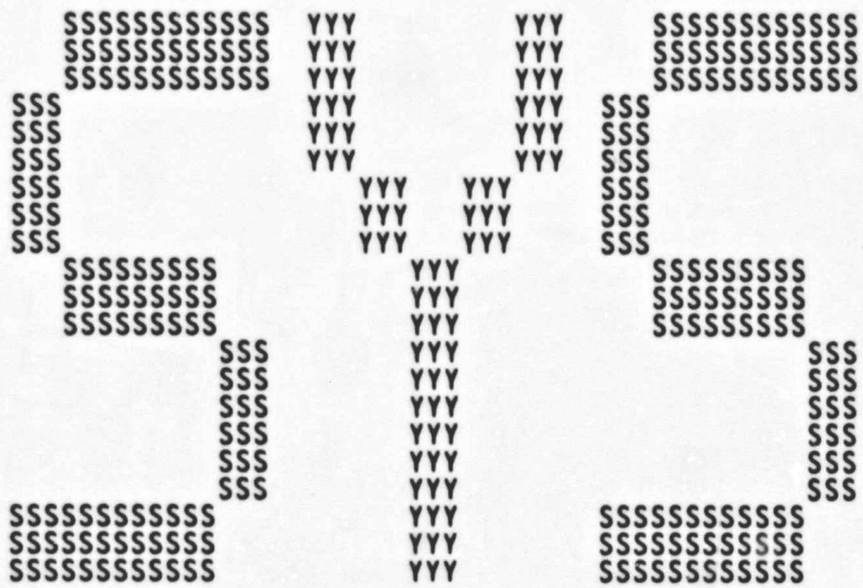
SSSSSSSSSSS YYY YYY SSSSSSSSSSS
SSSSSSSSSSS YYY YYY SSSSSSSSSSS
SSSSSSSSSSS YYY YYY SSSSSSSSSSS
SSS YYY YYY SSS
SSS YYY YYY SSS
SSS YYY YYY SSS
SSS YYY YYY SSS
SSS YYY YYY SSS
SSS YYY YYY SSS
SSSSSSSSS YYY SSSSSSSSS
SSSSSSSSS YYY SSSSSSSSS
SSSSSSSSS YYY SSSSSSSSS
SSS YYY SSS
SSS YYY SSS
SSS YYY SSS
SSS YYY SSS
SSS YYY SSS
SSS YYY SSS
SSSSSSSSSSS YYY SSSSSSSSSSS
SSSSSSSSSSS YYY SSSSSSSSSSS
SSSSSSSSSSS YYY SSSSSSSSSSS

```
LL            IIIIII    NN      NN  KK     KK  VV      VV  EEEEEEEEEE    CCCCCCCC
LL            IIIIII    NN      NN  KK     KK  VV      VV  EEEEEEEEEE    CCCCCCCC
LL              II      NN      NN  KK     KK  VV      VV  EE            CC
LL              II      NN      NN  KK     KK  VV      VV  EE            CC
LL              II      NNNN    NN  KK    KK   VV      VV  EE            CC
LL              II      NNNN    NN  KK    KK   VV      VV  EE            CC
LL              II      NN  NN  NN  KKKKKK     VV      VV  EEEEEEEE      CC
LL              II      NN  NN  NN  KKKKKK     VV      VV  EEEEEEEE      CC
LL              II      NN    NNNN  KK    KK   VV      VV  EE            CC
LL              II      NN    NNNN  KK    KK   VV      VV  EE            CC
LL              II      NN      NN  KK     KK   VV  VV    EE            CC              ....
LL              II      NN      NN  KK     KK   VV  VV    EE            CC              ....
LLLLLLLLLL    IIIIII    NN      NN  KK     KK     VV      EEEEEEEEEE    CCCCCCCC        ....
LLLLLLLLLL    IIIIII    NN      NN  KK     KK     VV      EEEEEEEEEE    CCCCCCCC        ....

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
```

