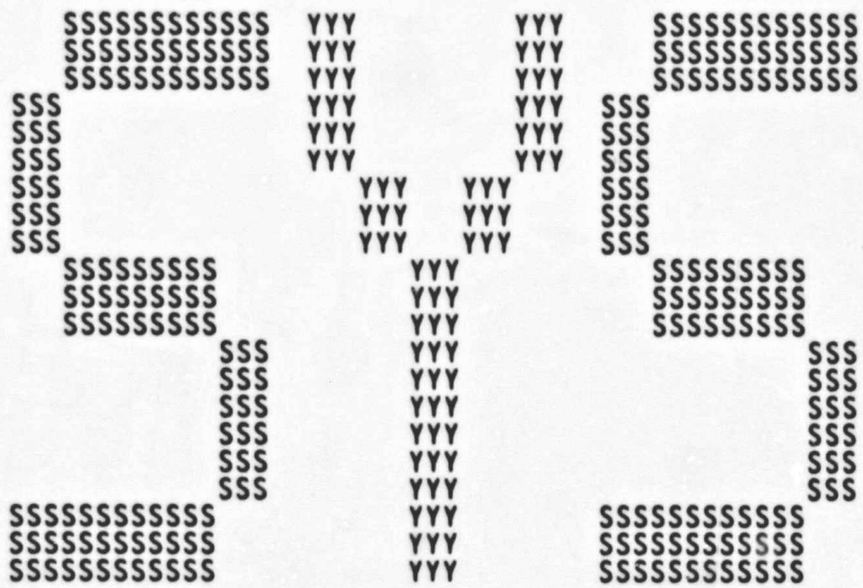
SSSSSSSSSSS YYY YYY SSSSSSSSSSS
SSSSSSSSSSS YYY YYY SSSSSSSSSSS
SSSSSSSSSSS YYY YYY SSSSSSSSSSS
SSS YYY YYY SSS
SSS YYY YYY SSS
SSS YYY YYY SSS
SSS YYY YYY SSS
SSS YYY YYY SSS
SSS YYY YYY SSS
SSSSSSSSS YYY SSSSSSSSS
SSSSSSSSS YYY SSSSSSSSS
SSSSSSSSS YYY SSSSSSSSS
SSS YYY SSS
SSS YYY SSS
SSS YYY SSS
SSS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

```
IIIIII     000000     SSSSSSSS  UU     UU  BBBBBBBB    NN     NN  PPPPPPPP     AAAAAA     GGGGGGGG
IIIIII     000000     SSSSSSSS  UU     UU  BBBBBBBB    NN     NN  PPPPPPPP     AAAAAA     GGGGGGGG
  II     OO     OO  SS        UU     UU  BB     BB   NN     NN  PP     PP   AA     AA   GG
  II     OO     OO  SS        UU     UU  BB     BB   NN     NN  PP     PP   AA     AA   GG
  II     OO     OO  SS        UU     UU  BB     BB   NNNN   NN  PP     PP   AA     AA   GG
  II     OO     OO  SS        UU     UU  BB     BB   NNNN   NN  PP     PP   AA     AA   GG
  II     OO     OO    SSSSSS  UU     UU  BBBBBBBB   NN  NN  NN  PPPPPPPP   AA     AA   GG
  II     OO     OO    SSSSSS  UU     UU  BBBBBBBB   NN  NN  NN  PPPPPPPP   AA     AA   GG
  II     OO     OO          SS UU     UU  BB     BB   NN    NNNN  PP         AAAAAAAAAA   GG  GGGGGG
  II     OO     OO          SS UU     UU  BB     BB   NN    NNNN  PP         AAAAAAAAAA   GG  GGGGGG
  II     OO     OO          SS UU     UU  BB     BB   NN     NN   PP         AA     AA   GG     GG     ....
  II     OO     OO          SS UU     UU  BB     BB   NN     NN   PP         AA     AA   GG     GG     ....
IIIIII     000000     SSSSSSSS  UUUUUUUUUU  BBBBBBBB    NN     NN  PP         AA     AA     GGGGGG       ....
IIIIII     000000     SSSSSSSS   UUUUUUUUU  BBBBBBBB    NN     NN  PP         AA     AA     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  IOSUBNPAG - NONPAGED I/O RELATED SUBROUTINES
0000     2          .IDENT  'V04-000'
0000     3  ;****************************************************************************
0000     4  ;*                                                                          *
0000     5  ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
0000     6  ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000     7  ;*  ALL RIGHTS RESERVED.                                                    *
0000     8  ;*                                                                          *
0000     9  ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
0000    10  ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000    11  ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
0000    12  ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
0000    13  ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
0000    14  ;*  TRANSFERRED.                                                            *
0000    15  ;*                                                                          *
0000    16  ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
0000    17  ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
0000    18  ;*  CORPORATION.                                                            *
0000    19  ;*                                                                          *
0000    20  ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
0000    21  ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
0000    22  ;*                                                                          *
0000    23  ;*                                                                          *
0000    24  ;****************************************************************************
0000    25  ;
0000    26  ; D. N. CUTLER 13-JUN-76
0000    27  ;
0000    28  ;
0000    29  ; NONPAGED I/O RELATED SUBROUTINES
0000    30  ;
0000    31  ; MODIFIED BY:
0000    32  ;
0000    33  ;       V03-038 WMC0004         Wayne Cardoza           23-Aug-1984
0000    34  ;               Add routine for emergency message to console.
0000    35  ;
0000    36  ;       V03-037 WMC0003         Wayne Cardoza           14-Aug-1984
0000    37  ;               Fix ROW0409 to restore the correct register.
0000    38  ;
0000    39  ;       V03-036 ACG0442         Andrew C. Goldstein,    7-Aug-1984  17:52
0000    40  ;               Save R8 in IOC$LAST_CHAN; fix order of tests in IOC$TESTUNIT
0000    41  ;               for correct allocation and mount checks. Fix handling of
0000    42  ;               lock value block on device lock in IOC$TESTUNIT.
0000    43  ;
0000    44  ;       V03-035 ROW0409         Ralph O. Weber          6-AUG-1984
0000    45  ;               Fix release map registers processing of requests waiting for
0000    46  ;               map registers.  Restore saved fork registers -- including the
0000    47  ;               PDT address -- before the calling IOC$ALOMAPUDA at
0000    48  ;               REALLOC_CD_MAPREGS.
0000    49  ;
0000    50  ;       V03-034 TCM0006         Trudy C. Matthews       20-Jul-1984
0000    51  ;               Add routine IOC$THREADCRB.
0000    52  ;
0000    53  ;       V03-033 WMC0002         Wayne Cardoza           03-May-1984
0000    54  ;               Add support for MNTVERPND bit.
0000    55  ;
0000    56  ;       V03-032 RAS0300         Ron Schaefer            2-May-1984
0000    57  ;               Change IOC$CVT_DEVNAM to only prefix cluster node names if
```

```
0000     58 ;                  the DEV$V_NNM device characteristic is set in UCB$L_DEVCHAR2.
0000     59 ;                  Add additional itemcode (4) to IOC$CVT_DEVNAM to provide
0000     60 ;                  the device name string sans unit number.
0000     61 ;
0000     62 ;          V03-031 TMK0001         Todd M. Katz            23-Apr-1984
0000     63 ;                  Remove the $LOGDEF data definitions.
0000     64 ;
0000     65 ;          V03-030 RLRPDTADP       Robert L. Rappaport     9-Apr-1984
0000     66 ;                  Modify entrypoints used for allocating and deallocating
0000     67 ;                  Buffered Data Paths and UNIBUS Map Registers for UQPORTS (UDA),
0000     68 ;                  to pickup pointer for ADP from PDT$L_ADP(R4).
0000     69 ;
0000     70 ;          V03-029 ACG0414         Andrew C. Goldstein,    30-Mar-1984  15:49
0000     71 ;                  Minor parse and searching fixes in IOC$SEARCH...
0000     72 ;                  add IOC$V_ALLOC to force allocation
0000     73 ;
0000     74 ;          V03-028 ACG0406         Andrew C. Goldstein,    16-Mar-1984  15:42
0000     75 ;                  Fix bugs in searching for allocation class
0000     76 ;
0000     77 ;          V03-027 ACG0399         Andrew C. Goldstein,    24-Feb-1984  22:28
0000     78 ;                  Add IOC$LAST_CHAN subroutine, and move in internal I/O
0000     79 ;                  database parse and search routines, so they can be called
0000     80 ;                  by IPC.
