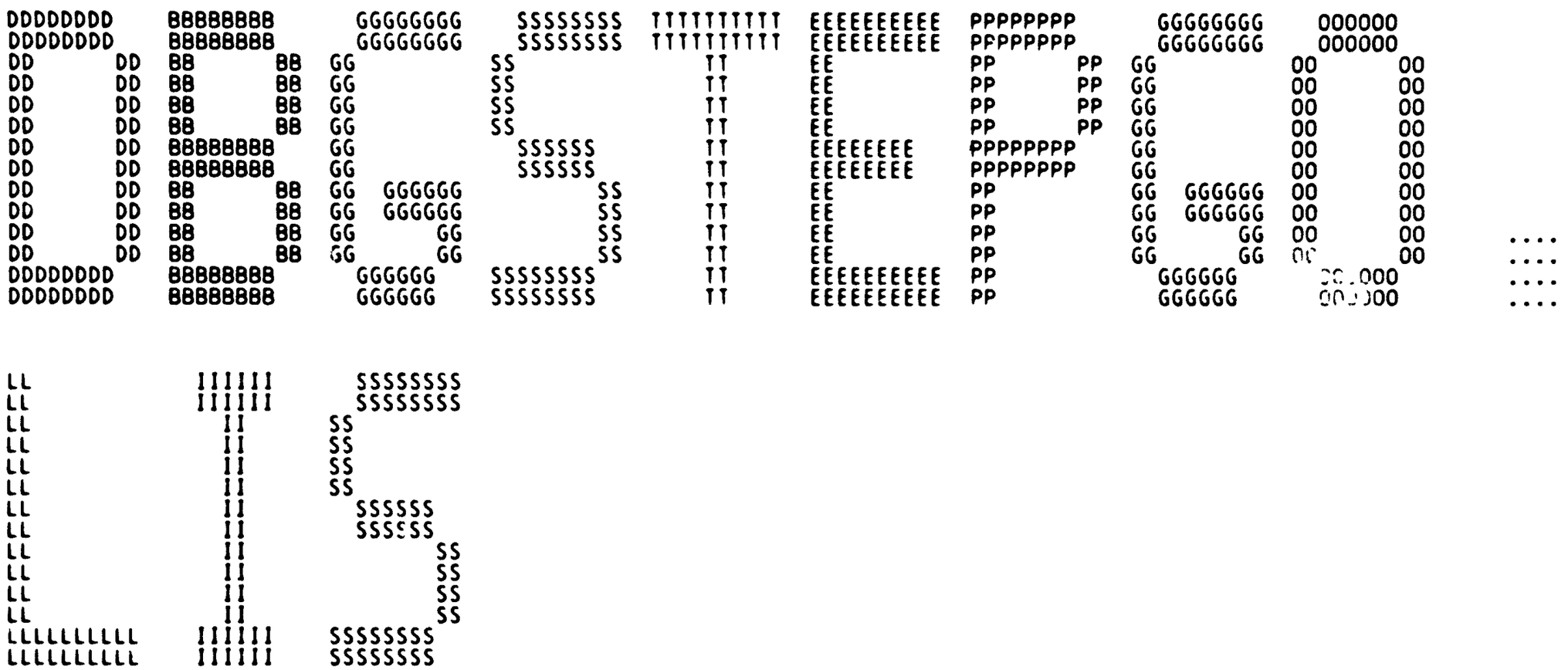
```
DDDDDDDDDDD     EEEEEEEEEEEEEEE  BBBBBBBBBBB      UUU         UUU     GGGGGGGGGGG
DDDDDDDDDDDD    EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUU         UUU     GGGGGGGGGGGG
DDDDDDDDDDDD    EEEEEEEEEEEEEEE  BBBBBBBBBBBB     UUU         UUU     GGGGGGGGGGGG
DDD         DDD EEE              BBB         BBB  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 SSSSSSSS TTTTTTTTTT EEEEEEEEEE PPPPPPPP GGGGGGGG 000000
DDDDDDDD BBBBBBBB GGGGGGGG SSSSSSSS TTTTTTTTTT EEEEEEEEEE PPPPPPPP GGGGGGGG 000000
DD DD BB BB GG SS TT EE PP PP GG 00 00
DD DD BB BB GG SS TT EE PP PP GG 00 00
DD DD BB BB GG SS TT EE PP PP GG 00 00
DD DD BB BB GG SS TT EE PP PP GG 00 00
DD DD BBBBBBBB GG SSSSSS TT EEEEEEEE PPPPPPPP GG 00 00
DD DD BBBBBBBB GG SSSSSS TT EEEEEEEE PPPPPPPP GG 00 00
DD DD BB BB GG GGGGGG SS TT EE PP GG GGGGGG 00 00
DD DD BB BB GG GGGGGG SS TT EE PP GG GGGGGG 00 00
DD DD BB BB GG GG SS TT EE PP GG GG 00 00 ....
DD DD BB BB GG GG SS TT EE PP GG GG 00 00 ....
DDDDDDDD BBBBBBBB GGGGGG SSSSSSSS TT EEEEEEEEEE PP GGGGGG 000000 ....
DDDDDDDD BBBBBBBB GGGGGG SSSSSSSS TT EEEEEEEEEE PP GGGGGG 000000 ....
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 DBGSTEPGO (IDENT = 'V04-000') =
;     2          0002  1 BEGIN
;     3          0003  1 !
