type of right operand (one
;   121          0253  2                                         !    of the above codes.
;   122          0254  2
;   123          0255  2
;   124          0256  2     ! The following macro containing the processing that is done on
;   125          0257  2     ! the ADD and SUBTRACT operators. This processing is the same
;   126          0258  2     ! except for occurences of either a "+" or a "-". For that reason,
;   127          0259  2     ! it was separated out as a macro.
;   128          0260  2     !
;   129        M 0261  2     MACRO PROCESS_ADD_OR_SUBTRACT (OP) =
;   130        M 0262  2
;   131        M 0263  2             CASE .LEFT_TYPE FROM ADDR$K_MINTYPE TO ADDR$K_MAXTYPE OF
;   132        M 0264  2                 SET
;   133        M 0265  2
;   134        M 0266  2                 ! Left argument is a Primary Descriptor.
;   135        M 0267  2                 !
;   136        M 0268  2                 [ADDR$K_PRIMARY] :
;   137        M 0269  2                     BEGIN
```

```
;  138          M 0270  2
;  139          M 0271  2                    ! For all right args, we need to convert
;  140          M 0272  2                    ! the Primary Descriptor to Value Descriptor before we
;  141          M 0273  2                    ! do any operation.
;  142          M 0274  2                    !
;  143          M 0275  2                    DBG$PRIM_TO_VAL (.LEFT_ARG, DBG$K_V_VALUE_DESC, LEFT_ARG);
;  144          M 0276  2                    RETURN DBG$EVAL_ADDR_OPERATOR (
;  145          M 0277  2                                                   .OPERATOR,
;  146          M 0278  2                                                   .LEFT_ARG,
;  147          M 0279  2                                                   .RIGHT_ARG );
;  148          M 0280  2                    END;
;  149          M 0281  2
;  150          M 028?  2
;  151          M 0283  2                [ADDR$K_LITERAL] :
;  152          M 0284  2                    BEGIN
;  153          M 0285  2
;  154          M 028?  2                    IF .RIGHT_TYPE EQL ADDR$K_PRIMARY
;  155          M 0287  2                    THEN
;  156          M 0288  2                        BEGIN
;  157          M 0289  2                        DBG$PRIM_TO_VAL (.RIGHT_ARG, DBG$K_V_VALUE_DESC, RIGHT_ARG);
;  158          M 0290  2                        RETURN DBG$EVAL_ADDR_OPERATOR (
;  159          M 0291  2                                .OPERATOR,
;  160          M 0292  2                                .LEFT_ARG,
;  161          M 0293  2                                .RIGHT_ARG);
;  162          M 0294  2                        END;
;  163          M 0295  2
;  164          M 0296  2                    ! For all other right operands, just add the literal
;  165          M 0297  2                    ! value to the right operand, and return the right
;  166          M 0298  2                    ! operand. Thus the result retains the type of the
;  167          M 0299  2                    ! right operand.
;  168          M 0300  2                    !
;  169          M 0301  2                    RIGHT_ARG [DBG$L_VALUE_POINTER] =
;  170          M 0302  2                        .LEFT_ARG [DBG$L_VALUE_POINTER] OP
;  171          M 0303  2                        .RIGHT_ARG [DBG$L_VALUE_POINTER];
;  172          M 0304  2
;  173          M 0305  2                    ! Handle result type of instruction.
;  174          M 0306  2                    !
;  175          M 0307  2                    IF .RIGHT_TYPE EQL ADDR$K_INST
;  176          M 0308  2                    THEN
;  177          M 0309  2                        RIGHT_ARG[DBG$W_VALUE_LENGTH] =
;  178          M 0310  2                            DBG$INS_DECODE(.RIGHT_ARG[DBG$L_VALUE_POINTER],
;  179          M 0311  2                                           FALSE, FALSE) -
;  180          M 0312  2                            .RIGHT_ARG[DBG$L_VALUE_POINTER];
;  181          M 0313  2
;  182          M 0314  2                    ! Handle bitfields on the right.
;  183          M 0315  2                    !
;  184          M 0316  2                    IF .RIGHT_TYPE EQL ADDR$K_BITFIELD
;  185          M 0317  2                    THEN
;  186          M 0318  2                        BEGIN
;  187          M 0319  2                        RIGHT_ARG [DBG$L_VALUE_POS] =
;  188          M 0320  2                            .LEFT_ARG [DBG$L_VALUE_POS] OP
;  189          M 0321  2                            .RIGHT_ARG [DBG$L_VALUE_POS];
;  190          M 0322  2
;  191          M 0323  2                        ! Normalize the bit offset.
;  192          M 0324  2                        !
;  193          M 0325  2                        RIGHT_ARG [DBG$L_VALUE_POINTER] =
;  194          M 0326  2                            .RIGHT_ARG [DBG$L_VALUE_POINTER] +
```

```
;  195          M 0327  2                .RIGHT_ARG [DBG$L_VALUE_POS] / 8;
;  196          M 0328  2            RIGHT_ARG [DBG$L_VALUE_POS] =
;  197          M 0329  2                .RIGHT_ARG [DBG$L_VALUE_POS] MOD 8;
;  198          M 0330  2            END;
;  199          M 0331  2
;  200          M 0332  2        RETURN .RIGHT_ARG;
;  201          M 0333  2        END;
;  202          M 0334  2
;  203          M 0335  2    [ADDR$K_INST, ADDR$K_DATA] :
;  204          M 0336  2        BEGIN
;  205          M 0337  2
;  206          M 0338  2        ! Instruction OP Primary
;  207          M 0339  2        ! Convert the Primary and try again.
;  208          M 0340  2        !
;  209          M 0341  2        IF .RIGHT_TYPE EQL ADDR$K_PRIMARY
;  210          M 0342  2        THEN
;  211          M 0343  2            BEGIN
;  212          M 0344  2            DBG$PRIM_TO_VAL (.RIGHT_ARG, DBG$K_V_VALUE_DESC, RIGHT_ARG);
;  213          M 0345  2            RETURN DBG$EVAL_ADDR_OPERATOR (
;  214          M 0346  2                    .OPERATOR,
;  215          M 0347  2                    .LEFT_ARG,
;  216          M 0348  2                    .RIGHT_ARG);
;  217          M 0349  2            END;
;  218          M 0350  2
;  219          M 0351  2        ! If the right arg is a bitfield, retain the type
;  220          M 0352  2        ! of the right arg.
;  221          M 0353  2        !
;  222          M 0354  2        IF .RIGHT_TYPE EQL ADDR$K_BITFIELD
;  223          M 0355  2        THEN
;  224          M 0356  2            BEGIN
;  225          M 0357  2            RIGHT_ARG [DBG$L_VALUE_POINTER] =
;  226          M 0358  2                .LEFT_ARG  [DBG$L_VALUE_POINTER] OP
;  227          M 0359  2                .RIGHT_ARG [DBG$L_VALUE_POINTER];
;  228          M 0360  2            RIGHT_ARG [DBG$L_VALUE_POS] =
;  229          M 0361  2                .LEFT_ARG [DBG$L_VALUE_POS] OP
;  230          M 0362  2                .RIGHT_ARG [DBG$L_VALUE_POS];
;  231          M 0363  2
;  232          M 0364  2            ! Normalize the bit offset.
;  233          M 0365  2            !
;  234          M 0366  2            RIGHT_ARG [DBG$L_VALUE_POINTER] =
;  235          M 0367  2                .RIGHT_ARG [DBG$L_VALUE_POINTER] +
;  236          M 0368  2                .RIGHT_ARG [DBG$L_VALUE_POS] / 8;
;  237          M 0369  2            RIGHT_ARG [DBG$L_VALUE_POS] =
;  238          M 0370  2                .RIGHT_ARG [DBG$L_VALUE_POS] MOD 8;
;  239          M 0371  2            RETURN .RIGHT_ARG;
;  240          M 0372  2            END;
;  241          M 0373  2
;  242          M 0374  2        ! Add the addresses.
;  243          M 0375  2        !
;  244          M 0376  2        LEFT_ARG [DBG$L_VALUE_POINTER] =
;  245          M 0377  2            .LEFT_ARG  [DBG$L_VALUE_POINTER] OP
;  246          M 0378  2            .RIGHT_ARG [DBG$L_VALUE_POINTER];
;  247          M 0379  2
;  248          M 0380  2        ! If we are retaining instruction type then
;  249          M 0381  2        ! fill in the length correctly.
;  250          M 0382  2        !
;  251          M 0383  2        IF .RIGHT_TYPE EQL ADDR$K_LITERAL
```