0000     81 ;
0000     82 ;          V03-026 RLRMAPSP        Robert L. Rappaport     15-Feb-1984
0000     83 ;                  Correct bug in BEQL destination in IOC$ALOUBAMAPSP that is
0000     84 ;                  only triggered if the range specified, coincides with the
0000     85 ;                  exact end of an extent of map registers.
0000     86 ;
0000     87 ;          V03-025 ROW0292         Ralph O. Weber          4-FEB-1984
0000     88 ;                  Fix branch displacements broken by movement of EXE$MOUNTVER to
0000     89 ;                  SYSLOAxxx.
0000     90 ;
0000     91 ;          V03-024 KPL0001         Peter Lieberwirth       7-Nov-1983
0000     92 ;                  Add paths for new processors to CPUDISP invocation.
0000     93 ;
0000     94 ;          V03-023 ROW0244         Ralph O. Weber          17-OCT-1983
0000     95 ;                  Change the IOC$CVT_DEVNAM name string formation rules to
0000     96 ;                  eliminate _$1$TTA0: and other allocation class based names
0000     97 ;                  for devices which can never be dual pathed.  See routine
0000     98 ;                  comments for details of current operation mode.
0000     99 ;
0000    100 ;          V03-022 ROW0239         Ralph O. Weber          11-OCT-1983
0000    101 ;                  Fix IOC$CVT_DEVNAM to not insert node name or trailing dollar
0000    102 ;                  sign when node name is null.  Also correct comments describing
0000    103 ;                  the R4 argument to IOC$CVT_DEVNAM.
0000    104 ;
0000    105 ;          V03-021 ROW0234         Ralph O. Weber          5-OCT-1983
0000    106 ;                  Change IOC$CVT_DEVNAM to produce $allocation-class$device
0000    107 ;                  strings completely in ASCII, when allocation class output is
0000    108 ;                  requested.  In the process rip up the whole thing because that
0000    109 ;                  was the only way to get something that worked and didn't
0000    110 ;                  occupy all non-page memory
0000    111 ;
0000    112 ;          V03-020 TCM0005         Trudy C. Matthews       5-OCT-1983
0000    113 ;                  Add IOC$SCAN_IODB_2P which is functionally the same as
0000    114 ;                  IOC$SCAN_IOCB except that both primary and secondary paths to
```

```
0000   115 ;                a device are scanned.
0000   116 ;
0000   117 ;        V03-019 KDM0084         Kathleen D. Morse       26-Sep-1983
0000   118 ;                Added MicroVAX I support to CPUDISP macros.
0000   119 ;
0000   120 ;        V03-018 ROW0221         Ralph O. Weber          8-SEP-1983
0000   121 ;                Change IOC$UNITINIT to look for a unit initialization routine
0000   122 ;                in the DDT before looking in the CRB.  See the note in the
0000   123 ;                routine's header for details.
0000   124 ;
0000   125 ;        V03-017 ROW0203         Ralph O. Weber          5-AUG-1983
0000   126 ;                Add two new routines IOC$CTRLINIT and IOC$UNITINIT.  These are
0000   127 ;                the proscribed mechanism for calling device driver controller
0000   128 ;                and unit initialization routines.  These routines correctly
0000   129 ;                setup for, locate, and call the appropriate driver routines.
0000   130 ;
0000   131 ;        V03-016 TCM0004         Trudy C. Matthews       26-Jul-1983
0000   132 ;                Change IOC$CVT_DEVNAM to return the <allocation_class>+
0000   133 ;                <devnam> form of device name if R4 > 0.
0000   134 ;
0000   135 ;        V03-015 RLRBYTEOFF      Robert L. Rappaport     27-Jun-1983
0000   136 ;                Correct error in IOC$REQDATAPUDA.  Error is that this
0000   137 ;                routine has operated in a NOWAIT mode, that is, if no
0000   138 ;                Buffered Datapath was available, we just used the
0000   139 ;                Direct Datapath.  Unfortunately, this doesn't work on
0000   140 ;                780's and 790's if the user buffer is located at an
0000   141 ;                odd byte address since Byte Offset doesn't work on the
0000   142 ;                Direct Datapath for the UNIBUS Adapters on these
0000   143 ;                processors.
0000   144 ;
0000   145 ;        V03-014 LMPBUILD        L. Mark Pilant,         26-Jun-1983  23:11
0000   146 ;                Change references from TTY$K_WB_HDRLEN to TTY$K_WB_LENGTH.
0000   147 ;
0000   148 ;        V03-013 TCM0003         Trudy C. Matthews       17-Jun-1983
0000   149 ;                Change the way cluster-style device names are conditionally
0000   150 ;                returned, such that cluster-style names are returned for
0000   151 ;                local disk devices if the system is participating in a
0000   152 ;                cluster (routine IOC$CVT_DEVNAM).
0000   153 ;
0000   154 ;        V03-012 TCM0002         Trudy C. Matthews       09-Jun-1983
0000   155 ;                Fix bug in TCM0001.
0000   156 ;
0000   157 ;        V03-011 TCM0001         Trudy C. Matthews       21-Apr-1983
0000   158 ;                Add new parameter to IOC$CVT_DEVNAM that allows caller
0000   159 ;                to specify whether he wants the node name returned for
0000   160 ;                local devices or not.
0000   161 ;
0000   162 ;        V03-010 ROW0188         Ralph O. Weber          30-APR-1983
0000   163 ;                Fix broken branches to PMS$ routines.
0000   164 ;
0000   165 ;        V03-009 KTA3022         Kerbey T. Altmann       29-Dec-1982
0000   166 ;                Enhance KTA3018. Add new routine to scan the IO
0000   167 ;                data base and return the blocks.
0000   168 ;
0000   169 ;        V03-008 ROW0140         Ralph O. Weber          18-NOV-1982
0000   170 ;                Cause IOC$DALOCUBAMAP to give non-fatal INCONSTATE,
0000   171 ;                "Inconsistant UBA data base" bugcheck if number of map
```

```
0000   172 ;               registers to deallocate is zero.
0000   173 ;
0000   174 ;       V03-007 MLJ0101         Martin L. Jack          11-Nov-1982
0000   175 ;               Add $SBDEF.
0000   176 ;
0000   177 ;       V03-006 KTA3018         Kerbey T. Altmann       01-Nov-1982
0000   178 ;               Modify CVT_DEVNAME for new IO database.
0000   179 ;
0000   180 ;       V03-005 ROW0130         Ralph O. Weber           5-OCT-1982
0000   181 ;               Remove IOC$DELMBX whose functionality is replaced by new
0000   182 ;               routines in module UCBCREDEL.
0000   183 ;
0000   184 ;       V03-004 KDM0002         Kathleen D. Morse       28-Jun-1982
0000   185 ;               Added $DCDEF.
0000   186 ;
0000   187 ;       V03-003 RLR0003         Robert L. Rappaport     1-June-1982
0000   188 ;               Correct errors in UNIBUS map register allocation and
0000   189 ;               deallocation that occur when the number of active
0000   190 ;               descriptors is zero.  Errors were in IOC$ALOUBAMAPSP
0000   191 ;               (allocation error), IOC$ALOUBAPRM (allocation error),
0000   192 ;               and IOC$DALOCUBAMAP (deallocation error).  The error
0000   193 ;               in IOC$DALOCUBAMAP is corrected in a patch to V3.1.
0000   194 ;
0000   195 ;       V03-002 RLR0002         Robert L. Rappaport     22-May-1982
0000   196 ;               Remove IOC$REQMAPREGN and all comments that reference it.
0000   197 ;
0000   198 ;       V03-001 RLR0001 Robert L. Rappaport     22-May-1982
0000   199 ;               Correct error in UNIBUS map register allocation that
0000   200 ;               doubly allocated registers when the number of active
0000   201 ;               descriptors was zero.
0000   202 ;               This bug corrected in patch to V3.1.
