```
   SSSSSSSSSSS  YYY         YYY    SSSSSSSSSSS  LLL              OOOOOOOOO       AAAAAAAAA
   SSSSSSSSSSS  YYY         YYY    SSSSSSSSSSS  LLL              OOOOOOOOO       AAAAAAAAA
   SSSSSSSSSSS  YYY         YYY    SSSSSSSSSSS  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
SSSSSSSSSSS           YYY SSSSSSSSSSS     LLLLLLLLLLLLLLL        OOOOOOOOO     AAA         AAA
SSSSSSSSSSS           YYY SSSSSSSSSSS     LLLLLLLLLLLLLLL        OOOOOOOOO     AAA         AAA
SSSSSSSSSSS           YYY SSSSSSSSSSS     LLLLLLLLLLLLLLL        OOOOOOOOO     AAA         AAA
```

FILEID**ACKMSG

```
  AAAAAA      CCCCCCCC  KK      KK  MM      MM    SSSSSSSS    GGGGGGGG
  AAAAAA      CCCCCCCC  KK      KK  MM      MM    SSSSSSSS    GGGGGGGG
AA      AA  CC          KK      KK  MMMM  MMMM  SS          GG
AA      AA  CC          KK      KK  MMMM  MMMM  SS          GG
AA      AA  CC          KK    KK    MM  MM  MM  SS          GG
AA      AA  CC          KK    KK    MM  MM  MM  SS          GG
AA      AA  CC          KKKKKK      MM      MM    SSSSSS    GG
AA      AA  CC          KKKKKK      MM      MM    SSSSSS    GG
AAAAAAAAAA  CC          KK    KK    MM      MM          SS  GG  GGGGGG
AAAAAAAAAA  CC          KK    KK    MM      MM          SS  GG  GGGGGG
AA      AA  CC          KK      KK  MM      MM          SS  GG      GG      ....
AA      AA  CC          KK      KK  MM      MM          SS  GG      GG      ....
AA      AA    CCCCCCCC  KK      KK  MM      MM  SSSSSSSS      GGGGGG        ....
AA      AA    CCCCCCCC  KK      KK  MM      MM  SSSSSSSS      GGGGGG        ....

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  ACKMSG - Acknowledged Message Services
0000     2          .IDENT  'V04-001'
0000     3
0000     4  ;
0000     5  ;*****************************************************************************
0000     6  ;*                                                                           *
0000     7  ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
0000     8  ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000     9  ;*  ALL RIGHTS RESERVED.                                                     *
0000    10  ;*                                                                           *
0000    11  ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED     *
0000    12  ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE    *
0000    13  ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER     *
0000    14  ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY     *
0000    15  ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY     *
0000    16  ;*  TRANSFERRED.                                                             *
0000    17  ;*                                                                           *
0000    18  ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE     *
0000    19  ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
0000    20  ;*  CORPORATION.                                                             *
0000    21  ;*                                                                           *
0000    22  ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS     *
0000    23  ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
0000    24  ;*                                                                           *
0000    25  ;*                                                                           *
0000    26  ;*****************************************************************************
0000    27  ;
0000    28
0000    29  ;++
0000    30  ; FACILITY: EXECUTIVE, CLUSTER MANAGEMENT
0000    31  ;
0000    32  ; ABSTRACT:
0000    33  ;       This module provides an acknowledged message service based on
0000    34  ;       SCS for VAX/VMS Clusters.
0000    35  ;
0000    36  ; ENVIRONMENT: VAX/VMS
0000    37  ;
0000    38  ; AUTHOR: Steve Beckhardt,       CREATION DATE: 17-Aug-1982
0000    39  ;
0000    40  ; MODIFIED BY:
0000    41  ;
0000    42  ;       V04-001 DWT0241         David W. Thiel          7-Sep-1984
0000    43  ;               Close window (which occurs when a connection breaks)
0000    44  ;               in block transfer partner logic where the actual
0000    45  ;               state is not properly anticipated.
0000    46  ;
0000    47  ;       V03-022 DWT0236         David W. Thiel          10-Aug-1984
0000    48  ;               Update use of RDT$L_MAXRDIDX to match reinterpretation
0000    49  ;               of this field as the maximum index rather than the
0000    50  ;               number of indices (maximum+1).
0000    51  ;
0000    52  ;       V03-021 DWT0234         David W. Thiel          7-Aug-1984
0000    53  ;               Bugcheck on lost message detection.
0000    54  ;
0000    55  ;       V03-020 DWT0227         David W. Thiel          24-Jul-1984
0000    56  ;               Change warm CDRP cache limit from 3 to 2.
0000    57  ;
```

```
0000     58 ;       V03-019 DWT0215         David W. Thiel          30-Apr-1984
0000     59 ;               Correct missing value in CDRP$L_CDT field.
0000     60 ;
0000     61 ;       V03-018 SRB0112 DWT0183    Steve Beckhardt / Dave Thiel 20-Mar-1984
0000     62 ;               Implemented new SEND_MSG design whereby only one
0000     63 ;               CDRP (other than block transfers) may be in a resource
0000     64 ;               wait state at a time.  This more rigorously preserves
0000     65 ;               message sequentiality and as a by-product simplifies the
0000     66 ;               cleanup code.  This involves a major revision of all of
0000     67 ;               the logical involved in sending messages.
0000     68 ;
0000     69 ;       V03-017 DWT0167         David W. Thiel          28-Feb-1984
0000     70 ;               Use three-state dispatch on CDRP$B_CNXSTATE whereever
0000     71 ;               this field is used instead of two-state dispatch.
0000     72 ;               Return with SS$_NODELEAVE when a message is sent
0000     73 ;               to a node in long_break state, rather than bugcheck.
0000     74 ;               Bugcheck when a message buffer is returned for an
0000     75 ;               undefined CSID, rather than dropping the buffer.
0000     76 ;               Delete routines CNX$DEALL_WARMCDRP and CNX$DEALL_MSG_BUF.
0000     77 ;
0000     78 ;       V03-016 DWT0182         David W. Thiel          28-Feb-1984
0000     79 ;               Return error instead of bugchecking when a message
0000     80 ;               is sent to a permanently broken connection.
0000     81 ;
0000     82 ;       V03-015 ADE0002         Alan D. Eldridge        14-Feb-1984
0000     83 ;               Initialize more CDRP fields when recycling.
0000     84 ;
0000     85 ;       V03-014 ADE0001         Alan D. Eldridge        10-Jan-1984
0000     86 ;               Add CSP to list of ACKMSG clients.
0000     87 ;
0000     88 ;       V03-013 DWT0155         David W. Thiel          1-DEC-1983
0000     89 ;               Send all SCS messages with an explicit size computed
0000     90 ;               as the size of the largest message.  Perform a few
0000     91 ;               minor code cleanups.
0000     92 ;
0000     93 ;       V03-012 DWT0134         David W. Thiel          5-OCT-1983
0000     94 ;               Correct error patch in CNX$SEND_MSG so that when an
0000     95 ;               invalid CSID is given, the cleanup of the CDRP will
0000     96 ;               BUGCHECK if a message buffer is present, rather than
0000     97 ;               incorrectly attempting to deallocate the message buffer.
0000     98 ;
0000     99 ;       V03-011 ROW0206         Ralph O. Weber          8-AUG-1983
0000    100 ;               Cleanup bugs found in a code review and testing of block
0000    101 ;               transfers:
0000    102 ;                 - A missing @ sign on a REMQUE in CLEANUP_PARTNERS.
0000    103 ;                 - Fix CNX$PARTNER_INIT_CSB to return address of message
0000    104 ;                   buffer in R2 as advertized.
0000    105 ;                 - Have CNX$PARTNER_INIT_CSB correctly init CDRP$L_CDT before
0000    106 ;                   calling DEALLOC_MSG_BUF.
0000    107 ;                 - Fix numerous incorrect register usages in
0000    108 ;                   CNX$PARTNER_INIT_CSB.
0000    109 ;                 - Fix incorrect MOVC3 byte count in CNX$PARTNER_INIT_CSB.
0000    110 ;                 - Fix CNX$BLOCK_xxxxx to get CDT address into the CDRP.
0000    111 ;                 - Fix CNX$RCV_MSG to save R3 when deallocating a BTX.
0000    112 ;                 - Correct numerous typographical errors in the comments.
0000    113 ;                 - Fix CNX$PARTNER_RESPOND to use a unique BTX field to save
0000    114 ;                   the caller's return PC.  The new field is added to the BTX
```

```
0000    115 ;                   by ROW0214.  It is required to properly handle connection
0000    116 ;                   failure while the response message is being sent.
0000    117 ;
0000    118 ;       V03-010 BLS0233         Benn Schreiber          7-Aug-1983
0000    119 ;               Fix truncation error in CNX$BLOCK_READ_IRP.
0000    120 ;
0000    121 ;       V03-009 ROW0195         Ralph O. Weber          27-JUL-1983
0000    122 ;               Add CNX$PARTNER_RESPOND which responds to a block transfer
0000    123 ;               request, thus closing out a block transfer operation, but
0000    124 ;               returns control to the caller after the response message has
0000    125 ;               been sent.
0000    126 ;
0000    127 ;       V03-008 ROW0193         Ralph O. Weber          28-JUN-1983
0000    128 ;               Correct calling sequence for CNX$SEND_MSG_CSB in
0000    129 ;               CNX$SEND_MNY_MSGS.  Cause CNX$INIT_CDRP, CNX$ALLOC_CDRP, and
0000    130 ;               CNX$ALLOC_CDRP_ONLY to initialize CDRP$B_FIPL to IPL$_SCS.
0000    131 ;
0000    132 ;       V03-007 ROW0191         Ralph O. Weber          14-JUN-1983
0000    133 ;               Add dispatching for GETLKI.  Add paranoia checks to broken-
0000    134 ;               connection cleanup.  Fix CNX$ALLOC_CDRP to return SS$_INSFMEM
0000    135 ;               like the comments say it does.
0000    136 ;
0000    137 ;       V03-006 ROW0185         Ralph O. Weber          24-APR-1983
0000    138 ;               Add block transfer support including the following routines:
0000    139 ;                   - CNX$BLOCK_XFER to initiate a block transfer
0000    140 ;                   - CNX$BLOCK_XFER_IRP to initiate a block transfer with a
0000    141 ;                     CDRP/IRP pair
0000    142 ;                   - CNX$PARTNER_INIT_CSB to initialize partner portion of a
0000    143 ;                     block transfer
0000    144 ;                   - CNX$PARTNER_FINISH to complete partner portion of a block
0000    145 ;                     transfer
0000    146 ;                   - CNX$BLOCK_READ, CNX$BLOCK_WRITE, CNX$BLOCK_READ_IRP, and
0000    147 ;                     CNX$BLOCK_WRITE_IRP to actually do partner block transfers
0000    148 ;                   - CLEANUP_PARTNERS and CALL_PARTNER_ERROR to handle broken
0000    149 ;                     connection recovery on partner nodes
0000    150 ;               Correct CNX$SEND_MNY_MSGS to not send message to the local
0000    151 ;               node.
0000    152 ;
0000    153 ;       V03-005 ROW0183         Ralph O. Weber          18-APR-1983
0000    154 ;               Change CNX$CLEANUP, CNX$FAIL_MSG, CNX$RESEND_MSGS, and other
0000    155 ;               assorted routines to use SCS lookup threads routines.  This
0000    156 ;               should reduce time spent on the send message path but increase
0000    157 ;               time spent in failed virtual circuit cleanup.
0000    158 ;
0000    159 ;       V03-004 ROW0179         Ralph O. Weber          5-APR-1983
0000    160 ;               - Add support for use of CSIDs as input to CNX$SEND_MSG.
0000    161 ;               - Change incoming new message dispatching to a two level
0000    162 ;                 dispatch.
0000    163 ;               - Setup internal allocation of a CDRP for new incoming
0000    164 ;                 messages.
0000    165 ;               - Change CNX$DEALL_WARMCDRP to use RECYCL_RSPID.
0000    166 ;               - Add CNX$SEND_MNY_MSGS.
0000    167 ;               - Cause the sent and received message counters to be
0000    168 ;                 incremented.
0000    169 ;               - Change CNX$ALLOC_WARMCDRP and CNX$ALLOC_CDRP to use
0000    170 ;                 CSID input.  Add CNX$ALLOC_WARMCDRP_CSB.
0000    171 ;               - Add CNX$ALLOC_CDRP_ONLY and CNX$INIT_CDRP.
```

```
0000   172 ;                - Add CNX$SEND_MSG_CSB, CNX$DEALL_WARMCDRP_CSB,
0000   173 ;                  CNX$DEALL_MSG_BUF, and CNX$DEALL_MSG_BUF_CSB.
0000   174 ;
0000   175 ;        V03-003 SRB0074         Steve Beckhardt         27-Mar-1983
0000   176 ;                Fixed bug involving resuming a thread whose CDRP
0000   177 ;                was on a resource wait queue when the connection broke.
0000   178 ;
0000   179 ;        V03-002 DWT0083         David W. Thiel          7-Mar-1983
0000   180 ;                Replace HALTs with generic connection manager
0000   181 ;                BUG_CHECKs.
0000   182 ;
0000   183 ;        V03-001 DWT0070         David W. Thiel          17-Feb-1983
0000   184 ;                Split this module out of CNXMAN as part of a general
0000   185 ;                rewrite and reorganization of that module.
0000   186 ;
0000   187 ;--
0000   188
```

```
                        .SBTTL  DECLARATIONS
                ;
                ; INCLUDE FILES:
                ;
                        $CDRPDEF                        ; CDRP Offsets
                        $CDTDEF                         ; CDT Offsets
                        $CLSMSGDEF                      ; Cluster message offsets
                        $CLUBDEF                        ; Cluster block
                        $CLUBTXDEF                      ; Cluster block-xfer CDRP extension
                        $CSBDEF                         ; CSB Offsets
                        $DYNDEF                         ; Data structure type codes
                        $IPLDEF                         ; IPL definitions
                        $IRPDEF                         ; I/O request packet offsets
                        $PBDEF                          ; PB Offsets
                        $PDTDEF                         ; PDT Offsets
                        $RDDEF                          ; RD offsets
                        $RDTDEF                         ; RDT offsets
                        $SSDEF                          ; System return codes

                ;
                ; MACROS:
                ;

                ; MACRO:  FAC_POOL
                ;
                ;       This macro creates the list of allocated pool sizes -- one for each
                ;       facility -- which is used when an unsolicited message arrives.

                        .MACRO  FAC_POOL LIST

                        .MACRO  ONE_FAC_POOL, FAC, SIZE
                        .=$$FIRST+C[SMSG$K_FAC_'FAC'
                        .BYTE   SIZE
                        .IIF    GT .-$$BIGEST, $$BIGEST=.
                        .ENDM   ONE_FAC_POOL

                $$BIGEST=.
                $$FIRST:

                        .IRP    ITEM, <LIST>
                        ONE_FAC_POOL    ITEM
                        .ENDR

                        .=$$BIGEST

                        .ENDM   FAC_POOL


                ;
                ; EQUATED SYMBOLS:
                ;

00000002        MAXWARMCDRPS = 2                        ; Maximum number of CDRPs to cache
                                                        ; on CSB free list

                ;******************************************************************************
                ;
```

```
                        ; NOTE: The following assumptions are in effect for this entire module.
                        ;
                        ;****************************************************************************

                                ASSUME  IPL$_SYNCH  EQ  IPL$_SCS

                                .DEFAULT        DISPLACEMENT,WORD

                        00000000        .PSECT  $$$100,LONG

                        ;****************************************************************************
                        ;
                        ;       DESIGN NOTES:
                        ;
                        ;       The key to understanding this entire module is the strategy for keeping
                        ;       track of CDRPs and for cleaning up CDRPs when connections break.  This
                        ;       is the result of the design of SCS and of the mainline paths through
                        ;       this module that opt for simplicity and efficiency of the mainline paths
                        ;       at the expense of a complicted failure cleanup and recovery.
                        ;
                        ;       The cells CSB$L_RESENDQFL and CSB$L_RESENDQBL form the resend list (a
                        ;       list of all CDRPs pending transmission).  This list is organized as
                        ;       a single linked list with a pointer to the last element in the list
                        ;       (essentially a FIFO).  CSB$L_RESENDQFL contains the address of the
                        ;       first member of the list or zero, if the list is empty.  CSB$L_RESENDQBL
                        ;       contains the address of the last member of the list or the address of
                        ;       CSB$L_RESENDQFL, if the list is empty.  This structure is used instead
                        ;       of VAX queues because improved performance can be achieved in critical
                        ;       code paths.  The list is organized so that new messages are added to
                        ;       the end and the message to be sent next is taken from the front.
                        ;
                        ;       Similarly, the cells CSB$L_SENTQFL and CSB$L_SENTQBL form the sent list,
                        ;       a list of all CDRPs that have been transmitted but not yet acknowledged.
                        ;
                        ;****************************************************************************
```

```
                                                L 5
0000   284          .SBTTL  CNX$PRE_CLEANUP - Cleanup Outstanding Messages before Disconnecting
0000   285
0000   286 ;++
0000   287 ;
0000   288 ; FUNCTIONAL DESCRIPTION:
0000   289 ;
0000   290 ;       This routine is called by SCS when a connection breaks, before
0000   291 ;       a DISCONNECT is done.  The connection must be open when this
0000   292 ;       routine is called.
0000   293 ;
0000   294 ;       Simply stated, this routines finds all CDRPs that are in various stages
0000   295 ;       of being sent and puts them on the CSB resend list.
0000   296 ;
0000   297 ; CALLING SEQUENCE:
0000   298 ;
0000   299 ;       JSB     CNX$PRE_CLEANUP
0000   300 ;       IPL is at SCS fork level (8)
0000   301 ;
0000   302 ; INPUT PARAMETERS:
0000   303 ;
0000   304 ;       R5      Address of CSB
0000   305 ;
0000   306 ; IMPLICIT INPUTS:
0000   307 ;
0000   308 ;       None
0000   309 ;
0000   310 ; OUTPUT PARAMETERS:
0000   311 ;
0000   312 ;       None
0000   313 ;
0000   314 ; IMPLICIT OUTPUTS:
0000   315 ;
0000   316 ;       None
0000   317 ;
0000   318 ; SIDE EFFECTS:
0000   319 ;
0000   320 ;       R0-R4 destroyed
0000   321 ;
0000   322 ;--
0000   323
0000   324 CNX$PRE_CLEANUP::
0000   325
    56 DD 0000   326         PUSHL   R6                      ; Save a register.
56  55 D0 0002   327         MOVL    R5, R6                  ; Copy CSB address.
0005   328
0005   329 ; At this time, the CDRPs to be cleaned up are in the following states:
0005   330 ;
0005   331 ;       1) In critical section, waiting for RSPID, MSGBUF, or MAP with
0005   332 ;          CNXSTATE = NORMAL, REQUESTOR, or PARTNER.  Only in the
0005   333 ;          REQUESTOR and PARTNER states can the resource be MAP.
0005   334 ;          Resources that may be held are RSPID and MSGBUF, and in the
0005   335 ;          case of REQUESTORs and PARTNERs, MAP.  These messages all
0005   336 ;          have non-zero sequence numbers, except for PARTNERs which
0005   337 ;          have zero sequence numbers.
0005   338 ;
0005   339 ;       2) On the resend list with CNXSTATE = NORMAL, REQUESTOR, or
0005   340 ;          PARTNER.  The resources that may be held are RSPID, and in
```

```
                     ;                the latter two cases, MAP.  These are messages awaiting
                     ;                transmission or retransmission.  These messages all have
                     ;                non-zero sequence numbers, except for PARTNERs which have
                     ;                zero sequence numbers.
                     ;
                     ;            3) On sent list with CNXSTATE = NORMAL or REQUESTOR.  The only
                     ;                resource that may be held is RSPID.  These are messages
                     ;                awaiting acknowledgement.
                     ;
                     ;            4) Linked to the RDT and not in any of the above states.
                     ;                CNXSTATE = NORMAL. REQUESTOR, or PARTNER.  In the NORMAL state,
                     ;                the only resource that may be held is RSPID.  In the latter
                     ;                cases, the resources RSPID and MAP are held.  These messages
                     ;                have been acknowledged but have not yet been responded to.
                     ;                These messages all have zero sequence numbers.
                     ;
                     ;            5) Linked to the PARTNER queue with CNXSTATE = REQ_MAP or PART_MAP.
                     ;                These CDRPs are awaiting mapping resources outside of the
                     ;                critical section and are on this queue only to provide a way
                     ;                of finding these CDRPs.  Resources help may include RSPID and
                     ;                MSGBUF in the case of REQ_MAP.
                     ;
                     ;            6) Linked to the PARTNER queue with CNXSTATE = PART_IDLE.  This is
                     ;                an inactive partner thread that holds no resources.
                     ;
                     ; The purpose of this routine is to build a RESEND list containing a CDRPs that
                     ; may need to be resent or cleaned up, except for PARTNERs, PART_IDLEs, and
                     ; PART_MAPs which will always be failed and which are found via the PARTNER
                     ; queue.  The resulting RESEND list will contain (in this order):
                     ;       a) CDRPs with sequence number = 0.  These messages have been acknowledged
                     ;          and should never be resent.  CNXSTATE = NORMAL or REQUESTOR.  PARTNER
                     ;          block transfer requests are also in this category, are never
                     ;          acknowledged, and never resent.
                     ;       b) CDRPs with sequence number non-zero.  These messages may have been
                     ;          sent and may have been received; their disposition will be sorted
                     ;          out when and if the connection is reestablished.
                     ;
55   34 A6   D0  0005 378          MOVL    CSB$L_CURRCDRP(R6),R5   ; Get current CDRP, if any
         2D   18  0009 379          BGEQ    30$                     ; Don't have one
   55    65   0F  000B 380          REMQUE  CDRP$L_FQFL(R5),R5      ; Remove it from resource wait queue
   53    56   D0  000E 381          MOVL    R6,R3                   ; Set up CSB address
      0172   30  0011 382          BSBW    CLEANUP_CDRP            ; Clean out RSPID and message buffer
         65   7C  0014 383          CLRQ    CDRP$L_FQFL(R5)         ; Clean out queue linkage
      50 A5   D0  0016 384          MOVL    CDRP$L_SAVEPC(R5),-     ; Move saved PC to be fork PC so that
      0C A5         0019 385                  CDRP$L_FPC(R5)          ; thread is resumed correctly on error
                    001B 386          DISPATCH        CDRP$B_CNXSTATE(R5),TYPE=B,PREFIX=CDRP$K_, -
                    001B 387                  < -
                    001B 388                  <NORMAL,10$>, -         ; Normal message, link to resend list
                    001B 389                  <REQUESTOR,10$>, -      ; Block transfer requestor, link to resend l
                    001B 390                  <PARTNER,10$>, -        ; Block transfer partner, link to resend lis
                    001B 391                  >
                    0026 392          BUG_CHECK       CNXMGRERR,FATAL ; Invalid CDRP state
                    002A 393
65   1C A6   D0  002A 394 10$:    MOVL    CSB$L_RESENDQFL(R6), -  ; Insert at head of RESEND list
                    002E 395                  CDRP$L_FQFL(R5)
         04   12  002E 396          BNEQ    20$                     ; Branch if not only element in list
20 A6    65   DE  0030 397          MOVAL   CDRP$L_FQFL(R5), -      ; Make tail of list
```

```
                        0034   398                          CSB$L_RESENDQBL(R6)
      1C A6    65   DE  0034   399 20$:     MOVAL   CDRP$L_FQFL(R5), -      ; Update head pointer
                        0038   400                          CSB$L_RESENDQFL(R6)
      34 A6    01   D0  0038   401 30$:     MOVL    #1,CSB$L_CURRCDRP(R6)   ; Indicate no current CDRP, block activity
                        003C   402
                        003C   403 ; Cleanup warm CDRPs
                        003C   404
         53    56   D0  003C   405         MOVL    R6,R3                   ; Move CSB address
               0124 30   003F   406         BSBW    FLUSH_WARMCDRPS
                        0042   407
                        0042   408 ; Remove elements one-by-one from the head of the RESEND list.
                        0042   409 ; If the sequence number is non-zero, add to tail of SENT list.
                        0042   410 ; If the sequence number is zero, add to the head of SENT list.
                        0042   411 ; Finally, move SENT list to RESEND list, initialize SENT list.
                        0042   412
      55    1C A6    D0  0042   413 40$:     MOVL    CSB$L_RESENDQFL(R6),R5  ; Get first element in RESEND list
                  2C 13   0046   414         BEQL    80$                     ; Branch if RESEND list is empty
      1C A6    65   D0  0048   415         MOVL    CDRP$L_FQFL(R5), -      ; Set new first element in RESEND list
                        004C   416                          CSB$L_RESENDQFL(R6)
                  05 12   004C   417         BNEQ    50$                     ; Branch if list not empty
   20 A6    1C A6    DE  004E   418         MOVAL   CSB$L_RESENDQFL(R6), -  ; Reinitialize tail pointer
                        0053   419                          CSB$L_RESENDQBL(R6)
         54 A5    B5  0053   420 50$:     TSTW    CDRP$W_SENDSEQNM(R5)    ; Is sequence number non-zero?
                  10 12   0056   421         BNEQ    70$                     ; Branch if non-zero sequence number
                        0058   422         ;
                        0058   423         ; Add to head of SENTQ
                        0058   424         ;
      65    14 A6    D0  0058   425         MOVL    CSB$L_SENTQFL(R6), -    ; Link to front of SENTQ
                        005C   426                          CDRP$L_FQFL(R5)
                  04 12   005C   427         BNEQ    60$                     ; Branch if not first element in list
      18 A6    65   DE  005E   428         MOVAL   CDRP$L_FQFL(R5), -      ; Set up SENTQ tail pointer
                        0062   429                          CSB$L_SENTQBL(R6)
      14 A6    65   DE  0062   430 60$:     MOVAL   CDRP$L_FQFL(R5), -      ; Set new head pointer for SENTQ
                        0066   431                          CSB$L_SENTQFL(R6)
                  DA 11   0066   432         BRB     40$
                        0068   433
                        0068   434         ;
                        0068   435         ; Add to tail of SENTQ
                        0068   436         ;
                  65 D4   0068   437 70$:     CLRL    CDRP$L_FQFL(R5)         ; Zero list pointer
      18 B6    65   DE  006A   438         MOVAL   CDRP$L_FQFL(R5), -      ; Link to tail of list
                        006E   439                          @CSB$L_SENTQBL(R6)
      18 A6    65   DE  006E   440         MOVAL   CDRP$L_FQFL(R5), -      ; Update tail pointer
                        0072   441                          CSB$L_SENTQBL(R6)
                  CE 11   0072   442         BRB     40$
                        0074   443
                        0074   444 80$:     ;
                        0074   445         ; Move SENTQ to RESENDQ.
                        0074   446         ; Note that RESEND list is empty.
                        0074   447         ;
   1C A6    14 A6    D0  0074   448         MOVL    CSB$L_SENTQFL(R6), -    ; Copy head pointer
                        0079   449                          CSB$L_RESENDQFL(R6)
                  05 13   0079   450         BEQL    90$                     ; Branch if list is empty
   20 A6    18 A6    D0  007B   451         MOVL    CSB$L_SENTQBL(R6), -    ; Copy tail pointer
                        0080   452                          CSB$L_RESENDQBL(R6)
                        0080   453 90$:     ;
                        0080   454         ; Make SENTQ empty
```

```
                                0080   455          ;
            14 A6    D4         0080   456          CLRL    CSB$L_SENTQFL(R6)       ; Zero head pointer
      18 A6 14 A6    DE         0083   457          MOVAL   CSB$L_SENTQFL(R6), -    ; Initialize tail pointer
                                0088   458                  CSB$L_SENTQBL(R6)
                                0088   459
                                0088   460 ; Scan the PARTNER queue to:
                                0088   461 ;       a) Remove MAP waiters from their queues and clean them up.
                                0088   462 ;       b) Put idle partners onto the RESEND list.
                                0088   463
         53 58 A6    7E         0088   464          MOVAQ   CSB$L_PARTNERQFL(R6),R3 ; Address of BTX queue header
            54 53    D0         008C   465          MOVL    R3,R4
            54 64    D0         008F   466 100$:     MOVL    (R4),R4                 ; Next element of BTX queue
            54 53    D1         0092   467          CMPL    R3,R4                   ; End of list?
               42    13         0095   468          BEQL    140$                    ; Branch when scan is done
         55 18 A4    D0         0097   469          MOVL    CLUBTX$L_CDRP(R4),R5    ; CDRP address
                                009B   470          DISPATCH        CDRP$B_CNXSTATE(R5),TYPE=B,PREFIX=CDRP$K_, -
                                009B   471                  < -
                                009B   472                  <REQ_MAP,110$>, -       ; Fail map waiters
                                009B   473                  <PART_IDLE,120$>, -     ; Link to head of RESEND list
                                009B   474                  <PART_MAP,110$>, -      ; Fail map waiters
                                009B   475                  <PARTNER,100$>, -       ; Ignore partners - on other lists
                                009B   476                  >
                                00A8   477          BUG_CHECK       CNXMGRERR,FATAL ; Invalid CDRP state
                                00AC   478
                                00AC   479 ; Clean up map waiters