```
;  252          M 0384  2                THEN
;  253          M 0385  2                    BEGIN
;  254          M 0386  2                    IF .LEFT_TYPE EQL ADDR$K_INST
;  255          M 0387  2                    THEN
;  256          M 0388  2                        LEFT_ARG[DBG$W_VALUE_LENGTH] =
;  257          M 0389  2                            DBG$INS_DECODE(.LEFT_ARG[DBG$L_VALUE_POINTER],
;  258          M 0390  2                                           FALSE, FALSE) =
;  259          M 0391  2                                .LEFT_ARG[DBG$L_VALUE_POINTER];
;  260          M 0392  2                    END
;  261          M 0393  2
;  262          M 0394  2                    ! If we are not adding a literal value, zero out
;  263          M 0395  2                    ! the type information. This reflects the fact
;  264          M 0396  2                    ! that in expressions such as EX INST OP DATA,
;  265          M 0397  2                    ! the address being examined is of unknown type.
;  266          M 0398  2                    !
;  267          M 0399  2                    ELSE
;  268          M 0400  2                    BEGIN
;  269          M 0401  2                    LEFT_ARG [DBG$B_DHDR_KIND] = RST$K_DATA;
;  270          M 0402  2                    LEFT_ARG [DBG$B_DHDR_FCODE] = RST$R_TYPE_DESCR;
;  271          M 0403  2                    LEFT_ARG [DBG$V_DHDR_OVERRIDE] = TRUE;
;  272          M 0404  2                    LEFT_ARG [DBG$B_VALUE_CLASS] = 0;
;  273          M 0405  2                    LEFT_ARG [DBG$B_VALUE_DTYPE] = .DBG$GL_DFLTTYP;
;  274          M 0406  2                    IF .DBG$GL_DFLTTYP EQL DSC$K_DTYPE_ZI
;  275          M 0407  2                    THEN
;  276          M 0408  2                        LEFT_ARG[DBG$W_VALUE_LENGTH] =
;  277          M 0409  2                            DBG$INS_DECODE(.LEFT_ARG[DBG$L_VALUE_POINTER],
;  278          M 0410  2                                           FALSE, FALSE) =
;  279          M 0411  2                                .LEFT_ARG[DBG$L_VALUE_POINTER]
;  280          M 0412  2                    ELSE
;  281          M 0413  2                        LEFT_ARG [DBG$W_VALUE_LENGTH] = .DBG$GW_DFLTLENG;
;  282          M 0414  2                    END;
;  283          M 0415  2
;  284          M 0416  2                    ! Return the left argument.
;  285          M 0417  2                    !
;  286          M 0418  2                    RETURN .LEFT_ARG;
;  287          M 0419  2                    END;
;  288          M 0420  2
;  289          M 0421  2
;  290          M 0422  2                [ADDR$K_BITFIELD] :
;  291          M 0423  2                    BEGIN
;  292          M 0424  2
;  293          M 0425  2                    ! Bitfield OP Primary.
;  294          M 0426  2                    ! Convert the Primary and try again.
;  295          M 0427  2                    !
;  296          M 0428  2                    IF .RIGHT_TYPE EQL ADDR$K_PRIMARY
;  297          M 0429  2                    THEN
;  298          M 0430  2                        BEGIN
;  299          M 0431  2                        DBG$PRIM_TO_VAL (.RIGHT_ARG, DBG$K_V_VALUE_DESC, RIGHT_ARG);
;  300          M 0432  2                        RETURN DBG$EVAL_ADDR_OPERATOR (
;  301          M 0433  2                                .OPERATOR,
;  302          M 0434  2                                .LEFT_ARG,
;  303          M 0435  2                                .RIGHT_ARG);
;  304          M 0436  2                        END;
;  305          M 0437  2
;  306          M 0438  2                    ! In other cases, add the addresses.
;  307          M 0439  2                    !
;  308          M 0440  2                    LEFT_ARG [DBG$L_VALUE_POINTER] =
```

```
;   309         M 0441  2                       .LEFT_ARG  [DBG$L_VALUE_POINTER] OP
;   310         M 0442  2                       .RIGHT_ARG [DBG$L_VALUE_POINTER];
;   311         M 0443  2                   LEFT_ARG [DBG$L_VALUE_POS] =
;   312         M 0444  2                       .LEFT_ARG [DBG$L_VALUE_POS] OP
;   313         M 0445  2                       .RIGHT_ARG [DBG$L_VALUE_POS];
;   314         M 0446  2
;   315         M 0447  2                   ! Normalize the bit offset.
;   316         M 0448  2                   !
;   317         M 0449  2                   LEFT_ARG [DBG$L_VALUE_POINTER] =
;   318         M 0450  2                           .LEFT_ARG [DBG$L_VALUE_POINTER] +
;   319         M 0451  2                           .LEFT_ARG [DBG$L_VALUE_POS] / 8;
;   320         M 0452  2                   LEFT_ARG [DBG$L_VALUE_POS] =
;   321         M 0453  2                           .LEFT_ARG [DBG$L_VALUE_POS] MOD 8;
;   322         M 0454  2
;   323         M 0455  2                   RETURN .LEFT_ARG;
;   324         M 0456  2                   END;
;   325         M 0457  2
;   326           0458  2               TES %;
;   327           0459  2
;   328           0460  2
;   329           0461  2       ! Multiply and divide are similar to add and subtract except that
;   330           0462  2       ! we never try to retain a Primary for symbolization purposes.
;   331           0463  2       !
;   332         M 0464  2       MACRO PROCESS_MULTIPLY_OR_DIVIDE (OP) =
;   333         M 0465  2           BEGIN
;   334         M 0466  2
;   335         M 0467  2           ! Handle the case where the left arg is a Primary.
;   336         M 0468  2           !
;   337         M 0469  2           IF .LEFT_TYPE EQL ADDR$K_PRIMARY
;   338         M 0470  2           THEN
;   339         M 0471  2               BEGIN
;   340         M 0472  2
;   341         M 0473  2               ! For all right args, we need to convert
;   342         M 0474  2               ! the Primary Descriptor to Value Descriptor before we
;   343         M 0475  2               ! do any operation.
;   344         M 0476  2               !
;   345         M 0477  2               DBG$PRIM_TO_VAL (.LEFT_ARG, DBG$K_V_VALUE_DESC, LEFT_ARG);
;   346         M 0478  2               RETURN DBG$EVAL_ADDR_OPERATOR (
;   347         M 0479  2                                           .OPERATOR,
;   348         M 0480  2                                           .LEFT_ARG,
;   349         M 0481  2                                           .RIGHT_ARG );
;   350         M 0482  2               END;
;   351         M 0483  2
;   352         M 0484  2           ! Handle the case where the right arg is a Primary.
;   353         M 0485  2           !
;   354         M 0486  2           IF .RIGHT_TYPE EQL ADDR$K_PRIMARY
;   355         M 0487  2           THEN
;   356         M 0488  2               BEGIN
;   357         M 0489  2
;   358         M 0490  2               ! Convert the Primary and try again.
;   359         M 0491  2               !
;   360         M 0492  2               DBG$PRIM_TO_VAL (.RIGHT_ARG, DBG$K_V_VALUE_DESC, RIGHT_ARG);
;   361         M 0493  2               RETURN DBG$EVAL_ADDR_OPERATOR (
;   362         M 0494  2                                       .OPERATOR,
;   363         M 0495  2                                       .LEFT_ARG,
;   364         M 0496  2                                       .RIGHT_ARG);
;   365         M 0497  2
```

```
;  366          M 0498  2              END;
;  367          M 0499  2
;  368          M 0500  2          ! Neither arg is a Primary.
;  369          M 0501  2          ! Multiply the addresses. Zero out the bit offset.
;  370          M 0502  2          !
;  371          M 0503  2          LEFT_ARG [DBG$L_VALUE_POINTER] =
;  372          M 0504  2                  .LEFT_ARG  [DBG$L_VALUE_POINTER] OP
;  373          M 0505  2                  .RIGHT_ARG [DBG$L_VALUE_POINTER];
;  374          M 0506  2
;  375          M 0507  2          ! Zero out the bit offset. Change the type to
;  376          M 0508  2          ! unknown and return the left arg.
;  377          M 0509  2          !
;  378          M 0510  2          LEFT_ARG [DBG$V_DHDR_LITERAL] = FALSE;
;  379          M 0511  2          LEFT_ARG [DBG$L_VALUE_POS] = 0;
;  380          M 0512  2          LEFT_ARG [DBG$B_DHDR_KIND] = RST$K_DATA;
;  381          M 0513  2          LEFT_ARG [DBG$B_DHDR_FCODE] = RST$K_TYPE_DESCR;
;  382          M 0514  2          LEFT_ARG [DBG$V_DHDR_OVERRIDE] = TRUE;
;  383          M 0515  2          LEFT_ARG [DBG$B_VALUE_CLASS] = 0;
;  384          M 0516  2          LEFT_ARG [DBG$B_VALUE_DTYPE] = .DBG$GL_DFLTTYP;
;  385          M 0517  2          IF .DBG$GL_DFLTTYP EQL DSC$K_DTYPE_ZI
;  386          M 0518  2          THEN
;  387          M 0519  2              LEFT_ARG[DBG$W_VALUE_LENGTH] =
;  388          M 0520  2                  DBG$INS_DECODE(.LEFT_ARG[DBG$L_VALUE_POINTER],
;  389          M 0521  2                                 FALSE, FALSE) =
;  390          M 0522  2                  .LEFT_ARG[DBG$L_VALUE_POINTER]
;  391          M 0523  2          ELSE
;  392          M 0524  2              LEFT_ARG [DBG$W_VALUE_LENGTH] = .DBG$GW_DFLTLENG;
;  393          M 0525  2          RETURN .LEFT_ARG;
;  394            0526  2          END %;
;  395            0527  2
;  396            0528  2
;  397            0529  2          ! Get the opcode and set the flag saying whether we are processing
;  398            0530  2          ! a binary operator. Also initialize the left and right type codes
;  399            0531  2          ! to unknown.
;  400            0532  2          !
;  401            0533  2          OPCODE = .OPERATOR [TOKEN$W_CODE];
;  402            0534  2          BINARY_FLAG =  .OPCODE EQL TOKEN$K_ADD
;  403            0535  2                      OR .OPCODE EQL TOKEN$K_SUBTRACT
;  404            0536  2                      OR .OPCODE EQL TOKEN$K_MULTIPLY
;  405            0537  2                      OR .OPCODE EQL TOKEN$K_DIVIDE;
;  406            0538  2
;  407            0539  2
;  408            0540  2          ! Call the routine which fills in the codes for LEFT_TYPE and
;  409            0541  2          ! RIGHT_TYPE. If an unconverted value needs to be converted,
;  410            0542  2          ! this routine constructs a new descriptor and the LEFT_ARG and
;  411            0543  2          ! RIGHT_ARG pointers may be modified to point to these new
;  412            0544  2          ! descriptors.
;  413            0545  2          !
;  414            0546  2          DETERMINE_TYPE (.LEFT_ARG, LEFT_ARG, LEFT_TYPE);
;  415            0547  2          IF .BINARY_FLAG
;  416            0548  2          THEN
;  417            0549  2              DETERMINE_TYPE (.RIGHT_ARG, RIGHT_ARG, RIGHT_TYPE);
;  418            0550  2
;  419            0551  2          ! Select the operation to perform based on the operation code.
;  420            0552  2          !
;  421            0553  2          CASE .OPERATOR [TOKEN$W_CODE] FROM TOKEN$K_MIN_OPERATOR TO TOKEN$K_MAX_OPERATOR OF
;  422            0554  2              SET
```

