```
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    GGGGGGGG  EEEEEEEEEE  NN       NN
DDDDDDDD    BBBBBBBB      GGGGGGGG    GGGGGGGG  EEEEEEEEEE  NN       NN
DD      DD  BB      BB  GG          GG          EE          NN       NN
DD      DD  BB      BB  GG          GG          EE          NN       NN
DD      DD  BB      BB  GG          GG          EE          NNNN     NN
DD      DD  BB      BB  GG          GG          EE          NNNN     NN
DD      DD  BBBBBBBB    GG          GG          EEEEEEEE    NN  NN   NN
DD      DD  BBBBBBBB    GG          GG          EEEEEEEE    NN  NN   NN
DD      DD  BB      BB  GG  GGGGGG  GG  GGGGGG  EE          NN    NNNN
DD      DD  BB      BB  GG  GGGGGG  GG  GGGGGG  EE          NN    NNNN
DD      DD  BB      BB  GG      GG  GG      GG  EE          NN      NN      ....
DD      DD  BB      BB  GG      GG  GG      GG  EE          NN      NN      ....
DDDDDDDD    BBBBBBBB      GGGGGG      GGGGGG    EEEEEEEEEE  NN      NN      ....
DDDDDDDD    BBBBBBBB      GGGGGG      GGGGGG    EEEEEEEEEE  NN      NN      ....

LL            IIIIII      SSSSSSS
LL            IIIIII      SSSSSSS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II        SSSSS
LL              II        SSSSS
LL              II             SS
LL              II             SS
LL              II             SS
LL              II             SS
LLLLLLLLLL    IIIIII    SSSSSSS
LLLLLLLLLL    IIIIII    SSSSSSS
```

```
;  0001  0      !       DBGGEN.REQ - Require file for VAX/VMS DEBUG facility.
;  0002  0      !
;  0003  0      ! Version:      'V04-000'
;  0004  0      !
;  0005  0      !****************************************************************************
;  0006  0      !*                                                                          *
;  0007  0      !*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
;  0008  0      !*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
;  0009  0      !*  ALL RIGHTS RESERVED.                                                    *
;  0010  0      !*                                                                          *
;  0011  0      !*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
;  0012  0      !*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE  *
;  0013  0      !*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
;  0014  0      !*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
;  0015  0      !*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
;  0016  0      !*  TRANSFERRED.                                                            *
;  0017  0      !*                                                                          *
;  0018  0      !*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
;  0019  0      !*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
;  0020  0      !*  CORPORATION.                                                            *
;  0021  0      !*                                                                          *
;  0022  0      !*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
;  0023  0      !*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
;  0024  0      !*                                                                          *
;  0025  0      !*                                                                          *
;  0026  0      !****************************************************************************
;  0027  0      !
;  0028  0      !++
;  0029  0      !       Modified by:
;  0030  0      !               Bruce Olsen, 30 Aug 1979
;  0031  0      !               Richard Title, 20 Aug 1981
;  0032  0      !
;  0033  0      !       29-Jun-80       DLP     Added literals for PL/I, PASCAL, and C
;  0034  0      !       29-JAN-81       DLP     chs_per_lexeme is now 50
;  0035  0      !       20-AUG-81        RT     Changed step_lvl_size and added step_source
;  0036  0      !                               because of new command
;  0037  0      !                               SET STEP SOURCE
;  0038  0      !       20-JAN-82        RT     Moved CIS declarations from here to DBGLIB.
;  0039  0      !       06-MAY-82        RT     Added structures for DEFINE
;  0040  0      !--
```

```
;    0041  0      MACRO
;    0042  0              NULL_POS_SIZE   =0, 0, 0%,      ! Null PSE for undotted references to
;    0043  0                                              !    blocks
;    0044  0              OPERAND_MODE    =0, 4, 4, 0%,   ! Mode part of an operand
;    0045  0              OPERAND_VALUE   =0, 0, 4, 0%;   ! Value part of an operand
;    0046  0
;    0047  0      LITERAL
;    0048  0              DBG_FAC_CODE    = 2 * 65536,    ! DEBUG facility code
;    0049  0              THREAD_CODE     =1,             ! Conditional switch for threaded code
;    0050  0              FATAL_BIT       =4,             ! Mask for fatal bit in error codes
;    0051  0              ADD_THE_OFFSET  =1,             ! Add offset to value
;    0052  0              SUB_THE_OFFSET  =0,             ! Subtract offset from value
;    0053  0
;    0054  0
;    0055  0              ! The next three literals used to be defined in SYSLIT.REQ
;    0056  0              !