                                00AC   480 ;
            55 65    0F         00AC   481 110$:     REMQUE  CDRP$L_FQFL(R5),R5      ; Remove it from map resource wait queue
               65    7C         00AF   482          CLRQ    CDRP$L_FQFL(R5)         ; Clean out linkage
            53 57    D0         00B1   483          MOVL    R7,R3                   ; Set up CSB address
                00CF 30         00B4   484          BSBW    CLEANUP_CDRP            ; Clean up RSPID and message buffer
                                00B7   485
                                00B7   486          ; Have a CDRP waiting for mapping resources that must be failed.
                                00B7   487          ; Inputs to fork process are:
                                00B7   488          ;
                                00B7   489          ;       R0      contains 0 (failure)
                                00B7   490          ;       R3      Address of CSB
                                00B7   491          ;       R4      Address of PDT
                                00B7   492          ;       R5      Address of CDRP
                                00B7   493          ;
                                00B7   494          ; Fork routine may use R0 - R5.
                                00B7   495
               18    BB         00B7   496          PUSHR   #^M<R3,R4>              ; Save registers
         54 10 A3    D0         00B9   497          MOVL    CSB$L_PDT(R3),R4        ; PDT address
               50    D4         00BD   498          CLRL    R0                      ; Set failure status
            0C B5    16         00BF   499          JSB     @CDRP$L_FPC(R5)         ; Resume fork process
               18    BA         00C2   500          POPR    #^M<R3,R4>              ; Restore registers
               C9    11         00C4   501          BRB     100$                    ; Continue processing
                                00C6   502
            54 A5    B4         00C6   503 120$:     CLRW    CDRP$W_SENDSEQNM(R5)    ; Clean out sequence number (just in case)
         65 1C A6    D0         00C9   504          MOVL    CSB$L_RESENDQFL(R6), -  ; Put at front of RESEND list
                                00CD   505                  CDRP$L_FQFL(R5)
               04    12         00CD   506          BNEQ    130$                    ; Branch if not first in list
      20 A6    65    DE         00CF   507          MOVAL   CDRP$L_FQFL(R5), -      ; Update back pointer
                                00D3   508                  CSB$L_RESENDQBL(R6)
      1C A6    65    DE         00D3   509 130$:     MOVAL   CDRP$L_FQFL(R5), -      ; Update list head pointer
                                00D7   510                  CSB$L_RESENDQFL(R6)
               B6    11         00D7   511          BRB     100$
```

```
                        00D9    512
                        00D9    513 140$:
                        00D9    514 ;
                        00D9    515 ; Locate and prefix onto the resend list CDRPs left in the RDT with sequence numbers
                        00D9    516 ; of zero (these are messages that have been acknowledged and may have CNXSTATE =
                        00D9    517 ; NORMAL, REQUESTOR, or PARTNER).
                        00D9    518 ;
                        00D9    519             ASSUME  <CSB$L_CDT+4>,EQ,CSB$L_PDT
53   0C A6    7D        00D9    520             MOVQ    CSB$L_CDT(R6),R3        ; Restore CDT and PDT address
                        00DD    521             SCAN_RDT action=MERGE_CDRP
                        00E8    522
   55   56    D0        00E8    523             MOVL    R6, R5                  ; Restore CSB address in R5.
        56 8ED0         00EB    524             POPL    R6                      ; Restore saved R6.
             05         00EE    525             RSB
                        00EF    526
                        00EF    527
```

```
                     00EF    529          .SBTTL  CNX$POST_CLEANUP - Cleanup Outstanding Messages after Disconnecting
                     00EF    530
                     00EF    531 ;++
                     00EF    532 ;
                     00EF    533 ; FUNCTIONAL DESCRIPTION:
                     00EF    534 ;
                     00EF    535 ;       This routine is called by SCS when a connection breaks, after
                     00EF    536 ;       a DISCONNECT has completed.
                     00EF    537 ;       The major purpose of this routine is to deallocate map resources.
                     00EF    538 ;
                     00EF    539 ; CALLING SEQUENCE:
                     00EF    540 ;
                     00EF    541 ;       JSB     CNX$POST_CLEANUP
                     00EF    542 ;       IPL is at SCS fork level (8)
                     00EF    543 ;
                     00EF    544 ; INPUT PARAMETERS:
                     00EF    545 ;
                     00EF    546 ;       R5      Address of CSB
                     00EF    547 ;
                     00EF    548 ; IMPLICIT INPUTS:
                     00EF    549 ;
                     00EF    550 ;       None
                     00EF    551 ;
                     00EF    552 ; OUTPUT PARAMETERS:
                     00EF    553 ;
                     00EF    554 ;       None
                     00EF    555 ;
                     00EF    556 ; IMPLICIT OUTPUTS:
                     00EF    557 ;
                     00EF    558 ;       None
                     00EF    559 ;
                     00EF    560 ; SIDE EFFECTS:
                     00EF    561 ;
                     00EF    562 ;       R0-R4 destroyed
                     00EF    563 ;
                     00EF    564 ;--
                     00EF    565
                     00EF    566 CNX$POST_CLEANUP::
                     00EF    567
      7E    56   7D  00EF    568          MOVQ    R6,-(SP)                ; Save some registers.
            57   D4  00F2    569          CLRL    R7                      ; Sequence number checker
      56    55   D0  00F4    570          MOVL    R5, R6                  ; Copy CSB address.
                     00F7    571 ;
                     00F7    572 ; Scan resend list, unmapping REQUESTORs and PARTNERs
                     00F7    573 ;
55     1C A6   D0    00F7    574          MOVL    CSB$L_RESENDQFL(R6),R5  ; First CDRP in list
            51   13  00FB    575          BEQL    70$                     ; Branch when done
50     54 A5   3C    00FD    576 10$:     MOVZWL  CDRP$W_SENDSEQNM(R5),R0 ; Sequence number
            0D   13  0101    577          BEQL    30$                     ; Branch if zero
            57   B5  0103    578          TSTW    R7                      ; Send a real sequence number yet?
            05   12  0105    579          BNEQ    20$                     ; Branch if yes
      57    50   B0  0107    580          MOVW    R0,R7                   ; Use the first
            04   11  010A    581          BRB     30$
                     010C    582
            57   B6  010C    583 20$:     INCW    R7                      ; Bump expected sequence number
            FC   13  010E    584          BEQL    20$                     ; Avoid sequence number 0
                     0110    585
```

```
                54 A5   57   B1  0110   586 30$:    CMPW    R7,CDRP$W_SENDSEQNM(R5) ; Check ordering
                        04   13  0114   587          BEQL    40$                     ; Branch if as expected
                                0116   588          BUG_CHECK       CNXMGRERR,FATAL ; Mis-ordered RESEND list
                                011A   589
                                011A   590 40$:    DISPATCH        CDRP$B_CNXSTATE(R5),TYPE=B,PREFIX=CDRP$K_, -
                                011A   591                 < -
                                011A   592                 <NORMAL,60$>, -
                                011A   593                 <REQUESTOR,50$>, -
                                011A   594                 <PARTNER,45$>, -
                                011A   595                 <PART_IDLE,60$>, -
                                011A   596                 >
                                0127   597          BUG_CHECK       CNXMGRERR,FATAL ; Invalid CDRP state
                                012B   598
            0C A5   50 A5   D0  012B   599 45$:    MOVL    CDRP$L_SAVEPC(R5), -    ; Fix resumption address for
                                0130   600                 CDRP$L_FPC(R5)          ; block transfers that were in progress
                    22 A5   B5  0130   601          TSTW    CDRP$L_RSPID+2(R5)      ; Is there a RSPID allocated?
                        0A  13  0133   602          BEQL    50$                     ; Branch if no
                                0135   603          DEALLOC_RSPID                   ; Deallocate RSPID
                20 A5   01  D0  013B   604          MOVL    #1, CDRP$L_RSPID(R5)    ; Indicate that a RSPID will be needed.
                                013F   605 50$:    ASSUME  CSB$L_PDT,EQ,<CSB$L_CDT+4>
                53   0C A6  7D  013F   606          MOVQ    CSB$L_CDT(R6),R3        ; Fetch CDT, PDT addresses
                                0143   607          UNMAP                           ; Deallocate mapping resources
                    24 A5   D4  0146   608 60$:    CLRL    CDRP$L_CDT(R5)          ; Clean out obsolete CDT address
                55  65      D0  0149   609          MOVL    CDRP$L_FQFL(R5),R5      ; Link to next CDRP
                        AF  12  014C   610          BNEQ    10$                     ; Continue scan
                                014E   611
                        57  B5  014E   612 70$:    TSTW    R7                      ; Were any sequence numbers found?
                        06  13  0150   613          BEQL    80$                     ; Branch if no
                2C A6   57  B1  0152   614          CMPW    R7,CSB$W_SENDSEQNM(R6)  ; Must match last used number
                        0A  12  0156   615          BNEQ    90$                     ; Branch on mismatch
                                0158   616 80$:
                  32 A6 94  0158   617          CLRB    CSB$B_UNACKEDMSGS(R6)   ; By definition, no messages need ACKs
            55  56      D0  015B   618          MOVL    R6,R5                   ; CSB Address
            56  8E      7D  015E   619          MOVQ    (SP)+,R6                ; Restore R6 and R7
                        05  0161   620          RSB
                                0162   621
                                0162   622 90$:    BUG_CHECK       CNXMGRERR,FATAL ; Sequence number error
                                0166   623
```

```
                     0166   625          .SBTTL  FLUSH_WARMCDRPS - Flush warm CDRP cache
                     0166   626 ;++
                     0166   627 ; FUNCTIONAL DESCRIPTION
                     0166   628 ;
                     0166   629 ;       This routine is called to deallocate all resources from all
                     0166   630 ;       all CDRPs in the warm CDRP cache.  Note that deallocating
                     0166   631 ;       resources may resume other threads.
                     0166   632 ;
                     0166   633 ; CALLING SEQUENCE:
                     0166   634 ;
                     0166   635 ;       BSBW    FLUSH_WARMCDRPS
                     0166   636 ;       IPL must be at IPL$_SCS
                     0166   637 ;
                     0166   638 ; INPUTS:
                     0166   639 ;
                     0166   640 ;       R3      Address of CSB
                     0166   641 ;
                     0166   642 ; OUTPUTS:
                     0166   643 ;
                     0166   644 ;       NONE
                     0166   645 ;
                     0166   646 ; SIDE EFFECTS:
                     0166   647 ;
                     0166   648 ;       Note that other threads may be resumed as we deallocate resources.
                     0166   649 ;       R0 - R2 destroyed.
                     0166   650 ;--
                     0166   651
                     0166   652 FLUSH_WARMCDRPS:
           28   BB   0166   653          PUSHR   #^M<R3,R5>
55    24 B3   0F   0168   654 10$:       REMQUE  @CSB$L_WARMCDRPQFL(R3), R5 ; Get the next warm CDRP.
              0C   1D   016C   655          BVS     20$                     ; Branch if no warm CDRPs left.
           42 A3   97   016E   656          DECB    CSB$B_WARMCDRPS(R3)     ; Adjust count
52    1C A5   D0   0171   657          MOVL    CDRP$L_MSG_BUF(R5),R2   ; Message buffer address
            0816   30   0175   658          BSBW    DEALLOC_WARMCDRP        ; Deallocate warm CDRP
              EE   11   0178   659          BRB     10$                     ; Loop till no more warm CDRPs.
                     017A   660
           42 A3   95   017A   661 20$:       TSTB    CSB$B_WARMCDRPS(R3)     ; Make sure count is correct
              03   12   017D   662          BNEQ    80$                     ; Error!
              28   BA   017F   663          POPR    #^M<R3,R5>
                   05   0181   664          RSB
                     0182   665
                     0182   666 80$:       BUG_CHECK       CNXMGRERR,FATAL ; Warm CDRP count and queue disagree
```

```
                     0186   668          .SBTTL  CLEANUP_CDRP - Routine to cleanup a CDRP
                     0186   669 ;++
                     0186   670 ;
                     0186   671 ; FUNCTIONAL DESCRIPTION:
                     0186   672 ;
                     0186   673 ;       This routine is called when SCS resources held by the CDRP must
                     0186   674 ;       be deallocated.
                     0186   675 ;
                     0186   676 ; INPUTS:
                     0186   677 ;
                     0186   678 ;       R3      CSB address
                     0186   679 ;       R5      CDRP address
                     0186   680 ;
                     0186   681 ; IMPLICIT INPUTS:
                     0186   682 ;
                     0186   683 ;       It is assumed that the input CDRP makes no use of the CDRP$L_RWCPTR.
                     0186   684 ;       That field is ignored.
                     0186   685 ;
                     0186   686 ; OUTPUTS:
                     0186   687 ;
                     0186   688 ;       None.
                     0186   689 ;
                     0186   690 ; SIDE EFFECTS:
                     0186   691 ;
                     0186   692 ;       SCS resources held by input CDRP are deallocated.  This may cause
                     0186   693 ;       other threads to begin executing.
                     0186   694 ;
                     0186   695 ;       R0,R1,R2 are destroyed.
                     0186   696 ;--
                     0186   697
                     0186   698 CLEANUP_CDRP:
                     0186   699
          18   BB    0186   700          PUSHR   #^M<R3,R4>                      ; Save registers
0A A5     39   91    0188   701          CMPB    #DYN$C_CDRP, -                  ; Is this a CDRP structure?
                     018C   702                  CDRP$B_CD_TYPE(R5)
          3E   12    018C   703          BNEQ    900$                            ; Branch if not a CDRP (very bad).
                     018E   704          ASSUME  <CSB$L_CDT+4> EQ CSB$L_PDT
53   0C   A3   7D    018E   705          MOVQ    CSB$L_CDT(R3),R3                ; CDT address, PDT address
     22   A5   B5    0192   706          TSTW    CDRP$L_RSPID+2(R5)              ; Holding a RSPID?
          0C   13    0195   707          BEQL    40$                             ; Branch if not holding a RSPID.
          37   10    0197   708          BSBB    CHECK_RSPID                     ; Check for valid RSPID.
                     0199   709          DEALLOC_RSPID                           ; Deallocate the RSPID.
20 A5     01   D0    019F   710          MOVL    #1, CDRP$L_RSPID(R5)            ; Indicate that a RSPID will be needed.
                     01A3   711 40$:
     1C   A5   D5    01A3   712          TSTL    CDRP$L_MSG_BUF(R5)              ; Is a message buffer held by CDRP?
          07   13    01A6   713          BEQL    50$                             ; Branch if no message buffer held.
          54   D5    01A8   714          TSTL    R4                              ; Is PDT defined?
          1C   13    01AA   715          BEQL    800$                            ; Branch if no
                     01AC   716          DEALLOC_MSG_BUF                         ; Deallocate the message buffer.
                     01AF   717 50$:
                     01AF   718          DISPATCH        CDRP$B_CNXSTATE(R5),TYPE=B,PREFIX=CDRP$K_, -
                     01AF   719                  < -
                     01AF   720                  <NORMAL,70$>, -                 ; Normal messages
                     01AF   721                  <PARTNER,60$>, -                ; Block transfer partners
                     01AF   722                  <REQUESTOR,60$>, -              ; Block transfer requestors
                     01AF   723                  >
                     01BA   724          BUG_CHECK       CNXMGRERR,FATAL ; Unexpected CDRP state
```

```
                                                H  6

                 01BE   725
54   D5  01BE   726 60$:    TSTL    R4                      ; PDT still defined?
03   13  01C0   727         BEQL    70$                     ; Branch if not
         01C2   728         UNMAP                           ; Release buffer handle
18   BA  01C5   729 70$:    POPR    #^M<R3,R4>              ; Restore registers
     05  01C7   730         RSB
         01C8   731
         01C8   732 800$:   BUG_CHECK       CNXMGRERR,FATAL ; Can't deallocate resource
         01CC   733
         01CC   734 900$:   BUG_CHECK       CNXMGRERR,FATAL ; Data structure not a CDRP.
```

```
                                                I  6
01D0   736                .SBTTL  CHECK_RSPID - Validate RSPID in given CDRP
01D0   737 ;++
01D0   738 ;
01D0   739 ; FUNCTIONAL DESCRIPTION:
01D0   740 ;
01D0   741 ;       This routine validates the RSPID in the CDRP whose address is in R5.
01D0   742 ;       If the RSPID in the CDRP cannot be located in the RDT or if the RDT
01D0   743 ;       entry associated with the RSPID points to something other that in
01D0   744 ;       given CDRP, the system is bugchecked.
01D0   745 ;
01D0   746 ; INPUTS:
01D0   747 ;
01D0   748 ;       R5      Address of a CDRP
01D0   749 ;
01D0   750 ; IMPLICIT INPUTS:
01D0   751 ;
01D0   752 ;       CDRP$L_RSPID(R5) a RSPID
01D0   753 ;       The RDT.
01D0   754 ;
01D0   755 ; OUTPUTS:
01D0   756 ;
01D0   757 ;       None.
01D0   758 ;
01D0   759 ; IMPLICIT OUTPUTS:
01D0   760 ;
01D0   761 ;       R0 and R1 destroyed.
01D0   762 ;       All other registers preserved.
01D0   763 ;
01D0   764 ; SIDE EFFECTS:
01D0   765 ;
01D0   766 ;       System is bugchecked if error is located in RSPID.
01D0   767 ;--
01D0   768
01D0   769 CHECK_RSPID:
01D0   770
     55    DD  01D0   771          PUSHL   R5                              ; Save input CDRP address.
55   20 A5    D0  01D2   772          MOVL    CDRP$L_RSPID(R5), R5            ; Get RSPID.
                  01D6   773          FIND_RSPID_RDTE                         ; Locate RDTE for this RSPID.
     09 50    E9  01DC   774          BLBC    R0, 900$                        ; Branch if lookup failed.
   6E    65    D1  01DF   775          CMPL    RD$L_CDRP(R5), (SP)             ; Is the CDRP address right.
         04    12  01E2   776          BNEQ    900$                            ; Branch if address is wrong.
      55 8ED0    05  01E4   777          POPL    R5                              ; Restore CDRP address.
                 05  01E7   778          RSB                                     ; Return to caller.
                  01E8   779
                  01E8   780 900$:    BUG_CHECK       CNXMGRERR,FATAL ; RSPID is wrong.
```

```
                                J 6
01EC    782          .SBTTL  MERGE_CDRP - Scan action routine to merge a CDRP
01EC    783 ;++
01EC    784 ;
01EC    785 ; FUNCTIONAL DESCRIPTION:
01EC    786 ;
01EC    787 ;       This action routine is called when a CDRP thread is located in the RDT
01EC    788 ;       after all other processing has been completed.  This routine finds CDRPs
01EC    789 ;       that have been acknowledged and are no longer on the SENT list as well
01EC    790 ;       as CDRPs that have not yet been ackowledged and are still on the SENT
01EC    791 ;       list.  Acknowledged CDRPs can be identified by fact that they have a
01EC    792 ;       zero send sequence number.  Acknowledged CDRPs are inserted onto the
01EC    793 ;       head of the RESEND list in no particular order.  Note that block
01EC    794 ;       transfer requests always look as though they have been acknowledged
01EC    795 ;       and are never on the SENT list.
01EC    796 ;
01EC    797 ; INPUTS:
01EC    798 ;
01EC    799 ;       R3      CDT address
01EC    800 ;       R4      PDT address
01EC    801 ;       R5      located CDRP address
01EC    802 ;
01EC    803 ; IMPLICIT INPUTS:
01EC    804 ;
01EC    805 ;       CDT$L_AUXSTRUC(R3) CSB address (we could use R6, but that would assume
01EC    806 ;       that the SCS lookup routines do not corrupt it).
01EC    807 ;
01EC    808 ;       The input CDRP is assumed to be on the SENT list if the sequence number
01EC    809 ;       is non-zero and on no list or queue if the sequence number is zero.
01EC    810 ;
01EC    811 ;       It is assumed that the RSPID held by this located CDRP has been
01EC    812 ;       transmitted to a remote node and must be retained for future
01EC    813 ;       identification of the response from that node.
01EC    814 ;
01EC    815 ;       It is assumed that the input CDRP makes no use of the CDRP$L_RWCPTR.
01EC    816 ;       That field is ignored.
01EC    817 ;
01EC    818 ; OUTPUTS:
01EC    819 ;
01EC    820 ;       None.
01EC    821 ;
01EC    822 ; IMPLICIT OUTPUTS:
01EC    823 ;
01EC    824 ;       If the sequence number is zero, indicating a CDRP not on the SENT list,
01EC    825 ;       the input CDRP is inserted at the head of the CSB resend list.
01EC    826 ;
01EC    827 ; SIDE EFFECTS:
01EC    828 ;
01EC    829 ;       None.
01EC    830 ;--
01EC    831
01EC    832 MERGE_CDRP:
01EC    833
0A A5   39   91 01EC    834          CMPB    #DYN$C_CDRP, -          ; Is this a CDRP structure?
01F0    835                          CDRP$B_CD_TYPE(R5)
        2E   12 01F0    836          BNEQ    900$                    ; Branch if not a CDRP (very bad).
        DC   10 01F2    837          BSBB    CHECK_RSPID             ; Validate the RSPID.
01F4    838
```

```
                        01F4   839          DISPATCH        CDRP$B_CNXSTATE(R5),TYPE=B,PREFIX=CDRP$K_, -
                        01F4   840                                  < -
                        01F4   841                                  <NORMAL,20$>, -          ; Normal message
                        01F4   842                                  <REQUESTOR,20$>, -      ; Block transfer request
                        01F4   843                                  <PARTNER,10$>, -        ; Block transfer partner
                        01F4   844                                  >
                        01FF   845          BUG_CHECK       CNXMGRERR,FATAL ; Invalid CDRP state
                        0203   846
OC A5    1F'AF   9E     0203   847 10$:     MOVAB   B^40$,CDRP$L_FPC(R5)    ; Set up completion address
   52    5C A3   D0     0208   848 20$:     MOVL    CDT$L_AUXSTRUC(R3), R2  ; Get CSB address.
         54 A5   B5     020C   849          TSTW    CDRP$W_SENDSEQNM(R5)    ; If nonzero, ignore this CDRP
            0E   12     020F   850          BNEQ    40$
   65    1C A2   D0     0211   851          MOVL    CSB$L_RESENDQFL(R2), -  ; Link to head of RESEND list
                        0215   852                  CDRP$L_FQFL(R5)
            04   12     0215   853          BNEQ    30$                     ; Branch if list already populated
   20 A2    65   DE     0217   854          MOVAL   CDRP$L_FQFL(R5), -      ; Set up tail pointer
                        021B   855                  CSB$L_RESENDQBL(R2)
   1C A2    65   DE     021B   856 30$:     MOVAL   CDRP$L_FQFL(R5), -      ; Set up new head pointer
                        021F   857                  CSB$L_RESENDQFL(R2)
                   05   021F   858 40$:     RSB
                        0220   859
                        0220   860 900$:    BUG_CHECK       CNXMGRERR,FATAL ; Data structure not a CDRP.
```

```
                     0224   862          .SBTTL  CNX$FAIL_MSG - Complete outstanding I/O with failure status
                     0224   863 ;++
                     0224   864 ; FUNCTIONAL DESCRIPTION:
                     0224   865 ;
                     0224   866 ;       All un-acked messages have their fork process resumed with
                     0224   867 ;       a failure status.
                     0224   868 ;
                     0224   869 ; CALLING SEQUENCE:
                     0224   870 ;
                     0224   871 ;       BSBW    CNX$FAIL_MSG
                     0224   872 ;       IPL must be at IPL$_SCS
                     0224   873 ;
                     0224   874 ; INPUT PARAMETERS:
                     0224   875 ;
                     0224   876 ;       R5      Address of CSB
                     0224   877 ;
                     0224   878 ; OUTPUT PARAMETERS:
                     0224   879 ;
                     0224   880 ;       None
                     0224   881 ;
                     0224   882 ; SIDE EFFECTS:
                     0224   883 ;
                     0224   884 ;       R0 and R1 are destroyed.
                     0224   885 ;--
                     0224   886
                     0224   887 CNX$FAIL_MSG::
     007C 8F    BB   0224   888          PUSHR   #^M<R2,R3,R4,R5,R6>     ; Save registers
     56   55    D0   0228   889          MOVL    R5,R6                   ; Move address of CSB
55     1C A6    D0   022B   890 10$:     MOVL    CSB$L_RESENDQFL(R6),R5  ; Remove CDRP from head
            2E   13   022F   891          BEQL    60$                     ; Queue empty
   1C A6    65   D0   0231   892          MOVL    CDRP$L_FQFL(R5), -      ; Update queue head
                   0235   893                  CSB$L_RESENDQFL(R6)
            05   12   0235   894          BNEQ    20$                     ; Branch if queue not empty
20 A6    1C A6    DE   0237   895          MOVAL   CSB$L_RESENDQFL(R6), -  ; Update end pointer
                   023C   896                  CSB$L_RESENDQBL(R6)
            65   7C   023C   897 20$:     CLRQ    CDRP$L_FQFL(R5)         ; Zap queue linkage
          24 A5  D4   023E   898          CLRL    CDRP$L_CDT(R5)          ; Invalidate CDT address
          22 A5  B5   0241   899          TSTW    CDRP$L_RSPID+2(R5)      ; Is there a RSPID?
             0A  13   0244   900          BEQL    30$                     ; Branch if no RSPID
                   0246   901          DEALLOC_RSPID
       20 A5    01   D0   024C   902          MOVL    #1,CDRP$L_RSPID(R5)     ; Set RSPID neeed flag
50     223C 8F    3C   0250   903 30$:     MOVZWL  #SS$_NODELEAVE,R0       ; Indicate error to fork process
       53   56    D0   0255   904          MOVL    R6,R3                   ; Restore CSB address
            54    D4   0258   905          CLRL    R4                      ; Bug trap
                   025A   906
                   025A   907          ; Resume Fork process.  Inputs are:
                   025A   908          ;
                   025A   909          ;       R0      SS$_NODELEAVE (Indicates failover)
                   025A   910          ;       R3      Address of CSB
                   025A   911          ;       R5      Address of CDRP
                   025A   912
          0C B5    16   025A   913          JSB     @CDRP$L_FPC(R5)         ; Resume fork process
             CC    11   025D   914          BRB     10$                     ; Continue until queue is empty
                   025F   915
     007C 8F    BA   025F   916 60$:     POPR    #^M<R2,R3,R4,R5,R6>     ; Restore registers
                05   0263   917          RSB
```

```
                     0264   919          .SBTTL  CNX$RESEND_MSGS - Resend messages
                     0264   920
                     0264   921 ; ++
                     0264   922 ;
                     0264   923 ; FUNCTIONAL DESCRIPTION:
                     0264   924 ;
                     0264   925 ;       This routine uses the the last sequence number the remote
                     0264   926 ;       side received to resume the fork process for all messages
                     0264   927 ;       that have been acked and to resend all messages that weren't
                     0264   928 ;       acked.  In addition all messages that have been queued since the
                     0264   929 ;       previous connection broke are sent.
                     0264   930 ;
                     0264   931 ; CALLING SEQUENCE:
                     0264   932 ;
                     0264   933 ;       BSBW    CNX$RESEND_MSGS
                     0264   934 ;
                     0264   935 ; INPUT PARAMETERS:
                     0264   936 ;
                     0264   937 ;       R5      Address of CSB
                     0264   938 ;
                     0264   939 ; IMPLICIT INPUTS:
                     0264   940 ;
                     0264   941 ;       CSB$W_ACKRSEQNM contains last sequence number (of ours) received by
                     0264   942 ;       remote side  (equivalent to CLSMSG$L_ACKSEQ).
                     0264   943 ;
                     0264   944 ;       It is assumed that the resend queue contains only normal and block
                     0264   945 ;       transfer requestor CDRPs.
                     0264   946 ;
                     0264   947 ; OUTPUT PARAMETERS:
                     0264   948 ;
                     0264   949 ;       None
                     0264   950 ;
                     0264   951 ; SIDE EFFECTS:
                     0264   952 ;
                     0264   953 ;       R0, and R1 are destroyed
                     0264   954 ;--
                     0264   955
                     0264   956 CNX$RESEND_MSGS::
                     0264   957
                     0264   958          ; Remove CDRPs from the CSB RESEND queue.  For each CDRP,
                     0264   959          ;       a)  If its sequence number is zero, reset the CDT address,
                     0264   960          ;           remove the CDRP from the list, and forget it.
                     0264   961          ;       b)  If its sequence number is less than or equal to the acked
                     0264   962          ;           sequence number, then if it doesn't have a RSPID
                     0264   963          ;           then resume its fork process.  If it does have a RSPID,
                     0264   964          ;           then reset the CDT address and skip over it.
                     0264   965          ;       c)  if its sequence number is greater than the acked
                     0264   966          ;           sequence, then resend it.
                     0264   967          ; This loop is terminated as soon as we find the first CDRP to resend
                     0264   968
    00BC 8F    BB    0264   969          PUSHR   #^M<R2,R3,R4,R5,R7>
    57   55    D0    0268   970          MOVL    R5,R7                           ; Address of CSB
55   1C A7    D0    026B   971 10$:      MOVL    CSB$L_RESENDQFL(R7),R5  ; Get the next CDRP
         5D    13    026F   972          BEQL    80$                             ; Branch if no more
50   54 A5    3C    0271   973          MOVZWL  CDRP$W_SENDSEQNM(R5),R0 ; CDRP's sequence number
         06    13    0275   974          BEQL    20$                             ; Branch if there is no sequence number