;     4          0004  1 !**************************************************************************
;     5          0005  1 !*                                                                        *
;     6          0006  1 !*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                               *
;     7          0007  1 !*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                 *
;     8          0008  1 !*  ALL RIGHTS RESERVED.                                                   *
;     9          0009  1 !*                                                                        *
;    10          0010  1 !*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
;    11          0011  1 !*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE  *
;    12          0012  1 !*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
;    13          0013  1 !*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
;    14          0014  1 !*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
;    15          0015  1 !*  TRANSFERRED.                                                           *
;    16          0016  1 !*                                                                        *
;    17          0017  1 !*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
;    18          0018  1 !*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
;    19          0019  1 !*  CORPORATION.                                                           *
;    20          0020  1 !*                                                                        *
;    21          0021  1 !*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
;    22          0022  1 !*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
;    23          0023  1 !*                                                                        *
;    24          0024  1 !*                                                                        *
;    25          0025  1 !**************************************************************************
;    26          0026  1 !
;    27          0027  1
;    28          0028  1 !++
;    29          0029  1 ! FACILITY:
;    30          0030  1 !
;    31          0031  1 !       DEBUG
;    32          0032  1 !
;    33          0033  1 ! ABSTRACT:
;    34          0034  1 !
;    35          0035  1 !       This module contains the command parse and execution networks to support
;    36          0036  1 !       the STEP and the GO commands.
;    37          0037  1 !       Parsing is done by means of ATN's. During parsing, a
;    38          0038  1 !       linked list known as the command execution tree is constructed. This
;    39          0039  1 !       tree contains components which represent keywords and operands of the
;    40          0040  1 !       user's input command. The command execution tree is passed to the command
;    41          0041  1 !       execution network as input.
;    42          0042  1 !
;    43          0043  1 ! ENVIRONMENT:
;    44          0044  1 !
;    45          0045  1 !       VAX/VMS
;    46          0046  1 !
;    47          0047  1 ! AUTHOR:
;    48          0048  1 !
;    49          0049  1 !       David Plummer
;    50          0050  1 !
;    51          0051  1 ! CREATION DATE:
;    52          0052  1 !
;    53          0053  1 !       9-Jul-80
;    54          0054  1 !
;    55          0055  1 ! VERSION:
;    56          0056  1 !
;    57          0057  1 !       V02.2-001
```

```
;    58          0058  1 ;
;    59          0059  1 ; MODIFIED BY:
;    60          0060  1 ;
;    61          0061  1 ;       Richard Title   15 SEP 1981
;    62          0062  1 ;
;    63          0063  1 ; REVISION HISTORY:
;    64          0064  1 ;
;    65          0065  1 ; 3.01  15-SEP-81       RT      Modified the STEP command to allow
;    66          0066  1 ;                               STEP/SOURCE and STEP/NOSOURCE
;    67          0067  1 ; 3.02  21-Dec-81       RT      Disallowed STEP from an exception break.
;    68          0068  1 ;--
```

```
;    70          0069  1 !
;    71          0070  1 ! TABLE OF CONTENTS:
;    72          0071  1 !
;    73          0072  1 !
;    74          0073  1 ! REQUIRE FILES:
;    75          0074  1 !
;    76          0075  1 REQUIRE 'SRC$:DBGPROLOG.REQ';
;    77          0209  1 LIBRARY 'LIB$:DBGGEN.L32';
;    78          0210  1
;    79          0211  1
;    80          0212  1 FORWARD ROUTINE
;    81          0213  1         DBG$NPARSE_STEP,                ! STEP parse network
;    82          0214  1         DBG$NEXECUTE_STEP,              ! STEP execution network
;    83          0215  1         DBG$NPARSE_GO,                  ! Parse network for GO
;    84          0216  1         DBG$NEXECUTE_GO;                ! Execution network for GO
;    85          0217  1
;    86          0218  1 ! EQUATED SYMBOLS:
;    87          0219  1 !
;    88          0220  1 LITERAL
;    89          0221  1
;    90          0222  1         ! Legal verb composites
;    91          0223  1         !
;    92          0224  1         GO_NOADDR = 1,
;    93          0225  1         GO_ADDR = 2,
;    94          0226  1
;    95          0227  1
;    96          0228  1         ! Legal adverb literals
;    97          0229  1         !
;    98          0230  1         ADVERB_LITERAL_LINE     = 1,
;    99          0231  1         ADVERB_LITERAL_OVER     = 2,
;   100          0232  1         ADVERB_LITERAL_NOSYSTEM = 3,
;   101          0233  1         ADVERB_LITERAL_SOURCE   = 4;
;   102          0234  1
;   103          0235  1
;   104          0236  1 ! EXTERNAL REFERNECES
;   105          0237  1 !
;   106          0238  1 EXTERNAL ROUTINE
;   107          0239  1         DBG$EVENT_SEMANTICS,            ! Event semantics
;   108          0240  1         DBG$EVENT_SYNTAX,               ! Event syntax (parser)
;   109          0241  1         DBG$GET_TEMPMEM,                ! Allocates listed dynamic storage
;   110          0242  1         DBG$IS_IT_ENTRY,                ! Checks for address = entry point
;   111          0243  1         DBG$NGET_ADDRESS,               ! Obtains an address value from an addr exp desc
;   112          0244  1         DBG$NMAKE_ARG_VECT,             ! Constructs a message argument vector
;   113          0245  1         DBG$NMATCH,                     ! Matches counted strings to input
;   114          0246  1         DBG$NNEXT_WORD,                 ! Obtains next word of input string
;   115          0247  1         DBG$NPARSE_ADDRESS,             ! Interface to Address Expression Interpreter
;   116          0248  1         DBG$NSAVE_DECIMAL_INTEGER,      ! Converts ascii input into an integer
;   117          0249  1         DBG$NSYNTAX_ERROR,              ! Formats a syntax error
;   118          0250  1         DBG$SET_STP_LVL,                ! Sets step structure pointer
;   119          0251  1         DBG$THREAD_RET;                 ! Address threaded breakpoints return to
;   120          0252  1
;   121          0253  1 EXTERNAL
;   122          0254  1         DBG$GB_UNHANDLED_EXC: BYTE,             ! Unhandled exception in user
;   123          0255  1                                                 !    program was just encountered
;   124          0256  1         DBG$GB_EXC_BRE_FLAG: BYTE,              ! TRUE during an exception break.