;    0057  0              NO_OVERRIDE     = 0,            ! Use current length and radix
;    0058  0              TTY_OUT_WIDTH   = 132,          ! Width of terminal line
;    0059  0              TTY_ASCII_LEN   = 80,           ! Max size of character variable
;    0060  0
;    0061  0
;    0062  0              ! Values for register name tables
;    0063  0              !
;    0064  0              REGISTER_COUNT  =17,            ! Seventeen registers counting PSL
;    0065  0
;    0066  0
;    0067  0              ! size parameters
;    0068  0              !
;    0069  0              MAX_ASCII_LENG  = 4096,         ! Largest N in "ASCII:N"
;    0070  0              MAX_STACK_PTR   =20,            ! Depth of the parser's parse stack.
;    0071  0              NO_OF_INP_CHARS =132,           ! Max number of characters in input line
;    0072  0                                              ! ***Must be divisible by 4***
;    0073  0              CHS_PER_LEXEME  =50,            ! Max number of characters in a single lexeme
;    0074  0                                              ! ***Must be divisible by 4***
;    0075  0              NO_OF_TOKENS    =30,            ! Max number of tokens permitted
;    0076  0                                              ! ***Must be an even number***
;    0077  0              MAX_BYTE_INT    = 127,          ! Largest byte integer
;    0078  0              ALL_BPTS        =-1,            ! All breakpoints in chain
;    0079  0              BPT_INSTRUCTION =%X'03',        ! Opcode assignment for bpt instruction (srm revision 3)
;    0080  0              BREAK_POINT     = 1,            ! Table entry is breakpoint
;    0081  0              TRACE_POINT     = 2,            ! Table entry is tracepoint
;    0082  0              WATCH_POINT     = 3,            ! Table entry is watchpoint
;    0083  0
;    0084  0              WRITE_WATCH     = 1,            ! Watchpoint is for write access
;    0085  0              READ_WRIT_WATCH = 2,            ! Watchpoint is for read/write access
;    0086  0              EXECUTE_WATCH   = 3,            ! This is break or trace point
;    0087  0
;    0088  0              NUM_MAX_LENGTH  =11,            ! Maximum number of characters per numeric string
;    0089  0              SYM_MAX_LENGTH  =31,            ! Maximum number of characters per symbol
;    0090  0              UPPER_CASE_DIF  ='A' - 'A',     ! Difference between ASCII representation of upper and lower case
;    0091  0              ASCII_OFFSET    =%O'60',        ! Offset from numeric value to ASCII value
;    0092  0
;    0093  0
;    0094  0              ! ASCII character representations
;    0095  0              !
;    0096  0              LINEFEED        =%O'12',        ! ASCII representation of linefeed
;    0097  0              CARRIAGE_RET    =%O'15',        ! ASCII representation of carriage return
```

```
;    0098  0          ASC_AT_SIGN     =%ASCII '@',    ! ASCII representation of an at sign
;    0099  0          ASC_CLOS_PAREN  =%ASCII ')',    ! ASCII representation of closed parenthesis
;    0100  0          ASC_COMMA       =%ASCII ',',    ! ASCII representation of a comma
;    0101  0          ASC_DOUB_QUOTE  =%ASCII '""',   ! ASCII representation of a double quote
;    0102  0          ASC_MINUS       =%ASCII '-',    ! ASCII representation of a minus sign
;    0103  0          ASC_OPEN_PAREN  =%ASCII '(',    ! ASCII representation of open parenthesis
;    0104  0          ASC_PERCENT     =%ASCII '%',    ! ASCII representation of a percent sign
;    0105  0          ASC_PERIOD      =%ASCII '.',    ! ASCII representation of a period
;    0106  0          ASC_PLUS        =%ASCII '+',    ! ASCII representation of a plus sign
;    0107  0          ASC_POUNDS      =%ASCII '#',    ! ASCII representation of a pounds sign
;    0108  0          ASC_QUOTE       =%ASCII '''',   ! ASCII representation of a quote character
;    0109  0          ASC_SPACE       =%ASCII ' ',    ! ASCII representation of a space
;    0110  0          ASC_SQ_CLO_BRAK =%ASCII ']',    ! ASCII representation of a closed square bracket
;    0111  0          ASC_SQ_OPN_BRAK =%ASCII '[',    ! ASCII representation of an open square bracket
;    0112  0          ASC_TAB         = 9,            ! ASCII representation of a tab
;    0113  0          ASC_UP_ARROW    =%ASCII '^',    ! ASCII representation of an up arrow
;    0114  0
;    0115  0
;    0116  0          ! The 'mode' data structure is really just
;    0117  0          ! a byte vector with the following characteristics.