Table of contents

```
0000     1          .TITLE  LINKVEC - link loadable EXEC to vectors
0000     2          .IDENT  'V04-000'
0000     3 ;
0000     4 ;*****************************************************************************
0000     5 ;*                                                                           *
0000     6 ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
0000     7 ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000     8 ;*  ALL RIGHTS RESERVED.                                                     *
0000     9 ;*                                                                           *
0000    10 ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
0000    11 ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000    12 ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
0000    13 ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
0000    14 ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
0000    15 ;*  TRANSFERRED.                                                             *
0000    16 ;*                                                                           *
0000    17 ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
0000    18 ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
0000    19 ;*  CORPORATION.                                                             *
0000    20 ;*                                                                           *
0000    21 ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
0000    22 ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
0000    23 ;*                                                                           *
0000    24 ;*                                                                           *
0000    25 ;*****************************************************************************
0000    26
0000    27 ;++
0000    28 ; FACILITY: VMS Executive, system initialization services.
0000    29 ;
0000    30 ; ABSTRACT:
0000    31 ;
0000    32 ;       This module contains the code to connect various pieces of the
0000    33 ;       loadable EXEC up to their system vectors.  This code is used
0000    34 ;       at system boot time by the module INIT, but may be used later
0000    35 ;       in the life of the system as well.
0000    36 ;
0000    37 ; ENVIRONMENT:
0000    38 ;
0000    39 ;       Kernel Mode
0000    40 ;
0000    41 ; AUTHOR:
0000    42 ;
0000    43 ;       Steven T. Jeffreys
0000    44 ;
0000    45 ; CREATION DATE:
0000    46 ;
0000    47 ;        27 November, 1982
0000    48 ;
0000    49 ; MODIFIED BY:
0000    50 ;
0000    51 ;       V03-001 JWH0205         Jeffrey W. Horn         24-Mar-1983
0000    52 ;               Add two vector type codes, SLV$K_SDATA and SLV$K_SJUMP.
0000    53 ;               Also fix bug in ADATA vector processing, code was not
0000    54 ;               accounting for bytes skiped in SYS.EXE because of .ALIGN
0000    55 ;               directives.
0000    56 ;
0000    57 ;
```

```
0000     58 ;--
0000     59
0000     60 ;
0000     61 ; Declarations:
0000     62 ;
0000     63
0000     64          $DYNDEF                         ; Define data structure id codes
0000     65          $SLVDEF                         ; Define system loadable vector offsets
0000     66          $SSDEF                          ; define status codes
```

```
                     0000     68          .SBTTL  EXE$LINK_VEC - Connect vector to loaded code
                     0000     69 ;++
                     0000     70 ;
                     0000     71 ; FUNCTIONAL DESCRIPTION:
                     0000     72 ;
                     0000     73 ;       Given a list of self-relative offsets to loadable routines/data,
                     0000     74 ;       and a parallel list of system vectors, ensure that both lists
                     0000     75 ;       look reasonable, then relocate the info and plug it into the
                     0000     76 ;       appropriate system vector.
                     0000     77 ;
                     0000     78 ; CALLING SEQUENCE:
                     0000     79 ;
                     0000     80 ;       JSB/BSB EXE$LINK_VEC
                     0000     81 ;
                     0000     82 ; INPUTS:
                     0000     83 ;
                     0000     84 ;       R2 = pointer into list of self-relative offsets into loaded code
                     0000     85 ;       R3 = address at which calculated address of loaded routines/data
                     0000     86 ;            structures should be written
                     0000     87 ;
                     0000     88 ; OUTPUTS:
                     0000     89 ;
                     0000     90 ;       The contents of all registers, save R0, are preserved.
                     0000     91 ;
                     0000     92 ; SIDE EFFECTS:
                     0000     93 ;
                     0000     94 ;       None.
                     0000     95 ;
                     0000     96 ; ROUTINE VALUE:
                     0000     97 ;
                     0000     98 ;       R0 = SS$_BADIMGHDR      : bad data structure pointed to be R2, no load.
                     0000     99 ;
                     0000    100 ;       All other status codes are returned by called routines, and are simply
                     0000    101 ;       passed back to the caller of EXE$LINK_VEC.  They are listed here for
                     0000    102 ;       convenience sake, and also in the appropriate routine header.
                     0000    103 ;
                     0000    104 ;       R0 = SS$_NORMAL         : normal successful completion, code loaded.