;   125          0257  1         DBG$GB_GO_ARG_FLAG: BYTE,               ! Flag saying whether GO has
;   126          0258  1                                                 ! an argument
```

```
;  127          0259  1          DBG$GB_STP_PTR :    REF EVENT$STEPPING_DESCRIPTOR, ! current stepping
;  128          0260  1          DBG$GB_TAKE_CMD : BYTE,                            ! Flag for taking commands
;  129          0261  1          DBG$GL_CONTEXT  : BITVECTOR,                       ! Context word
;  130          0262  1          DBG$GL_STEP_NUM,                                   ! Holds step number
;  131          0263  1          DBG$RUNFRAME : BLOCK [,BYTE];                      ! Current runframe
;  132          0264  1
;  133          0265  1
```

```
;  135     0266  1 GLOBAL ROUTINE DBG$NPARSE_STEP (INPUT_DESC, VERB_NODE, MESSAGE_VECT) =
;  136     0267  1 !
;  137     0268  1 ! FUNCTIONAL DESCRIPTION:
;  138     0269  1 !
;  139     0270  1 !       This routine comprises the ATN parse network for the STEP command.
;  140     0271  1 !       During processing, other routines are invoked to capture operands.
;  141     0272  1 !       This routine recognizes keywords and constructs a command execution tree
;  142     0273  1 !       to be used as input to the command execution network. Upon detection of
;  143     0274  1 !       errors, a message argument vector is constructed and returned.
;  144     0275  1 !
;  145     0276  1 ! FORMAL PARAMETERS:
;  146     0277  1 !
;  147     0278  1 !       INPUT_DESC      - A longword containing the address of a standard ascii
;  148     0279  1 !                         string descriptor representing the user's input
;  149     0280  1 !
;  150     0281  1 !       VERB_NODE       - A longword containing the address of the verb (head)
;  151     0282  1 !                         node of the command execution tree. The string corresponding
;  152     0283  1 !                         to the verb has already been processed upon entry.
;  153     0284  1 !
;  154     0285  1 !       MESSAGE_VECT    - The address of a longword to contain the address of a
;  155     0286  1 !                         message argument vector on errors
;  156     0287  1 !
;  157     0288  1 !
;  158     0289  1 ! IMPLICIT INPUTS:
;  159     0290  1 !
;  160     0291  1 !       NONE
;  161     0292  1 !
;  162     0293  1 ! IMPLICIT OUTPUTS:
;  163     0294  1 !
;  164     0295  1 !       On success, the entire command execution tree corresponding to the parsed
;  165     0296  1 !       STEP command is constructed.
;  166     0297  1 !
;  167     0298  1 !       On error, a message argument vector is constructed and returned.
;  168     0299  1 !
;  169     0300  1 ! ROUTINE VALUE:
;  170     0301  1 !
;  171     0302  1 !       An unsigned integer longword completion code
;  172     0303  1 !
;  173     0304  1 ! COMPLETION CODES:
;  174     0305  1 !
;  175     0306  1 !       STS$K_SUCCESS (1)       - Success. Input parsed and execution tree constructed.
;  176     0307  1 !
;  177     0308  1 !       STS$K_SEVERE  (4)       - Failure. Error detected and message argument
;  178     0309  1 !                                 vector constructed.
;  179     0310  1 !
;  180     0311  1 ! SIDE EFFECTS:
;  181     0312  1 !
;  182     0313  1 !       NONE
;  183     0314  1 !
;  184     0315  1 !
;  185     0316  2     BEGIN
;  186     0317  2     MAP
;  187     0318  2         VERB_NODE : REF DBG$VERB_NODE;
;  188     0319  2     VERB_NODE [DBG$B_VERB_COMPOSITE] = EVENT$K_STEP;
;  189     0320  2     RETURN DBG$EVENT_SYNTAX (.INPUT_DESC,
;  190     0321  2                              .VERB_NODE,
;  191     0322  2                              .MESSAGE_VECT
```

```
;  192        0323  2                              );
;  193        0324  1     END;

                                              .TITLE  DBGSTEPGO
                                              .IDENT  \V04-000\

                                              .EXTRN  DBG$EVENT_SEMANTICS
                                              .EXTRN  DBG$EVENT_SYNTAX
                                              .EXTRN  DBG$GET_TEMPMEM
                                              .EXTRN  DBG$IS_IT_ENTRY
                                              .EXTRN  DBG$NGET_ADDRESS
                                              .EXTRN  DBG$NMAKE_ARG_VECT
                                              .EXTRN  DBG$NMATCH, DBG$NNEXT_WORD
                                              .EXTRN  DBG$NPARSE_ADDRESS
                                              .EXTRN  DBG$NSAVE_DECIMAL_INTEGER
                                              .EXTRN  DBG$NSYNTAX_ERROR
                                              .EXTRN  DBG$SET_STP_LVL
                                              .EXTRN  DBG$THREAD_RET, DBG$GB_UNHANDLED_EXC
                                              .EXTRN  DBG$GB_EXC_BRE_FLAG
                                              .EXTRN  DBG$GB_GO_ARG_FLAG
                                              .EXTRN  DBG$GB_STP_PTR, DBG$GB_TAKE_CMD
                                              .EXTRN  DBG$GL_CONTEXT, DBG$GL_STEP_NUM
                                              .EXTRN  DBG$RUNFRAME

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

                                   0000 00000  .ENTRY  DBG$NPARSE_STEP, Save nothing           ; 0266
   08   BC        08       08        0C  F0 00002  INSV    #12, #8, #8, @VERB_NODE             ; 0319
                           7E    08  AC  7D 00008  MOVQ    VERB_NODE, -(SP)                    ; 0321
                                 04  AC  DD 0000C  PUSHL   INPUT_DESC                          ; 0320
              00000000G    00        03  FB 0000F  CALLS   #3, DBG$EVENT_SYNTAX                ;
                                     04 00016  RET                                             ; 0324

; Routine Size:  23 bytes,    Routine Base:  DBG$CODE + 0000
```

```
;  195     0325  1 GLOBAL ROUTINE DBG$NEXECUTE_STEP (VERB_NODE, MESSAGE_VECT) =
;  196     0326  1 !