```
;  423          0555  2
;  424          0556  2          ! Do the identity operation.
;  425          0557  2          !
;  426          0558  2          [TOKEN$K_IDENTITY]:
;  427          0559  3              BEGIN
;  428          0560  3              RETURN .LEFT_ARG;
;  429          0561  2              END;
;  430          0562  2
;  431          0563  2          ! Do the indirection (dereferencing) operation.
;  432          0564  2          !
;  433          0565  2          [TOKEN$K_INDIRECT]:
;  434          0566  3              BEGIN
;  435          0567  3
;  436          0568  3              BUILTIN
;  437          0569  3                  PROBER;
;  438          0570  3
;  439          0571  3              LOCAL
;  440          0572  3                  ADDRESS,                ! Address which is value of left arg
;  441          0573  3                  LENGTH;                 ! Bit length given in left arg.
;  442          0574  3
;  443          0575  3              ! Primary Descriptors.
;  444          0576  3              !
;  445          0577  3              IF .LEFT_TYPE EQL ADDR$K_PRIMARY
;  446          0578  3              THEN
;  447          0579  4                  BEGIN
;  448          0580  4                  LOCAL
;  449          0581  4                      LANG,
;  450          0582  4                      VMS_DESC: DBG$STG_DESC;
;  451          0583  4
;  452          0584  4                  ! If we can dereference a typed pointer, and return
;  453          0585  4                  ! a Primary representing the pointed-to object,
;  454          0586  4                  ! then do so.
;  455          0587  4                  !
;  456          0588  4                  IF GET_DEREFERENCE(.LEFT_ARG)
;  457          0589  4                  THEN
;  458          0590  4                      RETURN .LEFT_ARG;
;  459          0591  4
;  460          0592  4                  ! Use MAKE_VMS_DESC and MAKE_VAL_DESC to do the fetch.
;  461          0593  4                  ! Preserve the language code.
;  462          0594  4                  !
;  463          0595  4                  LANG = .LEFT_ARG[DBG$B_DHDR_LANG];
;  464          0596  4                  DBG$MAKE_VMS_DESC (.LEFT_ARG, VMS_DESC);
;  465          0597  4                  LEFT_ARG = DBG$MAKE_VAL_DESC (VMS_DESC, DBG$K_VALUE_DESC);
;  466          0598  4                  LEFT_ARG[DBG$B_DHDR_LANG] = .LANG;
;  467          0599  4
;  468          0600  4                  ! Change the descriptor back to a Volatile Value Descriptor.
;  469          0601  4                  !
;  470          0602  4                  LEFT_ARG[DBG$B_DHDR_TYPE] = DBG$K_V_VALUE_DESC;
;  471          0603  4                  LEFT_ARG[DBG$L_VALUE_POINTER] = .LEFT_ARG[DBG$L_VALUE_VALUE0];
;  472          0604  4                  IF .LEFT_ARG[DBG$B_VALUE_CLASS] EQL DSC$K_CLASS_UBS
;  473          0605  4                  THEN
;  474          0606  4                      LEFT_ARG[DBG$L_VALUE_POS] = .LEFT_ARG[DBG$L_VALUE_VALUE1];
;  475          0607  4
;  476          0608  4                  ! Do a special case check for ".PC"
;  477          0609  4                  !
;  478          0610  4                  IF .VMS_DESC[DSC$A_POINTER] EQLA DBG$REG_VALUES[15]
;  479          0611  4                  THEN
```

```
;  480          0612  5                BEGIN
;  481          0613  5
;  482          0614  5                ! Change dtype to instruction.
;  483          0615  5                !
;  484          0616  5                LEFT_ARG[DBG$B_VALUE_CLASS] = DSC$K_CLASS_S;
;  485          0617  5                LEFT_ARG[DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_ZI;
;  486          0618  5                LEFT_ARG[DBG$W_VALUE_LENGTH] =
;  487          0619  5                    DBG$INS_DECODE(.LEFT_ARG[DBG$L_VALUE_POINTER],
;  488          0620  5                                   FALSE, FALSE) =
;  489          0621  5                    .LEFT_ARG[DBG$L_VALUE_POINTER];
;  490          0622  4                END;
;  491          0623  4            RETURN .LEFT_ARG;
;  492          0624  3            END;
;  493          0625  3
;  494          0626  3        ! Check that we have read access to the address
;  495          0627  3        ! given in the value descriptor.
;  496          0628  3        !
;  497          0629  3        IF NOT PROBER (%REF(0), %REF(1), .LEFT_ARG[DBG$L_VALUE_POINTER])
;  498          0630  3        THEN
;  499          0631  3            SIGNAL (DBG$_NOACCESSR, 1, .LEFT_ARG[DBG$L_VALUE_POINTER]);
;  500          0632  3
;  501          0633  3        ! For bitfields, do the bit selection.
;  502          0634  3        !
;  503          0635  3        IF .LEFT_TYPE EQL ADDR$K_BITFIELD
;  504          0636  3        THEN
;  505          0637  4            BEGIN
;  506          0638  4
;  507          0639  4            ! Do additional checking for length <= 32 bits.
;  508          0640  4            !
;  509          0641  4            LENGTH = .LEFT_ARG [DBG$W_VALUE_LENGTH];
;  510          0642  4            IF .LENGTH LSS 0
;  511          0643  4            OR .LENGTH GTR 32
;  512          0644  4            THEN
;  513          0645  4                SIGNAL (DBG$_ILLSIZFLD, 1, .LENGTH);
;  514          0646  4
;  515          0647  4            ! Also check for read access to top of range.
;  516          0648  4            !
;  517          0649  4            ADDRESS = .LEFT_ARG[DBG$L_VALUE_POINTER] +
;  518          0650  4                      (.LEFT_ARG[DBG$L_VALUE_POS] + .LENGTH - 1) / 8;
;  519          0651  4            IF NOT PROBER (%REF(0), %REF(1), .ADDRESS)
;  520          0652  4            THEN
;  521          0653  4                SIGNAL (DBG$_NOACCESSR, 1, .ADDRESS);
;  522          0654  4
;  523          0655  4            ! Do the bit selection.
;  524          0656  4            !
;  525          0657  4            LEFT_ARG [DBG$L_VALUE_POINTER] =
;  526          0658  5                (IF .LEFT_ARG [DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_SV
;  527          0659  5                 OR .LEFT_ARG [DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_SVU
;  528          0660  5                 THEN
;  529          0661  5                     .(.LEFT_ARG [DBG$L_VALUE_POINTER]) <
;  530          0662  5                             .LEFT_ARG [DBG$L_VALUE_POS],
;  531          0663  5                             .LEFT_ARG [DBG$W_VALUE_LENGTH],
;  532          0664  5                             1>
;  533          0665  5                 ELSE
;  534          0666  5                     .(.LEFT_ARG [DBG$L_VALUE_POINTER]) <
;  535          0667  5                             .LEFT_ARG [DBG$L_VALUE_POS],
;  536          0668  5                             .LEFT_ARG [DBG$W_VALUE_LENGTH],
```