                     0000    105 ;          = SS$_BADVEC         : data structure has a bad vector, no load.
                     0000    106 ;--
                     0000    107
                 00000000    108          .PSECT  Z$INIT                  ; This code must be part of INIT
                     0000    109
                     0000    110 EXE$LINK_VEC::
                     0000    111          ;
                     0000    112          ;
                     0000    113          ; The list of self-relative offsets has a fixed header.
                     0000    114          ; Perform some sanity checks on that header.
                     0000    115          ;
                     0000    116          ;
50   0044 8F   3C    0000    117          MOVZWL  #SS$_BADIMGHDR,R0       ; Assume not the right kind of image
          62   B1    0005    118          CMPW    SLV$C_CODESIZE(R2),-    ; Check redundant code size info
        08 A2         0007    119                  SLV$W_SIZE(R2)  ;
             1F   12  0009    120          BNEQ    69$                     ; Branch if error
          62 8F   91  000B    121          CMPB    #DYN$C_LOADCODE,-       ; Check the data structure type
        0A A2         000E    122                  SLV$B_TYPE(R2)          ;
             18   12  0010    123          BNEQ    69$                     ; Branch if error
                      0012    124          ;
```

```
                     0012   125          ;
                     0012   126          ; Make two passes through the fixup vector information; the
                     0012   127          ; first to verify that the information is valid, the second
                     0012   128          ; to actually plug the information into the system vectors.
                     0012   129          ;
                     0012   130          ;
          1C BB      0012   131          PUSHR   #^M<R2,R3,R4>           ; Save R2..R4
52   24 A2 DE      0014   132          MOVAL   SLV$T_LIST(R2),R2       ; Step past the header
          54 D4      0018   133          CLRL    R4                      ; Indicate test mode
          11 10      001A   134          BSBB    SCAN_VEC_LIST           ; Check the fixup info
       0B 50 E9      001C   135          BLBC    R0,69$                  ; Exit if error
          54 D6      001F   136          INCL    R4                      ; Indicate fixup mode
    52    6E 7D      0021   137          MOVQ    (SP),R2                 ; Restore R2 and R3
52   24 A2 DE      0024   138          MOVAL   SLV$T_LIST(R2),R2       ; Step past the header
          03 10      0028   139          BSBB    SCAN_VEC_LIST           ; Plug the system vectors
          1C BA      002A   140 69$:     POPR    #^M<R2,R3,R4>           ; Restore R2..R4
          05 002C   141          RSB                             ; Return
```

```
                     002D   143          .SBTTL  SCAN_VEC_LIST - scan fixup vector list
                     002D   144 ;++
                     002D   145 ;
                     002D   146 ; FUNCTIONAL DESCRIPTION:
                     002D   147 ;
                     002D   148 ;       Scan through a list of self relative vectors, and conditionally
                     002D   149 ;       check the validity of the list or load the information in the list
                     002D   150 ;       into a system vector.
                     002D   151 ;
                     002D   152 ; CALLING SEQUENCE:
                     002D   153 ;
                     002D   154 ;       JSB/BSB SCAN_VEC_LIST
                     002D   155 ;
                     002D   156 ; INPUTS:
                     002D   157 ;
                     002D   158 ;       R2 = pointer into list of self-relative offsets into loaded code
                     002D   159 ;       R3 = address at which calculated address of loaded routines/data
                     002D   160 ;          structures should be written
                     002D   161 ;       R4 = action indicator. 0 implies sanity check, 1 implies load vectors.
                     002D   162 ;
                     002D   163 ; SIDE EFFECTS:
                     002D   164 ;
                     002D   165 ;       None.
                     002D   166 ;
                     002D   167 ; ROUTINE VALUE:
                     002D   168 ;
                     002D   169 ;       R0 = SS$_NORMAL         : normal successful completion, code loaded.
                     002D   170 ;          = SS$_BADVEC         : data structure has a bad vector, no load.
                     002D   171 ;
                     002D   172 ;--
                     002D   173
                     002D   174 SCAN_VEC_LIST:                               ; Scan fixup vector list
           51   DD  002D   175          PUSHL   R1                          ; Save R1
 51  0080 8F   3C  002F   176          MOVZWL  #SLV$K_MAXVEC,R1            ; Set loop limit
     50   82   98  0034   177 1$:      CVTBL   (R2)+,R0                    ; Pick up the type byte
          0F   15  0037   178          BLEQ    11$                         ; Leave if <= 0
          13   10  0039   179          BSBB    PROCESS_VECTOR              ; Perform vector type-dependent work
        0D 50  E9  003B   180          BLBC    R0,13$                      ; Branch if error
        F3 51  F5  003E   181          SOBGTR  R1,1$                       ; Branch if more to go
 50  2064 8F   3C  0041   182          MOVZWL  #SS$_BADVEC,R0              ; Assume bad vector