;  197     0327  1 ! FUNCTIONAL DESCRIPTION:
;  198     0328  1 !
;  199     0329  1 !       This routine accepts as input the command execution tree constructed
;  200     0330  1 !       by the parse network and performs semantic actions corresponding to the
;  201     0331  1 !       parsed input STEP command. If the action cannot be performed, a message
;  202     0332  1 !       argument vector is constructed and returned. Actual stepping is NOT
;  203     0333  1 !       performed in this network. This is handled by the DEBUG monitor.
;  204     0334  1 !
;  205     0335  1 !       This routine manipulates the dbg$gb_def_stp structure, as well as the
;  206     0336  1 !       step level pointer.
;  207     0337  1 !
;  208     0338  1 ! FORMAL PARAMETERS:
;  209     0339  1 !
;  210     0340  1 !       VERB_NODE       - A longword containing the address of the verb (head)
;  211     0341  1 !                         node of the command execution tree
;  212     0342  1 !
;  213     0343  1 !       MESSAGE_VECT    - The address of a longword to contain the address of
;  214     0344  1 !                         a standard message argument vector upon detection of
;  215     0345  1 !                         of errors
;  216     0346  1 !
;  217     0347  1 ! IMPLICIT INPUTS:
;  218     0348  1 !
;  219     0349  1 !       The entire linked list command execution tree as pointed to by the
;  220     0350  1 !       verb node.
;  221     0351  1 !
;  222     0352  1 ! IMPLICIT OUTPUTS:
;  223     0353  1 !
;  224     0354  1 !       On error, a message argument vector is constructed and returned.
;  225     0355  1 !
;  226     0356  1 ! ROUTINE VALUE:
;  227     0357  1 !
;  228     0358  1 !       An unsigned integer longword completion code
;  229     0359  1 !
;  230     0360  1 ! COMPLETION CODES:
;  231     0361  1 !
;  232     0362  1 !       STS$K_SUCCESS (1)       - Success. A STEP will be performed.
;  233     0363  1 !
;  234     0364  1 !       STS$K_SEVERE  (4)       - Failure. The STEP will not be performed.
;  235     0365  1 !                                 Message argument vector returned.
;  236     0366  1 !
;  237     0367  1 ! SIDE EFFECTS:
;  238     0368  1 !
;  239     0369  1 !       The DEBUG monitor will be informed to perform a step.
;  240     0370  1 !
;  241     0371  1 !--
;  242     0372  2     BEGIN
;  243     0373  2
;  244     0374  2
;  245     0375  2     ! If we are continuing from an unhandled exception, signal an
;  246     0376  2     ! informational message to that effect.
;  247     0377  2     !
;  248     0378  2     IF .DBG$GB_UNHANDLED_EXC
;  249     0379  2     THEN
;  250     0380  3         BEGIN
;  251     0381  3         SIGNAL(DBG$_CONFROMEXC);
```

```
;  252          0382  !          DBG$GB_UNHANDLED_EXC = 0;
;  253          0383  !          END;
;  254          0384  !
;  255          0385  2      ! Call the event semantics routine.
;  256          0386  2      !
;  257          0387  2      RETURN DBG$EVENT_SEMANTICS (.VERB_NODE, .MESSAGE_VECT);
;  258          0388  2
;  259          0389  1      END;


                                        0004 00000          .ENTRY  DBG$NEXECUTE_STEP, Save R2                      ; 0325
                                52 00000000G   00  9E 00002  MOVAB   DBG$GB_UNHANDLED_EXC, R2                        ;
                                0F             62  E9 00009  BLBC   DBG$GB_UNHANDLED_EXC, 1$                       ; 0378
                                   00028783    8F  DD 0000C  PUSHL  #165763                                         ; 0381
                    00000000G   00             01  FB 00012  CALLS  #1, LIB$SIGNAL                                ;
                                               62  94 00019  CLRB   DBG$GB_UNHANDLED_EXC                           ; 0382
                                7E        04   AC  7D 0001B 1$:    MOVQ   VERB_NODE, -(SP)                         ; 0387
                    00000000G   00             02  FB 0001F  CALLS  #2, DBG$EVENT_SEMANTICS                       ;
                                               04 00026    RET                                                     ; 0389

; Routine Size:  39 bytes,    Routine Base:  DBG$CODE + 0017
```

```
;  261     0390  1 GLOBAL ROUTINE DBG$NPARSE_GO (INPUT_DESC, VERB_NODE, MESSAGE_VECT) =
;  262     0391  1 !
;  263     0392  1 ! FUNCTIONAL DESCRIPTION:
;  264     0393  1 !