;    0118  0          !
;    0119  0          MODE_LVL_SIZE   =10,            ! Number of bytes in each level.
;    0120  0          MODE_LEVELS     =4,             ! Number of levels for mode settings
;    0121  0
;    0122  0
;    0123  0          ! Each level of the mode data strucutre has the following entries:
;    0124  0          !
;    0125  0          MODE_RADIX      =0,             ! Radix - dec, hex, oct, etc.
;    0126  0          MODE_LENGTH     =1,             ! Length - long, word, byte, etc.
;    0127  0          MODE_SYMBOLS    =2,             ! Boolean -> whether we know values
;    0128  0                                          !      as "extern + offset" or not.
;    0129  0          MODE_INSTRUC    =3,             ! Boolean -> whether we input/output
;    0130  0                                          !      values as machine instruction.
;    0131  0          MODE_ASCII      =4,             ! Boolean -> whether we output (only!)
;    0132  0                                          !      values as ASCII strings or not.
;    0133  0          MODE_SCOPE      =5,             ! Whether or not there is a CSP,
;    0134  0                                          !      (and whether we should apply it)
;    0135  0          MODE_GLOBALS    =6,             ! Whether or not we should apply global
;    0136  0                                          !      scope first in the search rules.
;    0137  0          MODE_FORTRAN    =7,             ! %LINE, %LABEL
;    0138  0          MODE_IMMEDIATE  =8,             ! FORTRAN address or contents of mode
;    0139  0          MODE_G_FLOATS   = 9,            ! Set the mode to G_FLOAT, so we can
;    0140  0                                          !      pick up the constant as G_FLOAT
;    0141  0                                          !      constant
;    0142  0
;    0143  0          ! The four levels have the following names and indices.
;    0144  0          !
;    0145  0          DEFAULT_MODE    =0,             ! Default system initialized mode
;    0146  0          USER_DEF_MODE   =1,             ! User-set default mode
;    0147  0          OVERRIDE_MODE   =2,             ! One-line override mode
;    0148  0          LOCAL_MODE      =3,             ! Local mode
;    0149  0
;    0150  0          ! The mode_length field should be one of the following.
;    0151  0          !
;    0152  0          BYTE_LENGTH     =1,             ! Byte length
;    0153  0          WORD_LENGTH     =2,             ! Word length
;    0154  0          LONG_LENGTH     =4,             ! Longword length
```

```
;    0155  0
;    0156  0              ! And the mode_radix field should be one of:
;    0157  0              !
;    0158  0              DECIMAL_RADIX   =10,          ! Decimal radix
;    0159  0              HEX_RADIX       =16,          ! Hexadecimal radix
;    0160  0              OCTAL_RADIX     =8,           ! Octal radix
;    0161  0              BINARY_RADIX    =2,           ! Binary radix
;    0162  0
;    0163  0              ! The FORTRAN mode fields are as follows:
;    0164  0              !
;    0165  0              LABEL_MODE      =1,           ! Numbers are label numbers
;    0166  0              LINE_MODE       =2,           ! Numbers are line numbers
;    0167  0              LITERAL_MODE    =3,           ! Numbers are literal values
;    0168  0
;    0169  0
;    0170  0              ! The 'STEP' data structure is really just a byte vector with the
;    0171  0              ! following characteristics. It is isomorphic to the 'mode' structure.
;    0172  0              !
;    0173  0              STEP_LVL_SIZE   =4,           ! Number of bytes in each level.
;    0174  0              STEP_LEVELS     =3,           ! Number of levels for step settings
;    0175  0
;    0176  0
;    0177  0              ! Each level of the step data strucutre has the following entries:
;    0178  0              !
;    0179  0              STEP_LINE       =0,           ! LINE or INSTRUCTION (a boolean)
;    0180  0              STEP_NOSYSTEM   =1,           ! [NO]SYSTEM steps (a boolean)
;    0181  0              STEP_OVER       =2,           ! INTO or OVER (a boolean)
;    0182  0              STEP_SOURCE     =3,           ! SOURCE or NOSOURCE
;    0183  0
;    0184  0              ! The three levels have the following names and indices.
;    0185  0              !
