L 5

```
   SSSSSSSSSSSS  YYY         YYY     SSSSSSSSSSSS  LLL                 OOOOOOOOO        AAAAAAAAA
   SSSSSSSSSSSS  YYY         YYY     SSSSSSSSSSSS  LLL                 OOOOOOOOO        AAAAAAAAA
   SSSSSSSSSSSS  YYY         YYY     SSSSSSSSSSSS  LLL                 OOOOOOOOO        AAAAAAAAA
SSS              YYY         YYY  SSS              LLL              OOO         OOO  AAA         AAA
SSS              YYY         YYY  SSS              LLL              OOO         OOO  AAA         AAA
SSS              YYY         YYY  SSS              LLL              OOO         OOO  AAA         AAA
SSS                 YYY   YYY     SSS              LLL              OOO         OOO  AAA         AAA
SSS                 YYY   YYY     SSS              LLL              OOO         OOO  AAA         AAA
SSS                 YYY   YYY     SSS              LLL              OOO         OOO  AAA         AAA
   SSSSSSSSS           YYY           SSSSSSSSS     LLL              OOO         OOO  AAA         AAA
   SSSSSSSSS           YYY           SSSSSSSSS     LLL              OOO         OOO  AAA         AAA
   SSSSSSSSS           YYY           SSSSSSSSS     LLL              OOO         OOO  AAA         AAA
            SSS        YYY                    SSS  LLL              OOO         OOO  AAAAAAAAAAAAAAA
            SSS        YYY                    SSS  LLL              OOO         OOO  AAAAAAAAAAAAAAA
            SSS        YYY                    SSS  LLL              OOO         OOO  AAAAAAAAAAAAAAA
            SSS        YYY                    SSS  LLL              OOO         OOO  AAA         AAA
            SSS        YYY                    SSS  LLL              OOO         OOO  AAA         AAA
            SSS        YYY                    SSS  LLL              OOO         OOO  AAA         AAA
SSSSSSSSSSSS           YYY        SSSSSSSSSSSS     LLLLLLLLLLLLLLL     OOOOOOOOO     AAA         AAA
SSSSSSSSSSSS           YYY        SSSSSSSSSSSS     LLLLLLLLLLLLLLL     OOOOOOOOO     AAA         AAA
SSSSSSSSSSSS           YYY        SSSSSSSSSSSS     LLLLLLLLLLLLLLL     OOOOOOOOO     AAA         AAA
```

```
_$2

Sym
---
SS$
SS$

SS$
SS$
SS$
SS$
SS$
SS$
SYS
SYS
SYS
TRY
UNL
WRI
```

```
**FILE**ID**CSPWAIT                                                    D 9



  CCCCCCCC    SSSSSSSS  PPPPPPPP    WW      WW    AAAAAA      IIIIII    TTTTTTTTTT
  CCCCCCCC    SSSSSSSS  PPPPPPPP    WW      WW    AAAAAA      IIIIII    TTTTTTTTTT
CC          SS          PP      PP  WW      WW  AA      AA      II          TT
CC          SS          PP      PP  WW      WW  AA      AA      II          TT
CC          SS          PP      PP  WW      WW  AA      AA      II          TT
CC          SS          PP      PP  WW      WW  AA      AA      II          TT
CC            SSSSSS    PPPPPPPP    WW      WW  AA      AA      II          TT
CC            SSSSSS    PPPPPPPP    WW      WW  AA      AA      II          TT
CC                  SS  PP          WW  WW  WW  AAAAAAAAAA      II          TT
CC                  SS  PP          WW  WW  WW  AAAAAAAAAA      II          TT
CC                  SS  PP          WWWW  WWWW  AA      AA      II          TT          ....
CC                  SS  PP          WWWW  WWWW  AA      AA      II          TT          ....
  CCCCCCCC  SSSSSSSS    PP          WW      WW  AA      AA    IIIIII        TT          ....
  CCCCCCCC  SSSSSSSS    PP          WW      WW  AA      AA    IIIIII        TT          ....