0000   203 ;
```

```
0000    205 ;
0000    206 ;
0000    207 ; MACRO LIBRARY CALLS
0000    208 ;
0000    209
0000    210         $ADPDEF                 ;DEFINE ADP OFFSETS
0000    211         $CADEF                  ;DEFINE CONDITIONAL ASSEMBLY PARAMETERS
0000    212         $CANDEF                 ;DEFINE CANCEL I/O REASON CODES
0000    213         $CDRPDEF                ;DEFINE CLASS DRIVER I/O REQUEST PACKET
0000    214         $CRBDEF                 ;DEFINE CRB OFFSETS
0000    215         $DCDEF                  ;DEFINE DEVICE CLASSES
0000    216         $DDBDEF                 ;DEFINE DDB OFFSETS
0000    217         $DDTDEF                 ;DEFINE DDT OFFSETS
0000    218         $DEVDEF                 ;DEFINE DEVICE CHARACTERISTICS FLAGS
0000    219         $DYNDEF                 ;DEFINE DYNAMIC POOL BLOCK TYPES
0000    220         $EMBDEF                 ;DEFINE EMB OFFSETS
0000    221         $IDBDEF                 ;DEFINE IDB OFFSETS
0000    222         $IOCDEF                 ;DEFINE IOC$SEARCHxxx FLAGS
0000    223         $IPLDEF                 ;DEFINE INTERRUPT PRIORITY LEVELS
0000    224         $IRPDEF                 ;DEFINE IRP OFFSETS
0000    225         $JIBDEF                 ;DEFINE JIB OFFSETS
0000    226         $LCKDEF                 ;DEFINE LOCK MANAGER SYMBOLS
0000    227         $MSCPDEF                ;DEFINE MSCP STRUCTURES
0000    228         $PCBDEF                 ;DEFINE PCB OFFSETS
0000    229         $PDTDEF                 ;Define PDT offsets
0000    230         $PRDEF                  ;DEFINE PROCESSOR REGISTERS
0000    231         $PRVDEF                 ;DEFINE PRIVILEGE BITS
0000    232         $SBDEF                  ; Define system block offsets
0000    233         $SSDEF                  ;DEFINE SYSTEM STATUS CODES
0000    234         $TTYDEF                 ;DEFINE TERMINAL WRITE PACKET OFFSETS
0000    235         $UBMDDEF                ;Define UNIBUS Map Descriptor structure
0000    236         $UCBDEF                 ;DEFINE UCB OFFSETS
0000    237         $VECDEF                 ;DEFINE CRB VECTOR OFFSETS
```

```
                                                E 3
                     0000   239          .SBTTL  CANCEL I/O ON CHANNEL
                     0000   240 ;+
                     0000   241 ; IOC$CANCELIO - CANCEL I/O ON CHANNEL
                     0000   242 ;
                     0000   243 ; THIS ROUTINE IS A DEVICE INDEPENDENT CANCEL I/O ROUTINE THAT CONDITIONALLY
                     0000   244 ; MARKS THE UCB SUCH THAT THE CURRENT I/O REQUEST WILL BE CANCELED IF CONDITIONS
                     0000   245 ; WARRANT SUCH A ACTION.
                     0000   246 ;
                     0000   247 ; INPUTS:
                     0000   248 ;
                     0000   249 ;       R2 = NEGATIVE OF THE CHANNEL NUMBER.
                     0000   250 ;       R3 = CURRENT IO PACKET.
                     0000   251 ;       R4 = PCB ADDRESS.
                     0000   252 ;       R5 = UCB ADDRESS.
                     0000   253 ;
                     0000   254 ; OUTPUTS:
                     0000   255 ;
                     0000   256 ;       IF THE DEVICE IS BUSY, THE REQUEST IS FOR THE CURRENT PROCESS, AND
                     0000   257 ;       THE I/O WAS ISSUED FROM THE DESIGNATED CHANNEL, THEN THE CANCEL I/O
                     0000   258 ;       BIT IS SET IN THE CORRESPONDING UCB.
                     0000   259 ;
                     0000   260 ;       R2, R3, R4, AND R5 ARE PRESERVED ACROSS CALL.
                     0000   261 ;-
                     0000   262
                00000000    263          .PSECT  WIONONPAGED
                     0000   264 IOC$CANCELIO::                               ;CANCEL I/O ON CHANNEL
11 64 A5    08   E1  0000   265          BBC     #UCB$V_BSY,UCB$W_STS(R5),10$ ;IF CLR, DEVICE NOT BUSY
60 A4    0C A3   D1  0005   266          CMPL    IRP$L_PID(R3),PCB$L_PID(R4) ;PROCESS ID MATCH?
             0A  12  000A   267          BNEQ    10$                         ;IF NEQ NO
   28 A3     52  B1  000C   268          CMPW    R2,IRP$W_CHAN(R3)           ;CHANNEL NUMBER MATCH
             04  12  0010   269          BNEQ    10$                         ;IF NEQ NO
   64 A5     08  A8  0012   270          BISW    #UCB$M_CANCEL,UCB$W_STS(R5) ;SET CANCEL PENDING
                   05  0016   271 10$:    RSB                                 ;
```

```
                     0017   273          .SBTTL  Handle Last Channel Deassign
                     0017   274
                     0017   275 ;+
                     0017   276 ; IOC$LAST_CHAN - Last Channel Deassign Specific
                     0017   277 ; IOC$LAST_CHAN_AMBX - Last Assoc. MBX Channel Deassign Specific
                     0017   278 ;
                     0017   279 ; Functional Description:
                     0017   280 ;
                     0017   281 ;       Common functions done on last channel deassignment are handled.  The
                     0017   282 ;       driver's cancel I/O routine is called with an appropriate reason code
                     0017   283 ;       (CAN$C_DASSGN for regular deassign, or CAN$C_AMBXDGN for associated
                     0017   284 ;       mailboxes).  If after the cancel routine finished UCB$V_DELETEUCB is
                     0017   285 ;       set, the UCB is credited and deleted.
                     0017   286 ;
                     0017   287 ; Inputs:
                     0017   288 ;
                     0017   289 ;       R5      UCB address
                     0017   290 ;       R2      Channel index (LAST_CHAN only)
                     0017   291 ;
                     0017   292 ; Outputs:
                     0017   293 ;
                     0017   294 ;       R0 thru R3 destroyed.
                     0017   295 ;       If appropriate, UCB is deallocated.
                     0017   296 ;-
                     0017   297
                     0017   298          .ENABLE LSB
                     0017   299
                     0017   300
                     0017   301 IOC$LAST_CHAN_AMBX::
                58   DD  0017   302          PUSHL   R8                      ; Save R8
                52   7C  0019   303          CLRQ    R2                      ; Clear unused cancel inputs.
             58 02   9A  001B   304          MOVZBL  #CAN$C_AMBXDGN, R8      ; Set cancel reason code.
                09   11  001E   305          BRB     10$
                     0020   306
                     0020   307 IOC$LAST_CHAN::
                58   DD  0020   308          PUSHL   R8                      ; Save R8
          53    58 A5   D0  0022   309          MOVL    UCB$L_IRP(R5), R3       ; Get active packet address.
             58 01   9A  0026   310          MOVZBL  #CAN$C_DASSGN, R8       ; Set cancel reason code.
                     0029   311
          50  0088 C5   D0  0029   312 10$:    MOVL    UCB$L_DDT(R5), R0       ; Get DDT address.
                     002E   313          SETIPL  UCB$B_FIPL(R5)          ; Raise to fork IPL.
             0C B0   16  0032   314          JSB     @DDT$C_CANCEL(R0)       ; Call driver's cancel I/O routine.
                     0035   315          SETIPL  #IPL$_ASTDEL            ; Lower IPL.
       1A 38 A5   17   E0  0038   316          BBS     #DEV$V_ALL, -           ; Branch if still allocated
                     003D   317                  UCB$L_DEVCHAR(R5),30$
38 A5  00100004 8F   D3  003D   318          BITL    #DEV$M_TRM!DEV$M_MBX, - ; Is this a terminal, remote terminal
                     0045   319                  UCB$L_DEVCHAR(R5)       ; or mailbox?
                05   13  0045   320          BEQL    20$                     ; Branch if not.
       00 38 A5   07   E4  0047   321          BBSC    #DEV$V_OPR, -           ; Else, clear OPR bit.
                     004C   322                  UCB$L_DEVCHAR(R5), 20$  ; This is an implicit operator disable.
                10   E1  004C   323 20$:     BBC     #UCB$V_DELETEUCB,-      ; Branch if UCB not to be deleted.
             06 64 A5        004E   324                  UCB$L_STS(R5), 30$
                 FFAC'   30  0051   325          BSBW    IOC$CREDIT_UCB          ; Else credit UCB quotas,
                 FFA9'   30  0054   326          BSBW    IOC$DELETE_UCB          ; and delete the UCB.
                   58 8ED0   0057   327 30$:     POPL    R8                      ; Restore R8
                      05     005A   328          RSB
                     005B   329
```

```
005B   330          .DISABLE LSB
```

```
                                                H 3
                     005B   332          .SBTTL  FILL DIAGNOSTIC BUFFER
                     005B   333 ;+
                     005B   334 ; IOC$DIAGBUFILL - FILL DIAGNOSTIC BUFFER
                     005B   335 ;
                     005B   336 ; THIS ROUTINE IS CALLED AT THE END OF AN I/O OPERATION, BUT BEFORE RELEASING
                     005B   337 ; THE I/O CHANNEL, TO FILL THE FINAL DEVICE PARAMETERS INTO AN INTERNAL DIAG-
                     005B   338 ; NOSTIC BUFFER IF ONE IS SPECIFIED.