;  265     0394  1 !       This routine comprises the ATN parse network for the GO verb. During
;  266     0395  1 !       processing of the input command, a command execution tree containing the
;  267     0396  1 !       keywords and operands of the input command is constructed. On a non-
;  268     0397  1 !       successful parse, a message argument vector is constructed and returned.
;  269     0398  1 !
;  270     0399  1 ! FORMAL PARAMETERS:
;  271     0400  1 !
;  272     0401  1 !       INPUT_DESC      - A longword containing the address of a standard ascii
;  273     0402  1 !                         string descriptor representing the user's input command
;  274     0403  1 !
;  275     0404  1 !       VERB_NODE       - A longword containing the address of the verb (head)
;  276     0405  1 !                         node of the command execution tree.
;  277     0406  1 !
;  278     0407  1 !       MESSAGE_VECT    - The address of a longword to contain the address of a
;  279     0408  1 !                         message argument vector for errors
;  280     0409  1 !
;  281     0410  1 ! IMPLICIT INPUTS:
;  282     0411  1 !
;  283     0412  1 !       NONE
;  284     0413  1 !
;  285     0414  1 ! IMPLICIT OUTPUTS:
;  286     0415  1 !
;  287     0416  1 !       On success, the command execution tree is constructed.
;  288     0417  1 !
;  289     0418  1 !       On failure, a message argument vector is constructed and returned.
;  290     0419  1 !
;  291     0420  1 ! ROUTINE VALUE:
;  292     0421  1 !
;  293     0422  1 !       An unsigned integer longword completion code
;  294     0423  1 !
;  295     0424  1 ! COMPLETION CODES:
;  296     0425  1 !
;  297     0426  1 !       STS$K_SUCCESS (1)       - Success. Input parsed and execution tree constructed.
;  298     0427  1 !
;  299     0428  1 !       STS$K_SEVERE  (4)       - Failure. Error detected. Message argument vector
;  300     0429  1 !                                 constructed and returned.
;  301     0430  1 !
;  302     0431  1 ! SIDE EFFECTS:
;  303     0432  1 !
;  304     0433  1 !       NONE
;  305     0434  1 !
;  306     0435  1
;  307     0436  2     BEGIN
;  308     0437  2
;  309     0438  2     MAP
;  310     0439  2         VERB_NODE : REF dbg$verb_node;
;  311     0440  2
;  312     0441  2     BIND
;  313     0442  2         DBG$CS_CR = UPLIT BYTE (1, dbg$k_car_return);
;  314     0443  2
;  315     0444  2     LOCAL
;  316     0445  2         STATUS,                                 ! Holds return status
;  317     0446  2         NOUN_NODE : REF dbg$noun_node;          ! Will contain GO address, if any given
```

```
2
2
2   ! The GO has already been accepted. Check for GO <CR>.
2   !
2   IF dbg$nmatch (.input_desc, dbg$cs_cr, 1)
2   THEN
3       BEGIN
3
3       ! Nothing left. Set the appropriate verb composite and return
3       !
3       verb_node [dbg$b_verb_composite] = go_noaddr;
3       RETURN sts$k_success;
2       END;
2
2
2   ! There is some input left.  Try to parse the address expression.
2   !
2   noun_node = dbg$get_tempmem(dbg$k_noun_node_size);
2   VERB_NODE [DBG$L_VERB_OBJECT_PTR] = .NOUN_NODE;
3   IF NOT (STATUS = DBG$NPARSE_ADDRESS(.INPUT_DESC,
3                       NOUN_NODE [DBG$L_NOUN_VALUE], DBG$K_DEFAULT,
3                       TOKEN$K_TERM_NONE, .MESSAGE_VECT))
2   THEN
3       BEGIN
3
3
3       ! We are responsible for syntax error on STS$K_WARNING.
3       !
3       IF .status EQL sts$k_warning
3       THEN
3           .message_vect = dbg$nsyntax_error (dbg$nnext_word (.input_desc));
3
3       RETURN sts$k_severe;
2       END;
2
2
2   ! Set the correct verb composite
2   !
2   verb_node [dbg$b_verb_composite] = go_addr;
2
2   RETURN sts$k_success;
2
1   END;


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

             0D  01  00000 P.AAA:        .BYTE   1, 13                                   ;

                         DBG$CS_CR=              P.AAA

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

               0004 00000                .ENTRY  DBG$NPARSE_GO, Save R2                  ; 0390
             01  DD 00002                PUSHL   #1                                      ; 0451
```

```
                                00000000'  EF  9F 00004          PUSHAB  DBG$CS_CR                                ;
                                      04   AC  DD 0000A          PUSHL   INPUT_DESC                               ;
                00000000G     00         03  FB 0000D          CALLS   #3, DBG$NMATCH                           ;
                              08         50  E9 00014          BLBC    R0, 1$                                   ;
      08  BC            08    08         01  F0 00017          INSV    #1, #8, #8, @VERB_NODE                   ; 0457
                                       49  11 0001D          BRB     4$                                       ; 0458
                                       04  DD 0001F  1$:      PUSHL   #4                                       ; 0464
                00000000G     00         01  FB 00021          CALLS   #1, DBG$GET_TEMPMEM
                              52      08   AC  D0 00028          MOVL    VERB_NODE, R2                            ; 0465
                         08   A2         50  D0 0002C          MOVL    NOUN_NODE, 8(R2)
                                      0C   AC  DD 00030          PUSHL   MESSAGE_VECT                             ; 0468
                              7E         01  7D 00033          MOVQ    #1, -(SP)                                ; 0467
                                       50  DD 00036          PUSHL   NOUN_NODE
                                      04   AC  DD 00038          PUSHL   INPUT_DESC
                00000000G     00         05  FB 0003B          CALLS   #5, DBG$NPARSE_ADDRESS
                              1F         50  E8 00042          BLBS    STATUS, 3$
                                       50  D5 00045          TSTL    STATUS                                   ; 0475
                                       17  12 00047          BNEQ    2$
                                      04   AC  DD 00049          PUSHL   INPUT_DESC                               ; 0477
                00000000G     00         01  FB 0004C          CALLS   #1, DBG$NNEXT_WORD
                                       50  DD 00053          PUSHL   R0
                00000000G     00         01  FB 00055          CALLS   #1, DBG$NSYNTAX_ERROR
                         0C   BC         50  D0 0005C          MOVL    R0, @MESSAGE_VECT
                              50         04  D0 00060  2$:      MOVL    #4, R0                                   ; 0479
                                       04  00063              RET
                         01   A2         02  90 00064  3$:      MOVB    #2, 1(R2)                                ; 0485
                              50         01  D0 00068  4$:      MOVL    #1, R0                                   ; 0487
                                       04  0006B              RET                                              ; 0489

; Routine Size:  108 bytes,    Routine Base:  DBG$CODE + 003E
```

```
;  362     0490  1 GLOBAL ROUTINE DBG$NEXECUTE_GO (VERB_NODE, MESSAGE_VECT) =
;  363     0491  1 !