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
```

CSP
V04

Table of contents

```
0000
0000         .TITLE   CSPWAIT
0000         .IDENT  'V04-000'
0000
0000    ;*************************************************************************
0000    ;*                                                                       *
0000    ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                              *
0000    ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.               *
0000    ;*  ALL RIGHTS RESERVED.                                                 *
0000    ;*                                                                       *
0000    ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED *
0000    ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE *
0000    ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER *
0000    ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY *
0000    ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY *
0000    ;*  TRANSFERRED.                                                         *
0000    ;*                                                                       *
0000    ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE *
0000    ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT *
0000    ;*  CORPORATION.                                                         *
0000    ;*                                                                       *
0000    ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS *
0000    ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.              *
0000    ;*                                                                       *
0000    ;*                                                                       *
0000    ;*************************************************************************
0000    ;
0000    ;++
0000    ;
0000    ; FACILITY:     VMS Cluster Server Process
0000    ;
0000    ; ABSTRACT:     Subroutines to initiate "wait" states for Cluster Servers
0000    ;               and to create new threads of context.
0000    ;
0000    ; AUTHOR:       Paul R. Beck
0000    ;
0000    ; DATE:          3-MAR-1983     Last Edit:  22-JUN-1983 19:17:35
0000    ;
0000    ; REVISION HISTORY:
0000    ;
0000    ;       V03-005 ADE0004         Alan D. Eldridge                24-Apr-1984
0000    ;               Use CSP$$CRASH rather than BUG_CHECK.
0000    ;
0000    ;       V03-004 ADE0003         Alan D. Eldridge                22-Mar-1984
0000    ;               Fix synchronization between CSP$$WAIT and CSP$$RESUME.
0000    ;
0000    ;       V03-003 ADE0002         Alan D. Eldridge                28-Feb-1984
0000    ;               Change name of CSP$$OPCOM o CSP$TELL_OPCOM
0000    ;
0000    ;       V03-002 ADE0001         Alan D. Eldridge                3-Dec-1983
0000    ;               Move CSP$$SAVE_STACK to this module from CSP.B32 since BLISS is
0000    ;               not really appropriate for munging the stack.  Move CSP$RESUME,
0000    ;               CSP$CREATE_CTX, and CSP$DELETE_CTX as well so that all routines
0000    ;               callable by the client threads are in one module.  Changed
0000    ;               synchronization between CSP$$RESUME and CSP$$WAIT.
0000    ;
0000    ;
```

```
CSP
Sym

CLX
CLX
CLX
CLX
CLX
CLX
CLX
CLX
CLX
CLX
CLX
CLX
CLX
CLX
CON
CSP
CSP
CSP
CSP
CSP
CSP
CSP
CSP
CSP
CSP
CSP
CSP
LIB
LIB
MUM
P.A
P.A
SS$
SS$
SYS

PSE
---
.
. B
$AB
$PL
$CO

Pha
---
Ini
Com
Pas
```

```
                         0000  ;        V03-001 PRB0205         Paul Beck                     6-JUN-1983
                         0000  ;                Change CTX$ symbols to CLX$ to prevent conflict with RCP.
                         0000  ;--
```

```
CSP
VAX

Sym
Pas
Sym
Pse
Cro
Ass

The
100
The
393
12

Mac
---
_$2
_$2
_$2
TOT

155

The

MAC
```

```
                   0000     62 ;
                   0000     63 ; Include files
                   0000     64 ;
                   0000     65          $SFDEF                          ; define stack frame offsets
                   0000     66          $CLXDEF                         ; define context block offsets
                   0000     67
                   0000     68 ;
                   0000     69 ; Own storage
                   0000     70 ;
         00000001  0000     71 CONTEXT_ID:    .LONG   1               ; Storage for next CLX index
         00000140  0004     72 CLX_SIZE:      .LONG   CLX$K_LENGTH    ; CLX length used as input an parameter
                   0008     73                                        ; when 'call by reference' is needed
                   0008     74
```

```
                       0008  .SBTTL  'CSP$$RESUME'
                       0008  ;++
                       0008  ;
                       0008  ;  Completion of AST for asynchronous calls.  Reschedule the thread.  This
                       0008  ;  routine can be specified directly as an AST, or it may be called from
                       0008  ;  another AST.