          03   11  0046   183          BRB     13$                         ; Return with error status
     50   01   3C  0048   184 11$:     MOVZWL  #SS$_NORMAL,R0              ; Set success status
          02   BA  004B   185 13$:     POPR    #^M<R1>                     ; Restor R1
               05  004D   186          RSB                                 ; Return
```

```
                     004E   188          .SBTTL  PROCESS_VECTOR - vector type-dependent processing
                     004E   189 ;++
                     004E   190 ;
                     004E   191 ; FUNCTIONAL DESCRIPTION:
                     004E   192 ;
                     004E   193 ;
                     004E   194 ; CALLING SEQUENCE:
                     004E   195 ;
                     004E   196 ;       JSB/BSB PROCESS_VECTOR
                     004E   197 ;
                     004E   198 ; INPUTS:
                     004E   199 ;
                     004E   200 ;       R0 = vector type code
                     004E   201 ;       R2 = pointer into list of self-relative offsets into loaded code
                     004E   202 ;       R3 = address of the system vector
                     004E   203 ;       R4 = action indicator. 0 implies sanity check, 1 implies load vectors.
                     004E   204 ;
                     004E   205 ; OUTPUT:
                     004E   206 ;
                     004E   207 ;       R2 and R3 are updated to point to the next entries
                     004E   208 ;       in their respective lists.  However, if an error is
                     004E   209 ;       detected, the contents of R2 and R3 are unpredictable.
                     004E   210 ;
                     004E   211 ; SIDE EFFECTS:
                     004E   212 ;
                     004E   213 ;       None.
                     004E   214 ;
                     004E   215 ; ROUTINE VALUE:
                     004E   216 ;
                     004E   217 ;       R0 = SS$_NORMAL         : normal successful completion, code loaded.
                     004E   218 ;          = SS$_BADVEC         : data structure has a bad vector, no load.
                     004E   219 ;
                     004E   220 ;--
                     004E   221
                00009F17   004E   222 ABSOLUTE_JMP    = ^X9F17                ; Hex equivalent of "JMP @#"
                     004E   223
                     004E   224 PROCESS_VECTOR:                             ; Vector type-dependent checks
                     004E   225          ;
                     004E   226          ;
                     004E   227          ; CASE on the vector type code to the appropriate vector handler.
                     004E   228          ;
                     004E   229          ;
                     004E   230
                     004E   231          ASSUME  SLV$K_LDATA EQ SLV$K_MINTYPE
                     004E   232          ASSUME  SLV$K_AJUMP EQ SLV$K_MINTYPE+1
                     004E   233          ASSUME  SLV$K_UJUMP EQ SLV$K_MINTYPE+2
                     004E   234          ASSUME  SLV$K_SDATA EQ SLV$K_MINTYPE+3
                     004E   235          ASSUME  SLV$K_SJUMP EQ SLV$K_MINTYPE+4
                     004E   236          ASSUME  SLV$K_SJUMP EQ SLV$K_MAXTYPE
                     004E   237
05   01   50   AF     004E   238          CASEW   R0,#SLV$K_MINTYPE,#SLV$K_MAXTYPE
                     0052   239                                                 ; Branch displacement table
                000C' 0052   240 1$:      .WORD   100$-1$                       ; If SLV$K_LDATA
                0014' 0054   241          .WORD   200$-1$                       ; If SLV$K_AJUMP
                001A' 0056   242          .WORD   300$-1$                       ; If SLV$K_UJUMP
                0026' 0058   243          .WORD   400$-1$                       ; If SLV$K_SDATA
                002B' 005A   244          .WORD   500$-1$                       ; If SLV$K_SJUMP
```

```
                     005C   245
                     005C   246          BRB     696969$                 ; Branch to common failure path
                     005E   247          ;                               ; Fall through if value out of range
                     005E   248          ;
                     005E   249          ; R0 = SLV$K_LDATA
                     005E   250          ; The vector is for a longword of data, and has the form
                     005E   251          ;
                     005E   252          ;       .ALIGN
                     005E   253          ;       .LONG   0
                     005E   254          ;
53   03   C0        005E   255 100$:     ADDL    #3,R3                   ; Round up to nex longword boundry
53   03   CA        0061   256          BICL    #3,R3                   ;
     0D   11        0064   257          BRB     10000$                  ; continue with common code
                     0066   258          ;
                     0066   259          ;
                     0066   260          ; R0 = SLV$K_AJUMP
                     0066   261          ; The vector is for an aligned jump, and has the form
                     0066   262          ;
                     0066   263          ;       .ALIGN
                     0066   264          ;       JMP     @#<32 bit address>
                     0066   265          ;
                     0066   266          ;
53   03   C0        0066   267 200$:     ADDL    #3,R3                   ; Round up to next longword boundary
53   03   CA        0069   268          BICL    #3,R3                   ;