;  364     0492  1 ! FUNCTIONAL DESCRIPTION:
;  365     0493  1 !
;  366     0494  1 !       This routine accepts the command execution tree constructed by the parse
;  367     0495  1 !       network and performs the semantic actions associated with the parsed
;  368     0496  1 !       GO command as given by the user. The actual GO is not performed by this
;  369     0497  1 !       routine. Rather the DEBUG monitor is informed to perform a GO.
;  370     0498  1 !
;  371     0499  1 ! FORMAL PARAMETERS:
;  372     0500  1 !
;  373     0501  1 !       VERB_NODE       - A longword containing the address of the verb (head)
;  374     0502  1 !                         node of the command execution tree
;  375     0503  1 !
;  376     0504  1 !       MESSAGE_VECT    - The address of a longword to contain the address of a
;  377     0505  1 !                         standard message argument vector upon detection of
;  378     0506  1 !                         errors.
;  379     0507  1 !
;  380     0508  1 ! IMPLICIT INPUTS:
;  381     0509  1 !
;  382     0510  1 !       The entire command execution tree linked list pointed to by the verb node.
;  383     0511  1 !
;  384     0512  1 ! IMPLICIT OUTPUTS:
;  385     0513  1 !
;  386     0514  1 !       NONE
;  387     0515  1 !
;  388     0516  1 ! ROUTINE VALUE:
;  389     0517  1 !
;  390     0518  1 !       An unsigned integer longword completion code
;  391     0519  1 !
;  392     0520  1 ! COMPLETION CODES:
;  393     0521  1 !
;  394     0522  1 !       STS$K_SUCCESS (1)       - Success. The GO command will be executed.
;  395     0523  1 !
;  396     0524  1 !       STS$K_SEVERE  (4)       - Failure. THE GO will not executed. Message argument
;  397     0525  1 !                                 vector constructed and returned.
;  398     0526  1 !
;  399     0527  1 ! SIDE EFFECTS:
;  400     0528  1 !
;  401     0529  1 !       Semantic actions coresponding to the execution of the GO command are taken.
;  402     0530  1 !
;  403     0531  1 !
;  404     0532  1
;  405     0533  2     BEGIN
;  406     0534  2
;  407     0535  2     MAP
;  408     0536  2         VERB_NODE       : REF dbg$verb_node;
;  409     0537  2
;  410     0538  2     BUILTIN
;  411     0539  2         PROBER;                                 ! Probes read access
;  412     0540  2
;  413     0541  2     LOCAL
;  414     0542  2         OLD_PC          : REF VECTOR [, WORD],  ! Used to access instruction
;  415     0543  2         NEW_PC,                                 ! Starting PC value
;  416     0544  2         NOUN_NODE       : REF dbg$noun_node,    ! Noun node
;  417     0545  2         ADDRESS         : VECTOR [2],           ! Address contained by addr exp desc
;  418     0546  2         TYPE;                                   ! Types of object described by addr exp desc
```

```
;  419          0547  2
;  420          0548  2      ! Set up to do the go depending on whether there is a specified start address or not
;  421          0549  2      !
;  422          0550  2      IF .verb_node [dbg$b_verb_composite] EQL go_addr
;  423          0551  2      THEN
;  424          0552  3          BEGIN
;  425          0553  3
;  426          0554  3          ! Recover the noun node and obtain a PC value
;  427          0555  3          !
;  428          0556  3          noun_node = .verb_node [dbg$l_verb_object_ptr];
;  429          0557  3          IF NOT dbg$nget_address (.noun_node [dbg$l_noun_value], address [0],
;  430          0558  3                                   type, false, .message_vect)
;  431          0559  3          THEN
;  432          0560  4              RETURN sts$k_severe;
;  433          0561  3
;  434          0562  3
;  435          0563  3          ! The GO address is now contained in address [0]. We don't care about the type.
;  436          0564  3          !
;  437          0565  3          new_pc = .address [0];
;  438          0566  3
;  439          0567  3
;  440          0568  3          ! Check for an entry mask
;  441          0569  3          !
;  442          0570  3          IF dbg$is_it_entry (.new_pc)
;  443          0571  3          THEN
;  444          0572  4              new_pc = .new_pc + 2;
;  445          0573  3
;  446          0574  3
;  447          0575  3          ! Check for read access before we load the address into the user's PC
;  448          0576  3          !