                       0008  ;
                       0008  ;  CALLING SEQUENCE:    Standard AST (or called from and AST jacket routine).
                       0008  ;                       May also be called from "normal" level
                       0008  ;
                       0008  ;  FORMAL PARAMETERS:   P1 =    address of thread's context block
                       0008  ;
                       0008  ;  COMPLETION CODES:    N/A
                       0008  ;
                       0008  ;--
                 0000  0008  .ENTRY  CSP$$RESUME, 0                        ; Save nothing
                       000A          ;
                       000A          ;
                       000A          ;     This routine, since it is most often called via an AST, can
                       000A          ;     come before or may interrupt the execution of CSP$$WAIT.
                       000A          ;
                       000A          ;
      50   04 AC   D0  000A          MOVL    4(AP),R0                         ; Get context block
   07 0B A0   01   E3  000E          BBCS    #CLX$V_MUTEX,CLX$B_FLAGS(R0),10$ ; If BS, blocked by CSP$$WAIT
                       0013          ;
                       0013          ;
                       0013          ;     MUTEX was set.  We cannot use the queue linkage in the CLX
                       0013          ;
                       0013          ;
              02   E2  0013          BBSS    #CLX$V_RESUME_REQ,-              ; Tell CSP$$WAIT we interrupted
      22 0B A0         0015                   CLX$B_FLAGS(R0),70$            ; its execution
              1F   11  0018          BRB     50$                              ; Done
                       001A  10$:    ;
                       001A          ;
                       001A          ;     MUTEX was clear.  The queue linkage in the CLX is ours to use.
                       001A          ;
                       001A          ;
   03 0B A0   00   E3  001A          BBCS    #CLX$V_QUEUED,CLX$B_FLAGS(R0),30$ ; If BC, not yet queued
         50   60   0F  001F          REMQUE  (R0),R0                          ; Remove block from old queue
00000004'FF   60   0E  0022  30$:    INSQUE  (R0),@CSP$GQ_RESUME+4            ; ...and reschedule the thread
   14 0B A0   01   E5  0029          BBCC    #CLX$V_MUTEX,CLX$B_FLAGS(R0),90$; Release interlock
                       002E          $WAKE_S                                  ; Wake the CSP for processing
                   04  0039  50$:    RET                                      ; Done
                       003A
     00000004'8F   DD  003A  70$:    PUSHL   #SS$_NOPRIVSTRT+4                ; RESUME_REQ should have been 0
              06   11  0040          BRB     100$                             ;
     00000008'8F   DD  0042  90$:    PUSHL   #SS$_NOPRIVSTRT+8                ; MUTEX should have been set
00000000'EF   01   FB  0048  100$:   CALLS   #1,CSP$$CRASH                    ; Report bug
                   00  004F          HALT                                     ; Should never get here
                       0050
```

```
                            0050  .SBTTL          CSP$$WAIT - Asynchronous wait for AST completion.
                            0050  ;++
                            0050  ;
                            0050  ; The current stack is saved in an allocated block, which is saved in the
                            0050  ; current thread's context block. A test is then done to see if the completion
                            0050  ; AST completed prior to this routine; if so, the context is rescheduled.  The
                            0050  ; routine then forces a scheduler run by collapsing the stack and returning.
                            0050  ;
                            0050  ; CALLING SEQUENCE:     CALL    - never called from an AST routine.
                            0050  ;
                            0050  ; INPUT PARAMETERS:     none
                            0050  ;
                            0050  ; OUTPUT PARAMETERS:    N/A
                            0050  ;
                            0050  ; COMPLETION CODES:     N/A
                            0050  ;
                            0050  ;--
                     0FFC   0050  .ENTRY   CSP$$WAIT,^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11> ; save all registers
                            0052           ;
                            0052           ;
                            0052           ;     Save the current thread context.  The context is now on the stack
                            0052           ;     including all registers except R0 and R1.  R0 and R1 are saved
                            0052           ;     separately since the calling standard enforced by the VAX
                            0052           ;     architecture does not allow saving them in the entry mask.