                     005B   339 ;
                     005B   340 ; INPUTS:
                     005B   341 ;
                     005B   342 ;       R4 = ADDRESS OF DEVICE CSR REGISTER.
                     005B   343 ;       R5 = DEVICE UNIT UCB ADDRESS.
                     005B   344 ;
                     005B   345 ; OUTPUTS:
                     005B   346 ;
                     005B   347 ;       IF A DIAGNOSTIC BUFFER WAS SPECIFIED IN THE ORIGINAL REQUEST, THEN
                     005B   348 ;       THE COMPLETION TIME, FINAL ERROR COUNTERS, AND DEVICE REGISTERS ARE
                     005B   349 ;       FILLED INTO THE DIAGNOSTIC BUFFER.
                     005B   350 ;-
                     005B   351
                     005B   352 IOC$DIAGBUFILL::                             ;FILL DIAGNOSTIC BUFFER
     53    58 A5    D0   005B   353          MOVL    UCB$L_IRP(R5),R3         ;GET ADDRESS OF I/O PACKET
  1B 2A A3    07    E1   005F   354          BBC     #IRP$V_DIAGBUF,IRP$W_STS(R3),10$ ;IF CLR, NO DIAGNOSTIC BUFFER
     50    4C B3    D0   0064   355          MOVL    @IRP$L_DIAGBUF(R3),R0    ;GET ADDRESS OF INTERNAL BUFFER DATA AREA
     50    08 C0    C0   0068   356          ADDL    #8,R0                    ;POINT PAST START TIME
80   00000000'EF    7D   006B   357          MOVQ    EXE$GQ_SYSTIME,(R0)+     ;INSERT COMPLETION TIME
   80    0080 C5    3C   0072   358          MOVZWL  UCB$B_ERTCNT(R5),(R0)+   ;INSERT FINAL ERROR COUNTERS
   52    0088 C5    D0   0077   359          MOVL    UCB$L_DDT(R5),R2         ;GET ADDRESS OF DDT
         10 B2      16   007C   360          JSB     @DDT$L_REGDUMP(R2)       ;CALL DEVICE SPECIFIC REGISTER DUMP ROUTINE
                    05   007F   361 10$:     RSB                              ;
```

```
                                                I 3
                     0080   363          .SBTTL  RELEASE I/O CHANNEL
                     0080   364 ;+
                     0080   365 ; IOC$RELCHAN - RELEASE ALL I/O CHANNELS
                     0080   366 ; IOC$RELSCHAN - RELEASE SECONDARY I/O CHANNEL
                     0080   367 ;
                     0080   368 ; THIS ROUTINE IS CALLED AT THE END OF AN I/O OPERATION TO RELEASE ALL
                     0080   369 ; CHANNELS THE I/O WAS BEING PERFORMED ON.
                     0080   370 ;
                     0080   371 ; INPUTS:
                     0080   372 ;
                     0080   373 ;       R5 = UCB ADDRESS OF DEVICE UNIT.
                     0080   374 ;
                     0080   375 ; OUTPUTS:
                     0080   376 ;
                     0080   377 ;       THE CHANNELS ARE RELEASED AND AN ATTEMPT IS MADE TO REMOVE THE NEXT
                     0080   378 ;       WAITING DRIVER PROCESS FROM EACH CHANNEL QUEUE. IF A DRIVER PROCESS
                     0080   379 ;       IS WAITING, THEN THE CHANNEL IS ASSIGNED TO THAT DRIVER PROCESS AND
                     0080   380 ;       IT IS CALLED VIA A JSB TO ITS CHANNEL WAIT RETURN ADDRESS. WHEN THE
                     0080   381 ;       CALLED DRIVER PROCESS RETURNS, A RETURN IS MADE TO THE DRIVER PROCESS
                     0080   382 ;       THAT RELEASED THE CHANNEL. IF THERE IS NO DRIVER PROCESS WAITING FOR
                     0080   383 ;       THE CHANNEL, THEN THE CHANNEL STATUS IS SET TO IDLE.
                     0080   384 ;
                     0080   385 ;       R3 AND R4 ARE PRESERVED ACROSS CALL.
                     0080   386 ;-
                     0080   387
                     0080   388          .ENABL  LSB
                     0080   389 IOC$RELSCHAN::                           ;RELEASE SECONDARY I/O CHANNEL
 50   24 A5   D0     0080   390          MOVL    UCB$L_CRB(R5),R0        ;GET ADDRESS OF PRIMARY CRB
 50   20 A0   D0     0084   391          MOVL    CRB$L_LINK(R0),R0       ;GET ADDRESS OF SECONARY CRB
        10   11     0088   392          BRB     20$                     ;
                     008A   393 IOC$RELCHAN::                            ;RELEASE I/O CHANNEL
 50   24 A5   D0     008A   394          MOVL    UCB$L_CRB(R5),R0        ;GET ADDRESS OF PRIMARY CRB
 50   20 A0   D0     008E   395          MOVL    CRB$L_LINK(R0),R0       ;GET ADDRESS OF SECONDARY CRB
        02   13     0092   396          BEQL    10$                     ;IF EQL NONE
        04   10     0094   397          BSBB    20$                     ;RELEASE SECONDARY CHANNEL
 50   24 A5   D0     0096   398 10$:     MOVL    UCB$L_CRB(R5),R0        ;GET ADDRESS OF PRIMARY CRB
25 0E A0      00     E1     009A   399 20$:     BBC     #CRB$V_BSY,CRB$B_MASK(R0),30$ ;IF CLR, THEN CHANNEL NOT BUSY
 51   2C A0   D0     009F   400          MOVL    CRB$L_INTD+VEC$L_IDB(R0),R1 ;GET ADDRESS OF IDB
 04 A1   55     D1     00A3   401          CMPL    R5,IDB$L_OWNER(R1)      ;DRIVER PROCESS OWN CHANNEL?
        1B   12     00A7   402          BNEQ    30$                     ;IF NEQ NO
 52   00 B0   0F     00A9   403          REMQUE  @CRB$L_WQFL(R0),R2      ;GET ADDRESS OF NEXT DRIVER FORK BLOCK
        16   1D     00AD   404          BVS     40$                     ;IF VS NO DRIVER PROCESS WAITING
        38   BB     00AF   405          PUSHR   #^M<R3,R4,R5>           ;SAVE CONTEXT OF CURRENT DRIVER PROCESS
 55     52   D0     00B1   406          MOVL    R2,R5                   ;COPY ADDRESS OF DRIVER PROCESS FORK BLOCK
 53   10 A5   D0     00B4   407          MOVL    UCB$L_FR3(R5),R3        ;LOAD WAITING DRIVER PROCESS CONTEXT
 54     61   D0     00B8   408          MOVL    IDB$L_CSR(R1),R4        ;SET ASSIGNED CHANNEL CSR ADDRESS
 04 A1   55     D0     00BB   409          MOVL    R5,IDB$L_OWNER(R1)      ;SET ADDRESS OF OWNER PROCESS UCB
    0C B5   16     00BF   410          JSB     @UCB$L_FPC(R5)          ;CALL DRIVER AT CHANNEL WAIT RETURN ADDRESS
        38   BA     00C2   411          POPR    #^M<R3,R4,R5>           ;RESTORE PREVIOUS DRIVER PROCESS CONTEXT
        05   00C4   412 30$:     RSB                                     ;
    04 A1   D4     00C5   413 40$:     CLRL    IDB$L_OWNER(R1)         ;CLEAR OWNER UNIT UCB ADDRESS
0E A0   01     8A     00C8   414          BICB    #CRB$M_BSY,CRB$B_MASK(R0) ;CLEAR CHANNEL BUSY
        05   00CC   415          RSB                                     ;
                     00CD   416          .DSABL  LSB
```

```
                                                J 3
00CD   418          .SBTTL  REQUEST I/O CHANNEL
00CD   419 ;+
00CD   420 ; IOC$REQPCHANH - REQUEST PRIMARY I/O CHANNEL HIGH PRIORITY
00CD   421 ; IOC$REQSCHANH - REQUEST SECONDARY I/O CHANNEL HIGH PRIORITY
00CD   422 ; IOC$REQPCHANL - REQUEST PRIMARY I/O CHANNEL LOW PRIORITY
00CD   423 ; IOC$REQSCHANL - REQUEST SECONDARY I/O CHANNEL LOW PRIORITY
00CD   424 ;
00CD   425 ; THESE ROUTINES ARE CALLED TO REQUEST AN I/O CHANNEL TO PERFORM AN I/O
00CD   426 ; OPERATION ON.