;    0186  0              DEFAULT_STEP    =0,           ! Default system initialized step
;    0187  0              USER_DEF_STEP   =1,           ! User-set default step type
;    0188  0              OVERRIDE_STEP   =2,           ! One-line override step type
;    0189  0
;    0190  0
;    0191  0              ! The data structure which holds the settings for SET SEARCH is
;    0192  0              ! a byte vector with the following characteristics (it works the
;    0193  0              ! same as the STEP data structure above)
;    0194  0              !
;    0195  0              SEARCH_LVL_SIZE = 2,
;    0196  0              SEARCH_LEVELS = 3,
;    0197  0
;    0198  0
;    0199  0              ! Each level of the data structure has the following entries
;    0200  0              !
;    0201  0              SEARCH_ALL = 0,
;    0202  0              SEARCH_IDENT = 1,
;    0203  0
;    0204  0
;    0205  0              ! The three levels have the following names and indices
;    0206  0              !
;    0207  0              DEFAULT_SEARCH = 0,
;    0208  0              USER_DEF_SEARCH = 1,
;    0209  0              OVERRIDE_SEARCH = 2,
;    0210  0
;    0211  0
```

```
0212  0          ! The data structure which holds the settings for SET DEFINE is
0213  0          ! a byte vector with the following characteristics (it works the
0214  0          ! same as the STEP data structure above)
0215  0          !
0216  0          DEFINE_LVL_SIZE = 1,
0217  0          DEFINE_LEVELS = 3,
0218  0
0219  0
0220  0          ! Each level of the data structure has only one entry
0221  0          !
0222  0          DEFINE_ONLY = 0,
0223  0
0224  0
0225  0          ! The three levels have the following names and indices
0226  0          !
0227  0          DEFAULT_DEFINE = 0,
0228  0          USER_DEF_DEFINE = 1,
0229  0          OVERRIDE_DEFINE = 2,
0230  0
0231  0
0232  0          ! The output configuration data structure is comparable to the STEP or
0233  0          ! MODE data structures except that it has only one level.  It is used
0234  0          ! to store information pertaining to DEBUG's output setting.
0235  0          !
0236  0          OUTPUT_SIZE = 3,                ! Number of bytes in the data structure
0237  0
0238  0
0239  0          ! The following entries are contained in the output config vector:
0240  0          !
0241  0          OUT_LOG          = 0,           ! LOG or NOLOG
0242  0          OUT_TERM         = 1,           ! TERMINAL or NOTERMINAL
0243  0          OUT_VERIFY       = 2,           ! VERIFY or NOVERIFY
0244  0          OUT_SCREEN       = 3,           ! SCREEN_LOG or NOSCREEN_LOG
0245  0
0246  0
0247  0          ! Used to identify the flavor of type between dbg$show_type and its
0248  0          ! callers.
0249  0          !
0250  0          DEFAULT          = 0,
0251  0          OVERRIDE         = 1,
0252  0
0253  0
0254  0          ! Bit configurations for context flags.
0255  0          !
0256  0          CONTEXT_BITS    =32,            ! Number of context bits
0257  0
0258  0
0259  0          ! Position of opcode byte at a breakpoint
0260  0          !
0261  0          OPCODE_BYTE     =0,             ! Opcode offset
0262  0
0263  0
0264  0          ! Location types for end range arguments
0265  0          !
0266  0          MEMORY_LOC      =0,             ! Memory location
0267  0
0268  0
```

```
;  0269  0              ! Names of exception types for traceback.
;  0270  0              !
;  0271  0              NOT_AN_EXC      =0,             ! Line number searching for pc
;  0272  0              TRAP_EXC        =1,             ! PC of trap searching for line number
;  0273  0              FAULT_EXC       =2,             ! PC of fault searching for line number
;  0274  0              LOOKUP_EXC      =3;             ! Like TRAP only don't do VAL_TO_SYM
;  0275  0                                              !      again.
;  0276  0
;  0277  0
;  0278  0              ! Literals to define the various address spaces in the VAX architecture.
;  0279  0              !
;  0280  0      LITERAL
;  0281  0              SYSTEM_SPACE    = %X'80000000',
;  0282  0              P1_SPACE        = %X'40000000' ;
;  0283  0
;  0284  0      LITERAL
;  0285  0              ! Language names
;  0286  0              !