                            0052           ;
                            0052           ;
     00000000'GF     16     0052           JSB      G^CSP$$SAVE_STACK               ; Save stack in current CLX
54   00000000'GF     D0     0058           MOVL     G^CSP$GL_CURCTX,R4              ; Get current CLX
     00000000'GF     D4     005F           CLRL     G^CSP$GL_CURCTX                 ; This thread no longer active
                            0065           ;
                            0065           ;
                            0065           ;     We must test for a race condition with the completion AST.
                            0065           ;
                            0065           ;     Since this routine is never called from AST level, it may be
                            0065           ;     interrupted by CSP$$RESUME -- but never vice versa.
                            0065           ;
                            0065           ;
     2B 0B A4   01   E2     0065           BBSS     #CLX$V_MUTEX, CLX$B_FLAGS(R4),90$       ; If BS, interlocked
     07 0B A4   00   E2     006A           BBSS     #CLX$V_QUEUED,CLX$B_FLAGS(R4),50$       ; If BS, RESUME occured
                            006F                                                    ; before we interlocked
  00000004'FF   64   0E     006F           INSQUE   (R4),@CSP$GQ_WAIT+4                     ; Queue it to wait list
                            0076  50$:     ;
                            0076           ;
                            0076           ;     Return to the scheduler. This is done by collapsing the stack to a
                            0076           ;     known point, where there is a call frame used to enter the
                            0076           ;     scheduler.  Then return with a success code.  This will cause the
                            0076           ;     scheduler to be reentered.
                            0076           ;
                            0076           ;
     1A 0B A4   01   E5     0076           BBCC     #CLX$V_MUTEX,CLX$B_FLAGS(R4),90$        ; Release interlock
     0A 0B A4   02   E5     007B           BBCC     #CLX$V_RESUME_REQ,CLX$B_FLAGS(R4),70$   ; If BS, RESUME occured
                            0080                                                    ; since we interlocked
          54    64   0F     0080           REMQUE   (R4),R4                                 ; Remove CLX from WAIT
  00000004'FF   64   0E     0083           INSQUE   (R4),@CSP$GQ_RESUME+4                   ; Que it to RESUME list
5D   00000000'GF     D0     008A  70$:     MOVL     G^CSP$GL_BASE_FP,FP                     ; Point to scheduler
                            0091                                                    ; stack frame
          50    00'  D0     0091           MOVL     S^#SS$_NORMAL,R0                        ; Declare success
```

```
                              04  0094   184           RET                                    ; Go reschedule.
                                  0095   185
            0000000C'8F       DD  0095   186 90$:      PUSHL    #SS$_NOPRIVSTRT+12            ; Use phoney status
       00000000'EF    01      FB  009B   187           CALLS    #1,CSP$$CRASH                 ; report MUTEX conflict
                              00  00A2   188           HALT                                   ; Should never get here
                                  00A3   189
                                  00A3   190 .dsabl    lsb
```

```
                                00A3  .SBTTL          CSP$$FORK - create new execution thread
                                00A3  ;++
                                00A3  ;
                                00A3  ;  This is a fork routine. A new context block is allocated and initialized,
                                00A3  ;  and the current context is saved and queued to the thread resume (grant)
                                00A3  ;  queue. The stack is NOT reclaimed, and the scheduler is NOT called.  When the
                                00A3  ;  scheduler is eventually entered, each thread thus queued is resumed at the
                                00A3  ;  return from this routine. The completion code is used to determine whether
                                00A3  ;  the execution context is the new thread (SS$_NORMAL) or simply the creator
                                00A3  ;  of the thread (0). For example:
                                00A3  ;
                                00A3  ;                 CALLS   #0,CSP$$FORK
                                00A3  ;                 BLBC    R0, 10$         ; continue executing old thread
                                00A3  ;                 BRW     NEW_THREAD      ; start executing new thread
                                00A3  ;
                                00A3  ;
                                00A3  ;
                                00A3  ;                                 CAVEAT
                                00A3  ;
                                00A3  ;       When creating a thread this way, be aware that the context saved
                                00A3  ;       is in the registers and stack. Local variables should be so defined.