00CD   427 ;
00CD   428 ; INPUTS:
00CD   429 ;
00CD   430 ;       R5 = UCB ADDRESS OF DEVICE UNIT.
00CD   431 ;       04(SP) = RETURN ADDRESS OF CALLER'S CALLER.
00CD   432 ;
00CD   433 ; OUTPUTS:
00CD   434 ;
00CD   435 ;       IF THE SPECIFIED I/O CHANNEL IS IDLE, THEN IT IS IMMEDIATELY
00CD   436 ;       ASSIGNED TO THE CURRENT DRIVER PROCESS. ELSE THE DRIVER PROCESS
00CD   437 ;       CONTEXT IS SAVED IN ITS FORK BLOCK, THE FORK BLOCK IS INSERTED
00CD   438 ;       IN THE CHANNEL WAIT QUEUE, AND A RETURN TO THE DRIVER PROCESS'
00CD   439 ;       CALLER IS EXECUTED.
00CD   440 ;
00CD   441 ;       WHEN THE CHANNEL IS ASSIGNED, THE CSR ADDRESS OF THE ASSIGNED
00CD   442 ;       CONTROLLER IS RETURNED TO THE CALLER IN REGISTER R4.
00CD   443 ;
00CD   444 ;       R3 IS PRESERVED ACROSS CALL.
00CD   445 ;-
00CD   446
00CD   447          .ENABL  LSB
00CD   448 IOC$REQSCHANH::                              ;REQUEST SECONDARY I/O CHANNEL HIGH PRIORITY
50   24 A5   D0  00CD   449          MOVL    UCB$L_CRB(R5),R0        ;GET ADDRESS OF PRIMARY CRB
50   20 A0   D0  00D1   450          MOVL    CRB$L_LINK(R0),R0       ;GET ADDRESS OF SECONDARY CRB
        0E   11  00D5   451          BRB     10$                     ;
00D7   452 IOC$REQSCHANL::                              ;REQUEST SECONDARY I/O CHANNEL LOW PRIORITY
50   24 A5   D0  00D7   453          MOVL    UCB$L_CRB(R5),R0        ;GET ADDRESS OF PRIMARY CRB
50   20 A0   D0  00DB   454          MOVL    CRB$L_LINK(R0),R0       ;GET ADDRESS OF SECONDARY CRB
        0D   11  00DF   455          BRB     20$                     ;
00E1   456 IOC$REQPCHANH::                              ;REQUEST PRIMARY I/O CHANNEL HIGH PRIORITY
50   24 A5   D0  00E1   457          MOVL    UCB$L_CRB(R5),R0        ;GET ADDRESS OF PRIMARY CRB
52   50      D0  00E5   458 10$:      MOVL    R0,R2                   ;SET ADDDRESS OF WAIT QUEUE LISTHEAD
        08   11  00E8   459          BRB     30$                     ;
00EA   460 IOC$REQPCHANL::                              ;REQUEST PRIMARY I/O CHANNEL LOW PRIORITY
50   24 A5   D0  00EA   461          MOVL    UCB$L_CRB(R5),R0        ;GET ADDRESS OF PRIMARY CRB
52   04 A0   D0  00EE   462 20$:      MOVL    CRB$L_WQBL(R0),R2       ;GET ADDRESS OF LAST ENTRY IN QUEUE
51   2C A0   D0  00F2   463 30$:      MOVL    CRB$L_INTD+VEC$L_IDB(R0),R1 ;GET ADDRESS OF IDB
08 0E A0   00   E2  00F6   464          BBSS    #CRB$V_BSY,CRB$B_MASK(R0),40$ ;IF SET, THEN CHANNEL BUSY
54   61      D0  00FB   465          MOVL    IDB$L_CSR(R1),R4        ;SET ASSIGNED CHANNEL CSR ADDRESS
04 A1   55      D0  00FE   466          MOVL    R5,IDB$L_OWNER(R1)      ;SET OWNER UCB ADDRESS
        05      0102   467          RSB                             ;
10 A5   53      D0  0103   468 40$:      MOVL    R3,UCB$L_FR3(R5)        ;SAVE R3 IN FORK BLOCK
   0C A5 8ED0   0107   469          POPL    UCB$L_FPC(R5)           ;SAVE CHANNEL WAIT RETURN ADDRESS
62   65      0E  010B   470          INSQUE  UCB$L_FQFL(R5),CRB$L_WQFL(R2) ;INSERT DRIVER PROCESS IN CHANNEL WAIT
04 A1   55      D1  010E   471          CMPL    R5,IDB$L_OWNER(R1)      ;CURRENT DRIVER PROCESS OWNER?
        03      12  0112   472          BNEQ    50$                     ;IF NEQ, BRANCH TO RETURN
      FF73      31  0114   473          BRW     IOC$RELCHAN             ;IF EQL BRW TO RELEASE CHANNELS
0117   474 50$:
```

```
05  0117   475          RSB                                ;
    0118   476          .DSABL  LSB
```

```
                                                L 3
                     0118   478          .SBTTL  I/O Request Completion Processing for Class Drivers
                     0118   479
                     0118   480 ;+
                     0118   481 ; IOC$ALTREQCOM - I/O Request Complete Alternate Entry.
                     0118   482 ;
                     0118   483 ; This routine is entered when an I/O operation is completed on one
                     0118   484 ;       one of the devices using the disk or tape class drivers.
                     0118   485 ;       The packet is inserted in the I/O finish queue for I/O post
                     0118   486 ;       processing.
                     0118   487 ;
                     0118   488 ; INPUTS:
                     0118   489 ;
                     0118   490 ;       R0 = First  longword of I/O status
                     0118   491 ;       R1 = Second longword of I/O status
                     0118   492 ;       R5 = CDRP address
                     0118   493 ;
                     0118   494 ; OUTPUTS:
                     0118   495 ;
                     0118   496 ;       The I/O packet is inserted in the I/O Post Processing Queue,
                     0118   497 ;       a Software interrupt is requested to initiate I/O Post
                     0118   498 ;       Processing.
                     0118   499 ;-
                     0118   500
                     0118   501 IOC$ALTREQCOM::
 53   A0 A5    9E   0118   502          MOVAB   CDRP$L_IOQFL(R5),R3     ; R3 => IRP section of CDRP. This is
                     011C   503                                          ;  for compatibility with rest of QIO
                     011C   504                                          ;  logic.
 55   1C A3    D0   011C   505          MOVL    IRP$L_UCB(R3),R5        ; R5 => UCB.
      70 A5    D6   0120   506          INCL    UCB$L_OPCNT(R5)         ; Increment operations completed
                     0123   507
      15 50    E9   0123   508          BLBC    R0,20$                  ; LBC implies I/O error, so goto call
                     0126   509                                          ;  MOUNT VERIFICATION just in case.
                     0126   510 10$:
   38 A3    50    7D   0126   511          MOVQ    R0,IRP$L_MEDIA(R3)      ; Save final I/O status in IRP.
                     012A   512
                     012A   513          .IF DF  CA$_MEASURE_IOT
                     012A   514
00000000'GF    16   012A   515          JSB     G^PMS$END_IO            ; Insert end of I/O transaction message
                     0130   516
                     0130   517          .ENDC
                     0130   518
00000000'FF    63   0E   0130   519          INSQUE  (R3),@L^IOC$GL_PSBL     ; Insert packet in POST process queue
                     0137   520          SOFTINT #IPL$_IOPOST            ; Initiate SOFTWARE INTERRUPT
                05   013A   521          RSB
                     013B   522 20$:
00000000'GF    16   013B   523          JSB     G^EXE$MOUNTVER          ; If LBC, call MOUNT VERIFICATION.
          E3    11   0141   524          BRB     10$                     ; Go back to normal flow.
```

```
                            0143          .SBTTL  I/O REQUEST COMPLETION PROCESSING
                            0143  ;+
                            0143  ; IOC$REQCOM - I/O REQUEST COMPLETE
                            0143  ;
                            0143  ; THIS ROUTINE IS ENTERED WHEN AN I/O OPERATION IS COMPLETED ON A
                            0143  ; DEVICE UNIT. THE FINAL I/O STATUS IS STORED IN THE ASSOCIATED I/O
                            0143  ; PACKET AND THE PACKET IS INSERTED IN THE I/O FINISH QUEUE FOR
                            0143  ; I/O POST PROCESSING. DEVICE UNIT BUSY IS CLEARED AND AN ATTEMPT
                            0143  ; IS MADE TO START ANOTHER I/O REQUEST ON THE DEVICE UNIT.