                     006C   269                                          ; Fall through to common code
                     006C   270          ;
                     006C   271          ;
                     006C   272          ; R0 = SLV$K_UJUMP
                     006C   273          ; The vector is for an unligned jump, and has the form
                     006C   274          ;
                     006C   275          ;       JMP     @#<32 bit address>
                     006C   276          ;
                     006C   277          ;
83   9F17 8F   B1    006C   278 300$:     CMPW    #ABSOLUTE_JMP,(R3)+     ; First two bytes must be JMP @#
     25   12        0071   279          BNEQ    696969$                 ; Branch if error
                     0073   280
                     0073   281
     50   83   DE   0073   282 10000$:   MOVAL   (R3)+,R0                ; get system vector address
     0F   11        0076   283          BRB     20000$
                     0078   284
                     0078   285          ;
                     0078   286          ; R0 = SLV$K_SDATA
                     0078   287          ; The system vector is for a longword of data, and has the form
                     0078   288          ;
                     0078   289          ;       .ALIGN
                     0078   290          ;ENTRY::.LONG   0
                     0078   291          ;
                     0078   292          ; The load vector has the form:
                     0078   293          ;
                     0078   294          ;       .BYTE   SLV$K_SDATA
                     0078   295          ;       .ADDRESS ENTRY
                     0078   296          ;       .LONG   offset_to_data
                     0078   297          ;
     50   82   D0    0078   298 400$:     MOVL    (R2)+,R0                ; get address
     0A   11        007B   299          BRB     20000$                  ; no special processing
                     007D   300
                     007D   301          ;
```

```
                     007D   302          ; The system vector is for a jump, and has the form
                     007D   303          ;
                     007D   304          ;       .ALIGN
                     007D   305          ;ENTRY::JMP     @#<32 bit address>
                     007D   306          ;
                     007D   307          ; The load vector has the form:
                     007D   308          ;
                     007D   309          ;       .BYTE   SLV$K_SJUMP
                     007D   310          ;       .ADDRESS ENTRY
                     007D   311          ;       .LONG   offset_to_routine
                     007D   312          ;
                     007D   313
      50    82   D0  007D   314 500$:    MOVL    (R2)+,R0
9F17  8F    80   B1  0080   315          CMPW    (R0)+,#ABSOLUTE_JMP     ; First two bytes must be JMP @#
             11   12  0085   316          BNEQ    696969$                 ; Branch if error
                     0087   317
                     0087   318          ;
                     0087   319          ; Common code for processing a longword of information
                     0087   320          ; If R4 is 0, the contents of the system vector are not modified.
                     0087   321          ; At this point,
                     0087   322          ;       R0 = pointer to a longword system vector
                     0087   323          ;       R2 = pointer to longword of loadable info
                     0087   324          ;       R4 = action indicator. 0 impies CHECK, 1 implies LOAD
                     0087   325
                     0087   326 20000$:
             54   D5  0087   327          TSTL    R4                      ; CHECK mode?
             05   12  0089   328          BNEQ    20001$                  ; Branch if not
      52     04   C0  008B   329          ADDL    #4,R2                   ; point to next item
             04   11  008E   330          BRB     20002$                  ; Rejoin common code
60    82     52   C1  0090   331 20001$: ADDL3   R2,(R2)+,(R0)           ; Relocate info and plug the vector
      50     01   3C  0094   332 20002$: MOVZWL  #SS$_NORMAL,R0           ; Set success status
                05   0097   333          RSB
                     0098   334          ;
                     0098   335          ;
                     0098   336          ; Common failure path.
                     0098   337          ;
                     0098   338          ;
50    2064   8F   3C  0098   339 696969$:MOVZWL  #SS$_BADVEC,R0          ; Set failure status
                05   009D   340          RSB                             ; Return with error
                     009E   341          .END
```

```
ABSOLUTE_JMP                    = 00009F17
DYN$C_LOADCODE                  = 00000062
EXE$LINK_VEC                      00000000 RG    02
PROCESS_VECTOR                    0000004E R     02
SCAN_VEC_LIST                     0000002D R     02
SLV$B_TYPE                      = 0000000A
SLV$K_AJUMP                     = 00000002
SLV$K_LDATA                     = 00000001
SLV$K_MAXTYPE                   = 00000005
SLV$K_MAXVEC                    = 00000080
SLV$K_MINTYPE                   = 00000001
SLV$K_SDATA                     = 00000004
SLV$K_SJUMP                     = 00000005
SLV$K_UJUMP                     = 00000003
SLV$L_CODESIZE                  = 00000000
SLV$T_LIST                      = 00000024
SLV$W_SIZE                      = 00000008
SS$_BADIMGHDR                   = 00000044
SS$_BADVEC                      = 00002064
SS$_NORMAL                      = 00000001
```