50   30 A7    A2    0277   975          SUBW    CSB$W_ACKRSEQNM(R7),R0  ; Compare with acknowledged sequence number
```

```
                51   14   027B    976              BGTR    80$                     ; Branch if it has not been ack'ed
          1C A7 65   D0   027D    977 20$:         MOVL    CDRP$L_FQFL(R5), -      ; Update list head pointer
                                  978                      CSB$L_RESENDQFL(R7)
                05   12   0281    979              BNEQ    30$                     ; Branch if list not empty
       20 A7  1C A7 DE   0283    980              MOVAL   CSB$L_RESENDQFL(R7), -  ; Point end at list head
                                  981                      CSB$L_RESENDQBL(R7)
                                  982 30$:         DISPATCH        CDRP$B_CNXSTATE(R5),TYPE=B,PREFIX=CDRP$K_, -
                                  983                      < -
                                  984                      <NORMAL,50$>, -         ; If normal message
                                  985                      <REQUESTOR,50$>, -      ; If block transfer
                                  986                      <PARTNER,40$>, -        ; Fail partner
                                  987                      <PART_IDLE,40$>, -      ; Fail idle partner
                                  988                      >
                                  989              BUG_CHECK       CNXMGRERR,FATAL ; Inconsistent state -- should never get her
                                  990
                                  991 40$:
                                  992              ; Have a PARTNER-type CDRP that must be failed.
                                  993              ; Inputs to fork process are:
                                  994              ;
                                  995              ;       R0      contains 0 (failure)
                                  996              ;       R3      Address of CSB
                                  997              ;       R4      Address of PDT
                                  998              ;       R5      Address of CDRP
                                  999              ;
                                 1000              ; Fork routine may use R0 - R5.
                                 1001
          53   57   D0   0299   1002              MOVL    R7,R3                   ; CSB address
       54   10 A3   D0   029C   1003              MOVL    CSB$L_PDT(R3),R4        ; PDT address
                50   D4   02A0   1004              CLRL    R0                      ; Set failure status
             0C B5   16   02A2   1005              JSB     @CDRP$L_FPC(R5)         ; Resume fork process
                C4   11   02A5   1006              BRB     10$                     ; Continue processing
                          02A7   1007
                          02A7   1008 50$:
                          02A7   1009 ;
                          02A7   1010 ; Have a CDRP with a non-zero sequence number
                          02A7   1011 ;
    24 A5    0C A7   D0   02A7   1012              MOVL    CSB$L_CDT(R7), -        ; Update CDT address in CDRP
                          02AC   1013                      CDRP$L_CDT(R5)
             22 A5   B5   02AC   1014              TSTW    CDRP$L_RSPID+2(R5)      ; Is there a RSPID?
                05   13   02AF   1015              BEQL    60$                     ; Branch if no RSPID
             54 A5   B4   02B1   1016              CLRW    CDRP$W_SENDSEQNM(R5)    ; Flag it acknowledged
                B5   11   02B4   1017              BRB     10$                     ; Note that the RDT still point this this on
                          02B6   1018
             54 A5   B5   02B6   1019 60$:         TSTW    CDRP$W_SENDSEQNM(R5)    ; Is there a sequence number?
                0F   13   02B9   1020              BEQL    70$                     ; Branch and bugcheck if no sequence number
                          02BB   1021
                          02BB   1022              ; Have a CDRP whose message has been ack'ed and who doesn't
                          02BB   1023              ; have a response id.  Inputs to fork process are:
                          02BB   1024              ;
                          02BB   1025              ;       R0      contains 1 (successful acknowledge)
                          02BB   1026              ;       R3      Address of CSB
                          02BB   1027              ;       R4      Address of PDT
                          02BB   1028              ;       R5      Address of CDRP
                          02BB   1029              ;
                          02BB   1030              ; Fork routine may use R0 - R5.
                          02BB   1031
          53   57   D0   02BB   1032              MOVL    R7,R3                   ; CSB address
```

```
54   10 A3   D0  02BE  1033          MOVL    CSB$L_PDT(R3),R4          ; PDT address
     50 01   D0  02C2  1034          MOVL    #SS$_NORMAL,R0            ; Get successful acknowledge
   0C B5     16  02C5  1035          JSB     @CDRP$L_FPC(R5)           ; Resume fork process
      A1     11  02C8  1036          BRB     10$                       ; process and continue
                 02CA  1037
                 02CA  1038 70$:     BUG_CHECK       CNXMGRERR,FATAL ; Missing sequence number
                 02CE  1039
                 02CE  1040 80$:
                 02CE  1041          ; Resend any messages that must be resent.
                 02CE  1042          ; If there are no messages to resend, this will zero CURRCDRP
                 02CE  1043          ; and allow future messages to go through.
                 02CE  1044
53   57   D0     02CE  1045          MOVL    R7,R3                     ; Set up CSB address
   011B   30     02D1  1046          BSBW    RESEND_MSG                ; Resend it and start pipeline
                 02D4  1047
00BC 8F   BA     02D4  1048          POPR    #^M<R2,R3,R4,R5,R7>
          05     02D8  1049          RSB
```

```
02D9  1051          .SBTTL  CNX$SEND_MSG - Send an acknowledged message
02D9  1052          .SBTTL  CNX$SEND_MSG_CSB - Send a message using CSB
02D9  1053          .SBTTL  CNX$SEND_MSG_RSPID - Send a message with response id
02D9  1054          .SBTTL  CNX$SEND_MSG_RESP - Send a message & recycle message buffer
02D9  1055
02D9  1056 ;++
02D9  1057 ;
02D9  1058 ; FUNCTIONAL DESCRIPTION:
02D9  1059 ;
02D9  1060 ;       This routine sends an acknowledged message.  An acknowledged message
02D9  1061 ;       is one that is guaranteed to be received by VMS at the remote
02D9  1062 ;       system or a failover is initiated.  The message is automatically
02D9  1063 ;       resent if the connection breaks and another connection is
02D9  1064 ;       subsequently established to the same system and software
02D9  1065 ;       incarnation.  Furthermore, the caller of this routine is returned
02D9  1066 ;       to when the message has been acknowledged.  The caller's caller
02D9  1067 ;       is returned to immediately.
02D9  1068 ;
02D9  1069 ; CALLING SEQUENCE:
02D9  1070 ;
02D9  1071 ;       BSBW    CNX$SEND_MSG            Send a message
02D9  1072 ;       BSBW    CNX$SEND_MSG_CSB        Send a message using CSB address
02D9  1073 ;       BSBW    CNX$SEND_MSG_RSPID      Send a message with response id
02D9  1074 ;       BSBW    CNX$SEND_MSG_RESP       Send a message and recycle message bfr
02D9  1075 ;       BSBW    RESEND_MSG              Internal entry point (used for resends)
02D9  1076 ;       BSBW    SEND_UNSEQ_MSG          Internal entry point (used for block transfe
02D9  1077 ;
02D9  1078 ;       This routine returns to its caller when the message has been
02D9  1079 ;       acknowledged.  It returns to its caller's caller immediately.
02D9  1080 ;       The standard fork process convention that the caller must not
02D9  1081 ;       push anything onto the stack is in effect.
02D9  1082 ;       An exception is when R0 contains SS$_NOSUCHNODE return status.
02D9  1083 ;       In this case, the return address of the caller's orginal caller is
02D9  1084 ;       still on the top of the stack.  In some cases, this may require
02D9  1085 ;       special action on the part of this routine's caller.
02D9  1086 ;       The other exception is one case of SS$_NODELEAVE.  When an attempt
02D9  1087 ;       is made to send a message to a node in the LONG_BREAK state, a
02D9  1088 ;       synchronous return is made with the stack in the condition just
02D9  1089 ;       described.
02D9  1090 ;
02D9  1091 ;       IPL must be at IPL$_SCS
02D9  1092 ;
02D9  1093 ; INPUT PARAMETERS:
02D9  1094 ;
02D9  1095 ;       R2      Address of message buffer (CNX$SEND_MSG_RESP entry only)
02D9  1096 ;       R3      CSID (for all routines except CNX$SEND_MSG_CSB)
02D9  1097 ;       R3      CSB (CNX$SEND_MSG_CSB only)
02D9  1098 ;       R5      Address of CDRP
02D9  1099 ;
02D9  1100 ; IMPLICIT INPUTS:
02D9  1101 ;
02D9  1102 ;       CDRP$L_MSGBLD must contain the address of a message build routine.
02D9  1103 ;
02D9  1104 ;       CDRP$L_RSPID must contain one of the following values:
02D9  1105 ;               0               No RSPID allocated and none needed
02D9  1106 ;               1               No RSPID allocated but one is needed
02D9  1107 ;               A valid RSPID   A RSPID is needed and is already allocated.
```

```
                     02D9  1108 ;
                     02D9  1109 ;       CDRP$L_MSG_BUF must contain a valid message buffer address or zero.
                     02D9  1110 ;
                     02D9  1111 ;       Any information that the message build routine requires should
                     02D9  1112 ;       be in the CDRP or pointed to by pointers in the CDRP.
                     02D9  1113 ;
                     02D9  1114 ;       This routine requires that several CDRP fields be initialized to zero.
                     02D9  1115 ;       CNX$INIT_CDRP should be called to perform this initialization.
                     02D9  1116 ;
                     02D9  1117 ;
                     02D9  1118 ; OUTPUT PARAMETERS:
                     02D9  1119 ;
                     02D9  1120 ;       R0      Status
                     02D9  1121 ;                   SS$_NORMAL ==> Message successfully acknowledged
                     02D9  1122 ;                                  (if response requested, response received)
                     02D9  1123 ;                   SS$_NOSUCHNODE ==> Invalid CSID (Not possible for
                     02D9  1124 ;                                  CNX$SEND_MSG_CSB.  N.B. no fork occurs in
                     02D9  1125 ;                                  this case)
                     02D9  1126 ;                   SS$_NODELEAVE ==> Requested node is leaving the cluster
                     02D9  1127 ;                                  or you are (a fork may or may not have occurred)
                     02D9  1128 ;       R2      Message buffer address
                     02D9  1129 ;                                  (if response requested and R0 = SS$_NORMAL)
                     02D9  1130 ;       R3
                     02D9  1131 ;               If status is anything but SS$_NOSUCHNODE :      CSB
                     02D9  1132 ;               If status is SS$_NOSUCHNODE :                   CSID
                     02D9  1133 ;       R4
                     02D9  1134 ;               If status is SS$_NOSUCHNODE :                   unchanged
                     02D9  1135 ;               If status is SS$_NODELEAVE (synchronous return) :      0
                     02D9  1136 ;               In all other caes :                             PDT address
                     02D9  1137 ;       R5      CDRP address
                     02D9  1138 ;
                     02D9  1139 ; IMPLICIT OUTPUTS:
                     02D9  1140 ;
                     02D9  1141 ;       None
                     02D9  1142 ;
                     02D9  1143 ; SIDE EFFECTS:
                     02D9  1144 ;
                     02D9  1145 ;       R0 - R2 and R4 are destroyed.
                     02D9  1146 ;
                     02D9  1147 ;--
                     02D9  1148
                     02D9  1149         .ENABLE LSB
                     02D9  1150
                     02D9  1151 ;
                     02D9  1152 ; Error in input CSID.
                     02D9  1153 ;       Cleanup allocated SCS resources and return SS$_NOSUCHNODE immediately.
                     02D9  1154 ;
                     02D9  1155 ; N.B. This is the synchronous return from CNX$SEND_MSG.  See notes in
                     02D9  1156 ; module header above.
                     02D9  1157 ;
                     02D9  1158
                     02D9  1159 SEND_CSID_ERROR:
7E   028C 8F    3C   02D9  1160         MOVZWL  #SS$_NOSUCHNODE,-(SP)   ; Set bad CSID error status.
          2C    11   02DE  1161         BRB     20$                     ; Cleanup and return synchronously
                     02E0  1162
                     02E0  1163
     FE83         30 02E0  1164 10$:    BSBW    FLUSH_WARMCDRPS         ; Flush warm CDRP cache
```

```
                34 A3    D5   02E3  1165          TSTL    CSB$L_CURRCDRP(R3)      ; Increment count again
                   7B    13   02E6  1166          BEQL    SEND_MSG_NOWAIT         ; Don't wait after all
                 FE9B    30   02E8  1167          BSBW    CLEANUP_CDRP            ; Deallocate RSPID, MAP, and MSGBUF resource
                34 A3    D5   02EB  1168          TSTL    CSB$L_CURRCDRP(R3)      ; Increment count again
                   73    13   02EE  1169          BEQL    SEND_MSG_NOWAIT         ; Don't wait after all
                32 A3    94   02F0  1170          CLRB    CSB$B_UNACKEDMSGS(R3)   ; Prevent explicit ACK attempts
                0C A5  8ED0   02F3  1171          POPL    CDRP$L_FPC(R5)          ; Save our caller's PC in fork block
                   65    D4   02F7  1172          CLRL    CDRP$L_FQFL(R5)         ; Zero end of list
          20 B3    55    D0   02F9  1173          MOVL    R5,@CSB$L_RESENDQBL(R3) ; Link this CDRP at end of list
          20 A3    55    D0   02FD  1174          MOVL    R5,CSB$L_RESENDQBL(R3)  ; Update end of list pinter
                   05    0301  1175          RSB                             ; Return to caller's caller
                         0302  1176
                         0302  1177 CDRP_MUST_WAIT:
                         0302  1178          ; Another CDRP is in resource wait or the connection is currently
                         0302  1179          ; down.  Place CDRP on resend queue and return to our caller's caller.
                         0302  1180          ;
                         0302  1181          ; NOTE:  The warm CDRP cache must be flushed AFTER we insert
                         0302  1182          ; this CDRP on the queue for the following reason.  Flushing the
                         0302  1183          ; cache (and the resources in this CDRP) may free up the waiting
                         0302  1184          ; thread which will in turn try to resume other waiters.  It may
                         0302  1185          ; seem that the correct solution to this is to insert this CDRP
                         0302  1186          ; on the RESEND queue BEFORE flushing the cache.  The reason this
                         0302  1187          ; doesn't work is that we can't deallocate both the RSPID and message
                         0302  1188          ; buffer atomically.  In other words, deallocation of the RSPID
                         0302  1189          ; might start up this very CDRP (if it were on the queue) before
                         0302  1190          ; the RSPID field had been set to 1 and before the message buffer
                         0302  1191          ; had been deallocated.  This problem could be circumvented (and
                         0302  1192          ; the cache flushed after the INSQUE) if we could deallocate the RSPID
                         0302  1193          ; with a register entry point (message buffers already can be
                         0302  1194          ; deallocated with a register entry point).  Then we could pick up
                         0302  1195          ; the resources, initialize the CDRP, and then deallocate the
                         0302  1196          ; resources.  This, in turn, might start up the very CDRP we had
                         0302  1197          ; just INSQUEd.
                         0302  1198
       D9 60 A3    00    E1   0302  1199          BBC     #CSB$V_LONG_BREAK, -    ; Try to free resources if no long
                                0307  1200                  CSB$L_STATUS(R3),10$    ;   connection break has yet occurred
        7E   223C 8F    3C   0307  1201          MOVZWL  #SS$_NODELEAVE,-(SP)    ; Return status
                1C A5   D5   030C  1202 20$:     TSTL    CDRP$L_MSG_BUF(R5)      ; Is there a message buffer?
                   12   12   030F  1203          BNEQ    40$                     ; Branch if buffer present.
                22 A5   B5   0311  1204          TSTW    CDRP$L_RSPID+2(R5)      ; Is there a RSPID allocated?
                   0A   13   0314  1205          BEQL    30$                     ; Branch if no RSPID allocated.
                         0316  1206          DEALLOC_RSPID                   ; Else, deallocate RSPID.
             20 A5   01   D0   031C  1207          MOVL    #1, CDRP$L_RSPID(R5)    ; Indicate that a RSPID will be needed.
                   01   BA   0320  1208 30$:     POPR    #^M<R0>                 ; Fetch return status
                   05   0322  1209          RSB                             ; Return synchronously to caller.
                         0323  1210
                         0323  1211 40$:     BUG_CHECK       CNXMGRERR,FATAL ; CDRP contains message buffer which
                         0327  1212                                          ; can not be deallocated without
                         0327  1213                                          ; CSB / PDT context
                         0327  1214          .DISABLE        LSB
                         0327  1215          .ENABL  LSB
                         0327  1216
                         0327  1217 CNX$SEND_MSG_RSPID::
             20 A5   01   D0   0327  1218          MOVL    #1,CDRP$L_RSPID(R5)     ; Indicate a RSPID is needed
                   09   11   032B  1219          BRB     CNX$SEND_MSG
                         032D  1220
                         032D  1221 CNX$SEND_MSG_RESP::
```

```
                1C A5    52   D0  032D  1222          MOVL    R2,CDRP$L_MSG_BUF(R5)   ; Save message buffer address
                   04 A2 D0   0331  1223          MOVL    CLSMSG$L_RSPID(R2),-    ; Store RSPID to return in CDRP
                   58 A5          0334  1224                  CDRP$L_RETRSPID(R5)
                                 0336  1225
                                 0336  1226 CNX$SEND_MSG::
                                 0336  1227
                                 0336  1228          ; First determine if the connection is open.  If not, the CDRP
                                 0336  1229          ; is simply placed on the resend queue.  If the connection comes back,
                                 0336  1230          ; we will build and send the message then.  Otherwise we will
                                 0336  1231          ; do a failover.  If the connection is open then save
                                 0336  1232          ; our caller's return PC in the CDRP in case SCS calls (e.g.
                                 0336  1233          ; ALLOC_MSG_BUF OR ALLOC_RSPID) wait and return to our caller's
                                 0336  1234          ; caller.
                                 0336  1235          ; Finally, prepare to call message build routine.  Put the PDT address
                                 0336  1236          ; in R4, the CDT address in the CDRP, conditionally allocate a
                                 0336  1237          ; response id., and allocate or recycle a message buffer.
                                 0336  1238
                                 0336  1239          CSID_TO_CSB csb=R3, error=SEND_CSID_ERROR
                                 034F  1240
                                 034F  1241 CNX$SEND_MSG_CSB::
                   2C A3    B6   034F  1242 5$:      INCW    CSB$W_SENDSEQNM(R3)     ; Increment sequence number
                      FB    13   0352  1243          BEQL    5$                      ; Don't use zero as a sequence number
                   2C A3    B0   0354  1244          MOVW    CSB$W_SENDSEQNM(R3),-   ; Put sequence number into the CDRP
                   54 A5          0357  1245                  CDRP$W_SENDSEQNM(R5)
                                 0359  1246
                                 0359  1247          ; The block transfer code enters here to send an unsequenced message
                                 0359  1248          ; requesting data movement using the common resource allocation /
                                 0359  1249          ; cleanup apparatus.
                                 0359  1250
                                 0359  1251 SEND_UNSEQ_MSG:
                   0C A3    D0   0359  1252          MOVL    CSB$L_CDT(R3),-         ; Put CDT address into CDRP
                   24 A5          035C  1253                  CDRP$L_CDT(R5)
                                 035E  1254
                                 035E  1255          ; The following code begins a critical section meaning only one
                                 035E  1256          ; CDRP thread may be in this section at a time.
                                 035E  1257
                      34 A3    D5   035E  1258          TSTL    CSB$L_CURRCDRP(R3)      ; Branch if critical section is locked (>0)
                         9F    12   0361  1259          BNEQ    CDRP_MUST_WAIT          ; or busy (<0)
                                 0363  1260 SEND_MSG_NOWAIT:                          ; Return here if we don't wait after all
                34 A3    55    D0   0363  1261          MOVL    R5,CSB$L_CURRCDRP(R3)   ; This becomes the "current" CDRP
                   50 A5 8ED0   0367  1262          POPL    CDRP$L_SAVEPC(R5)       ; Save return PC
                                 036B  1263
                54    10 A3    D0   036B  1264          MOVL    CSB$L_PDT(R3),R4        ; Get PDT address
                                 036F  1265
                                 036F  1266 SEND_ALLOC:
                                 036F  1267
                                 036F  1268          ; Allocate resources
                                 036F  1269
                      20 A5    D5   036F  1270          TSTL    CDRP$L_RSPID(R5)        ; Is a response id needed?
                         0B    13   0372  1271          BEQL    10$                     ; Branch if no
                      22 A5    B5   0374  1272          TSTW    CDRP$L_RSPID+2(R5)      ; Yes, is a response id allocated?
                         06    12   0377  1273          BNEQ    10$                     ; Branch if yes
                                 0379  1274          ALLOC_RSPID                     ; No, allocate a response id.
                      1C A5    D5   037F  1275 10$:     TSTL    CDRP$L_MSG_BUF(R5)      ; Is there already a message buffer?
                         0A    13   0382  1276          BEQL    20$                     ; Branch if no
                                 0384  1277          RECYCL_MSG_BUF                  ; Yes, recycle it
                      0A 50    E8   0387  1278          BLBS    R0,30$                  ; Branch if no error
```

```
                        038A  1279 15$:    BUG_CHECK       CNXMGRERR,FATAL ; Error allocating/recycling message buffer
                        038E  1280
                        038E  1281 20$:    ALLOC_MSG_BUF                   ; Allocate a message buffer
                F6 50   E9  0391  1282     BLBC    R0,15$                  ; Branch on error
                        0394  1283
                        0394  1284 30$:    ; Now call the message build routine.  Inputs to this routine are:
                        0394  1285         ;
                        0394  1286         ;       R2      Address of message buffer
                        0394  1287         ;       R3      Address of CSB
                        0394  1288         ;       R4      Address of PDT
                        0394  1289         ;       R5      Address of CDRP
                        0394  1290         ;
                        0394  1291         ;       R0 and R1 may be destroyed.  Everything else must be preserved.
                        0394  1292
                4C B5   16  0394  1293     JSB     @CDRP$L_MSGBLD(R5)      ; Call message build routine
                        0397  1294
                        0397  1295         ; Add message header.  This consists of this message's sequence
                        0397  1296         ; number and the last received sequence number from the remote side.
                        0397  1297
             02 A2   2E A3  B0  0397  1298     MOVW    CSB$W_RCVDSEQNM(R3), -  ; Get highest received (remote) sequence
                        039C  1299                 CLSMSG$W_ACKSEQ(R2)     ; and return acknowledgement.
                62   54 A5  B0  039C  1300     MOVW    CDRP$W_SENDSEQNM(R5), - ; Get sequence number for this message
                        03A0  1301                 CLSMSG$W_SEQNUM(R2)
                32 A3   94  03A0  1302     CLRB    CSB$B_UNACKEDMSGS(R3)   ; Zero count of un-acked messages
                        03A3  1303
                        03A3  1304         ; Now send the message.  If there is a response id. then SCS will
                        03A3  1305         ; set up the fork block so we have to make it appear
                        03A3  1306         ; as if our caller called SEND_MSG_BUF.  Otherwise, we set up
                        03A3  1307         ; the fork block.
                        03A3  1308
                   65   D4  03A3  1309     CLRL    CDRP$L_FQFL(R5)         ; Clear linkage
             18 B3    55   D0  03A5  1310     MOVL    R5,@CSB$L_SENTQBL(R3)   ; Link to tail of sent list
             18 A3    55   D0  03A9  1311     MOVL    R5,CSB$L_SENTQBL(R3)    ; Update tail pointer
             50   20 A5   D0  03AD  1312     MOVL    CDRP$L_RSPID(R5),R0     ; Get response id. (if there is one)
                   2B   13  03B1  1313     BEQL    50$                     ; No response id.
                        03B3  1314
             04 A2    50   D0  03B3  1315     MOVL    R0,CLSMSG$L_RSPID(R2)   ; Store response id in message
                EF'AF   DF  03B7  1316     PUSHAL  B^60$                   ; Place to return to after send
                50 A5   DD  03BA  1317     PUSHL   CDRP$L_SAVEPC(R5)       ; Put our caller's PC back on the stack
                        03BD  1318     ASSUME  CLSMSG$K_MAXMSG,LT,256
             51   6B 8F   9A  03BD  1319     MOVZBL  #CLSMSG$K_MAXMSG,R1     ; Message size
                60 B4   17  03C1  1320     JMP     @PDT$L_SNDCNTMSG(R4)    ; This is a JMP to SEND_CNT_MSG_BUF rather
                        03C4  1321                                         ; than a JSB
                        03C4  1322
                        03C4  1323         ; The following two named routines are special message build routines
                        03C4  1324         ; to handle block transfer requests.  They replace the message build
                        03C4  1325         ; routine and all following code.
                        03C4  1326         ;
                        03C4  1327 SEND_DATA:
                54 A5   B4  03C4  1328     CLRW    CDRP$W_SENDSEQNM(R5)    ; No sequence number
             6E EF'AF   9E  03C7  1329     MOVAB   B^60$,(SP)              ; Replace message build routine return
                31'AF   9F  03CB  1330     PUSHAB  B^100$                  ; Return point when transfer complete
                54 B4   17  03CE  1331     JMP     @PDT$L_SENDDATA(R4)     ; This is a JMP to request data movement
                        03D1  1332
                        03D1  1333 REQUEST_DATA:
                54 A5   B4  03D1  1334     CLRW    CDRP$W_SENDSEQNM(R5)    ; No sequence number
             6E EF'AF   9E  03D4  1335     MOVAB   B^60$,(SP)              ; Replace message build routine return
```

```
                31'AF   9F   03D8  1336          PUSHAB  B^100$                  ; Return point when transfer complete
                50 B4   17   03DB  1337          JMP     @PDT$L_REQDATA(R4)      ; This is a JMP to request data movement
                             03DE  1338
                58 A5   D0   03DE  1339 50$:     MOVL    CDRP$L_RETRSPID(R5),-   ; Store return RSPID (or 0)
                04 A2        03E1  1340                  CLSMSG$L_RSPID(R2)
                             03E3  1341          ASSUME  CLSMSG$K_MAXMSG,LT,256
              51   6B 8F   9A   03E3  1342          MOVZBL  #CLSMSG$K_MAXMSG,R1     ; Message size
                             03E7  1343          SEND_CNT_MSG_BUF
                50 A5   D0   03EA  1344          MOVL    CDRP$L_SAVEPC(R5),-     ; Put our caller's return PC into
                0C A5        03ED  1345                  CDRP$L_FPC(R5)          ; CDRP fork block
                             03EF  1346
                             03EF  1347 60$:     ; Come here after message has been sent to see if we have to resume
                             03EF  1348          ; any CDRPs that were placed on the RESEND queue.  This represents
                             03EF  1349          ; the end of the critical section.  Also come here to initiate
                             03EF  1350          ; resending messages when a connection is re-opened.
                             03EF  1351
                             03EF  1352 RESEND_MSG:
            34 A3   1C A3    D0   03EF  1353          MOVL    CSB$L_RESENDQFL(R3), -  ; Update current CDRP
                             03F4  1354                  CSB$L_CURRCDRP(R3)
                   01    12   03F4  1355          BNEQ    70$                     ; Have a waiter
                         05   03F6  1356          RSB
                             03F7  1357
                             03F7  1358 70$:     ; Resume a waiting CDRP thread.
                             03F7  1359
                55   1C A3   D0   03F7  1360          MOVL    CSB$L_RESENDQFL(R3),R5  ; Get next waiting CDRP
                1C A3   65   D0   03FB  1361          MOVL    CDRP$L_FQFL(R5), -      ; Update list head pointer
                             03FF  1362                  CSB$L_RESENDQFL(R3)
                   05    12   03FF  1363          BNEQ    80$                     ; Branch if list not yet empty
                20 A3   1C A3    DE   0401  1364          MOVAL   CSB$L_RESENDQFL(R3), -  ; Update list tail pointer
                             0406  1365                  CSB$L_RESENDQBL(R3)
                             0406  1366 80$:
                0C A5   D0   0406  1367          MOVL    CDRP$L_FPC(R5),-        ; Use original caller's return address
                50 A5        0409  1368                  CDRP$L_SAVEPC(R5)       ; as the saved PC
              54   10 A3   D0   040B  1369          MOVL    CSB$L_PDT(R3),R4        ; Get PDT address
                0C A3   D0   040F  1370          MOVL    CSB$L_CDT(R3),-         ; Put CDT address into CDRP
                24 A5        0412  1371                  CDRP$L_CDT(R5)
                             0414  1372          DISPATCH  CDRP$B_CNXSTATE(R5),type=B,prefix=CDRP$K_ -
                             0414  1373                  <-
                             0414  1374                  <NORMAL,SEND_ALLOC>, -  ; Normal messages