                            0143  ;
                            0143  ; IF THE I/O REQUEST COMPLETED WITH AN ERROR, AND THE DEVICE IS
                            0143  ; A DISK, THEN BRANCH TO THE MOUNT VERIFICATION CODE, WHICH WILL
                            0143  ; DETERMINE IF THE SITUATION REQUIRES MOUNT VERIFICATION.
                            0143  ;
                            0143  ; IF MOUNT VERIFICATION IS IN PROGRESS, NO FURTHER I/O REQUESTS WILL
                            0143  ; BE INITIATED.  THIS HAS A SIDE EFFECT OF KEEPING THE 'BSY' BIT IN
                            0143  ; WHATEVER STATE IT IS CURRENTLY IN.  FOR CONVENTIONAL DISK DRIVERS,
                            0143  ; THE BSY BIT WILL BE LEFT ON, WHICH WILL BLOCK $QIO FROM INITIATING
                            0143  ; ANY NEW I/O ON THE DEVICE.  FOR THE DISK CLASS DRIVER, THE BUSY
                            0143  ; BIT WILL BE OFF, WHICH WILL ALLOW $QIO TO INITIATE NEW I/O.
                            0143  ;
                            0143  ; INPUTS:
                            0143  ;
                            0143  ;         R0 = FIRST LONGWORD OF I/O STATUS.
                            0143  ;         R1 = SECOND LONGWORD OF I/O STATUS.
                            0143  ;         R5 = UCB ADDRESS OF DEVICE UNIT.
                            0143  ;
                            0143  ; OUTPUTS:
                            0143  ;
                            0143  ;         THE I/O PACKET IS INSERTED IN THE I/O POST PROCESSING QUEUE
                            0143  ;         AND DEVICE UNIT BUSY IS CLEARED. A SOFTWARE INTERRUPT IS
                            0143  ;         REQUESTED TO INITIATE I/O POST PROCESSING.
                            0143  ;-
                            0143
                            0143          .ENABL  LSB
                            0143  IOC$REQCOM::                                    ;I/O DONE PROCESSING
 1C 64 A5      02      E5   0143          BBCC    #UCB$V_ERLOGIP,UCB$W_STS(R5),10$ ;IF CLR, ERROR LOG NOT IN PROGRESS
   52   0094 C5        D0   0148          MOVL    UCB$L_EMB(R5),R2                ;GET ADDRESS OF ERROR MESSAGE BUFFER
  1A A2    64 A5       B0   014D          MOVW    UCB$W_STS(R5),EMB$W_DV_STS(R2)  ;INSERT FINAL DEVICE STATUS
 10 A2   0080 C5       B0   0152          MOVW    UCB$B_ERTCNT(R5),EMB$B_DV_ERTCNT(R2) ;INSERT FINAL ERROR COUNTERS
    12 A2      50      7D   0158          MOVQ    R0,EMB$Q_DV_IOSB(R2)            ;INSERT FINAL I/O STATUS
               50      DD   015C          PUSHL   R0                              ;SAVE R0
           FE9F'       30   015E          BSBW    ERL$RELEASEMB                   ;RELEASE ERROR MESSAGE BUFFER
               50    8ED0   0161          POPL    R0                              ;RESTORE R0
    53    58 A5        D0   0164  10$:    MOVL    UCB$L_IRP(R5),R3                ;GET ADDRESS OF I/O PACKET
          70 A5        D6   0168          INCL    UCB$L_OPCNT(R5)                 ;INCREMENT OPERATIONS COMPLETED
          2A 50        E9   016B          BLBC    R0,DISKCHK                      ;IF I/O ERROR, CHECK FOR DISK DEVICE
                            016E  ;
                            016E  ; DO NOT SAVE THE I/O STATUS IN THE IRP UNTIL IT HAS BEEN DECIDED THAT
                            016E  ; MOUNT VERIFICATION IS NOT NECESSARY.  THIS IS TO AVOID OVERWRITING THE
                            016E  ; PHYSICAL DISK ADDRESS STORED IN THE IRP AT OFFSET IRP$L_MEDIA.
                            016E  ;
    38 A3      50      7D   016E  20$:    MOVQ    R0,IRP$L_MEDIA(R3)              ;STORE FINAL I/O STATUS
                            0172
                            0172          .IF DF  CA$_MEASURE_IOT
                            0172
      00000000'EF      D5   0172          TSTL    L^PMS$GL_IOPFMPDB               ;DATA COLLECTION ENABLED?
```

```
                                                N 3
                     36   12  0178   583          BNEQ    DO_PMS                  ;BRANCH IF YES
                                 017A   584
                                 017A   585          .ENDC
                                 017A   586
00000000'FF          63   0E  017A   587 PMSEND:  INSQUE  (R3),@L^IOC$GL_PSBL      ;INSERT PACKET IN POST PROCESS QUEUE
                                 0181   588          SOFTINT #IPL$_IOPOST         ;INITIATE SOFTWARE INTERRUPT
                     0E   E0    0184   589          BBS     #UCB$V_MNTVERIP,-    ;BRANCH IF MOUNT VERIFICATION IN PROGRESS
                   2F 64  A5     0186   590                  UCB$W_STS(R5),MNTVERPNDCHK ;(NOTE THIS LEAVES 'BSY' AS IS)
                 53   4C  B5   0189   591 NXTIRP:  REMQUE   @UCB$L_IOQFL(R5),R3   ;REMOVE I/O PACKET FROM DEVICE UNIT QUEUE
                     4C   1C    018D   592          BVC     IOC$INITIATE         ;IF VC INITIATE NEXT FUNCTION
                64 A5  0100 8F   AA   018F   593          BICW    #UCB$M_BSY,UCB$W_STS(R5) ;CLEAR UNIT BUSY
                                 0195   594 RELEASE:                            ;RELEASE ALL CHANNELS
                         FEF2   31   0195   595          BRW     IOC$RELCHAN          ;
                                 0198   596 ;
                                 0198   597 ; IF THIS IS A DISK DEVICE, CALL THE MOUNT VERIFICATION ROUTINE
                                 0198   598 ; TO DETERMINE IF MOUNT VERIFICATION IS NECESSARY.  IF NOT, CONTROL
                                 0198   599 ; WILL RETURN, AND THE REQUEST WILL BE COMPLETED IN THE NORMAL MANNER.
                                 0198   600 ;
                                 0198   601 DISKCHK:
                        01   91   0198   602          CMPB    #DC$_DISK,-          ;IS THIS DEVICE A DISK?
                     40 A5        019A   603                  UCB$B_DEVCLASS(R5)   ;
                        D0   12   019C   604          BNEQ    20$                  ;BRANCH IF NOT
                        13   E5   019E   605          BBCC    #UCB$V_MNTVERPND,-   ;CHECK FOR MOUNT VERIFICATION PENDING
                  05 64 A5        01A0   606                  UCB$L_STS(R5),30$    ;IF NOT, JUST ENTER MOUNT VERIFICATION
                        0E   E5   01A3   607          BBCC    #UCB$V_MNTVERIP,-    ;CLEAR IN-PROGRESS BIT BEFORE CALL
                  00 64 A5        01A5   608                  UCB$L_STS(R5),30$    ; SO IT WILL REALLY START
               00000000'GF   16   01A8   609 30$:     JSB     G^EXE$MOUNTVER       ;START MOUNT VERIFICATION
                        BE   11   01AE   610          BRB     20$                  ;COMPLETE I/O REQUEST
                                 01B0   611
                                 01B0   612          .IF DF  CA$_MEASURE_IOT
                                 01B0   613
00000000'GF          16   01B0   614 DO_PMS:  JSB     G^PMS$END_IO         ;INSERT END OF I/O TRANSACTION MESSAGE
                     C2   11   01B6   615          BRB     PMSEND               ;REJOIN COMMON CODE
                                 01B8   616
                                 01B8   617          .ENDC
                                 01B8   618 ;
                                 01B8   619 ; THE MOUNT-VERIFICATION-PENDING BIT IS USED TO INDICATE THAT A DISK SHOULD GO
                                 01B8   620 ; INTO MOUNT VERIFICATION AS SOON AS THE CURRENT I/O IS DONE.  THIS IS INTENDED
                                 01B8   621 ; FOR USE IN A CLUSTER TO STALL I/O WHEN QUORUM IS LOST.