                                00A3  ;
                                00A3  ;  CALLING SEQUENCE:    CALL
                                00A3  ;
                                00A3  ;  FORMAL PARAMETERS:   none
                                00A3  ;
                                00A3  ;  COMPLETION CODES:
                                00A3  ;
                                00A3  ;                 SS$_NORMAL =    The new thread has been resumed by the scheduler
                                00A3  ;                 0 =             The thread has been queued, context is intact
                                00A3  ;--
                         0FFC   00A3  .entry  CSP$$FORK,^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11> ; save all registers
                                00A5                                                    ;
         00000000'GF       DD   00A5          PUSHL   G^CSP$GL_CURCTX                   ; Temp save current context
                                00AB          ;
                                00AB          ;
                                00AB          ;     First, create the context block.
                                00AB          ;
                                00AB          ;
         00000062'GF       16   00AB          JSB     G^CSP$$CREATE_CTX                 ; Allocate new CLX block
              52    50     D0   00B1          MOVL    R0,R2                             ; Copy of new CLX pointer
                    1A     13   00B4          BEQL    10$                               ; If EQL, allocation failure
                                00B6          ;
                                00B6          ;
                                00B6          ;     Now, save the current stack in the context block. This requires
                                00B6          ;     faking out CSP$$SAVE_STACK as to which is the current context.
                                00B6          ;
                                00B6          ;
   00000000'GF      52     D0   00B6          MOVL    R2,G^CSP$GL_CURCTX                ; Store new context pointer
         00000000'GF       16   00BD          JSB     G^CSP$$SAVE_STACK                 ; Save stack in new CLX
                                00C3          ;
                                00C3          ;
                                00C3          ;     Next, schedule it.  Force the saved R0 to SS$_NORMAL so that
                                00C3          ;     upon rescheduling the caller will sense that this is the fork
                                00C3          ;     thread executing.
                                00C3          ;
                                00C3          ;
```

```
     28 A2     00'   D0  00C3          MOVL    S^#SS$_NORMAL,CLX$L_R0(R2)      ; "jam" success code
               52    DD  00C7          PUSHL   R2                              ; Setup parameter
  FF3A CF      01    FB  00C9          CALLS   #1,CSP$$RESUME                  ; ...and reschedule the thread
                         00CE          ;
                         00CE          ;
                         00CE          ;   Return with status=0 indicating that this isn't the fork thread yet.
                         00CE          ;
                         00CE          ;
               50    D4  00CE          CLRL    R0                              ; "SS$_NOT_FORK"
   00000000'GF 8ED0      00D0  10$:    POPL    G^CSP$GL_CURCTX                 ; Restore current CLX pointer
               04        00D7          RET
                         00D8
```

```
                             00D8   262 .SBTTL  'CSP$$SAVE_STACK - save stack frames prior to suspending thread'
                             00D8   263 ;++
                             00D8   264 ;
                             00D8   265 ;  Allocate memory to save the current stack (from top to the scheduler call
                             00D8   266 ;  frame), and store it in the current thread's context block.
                             00D8   267 ;
                             00D8   268 ;  CALLING SEQUENCE:     JSB
                             00D8   269 ;
                             00D8   270 ;  FORMAL PARAMETERS:    none
                             00D8   271 ;
                             00D8   272 ;  IMPLIED INPUTS:       CSP$GL_CURCTX = address of context block in which to store
                             00D8   273 ;                                        the saved stack.
                             00D8   274 ;
                             00D8   275 ;  COMPLETION CODES:     SS$_NORMAL, or failure code from LIB$$GET_VM
                             00D8   276 ;
                             00D8   277 ;--
                             00D8   278
                         00000000   279         .PSECT  $PLIT$,NOWRT,NOEXE,2
                             0000   280
54 53 5F 45 56 41 53 24 24 50 53 43  0000   281 P.AAK:  .ASCII  'CSP$$SAVE_STACK: CURCTX=0'<0><0><0>         ;
3D 58 54 43 52 55 43 20 3A 4B 43 41  000C
                      00 00 00 30  0018
                             001C   282                                                                  ;
                 010E0019    001C   283 P.AAJ:  .LONG   17694745                                         ;
                 00000000'   0020   284         .ADDRESS P.AAK                                           ;
                             0024   285
                         00000000   286         .PSECT  $CODE$,NOWRT,2
                             0000   287
                             0000   288 CSP$$SAVE_STACK::
            007C 8F      BB  0000   289         PUSHR   #^M<R2,R3,R4,R5,R6>               ; Save regs
                             0004   290                                               ;
       52   00000000'EF  D0  0004   291         MOVL    CSP$GL_CURCTX,R2                  ; Get current CLX block
                   11    12  000B   292         BNEQ    10$                               ; If NEQ, it's there
          0000001C'EF    9F  000D   293         PUSHAB  P.AAJ                             ; Setup message desc.