                             0414  1375                  <REQUESTOR,90$>, -      ; Block transfer requestor messages
                             0414  1376                  <PARTNER,90$>, -        ; Block transfer partner messages
                             0414  1377                  >
                             041F  1378          BUG_CHECK       CNXMGRERR,FATAL ; Invalid CNX state
                             0423  1379
              51   40 A5    DE   0423  1380 90$:     MOVAL   CDRP$L_CNXSVAPTE(R5),R1 ; Get SVAPTE block address
              52   4A A5    9A   0427  1381          MOVZBL  CDRP$B_CNXRMOD(R5),R2   ; Get requestor's access mode
                             042B  1382          MAP                             ; Map transfer block
                 FF3E    31   042E  1383          BRW     SEND_ALLOC              ; Join normal message resend code
                             0431  1384
                             0431  1385 ; Get here when block transfer request completes
                             0431  1386 ;
                   50    DD   0431  1387 100$:    PUSHL   R0                      ; Save status
                             0433  1388          UNMAP                           ; Unmap buffer
                   01    BA   0436  1389          POPR    #^M<R0>                 ; Restore status
                50 B5    17   0438  1390          JMP     @CDRP$L_SAVEPC(R5)      ; Return to caller
                             043B  1391
                             043B  1392          .DSABL  LSB
```

```
                        .SBTTL  CNX$SEND_MNY_MSGS - Send acknowledged messages to all nodes
                ;++
                ; FUNCTIONAL DESCRIPTION:
                ;
                ;       This routine sends acknowledged messages to all nodes having valid CSB
                ;       addresses in the cluster system vector.  The messages are sent using
                ;       multiple concurrent fork threads executing the CNX$SEND_MSG
                ;       acknowledged message service.
                ;
                ;       NOTE:
                ;
                ;         o No attempt is made to detect pending changes in the cluster system
                ;           vector.
                ;
                ;         o Error status returns from the acknowledged message facility, i.e.
                ;           failing or failed systems, are ignored.
                ;
                ;       This routine will "broadcast" a message to all currently active
                ;       systems.  However, systems just entering or leaving the cluster may be
                ;       missed.  Callers of this routine are completely responsible for
                ;       handling "missed" systems.
                ;
                ; CALLING SEQUENCE:
                ;
                ;       BSBW    CNX$SEND_MNY_MSGS       Send many messages
                ;
                ;       This routine returns to its caller when the all messages have been
                ;       queued for processing  by the CNX$SEND_MSG.  This does not guarantee
                ;       that all messages have been received at remote nodes.  Because of the
                ;       nature of CNX$SEND_MSG operation when no response is required, waiting
                ;       for all messages to be received and acknowledged at the remote nodes
                ;       could result in wait intervals of days.
                ;
                ;       If a wait for resources is necessary, control may be returned to the
                ;       caller's caller before control is returned to the caller.
                ;
                ;       IPL must be at IPL$_SCS
                ;
                ; INPUT PARAMETERS:
                ;
                ;       00(SP)  Return address for caller
                ;       04(SP)  Return address for caller's caller
                ;       R5      Address of CDRP
                ;
                ; IMPLICIT INPUTS:
                ;
                ;       CDRP$L_MSGBLD must contain the address of a message build routine.
                ;
                ;       CDRP$B_FIPL must contain IPL$_SCS
                ;
                ;       Because return from this routine does not guarantee that the message
                ;       build routine will never be called again, any information required by
                ;       the message build routine should be contained completely in the CDRP
                ;       or in data structures which will never disappear.
                ;
                ;       This routine requires that several CDRP fields be initialized to zero.
                ;       CNX$INIT_CDRP should be called to perform this initialization.
```

```
                        043B  1451 ;
                        043B  1452 ;       CLU$GL_CLUSVEC  starting address of the cluster system vector
                        043B  1453 ;       CLU$GW_MAXINDEX maximum CSID index
                        043B  1454 ;
                        043B  1455 ; OUTPUT PARAMETERS:
                        043B  1456 ;
                        043B  1457 ;       R5      CDRP address (unchanged)
                        043B  1458 ;
                        043B  1459 ; IMPLICIT OUTPUTS:
                        043B  1460 ;
                        043B  1461 ;       None
                        043B  1462 ;
                        043B  1463 ; SIDE EFFECTS:
                        043B  1464 ;
                        043B  1465 ;       R0 - R4 are destroyed.
                        043B  1466 ;
                        043B  1467 ;--
                        043B  1468
                        043B  1469 CNX$SEND_MNY_MSGS::
                        043B  1470
           50 A5 8ED0   043B  1471          POPL    CDRP$L_SAVEPC(R5)               ; Save caller's return address.
54 00000000'GF    D0    043F  1472          MOVL    G^CLU$GL_CLUSVEC, R4            ; Get cluster system vec. base.
53 00000000'GF    3C    0446  1473          MOVZWL  G^CLU$GW_MAXINDEX, R3           ; Get max CSID index.
           09 53    F5  044D  1474 10$:     SOBGTR  R3, 30$                         ; Loop through entire cluster
                        0450  1475                                                  ; system vector except idx. 0.
           50 B5    17  0450  1476          JMP     @CDRP$L_SAVEPC(R5)              ; Return to caller.
                        0453  1477
                        0453  1478 ;
                        0453  1479 ; Error allocating memory for CDRP.
                        0453  1480 ;
                        0453  1481
                        0453  1482 20$:     FORK_WAIT                               ; Wait a little while.
                        0459  1483
                        0459  1484 ;
                        0459  1485 ; Send message to one node
                        0459  1486 ;
     50   6443    D0    0459  1487 30$:     MOVL    (R4)[R3], R0                    ; Get system vector entry.
             EE    18  045D  1488          BGEQ    10$                             ; Branch if not valid CSB addr.
E9 60 A0     18    E0  045F  1489          BBS     #CSB$V_LOCAL, -                 ; Branch if this is the local node.
                        0464  1490                  CSB$L_STATUS(R0),10$
                        0464  1491
     24 A5   50    D0  0464  1492          MOVL    R0, CDRP$L_CDT(R5)              ; Save CSB of entry.
                        0468  1493
     51   60 8F    9A  0468  1494          MOVZBL  #CDRP$K_CM_LENGTH, R1           ; Get size of needed CDRP.
  00000000'GF    16    046C  1495          JSB     G^EXE$ALONONPAGED               ; Allocate memory for the CDRP.
           DE 50    E9  0472  1496          BLBC    R0, 20$                         ; Branch if allocation failed.
                        0475  1497
                        0475  1498          ASSUME  CDRP$B_CD_TYPE EQ <CDRP$W_CDRPSIZE + 2>
     08 A2   51    B0  0475  1499          MOVW    R1, CDRP$W_CDRPSIZE(R2)         ; Set allocation size.
             3C    BB  0479  1500          PUSHR   #^M<R2,R3,R4,R5>                ; Save more registers.
        0056 8F    28  047B  1501          MOVC3   #<CDRP$K_CM_LENGTH-CDRP$B_CD_TYPE>,-    ; Copy rest of user's
0A A2      0A A5        047F  1502                  CDRP$B_CD_TYPE(R5), CDRP$B_CD_TYPE(R2)  ; CDRP to new CDRP.
                        0483  1503
           55 8ED0      0483  1504          POPL    R5                              ; Restore new CDRP address.
     53   24 A5    D0  0486  1505          MOVL    CDRP$L_CDT(R5), R3              ; Restore saved CSB address.
          99'AF    9F  048A  1506          PUSHAB  B^60$                           ; Set caller's caller address.
            FEBF   30  048D  1507          BSBW    CNX$SEND_MSG_CSB                ; Send this message.
```

```
                     0490  1508
                     0490  1509
                     0490  1510
                     0490  1511          MOVL    R5, R0                  ; Copy CDRP address.
00000000'GF   17     0493  1512          JMP     G^EXE$DEANONPAGED       ; Deallocate it and return.
                     0499  1513
                     0499  1514
                     0499  1515
                     0499  1516 60$:     POPR    #^M<R3,R4,R5>           ; Restore saved registers.
                     049B  1517          BRB     10$                     ; Go process next index.
                     049D  1518
```

```
                     049D  1520          .SBTTL  CNX$RCV_MSG - Receive message routine
                     049D  1521
                     049D  1522 ;++
                     049D  1523 ; FUNCTIONAL DESCRIPTION:
                     049D  1524 ;
                     049D  1525 ;       This routine is the message input routine.  I.e. SCS calls
                     049D  1526 ;       this routine when a message has been received over our
                     049D  1527 ;       connection.  This routine firsts looks at the acknowledge
                     049D  1528 ;       sequence number and calls any fork processes waiting for message
                     049D  1529 ;       acknowledgement.  It then determines if this message is a
                     049D  1530 ;       response for a message we sent.  If it is, that fork process
                     049D  1531 ;       is resumed.  Otherwise, this message must be an unsolicited
                     049D  1532 ;       message in which case the appropriate function routine is called.
                     049D  1533 ;
                     049D  1534 ; CALLING SEQUENCE:
                     049D  1535 ;
                     049D  1536 ;       JSB     CNX$RCV_MSG     (called from fork dispatcher)
                     049D  1537 ;       IPL is at IPL$_SCS
                     049D  1538 ;       This routine operates as a fork process.
                     049D  1539 ;
                     049D  1540 ; INPUT PARAMETERS:
                     049D  1541 ;
                     049D  1542 ;       R1      Length of message
                     049D  1543 ;       R2      Address of message
                     049D  1544 ;       R3      Address of CDT
                     049D  1545 ;       R4      Address of PDT
                     049D  1546 ;
                     049D  1547 ; OUTPUT PARAMETERS:
                     049D  1548 ;
                     049D  1549 ;       NONE
                     049D  1550 ;
                     049D  1551 ; SIDE EFFECTS:
                     049D  1552 ;
                     049D  1553 ;       NONE
                     049D  1554 ;
                     049D  1555 ;--
                     049D  1556
                     049D  1557          .ENABL  LSB
                     049D  1558
                     049D  1559 ; Connection is not open, drop message and return
                     049D  1560 ;
43 A3    07    91    049D  1561 10$:     CMPB    #CSB$K_DISCONNECT, -    ; Is connection disconnecting?
                     04A1  1562                  CSB$B_STATE(R3)
       03    12      04A1  1563          BNEQ    20$                     ; Branch if not disconnecting
     0506    31      04A3  1564          BRW     CNX$DEALL_MSG_BUF_CSB   ; Deallocate message buffer and return
                     04A6  1565
                     04A6  1566 20$:     BUG_CHECK       CNXMGRERR,FATAL ; Connection in unexptected state
                     04AA  1567
                     04AA  1568 ; Sequence number error in received message
                     04AA  1569 ;
  2E A3    A3        04AA  1570 30$:     SUBW3   CSB$W_RCVDSEQNM(R3),-   ; Verify that this is a missing
50   62              04AD  1571                  CLSMSG$W_SEQNUM(R2),R0  ;   message sequence number
       03    19      04AF  1572          BLSS    40$                     ; Branch if not a missing message
                     04B1  1573
                     04B1  1574 ;       BRW     CNX$RCV_REJECT          ; Reject message and return
     0000    31      04B1  1575          BRW     40$                     ; *** temp to catch problems -- bugcheck on
                     04B4  1576
```

```
                                04B4  1577 40$:    BUG_CHECK       CNXMGRERR,FATAL ; Repeated or garbage sequence number
                                04B8  1578
                                04B8  1579 ; Acknowledged message sequence number precedes previous number
                                04B8  1580 ;
                                04B8  1581 50$:    BUG_CHECK       CNXMGRERR,FATAL ; Out of order acknowledgement
                                04BC  1582
                                04BC  1583 CNX$RCV_MSG::
          53    5C A3    D0     04BC  1584         MOVL    CDT$L_AUXSTRUC(R3),R3   ; Get address of CSB
          43 A3    01    91     04C0  1585         CMPB    #CSB$K_OPEN,-           ; Is connection open?
                                04C4  1586                 CSB$B_STATE(R3)
                   D7    12     04C4  1587         BNEQ    10$                     ; Branch if not open
                                04C6  1588
                                04C6  1589         ; Verify the sequence number on this message is 1 greater
                                04C6  1590         ; than the last we received.  Update the received sequence number
                                04C6  1591         ; field.  Determine if the ack'ed sequence number is greater
                                04C6  1592         ; than the last sequence number ack'ed.
                                04C6  1593
             2E A3    B6     04C6  1594 110$:    INCW    CSB$W_RCVDSEQNM(R3)     ; Increment highest seq. no. received
                FB    13     04C9  1595         BEQL    110$                    ; Skip over zero
             2E A3    B1     04CB  1596         CMPW    CSB$W_RCVDSEQNM(R3),-   ; Verify message sequence number
                62           04CE  1597                 CLSMSG$W_SEQNUM(R2)
                D9    12     04CF  1598         BNEQ    30$                     ; Message seq. number error
             32 A3    96     04D1  1599         INCB    CSB$B_UNACKEDMSGS(R3)   ; Incr. count of un-acked messages
                53    DD     04D4  1600         PUSHL   R3                      ; Save CSB address
                                04D6  1601
       50 02 A2    30 A3    A3     04D6  1602         SUBW3   CSB$W_ACKRSEQNM(R3), -  ; Is ack'ed sequence number
                                04DC  1603                 CLSMSG$W_ACKSEQ(R2),R0  ; bigger than the last one?
                   3C    13     04DC  1604         BEQL    150$                    ; It's the same - nothing new ack'ed
                   D8    19     04DE  1605         BLSS    50$                     ; It's smaller - seq. no. error
          30 A3    02 A2    B0     04E0  1606         MOVW    CLSMSG$W_ACKSEQ(R2), -  ; It's bigger - update ack'ed number
                                04E5  1607                 CSB$W_ACKRSEQNM(R3)
                                04E5  1608
                                04E5  1609         ; We've received a new ack'ed sequence number.  Resume fork process
                                04E5  1610         ; threads for all CDRPs that have just been ack'ed.  This doesn't
                                04E5  1611         ; include CDRPs that have RSPIDs as they are resumed when the
                                04E5  1612         ; response message arrives.  However, CDRPs with RSPIDs are not on
                                04E5  1613         ; the sent queue
                                04E5  1614
             55    14 A3    D0     04E5  1615 130$:    MOVL    CSB$L_SENTQFL(R3),R5    ; Get first CDRP in sent list
                   2F    13     04E9  1616         BEQL    150$                    ; No more CDRP's -- continue
       50    54 A5    30 A3    A3     04EB  1617         SUBW3   CSB$W_ACKRSEQNM(R3), -  ; Does CDRP's sequence number match
                                04F1  1618                 CDRP$Q_SENDSEQNM(R5),R0 ; next ack'ed sequence number?
                   27    14     04F1  1619         BGTR    150$                    ; This message not ack'ed
             14 A3    65    D0     04F3  1620         MOVL    CDRP$L_FQFL(R5), -      ; Update list head pointer
                                04F7  1621                 CSB$L_SENTQFL(R3)
                   05    12     04F7  1622         BNEQ    140$                    ; Branch if list not empty
          18 A3    14 A3    DE     04F9  1623         MOVAL   CSB$L_SENTQFL(R3), -    ; Reset list tail pointer
                                04FE  1624                 CSB$L_SENTQBL(R3)
                54 A5    B4     04FE  1625 140$:    CLRW    CDRP$Q_SENDSEQNM(R5)    ; Clear sequence number marking message ackn
                22 A5    B5     0501  1626         TSTW    CDRP$L_RSPID+2(R5)      ; Is there a RSPID?
                   DF    12     0504  1627         BNEQ    130$                    ; Branch if yes
                                0506  1628
                   52    DD     0506  1629         PUSHL   R2                      ; Save message buffer address
                                0508  1630
                                0508  1631         ; Have a CDRP whose message has been ack'ed and who doesn't
                                0508  1632         ; have a response id.  Resume fork process.  Inputs to fork process are:
                                0508  1633         ;
```

```
                                                N 7
                     0508  1634          ;       R0      contains 1 (successful acknowledge)
                     0508  1635          ;       R3      Address of CSB
                     0508  1636          ;       R4      Address of PDT
                     0508  1637          ;       R5      Address of CDRP
                     0508  1638          ;
                     0508  1639          ; Fork routine may destroy R0 - R5.
                     0508  1640
          50   01    D0  0508  1641          MOVL    #SS$_NORMAL,R0          ; Indicate success
            0C B5     16  050B  1642          JSB     @CDRP$L_FPC(R5)         ; Resume fork process
               52 8ED0  050E  1643          POPL    R2                      ; Restore message buffer address
          53   6E    D0  0511  1644          MOVL    (SP),R3                 ; Restore CSB address into R3
       54   10 A3    D0  0514  1645          MOVL    CSB$L_PDT(R3),R4        ; Fetch PDT address
               CB    11  0518  1646          BRB     130$                    ; Continue loop
                     051A  1647
                     051A  1648 150$:    ; Now handle incoming message.  Determine if it is a response
                     051A  1649          ; to a message we sent or an unsolicited message by looking
                     051A  1650          ; at the message fucntion code.  Responses have negative function codes.
                     051A  1651
          05A2'CF    9F  051A  1652          PUSHAB  W^200$                  ; All roads eventually return to 200$
                     051E  1653
       50   08 A2    98  051E  1654          CVTBL   CLSMSG$B_FACILITY(R2),R0 ; Get facility code
               3E    18  0522  1655          BGEQ    170$                    ; Branch if not a response
                     0524  1656
                     0524  1657          ; Look up the RSPID to find the corresponding CDRP.
                     0524  1658          ; Recycle the RSPID with inline code instead of calling SCS (for speed).
                     0524  1659
    50   00000000'GF    D0  0524  1660          MOVL    G^SCS$GL_RDT,R0         ; Get address of table of RSPIDs
      51   04 A2    3C  052B  1661          MOVZWL  CLSMSG$L_RSPID(R2),R1   ; Get sequence number of RSPID
      F8 A0   51    D1  052F  1662          CMPL    R1,RDT$L_MAXRDIDX(R0)   ; Check it against maximum
               29    1A  0533  1663          BGTRU   165$                    ; Too big - bugcheck
                     0535  1664          ASSUME  RD$C_LENGTH  EQ  8
         51    6041    7E  0535  1665          MOVAQ   (R0)[R1],R1             ; Compute address of entry
            06 A1    B1  0539  1666          CMPW    RD$W_SEQNUM(R1),-       ; Compare sequence numbers
            06 A2        053C  1667                  CLSMSG$L_RSPID+2(R2)
               1E    12  053E  1668          BNEQ    165$                    ; No match, bugcheck
                     0540  1669          ASSUME  RD$V_BUSY  EQ  0
         1A 04 A1    E9  0540  1670          BLBC    RD$W_STATE(R1),165$
          55   61    D0  0544  1671          MOVL    RD$L_CDRP(R1),R5        ; Get CDRP address
    20 A5   04 A2    D1  0547  1672          CMPL    CLSMSG$L_RSPID(R2), -   ; Check for RSPID match.
                     054C  1673                  CDRP$L_RSPID(R5)
               10    12  054C  1674          BNEQ    165$                    ; Branch if no match.
            06 A1    B6  054E  1675 160$:    INCW    RD$W_SEQNUM(R1)         ; Increment sequence number
               FB    13  0551  1676          BEQL    160$                    ; Skip over zero
            06 A1    B0  0553  1677          MOVW    RD$W_SEQNUM(R1),-       ; Copy new sequence number into CDRP
            22 A5        0556  1678                  CDRP$L_RSPID+2(R5)
                     0558  1679
                     0558  1680          ; We have a response to a previous message.  Resume fork process.
                     0558  1681          ; Inputs to fork process are:
                     0558  1682          ;
                     0558  1683          ;       R0      SS$_NORMAL (successful acknowledge)
                     0558  1684          ;       R2      Address of message
                     0558  1685          ;       R3      CSB
                     0558  1686          ;       R4      Address of PDT
                     0558  1687          ;       R5      Address of CDRP
                     0558  1688          ;
                     0558  1689          ; Fork routine may destroy R0 - R5.
                     0558  1690
```

```
                                                B 8
                50    01    D0  0558  1691          MOVL    #SS$_NORMAL,R0          ; Indicate success
                  0C  B5    17  055B  1692          JMP     @CDRP$L_FPC(R5)         ; Continue thread -- return to 200$
                              055E  1693
                              055E  1694 165$:      BUG_CHECK       CNXMGRERR,FATAL         ; Response id invalid
                              0562  1695
                              0562  1696 170$:
                              0562  1697          ; Message is an input message rather than a response.  Dispatch
                              0562  1698          ; to appropriate second level message dispatcher.
                              0562  1699          ;
                              0562  1700          ;       Inputs to second level dispatcher are:
                              0562  1701          ;
                              0562  1702          ;       R2      Address of message
                              0562  1703          ;       R3      CSB
                              0562  1704          ;       R4      Address of PDT
                              0562  1705          ;       R5      If CLSMSG$L_RSPID(R2) is non-zero the address of a
                              0562  1706          ;               non-initialized non-paged pool packet, usually a CDRP,
                              0562  1707          ;               (the size is determined on a per-facility basis from
                              0562  1708          ;               the table, FAC_SIZES, below)
                              0562  1709          ;
                              0562  1710          ;       Routine may destroy R0 - R5
                              0562  1711          ;
                              0562  1712          ;       N.B. the pool allocation does not check the legality of the
                              0562  1713          ;       facility code.  It prevents errors during the pool allocation
                              0562  1714          ;       request.  If the facility is bad, however, the first level
                              0562  1715          ;       dispatcher will bugcheck the system very soon.
                              0562  1716
                    04  A2    D5  0562  1717          TSTL    CLSMSG$L_RSPID(R2)      ; Is a pool packet needed?
                      1C    13  0565  1718          BEQL    180$                    ; Branch if no pool needed
                51  9A'AF40   9A  0567  1719          MOVZBL  B^FAC_SIZES[R0],R1      ; Get size of pool to allocate
                      15    13  056C  1720          BEQL    180$                    ; Branch if allocation size is zero
                      52    DD  056E  1721          PUSHL   R2                      ; Save message buffer address
          00000000'GF       16  0570  1722          JSB     G^EXE$ALONONPAGED       ; Allocate needed pool
                55    52    D0  0576  1723          MOVL    R2,R5                   ; Save packet address
                      52  8ED0  0579  1724          POPL    R2                      ; Restore message buffer address
                  56  50    E9  057C  1725          BLBC    R0,CNX$RCV_REJECT       ; Branch on failure and reject message
                              057F  1726                                          ;   return to 200$
                08  A5    51    B0  057F  1727          MOVW    R1,CDRP$W_CDRPSIZE(R5)  ; Setup packet size
                              0583  1728
                              0583  1729 180$:      ; Return to 200$
                              0583  1730          DISPATCH        CLSMSG$B_FACILITY(R2),TYPE=B,PREFIX=CLSMSG$K_FAC_,-
                              0583  1731                  <-
                              0583  1732                  <ACK,ACK_MSG>,-                 ; Explicit ACK message
                              0583  1733                  <CJF,CJF$DISPATCH>,-            ; Common journaling facility
                              0583  1734                  <CNX,CNX$DISPATCH>,-            ; Connection manager facility
                              0583  1735                  <CSP,CSP$DISPATCH>,-            ; Cluster Server Process
                              0583  1736                  <LCK,LCK$DISPATCH>,-            ; Lock manager facility
                              0583  1737                  <LKI,LKI$DISPATCH>,-            ; GETLKI facility
                              0583  1738                  <BLK,BLKXFR_RETRY>,-            ; Block transfer
                              0583  1739                  >
                              0596  1740          BUG_CHECK       CNXMGRERR,FATAL         ; Unrecognized function code
                              059A  1741
                              059A  1742 ;
                              059A  1743 ; Table of pool packet sizes
                              059A  1744 ; for automatic allocations on incoming new messages
                              059A  1745 ; with response requested
                              059A  1746 ;
                              059A  1747
```

```
                     059A  1748 FAC_SIZES:
                     059A  1749         FAC_POOL        <-
                     059A  1750                         <ACK,0>,-
                     059A  1751                         <CNX,CDRP$K_CM_LENGTH>,-
                     059A  1752                         <LCK,CDRP$K_CM_LENGTH>,-
                     059A  1753                         <CJF,IRP$K_LENGTH>,-
                     059A  1754                         <LKI,CDRP$K_CM_LENGTH>,-
                     059A  1755                         <CSP,CDRP$K_CM_LENGTH+8>,-
                     059A  1756                         <BLK,0>, -
                     059A  1757                         >
          00000008   05A2  1758 MAX_FACILITY = . - FAC_SIZES
                     05A2  1759
                     05A2  1760 200$:   ; Come here after handling input message is complete.
                     05A2  1761         ; Determine if an explicit ACK message should be sent back
                     05A2  1762
     53    8E    D0  05A2  1763         MOVL    (SP)+,R3                ; Restore CSB address
       32  A3    91  05A5  1764         CMPB    CSB$B_UNACKEDMSGS(R3),- ; Is it necessary to send an ACK?
       33  A3        05A8  1765                 CSB$B_REMACKLIM(R3)
          01    18  05AA  1766         BGEQ    SEND_ACK_MSG            ; Send explicit acknowledgement
                05  05AC  1767         RSB
                     05AD  1768
                     05AD  1769         .DSABL  LSB
```

```
                        05AD  1771          .SBTTL  SEND_ACK_MSG - Send an explicit ACK message
                        05AD  1772
                        05AD  1773 ;++
                        05AD  1774 ; FUNCTIONAL DESCRIPTION:
                        05AD  1775 ;
                        05AD  1776 ;       This routine sends an explicit ACK message back to the
                        05AD  1777 ;       remote side.
                        05AD  1778 ;
                        05AD  1779 ; CALLING SEQUENCE:
                        05AD  1780 ;
                        05AD  1781 ;       BSBW    SEND_ACK_MSG
                        05AD  1782 ;       IPL must be at IPL$_SCS
                        05AD  1783 ;
                        05AD  1784 ;       This routine may return to the caller before the message
                        05AD  1785 ;       has been sent (if we go into a SCS wait state).
                        05AD  1786 ;
                        05AD  1787 ; INPUT PARAMETERS:
                        05AD  1788 ;
                        05AD  1789 ;       R3      Address of CSB
                        05AD  1790 ;
                        05AD  1791 ; OUTPUT PARAMETERS:
                        05AD  1792 ;
                        05AD  1793 ;       None
                        05AD  1794 ;--
                        05AD  1795
                        05AD  1796 SEND_ACK_MSG:
              34 A3   D5 05AD  1797          TSTL    CSB$L_CURRCDRP(R3)      ; Test whether critical section blocked
                 17   12 05B0  1798          BNEQ    10$                     ; Branch if it is blocked and return
               0387   30 05B2  1799          BSBW    CNX$ALLOC_CDRP_ONLY     ; Allocate a CDRP
              14 50   E9 05B5  1800          BLBC    R0,20$                  ; If unable to allocate, just return
                        05B8  1801                                          ; without sending the message
        4C A5  CD'AF   9E 05B8  1802          MOVAB   B^50$,CDRP$L_MSGBLD(R5) ; Address of message build routine
                 FD8F   30 05BD  1803          BSBW    CNX$SEND_MSG_CSB        ; Send the message
                 50  55 D0 05C0  1804          MOVL    R5,R0                   ; Address of CDRP