```
;  537     0669  4                                0>);
;  538     0670  4                END
;  539     0671  4
;  540     0672  4            ELSE
;  541     0673  4
;  542     0674  4                ! For other left args, just do the indirection.
;  543     0675  4                !
;  544     0676  4                LEFT_ARG [DBG$L_VALUE_POINTER] = ..LEFT_ARG [DBG$L_VALUE_POINTER];
;  545     0677  4
;  546     0678  4            ! Change the type to unknown and return the argument.
;  547     0679  4            !
;  548     0680  4            LEFT_ARG [DBG$V_DHDR_LITERAL] = FALSE;
;  549     0681  4            LEFT_ARG [DBG$B_DHDR_KIND] = RST$K_DATA;
;  550     0682  4            LEFT_ARG [DBG$B_DHDR_FCODE] = RST$K_TYPE_DESCR;
;  551     0683  4            LEFT_ARG [DBG$V_DHDR_OVERRIDE] = TRUE;
;  552     0684  4            LEFT_ARG [DBG$B_VALUE_CLASS] = 0;
;  553     0685  4            LEFT_ARG [DBG$B_VALUE_DTYPE] = .DBG$GL_DFLTTYP;
;  554     0686  4            IF .DBG$GL_DFLTTYP EQL DSC$K_DTYPE_ZI
;  555     0687  4            THEN
;  556     0688  4                LEFT_ARG[DBG$W_VALUE_LENGTH] =
;  557     0689  4                    DBG$INS_DECODE(.[EFT_ARG[DBG$L_VALUE_POINTER],
;  558     0690  4                                    FALSE, FALSE) =
;  559     0691  4                    .LEFT_ARG[DBG$L_VALUE_POINTER]
;  560     0692  4            ELSE
;  561     0693  4                LEFT_ARG [DBG$W_VALUE_LENGTH] = .DBG$GW_DFLTLENG;
;  562     0694  4            LEFT_ARG [DBG$L_VALUE_POS] = 0;
;  563     0695  4            RETURN .LEFT_ARG;
;  564     0696  4            END;
;  565     0697  4
;  566     0698  4
;  567     0699  4        ! Do the add operation.
;  568     0700  4        !
;  569     0701  4        [TOKEN$K_ADD]:
;  570     0702  4            PROCESS_ADD_OR_SUBTRACT (+);
;  571     0703  4
;  572     0704  4
;  573     0705  4        ! Do the subtract operation.
;  574     0706  4        !
;  575     0707  4        [TOKEN$K_SUBTRACT]:
;  576     0708  4            PROCESS_ADD_OR_SUBTRACT (-);
;  577     0709  4
;  578     0710  4
;  579     0711  4        ! Do the unary plus operation.
;  580     0712  4        !
;  581     0713  4        [TOKEN$K_UNARY_PLUS]:
;  582     0714  4            RETURN .LEFT_ARG;
;  583     0715  4
;  584     0716  4
;  585     0717  4        ! Do the unary minus operation (i.e., negation).
;  586     0718  4        !
;  587     0719  4        [TOKEN$K_UNARY_MINUS]:
;  588     0720  5            BEGIN
;  589     0721  5
;  590     0722  5            ! First convert Primary Descriptors to Value Descriptors.
;  591     0723  5            !
;  592     0724  5            IF .LEFT_TYPE EQL ADDR$K_PRIMARY
;  593     0725  5            THEN
```

```
;  594     0726  3                DBG$PRIM_TO_VAL (.LEFT_ARG, DBG$K_V_VALUE_DESC, LEFT_ARG);
;  595     0727
;  596     0728                ! Negate the address.
;  597     0729                !
;  598     0730                LEFT_ARG [DBG$L_VALUE_POINTER] = - .LEFT_ARG [DBG$L_VALUE_POINTER];
;  599     0731                LEFT_ARG [DBG$L_VALUE_POS] = - .LEFT_ARG [DBG$L_VALUE_POS];
;  600     0732
;  601     0733                ! Normalize the address.
;  602     0734                !
;  603     0735                LEFT_ARG [DBG$L_VALUE_POINTER] =
;  604     0736                    .LEFT_ARG [DBG$L_VALUE_POINTER] +
;  605     0737                    .LEFT_ARG [DBG$L_VALUE_POS] / 8;
;  606     0738                LEFT_ARG [DBG$L_VALUE_POS] =
;  607     0739                    .LEFT_ARG [DBG$L_VALUE_POS] MOD 8;
;  608     0740
;  609     0741                ! Handle result type of instruction.
;  610     0742                !
;  611     0743                IF .LEFT_TYPE EQL ADDR$K_INST
;  612     0744                THEN
;  613     0745                    LEFT_ARG[DBG$W_VALUE_LENGTH] =
;  614     0746                                DBG$INS_DECODE(.LEFT_ARG[DBG$L_VALUE_POINTER],
;  615     0747                                               FALSE, FALSE) =
;  616     0748                                .LEFT_ARG[DBG$L_VALUE_POINTER];
;  617     0749
;  618     0750                ! Return the result.
;  619     0751                RETURN .LEFT_ARG;
;  620     0752                END;
;  621     0753
;  622     0754
;  623     0755  2         ! Do the multiply operation.
;  624     0756  2         !
;  625     0757  2         [TOKEN$K_MULTIPLY]:
;  626     0758  2             PROCESS_MULTIPLY_OR_DIVIDE (*);
;  627     0759  2
;  628     0760  2
;  629     0761  2         ! Do the divide operation.
;  630     0762  2         !
;  631     0763  2         [TOKEN$K_DIVIDE]:
;  632     0764  2             PROCESS_MULTIPLY_OR_DIVIDE (/);
;  633     0765  2
;  634     0766  2
;  635     0767  2         ! Do the bit-selection operation, i.e. X<pos,size,ext>.
;  636     0768  2         !
;  637     0769  2         [TOKEN$K_BITSELECT]:
;  638     0770  3             BEGIN
;  639     0771  3             LOCAL
;  640     0772  3                 ADDR_INCREMENT,         ! Increment to be added to byte address
;  641     0773  3                 BIT_OFFSET,             ! New bit offset from byte address
;  642     0774  3                 DTYPE,                  ! New dtype
;  643     0775  3                 POS;
;  644     0776  3
;  645     0777  3             ! Convert Primaries to Value Descriptors
;  646     0778  3             !
;  647     0779  3             IF .LEFT_ARG [DBG$B_DHDR_TYPE] EQL DBG$K_PRIMARY_DESC
;  648     0780  3             THEN
;  649     0781  3                 DBG$PRIM_TO_VAL (.LEFT_ARG, DBG$K_V_VALUE_DESC, LEFT_ARG);
;  650     0782  3
```

```
;  651          0783  3          !++
;  652          0784  3          ! Handle value descriptors.
;  653          0785  3          !--
;  654          0786  3
;  655          0787  3          ! Clear the literal flag - the result is of type bitfield.
;  656          0788  3          !
;  657          0789  3          LEFT_ARG [DBG$V_DHDR_LITERAL] = FALSE;
;  658          0790  3
;  659          0791  3          ! Add the bit offsets.
;  660          0792  3          !
;  661          0793  3          IF .LEFT_ARG[DBG$B_VALUE_CLASS] EQL DSC$K_CLASS_UBS
;  662          0794  3          THEN
;  663          0795  3              POS = .LEFT_ARG [DBG$L_VALUE_POS]
;  664          0796  3          ELSE
;  665          0797  3              POS = 0;
;  666          0798  3          BIT_OFFSET = .POS + .OPERATOR [TOKEN$W_BIT_OFFSET];
;  667          0799  3
;  668          0800  3          ! Compute the new byte address.
;  669          0801  3          !
;  670          0802  3          LEFT_ARG [DBG$L_VALUE_POINTER] = .LEFT_ARG [DBG$L_VALUE_POINTER] +
;  671          0803  3                                           .BIT_OFFSET / 8;
;  672          0804  3
;  673          0805  3          ! Compute the bit offset. From it and the sign extension bit,
;  674          0806  3          ! determine the new class and dtype.
;  675          0807  3          !
;  676          0808  3          BIT_OFFSET = .BIT_OFFSET MOD 8;
;  677          0809  3          IF .BIT_OFFSET EQL 0
;  678          0810  3          THEN
;  679          0811  4              BEGIN
;  680          0812  4              IF .OPERATOR [TOKEN$V_SGNEXT]
;  681          0813  4              THEN
;  682          0814  4                  DTYPE = DSC$K_DTYPE_SV
;  683          0815  4              ELSE
;  684          0816  4                  DTYPE = DSC$K_DTYPE_V;
;  685          0817  4              LEFT_ARG [DBG$B_VALUE_CLASS] = DSC$K_CLASS_S;
;  686          0818  4              END
;  687          0819  3          ELSE
;  688          0820  4              BEGIN
;  689          0821  4              IF .OPERATOR [TOKEN$V_SGNEXT]
;  690          0822  4              THEN
;  691          0823  4                  DTYPE = DSC$K_DTYPE_SVU
;  692          0824  4              ELSE
;  693          0825  4                  DTYPE = DSC$K_DTYPE_VU;
;  694          0826  4              LEFT_ARG [DBG$B_VALUE_CLASS] = DSC$K_CLASS_UBS;
;  695          0827  3              END;
;  696          0828  3
;  697          0829  3              ! Fill in the new dtype, bit offset, and length.
;  698          0830  3              !
;  699          0831  3              LEFT_ARG [DBG$B_VALUE_DTYPE] = .DTYPE;
;  700          0832  3              LEFT_ARG [DBG$W_VALUE_LENGTH] = .OPERATOR [TOKEN$W_BIT_LENGTH];
;  701          0833  3              LEFT_ARG [DBG$L_VALUE_POS] = .BIT_OFFSET;
;  702          0834  3              RETURN .LEFT_ARG;
;  703          0835  2              END;
;  704          0836  2
;  705          0837  2
;  706          0838  2          ! Any other code constitutes an error and is signalled as such.
;  707          0839  2          ! (The operator is not allowed in Address Expressions.)
```