Psect synopsis

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
ABS	00000000 (0.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
Z$INIT	0000009E (158.)	02 (2.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC BYTE

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	35	00:00:00.07	00:00:01.34
Command processing	116	00:00:00.49	00:00:04.13
Pass 1	238	00:00:05.45	00:00:18.22
Symbol table sort	0	00:00:00.82	00:00:02.14
Pass 2	73	00:00:01.25	00:00:04.82
Symbol table output	3	00:00:00.03	00:00:00.03
Psect synopsis output	2	00:00:00.03	00:00:00.08
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	469	00:00:08.14	00:00:30.76

```
The working set limit was 1350 pages.
31143 bytes (61 pages) of virtual memory were used to buffer the intermediate code.
There were 40 pages of symbol table space allocated to hold 582 non-local and 15 local symbols.
341 source lines were read in Pass 1, producing 13 object records in Pass 2.
11 pages of virtual memory were used to define 10 macros.
```

```
                          +----------------------------+
                          ! Macro library statistics !
                          +----------------------------+

Macro library name                        Macros defined
------------------                        --------------
_$255$DUA28:[SYS.OBJ]LIB.MLB;1                         2
_$255$DUA28:[SYSLIB]STARLET.MLB;2                      5
TOTALS (all libraries)                                 7

654 GETS were required to define 7 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS$:LINKVEC/OBJ=OBJ$:LINKVEC MSRC$:LINKVEC/UPDATE=(ENH$:LINKVEC)+EXECML$/LIB
```

IOSUBRAMS
LIS

IPCONTROL
LIS

IOSUBPAGD
LIS

LNMSUB
LIS

IOPERFORM
LIS

IOSUBNPAG
LIS

LINKVEC
LIS