          00000000'GF   17 05C3  1805          JMP     G^EXE$DEANONPAGED       ; Deallocate CDRP
                        05C9  1806
              32 A3   94 05C9  1807 10$:      CLRB    CSB$B_UNACKEDMSGS(R3)   ; Prevent further ACK attempts
                      05 05CC  1808 20$:      RSB
                        05CD  1809
           08 A2   04   90 05CD  1810 50$:      MOVB    #CLSMSG$K_FAC_ACK, -    ; Store message facility code
                        05D1  1811                      CLSMSG$B_FACILITY(R2)   ; (N.B. no sub-function code.)
                      05 05D1  1812          RSB
                        05D2  1813
                        05D2  1814 ; Come here upon receiving one of these messages
                        05D2  1815 ;
                        05D2  1816 ACK_MSG:
                 03D7   31 05D2  1817          BRW     CNX$DEALL_MSG_BUF_CSB   ; Deallocate input message buffer.
```

```
                     05D5  1819          .SBTTL  CNX$RCV_REJECT - Reject received message
                     05D5  1820
                     05D5  1821 ;++
                     05D5  1822 ; FUNCTIONAL DESCRIPTION:
                     05D5  1823 ;
                     05D5  1824 ;       This routine rejects a received message, i.e., pretends that
                     05D5  1825 ;       this message was never seen.  This is done by dropping the
                     05D5  1826 ;       message on the floor, breaking the connection, and undoing
                     05D5  1827 ;       the sequence number modifcation that has taken place.
                     05D5  1828 ;
                     05D5  1829 ;       This routine may be called ONLY if the following conditions hold:
                     05D5  1830 ;               a) Unbroken thread of execution contiguous with receipt
                     05D5  1831 ;                  of message.
                     05D5  1832 ;               b) No messages have been sent since this message was
                     05D5  1833 ;                  received.
                     05D5  1834 ;
                     05D5  1835 ; CALLING SEQUENCE:
                     05D5  1836 ;
                     05D5  1837 ;       BSBW    CNX$RCV_REJECT
                     05D5  1838 ;       IPL must be at IPL$_SCS
                     05D5  1839 ;
                     05D5  1840 ; INPUT PARAMETERS:
                     05D5  1841 ;
                     05D5  1842 ;       R2      Address of received message
                     05D5  1843 ;       R3      Address of CSB
                     05D5  1844 ;
                     05D5  1845 ; OUTPUT PARAMETERS:
                     05D5  1846 ;
                     05D5  1847 ;       None
                     05D5  1848 ;
                     05D5  1849 ; SIDE EFFECTS:
                     05D5  1850 ;
                     05D5  1851 ;       R0-R2 are destroyed.
                     05D5  1852 ;
                     05D5  1853 ;--
                     05D5  1854
                     05D5  1855 CNX$RCV_REJECT::
          38   BB    05D5  1856          PUSHR   #^M<R3,R4,R5>                   ; Save registers
        03D2   30    05D7  1857          BSBW    CNX$DEALL_MSG_BUF_CSB           ; Deallocate message buffer
      2E A3    B7    05DA  1858 10$:     DECW    CSB$W_RCVDSEQNM(R3)             ; Fix remembered received sequence
          FB   13    05DD  1859          BEQL    10$                             ;  number
          CC   10    05DF  1860          BSBB    SEND_ACK_MSG                    ; Acknowledge all ack'ed messages
    55    53   D0    05E1  1861          MOVL    R3,R5                           ; Address of CSB
       FA19'   30    05E4  1862          BSBW    CNX$DISC_PROTOCOL               ; Request disconnect
          38   BA    05E7  1863          POPR    #^M<R3,R4,R5>                   ; Restore registers
               05    05E9  1864          RSB
```

```
05EA  1866          .SBTTL  Principles of connection manager block transfers
05EA  1867 ;++
05EA  1868 ;
05EA  1869 ; The following paragraphs describe how block transfers are performed by the
05EA  1870 ; connection manager.
05EA  1871 ;
05EA  1872 ;       Connection manager block transfers require a cooperative effort on the
05EA  1873 ;       part of two cluster members.  This is very similar to (and based upon)
05EA  1874 ;       the mechanisms by which SCS block transfers are accomplished.
05EA  1875 ;
05EA  1876 ;       A block transfer sequence is initiated by one node (which will be
05EA  1877 ;       refered to as the requestor for the duration of this discussion)
05EA  1878 ;       sending a message to a second node (which will be called the partner).
05EA  1879 ;       This message signals that a block transfer operation is needed and
05EA  1880 ;       describes the requestor's resources associated with the requested
05EA  1881 ;       block transfer.  The message must require a response from the partner
05EA  1882 ;       node.  When this response is received, it is assumed that the block
05EA  1883 ;       transfer has been completed.
05EA  1884 ;
05EA  1885 ;       Before sending its message the requestor node must lock the virtual
05EA  1886 ;       address space associated with the block transfer buffer into physical
05EA  1887 ;       memory and request SCS mapping resources to map the buffer.  The
05EA  1888 ;       connection manager will allocate SCS mapping resources to map the
05EA  1889 ;       buffer.  However, the connection manager will not lock the virtual
05EA  1890 ;       address space into physical memory nor will it fully protect its
05EA  1891 ;       clients from knowing whether they are the requestor of or a partner
05EA  1892 ;       to a block transfer operation.
05EA  1893 ;
05EA  1894 ;       Upon receipt of a message requesting that a block transfer take place,
05EA  1895 ;       the partner node must:
05EA  1896 ;
05EA  1897 ;           1.  Make whatever preparations are necessary to perform the block
05EA  1898 ;               transfer (for example, reading information from a file).
05EA  1899 ;
05EA  1900 ;           2.  Lock into physical memory those pages which contain (or will
05EA  1901 ;               receive) its end of the block transfer information.
05EA  1902 ;
05EA  1903 ;           3.  Using information in the message received from the requestor
05EA  1904 ;               as well as information about its own mapping resources the
05EA  1905 ;               block transfer must be performed.  This may either be done in
05EA  1906 ;               a single operation or segmented.
05EA  1907 ;
05EA  1908 ;           4.  If further processing is required once the transfer is
05EA  1909 ;               complete (for example, writing information to a file), it
05EA  1910 ;               must be done.
05EA  1911 ;
05EA  1912 ;           5.  The response message must be sent to the requestor node.  This
05EA  1913 ;               should be the last act of the thread initiated by the incoming
05EA  1914 ;               request for a block transfer operation.
05EA  1915 ;
05EA  1916 ;       As with the requestor node, the connection manager will provide some,
05EA  1917 ;       but by no means all, the support required for the tasks listed above.
05EA  1918 ;
05EA  1919 ;
05EA  1920 ; The following paragraphs describe the connection manager routines
05EA  1921 ; assoicated with block transfers.  The order of presentation follows an
05EA  1922 ; block transfer operation as it progresses from requestor to partner
```

```
05EA  1923 ;  and finally back to the requestor.
05EA  1924 ;
05EA  1925 ;  CNX$BLOCK_XFER, or CNX$BLOCK_XFER_IRP
05EA  1926 ;
05EA  1927 ;       One of these routines is called by a fork process on the requestor to
05EA  1928 ;       begin the block transfer sequence.  Map resources are allocated for
05EA  1929 ;       the requestor's buffer, a message buffer and RSPID are allocated, the
05EA  1930 ;       client's message build routine is called, and a message is sent to the
05EA  1931 ;       parter node.  When the response message is received, control is
05EA  1932 ;       returned to the location following the subroutine call.
05EA  1933 ;
05EA  1934 ;  CNX$PARTNER_INIT_CSB
05EA  1935 ;
05EA  1936 ;       This routine is called by the partner's received message routine once
05EA  1937 ;       the need for a block transfer is recognized.  It must be called before
05EA  1938 ;       the thread initiated by the incoming message forks.  A data structure
05EA  1939 ;       to describe the partner's block transfer (including a copy of the
05EA  1940 ;       incoming message buffer and a buffer area whose size is specified as
05EA  1941 ;       parameter to this routine) is allocated and initialized.  The incoming
05EA  1942 ;       message buffer is deallocated.  Once control is returned from this
05EA  1943 ;       routine, the thread initiated by the incoming message may fork.
05EA  1944 ;       If data structures cannot be allocaed, no return to the caller will
05EA  1945 ;       be made.  The thread will be cleaned up and dropped, the connection
05EA  1946 ;       will be broken.
05EA  1947 ;
05EA  1948 ;       Many of the operations one might want to do in order to satisfy the
05EA  1949 ;       block transfer request (e.g. reading data from a local disk) will
05EA  1950 ;       require a fork at this point.  The purpose of CNX$PARTNER_INIT_CSB is
05EA  1951 ;       to save all necessary context and release all necessary resources so
05EA  1952 ;       that a fork can occur.
05EA  1953 ;
05EA  1954 ;  CNX$BLOCK_READ, CNX$BLOCK_WRITE, CNX$BLOCK_READ_IRP, and CNX$BLOCK_WRITE_IRP
05EA  1955 ;
05EA  1956 ;       One or more of these routines are called to actually cause a block
05EA  1957 ;       transfer to occur.  N.B. read and write are viewed from the
05EA  1958 ;       perspective of the partner node; read means transfer from requestor to
05EA  1959 ;       partner and write means transfer from partner to requestor.
05EA  1960 ;
05EA  1961 ;       Mapping resources for the partner's buffer are allocated and the block
05EA  1962 ;       transfer operation is performed.  This may transfer all or part of the
05EA  1963 ;       requestor's buffer to/from the partner.  The partner need only provide
05EA  1964 ;       sufficient buffer space for that portion of requestor's buffer which
05EA  1965 ;       is to be transfered.  There is no prohibition against both reading
05EA  1966 ;       from and writing to the requestor's buffer (i.e. a modify operation, as
05EA  1967 ;       viewed from the requestor node).  However, at this time, there is no
05EA  1968 ;       protocol provided for preventing a set of operation from being restarted
05EA  1969 ;       from the beginning if a connection breaks and is reestablished.
05EA  1970 ;
05EA  1971 ;  CNX$PARTNER_FINISH
05EA  1972 ;
05EA  1973 ;       Control is transfered to this routine when the partner's portion of
05EA  1974 ;       the block transfer operation has been completed.  A response message
05EA  1975 ;       is sent to the requestor node and the structure allocated by
05EA  1976 ;       CNX$PARTNER_INIT_CSB is deallocated.
05EA  1977 ;
05EA  1978 ;
05EA  1979 ;  Now, a few words about recovery from a connection breakage.
```

```
05EA  1980 ;
05EA  1981 ;       When the connection between a requestor and a partner is broken the
05EA  1982 ;       partner thread is terminated with a call to the partner's error
05EA  1983 ;       routine after a message is sent to the requestor asking that the
05EA  1984 ;       request be retried.  If the requestor has survived, it will repeat
05EA  1985 ;       the request.
05EA  1986 ;
05EA  1987 ;       This form of broken connection recovery is required to accomodate the
05EA  1988 ;       use of SCS mapping resources.  The message requesting a block transfer
05EA  1989 ;       operation (sent from the requestor to the partner) contains a
05EA  1990 ;       description of the requestor's SCS mapping resources allocated to the
05EA  1991 ;       requestor's block transfer buffer.  In the event of a connection
05EA  1992 ;       breakage, these SCS mapping resources must be deallocated.  This
05EA  1993 ;       invalidates the description stored at the partner node and therefore
05EA  1994 ;       the entire operation thread on the partner node.
05EA  1995 ;
05EA  1996 ;       The term "graceful" in the two paragraphs above is intended to imply
05EA  1997 ;       that termination of the partner node thread includes a call to a
05EA  1998 ;       client-specified error routine thus giving the client an opportunity
05EA  1999 ;       to perform whatever client-specific cleanup operations are deemed
05EA  2000 ;       necessary.
05EA  2001 ;
05EA  2002 ;--
```

```
05EA  2004          .SBTTL  CNX$BLOCK_XFER - Initiate a block transfer request
05EA  2005          .SBTTL  CNX$BLOCK_XFER_IRP - Initiate a block transfer request w/ IRP
05EA  2006
05EA  2007 ;++
05EA  2008 ;
05EA  2009 ; FUNCTIONAL DESCRPITION:
05EA  2010 ;
05EA  2011 ;       This routines begin a block transfer operation sequence.  NOTE: a
05EA  2012 ;       block transfer operation is actually a sequence of operations
05EA  2013 ;       performed by cooperating processors/processes.  These routines
05EA  2014 ;       represent the beginning of that sequence.  By no means, do they
05EA  2015 ;       perform all operations involved in that sequence.  Nothing in these
05EA  2016 ;       routines directly controls the direction of the block transfer.  It is
05EA  2017 ;       determined solely by the cooperating acknowledged message services
05EA  2018 ;       clients.
05EA  2019 ;
05EA  2020 ;       Calling one of these routines results in a message being sent to the
05EA  2021 ;       cluster member identified by the input CSID.  In addition to the usual
05EA  2022 ;       goodies (both acknowledged message goodies and client goodies), the
05EA  2023 ;       message contains a buffer handle for the block transfer buffer on
05EA  2024 ;       this, the local, system.  This node is the requestor of the block
05EA  2025 ;       transfer operation.  The remote node is its partner.
05EA  2026 ;
05EA  2027 ;       The messages sent by these routines ALWAYS use a RSPID.  The block
05EA  2028 ;       transfer operation sequence is not complete until the partner node
05EA  2029 ;       responds to the intial message sent by these routines.  If the
05EA  2030 ;       connection between the requestor and partner nodes breaks between the
05EA  2031 ;       time when the partner receives the request and when it sends its
05EA  2032 ;       response, the partner send a retry request message to the requestor
05EA  2033 ;       and forgets about the request.  The block
05EA  2034 ;       transfer resource allocation mechanisms require this method of
05EA  2035 ;       operation.
05EA  2036 ;
05EA  2037 ;       As with the other acknowledged message serivces, these routines
05EA  2038 ;       control allocation of all SCS resources.  Because these routines must
05EA  2039 ;       allocate the SCS mapping resources to be used for the local buffer
05EA  2040 ;       handle, they require specific use of CDRP$L_VAL1, CDRP$L_VAL6,
05EA  2041 ;       CDRP$L_VAL7, and CDRP$L_VAL8 which would otherwise be available to
05EA  2042 ;       a client routine.
05EA  2043 ;
05EA  2044 ;       Except as noted above, these routines operate just like CNX$SEND_MSG.
05EA  2045 ;
05EA  2046 ; CALLING SEQUENCE:
05EA  2047 ;
05EA  2048 ;       BSBW    CNX$BLOCK_XFER          Initiate a block transfer
05EA  2049 ;       BSBW    CNX$BLOCK_XFER_IRP      Initiate a block transfer with an IRP
05EA  2050 ;
05EA  2051 ;       This routine returns to its caller when the block transfer has been
05EA  2052 ;       completed and the partner has responded to the initial requestor
05EA  2053 ;       message.  It returns to its caller's caller immediately.  The standard
05EA  2054 ;       fork process convention that the caller must not push anything onto
05EA  2055 ;       the stack is in effect.  The single exception is when R0 contains
05EA  2056 ;       SS$_NOSUCHNODE return status.  This is the only synchronous return
05EA  2057 ;       possible.  In this case, the return address of the caller's orginal
05EA  2058 ;       caller is still on the top of the stack.  In some cases, this may
05EA  2059 ;       require special action on the part of this routine's caller.
05EA  2060 ;
```

```
;       IPL must be at IPL$_SCS
;
; INPUT PARAMETERS:
;
;       R3      CSID
;       R5      Address of CDRP
;
; IMPLICIT INPUTS:
;
;       CDRP$L_MSGBLD must contain the address of a message build routine.
;
;       CDRP$L_RSPID must contain valid RSPID or its high order word must be
;       zero and its low order word nonzero to indicate that a RSPID must be
;       allocated.
;
;       CDRP$L_MSG_BUF must contain a valid message buffer address or zero.
;
;       --- FOR CNX$BLOCK_XFER:
;
;       CDRP$L_CNXSVAPTE(R5)    System virtual address of the first PTE
;                               describing the block transfer buffer
;       CDRP$W_CNXBOFF(R5)      Byte offset of first byte in block transfer
;                               buffer
;       CDRP$L_CNXBCNT(R5)      Number of bytes in block transfer
;       CDRP$B_CNXRMOD(R5)      Access mode of requestor
;
;       --- FOR CNX$BLOCK_XFER_IRP:
;
;       CDRP$L_SVAPTE(R5)       System virtual address of the first PTE
;                               describing the block transfer buffer
;       CDRP$W_BOFF(R5)         Byte offset of first byte in block transfer
;                               buffer
;       CDRP$L_BCNT(R5)         Number of bytes in block transfer
;       CDRP$B_RMOD(R5)         Access mode of requestor
;
;       Any information that the message build routine requires should
;       be in the CDRP or pointed to by pointers in the CDRP.
;
;       This routine requires that several CDRP fields be initialized to zero.
;       CNX$INIT_CDRP should be called to perform this initialization.
;
; OUTPUT PARAMETERS:
;
;       R0      Status
;               SS$_NORMAL ==> Message successfully acknowledged
;                              (if response requested, response received)
;               SS$_NOSUCHNODE ==> Invalid CSID
;                              (N.B. no fork occurs in this case)
;               SS$_NODELEAVE ==> Requested node is leaving the cluster
;                              or you are
;       R2      Partner's response message buffer address
;       R3      CSB address
;       R4      PDT address
;       R5      CDRP address
;
; IMPLICIT OUTPUTS:
;
```

```
                     ;       CDRP$L_VAL1(R5) and CDRP$L_VAL6(R5) through CDRP$L_VAL8 are destroyed
                     ;       by this routine or overlayed by implicit inputs to this routine.
                     ;
                     ;       Assuming proper cooperation on the partner node, the block transfer
                     ;       buffer has either been copied to the partner node or over written with
                     ;       information from the partner node.
                     ;
                     ; SIDE EFFECTS:
                     ;
                     ;       R0 - R2 and R4 are destroyed.
                     ;
                     ; WARNING:
                     ;
                     ;       The connection manager header in messages sent by this routine is
                     ;       three longwords longer than normal.  This space contains the local
                     ;       buffer handle information.  This tactic has been chosen so that only
                     ;       block transfer messages pay the three longword penalty because three
                     ;       longwords is a significant amount of the space available in the
                     ;       message buffer to a connection manager client.
                     ;
                     ;--

                             ASSUME  CDRP$B_RMOD-CDRP$L_IOQFL        EQ      IRP$B_RMOD
                             ASSUME  CDRP$L_SVAPTE-CDRP$L_IOQFL      EQ      IRP$L_SVAPTE
                             ASSUME  CDRP$W_BOFF-CDRP$L_IOQFL        EQ      IRP$W_BOFF
                             ASSUME  CDRP$L_BCNT-CDRP$L_IOQFL        EQ      IRP$L_BCNT
                             ASSUME  <CDRP$W_CNXBOFF - CDRP$L_CNXSVAPTE> EQ -
                                                                     <CDRP$W_BOFF - CDRP$L_SVAPTE>
                             ASSUME  <CDRP$L_CNXBCNT - CDRP$L_CNXSVAPTE> EQ -
                                                                     <CDRP$L_BCNT - CDRP$L_SVAPTE>

                                     .ENABLE LSB

                     ;
                     ; Wait for pool, for connection to be re-established, or for the target to be
                     ; removed from the cluster.
                     ;
 FB99   30           190$:   BSBW    CLEANUP_CDRP            ; Deallocate RSPID and/or message buffer
53 4C A3    D0       190$:   MOVL    CSB$L_CSID(R3),R3       ; Get CSID
                             FORK_WAIT                       ; On allocation failure; fork, wait,
      19    11               BRB     MEMORY_RETRY            ; and try again.

50 A5       DD       900$:   PUSHL   CDRP$L_SAVEPC(R5)       ; Setup return address
 FCDA       31               BRW     SEND_CSID_ERROR

                     CNX$BLOCK_XFER_IRP::

40 A5  CC A5  7D     CNX$BLOCK_XFER_IRP::
                             MOVQ    CDRP$L_SVAPTE(R5), -    ; Copy SVAPTE and BOFF.
                                     CDRP$L_CNXSVAPTE(R5)
46 A5  D2 A5  D0             MOVL    CDRP$L_BCNT(R5), -      ; Copy BCNT.
                                     CDRP$L_CNXBCNT(R5)
4A A5  AB A5  90             MOVB    CDRP$B_RMOD(R5), -      ; Copy RMOD.
                                     CDRP$B_CNXRMOD(R5)

                     CNX$BLOCK_XFER::

50 A5 8ED0           CNX$BLOCK_XFER::
                             POPL    CDRP$L_SAVEPC(R5)       ; Save return PC.
```

```
                     0612  2175
                     0612  2176 MEMORY_RETRY:
                     0612  2177
                     0612  2178          CSID_TO_CSB csb=R3, error=900$  ; Get CSB for input CSID.
                     062B  2179
                     062B  2180          ; allocate and init BTX
                     062B  2181
          51    30   9A  062B  2182          MOVZBL  #CLUBTX$K_LENGTH, R1    ; Get size of a BTX.
 00000000'GF    16   9F  062E  2183          JSB     G^EXE$ALONONPAGED       ; Attempt to allocate a BTX.
       B3    50   E9  0634  2184          BLBC    R0, 190$                ; Branch on allocation failure.
    08 A2    51   B0  0637  2185          MOVW    R1, CLUBTX$W_SIZE(R2)   ; Set allocation size.
0A A2  0465  8F   B0  063B  2186          MOVW    #<DYN$C_CLU_BTX*^x100+ -; Set structure type and subtype
                     0641  2187                  DYN$C_CLU>, -           ; fields.
                     0641  2188                  CLUBTX$B_TYPE(R2)
    18 A2    55   D0  0641  2189          MOVL    R5,CLUBTX$L_CDRP(R2)    ; Link CDRP to BTX
                     0645  2190          ASSUME  CLUBTX$S_LBUFHNDL EQ 12
       0C A2    7C  0645  2191          CLRQ    CLUBTX$L_LBUFHNDL(R2)   ; Zero local buffer handle area.
       14 A2    D4  0648  2192          CLRL    CLUBTX$L_LBUFHNDL+8(R2)
 2C A5 0C A2    DE  064B  2193          MOVAL   CLUBTX$L_LBUFHNDL(R2), -; Set CDRP local buffer handle
                     0650  2194                  CDRP$L_LBUFH_AD(R5)     ; pointer to point to BTX area.
 28 A2 50 A5    D0  0650  2195          MOVL    CDRP$L_SAVEPC(R5), -    ; Move caller's return PC to BTX
                     0655  2196                  CLUBTX$L_SAVED_PC(R2)
 2C A2 4C A5    D0  0655  2197          MOVL    CDRP$L_MSGBLD(R5), -    ; Copy user's message build routine address
                     065A  2198                  CLUBTX$L_MSGBLD(R2)
 4C A5 C6'AF    9E  065A  2199          MOVAB   B^BLD_BLKXFR_HDR, -     ; Insert message prebuild routine address
                     065F  2200                  CDRP$L_MSGBLD(R5)
                     065F  2201
                     065F  2202 BLOCK_XFER:
                     065F  2203
                     065F  2204          ; Allocate a buffer handle.  If the allocation waits, there is a BTX on
                     065F  2205          ; the partner queue in the state REQMAP.  If the connection breaks, this
                     065F  2206          ; CDRP must be taken of the waiting queue.  When the connection is restored,
                     065F  2207          ; execution should be continued at BLOCK_XFR so that a new attempt to alloca
                     065F  2208          ; a buffer handle will occur.
                     065F  2209
                     065F  2210          TEST_CSB_OPEN no=10$            ; Is the CSB open?
                     0665  2211
    56 A5    04   90  0665  2212          MOVB    #CDRP$K_REQ_MAP, -      ; Mark CDRP as belonging to a
                     0669  2213                  CDRP$B_CNXSTATE(R5)     ; requestor in need of a buffer handle
24 A5    0C A3   D0  0669  2214          MOVL    CSB$L_CDT(R3), -        ; Get CDT address in CDRP.
                     066E  2215                  CDRP$L_CDT(R5)
    52   2C A5   D0  066E  2216          MOVL    CDRP$L_LBUFH_AD(R5),R2  ; Buffer handle address
5C B3    F4 A2   0E  0672  2217          INSQUE  -CLUBTX$L_LBUFHNDL(R2), - ; Link to tail of partner queue
                     0677  2218                  @CSB$L_PARTNERQBL(R3)
    54   10 A3   D0  0677  2219          MOVL    CSB$L_PDT(R3), R4       ; Get PDT address.
    51   40 A5   DE  067B  2220          MOVAL   CDRP$L_CNXSVAPTE(R5), R1; Get SVAPTE block address.
    52   4A A5   9A  067F  2221          MOVZBL  CDRP$B_CNXRMOD(R5), R2  ; Get requestor's access mode.
                     0683  2222          MAP                             ; Map transfer block.
    52   2C A5   D0  0686  2223          MOVL    CDRP$L_LBUFH_AD(R5),R2  ; Buffer handle address in BTX
    52   F4 A2   0F  068A  2224          REMQUE  -CLUBTX$L_LBUFHNDL(R2),R2 ; Dequeue BTX
                     068E  2225
                     068E  2226 10$:
                     068E  2227          ; If the connection broke, these is no map at this point.
                     068E  2228
    56 A5    01   90  068E  2229          MOVB    #CDRP$K_REQUESTOR, -    ; Mark CDRP as belonging to a
                     0692  2230                  CDRP$B_CNXSTATE(R5)     ; requestor that has a buffer handle
24 A5    0C A3   D0  0692  2231          MOVL    CSB$L_CDT(R3), -        ; Get CDT address in CDRP -- must
```

```
                                0697  2232                    CDRP$L_CDT(R5)          ;   be initialized for REQUESTOR
                      FCB5  30  0697  2233            BSBW    CNX$SEND_MSG_CSB        ; Join common send message code.
                        0D  BB  069A  2234            PUSHR   #^M<R0,R2,R3>           ; Save registers.
                   51   2C  A5  D0  069C  2235            MOVL    CDRP$L_LBUFH_AD(R5),R1  ; Buffer handle address
                   50   F4  A1  9E  06A0  2236            MOVAB   -CLUBTX$L_LBUFHNDL(R1),R0 ; Address of BTX
                50 A5   28  A0  D0  06A4  2237            MOVL    CLUBTX$L_SAVED_PC(R0), - ; Copy return PC
                                06A9  2238                    CDRP$L_SAVEPC(R5)
                4C A5   2C  A0  D0  06A9  2239            MOVL    CLUBTX$L_MSGBLD(R0), -  ; Restore user's message build routine addre
                                06AE  2240                    CDRP$L_MSGBLD(R5)
                        07  6E  E9  06AE  2241            BLBC    (SP),20$                ; Branch on failure -- map already deallocat
                        50      DD  06B1  2242            PUSHL   R0                      ; Save BTX address
                                06B3  2243            UNMAP                           ; Release buffer handle
                        01      BA  06B6  2244            POPR    #^M<R0>                 ; Restore BTX address
                00000000'GF     16  06B8  2245 20$:       JSB     G^EXE$DEANONPAGED       ; Deallocate the BTX.
                        2C  A5  D4  06BE  2246            CLRL    CDRP$L_LBUFH_AD(R5)     ; Forget deallocated storage
                        0D      BA  06C1  2247            POPR    #^M<R0,R2,R3>           ; Restore saved registers.
                        50  B5  17  06C3  2248            JMP     @CDRP$L_SAVEPC(R5)      ; Return to mainline code.
                                06C6  2249
                                06C6  2250            .DISABLE LSB
                                06C6  2251
                                06C6  2252 ;
                                06C6  2253 ; Pre-Message build routine for block transfer requests.
                                06C6  2254 ; Do block transfer specific message setup and then transfer control to
                                06C6  2255 ; user's message build routine.
                                06C6  2256 ;
                                06C6  2257 BLD_BLKXFR_HDR:
                   50   2C  A5  D0  06C6  2258            MOVL    CDRP$L_LBUFH_AD(R5), R0 ; Get local buffer handle address.
                   51   F4  A0  9E  06CA  2259            MOVAB   -CLUBTX$L_LBUFHNDL(R0),R1 ; BTX address
                                06CE  2260            ASSUME  CLUBTX$S_LBUFHNDL EQ 12
                0C A2       80  7D  06CE  2261            MOVQ    (R0)+, -                ; Plant local buffer handle in
                                06D2  2262                    CLSMSG$L_REQR_BUFH(R2)
                14 A2       60  D0  06D2  2263            MOVL    (R0), -                 ; in message buffer.
                                06D6  2264                    CLSMSG$L_REQR_BUFH+8(R2)
                     2C     B1  17  06D6  2265            JMP     @CLUBTX$L_MSGBLD(R1)    ; Jump to user's message build routine
                                06D9  2266
                                06D9  2267
                                06D9  2268 ; Enter here when a block transfer retry message is received from the partner.
                                06D9  2269 ; Deallocate the message buffer and the original RSPID.
                                06D9  2270 ; Branch to reissue the request.
                                06D9  2271 ;
                                06D9  2272 ;       R2:             Incoming message buffer address
                                06D9  2273 ;       R3:             CSB address
                                06D9  2274 ;       R4:             PDT address
                                06D9  2275 ;
                                06D9  2276 BLKXFR_RETRY:
                55   0C  A2  D0  06D9  2277            MOVL    CLMBLK$L_RSPID(R2),R5   ; Fetch RSPID from message
                                06DD  2278            FIND_RSPID_RDTE                 ; Look up RDPID
                     13     50  E9  06E3  2279            BLBC    R0,10$                  ; Branch on error
                   55   65  D0  06E6  2280            MOVL    RD$L_CDRP(R5),R5        ; Fetch CDRP of requestor
                56 A5   01  91  06E9  2281            CMPB    #CDRP$K_REQUESTOR, -    ; Test CDRP state
                                06ED  2282                    CDRP$B_CNXSTATE(R5)
                        0E  12  06ED  2283            BNEQ    20$                     ; Branch if state invalid
                1C A5   52  D0  06EF  2284            MOVL    R2,CDRP$L_MSG_BUF(R5)   ; Save message buffer
                     FA90   30  06F3  2285            BSBW    CLEANUP_CDRP            ; Deallocate RSPID and/or message buffer
                     FF66   31  06F6  2286            BRW     BLOCK_XFER              ; Branch to reissue the request
                                06F9  2287
                                06F9  2288 10$:       BUG_CHECK       CNXMGRERR,FATAL ; Invalid RSPID received
```

```
                06FD  2289
                06FD  2290 20$:     BUG_CHECK       CNXMGRERR,FATAL ; CDRP in unexpected state
                0701  2291
```

```