```
;  708          0840  2          !
;  709          0841  2          [INRANGE, OUTRANGE]:
;  710          0842  2              BEGIN
;  711          0843  2              SIGNAL(DBG$_INVOPADDR, 1, OPERATOR[TOKEN$B_OPLEN]);
;  712          0844  2
;  713          0845  2              ! We never get here but throw in a return to keep the BLISS
;  714          0846  2              ! compiler happy.
;  715          0847  2              !
;  716          0848  2              RETURN 0;
;  717          0849  2              END;
;  718          0850  2
;  719          0851  2          TES;
;  720          0852  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        .WORD   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 00030D          ADDL2   24(R0), 24(R3)                                  ;
                        1C   A3      1C   A0   C0 000312          ADDL2   28(R0), 28(R3)                                  ;
                    50  1C   A3           08   C7 000317          DIVL3   #8, 28(R3), R0                                 ;
                        18   A3           50   C0 00031C          ADDL2   R0, 24(R3)                                     ;
                7E  00  1C   A3           01   7A 000320          EMUL    #1, 28(R3), #0, -(SP)                          ;
                50  50       8E           08   7B 000326          EDIV    #8, (SP)+, R0, R0                              ;
                        1C   A3           50   D0 00032B          MOVL    R0, 28(R3)                                     ;
                                           67   11 00032F 38$:     BRB     45$                                            ;
                        52           08    AC   D0 000331 39$:     MOVL    LEFT_ARG, R2                                   ; 0708
                    04  01               6E   CF 000335          CASEL   LEFT_TYPE, #1, #4                              ;
            0063    0063    000A         0016      000339 40$:     .WORD   42$-40$,-                                      ;
                                         00B7      000341                  41$-40$,-                                      ;
                                                                           46$-40$,-                                      ;
                                                                           46$-40$,-                                      ;
                                                                           50$-40$                                        ;
                                           08   AC   9F 000343 41$:     PUSHAB  LEFT_ARG                                       ;
                        7E           83   8F   9A 000346          MOVZBL  #131, -(SP)                                    ;
                                          52   DD 00034A          PUSHL   R2                                             ;
                                        0149   31 00034C          BRW     63$                                            ;
                        02           04    AE   D1 00034F 42$:     CMPL    RIGHT_TYPE, #2                                 ;
                                          4B   13 000353 43$:     BEQL    47$                                            ;
                        53           0C    AC   D0 000355          MOVL    RIGHT_ARG, R3                                  ;
            18  A3      18   A2      18    A3   C3 000359          SUBL3   24(R3), 24(R2), 24(R3)                         ;
                        03           04    AE   D1 000360          CMPL    RIGHT_TYPE, #3                                 ;
                                          0E   12 000364          BNEQ    44$                                            ;
                                          7E   7C 000366          CLRQ    -(SP)                                          ;
                                          18   A3 DD 000368          PUSHL   24(R3)                                         ;
                        65                      03   FB 00036B          CALLS   #3, DBG$INS_DECODE                             ;
            14  A3          50           18    A3   A3 00036E          SUBW3   24(R3), R0, 20(R3)                             ;
                        05           04    AE   D1 000374 44$:     CMPL    RIGHT_TYPE, #5                                 ;
                                          1E   12 000378          BNEQ    45$                                            ;
                        50           1C    A3   9E 00037A          MOVAB   28(R3), R0                                     ;
                    60  1C   A2           60   C3 00037E          SUBL3   (R0), 28(R2), (R0)                             ;
                    51           60           08   C7 000383          DIVL3   #8, (R0), R1                                   ;
                        18   A3           51   C0 000387          ADDL2   R1, 24(R3)                                     ;
                7E  00           60           01   7A 00038B          EMUL    #1, (R0), #0, -(SP)                            ;
                51  51           8E           08   7B 000390          EDIV    #8, (SP)+, R1, R1                              ;
                                 60           51   D0 000395          MOVL    R1, (R0)                                       ;
                                 50           53   D0 000398 45$:     MOVL    R3, R0                                         ;
                                                   04   00039B          RET                                                    ;
                                 02    04     AE   D1 00039C 46$:     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   A2   CE 0043A          MNEGL   24(R2), 24(R2)                                  ; 0731
                                50        1C   A2   9E 0043F          MOVAB   28(R2), R0                                      ; 0737
                                60                60   CE 00443          MNEGL   (R0), (R0)                                      ; 0739
                51                      60             08   C7 00446          DIVL3   #8, (R0), R1                                    ;
                             18 A2              51   C0 0044A          ADDL2   R1, 24(R2)                                      ;
                7E   00                60             01   7A 0044E          EMUL    #1, (R0), #0, -(SP)                             ;
                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$:   BICB2   #64, 4(R2)                                 ;
                                       A2        1C    A2   D4 004B9       CLRL    28(R2)                                      ;
                                    06 A2      0603    8F   B0 004BC 66$:   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             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                 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                 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
```

```
;  722          0853  1 GLOBAL ROUTINE DBG$PRIM_TO_ADDR (DESC1, DTYPE, DESC2) =
;  723          0854  1 !
;  724          0855  1 ! FUNCTION
;  725          0856  1 !       Converts a Primary Descriptor into a value descriptor containing
;  726          0857  1 !       the address of the primary.
;  727          0858  1 !
;  728          0859  1 !       For Primary Descriptors representing data or code, the address is stored
;  729          0860  1 !       as a (byte address, bit offset) pair in the value field of the
;  730          0861  1 !       descriptor. If the given dtype is zero, the dtype is left alone. If the
;  731          0862  1 !       user supplied a dtype that he wants the result to be, this
;  732          0863  1 !       new dtype is stuffed in and the class and length fields are fixed
;  733          0864  1 !       up accordingly.
;  734          0865  1 !
;  735          0866  1 !       For Primary Descriptors representing literals, the literal value
;  736          0867  1 !       is stored in the value field of the resulting descriptor. The
;  737          0868  1 !       dtype information is left alone in this case - it presumably
;  738          0869  1 !       describes the type of the literal constant.
;  739          0870  1 !
;  740          0871  1 !       Note that BLISS fields are a special kind of literal. The four
;  741          0872  1 !       field values are stored in four longwords in the value field
;  742          0873  1 !       of the resulting value descriptor.
;  743          0874  1 !
;  744          0875  1 !
;  745          0876  1 ! INPUTS
;  746          0877  1 !       DESC1   - The address of the Primary Descriptor to be converted.
;  747          0878  1 !       DTYPE   - The desired Dtype code of the resulting value descriptor
;  748          0879  1 !       DESC2   - An address in which to leave a pointer to the new value
;  749          0880  1 !                 descriptor which is constructed.
;  750          0881  1 !
;  751          0882  1 ! OUTPUTS
;  752          0883  1 !       A Value Descriptor is allocated and filled in, and a pointer
;  753          0884  1 !       to it is left in DESC2. A status code is returned, which is
;  754          0885  1 !       one of:
;  755          0886  1 !       STS$K_SUCCESS   - Success.
;  756          0887  1 !       STS$K_SEVERE    - Failure.
;  757          0888  1 !
;  758          0889  2     BEGIN
;  759          0890  2
;  760          0891  2     MAP
;  761          0892  2         DESC1: REF DBG$PRIMARY;          ! Pointer to input primary descriptor
;  762          0893  2
;  763          0894  2     LOCAL
;  764          0895  2         COUNT,                           ! For BLISS fields, count of fields
;  765          0896  2         DSTPTR: REF DST$RECORD,          ! Pointer to DST record for the primary
;  766          0897  2         PTR,                             ! Scratch pointer
;  767          0898  2         RSTPTR: REF RST$ENTRY,           ! Pointer to RST entry for the primary
;  768          0899  2         SUBVECTOR: REF VECTOR[],         ! Pointer to vector of field values
;  769          0900  2         TEMP_DESC: REF DBG$VALDESC;      ! Pointer to a value descriptor
;  770          0901  2
;  771          0902  2
;  772          0903  2     ! Check for a volatile value descriptor coming in. In this case,
;  773          0904  2     ! turn it into an ordinary value descriptor with the "address" in
;  774          0905  2     ! the V. Value desc becoming the "value" in the ordinary value desc.
;  775          0906  2     !
;  776          0907  2     IF .DESC1[DBG$B_DHDR_TYPE] EQL DBG$K_V_VALUE_DESC
;  777          0908  2     THEN
;  778          0909  3         BEGIN
```