;  449          0577  3          IF NOT PROBER (%REF (0), %REF (1), .new_pc)
;  450          0578  3          THEN
;  451          0579  4              BEGIN
;  452          0580  4              .message_vect = dbg$nmake_arg_vect (dbg$_badstartpc, 1, .new_pc);
;  453          0581  4              RETURN sts$k_severe;
;  454          0582  3              END;
;  455          0583  3
;  456          0584  3
;  457          0585  3          ! For GO xxx, we load the new PC into the runframe and clear the FPD bit
;  458          0586  3          ! in the PSL. We must also check for threaded code.
;  459          0587  3          !
;  460          0588  3          old_pc = .dbg$runframe [dbg$l_user_pc];
;  461          0589  3
;  462          0590  3          IF .dbg$gl_context [dbg$k_thread]       ! Threaded code
;  463          0591  3          THEN
;  464          0592  4              BEGIN
;  465          0593  4              IF .old_pc [0] EQL %X'9B17' ! JMP @(R11)+
;  466          0594  4                  OR
;  467          0595  4                 .old_pc EQL DBG$THREAD_RET ! Address of returns from threaded BPTs
;  468          0596  4              THEN
;  469          0597  4                  dbg$runframe [dbg$l_user_r11] = .new_pc
;  470          0598  4              ELSE
;  471          0599  5                  BEGIN
;  472          0600  5                  .message_vect = dbg$nmake_ar _vect (dbg$_notlinbnd);
;  473          0601  5                  RETURN sts$k_severe;
;  474          0602  4                  END;
;  475          0603  4              END
```

```
;  476          0604  3          ELSE
;  477          0605  3              dbg$runframe [dbg$l_user_pc] = .new_pc;
;  478          0606  3
;  479          0607  3
;  480          0608  3          ! Clear the FPD bit in the PSL
;  481          0609  3          !
;  482          0610  3          dbg$runframe [dbg$l_user_psl] = .dbg$runframe [dbg$l_user_psl]
;  483          0611  3                                                  AND
;  484          0612  3                                                %X'F7FFFFFF';
;  485          0613  2          END;
;  486          0614  2
;  487          0615  2      ! Check PC for read access
;  488          0616  2      !
;  489          0617  2      new_pc = .dbg$runframe [dbg$l_user_pc];
;  490          0618  2
;  491          0619  2      IF NOT PROBER (%REF (0), %REF (1), .new_pc)
;  492          0620  2      THEN
;  493          0621  2          BEGIN
;  494          0622  3          .message_vect = dbg$nmake_arg_vect (dbg$_badstartpc, 1, .new_pc);
;  495          0623  3          RETURN sts$k_severe;
;  496          0624  3          END;
;  497          0625  2
;  498          0626  2      ! Inform the monitor to start the user program
;  499          0627  2      !
;  500          0628  2      DBG$GB_TAKE_CMD = FALSE;
;  501          0629  2
;  502          0630  2      ! Set the global flag saying whether GO has an argument.
;  503          0631  2      !
;  504          0632  2      IF .VERB_NODE[DBG$B_VERB_COMPOSITE] EQL GO_ADDR
;  505          0633  2      THEN
;  506          0634  2          DBG$GB_GO_ARG_FLAG = TRUE
;  507          0635  2      ELSE
;  508          0636  2          DBG$GB_GO_ARG_FLAG = FALSE;
;  509          0637  2
;  510          0638  2      ! If we are continuing from an unhandled exception, signal an
;  511          0639  2      ! informational message to that effect.
;  512          0640  2      !
;  513          0641  2      IF .DBG$GB_UNHANDLED_EXC
;  514          0642  2      THEN
;  515          0643  3          BEGIN
;  516          0644  3          SIGNAL(DBG$_CONFROMEXC);
;  517          0645  3          DBG$GB_UNHANDLED_EXC = 0;
;  518          0646  2          END;
;  519          0647  2
;  520          0648  2      RETURN STS$K_SUCCESS;
;  521          0649  2
;  522          0650  1      END;


                                00FC 00000          .ENTRY  DBG$NEXECUTE_GO, Save R2,R3,R4,R5,R6,R7     ; 0490
```

```
                57 00000000G   00   9E 00002          MOVAB   DBG$GB_UNHANDLED_EXC, R7                  ;
                56 00000000G   00   9E 00009          MOVAB   DBG$GB_GO_ARG_FLAG, R6                   ;
                55 00000000G   00   9E 00010          MOVAB   DBG$NMAKE_ARG_VECT, R5                   ;
                54 00000000G   00   9E 00017          MOVAB   DBG$RUNFRAME+64, R4                      ;
                5E             0C   C2 0001E          SUBL2   #12, SP                                  ;
                50         04  AC   D0 00021          MOVL    VERB_NODE, R0                            ; 0550
                               53   D4 00025          CLRL    R3                                       ;
                02         01  A0   91 00027          CMPB    1(R0), #2                                ;
                               6C   12 0002B          BNEQ    6$                                       ;
                               53   D6 0002D          INCL    R3                                       ;
                50         08  A0   D0 0002F          MOVL    8(R0), NOUN_NODE                         ; 0556
                               08   AC   DD 00033      PUSHL   MESSAGE_VECT                             ; 0558
                               7E   D4 00036          CLRL    -(SP)                                    ; 0557
                               08   AE   9F 00038      PUSHAB  TYPE                                     ;
                               10   AE   9F 0003B      PUSHAB  ADDRESS                                  ;
                               60   DD 0003E          PUSHL   (NOUN_NODE)                              ;
    00000000G  00                   05   FB 00040      CALLS   #5, DBG$NGET_ADDRESS                     ;
                69                 50   E9 00047      BLBC    R0, 9$                                   ;