0701  2293          .SBTTL  CNX$PARTNER_INIT_CSB - Init block transfer partner
0701  2294 ;++
0701  2295 ; FUNCTIONAL DESCRIPTION:
0701  2296 ;
0701  2297 ;       This routine is called by the partner's received message routine once
0701  2298 ;       the need for a block transfer is recognized.  It must be called before
0701  2299 ;       the thread initiated by the incoming message forks.  A BTX (or CLUBTX)
0701  2300 ;       is allocated.  It contains a fixed region in which the requestor's
0701  2301 ;       CSID and other useful information is stored, a copy of the incoming
0701  2302 ;       message buffer, and additional space as requested by the arguments to
0701  2303 ;       this routine.  The BTX is initialized.
0701  2304 ;
0701  2305 ;       The address of the client's broken connection error routine is among
0701  2306 ;       the arguments to this routine.  This address is stored in the BTX.
0701  2307 ;       Should the connection between the partner and the requestor break at
0701  2308 ;       anytime before the response message is successfully transmitted to the
0701  2309 ;       requestor, this error routine will be called.
0701  2310 ;
0701  2311 ;       ERROR ROUTINE INPUTS:
0701  2312 ;
0701  2313 ;               R1      Address of requested non-paged pool buffer (0 if none)
0701  2314 ;               R2      Address of copy of original message
0701  2315 ;               R3      CSB address (or zero if none exists)
0701  2316 ;               R5      CDRP address
0701  2317 ;
0701  2318 ;       ERROR ROUTINE OUTPUTS:
0701  2319 ;
0701  2320 ;               R0-R5 may be destroyed
0701  2321 ;
0701  2322 ;               Client is responsible for deallocating CDRP.  All other
0701  2323 ;               structures are deallocated by the connection manager.
0701  2324 ;
0701  2325 ;       Once control is returned from this routine, the thread initiated by
0701  2326 ;       the incoming message may fork.
0701  2327 ;
0701  2328 ; CALLING SEQUENCE:
0701  2329 ;
0701  2330 ;       BSBW    CNX$PARTNER_INIT_CSB
0701  2331 ;
0701  2332 ; INPUTS:
0701  2333 ;
0701  2334 ;       R1      Desired size of non-paged pool buffer
0701  2335 ;       R2      Incoming message buffer address
0701  2336 ;       R3      CSB address
0701  2337 ;       R4      Error cleanup routine address
0701  2338 ;       R5      CDRP address
0701  2339 ;       (SP)    Return address for the caller
0701  2340 ;       4(SP)   Return address for the caller's caller
0701  2341 ;
0701  2342 ; IMPLICIT INPUTS:
0701  2343 ;
0701  2344 ;       CSB$L_CSID(R3)  CSID of the node requesting this block transfer
0701  2345 ;       CDRP$L_SAVD_RTN(R5) & CDRP$L_MSG_BUF(R5) used as scratch areas
0701  2346 ;
0701  2347 ; OUTPUTS:
0701  2348 ;
0701  2349 ;       R0 - R1 Destroyed
```

```
                                0701  2350 ;       R2      Address of copy of requestor's message buffer
                                0701  2351 ;       R3      CSB address (unchanged)
                                0701  2352 ;       R4      Address of allocated non-paged pool buffer
                                0701  2353 ;       R5      CDRP address (unchanged)
                                0701  2354 ;
                                0701  2355 ; IMPLICIT OUTPUTS:
                                0701  2356 ;
                                0701  2357 ;--
                                0701  2358 CNX$PARTNER_INIT_CSB::
                                0701  2359
                      0214   30 0701  2360         BSBW    CNX$INIT_CDRP           ; Initialize the CDRP.
                                0704  2361         ASSUME  CDRP$L_MSG_BUF_EQ <CDRP$L_SAVD_RTN + 4>
                   18 A5   51 7D 0704  2362         MOVQ    R1, CDRP$L_SAVD_RTN(R5) ; Save requested buffer size and
                                0708  2363                                         ; message buffer address.
                                0708  2364
                51   009B C1 9E 0708  2365 10$:     MOVAB   <CLSMSG$K_MAXMSG+ -     ; Sum message buffer, requested buffer and B
                                070D  2366                 CLUBTX$K_LENGTH>(R1), R1
                 00000000'GF 16 070D  2367         JSB     G^EXE$ALONONPAGED       ; Allocate extended BTX.
                       6A 50 E9 0713  2368         BLBC    R0, 90$                 ; Branch if allocation failed.
                                0716  2369
                   08 A2   51 B0 0716  2370         MOVW    R1, CLUBTX$W_SIZE(R2)   ; Set allocation size.
                0A A2   0465 8F B0 071A  2371         MOVW    #<DYN$C_CLU_BTX*^x100+ -; Set structure type and subtype
                                0720  2372                 DYN$C_CLU>,-            ; fields.
                                0720  2373                 CLUBTX$B_TYPE(R2)
                   18 A2   55 D0 0720  2374         MOVL    R5, CLUBTX$L_CDRP(R2)   ; Setup CDRP pointer in BTX.
                   20 A2   54 D0 0724  2375         MOVL    R4, CLUBTX$L_ERRADDR(R2); Save error action routine address.
                                0728  2376         ASSUME  CLUBTX$S_LBUFHNDL EQ 12
                         0C A2 7C 0728  2377         CLRQ    CLUBTX$L_LBUFHNDL(R2)   ; Zero local buffer handle area.
                         14 A2 D4 072B  2378         CLRL    CLUBTX$L_LBUFHNDL+8(R2)
                2C A5    0C A2 DE 072E  2379         MOVAL   CLUBTX$L_LBUFHNDL(R2), -; Set CDRP local buffer handle
                                0733  2380                 CDRP$L_LBUFH_AD(R5)     ; pointer to point to BTX area.
                                0733  2381 ; *** I don't understand why we save the CSID since on every connection breakage
                                0733  2382 ; *** all of this is flushed!
                1C A2    4C A3 D0 0733  2383         MOVL    CSB$L_CSID(R3), -       ; Save CSID of requestor in BTX.
                                0738  2384                 CLUBTX$L_CSID(R2)       ; (Can't save CSB, since connection
                                0738  2385                                         ; status may change during local setup.)
                   56 A5   03 90 0738  2386         MOVB    #CDRP$K_PART_IDLE, -    ; Mark CDRP as belonging to an
                                073C  2387                 CDRP$B_CNXSTATE(R5)     ; idling partner
                24 A5    0C A3 D0 073C  2388         MOVL    CSB$L_CDT(R3), -        ; Get CDT address in CDRP.
                                0741  2389                 CDRP$L_CDT(R5)
                   5C B3   62 0E 0741  2390         INSQUE  CLUBTX$L_XQFL(R2), -    ; Queue BTX to partners queue.
                                0745  2391                 @CSB$L_PARTNERQBL(R3)
                                0745  2392
                   54   18 A5 D0 0745  2393         MOVL    CDRP$L_SAVD_RTN(R5),R4  ; Was a buffer requested?
                           05 13 0749  2394         BEQL    40$                     ; Branch in no buffer requested.
                                074B  2395         MOVAB   -                       ; Get BTX plus max message buf. size
                    54   009B C2 9E 074B  2396                 CLSMSG$K_MAXMSG+CLUBTX$K_LENGTH(R2), -
                                0750  2397                 R4                      ; plus requested buffer address.
                   24 A2   54 D0 0750  2398 40$:    MOVL    R4, CLUBTX$L_USER_BUF(R2) ; Save requested buffer address.
                                0754  2399
                      7E   53 7D 0754  2400         MOVQ    R3, -(SP)               ; Save CSB & user buffer addresses.
                         30 A2 9F 0757  2401         PUSHAB  CLUBTX$T_MSG_BUF(R2)    ; Save address of copied msg. buf.
                           55 DD 075A  2402         PUSHL   R5                      ; Save CDRP address.
30 A2    1C B5   006B 8F 28 075C  2403         MOVC3   #CLSMSG$K_MAXMSG, -     ; Copy incoming message to
                                0764  2404                 @CDRP$L_MSG_BUF(R5), -  ; the BTX.
                                0764  2405                 CLUBTX$T_MSG_BUF(R2)
                        55 8ED0 0764  2406         POPL    R5                      ; Restore CDRP address.
```

```
                53   04 AE    D0  0767  2407          MOVL    4(SP), R3                       ; Get CSB address.
                54   10 A3    D0  076B  2408          MOVL    CSB$L_PDT(R3), R4               ; Setup PDT address.
             24 A5   0C A3    D0  076F  2409          MOVL    CSB$L_CDT(R3), -                ; Setup CDT address.
                                  0774  2410                  CDRP$L_CDT(R5)
                                  0774  2411          DEALLOC_MSG_BUF                         ; Deallocate incoming message buffer.
                                  0777  2412
          0C A5   086A'CF    9E  0777  2413          MOVAB   W^BLOCK_FAIL, -                 ; Set up resumption address for
                                  077D  2414                  CDRP$L_FPC(R5)                  ; connection failure
                       1C    BA  077D  2415          POPR    #^M<R2,R3,R4>                   ; Restore copied message buf. addr.,
                                  077F  2416                                                  ; CSB, and user buffer addresses.
                             05  077F  2417          RSB                                     ; Return to caller.
                                  0780  2418
                                  0780  2419 ;
                                  0780  2420 ; BTX allocation failure
                                  0780  2421 ;       This is not an elegant solution to BTX allocation failure, but it is
                                  0780  2422 ;       easy.  If the BTX allocation fails, break the connection.
                                  0780  2423 ;
             52    1C A5    D0  0780  2424 90$:     MOVL    CDRP$L_MSG_BUF(R5),R2           ; Message buffer address
                   1C A5    D4  0784  2425          CLRL    CDRP$L_MSG_BUF(R5)
                     FE4B   30  0787  2426          BSBW    CNX$RCV_REJECT                  ; Break connection, rejecting received messa
                5E    04    C0  078A  2427          ADDL2   #4,SP                           ; Drop caller's address
                50    55    D0  078D  2428          MOVL    R5,R0                           ; CDRP address
          00000000'GF       17  0790  2429          JMP     G^EXE$DEANONPAGED               ; Delete CDRP and return to caller's caller
                                  0796  2430
```

```
0796  2432          .SBTTL  CNX$BLOCK_READ - Partner block read
0796  2433          .SBTTL  CNX$BLOCK_READ_IRP - Partner block read with IRP
0796  2434          .SBTTL  CNX$BLOCK_WRITE - Partner block write
0796  2435          .SBTTL  CNX$BLOCK_WRITE_IRP - Partner block write with IRP
0796  2436 ;++
0796  2437 ; FUNCTIONAL DESCRIPTION:
0796  2438 ;
0796  2439 ;       These routines are called on a block transfer partner node to initiate
0796  2440 ;       an actual block transfer.
0796  2441 ;
0796  2442 ;       These routines control allocation of all SCS resources.  Because these
0796  2443 ;       routines must allocate the SCS mapping resources to be used for the
0796  2444 ;       local buffer handle and use the supplied CDRP to perform a block
0796  2445 ;       transfer, they require specific use of CDRP$L_VAL1 through CDRP$L_VAL8
0796  2446 ;       which would otherwise be available to a client routine.
0796  2447 ;
0796  2448 ; CALLING SEQUENCE:
0796  2449 ;
0796  2450 ;       BSBW    CNX$BLOCK_READ (read from requestor to partner)
0796  2451 ;       BSBW    CNX$BLOCK_READ_IRP (read with an IRP on the partner)
0796  2452 ;       BSBW    CNX$BLOCK_WRITE (write from partner to requestor)
0796  2453 ;       BSBW    CNX$BLOCK_WRITE_IRP (write with an IRP on the partner)
0796  2454 ;
0796  2455 ; INPUT PARAMETERS:
0796  2456 ;
0796  2457 ;       R5      CDRP address
0796  2458 ;       (SP)    Return address for the caller
0796  2459 ;       4(SP)   Return address for the caller's caller
0796  2460 ;
0796  2461 ; IMPLICIT INPUTS:
0796  2462 ;
0796  2463 ;       CDRP$L_LBUFH_AD (CDRP)          address of buffer handle in BTX
0796  2464 ;       CLUBTX$L_CSID( BTX )            requestor's CSID
0796  2465 ;       CLUBTX$T_MSG_BUF( BTX )         copy of incoming message buffer
0796  2466 ;       CLSMSG$L_REQR_BUFH( MSG )       requestor's buffer handle descriptor
0796  2467 ;
0796  2468 ;       CDRP$L_RSPID(R5) and CDRP$L_MSG_BUF(R5) must contain zero.
0796  2469 ;
0796  2470 ;       CDRP$L_LBOFF must contain the offset (from the address described by
0796  2471 ;       SVAPTE - BOFF) in the local buffer at which the transfer is to begin.
0796  2472 ;       (This is provided to allow segmenting transfers.)
0796  2473 ;
0796  2474 ;       CDRP$L_RBOFF must contain the offset in the remote buffer at which the
0796  2475 ;       transfer is to begin.  (This is provided to allow segmenting
0796  2476 ;       transfers.)
0796  2477 ;
0796  2478 ;       CDRP$L_XCT_LEN must contain the number of bytes to transfer.
0796  2479 ;
0796  2480 ;       --- FOR CNX$BLOCK_READ and CNX$BLOCK_WRITE:
0796  2481 ;
0796  2482 ;       CDRP$L_CNXSVAPTE(R5)    System virtual address of the first PTE
0796  2483 ;                               describing the block transfer buffer
0796  2484 ;       CDRP$W_CNXBOFF(R5)      Byte offset of first byte in block transfer
0796  2485 ;                               buffer
0796  2486 ;       CDRP$L_CNXBCNT(R5)      Number of bytes in block transfer
0796  2487 ;       CDRP$B_CNXRMOD(R5)              Access mode of requestor
0796  2488 ;
```

```
                        0796  2489 ;       --- FOR CNX$BLOCK_READ_IRP and CNX$BLOCK_WRITE_IRP:
                        0796  2490 ;
                        0796  2491 ;       CDRP$L_SVAPTE(R5)       System virtual address of the first PTE
                        0796  2492 ;                               describing the block transfer buffer
                        0796  2493 ;       CDRP$W_BOFF(R5)         Byte offset of first byte in block transfer
                        0796  2494 ;                               buffer
                        0796  2495 ;       CDRP$L_BCNT(R5)         Number of bytes in block transfer
                        0796  2496 ;       CDRP$B_RMOD(R5)         Access mode of requestor
                        0796  2497 ;
                        0796  2498 ;       This routine requires that several CDRP fields be initialized to zero.
                        0796  2499 ;       CNX$PARTNER_INIT_CSB correctly performs this initialization.
                        0796  2500 ;
                        0796  2501 ; OUTPUT PARAMETERS:
                        0796  2502 ;
                        0796  2503 ;       R0 - R1 Destroyed
                        0796  2504 ;       R2      Address of copy of requestor's message buffer
                        0796  2505 ;       R3      Destroyed
                        0796  2506 ;       R4      Address of allocated non-paged pool buffer
                        0796  2507 ;       R5      CDRP address
                        0796  2508 ;
                        0796  2509 ; IMPLICIT OUTPUTS:
                        0796  2510 ;
                        0796  2511 ;       CDRP$L_VAL1(R5) through CDRP$L_VAL8 are destroyed by this routine or
                        0796  2512 ;       overlayed by implicit inputs to this routine.
                        0796  2513 ;
                        0796  2514 ;       Assuming proper cooperation on the partner node, the block transfer
                        0796  2515 ;       buffer has either been copied to the partner node or over written with
                        0796  2516 ;       information from the partner node.
                        0796  2517 ;
                        0796  2518 ; SIDE EFFECTS:
                        0796  2519 ;
                        0796  2520 ;       R0 - R4 are destroyed.
                        0796  2521 ;
                        0796  2522 ;--
                        0796  2523
                        0796  2524          ASSUME  CDRP$L_CNXSVAPTE GT CDRP$L_LBUFH_AD
                        0796  2525          ASSUME  CDRP$L_CNXSVAPTE GT CDRP$L_LBOFF
                        0796  2526          ASSUME  CDRP$L_CNXSVAPTE GT CDRP$L_RBUFH_AD
                        0796  2527          ASSUME  CDRP$L_CNXSVAPTE GT CDRP$L_RBOFF
                        0796  2528          ASSUME  CDRP$L_CNXSVAPTE GT CDRP$L_XCT_LEN
                        0796  2529
                        0796  2530          ASSUME  CDRP$B_RMOD-CDRP$L_IOQFL        EQ      IRP$B_RMOD
                        0796  2531          ASSUME  CDRP$L_SVAPTE-CDRP$L_IOQFL      EQ      IRP$L_SVAPTE
                        0796  2532          ASSUME  CDRP$W_BOFF-CDRP$L_IOQFL         EQ      IRP$W_BOFF
                        0796  2533          ASSUME  CDRP$L_BCNT-CDRP$L_IOQFL        EQ      IRP$L_BCNT
                        0796  2534          ASSUME  <CDRP$Q_CNXBOFF - CDRP$L_CNXSVAPTE> EQ -
                        0796  2535                                                  <CDRP$W_BOFF - CDRP$L_SVAPTE>
                        0796  2536          ASSUME  <CDRP$L_CNXBCNT - CDRP$L_CNXSVAPTE> EQ -
                        0796  2537                                                  <CDRP$L_BCNT - CDRP$L_SVAPTE>
                        0796  2538
                        0796  2539          .ENABLE LSB
                        0796  2540
                        0796  2541 CNX$BLOCK_READ_IRP::
                        0796  2542
4C A5   FC37 CF   9E    0796  2543          MOVAB   W^REQUEST_DATA, -       ; Setup for read function.
                        079C  2544                  CDRP$L_MSGBLD(R5)
              06  11    079C  2545          BRB     10$                     ; Branch to common IRP code.
```

```
                     079E  2546
                     079E  2547 CNX$BLOCK_WRITE_IRP::
                     079E  2548
4C A5   FC22 CF   9E 079E  2549          MOVAB   W^SEND_DATA, -          ; Setup for write function.
                     07A4  2550                  CDRP$L_MSGBLD(R5)
  40 A5    CC A5   7D 07A4  2551 10$:     MOVQ    CDRP$L_SVAPTE(R5), -    ; Copy SVAPTE and BOFF.
                     07A9  2552                  CDRP$L_CNXSVAPTE(R5)
  46 A5    D2 A5   D0 07A9  2553          MOVL    CDRP$L_BCNT(R5), -      ; Copy BCNT.
                     07AE  2554                  CDRP$L_CNXBCNT(R5)
  4A A5    AB A5   90 07AE  2555          MOVB    CDRP$B_RMOD(R5), -      ; Copy RMOD.
                     07B3  2556                  CDRP$B_CNXRMOD(R5)
             12      11 07B3  2557          BRB     20$                     ; Branch to common block xfer code.
                     07B5  2558
                     07B5  2559 CNX$BLOCK_READ::
                     07B5  2560
4C A5   FC18 CF   9E 07B5  2561          MOVAB   W^REQUEST_DATA, -       ; Setup for read function.
                     07BB  2562                  CDRP$L_MSGBLD(R5)
             0A      11 07BB  2563          BRB     20$                     ; Branch to common block xfer code.
                     07BD  2564
        5E   04      C0 07BD  2565 14$:     ADDL2   #4,SP                   ; Eliminate callers address
             05      05 07C0  2566 15$:     RSB                             ; Connection is failing, exit
                     07C1  2567
                     07C1  2568 CNX$BLOCK_WRITE::
                     07C1  2569
4C A5   FBFF CF   9E 07C1  2570          MOVAB   W^SEND_DATA, -          ; Setup for write function.
                     07C7  2571                  CDRP$L_MSGBLD(R5)
                     07C7  2572
                     07C7  2573 20$:     DISPATCH  CDRP$B_CNXSTATE(R5),type=B,prefix=CDRP$K_ -
                     07C7  2574                  <-
                     07C7  2575                  <PART_IDLE,30$>, -      ; Idle partner
                     07C7  2576                  <NORMAL,14$>, -         ; Return if turned into NORMAL
                     07C7  2577                  >
                     07D4  2578          BUG_CHECK       CNXMGRERR,FATAL ; Invalid CNX state
                     07D8  2579
                     07D8  2580 ;
                     07D8  2581 ; If the CSID of the remote node involved in the transfer is invalid,
                     07D8  2582 ; bugcheck (unless the following case pertains):
                     07D8  2583 ;
                     07D8  2584 ; The following closes a window where a node has been removed from the
                     07D8  2585 ; cluster, pre-cleanup has been done, and an SCS DISCONNECT is in progress.
                     07D8  2586 ; A block transfer partner may initiate a request at this time because the
                     07D8  2587 ; error entry has not yet been called.  The appropriate behavior is to
                     07D8  2588 ; detect this case and drop the thread -- it will be returned via the
                     07D8  2589 ; error entry after the DISCONNECT completes.
                     07D8  2590 ;
  53    24 A5   D0 07D8  2591 25$:     MOVL    CDRP$L_CDT(R5),R3       ; Fetch CDT address
             18      13 07DC  2592          BEQL    29$                     ; Serious error if no CDT address
  53    5C A3   D0 07DE  2593          MOVL    CDT$L_AUXSTRUC(R3),R3   ; Fetch CSB address
             12      13 07E2  2594          BEQL    29$                     ; Serious error if no CSB address
  07    43 A3   91 07E4  2595          CMPB    CSB$B_STATE(R3), -      ; Is connection in DISCONNECT
                     07E8  2596                  #CSB$K_DISCONNECT       ;   state?
             0C      12 07E8  2597          BNEQ    29$                     ; No, serious error
07 60 A3     02      E1 07EA  2598          BBC     #CSB$V_REMOVED, -       ; If node not removed from cluster,
                     07EF  2599                  CSB$L_STATUS(R3),29$    ;   bugcheck
4C A3   1C A2   D1 07EF  2600          CMPL    CLUBTX$L_CSID(R2), -    ; Double check CSID
                     07F4  2601                  CSB$L_CSID(R3)
             26      13 07F4  2602          BEQL    40$                     ; OK if match
```

```
                                07F6  2603 29$:    BUG_CHECK       CNXMGRERR,FATAL ; Invalid CNX state
                                07FA  2604
      52    2C A5    0C    C3   07FA  2605 30$:    SUBL3   #CLUBTX$L_LBUFHNDL, -   ; Get BTX address
                                07FF  2606                    CDRP$L_LBUFH_AD(R5),R2
           53    1C A2    D0   07FF  2607          MOVL    CLUBTX$L_CSID(R2), R3   ; Get CSID.
                                0803  2608          CSID_TO_CSB error=25$, csb=R3   ; Translate CSID to CSB.
                 28 A2 8ED0    081C  2609 40$:    POPL    CLUBTX$L_SAVED_PC(R2)   ; Save caller's return PC.
                                0820  2610          TEST_CSB_OPEN no=15$            ; Branch if CSB not open.
           54    10 A3    D0   0826  2611          MOVL    CSB$L_PDT(R3), R4       ; Setup PDT address.
        24 A5    0C A3    D0   082A  2612          MOVL    CSB$L_CDT(R3), -        ; Setup CDT address.
                                082F  2613                  CDRP$L_CDT(R5)
                                082F  2614
           20 A5    01    D0   082F  2615          MOVL    #1,CDRP$L_RSPID(R5)     ; Request RSPID
                                0833  2616
           56 A5    05    90   0833  2617          MOVB    #CDRP$K_PART_MAP, -     ; Mark CDRP as belonging to a
                                0837  2618                  CDRP$B_CNXSTATE(R5)     ; partner waiting for a buffer handle.
        34 A5    3C A2    9E   0837  2619          MOVAB   <CLUBTX$T_MSG_BUF + -       ; Setup remote buffer handle
                                083C  2620                  CLSMSG$L_REQR_BUFH>(R2), -  ; address.
                                083C  2621                  CDRP$L_RBUFH_AD(R5)
           51    40 A5    DE   083C  2622          MOVAL   CDRP$L_CNXSVAPTE(R5), R1; Get SVAPTE block address.
           52    4A A5    9A   0840  2623          MOVZBL  CDRP$B_CNXRMOD(R5), R2  ; Get requestor's access mode.
                                0844  2624          MAP                             ; Map the local buffer.
                                0847  2625
           56 A5    02    90   0847  2626          MOVB    #CDRP$K_PARTNER, -      ; Mark CDRP as belonging to a
                                084B  2627                  CDRP$B_CNXSTATE(R5)     ; partner.
                 FB0B    30   084B  2628          BSBW    SEND_UNSEQ_MSG          ; Send an unsequenced with a special
                                084E  2629                                          ; message build routine.
                 19 50    E9   084E  2630          BLBC    R0,BLOCK_FAIL           ; Branch if connection has broken
                                0851  2631
           56 A5    03    90   0851  2632          MOVB    #CDRP$K_PART_IDLE, -    ; Return to idle partner CDRP CNX state.
                                0855  2633                  CDRP$B_CNXSTATE(R5)
        0C A5    6A'AF    9E   0855  2634          MOVAB   B^BLOCK_FAIL, -         ; Set up failure return
                                085A  2635                  CDRP$L_FPC(R5)
      50    2C A5    0C    C3   085A  2636          SUBL3   #CLUBTX$L_LBUFHNDL, -   ; Get BTX address into R0
                                085F  2637                  CDRP$L_LBUFH_AD(R5), R0
           52    30 A0    9E   085F  2638          MOVAB   CLUBTX$T_MSG_BUF(R0),R2 ; Get requestor's message buffer address.
           54    24 A0    D0   0863  2639          MOVL    CLUBTX$L_USER_BUF(R0),R4 ; Get address of client requested buffer.
                 28 B0    17   0867  2640          JMP     @CLUBTX$L_SAVED_PC(R0)  ; Return to caller.
                                086A  2641
                                086A  2642 ;
                                086A  2643 ; Get here when connection breaks
                                086A  2644 ;
                                086A  2645 BLOCK_FAIL:
                    00AB    30  086A  2646          BSBW    CNX$INIT_CDRP           ; Initialize CDRP
        4C A5    A0'AF    9E   086D  2647          MOVAB   B^50$,CDRP$L_MSGBLD(R5) ; Message build routine
           52    2C A5    D0   0872  2648          MOVL    CDRP$L_LBUFH_AD(R5),R2  ; BTX address
           50    F4 A2    0F   0876  2649          REMQUE  -CLUBTX$L_LBUFHNDL(R2),R0 ; Remove from queue
                    60    7C   087A  2650          CLRQ    CLUBTX$L_XQFL(R0)       ; Invalidate linkage
           56 A5    00    90   087C  2651          MOVB    #CDRP$K_NORMAL, -       ; Set CNXSTATE to NORMAL
                                0880  2652                  CDRP$B_CNXSTATE(R5)
                 FACC    30   0880  2653          BSBW    CNX$SEND_MSG_CSB        ; Send retry message
      52    2C A5    0C    C3   0883  2654          SUBL3   #CLUBTX$L_LBUFHNDL, -   ; Get BTX address
                                0888  2655                  CDRP$L_LBUFH_AD(R5),R2
                                0888  2656 ;
                                0888  2657 ; ERROR ACTION ROUTINE INPUTS:
                                0888  2658 ;
                                0888  2659 ;       R1      Address of requested non-paged pool buffer (0 if none)
```

```
                        0888  2660 ;       R2      Address of copy of original message
                        0888  2661 ;       R3      CSB address
                        0888  2662 ;       R5      CDRP address
                        0888  2663 ;
                        0888  2664 ; ERROR ACTION ROUTINE OUTPUTS:
                        0888  2665 ;
                        0888  2666 ;       R0-R5 may be destroyed
                        0888  2667 ;
                        0888  2668 ;       Client is responsible for deallocating CDRP.  All other structures
                        0888  2669 ;       will be deallocated here.
                        0888  2670 ;--
                        0888  2671
            52    DD    0888  2672          PUSHL   R2                          ; Save BTX address.
                        088A  2673
     55   18 A2   D0    088A  2674          MOVL    CLUBTX$L_CDRP(R2),R5        ; Fetch CDRP address
     51   24 A2   D0    088E  2675          MOVL    CLUBTX$L_USER_BUF(R2), R1   ; Get requested pool address.
        52   30   C0    0892  2676          ADDL    #CLUBTX$T_MSG_BUF, R2       ; Get pointer to original message.
          F0 B2   16    0895  2677          JSB     @<CLUBTX$L_ERRADDR -        ; Call user's error action routine.
                        0898  2678          - CLUBTX$T_MSG_BUF>(R2)
                        0898  2679
            01    BA    0898  2680          POPR    #^M<R0>                     ; Restore BTX address.
 00000000'GF    17    089A  2681          JMP     G^EXE$DEANONPAGED           ; Deallocate it and return to caller.
                        08A0  2682
                        08A0  2683
                        08A0  2684 ; Message build routine for retry messages
                        08A0  2685 ;
   08 A2   07    90    08A0  2686 50$:    MOVB    #CLSMSG$K_FAC_BLK, -        ; Set up facility code
                        08A4  2687          CLSMSG$B_FACILITY(R2)
   50   2C A5   D0    08A4  2688          MOVL    CDRP$L_LBUFH_AD(R5),R0      ; Address of offset in BTX
 0C A2   28 A0   D0    08A8  2689          MOVL    <CLUBTX$T_MSG_BUF+ -        ; Set up response RSPID
                        08AD  2690          CLSMSG$L_RSPID- -
                        08AD  2691          CLUBTX$L_LBUFHNDL>(R0), -
                        08AD  2692          CLMBLK$L_RSPID(R2)
                  05    08AD  2693          RSB
                        08AE  2694
                        08AE  2695          .DISABLE        LSB
```

```
                     08AE  2697
                     08AE  2698          .SBTTL  CNX$PARTNER_FINISH - Complete partner's end of a block transfer
                     08AE  2699          .SBTTL  CNX$PARTNER_RESPOND - Send block transfer completed response
                     08AE  2700 ;++
                     08AE  2701 ; FUNCTIONAL DESCRIPTION:
                     08AE  2702 ;
                     08AE  2703 ;       One of these routines receives control when the partner's portion of
                     08AE  2704 ;       the block transfer operation has been completed.  A response message
                     08AE  2705 ;       is sent to the requestor node and the BTX, allocated by
                     08AE  2706 ;       CNX$PARTNER_INIT_CSB, is deallocated.  CNX$PARTNER_FINISH also
                     08AE  2707 ;       deallocates the input CDRP.
                     08AE  2708 ;
                     08AE  2709 ; CALLING SEQUENCE:
                     08AE  2710 ;
                     08AE  2711 ;       BRW     CNX$PARTNER_FINISH
                     08AE  2712 ;       BSBx    CNX$PARTNER_RESPOND
                     08AE  2713 ;
                     08AE  2714 ; INPUTS:
                     08AE  2715 ;
                     08AE  2716 ;       R5      CDRP address
                     08AE  2717 ;
                     08AE  2718 ; IMPLICIT INPUTS:
                     08AE  2719 ;
                     08AE  2720 ;       CDRP$L_LBUFH_AD(R5)     Fixed offset from BTX address
                     08AE  2721 ;       CLUBTX$L_CSID( BTX )    requestor's CSID
                     08AE  2722 ;       CLUBTX$T_MSG_BUF( BTX ) copy of incoming message buffer
                     08AE  2723 ;       CLSMSG$L_RSPID( MSG )   requestor's RSPID
                     08AE  2724 ;
                     08AE  2725 ;       CDRP$L_MSGBLD(R5) must contain the address of a message build routine.
                     08AE  2726 ;
                     08AE  2727 ;       CDRP$L_RSPID(R5) and CDRP$L_MSG_BUF(R5) must contain zero.
                     08AE  2728 ;
                     08AE  2729 ;       Any information that the message build routine requires should