```
;  779          0910  3          MAP
;  780          0911  3              DESC2: REF DBG$VALDESC;
;  781          0912  3          LOCAL
;  782          0913  3              DUMMY;
;  783          0914  3
;  784          0915  3          ! Slightly kludgy solution to an obscure problem:
;  785          0916  3          ! If you examine a string of length > 256 and language is
;  786          0917  3          ! set to BLISS or MACRO then you get into this routine
;  787          0918  3          ! with a volatile value descriptor representing the string.
;  788          0919  3          ! In this case we just want to return the descriptor unchanged:
;  789          0920  3          ! do not add the level of indirection.
;  790          0921  3          !
;  791          0922  3          IF .DESC1[DBG$W_VALUE_LENGTH] GTR 256
;  792          0923  3          THEN
;  793          0924  4              BEGIN
;  794          0925  4              .DESC2 = .DESC1;
;  795          0926  4              RETURN STS$K_SUCCESS;
;  796          0927  4              END;
;  797          0928  3
;  798          0929  3          ! Otherwise, copy the descriptor and make it into an ordinary
;  799          0930  3          ! value descriptor.
;  800          0931  3          !
;  801          0932  3          DBG$NCOPY_DESC(.DESC1, TEMP_DESC, DUMMY, FALSE);
;  802          0933  3          TEMP_DESC[DBG$B_DHDR_TYPE] = DBG$K_VALUE_DESC;
;  803          0934  3          TEMP_DESC[DBG$L_VALUE_VALUE0] = .TEMP_DESC[DBG$L_VALUE_POINTER];
;  804          0935  3          IF .TEMP_DESC[DBG$B_VALUE_CLASS] EQL DSC$K_CLASS_UBS
;  805          0936  3          THEN
;  806          0937  3              TEMP_DESC[DBG$L_VALUE_VALUE1] = .TEMP_DESC[DBG$L_VALUE_POS];
;  807          0938  3          TEMP_DESC[DBG$L_VALUE_POINTER] = TEMP_DESC[DBG$L_VALUE_VALUE0];
;  808          0939  3          .DESC2 = .TEMP_DESC;
;  809          0940  3          RETURN STS$K_SUCCESS;
;  810          0941  2          END;
;  811          0942  2
;  812          0943  2
;  813          0944  2      ! Check for BLISS field. This is a hack needed to support BLISS structure
;  814          0945  2      ! references X[fieldname]. Basically, when we parse the "fieldname"
;  815          0946  2      ! primary, we are expecting either an offset as in X[0,0,32,0] or a
;  816          0947  2      ! fieldname as in X[fieldname].
;  817          0948  2      ! If we get a fieldname, we indicate it by building a special
;  818          0949  2      ! kind of value descriptor whose fcode is
;  819          0950  2      ! RST$K_TYPE_BLIFLD, and in the value field we have the four integers
;  820          0951  2      ! that the fieldname translates into.
;  821          0952  2      !
;  822          0953  2      IF .DESC1 [DBG$B_DHDR_FCODE] EQL RST$K_TYPE_BLIFLD
;  823          0954  2      THEN
;  824          0955  3          BEGIN
;  825          0956  3          RSTPTR = .DESC1 [DBG$L_DHDR_SYMIDO];
;  826          0957  3          DSTPTR = .RSTPTR [RST$L_DSTPTR];
;  827          0958  3
;  828          0959  3          ! Set up a pointer to the BLISS field components,
;  829          0960  3          ! and copy those four components into the value descriptor.
;  830          0961  3          !
;  831          0962  3          COUNT = .DSTPTR[DST$L_BLIFLD_COMPS];
;  832          0963  3          PTR = 1 + DSTPTR[DST$B_NAME] + .DSTPTR[DST$B_NAME];
;  833          0964  3
;  834          0965  3          ! Allocate the result descriptor now that we know how big
;  835          0966  3          ! it needs to be.
```

```
;  836          0967  3          !
;  837          0968  3          TEMP_DESC = DBG$MAKE_SKELETON_DESC (DBG$K_VALUE_DESC, 4*(.COUNT+1));
;  838          0969  3          TEMP_DESC [DBG$B_DHDR_LANG]   = .DESC1 [DBG$B_DHDR_LANG];
;  839          0970  3          TEMP_DESC [DBG$B_DHDR_KIND]   = .DESC1 [DBG$B_DHDR_KIND];
;  840          0971  3          TEMP_DESC [DBG$B_DHDR_FCODE]  = .DESC1 [DBG$B_DHDR_FCODE];
;  841          0972  3          TEMP_DESC [DBG$L_DHDR_TYPEID] = .DESC1 [DBG$L_DHDR_TYPEID];
;  842          0973  3
;  843          0974  3          SUBVECTOR = TEMP_DESC [DBG$A_VALUE_ADDRESS];
;  844          0975  3          SUBVECTOR[0] = .COUNT;
;  845          0976  3          INCR I FROM 1 TO .COUNT DO
;  846          0977  4              BEGIN
;  847          0978  4              SUBVECTOR [.I] = ..PTR;
;  848          0979  4              PTR = .PTR + 4;
;  849          0980  3              END;
;  850          0981  3
;  851          0982  3          ! Fix up the pointer field, fill in the output parameter, and
;  852          0983  3          ! return success.
;  853          0984  3          !
;  854          0985  3          TEMP_DESC [DBG$L_VALUE_POINTER] = TEMP_DESC [DBG$A_VALUE_ADDRESS];
;  855          0986  3          .DESC2 = .TEMP_DESC;
;  856          0987  3          RETURN STS$K_SUCCESS;
;  857          0988  2          END;
;  858          0989  2
;  859          0990  2      ! Check for literal constants.
;  860          0991  2      !
;  861          0992  2      IF DBG$STA_SYM_IS_LITERAL (.DESC1 [DBG$L_DHDR_SYMIDO])
;  862          0993  2      THEN
;  863          0994  3          BEGIN
;  864          0995  3
;  865          0996  3          ! PRIM_TO_VAL does what we want with literals, so we just
;  866          0997  3          ! call that routine.
;  867          0998  3          !
;  868          0999  3          DBG$PRIM_TO_VAL (.DESC1, DBG$K_VALUE_DESC, .DESC2);
;  869          1000  3          RETURN STS$K_SUCCESS;
;  870          1001  2          END;
;  871          1002  2
;  872          1003  2      ! Build a value descriptor of the desired type.
;  873          1004  2      ! Fill in the header fields of this
;  874          1005  2      ! value descriptor.
;  875          1006  2      !
;  876          1007  2      TEMP_DESC = DBG$MAKE_SKELETON_DESC (DBG$K_VALUE_DESC, 4);
;  877          1008  2      TEMP_DESC [DBG$B_DHDR_LANG]   = .DESC1 [DBG$B_DHDR_LANG];
;  878          1009  2      TEMP_DESC [DBG$B_DHDR_KIND]   = .DESC1 [DBG$B_DHDR_KIND];
;  879          1010  2      TEMP_DESC [DBG$B_DHDR_FCODE]  = .DESC1 [DBG$B_DHDR_FCODE];
;  880          1011  2      TEMP_DESC [DBG$L_DHDR_TYPEID] = .DESC1 [DBG$L_DHDR_TYPEID];
;  881          1012  2
;  882          1013  2      ! Set up for a call to DBG$MAKE_VMS_DESC. This routine (in DBGVALUES)
;  883          1014  2      ! converts Primary Descriptors to VMS standard descriptors. Since the
;  884          1015  2      ! value descriptor has a VMS descriptor inside it, we pass this
;  885          1016  2      ! to DBG$MAKE_VMS_DESC.
;  886          1017  2      !
;  887          1018  2      IF NOT DBG$MAKE_VMS_DESC (.DESC1, TEMP_DESC[DBG$A_VALUE_VMSDESC])
;  888          1019  2      THEN
;  889          1020  2          RETURN STS$K_SEVERE;
;  890          1021  2
;  891          1022  2      ! Normalize the pointer to include the bit offset.
;  892          1023  2      !
```

```
;  893          1024  2     IF .TEMP_DESC [DBG$B_VALUE_CLASS] EQL DSC$K_CLASS_UBS
;  894          1025  2     THEN
;  895          1026  3         BEGIN
;  896          1027  3         TEMP_DESC [DBG$L_VALUE_POINTER] = .TEMP_DESC [DBG$L_VALUE_POINTER] +
;  897          1028  3                 .TEMP_DESC [DBG$L_VALUE_POS] / %BPUNIT;
;  898          1029  3         TEMP_DESC [DBG$L_VALUE_POS] = .TEMP_DESC [DBG$L_VALUE_POS] MOD %BPUNIT;
;  899          1030  2         END;
;  900          1031  2
;  901          1032  2     ! If this routine was passed a nonzero DTYPE, fill in the desired DTYPE.
;  902          1033  2     ! Also fill in all the related fields.
;  903          1034  2     ! Otherwise, leave it the way we got it from MAKE_VMS_DESC.
;  904          1035  2     !
;  905          1036  2     IF .DTYPE NEQ 0
;  906          1037  2     THEN
;  907          1038  2
;  908          1039  2         ! The only case where we dummy in a different dtype is for address
;  909          1040  2         ! arithmetic in BLISS. E.g., EVAL F+F, without dots, we want to
;  910          1041  2         ! treat the address of F as a longword integer.
;  911          1042  2         !
;  912          1043  2         ! In this case, the dtype we pass in is L.
;  913          1044  2         !
;  914          1045  2         IF .DTYPE EQL DSC$K_DTYPE_L
;  915          1046  2         THEN
;  916          1047  3             BEGIN
;  917          1048  3             TEMP_DESC [DBG$B_DHDR_KIND] = RST$K_DATA;
;  918          1049  3             TEMP_DESC [DBG$B_DHDR_FCODE] = RST$K_TYPE_DESCR;
;  919          1050  3             TEMP_DESC [DBG$B_VALUE_CLASS] = DSC$K_CLASS_S;
;  920          1051  3             TEMP_DESC [DBG$B_VALUE_DTYPE]  = DSC$K_DTYPE_L;
;  921          1052  3             TEMP_DESC [DBG$W_VALUE_LENGTH] = 4;
;  922          1053  3             END
;  923          1054  3
;  924          1055  3         ! We do not currently handle other override dtypes here.
;  925          1056  3         !
;  926          1057  2         ELSE
;  927          1058  2             $DBG_ERROR ('DBGADDEXP\DBG$PRIM_TO_ADDR unknown dtype');
;  928          1059  2
;  929          1060  2     ! Extract the desired information from the value descriptor.
;  930          1061  2     !
;  931          1062  2     TEMP_DESC[DBG$L_VALUE_VALUE0] = .TEMP_DESC[DBG$L_VALUE_POINTER];
;  932          1063  2     TEMP_DESC[DBG$L_VALUE_VALUE1] = .TEMP_DESC[DBG$L_VALUE_POS];
;  933          1064  2
;  934          1065  2     ! Fix up the pointer field, fill in the output parameter, and
;  935          1066  2     ! return success.
;  936          1067  2     !
;  937          1068  2     TEMP_DESC [DBG$L_VALUE_POINTER] = TEMP_DESC [DBG$A_VALUE_ADDRESS];
;  938          1069  2     .DESC2 = .TEMP_DESC;
;  939          1070  2     RETURN STS$K_SUCCESS
;  940          1071  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$MAKE_VMS_DESC                           ;
                                    50  E8 00100          BLBS    R0, 10$                                         ;
                                    50  04 D0 00103          MOVL    #4, R0                                      ; 1020
                                        04 00106          RET                                                     ;
                        0D      03  A4 91 00107 10$:      CMPB    3(R4), #13                                      ; 1024
                                    18  12 00108          BNEQ    11$                                             ;
                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
```