    00000000'EF    01    FB  0013   294         CALLS   #1, CSP$TELL_OPCOM                ; Display message
                   50    D4  001A   295         CLRL    R0                                ; Indicate error
                   3F    11  001C   296         BRB     40$                               ; Take common exit
                             001E   297 10$:    ;
                             001E   298         ;
                             001E   299         ;       We save the stack from CSP$GL_BASE_FP up to and including the
                             001E   300         ;       current stack frame (note JSB interface).  This assumes that this
                             001E   301         ;       routine is always called from a WAIT or a FORK routine which has
                             001E   302         ;       been CALL'ed by the thread which needs the context block saved.
                             001E   303         ;
                             001E   304         ;
                             001E   305         ASSUME  CLX$L_R1  EQ  4+CLX$L_R0
          28 A2    50    7D  001E   306         MOVQ     R0,CLX$L_R0(R2)                       ; Save R0,R1
          0B A2    08    8A  0022   307         BICB    #CLX$M_LOCAL_STACK,CLX$B_FLAGS(R2)     ; Init flag
   38 A2  00000000'GF    5D  C3  0026 308       SUBL3   FP,G^CSP$GL_BASE_FP,CLX$L_STACKSIZE(R2) ; Determine stack size
  00000100 8F    38 A2   D1  002F   309         CMPL    CLX$L_STACKSIZE(R2),#CLX$K_LOCAL_STACK ; Overflow CLX?
                   0B    1A  0037   310         BGTRU   20$                                    ; If GTRU, yes
       3C A2    40 A2    9E  0039   311         MOVAB   CLX$B_LOCAL_STACK(R2),CLX$A_STACK(R2)  ; Setup stack ptr
          0B A2    08    88  003E   312         BISB    #CLX$M_LOCAL_STACK,CLX$B_FLAGS(R2)     ; Indicate CLX stack
                   10    11  0042   313         BRB     30$                                    ; Continue
                             0044   314 20$:    ;
                             0044   315         ;
                             0044   316         ;       Must allocate a block to hold the stack
```

```
                                  0044   317              ;
                                  0044   318              ;
                      3C A2   9F  0044   319              PUSHAB  CLX$A_STACK(R2)                 ; Point to block ptr
                      38 A2   9F  0047   320              PUSHAB  CLX$L_STACKSIZE(R2)             ; Point to block size
          00000000'GF    02   FB  004A   321              CALLS   #2,G^LIB$GET_VM                 ; Allocate the block
                      09 50   E9  0051   322              BLBC    R0,40$                          ; If LBC, failed
    3C B2     6D      38 A2   28  0054   323 30$:         MOVC3   CLX$L_STACKSIZE(R2),(FP), -     ; Copy stack
                                  005A   324                      @CLX$A_STACK(R2)                ;
                                  005A   325                                                      ;
                      50    01    D0  005A   326              MOVL    #1,R0                           ; Setup success status
                    007C 8F   BA  005D   327 40$:         POPR    #^M<R2,R3,R4,R5,R6>             ; Restore regs
                              05  0061   328              RSB                                     ; Done
                                  0062   329
```

```
                                0062         .SBTTL  'CSP$$CREATE_CTX - allocate and initialize context block'
                                0062  ;++
                                0062  ;
                                0062  ;  Allocate and initialize a context block.
                                0062  ;
                                0062  ;  CALLING SEQUENCE:    JSB
                                0062  ;
                                0062  ;  INPUT PARAMETERS:    R0      Scratch
                                0062  ;
                                0062  ;  OUTPUT PARAMETERS:   R0      Address of context block is returned
                                0062  ;                              (0 if error).