                     08AE  2730 ;       be in the CDRP or pointed to by pointers in the CDRP.
                     08AE  2731 ;
                     08AE  2732 ; OUTPUTS:
                     08AE  2733 ;
                     08AE  2734 ;       R5      CDRP address (as input)
                     08AE  2735 ;
                     08AE  2736 ; IMPLICIT OUTPUTS:
                     08AE  2737 ;
                     08AE  2738 ;       CDRP$L_VAL8(R5) is overlayed by the client's status field CDRP$B_CLTSTS(R5).
                     08AE  2739 ;
                     08AE  2740 ;       The response message is sent to the requestor.  The BTX associated
                     08AE  2741 ;       with this partner operation is dequeued and deallocated.  For
                     08AE  2742 ;       CNX$PARTNER_FINISH, the input CDRP also is deallocated and this partner
                     08AE  2743 ;       request thread is terminated.
                     08AE  2744 ;
                     08AE  2745 ; SIDE EFFECTS:
                     08AE  2746 ;
                     08AE  2747 ;       The response message is sent to the requestor.
                     08AE  2748 ;
                     08AE  2749 ;--
                     08AE  2750
                     08AE  2751 CNX$PARTNER_FINISH::
          09   10    08AE  2752          BSBB    CNX$PARTNER_RESPOND     ; Send response to requestor
     50   55   D0    08B0  2753          MOVL    R5, R0                  ; Copy the CDRP address.
```

```
                00000000'GF   17  08B3  2754          JMP     G^EXE$DEANONPAGED       ; Deallocate CDRP and return
                              08B9  2755                                                ; (to whomever).
                              08B9  2756
                              08B9  2757 CNX$PARTNER_RESPOND::
                 56 A5   03   91  08B9  2758          CMPB    #CDRP$K_PART_IDLE, -    ; Test CDRP state
                              08BD  2759                  CDRP$B_CNXSTATE(R5)
                       22    12  08BD  2760          BNEQ    10$                     ; Branch if no expected state
                    50 2C A5 D0  08BF  2761          MOVL    CDRP$L_LBUFH_AD(R5),R0  ; Get offset in BTX
                    50 F4 A0 0F  08C3  2762          REMQUE  -CLUBTX$L_LBUFHNDL(R0),R0 ; Remove BTX from partner queue
             58 A5  34 A0 D0  08C7  2763          MOVL    <CLUBTX$T_MSG_BUF + -   ; Copy requestor's RSPID to
                              08CC  2764                  CLSMSG$L_RSPID>(R0), -  ; return RSPID (for response).
                              08CC  2765                  CDRP$L_RETRSPID(R5)     ; (This destroys the saved BTX address.)
                    2C A5  D4  08CC  2766          CLRL    CDRP$L_LBUFH_AD(R5)     ; No more BTX
                 56 A5   00   90  08CF  2767          MOVB    #CDRP$K_NORMAL, -       ; Enter the normal state
                              08D3  2768                  CDRP$B_CNXSTATE(R5)
                    1C A0  DD  08D3  2769          PUSHL   CLUBTX$L_CSID(R0)       ; Get requestor's CSID.
                00000000'GF   16  08D6  2770          JSB     G^EXE$DEANONPAGED       ; Deallocate the BTX
                        08    BA  08DC  2771          POPR    #^M<R3>                 ; Restore CSID
                      FA55    31  08DE  2772          BRW     CNX$SEND_MSG            ; Send the response message.
                              08E1  2773
                              08E1  2774 10$:     BUG_CHECK       CNXMGRERR,FATAL ; Invalid CDRP state
```

```
08E5  2776          .SBTTL  CNX$ALLOC_CDRP - Allocate a CDRP & Convert CSID
08E5  2777          .SBTTL  CNX$ALLOC_CDRP_ONLY - Allocate a CDRP
08E5  2778          .SBTTL  CNX$ALLOC_WARMCDRP - Allocate CDRP w/ RSPID and message buffer
08E5  2779          .SBTTL  CNX$ALLOC_WARMCDRP_CSB - Allocate warm CDRP using CSB
08E5  2780          .SBTTL  CNX$INIT_CDRP - Initialize a CDRP
08E5  2781 ;++
08E5  2782 ; FUNCTIONAL DESCRIPTION:
08E5  2783 ;
08E5  2784 ;       These routines are called to allocate CDRPs and initialize various
08E5  2785 ;       fields.
08E5  2786 ;
08E5  2787 ;       CNX$ALLOC_CDRP allocates a CDRP from non-paged pool and initializes
08E5  2788 ;       various fields and converts a CSID to a CSB address.
08E5  2789 ;       CNX$ALLOC_CDRP_ONLY performs the same allocation and initialization
08E5  2790 ;       but does nothing with any CSIDs.
08E5  2791 ;
08E5  2792 ;       CNX$ALLOC_WARMCDRP and CNX$ALLOC_WARMCDRP_CSB attempt to allocate a
08E5  2793 ;       CDRP from a free list on the CSB.  These CDRPs already have a response
08E5  2794 ;       id. and message buffer allocated.  If the free list is empty then a
08E5  2795 ;       CDRP is allocated from non-paged pool and initialized as before.
08E5  2796 ;       However, the CDRP$L_RSPID field is set to 1 so that CNX$SEND_MSG will
08E5  2797 ;       allocate a response id. (and also a message buffer).  The CSID
08E5  2798 ;       supplied as an argument to CNX$ALLOC_WARMCDRP is converted to a
08E5  2799 ;       CSB address.
08E5  2800 ;
08E5  2801 ;       CNX$INIT_CDRP simply initializes the CDRP whose address is supplied in
08E5  2802 ;       R5.
08E5  2803 ;
08E5  2804 ; CALLING SEQUENCE:
08E5  2805 ;
08E5  2806 ;       BSBW    CNX$ALLOC_CDRP - Allocate a CDRP and convert CSID to CSB
08E5  2807 ;       BSBW    CNX$ALLOC_CDRP_ONLY - Allocate a CDRP only
08E5  2808 ;       BSBW    CNX$ALLOC_WARMCDRP - Allocate a CDRP w/ RSPID and msg buffer
08E5  2809 ;       BSBW    CNX$ALLOC_WARMCDRP - Allocate a warm CDRP using CSB address
08E5  2810 ;       BSBW    CNX$INIT_CDRP
08E5  2811 ;
08E5  2812 ;       IPL is at IPL$_SYNCH
08E5  2813 ;
08E5  2814 ; INPUT PARAMETERS:
08E5  2815 ;
08E5  2816 ;       R3      CSID (CNX$ALLOC_CDRP and CNX$ALLOC_WARMCDRP)
08E5  2817 ;       R3      CSB (CNX$ALLOC_WARMCDRP_CSB)
08E5  2818 ;       R5      CDRP address (CNX$INIT_CDRP only)
08E5  2819 ;
08E5  2820 ; OUTPUT PARAMETERS:
08E5  2821 ;
08E5  2822 ;       R0      Completion code
08E5  2823 ;       R3      CSB (CNX$ALLOC_CDRP and CNX$ALLOC_WARMCDRP)
08E5  2824 ;       R5      Address of CDRP
08E5  2825 ;
08E5  2826 ; IMPLICIT OUTPUTS:
08E5  2827 ;
08E5  2828 ;       Various fields in the CDRP are initialized to zero.
08E5  2829 ;
08E5  2830 ;       CNX$ALLOC_WARMCDRP, CNX$ALLOC_WARMCDRP_CSB, CNX$ALLOC_CDRP, and
08E5  2831 ;       CNX$ALLOC_CDRP_ONLY set CDRP$W_CDRPSIZE to CDRP$K_CM_LENGTH on all
08E5  2832 ;       newly allocated CDRPs.  CNX$INIT_CDRP does not alter CDRP$W_CDRPSIZE.
```

```
                                                M 9
                     ;       This assumes the size has been correctly set by the caller and is
                     ;       consistant with the preallocation of CDRPs for messages requiring
                     ;       responses in CNX$RCV_MSG.
                     ;
                     ;       If CNX$ALLOC_WARMCDRP or CNX$ALLOC_WARMCDRP_CSB was called, then a
                     ;       CDRP with RSPID and message buffer will be returned if one was
                     ;       available.  If none were available, then a CDRP is returned with a 1
                     ;       in the CDRP$L_RSPID field.  No status is returned to indicate whether
                     ;       or not the CDRP has a RSPID and message buffer.  The caller does not
                     ;       have to be concerned about this as CNX$SEND_MSG_CSB will allocate
                     ;       either or both if they are needed.
                     ;
                     ;       CNX$ALLOC_CDRP and CNX$ALLOC_WARMCDRP convert a CSID address (input in
                     ;       R3) to a CSB address (output in R3).  For CNX$ALLOC_WARMCDRP, this is
                     ;       necessary because the CSB contains the listhead for the warm CDRP
                     ;       queue.  CNX$ALLOC_CDRP provides similar functionality for requests
                     ;       which do not need a RSPID.  It is also easier for acknowledged message
                     ;       services clients to detect and handle an error from the allocate CDRP
                     ;       routines than it is to detect and handle an error from CNX$SEND_MSG.
                     ;       Note: the use of either of these two routines implies the use of
                     ;       CNX$SEND_MSG_CSB instead of CNX$SEND_MSG.  When CSID conversion is not
                     ;       relivant, use CNX$ALLOC_CDRP_ONLY.
                     ;
                     ; COMPLETION CODES:
                     ;
                     ;       SS$_NORMAL      Normal successful completion
                     ;       SS$_INSFMEM     Insufficient memory
                     ;                       (WARNING: If a CSID was input, it will have been
                     ;                       converted to a CSB when this error is returned.)
                     ;       SS$_NOSUCHNODE  Invalid CSID (CNX$ALLOC_CDRP and CNX$ALLOC_WARMCDRP)
                     ;
                     ; SIDE EFFECTS:
                     ;
                     ;       R1 - R2 are destroyed
                     ;--
                     
                             .ENABL  LSB
                     
                     CNX$ALLOC_WARMCDRP::
                             CSID_TO_CSB csb=R3, error=INV_CSID_NO_CLEANUP
                     
                     CNX$ALLOC_WARMCDRP_CSB::
   42 A3    97       DECB    CSB$B_WARMCDRPS(R3)      ; Decr. count of warm CDRPs
      0E    19       BLSS    20$                      ; No more
55 24 B3    0F       REMQUE  @CSB$L_WARMCDRPQFL(R3),R5 ; Allocate a free one
      04    1D       BVS     10$                      ; List is empty
   50 01    D0       MOVL    S^#SS$_NORMAL,R0
            05       RSB
                     
                     10$:    BUG_CHECK       CNXMGRERR,FATAL ; *** TEMPORARY
                     
   42 A3    96       INCB    CSB$B_WARMCDRPS(R3)      ; Adjust count back
      01    DD       PUSHL   #1                       ; Push contents of CDRP$L_RSPID
      26    11       BRB     30$
                     
                     CNX$INIT_CDRP::
                             ASSUME  CDRP$B_FIPL EQ CDRP$B_CD_TYPE+1
```

```
                0A A5   0839 8F   B0   0918  2890          MOVW    #<IPL$_SCS@8+DYN$C_CDRP>, -     ; Set CDRP type and
                                     091E  2891                   CDRP$B_CD_TYPE(R5)              ; fork IPL of IPL$_SCS
                      20 A5   D4   091E  2892          CLRL    CDRP$L_RSPID(R5)          ; Clear RSPID field.
                         38   11   0921  2893          BRB     40$                       ; Join common code.
                                     0923  2894
                                     0923  2895 CNX$ALLOC_CDRP::
                                     0923  2896          CSID_TO_CSB csb=R3, error=INV_CSID_NO_CLEANUP
                                     093C  2897
                                     093C  2898 CNX$ALLOC_CDRP_ONLY::
                         00   DD     093C  2899          PUSHL   #0                        ; Push contents of CDRP$L_RSPID
                   51    0060 8F   3C   093E  2900 30$:    MOVZWL  #CDRP$K_CM_LENGTH,R1     ; Size of CDRP
                    00000000'GF   16   0943  2901          JSB     G^EXE$ALONONPAGED         ; Allocate it
                          1C 50   E9   0949  2902          BLBC    R0,80$                    ; Unable to allocate it
                       55    52   D0   094C  2903          MOVL    R2,R5                     ; Use R5 from now on
                                     094F  2904          ASSUME  CDRP$B_CD_TYPE  EQ  CDRP$W_CDRPSIZE+2
                                     094F  2905          ASSUME  CDRP$B_FIPL EQ CDRP$B_CD_TYPE+1
            08 A5   08390060 8F   D0   094F  2906          MOVL    #<<<IPL$_SCS@8>+DYN$C_CDRP@16>+CDRP$K_CM_LENGTH>, -
                                     0957  2907                   CDRP$W_CDRPSIZE(R5)             ; Set size, type, and fork IPL.
                      20 A5 8ED0   0957  2908          POPL    CDRP$L_RSPID(R5)          ; Initialize RSPID to 0 or 1
                      1C A5   D4   095B  2909 40$:     CLRL    CDRP$L_MSG_BUF(R5)        ; Clear MSG_BUF
                      28 A5   D4   095E  2910          CLRL    CDRP$L_RWCPTR(R5)         ; and RWCPTR
                                     0961  2911          ASSUME  CDRP$K_NORMAL EQ 0
                                     0961  2912          ASSUME  CDRP$B_CNXSTATE EQ <CDRP$W_SENDSEQNM + 2>
                                     0961  2913          ASSUME  CDRP$L_RETRSPID EQ <CDRP$W_SENDSEQNM + 4>
                      54 A5   7C   0961  2914          CLRQ    CDRP$W_SENDSEQNM(R5)
                   50    01   D0   0964  2915          MOVL    #SS$_NORMAL,R0
                                05   0967  2916          RSB
                                     0968  2917
                         8E   D5   0968  2918 80$:     TSTL    (SP)+                     ; Pop CDRP$L_RSPID info from stack.
                50    0124 8F   3C   096A  2919          MOVZWL  #SS$_INSFMEM,R0           ; Set error return status.
                                05   096F  2920          RSB
                                     0970  2921
                                     0970  2922 INV_CSID_NO_CLEANUP:
                50    028C 8F   3C   0970  2923          MOVZWL  #SS$_NOSUCHNODE,R0        ; Signal invalid CSID in
                                05   0975  2924          RSB                               ; CNX$ALLOC_WARMCDRP and CNX$ALLOC_WARMCDRP
                                     0976  2925
                                     0976  2926          .DSABL  LSB
```

```
                                B 10
0976  2928          .SBTTL  CNX$DEALL_WARMCDRP_CSB - Deallocate a Warm CDRP using CSB
0976  2929 ;++
0976  2930 ; FUNCTIONAL DESCRIPTION:
0976  2931 ;
0976  2932 ;       This routine is called to deallocate a CDRP that contains
0976  2933 ;       a RSPID and a message buffer (actually in R2).  If the queue
0976  2934 ;       of free CDRPs on the CSB contains less than a certain number
0976  2935 ;       of CDRPs then the CDRP is inserted on the CSB free queue as
0976  2936 ;       a package with the RSPID and message buffer.  Otherwise, all
0976  2937 ;       three (CDRP, RSPID, and message buffer) are deallocated.
0976  2938 ;
0976  2939 ;       The RSPID must already have been recycled.  This is the case
0976  2940 ;       when this entry point is called by a continuous thread of
0976  2941 ;       execution that began as the result of receiving a message
0976  2942 ;       with a RSPID and that calls this routine to deallocate that
0976  2943 ;       message buffer and RSPID that were in the received message.
0976  2944 ;
0976  2945 ;       This requirement allows the lookup and recycling of the RSPID
0976  2946 ;       to be combined into one in-line piece of code.
0976  2947 ;
0976  2948 ; CALLING SEQUENCE:
0976  2949 ;
0976  2950 ;       BSBW    CNX$DEALL_WARMCDRP_CSB
0976  2951 ;
0976  2952 ;       IPL must be at IPL$_SYNCH
0976  2953 ;
0976  2954 ; INPUT PARAMETERS:
0976  2955 ;
0976  2956 ;       R2      Address of message buffer
0976  2957 ;       R3      CSB
0976  2958 ;       R5      Address of CDRP
0976  2959 ;
0976  2960 ; IMPLICIT INPUTS:
0976  2961 ;
0976  2962 ;       CDRP$L_RSPID contains the response id.
0976  2963 ;       The CDT and PDT addresses are in the CSB.
0976  2964 ;       NOTE:  The CDT address MUST be valid; i.e. the connection must NOT
0976  2965 ;              be broken.  One may NOT receive an input message on a
0976  2966 ;              connection, FORK or otherwise delay processing that message
0976  2967 ;              and then later call this routine with that message in hand
0976  2968 ;              (without at least verifying that the SAME connection is still
0976  2969 ;              valid).
0976  2970 ;
0976  2971 ; OUTPUT PARAMETERS:
0976  2972 ;
0976  2973 ;       None
0976  2974 ;
0976  2975 ; IMPLICIT OUTPUTS:
0976  2976 ;
0976  2977 ;       CDRP$L_MSG_BUF contains the message buffer address if the
0976  2978 ;       CDRP is not deallocated.
0976  2979 ;
0976  2980 ; SIDE EFFECTS:
0976  2981 ;
0976  2982 ;       R0 - R2 are destroyed
0976  2983 ;--
0976  2984
```

```
                        0976  2985 CNX$DEALL_WARMCDRP_CSB::
          42 A3   91    0976  2986          CMPB    CSB$B_WARMCDRPS(R3),-   ; Is list of warm CDRPs full?
             02         0979  2987                  #MAXWARMCDRPS
             12   18    097A  2988          BGEQ    30$                     ; Yes
                        097C  2989
                        097C  2990          ; The list of free CDRPs is not full.  Initialize some fields,
                        097C  2991          ; store the message buffer address in the CDRP,
                        097C  2992          ; and insert this one on the list.
                        097C  2993
                        097C  2994          ASSUME  CDRP$K_NORMAL EQ 0
                        097C  2995          ASSUME  CDRP$B_CNXSTATE EQ <CDRP$W_SENDSEQNM + 2>
                        097C  2996          ASSUME  CDRP$L_RETRSPID EQ <CDRP$W_SENDSEQNM + 4>
          54 A5   7C    097C  2997          CLRQ    CDRP$W_SENDSEQNM(R5)    ; Clear sequence number, return RSPID,
                        097F  2998                                          ;   and set normal state
    1C A5    52   D0    097F  2999          MOVL    R2,CDRP$L_MSG_BUF(R5)   ; Put message buffer address in CDRP
       0C A5   D4    0983  3000          CLRL    CDRP$L_FPC(R5)          ; Ensure fork thread can't resume
    28 B3    65   0E    0986  3001          INSQUE  (R5),@CSB$L_WARMCDRPQBL(R3) ; Insert CDRP on free queue
          42 A3   96    098A  3002          INCB    CSB$B_WARMCDRPS(R3)     ; Incr. count of warm CDRPs
                   05   098D  3003          RSB
                        098E  3004
                        098E  3005 30$:
                        098E  3006          ; List of warm CDRPs is full.  Deallocate message buffer,
                        098E  3007          ; response id. and CDRP.
                        098E  3008
                        098E  3009 DEALLOC_WARMCDRP:        ; Internal entry point
                        098E  3010
                        098E  3011          ; R2 is address of message buffer
                        098E  3012          ; R3 is CSB address
                        098E  3013          ;       CSB$L_CDT is CDT address
                        098E  3014          ;       CSB$L_PDT is PDT address
                        098E  3015          ; R5 is CDRP address
                        098E  3016          ;       CDRP$L_RSPID contains RSPID
                        098E  3017          ;
                        098E  3018          ; R0-R2 destroyed, R5 invalidated.
                        098E  3019
                        098E  3020          ASSUME  CSB$L_PDT  EQ  CSB$L_CDT+4
                        098E  3021
             54   DD    098E  3022          PUSHL   R4                      ; Save R4
             53   DD    0990  3023          PUSHL   R3                      ; Save CSB address
    53       0C A3   7D    0992  3024          MOVQ    CSB$L_CDT(R3),R3        ; Get address of CDT and PDT
                        0996  3025          DEALLOC_MSG_BUF_REG             ; Deallocate message buffer
                        0999  3026          DEALLOC_RSPID                   ; Deallocate RSPID
          50   55   D0    099F  3027          MOVL    R5,R0                   ; Move address of CDRP
00000000'GF    16   09A2  3028          JSB     G^EXE$DEANONPAGED       ; Deallocate CDRP
          53   8E   7D    09A8  3029          MOVQ    (SP)+, R3               ; Restore registers
                   05   09AB  3030          RSB
```

```
                        09AC  3032          .SBTTL  CNX$DEALL_MSG_BUF_CSB - Deallocate a message buffer using a CSB
                        09AC  3033 ;++
                        09AC  3034 ; FUNCTIONAL DESCRIPTION:
                        09AC  3035 ;
                        09AC  3036 ;       This routine deallocates the message buffer whose address is in R2.
                        09AC  3037 ;
                        09AC  3038 ; CALLING SEQUENCE:
                        09AC  3039 ;
                        09AC  3040 ;       BSBW    CNX$DEALL_MSG_BUF_CSB
                        09AC  3041 ;
                        09AC  3042 ;       IPL must be at IPL$_SCS (equals IPL$_SYNCH)
                        09AC  3043 ;
                        09AC  3044 ; INPUT PARAMETERS:
                        09AC  3045 ;
                        09AC  3046 ;       R2      Address of message buffer
                        09AC  3047 ;       R3      CSB
                        09AC  3048 ;
                        09AC  3049 ; IMPLICIT INPUTS:
                        09AC  3050 ;
                        09AC  3051 ;       The CDT and PDT addresses are in the CSB.
                        09AC  3052 ;       NOTE:  The CDT address MUST be valid; i.e. the connection must NOT
                        09AC  3053 ;              be broken.  One may NOT receive an input message on a
                        09AC  3054 ;              connection, FORK or otherwise delay processing that message
                        09AC  3055 ;              and then later call this routine with that message in hand
                        09AC  3056 ;              (without at least verifying that the SAME connection is still
                        09AC  3057 ;              valid).
                        09AC  3058 ;
                        09AC  3059 ; OUTPUT PARAMETERS:
                        09AC  3060 ;
                        09AC  3061 ;       R0      Status
                        09AC  3062 ;               SS$_NORMAL ==> deallocation successful
                        09AC  3063 ;
                        09AC  3064 ; IMPLICIT OUTPUTS:
                        09AC  3065 ;
                        09AC  3066 ;       R0 through R2 are destroyed; all other registers are preserved.
                        09AC  3067 ;
                        09AC  3068 ; SIDE EFFECTS:
                        09AC  3069 ;
                        09AC  3070 ;       The message buffer is deallocated.
                        09AC  3071 ;--
                        09AC  3072
                        09AC  3073 CNX$DEALL_MSG_BUF_CSB::
    7E   53   7D        09AC  3074          MOVQ    R3, -(SP)                   ; Save sensitive registers.
                        09AF  3075          ASSUME  CSB$L_PDT EQ <CSB$L_CDT + 4>
53   0C A3   7D        09AF  3076          MOVQ    CSB$L_CDT(R3),R3            ; Get CDT and PDT addresses.
                        09B3  3077          DEALLOC_MSG_BUF_REG                 ; Deallocate the message buffer.
                        09B6  3078
    53   8E   7D        09B6  3079          MOVQ    (SP)+, R3                   ; Restore registers.
    50   01   D0        09B9  3080          MOVL    #SS$_NORMAL,R0              ; Set success status.
                   05   09BC  3081          RSB
                        09BD  3082
                        09BD  3083
                        09BD  3084          .END
```

```
$$BASE                          = 00000000
$$BIGEST                        = 000005A2 R      02
$$DISPL                         = 00000004
$$FIRST                           0000059A R      02
$$GENSW                         = 00000001
$$HIGH                          = 00000003
$$LIMIT                         = 00000003
$$LOW                           = 00000000
$$MNSW                          = 00000001
$$MXSW                          = 00000001
ACK_MSG                           000005D2 R      02
BLD_BLKXFR_HDR                    000006C6 R      02
BLKXFR_RETRY                      000006D9 R      02
BLOCK_FAIL                        0000086A R      02
BLOCK_XFER                        0000065F R      02
BUG$_CNXMGRERR                    ******** X      02
CDRP$B_CD_TYPE                  = 0000000A
CDRP$B_CNXRMOD                  = 0000004A
CDRP$B_CNXSTATE                 = 00000056
CDRP$B_FIPL                     = 0000000B
CDRP$B_RMOD                     = FFFFFFAB
CDRP$K_CM_LENGTH                = 00000060
CDRP$K_NORMAL                   = 00000000
CDRP$K_PARTNER                  = 00000002
CDRP$K_PART_IDLE                = 00000003
CDRP$K_PART_MAP                 = 00000005
CDRP$K_REQUESTOR                = 00000001
CDRP$K_REQ_MAP                  = 00000004
CDRP$L_BCNT                     = FFFFFFD2
CDRP$L_CDT                      = 00000024
CDRP$L_CNXBCNT                  = 00000046
CDRP$L_CNXSVAPTE                = 00000040
CDRP$L_FPC                      = 0000000C
CDRP$L_FQFL                     = 00000000
CDRP$L_IOQFL                    = FFFFFFA0
CDRP$L_LBOFF                    = 00000030
CDRP$L_LBUFH_AD                 = 0000002C
CDRP$L_MSGBLD                   = 0000004C
CDRP$L_MSG_BUF                  = 0000001C
CDRP$L_RBOFF                    = 00000038
CDRP$L_RBUFH_AD                 = 00000034
CDRP$L_RETRSPID                 = 00000058
CDRP$L_RSPID                    = 00000020
CDRP$L_RWCPTR                   = 00000028
CDRP$L_SAVD_RTN                 = 00000018
CDRP$L_SAVEPC                   = 00000050
CDRP$L_SVAPTE                   = FFFFFFCC
CDRP$L_XCT_LEN                  = 0000003C
CDRP$W_BOFF                     = FFFFFFD0
CDRP$W_CDRPSIZE                 = 00000008
CDRP$W_CNXBOFF                  = 00000044
CDRP$W_SENDSEQNM                = 00000054
CDRP_MOST_WAIT                    00000302 R      02
CDT$C_AUXSTRUC                  = 0000005C
CHECK_RSPID                       000001D0 R      02
CJF$DISPATCH                      ******** X      02
CLEANUP_CDRP                      00000186 R      02
CLMBLK$L_RSPID                  = 0000000C
CLSMSG$B_FACILITY               = 00000008
CLSMSG$K_FAC_ACK                = 00000004
CLSMSG$K_FAC_BLK                = 00000007
CLSMSG$K_FAC_CJF                = 00000003
CLSMSG$K_FAC_CNX                = 00000001
CLSMSG$K_FAC_CSP                = 00000006
CLSMSG$K_FAC_LCK                = 00000002
CLSMSG$K_FAC_LKI                = 00000005
CLSMSG$K_MAXMSG                 = 0000006B
CLSMSG$L_REQR_BUFH              = 0000000C
CLSMSG$L_RSPID                  = 00000004
CLSMSG$W_ACKSEQ                 = 00000002
CLSMSG$W_SEQNUM                 = 00000000
CLU$GL_CLUSVEC                    ******** X    02
CLU$GW_MAXINDEX                   ******** X    02
CLUBTX$B_TYPE                   = 0000000A
CLUBTX$K_LENGTH                 = 00000030
CLUBTX$L_CDRP                   = 00000018
CLUBTX$L_CSID                   = 0000001C
CLUBTX$L_ERRADDR                = 00000020
CLUBTX$L_LBUFHNDL               = 0000000C
CLUBTX$L_MSGBLD                 = 0000002C
CLUBTX$L_SAVED_PC               = 00000028
CLUBTX$L_USER_BUF               = 00000024
CLUBTX$L_XQFL                   = 00000000
CLUBTX$S_LBUFHNDL               = 0000000C
CLUBTX$T_MSG_BUF                = 00000030
CLUBTX$W_SIZE                   = 00000008
CNX$ALLOC_CDRP                    00000923 RG     02
CNX$ALLOC_CDRP_ONLY               0000093C RG     02
CNX$ALLOC_WARMCDRP                000008E5 RG     02
CNX$ALLOC_WARMCDRP_CSB            000008FE RG     02
CNX$BLOCK_READ                    000007B5 RG     02
CNX$BLOCK_READ_IRP                00000796 RG     02
CNX$BLOCK_WRITE                   000007C1 RG     02
CNX$BLOCK_WRITE_IRP               0000079E RG     02
CNX$BLOCK_XFER                    0000060E RG     02
CNX$BLOCK_XFER_IRP                000005FF RG     02
CNX$DEALL_MSG_BUF_CSB             000009AC RG     02
CNX$DEALL_WARMCDRP_CSB            00000976 RG     02
CNX$DISC_PROTOCOL                 ********   X    02
CNX$DISPATCH                      ********   X    02
CNX$FAIL_MSG                      00000224 RG     02
CNX$INIT_CDRP                     00000918 RG     02
CNX$PARTNER_FINISH                000008AE RG     02
CNX$PARTNER_INIT_CSB              00000701 RG     02
CNX$PARTNER_RESPOND               000008B9 RG     02
CNX$POST_CLEANUP                  000000EF RG     02
CNX$PRE_CLEANUP                   00000000 RG     02
CNX$RCV_MSG                       000004BC RG     02
CNX$RCV_REJECT                    000005D5 RG     02
CNX$RESEND_MSGS                   00000264 RG     02
CNX$SEND_MNY_MSGS                 0000043B RG     02
CNX$SEND_MSG                      00000336 RG     02
CNX$SEND_MSG_CSB                  0000034F RG     02
CNX$SEND_MSG_RESP                 0000032D RG     02
```

```
CNX$SEND_MSG_RSPID              00000327 RG    02
CSB$B_REMACKLIM               = 00000033
CSB$B_STATE                   = 00000043
CSB$B_UNACKEDMSGS             = 00000032
CSB$B_WARMCDRPS               = 00000042
CSB$K_DISCONNECT              = 00000007
CSB$K_OPEN                    = 00000001
CSB$L_CDT                     = 0000000C
CSB$L_CSID                    = 0000004C
CSB$L_CURRCDRP                = 00000034
CSB$L_PARTNERQBL              = 0000005C
CSB$L_PARTNERQFL              = 00000058
CSB$L_PDT                     = 00000010
CSB$L_RESENDQBL               = 00000020
CSB$L_RESENDQFL               = 0000001C
CSB$L_SENTQBL                 = 00000018
CSB$L_SENTQFL                 = 00000014
CSB$L_STATUS                  = 00000060
CSB$L_WARMCDRPQBL             = 00000028
CSB$L_WARMCDRPQFL             = 00000024
CSB$V_LOCAL                   = 00000018
CSB$V_LONG_BREAK              = 00000000
CSB$V_REMOVED                 = 00000002
CSB$W_ACKRSEQNM               = 00000030
CSB$W_RCVDSEQNM               = 0000002E
CSB$W_SENDSEQNM               = 0000002C
CSP$DISPATCH                    ********    X   02
DEALLOC_WARMCDRP                0000098E R      02
DYN$C_CDRP                    = 00000039
DYN$C_CLU                     = 00000065
DYN$C_CLU_BTX                 = 00000004
EXE$ALONONPAGED                 ********    X   02
EXE$DEANONPAGED                 ********    X   02
EXE$FORK_WAIT                   ********    X   02
FAC_SIZES                       0000059A R      02
FLUSH_WARMCDRPS                 00000166 R      02
INV_CSID_NO_CLEANUP             00000970 R      02
IPL$_SCS                      = 00000008
IPL$_SYNCH                    = 00000008
IRP$B_RMOD                    = 0000000B
IRP$K_LENGTH                  = 000000C4
IRP$L_BCNT                    = 00000032
IRP$L_SVAPTE                  = 0000002C
IRP$W_BOFF                    = 00000030
LCK$DISPATCH                    ********    X   02
LKI$DISPATCH                    ********    X   02
MAXWARMCDRPS                  = 00000002
MAX_FACILITY                  = 00000008
MEMORY_RETRY                    00000612 R      02
MERGE_CDRP                      000001EC R      02
PDT$L_ALLOCMSG                = 00000014
PDT$L_DEALLOMSG               = 00000020
PDT$L_DEALRGMSG               = 00000024
PDT$L_MAP                     = 0000002C
PDT$L_RCLMSGBUF               = 00000048
PDT$L_REQDATA                 = 00000050
PDT$L_SENDDATA                = 00000054
PDT$L_SNDCNTMSG               = 00000060
PDT$L_UNMAP                   = 00000064
RD$C_LENGTH                   = 00000008
RD$L_CDRP                     = 00000000
RD$V_BUSY                     = 00000000
RD$W_SEQNUM                   = 00000006
RD$W_STATE                    = 00000004
RDT$L_MAXRDIDX                = FFFFFFF8
REQUEST_DATA                    000003D1 R      02
RESEND_MSG                      000003EF R      02
SCS$ALLOC_RSPID                 ********    X   02
SCS$DEALL_RSPID                 ********    X   02
SCS$FIND_RDTE                   ********    X   02
SCS$GL_RDT                      ********    X   02
SCS$LKP_RDTCDRP                 ********    X   02
SEND_ACK_MSG                    000005AD R      02
SEND_ALLOC                      0000036F R      02
SEND_CSID_ERROR                 000002D9 R      02
SEND_DATA                       000003C4 R      02
SEND_MSG_NOWAIT                 00000363 R      02
SEND_UNSEQ_MSG                  00000359 R      02
SS$_INSFMEM                   = 00000124
SS$_NODELEAVE                 = 0000223C
SS$_NORMAL                    = 00000001
SS$_NOSUCHNODE                = 0000028C
```