                                 01B8   622 ;
                                 01B8   623 MNTVERPNDCHK:
                        13   E5   01B8   624          BBCC    #UCB$V_MNTVERPND,-   ;CHECK FOR MOUNT VERIFICATION PENDING
                  D8 64 A5        01BA   625                  UCB$L_STS(R5),RELEASE ;IF NOT, JUST CLEAN UP
                        01   91   01BD   626          CMPB    #DC$_DISK,-          ;IS THIS DEVICE A DISK?
                     40 A5        01BF   627                  UCB$B_DEVCLASS(R5)   ;
                        D2   12   01C1   628          BNEQ    RELEASE              ;BRANCH IF NOT
                        0E   E5   01C3   629          BBCC    #UCB$V_MNTVERIP,-    ;CLEAR IN-PROGRESS BIT BEFORE CALL
                  00 64 A5        01C5   630                  UCB$L_STS(R5),40$
                        53   D4   01C8   631 40$:     CLRL    R3                   ;NO IRP PASSED TO MOUNT VERIFICATION
               00000000'GF   16   01CA   632          JSB     G^EXE$MOUNTVER       ;TRY TO START MOUNT VERIFICATION
                        B7   11   01D0   633          BRB     NXTIRP               ;WASN'T NECESSARY
                                 01D2   634
                                 01D2   635          .DSABL  LSB
```

```
                                                B 4
                     01D2   637          .SBTTL  MOUNT VERIFICATION HELPER
                     01D2   638 ;++
                     01D2   639 ; IOC$MNTVER    - Assist driver with mount verification.
                     01D2   640 ;
                     01D2   641 ; This routine is called by EXE$MOUNTVER to perform some driver-specific
                     01D2   642 ; actions necessary for mount verification.  This routine is used by non-
                     01D2   643 ; CLASS drivers, and is called by default if EXE$MOUNTVER finds the address
                     01D2   644 ; of IOC$RETURN in DDT$L_MNTVER.
                     01D2   645 ;
                     01D2   646 ; Inputs:
                     01D2   647 ;
                     01D2   648 ;       R3      = IRP address or 0
                     01D2   649 ;       R5      = UCB address
                     01D2   650 ;
                     01D2   651 ; Outputs:
                     01D2   652 ;
                     01D2   653 ;       None.
                     01D2   654 ;
                     01D2   655 ; Side effects:
                     01D2   656 ;
                     01D2   657 ;       If R3 contains an IRP address, the IRP will be queued to the
                     01D2   658 ;       head of the UCB's IRP work queue.  If R3 contains is zero, then
                     01D2   659 ;       remove the IRP from the head of the UCB's work queue and attempt
                     01D2   660 ;       to initiate the I/O.
                     01D2   661 ;--
                     01D2   662
                     01D2   663 IOC$MNTVER::                            ;Driver-specific mount verification code
     53    D5        01D2   664          TSTL    R3                     ;Check IRP address
     B3    13        01D4   665          BEQL    NXTIRP                 ;Branch if none
     63    0E        01D6   666          INSQUE  IRP$L_IOQFL(R3),-      ;Requeue the IRP
4C   A5              01D8   667                  UCB$L_IOQFL(R5)        ;
                05    01DA   668          RSB                            ;Return
```

```
                                                                C 4
                     01DB   670          .SBTTL  INITIATE I/O FUNCTION ON DEVICE
                     01DB   671 ;+
                     01DB   672 ; IOC$INITIATE - INITIATE NEXT FUNCTION ON DEVICE
                     01DB   673 ;
                     01DB   674 ; THIS ROUTINE IS CALLED TO INITIATE THE NEXT FUNCTION ON A DEVICE BY CLEARING
                     01DB   675 ; STATUS BITS, SETTING THE OPERATION START TIME IF A DIAGNOSTIC BUFFER IS
                     01DB   676 ; SPECIFIED, AND CALLING THE DRIVER AT ITS START I/O ENTRY POINT.
                     01DB   677 ;
                     01DB   678 ; INPUTS:
                     01DB   679 ;
                     01DB   680 ;       R3 = ADDRESS OF I/O REQUEST PACKET.
                     01DB   681 ;       R5 = DEVICE UNIT UCB ADDRESS.
                     01DB   682 ;
                     01DB   683 ; OUTPUTS:
                     01DB   684 ;
                     01DB   685 ;       CANCEL I/O, POWERFAIL, AND TIME OUT STATUS BITS ARE CLEARED, THE
                     01DB   686 ;       CURRENT SYSTEM TIME IS FILLED INTO THE INTERNAL DIAGNOSTIC BUFFER
                     01DB   687 ;       IF ONE IS SPECIFIED, AND THE DRIVER IS CALLED AT ITS START I/O ENTRY
                     01DB   688 ;       POINT.
                     01DB   689 ;-
                     01DB   690
                     01DB   691 IOC$INITIATE::                          ;INITIATE I/O FUNCTION
        58 A5   53   D0  01DB   692         MOVL    R3,UCB$L_IRP(R5)      ;SAVE I/O PACKET ADDRESS
                     01DF   693
                     01DF   694         .IF DF  CA$_MEASURE_IOT
                     01DF   695
 00000000'GF    16   01DF   696         JSB     G^PMS$START_IO            ;INSERT START OF I/O TRANSACTION MESSAGE
                     01E5   697
                     01E5   698         .ENDC
                     01E5   699
    78 A5    2C A3   7D  01E5   700         MOVQ    IRP$L_SVAPTE(R3),UCB$L_SVAPTE(R5) ;COPY TRANSFER PARAMETERS
  64 A5    0048 8F   AA  01EA   701         BICW    #UCB$M_CANCEL!UCB$M_TIMOUT,UCB$W_STS(R5) ;CLEAR CANCEL AND TIME OUT
    0B 2A A3    07   E1  01F0   702         BBC     #IRP$V_DIAGBUF,IRP$W_STS(R3),10$ ;IF CLR, NO DIAGNOSTIC BUFFER
       50    4C B3   D0  01F5   703         MOVL    @IRP$L_DIAGBUF(R3),R0     ;GET ADDRESS OF DIAGNOSTIC BUFFER DATA AREA
60    00000000'EF   7D  01F9   704         MOVQ    EXE$GQ_SYSTIME,(R0)       ;INSERT I/O OPERATION START TIME
     50    0088 C5   D0  0200   705 10$:    MOVL    UCB$L_DDT(R5),R0          ;GET ADDRESS OF DRIVER DISPATCH TABLE
           00 B0   17  0205   706         JMP     @DDT$L_START(R0)          ;START I/O OPERATION
```

```
                                                D 4
                     0208   708          .SBTTL  Allocate Buffered Data Path
                     0208   709 ;+
                     0208   710 ; ALLOCATE BUFFERED DATA PATH CODE -
                     0208   711 ;
                     0208   712 ; IOC$REQDATAP - Entrypoint (called from traditional drivers) where caller
                     0208   713 ;       wishes to be queued (using UCB fork block) if no buffered data path
                     0208   714 ;       is available at the time of the call.
                     0208   715 ; INPUT:
                     0208   716 ;                   R5 => UCB.
                     0208   717 ;
                     0208   718 ; IOC$REQDATAPNW - Entrypoint to call when caller does not want to wait for
                     0208   719 ;       unavailable data path.
                     0208   720 ; INPUT:
                     0208   721 ;                   R5 => UCB
                     0208   722 ;
                     0208   723 ; IOC$REQDATAPUDA - Entrypoint (called from UDA port driver) where CDRP
                     0208   724 ;       is used as the source of information about the request and where
                     0208   725 ;       the caller does not want to wait for unavailable datapath.
                     0208   726 ;
                     0208   727 ; INPUT:
                     0208   728 ;                   R4 => PDT
                     0208   729 ;                   R5 => CDRP
                     0208   730 ;-
                     0208   731
                     0208   732 IOC$REQDATAP::
          10   10    0208   733          BSBB    IOC$REQDATAPNW          ; Try to alloc. and get control after.
       0C 50   E8    020A   734          BLBS    R0,10$                  ; LBS implies allocation success.
                     020D   735
    10 A5   53   7D  020D   736          MOVQ    R3,UCB$L_FR3(R5)        ; Save driver context in UCB fork block.
       0C A5 8ED0    0211   737          POPL    UCB$L_FPC(R5)           ; Save caller's return point.
          65   0E    0215   738          INSQUE  UCB$L_FQFL(R5),-        ; Queue fork block to resource wait queue.
    18 B1              0217   739                  @ADP$L_DPQBL(R1)        ;  Assumes IOC$ALODATAP saves R1=>ADP.
                05    0219   740 10$:     RSB                             ; Return to caller or caller's caller.
                      021A   741
                      021A   742 IOC$REQDATAPNW::
    50   24 A5   D0   021A   743          MOVL    UCB$L_CRB(R5),R0                ; R0=>CRB.