                                0062  ;
                                0062  ;--
                                0062  CSP$$CREATE_CTX::
                        3F   BB 0062         PUSHR   #^M<R0,R1,R2,R3,R4,R5>          ; Save regs
                                0064                                                 ;
                        6E   9F 0064         PUSHAB  (SP)                            ; Address of block pointer
                 00000004'EF 9F 0066         PUSHAB  CLX_SIZE                        ; Address of block length
            00000000'GF  02  FB 006C         CALLS   #2,G^LIB$GET_VM                 ; Allocate the block
                                0073                                                 ;
                    1C  50   E9 0073         BLBC    R0,10$                          ; If LBC, then failed
00 BE  0140 8F  00  6E  00   2C 0076         MOVC5   #0,(SP),#0,#CLX$K_LENGTH,@(SP)  ; Zero it
                    50  6E   D0 007F         MOVL    (SP),R0                         ; Pickup the block
       24 A0     00000000'EF D0 0082         MOVL    CONTEXT_ID,CLX$L_INDEX(R0)      ; Enter the i.d.
                 00000000'EF D6 008A         INCL    CONTEXT_ID                      ; Bump the i.d. for next time
                        02   11 0090         BRB     20$                             ; Take common exit
                        6E   D4 0092  10$:   CLRL    (SP)                            ; Say "no block allocated"
                                0094                                                 ;
                        3F   BA 0094  20$:   POPR    #^M<R0,R1,R2,R3,R4,R5>          ; Restore regs
                             05 0096         RSB                                     ; Done
                                0097
```

```
                            0097          .SBTTL  'CSP$$DELETE_CTX - terminate thread'
                            0097  ;++
                            0097  ;
                            0097  ;  Terminate an execution thread by deleting the context block and
                            0097  ;  clearing the pointer.
                            0097  ;
                            0097  ;  CALLING SEQUENCE:    JSB
                            0097  ;
                            0097  ;  FORMAL PARAMETERS:   None
                            0097  ;
                            0097  ;  COMPLETION CODES:    N/A
                            0097  ;
                            0097  ;--
                            0097  CSP$$DELETE_CTX::
       00000000'GF     9F   0097          PUSHAB  G^CSP$GL_CURCTX                 ; Create pointer to CLX pointer
                            009D                                                  ;
             00 BE     D5   009D          TSTL    @(SP)                           ; Test current CLX block ptr
                09     12   00A0          BNEQ    10$                             ; If NEQ, there was one
    00000000'EF 00     FB   00A2          CALLS   #0, MUMBLE                      ; Report bug
                0F     11   00A9          BRB     20$                             ; Take common exit
                            00AB                                                  ;
                6E     DD   00AB  10$:    PUSHL   (SP)                            ; Setup ptr to block ptr
       00000004'EF     9F   00AD          PUSHAB  CLX_SIZE                        ; Setup ptr to length
    00000000'GF 02     FB   00B3          CALLS   #2,G^LIB$FREE_VM                ; Deallocate the block
                            00BA                                                  ;
                9E     D4   00BA  20$:    CLRL    @(SP)+                          ; Zero CSP$GL_CURCTX, fix stack
                       05   00BC          RSB                                     ; Return
                            00BD
```

```
                  00BD   392
                  00BD   393 .end
```

DIS
V04

```
CSPWAIT                                                                 16-SEP-1984 01:10:46  VAX/VMS Macro V04-00          Page  14
Symbol table                                                             5-SEP-1984 04:09:09  [SYSLOA.SRC]CSPWAIT.MAR;1           (10)

CLX$A_STACK                      = 0000003C
CLX$B_FLAGS                      = 0000000B
CLX$B_LOCAL_STACK                = 00000040
CLX$K_LENGTH                     = 00000140
CLX$K_LOCAL_STACK                = 00000100
CLX$L_INDEX                      = 00000024
CLX$L_R0                         = 00000028
CLX$L_R1                         = 0000002C
CLX$L_STACKSIZE                  = 00000038
CLX$M_LOCAL_STACK                = 00000008
CLX$V_MUTEX                      = 00000001
CLX$V_QUEUED                     = 00000000
CLX$V_RESUME_REQ                 = 00000002
CLX_SIZE                           00000004 R      01
CONTEXT_ID                         00000000 R      01
CSP$$CRASH                         ********   X    01
CSP$$CREATE_CTX                    00000062 RG     04
CSP$$DELETE_CTX                    00000097 RG     04
CSP$$FORK                          000000A3 RG     01
CSP$$RESUME                        00000008 RG     01
CSP$$SAVE_STACK                    00000000 RG     04
CSP$$WAIT                          00000050 RG     01
CSP$GL_BASE_FP                     ********   X    01
CSP$GL_CURCTX                      ********   X    01
CSP$GQ_RESUME                      ********   X    01
CSP$GQ_WAIT                        ********   X    01
CSP$TELL_OPCOM                     ********   X    04
LIB$FREE_VM                        ********   X    04
LIB$GET_VM                         ********   X    04
MUMBLE                             ********   X    04
P.AAJ                              0000001C R      03
P.AAK                              00000000 R      03
SS$_NOPRIVSTRT                     ********   X    01
SS$_NORMAL                         ********   X    01
SYS$WAKE                           ********  GX    01

                                                 +----------------+
                                                 ! Psect synopsis !