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
$$$100	000009BD (2493.)	02 (2.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC LONG

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	32	00:00:00.05	00:00:02.12
Command processing	123	00:00:00.44	00:00:02.35
Pass 1	560	00:00:17.09	00:01:07.21
Symbol table sort	0	00:00:01.94	00:00:06.69
Pass 2	428	00:00:05.31	00:00:20.46
Symbol table output	5	00:00:00.14	00:00:00.64
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	1152	00:00:24.99	00:01:39.50

The working set limit was 1950 pages.
144309 bytes (282 pages) of virtual memory were used to buffer the intermediate code.
There were 110 pages of symbol table space allocated to hold 1776 non-local and 122 local symbols.
3084 source lines were read in Pass 1, producing 23 object records in Pass 2.
44 pages of virtual memory were used to define 42 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA28:[SYSLOA.OBJ]CLUSTER.MLB;1	4
_255DUA28:[SYS.OBJ]LIB.MLB;1	25
_255DUA28:[SYSLIB]STARLET.MLB;2	5
TOTALS (all libraries)	34

1920 GETS were required to define 34 macros.

There were no errors, warnings or information messages.

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

ADPSUB780
LIS

ACKMSG
LIS

MCF790
SDL

MCDEF
MDL

ADPERR750
LIS

ADPSUB730
LIS

CSPDEF
SDL

CLUMBX
SDL

ADPERR780
LIS

ADPSUB750
LIS

CLUSTRMAC
MAR

CLUSTER
SDL