```
;  942          1072  1 ROUTINE DETERMINE_TYPE (ARG, NEW_ARG, TYPE) : NOVALUE =
;  943          1073  1 !
;  944          1074  1 ! FUNCTION
;  945          1075  1 !
;  946          1076  1 !       Given a pointer to a descriptor, this routine does two things:
;  947          1077  1 !       1 - Determines which of the address types the descriptor represents
;  948          1078  1 !       2 - If necessary, calls CONV_TEXT_VALUE and returns a converted arg
;  949          1079  1 !           in NEW_ARG.
;  950          1080  1 !
;  951          1081  1 ! INPUTS
;  952          1082  1 !
;  953          1083  1 !       ARG     - Points to either a Primary Descriptor or a Value Descriptor
;  954          1084  1 !                 or a Volatile Value Descriptor
;  955          1085  1 !       NEW_ARG - The address in which to leave a pointer to the converted
;  956          1086  1 !                 descriptor.
;  957          1087  1 !       TYPE    - The address in which to leave a type code
;  958          1088  1 !
;  959          1089  1 ! OUTPUTS
;  960          1090  1 !
;  961          1091  1 !       In TYPE, returns one of the five type codes ADDR$K_LITERAL, ADDR$K_INST,
;  962          1092  1 !       ADDR$K_DATA, ADDR$K_BITFIELD, ADDR$K_PRIMARY, saying what kind of
;  963          1093  1 !       address the descriptor represents.
;  964          1094  1 !
;  965          1095  1 !       In NEW_ARG, returns the address of the original argument if no conversion
;  966          1096  1 !       was done on the descriptor. Otherwise, returns the address of the
;  967          1097  1 !       new descriptor.
;  968          1098  1 !
;  969          1099  2     BEGIN
;  970          1100  2
;  971          1101  2     MAP
;  972          1102  2         ARG : REF DBG$VALDESC;
;  973          1103  2
;  974          1104  2     LOCAL
;  975          1105  2         DTYPE,                          ! Holds a dtype code
;  976          1106  2         NAME: REF VECTOR[,BYTE],        ! Holds name of a primary
;  977          1107  2         SYMID;                          ! Pointer to a SYMID
;  978          1108  2
;  979          1109  2     ! Turn value descriptors into volatile value descriptors.
;  980          1110  2     !
;  981          1111  2     IF .ARG [DBG$B_DHDR_TYPE] EQL DBG$K_VALUE_DESC
;  982          1112  2     THEN
;  983          1113  3         BEGIN
;  984          1114  3
;  985          1115  3         ! Check for "unconverted" value descriptors as input. These arise from
;  986          1116  3         ! constants, e.g. on EXAMINE 1000 we get a value descriptor with
;  987          1117  3         ! the string "1000".
;  988          1118  3         !
;  989          1119  3         IF .ARG [DBG$V_DHDR_UNCVT]
;  990          1120  3         THEN
;  991          1121  4             BEGIN
;  992          1122  4
;  993          1123  4             ! Check for the correct dtype.
;  994          1124  4             !
;  995          1125  4             SELECTONE .ARG [DBG$B_VALUE_DTYPE] OF
;  996          1126  4                 SET
;  997          1127  4                 [DSC$K_DTYPE_L]:
;  998          1128  4                     ARG = DBG$CONV_TEXT_VALUE (.ARG, .ARG, 0);
```

```
;  999          1129  4
; 1000          1130  4                    ! The number scanner may pick up the constant as pack decimal
; 1001          1131  4                    ! for some languages, take in pack decimal with no '.' and
; 1002          1132  4                    ! treat it as long.
; 1003          1133  4                    !
; 1004          1134  4                    [DSC$K_DTYPE_P]:
; 1005          1135  5                        BEGIN
; 1006          1136  5                        IF NOT CH$FIND_CH(.ARG [DBG$W_VALUE_LENGTH],
; 1007          1137  5                                          .ARG [DBG$L_VALUE_POINTER], %C'.')
; 1008          1138  5                        THEN
; 1009          1139  6                            BEGIN
; 1010          1140  6                            ARG [DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_L;
; 1011          1141  6                            ARG [DBG$W_VALUE_TOKENCODE] = TOKEN$K_INTEGER;
; 1012          1142  6                            ARG = DBG$CONV_TEXT_VALUE (.ARG, .ARG, 0);
; 1013          1143  6                            END
; 1014          1144  5                        ELSE
; 1015          1145  5                            SIGNAL (DBG$_ILLADDCON, 2, .ARG [DBG$W_VALUE_LENGTH],
; 1016          1146  5                                    .ARG [DBG$L_VALUE_POINTER]);
; 1017          1147  5
; 1018          1148  4                        END;
; 1019          1149  4
; 1020          1150  4                    [OTHERWISE]:
; 1021          1151  4                        SIGNAL (DBG$_ILLADDCON, 2, .ARG [DBG$W_VALUE_LENGTH],
; 1022          1152  4                                .ARG [DBG$L_VALUE_POINTER]);
; 1023          1153  4                    TES;
; 1024          1154  4
; 1025          1155  4                ! Fill in correct dtype and length information.
; 1026          1156  4                !
; 1027          1157  4                ARG [DBG$V_DHDR_LITERAL] = TRUE;
; 1028          1158  4                ARG [DBG$V_DHDR_OVERRIDE] = TRUE;
; 1029          1159  4                ARG [DBG$B_VALUE_CLASS] = 0;
; 1030          1160  4                ARG [DBG$B_VALUE_DTYPE] = .DBG$GL_DFLTTYP;
; 1031          1161  4                IF .DBG$GL_DFLTTYP EQL DSC$K_DTYPE_ZI
; 1032          1162  4                THEN
; 1033          1163  4                    ARG[DBG$W_VALUE_LENGTH] =
; 1034          1164  4                        DBG$INS_DECODE(..ARG[DBG$L_VALUE_POINTER],
; 1035          1165  4                                       FALSE, FALSE) -
; 1036          1166  4                        ..ARG[DBG$L_VALUE_POINTER]
; 1037          1167  4                ELSE
; 1038          1168  4                    ARG [DBG$W_VALUE_LENGTH] = .DBG$GW_DFLTLENG;
; 1039          1169  3                END;
; 1040          1170  3
; 1041          1171  3                ! Make the value descriptor into a volatile value descriptor.
; 1042          1172  3                !
; 1043          1173  3                ARG [DBG$B_DHDR_TYPE] = DBG$K_V_VALUE_DESC;
; 1044          1174  3                ARG [DBG$L_VALUE_POINTER] = .ARG [DBG$L_VALUE_VALUE0];
; 1045          1175  3                IF .ARG [DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_V
; 1046          1176  3                OR .ARG [DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_SV
; 1047          1177  3                OR .ARG [DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_VU
; 1048          1178  3                OR .ARG [DBG$B_VALUE_DTYPE] EQL DSC$K_DTYPE_SVU
; 1049          1179  3                THEN
; 1050          1180  3                    ARG [DBG$L_VALUE_POS] = .ARG [DBG$L_VALUE_VALUE1];
; 1051          1181  2                END;
; 1052          1182  2
; 1053          1183  2
; 1054          1184  2            ! Case on the kind of descriptor.
; 1055          1185  2            !
```