                52         04  AE   D0 0004A          MOVL    ADDRESS, NEW_PC                          ; 0565
                               52   DD 0004E          PUSHL   NEW_PC                                   ; 0570
    00000000G  00                   01   FB 00050      CALLS   #1, DBG$IS_IT_ENTRY                      ;
                03                 50   E9 00057      BLBC    R0, 1$                                   ;
                52                 02   C0 0005A      ADDL2   #2, NEW_PC                               ; 0572
            62  01                 00   0C 0005D  1$:  PROBER  #0, #1, (NEW_PC)                        ; 0577
                               3F   13 00061          BEQL    7$                                       ;
                51                 64   D0 00063      MOVL    DBG$RUNFRAME+64, OLD_PC                  ; 0588
            24 00000000G  00       04   E1 00066      BBC     #4, DBG$GL_CONTEXT+2, 4$                 ; 0590
                  9B17     8F      61   B1 0006E      CMPW    (OLD_PC), #39703                         ; 0593
                               0C   13 00073          BEQL    2$                                       ;
                50 00000000G   00   9E 00075          MOVAB   DBG$THREAD_RET, R0                       ; 0595
                50             51   D1 0007C          CMPL    OLD_PC, R0                               ;
                               06   12 0007F          BNEQ    3$                                       ;
                   F0     A4       52   D0 00081  2$:  MOVL    NEW_PC, DBG$RUNFRAME+48                  ; 0597
                               0E   11 00085          BRB     5$                                       ;
                       000281B8    8F   DD 00087  3$:  PUSHL   #164280                                  ; 0600
                65                 01   FB 0008D      CALLS   #1, DBG$NMAKE_ARG_VECT                   ;
                               1D   11 00090          BRB     8$                                       ;
                64                 52   D0 00092  4$:  MOVL    NEW_PC, DBG$RUNFRAME+64                  ; 0605
                   07     A4       08   8A 00095  5$:  BICB2   #8, DBG$RUNFRAME+71                    ; 0611
                52                 64   D0 00099  6$:  MOVL    DBG$RUNFRAME+64, NEW_PC                  ; 0618
            62  01                 00   0C 0009C      PROBER  #0, #1, (NEW_PC)                         ; 0620
                               15   12 000A0          BNEQ    10$                                      ;
                               52   DD 000A2  7$:     PUSHL   NEW_PC                                   ; 0623
                               01   DD 000A4          PUSHL   #1                                       ;
                       000281E0    8F   DD 000A6      PUSHL   #164320                                  ;
                65                 03   FB 000AC      CALLS   #3, DBG$NMAKE_ARG_VECT                   ;
                   08     BC       50   D0 000AF  8$:  MOVL    R0, @MESSAGE_VECT                        ;
                50                 04   D0 000B3  9$:  MOVL    #4, R0                                   ; 0624
                               04   000B6          RET                                              ;
                   00000000G   00   94 000B7  10$: CLRB    DBG$GB_TAKE_CMD                          ; 0630
                05                 53   E9 000BD      BLBC    R3, 11$                                  ; 0635
                66                 01   90 000C0      MOVB    #1, DBG$GB_GO_ARG_FLAG                   ; 0637
                               02   11 000C3          BRB     12$                                      ;
                               66   94 000C5  11$: CLRB    DBG$GB_GO_ARG_FLAG                       ; 0639
                0F                 67   E9 000C7  12$: BLBC    DBG$GB_UNHANDLED_EXC, 13$                ; 0645
                       00028783    8F   DD 000CA      PUSHL   #165763                                  ; 0648
    00000000G   00                 01   FB 000D0      CALLS   #1, LIB$SIGNAL                           ;
```

```
                                 67  94 000D7         CLRB    DBG$GB_UNHANDLED_EXC                       ; 0649
                   50            01  D0 000D9 13$:    MOVL    #1, R0                                     ; 0652
                                     04 000DC         RET                                                ; 0654

; Routine Size:  221 bytes,    Routine Base:  DBG$CODE + 00AA


;  527          0655  1
;  528          0656  0 END ELUDOM



                                                      .EXTRN  LIB$SIGNAL
```

```
;                              PSECT SUMMARY
;
;      Name                   Bytes                     Attributes
;
;   DBG$CODE                    391  NOVEC,NOWRT,  RD ,  EXE,  SHR,  LCL,  REL,  CON,  PIC,ALIGN(0)
;   DBG$PLIT                      2  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        3         0      1000        00:01.9
;   _$255$DUA28:[DEBUG.OBJ]STRUCDEF.L32;1           32        0         0         7        00:00.1
;   _$255$DUA28:[DEBUG.OBJ]DBGLIB.L32;1           1545       33         2        97        00:02.0
;   _$255$DUA28:[DEBUG.OBJ]DSTRECRDS.L32;1
;                                                  418        0         0        31        00:00.3
;   _$255$DUA28:[DEBUG.OBJ]DBGMSG.L32;1            386        7         1        22        00:00.3
;   _$255$DUA28:[DEBUG.OBJ]DBGGEN.L32;1            150        0         0        12        00:00.3
```

```
;                             COMMAND QUALIFIERS

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

; Size:          391 code + 2 data bytes
; Run Time:         00:13.6
; Elapsed Time:     00:16.4
; Lines/CPU Min:     2887
; Lexemes/CPU-Min:  7347
; Memory Used:  109 pages
; Compilation Complete
```

DBGSSV
LIS

DBGSTEPGO
LIS

DBGSTART
LIS

DBGSOURCE
LIS

DBGSYMBOL
LIS

DBGSTO
LIS

DBGSYMBLZ
LIS