;  0287  0              LANG_MACRO      = 0,
;  0288  0              LANG_FORTRAN    = 1,
;  0289  0              LANG_BLISS      = 2,
;  0290  0              LANG_COBOL      = 3,            ! COBOL
;  0291  0              LANG_BASIC      = 4,            ! BASIC+2
;  0292  0              LANG_PLI        = 5,            ! PL/I
;  0293  0              LANG_PASCAL     = 6,            ! Pascal
;  0294  0              LANG_C          = 7,            ! C
;  0295  0              MAX_LANGUAGE    = 6;            ! Languages 0 - 6 (C not supported yet)
;  0296  0
;  0297  0
;  0298  0              ! The semantic stack is a BLOCKVECTOR for which the following fields
;  0299  0              ! are defined...
;  0300  0              !
;  0301  0      FIELD STACK_FIELDS =
;  0302  0              SET
;  0303  0              STK$V_BASE    = [0,0,0,0],      ! Base of block.
;  0304  0              STK$V_VAL1    = [0,0,32,0],     ! First value field
;  0305  0              STK$V_VAL2    = [1,0,32,0],     ! Second value field
;  0306  0              STK$V_NT_PTR  = [2,0,32,0],     ! Pointer to name table entry
;  0307  0              STK$V_INDEX   = [3,0,16,0],     ! Index field
;  0308  0              STK$V_OFFSET  = [3,16,16,0],    ! Offset field
;  0309  0              STK$V_POS     = [4,0,16,0],     ! Position field
;  0310  0              STK$V_TYPE    = [4,16,8,0],     ! Token type field
;  0311  0              STK$V_SIZE    = [4,24,8,0],     ! Size field
;  0312  0              STK$V_EXT     = [5,0,1,0],      ! Sign extension bit
;  0313  0              STK$V_REF     = [5,1,1,0],      ! Is token a REF bit
;  0314  0              STK$V_IMMED   = [5,2,1,0],      ! Immediate mode bit
;  0315  0              STK$V_ARGS    = [5,3,1,0],      ! Access actuals supplied bit
;  0316  0              STK$V_FLDRF   = [5,4,2,0],      ! <field_ref> supplied flag
;  0317  0              STK$V_DOT     = [5,6,1,0],      ! Indirection flag
;  0318  0              STK$V_STRUC   = [5,8,4,0]       ! Kind of BLISS structure
;  0319  0              TES;
;  0320  0
;  0321  0      LITERAL
;  0322  0              STACK_ENTRY_SIZ = 6,            ! No of longwords in a semantic_stack BLOCK
;  0323  0              NO_STRUC        = 0,            ! BLISS structure types
;  0324  0              VECTR           = 1,
;  0325  0              BITVEC          = 2,
```

```
;    0326  0              BLOK            = 3,
;    0327  0              BLOCKVEC        = 4;
;    0328  0
;    0329  0      MACRO
;    0330  0              SEMANTIC_STACK = BLOCKVECTOR[20, STACK_ENTRY_SIZ] FIELD(STACK_FIELDS)%;
;    0331  0
;    0332  0
;    0333  0      ! General purpose structure for manipulation of singly linked lists.  Each cell
;    0334  0      ! contains two long-word fields. The first is a pointer to the next list link
;    0335  0      ! cell; the second is a pointer to the current element in the list.
;    0336  0      !
;    0337  0      LITERAL
;    0338  0              LIST_LINK_SIZE  =  8;           ! Bytes in a list link cell
;    0339  0
;    0340  0      FIELD LIST_LINK_FLDS =
;    0341  0              SET
;    0342  0              NEXT_ONE_PTR    =  [0,0,32,0],  ! Pointer to the next link cell
;    0343  0              THIS_ONE_PTR    =  [4,0,32,0]   ! Pointer to the current object in the list
;    0344  0              TES;
;    0345  0
;    0346  0      MACRO
;    0347  0              LIST_LINK = BLOCK[LIST_LINK_SIZE, BYTE] FIELD(LIST_LINK_FLDS)%;
;    0348  0      !
;    0349  0      ! End of DBGGEN.REQ
```

```
.                                       COMMAND QUALIFIERS

;       BLISS/LIBRARY=LIB$:DBGGEN.L32/LIST=LIS$:DBGGEN.LIS SRC$:DBGGEN.REQ

; Run Time:         00:02.6
; Elapsed Time:     00:04.3
; Lines/CPU Min:     8211
; Lexemes/CPU-Min: 21364
; Memory Used:   28 pages
; Library Precompilation Complete
```

DBGIFTHEN
LIS

DBGLANVEC
LIS

DBGLANGOP
LIS

DBGGEN
LIS

DBGLEVEL1
LIS