```
;  1056          1186  2     SELECTONE .ARG[DBG$B_DHDR_TYPE] OF
;  1057          1187  2         SET
;  1058          1188  2
;  1059          1189  2         ! Primary Descriptors.
;  1060          1190  2         !
;  1061          1191  2         [DBG$K_PRIMARY_DESC] :
;  1062          1192  3             BEGIN
;  1063          1193  3
;  1064          1194  3             ! Check for Primary representing a literal.
;  1065          1195  3             !   Example: In PASCAL,
;  1066          1196  3             !   CONST
;  1067          1197  3             !       X = 512;
;  1068          1198  3             !       Y = 1.1;
;  1069          1199  3             !
;  1070          1200  3             !   On EXAMINE X, we will construct a Value Descriptor with 512 in
;  1071          1201  3             !   the value field. On EXAMINE Y, we will signal an error.
;  1072          1202  3             !
;  1073          1203  3             IF DBG$STA_SYM_IS_LITERAL (.ARG [DBG$L_DHDR_SYMID0])
;  1074          1204  3             THEN
;  1075          1205  4                 BEGIN
;  1076          1206  4
;  1077          1207  4                 ! Save the symid
;  1078          1208  4                 !
;  1079          1209  4                 SYMID = .ARG [DBG$L_DHDR_SYMID0];
;  1080          1210  4
;  1081          1211  4                 ! Convert the Primary into a Value Descriptor and obtain the dtype.
;  1082          1212  4                 !
;  1083          1213  4                 DBG$PRIM_TO_VAL (.ARG, DBG$K_VALUE_DESC, ARG);
;  1084          1214  4                 DTYPE = .ARG[DBG$B_VALUE_DTYPE];
;  1085          1215  4
;  1086          1216  4                 ! Check for integer dtype. These are the only kinds of
;  1087          1217  4                 ! literals we can do address arithmetic on.
;  1088          1218  4                 !
;  1089          1219  4                 IF   .DTYPE NEQ DSC$K_DTYPE_Z
;  1090          1220  4                 AND .DTYPE NEQ DSC$K_DTYPE_L
;  1091          1221  4                 AND .DTYPE NEQ DSC$K_DTYPE_LU
;  1092          1222  4                 AND .DTYPE NEQ DSC$K_DTYPE_W
;  1093          1223  4                 AND .DTYPE NEQ DSC$K_DTYPE_WU
;  1094          1224  4                 AND .DTYPE NEQ DSC$K_DTYPE_B
;  1095          1225  4                 AND .DTYPE NEQ DSC$K_DTYPE_BU
;  1096          1226  4                 THEN
;  1097          1227  5                     BEGIN
;  1098          1228  5
;  1099          1229  5                     ! Call the routine that turns a symid into a name, in order
;  1100          1230  5                     ! to be able to signal the error.
;  1101          1231  5                     !
;  1102          1232  5                     DBG$STA_SYMNAME (.SYMID, NAME);
;  1103          1233  5                     SIGNAL (DBG$_ILLADDCON, 2, .NAME[0], NAME[1]);
;  1104          1234  4                     END;
;  1105          1235  4
;  1106          1236  4                 ! Make the value descriptor into a volatile value descriptor.
;  1107          1237  4                 !
;  1108          1238  4                 ARG [DBG$B_DHDR_TYPE] = DBG$K_V_VALUE_DESC;
;  1109          1239  4                 ARG [DBG$L_VALUE_POINTER] = .ARG [DBG$L_VALUE_VALUE0];
;  1110          1240  4                 IF .ARG [DBG$B_VALUE_CLASS] EQL DSC$K_CLASS_UBS
;  1111          1241  4                 THEN
;  1112          1242  4                     ARG [DBG$L_VALUE_POS] = .ARG [DBG$L_VALUE_VALUE1];
```

```
;  1113          1243  4
;  1114          1244  4                ! Put in the flag for literal.
;  1115          1245  4                !
;  1116          1246  4                ARG [DBG$V_DHDR_LITERAL] = TRUE;
;  1117          1247  4                .TYPE = ADDR$K_LITERAL;
;  1118          1248  4                END
;  1119          1249  4
;  1120          1250  3            ELSE
;  1121          1251  3
;  1122          1252  3                ! The value is not a literal. Set the type code to Primary
;  1123          1253  3                !
;  1124          1254  3                .TYPE = ADDR$K_PRIMARY;
;  1125          1255  3
;  1126          1256  2            END;
;  1127          1257  2
;  1128          1258  2        ! Volatile Value Descriptors.
;  1129          1259  2        !
;  1130          1260  2        [DBG$K_V_VALUE_DESC] :
;  1131          1261  3            BEGIN
;  1132          1262  3
;  1133          1263  3            ! If the descriptor came from a literal constant, fill in
;  1134          1264  3            ! dtype of literal.
;  1135          1265  3            !
;  1136          1266  3            IF .ARG [DBG$V_DHDR_LITERAL]
;  1137          1267  3            THEN
;  1138          1268  3                .TYPE = ADDR$K_LITERAL
;  1139          1269  3
;  1140          1270  3            ! Else look at the dtype field.
;  1141          1271  3            !
;  1142          1272  3            ELSE
;  1143          1273  3                SELECTONE .ARG [DBG$B_VALUE_DTYPE] OF
;  1144          1274  3                    SET
;  1145          1275  3
;  1146          1276  3                    [DSC$K_DTYPE_ZI, DSC$K_DTYPE_ZEM] :
;  1147          1277  4                        BEGIN
;  1148          1278  4
;  1149          1279  4                        ! Change type from entry mask to instruction if
;  1150          1280  4                        ! any operations are to be performed.
;  1151          1281  4                        !
;  1152          1282  4                        ARG [DBG$B_VALUE_DTYPE] = DSC$K_DTYPE_ZI;
;  1153          1283  4                        .TYPE = ADDR$K_INST;
;  1154          1284  3                        END;
;  1155          1285  3
;  1156          1286  3                    [DSC$K_DTYPE_VU, DSC$K_DTYPE_V,
;  1157          1287  3                     DSC$K_DTYPE_SVU, DSC$K_DTYPE_SV] :
;  1158          1288  3                        .TYPE = ADDR$K_BITFIELD;
;  1159          1289  3
;  1160          1290  3                    [OTHERWISE] :
;  1161          1291  3                        .TYPE = ADDR$K_DATA;
;  1162          1292  3
;  1163          1293  3                    TES;
;  1164          1294  3
;  1165          1295  3        ! For non-bitfield types, make sure the POS field is zero.
;  1166          1296  3        !
;  1167          1297  3        IF ..TYPE NEQ ADDR$K_BITFIELD
;  1168          1298  3        THEN
;  1169          1299  3            ARG [DBG$L_VALUE_POS] = 0;
```

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


                                                                .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  00048           .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$:      MOVL    36(R0), 28(R0)                              ; 1180
                         52    A0      04 AC  D0 000C7 9$:      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
```

```
;  1186          1315  1 ROUTINE GET_DEREFERENCE (PRIMPTR) =
;  1187          1316  1 !
;  1188          1317  1 ! FUNCTION
;  1189          1318  1 !       This routine is called upon seeing the dereference operator
;  1190          1319  1 !       in an address expression, when the operand is a primary.
;  1191          1320  1 !       E.g., "EXAM *PTR" in language C.
;  1192          1321  1 !
;  1193          1322  1 !       If the object being dereferenced is a typed pointer then
;  1194          1323  1 !       this routine dereferences it by modifying the Primary
;  1195          1324  1 !       Descriptor to refer to the pointed-to object. The value
;  1196          1325  1 !       "TRUE" is then returned indicating that the routine
;  1197          1326  1 !       was successful.
;  1198          1327  1 !
;  1199          1328  1 !       This routine is similar to the GET_DEREFERENCE routine
;  1200          1329  1 !       in DBGPARSER.
;  1201          1330  1 !
;  1202          1331  1 ! INPUTS
;  1203          1332  1 !       PRIMPTR         - A pointer to the Primary Descriptor
;  1204          1333  1 !                         being dereferenced.
;  1205          1334  1 !
;  1206          1335  1 ! OUTPUTS
;  1207          1336  1 !       If the Primary represents a typed pointer, then
;  1208          1337  1 !       it is modified and TRUE is returned.
;  1209          1338  1 !
;  1210          1339  1 !       If the Primary does not represent a typed pointer then
;  1211          1340  1 !       it is not modified and FALSE is returned.
;  1212          1341  1 !
;  1213          1342  2     BEGIN
;  1214          1343  2     MAP
;  1215          1344  2         PRIMPTR: REF DBG$PRIMARY;
;  1216          1345  2
;  1217          1346  2     LOCAL
;  1218          1347  2         FCODE,                          ! Local variable holding fcode info
;  1219          1348  2         JUNK,                           ! Dummy output parameter
;  1220          1349  2         NODEPTR: REF DBG$PRIM_NODE,     ! Points to a Primary Sub-node
;  1221          1350  2         TYPEID;                         ! Pointer to a RST type entry
;  1222          1351  2
;  1223          1352  2
;  1224          1353  2     ! Check that the object being dereferenced is actually a pointer.
;  1225          1354  2     !
;  1226          1355  2     IF  .PRIMPTR[DBG$B_DHDR_FCODE] NEQ RST$K_TYPE_TPTR
;  1227          1356  2     THEN
;  1228          1357  2         RETURN FALSE;
;  1229          1358  2
;  1230          1359  2
;  1231          1360  2     ! Obtain a pointer to the bottom level sub-node by following th
;  1232          1361  2     ! back-pointer. Light the EVAL bit in this subnode,
;  1233          1362  2     ! which indicates that pointer dereferencing is
;  1234          1363  2     ! taking place.
;  1235          1364  2     ! Then, obtain the pointer to the RST type entry for the object
;  1236          1365  2     ! dereferenced.
;  1237          1366  2     !
;  1238          1367  2     NODEPTR = .PRIMPTR [DBG$L_PRIM_BLINK];
;  1239          1368  2     NODEPTR [DBG$V_PNODE_EVAL] = TRUE;
;  1240          1369  2     TYPEID = .NODEPTR [DBG$L_PNODE_TYPEID];
;  1241          1370  2
;  1242          1371  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:[DEB 'G.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
```

STRUCDEF
REQ

TEMPREQ
REQ

DBGADDEXP
LIS

DBGATSIGN
LIS

SSIDEF
REQ