                                                 +----------------+

PSECT name                       Allocation           PSECT No.  Attributes
----------                       ----------           ---------  ----------
.  ABS  .                        00000000  (    0.)   00 (  0.)  NOPIC   USR   CON   ABS   LCL NOSHR NOEXE NORD   NOWRT NOVEC BYTE
. BLANK .                        000000D8  (  216.)   01 (  1.)  NOPIC   USR   CON   REL   LCL NOSHR   EXE   RD     WRT NOVEC BYTE
$ABS$                            00000000  (    0.)   02 (  2.)  NOPIC   USR   CON   ABS   LCL NOSHR   EXE   RD     WRT NOVEC BYTE
$PLIT$                           00000024  (   36.)   03 (  3.)  NOPIC   USR   CON   REL   LCL NOSHR NOEXE   RD   NOWRT NOVEC LONG
$CODE$                           000000BD  (  189.)   04 (  4.)  NOPIC   USR   CON   REL   LCL NOSHR   EXE   RD   NOWRT NOVEC LONG

                                              +------------------------+
                                              ! Performance indicators !
                                              +------------------------+

Phase                       Page faults    CPU Time       Elapsed Time
-----                       -----------    --------       ------------
Initialization                       29    00:00:00.03    00:00:01.28
Command processing                  107    00:00:00.45    00:00:03.74
Pass 1                              164    00:00:01.40    00:00:06.52
```

DIS
V04

```
Symbol table sort                  0     00:00:00.07     00:00:00.33
Pass 2                            85     00:00:00.60     00:00:03.10
Symbol table output                5     00:00:00.03     00:00:00.03
Psect synopsis output              2     00:00:00.02     00:00:00.02
Cross-reference output             0     00:00:00.00     00:00:00.00
Assembler run totals             394     00:00:02.60     00:00:15.02

The working set limit was 1350 pages.
10006 bytes (20 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 95 non-local and 18 local symbols.
393 source lines were read in Pass 1, producing 25 object records in Pass 2.
12 pages of virtual memory were used to define 11 macros.

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

Macro library name                                   Macros defined
------------------                                   --------------

_$255$DUA28:[SYSLOA.OBJ]CLUSTER.MLB;1                       1
_$255$DUA28:[SYS.OBJ]LIB.MLB;1                              0
_$255$DUA28:[SYSLIB]STARLET.MLB;2                           7
TOTALS (all libraries)                                      8

155 GETS were required to define 8 macros.

There were no errors, warnings or information messages.

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

0394 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

CSPOPCOM
LIS

CSPWAIT
LIS

CSPRCPCAC
LIS

CSPCJFRES
LIS

CSPQUORUM
LIS

DISTRLKI
LIS

CSPMOUNT
LIS

CSPVECTOR
LIS

CSPCLIENT
LIS

DSTRDLCK
LIS

DSTRLOCK
LIS