    51   38 A0   D0   021E   744          MOVL    CRB$L_INTD+VEC$L_ADP(R0),R1     ; R1=>ADP (pass to IOC$ALODATAP)
    52   34 A0   9E   0222   745          MOVAB   CRB$L_INTD+VEC$W_MAPREG(R0),R2  ; R2=>UBMD      "  "  "
                      0226   746
            40   11   0226   747          BRB     IOC$ALODATAP            ; NOWAIT, RSB from IOC$ALODATAP
                      0228   748                                          ;  returns to our caller.
                      0228   749 IOC$REQDATAPUDA::
 51   00E0 C4   D0   0228   750          MOVL    PDT$L_ADP(R4),R1        ; R1=>ADP (pass to IOC$ALODATAP)
   52   3C A5   9E   022D   751          MOVAB   CDRP$L_UBARSRCE(R5),R2  ; R2=>UBMD      "  "  "
                      0231   752
            35   10   0231   753          BSBB    IOC$ALODATAP            ; Call to allocate a data path.
       2D 50   E8    0233   754          BLBS    R0,20$                  ; LBS means we got one.
    29 D0 A5   E9    0236   755          BLBC    CDRP$W_BOFF(R5),20$     ; LBC means, user buffer is on an
                      023A   756                                          ;  even byte address so we can use
                      023A   757                                          ;  the Direct Data Path.
                      023A   758
                      023A   759 ; Here we have a transfer to a user buffer located at an odd byte address.
                      023A   760 ; On those processors which support Byte Offset on the Direct Datapath, we
                      023A   761 ; can continue processing.  On other processors, we must wait for a buffered
                      023A   762 ; datapath.
                      023A   763
                      023A   764          CPUDISP <<780,10$>,-            ; On 11-780 we wait.
```

```
                     023A   765                      <750,20$>,-             ; On 11-750 we continue.
                     023A   766                      <730,20$>,-             ; On 11-730 we continue.
                     023A   767                      <790,10$>,-             ; On 11-790 we wait.
                     023A   768                      <8SS,10$>,-             ; On SCORPIO we wait.
                     023A   769                      <8NN,10$>,-             ; On NAUTILUS we wait.
                     023A   770                      <UV1,30$>>              ; On MicroVAX we bugcheck.
                     0254   771
10 A5    53    7D    0254   772 10$:    MOVQ    R3,CDRP$L_FR3(R5)           ; Save driver context in CDRP fork block.
      0C A5  8ED0    0258   773          POPL    CDRP$L_FPC(R5)              ; Save caller's return point.
      28 B5    B6    025C   774          INCW    @CDRP$L_RWCPTR(R5)          ; Increment RWAITCNT.
         65    0E    025F   775          INSQUE  CDRP$L_FQFL(R5),-           ; Queue fork block to resource wait queue.
      18 B1          0261   776                  @ADP$L_DPQBL(R1)            ;  Assumes IOC$ALODATAP saves R1=>ADP.
                     0263   777 20$:
                05   0263   778          RSB                                 ; Return to caller or caller's caller.
                     0264   779
                     0264   780 30$:    BUG_CHECK       IVBYTEALGN,FATAL
```

```
                                                F 4
                     0268   782 ; IOC$ALODATAP - Common subroutine called by above routines to allocate
                     0268   783 ;       a UNIBUS buffered datapath.
                     0268   784 ;
                     0268   785 ; INPUTS:
                     0268   786 ;       R1 => ADP wherein the datapath allocation bit map is stored.
                     0268   787 ;       R2 => UBA mapping descriptor in user's data structure.
                     0268   788 ;
                     0268   789 ; OUTPUTS:
                     0268   790 ;       R0 LBS - implies allocation success
                     0268   791 ;               datapath field in R2 => UBA mapping descriptor is set to the
                     0268   792 ;                       number of the datapath allocated.
                     0268   793 ;               appropriate bit in datapath allocation bit map is cleared.
                     0268   794 ;       R0 LBC - implies allocation failure.
                     0268   795 ;
                     0268   796
                     0268   797 IOC$ALODATAP:
          07   E0    0268   798         BBS     #VEC$V_PATHLOCK,-       ; If this user has a permanently allocated
17 03 A2             026A   799                 UBMD$B_DATAPATH(R2),10$ ;  datapath, branch around to success.
                     026D   800
                     026D   801         ASSUME  ADP$C_NUMDATAP  EQ  16
          00   EA    026D   802         FFS     #0,-                    ; Find first available datapath,
          10         026F   803                 #ADP$C_NUMDATAP,-       ;  according to bit map.  Note failure
60 A1                0270   804                 ADP$W_DPBITMAP(R1),-    ;  leaves R0 with the value "16", an
          50         0272   805                 R0                      ;  even number with the low bit clear.
          12   13    0273   806         BEQL    20$                     ; EQL implies failure.
                     0275   807
          50   F0    0275   808         INSV    R0,-                    ; Upon success, R0 has number of the
                     0277   809                 #VEC$V_DATAPATH,-       ;  available datapath to allocate.
05   00              0277   810                 #VEC$S_DATAPATH,-       ;  So we update the user's datapath
  03 A2              0279   811                 UBMD$B_DATAPATH(R2)     ;  descriptor pointed at by R2.
                     027B   812
04 60 A1   50   E4   027B   813         BBSC    R0,ADP$W_DPBITMAP(R1),10$; And we update the bit map.
                     0280   814         BUG_CHECK INCONSTATE            ; We shouldn't be here obviously.
                     0284   815
     50   01   D0   0284   816 10$:    MOVL    S^#SS$_NORMAL,R0        ; Indicate allocation success.
                05   0287   817 20$:    RSB                             ; And we return to our caller.
```

```
                     0288   819          .SBTTL  Release Buffered Data Path
                     0288   820 ;+
                     0288   821 ; RELEASE BUFFERED DATA PATH CODE -
                     0288   822 ;
                     0288   823 ; IOC$RELDATAPUDA - Entry point called from UDA port driver in response
                     0288   824 ;       to an UNMAP call.  Here the data as to the buffered data path
                     0288   825 ;       is in the CDRP.
                     0288   826 ;
                     0288   827 ; INPUTS:
                     0288   828 ;       R4 => PDT
                     0288   829 ;       R5 => CDRP
                     0288   830 ;
                     0288   831 ; IOC$RELDATAP - Entry point called from traditional drivers to release
                     0288   832 ;       the buffered datapath described in CRB$L_INTD+VEC$B_DATAPATH.
                     0288   833 ;
                     0288   834 ; INPUTS:
                     0288   835 ;       R5 => UCB
                     0288   836 ;
                     0288   837 ; OUTPUTS:
                     0288   838 ;       Datapath re-allocated (if any waiters). R0, R1, and R2 modified.
                     0288   839 ; NOTE: Since calls to IOC$REQDATAPUDA are NOWAIT, fork blocks dequeued
                     0288   840 ;       here from ADP$L_DPQFL are guaranteed to be UCB's.
                     0288   841 ;
                     0288   842
                     0288   843 IOC$RELDATAPUDA::
51   00E0 C4    D0   0288   844          MOVL    PDT$L_ADP(R4),R1          ; R1 => ADP.
  52   3C A5    9E   028D   845          MOVAB   CDRP$L_UBARSRCE(R5),R2    ; R2 => UBMD.
       0C       11   0291   846          BRB     RELDATAP_COMMON
                     0293   847 IOC$RELDATAP::
  50   24 A5    D0   0293   848          MOVL    UCB$L_CRB(R5),R0                  ; R0 => CRB.
  52   34 A0    9E   0297   849          MOVAB   CRB$L_INTD+VEC$W_MAPREG(R0),R2    ; R2 => UBMD.
  51   38 A0    D0   029B   850          MOVL    CRB$L_INTD+VEC$L_ADP(R0),R1       ; R1 => ADP.
                     029F   851 RELDATAP_COMMON:
  50   03 A2    98   029F   852          CVTBL   UBMD$B_DATAPATH(R2),R0            ; Get datapath designator.
       36       15   02A3   853          BLEQ    10$                               ; If LSS permanent assignment.
                     02A5   854                                                    ; If EQL we had NO datapath to
                     02A5   855                                                    ;  release.
       00       F0   02A5   856          INSV    #0,-                              ; Zero datapath number.
     05 00              02A7   857                  #VEC$V_DATAPATH,#VEC$S_DATAPATH,-
        03 A2           02A9   858                  UBMD$B_DATAPATH(R2)
       00       EF   02AB   859          EXTZV   #VEC$V_DATAPATH,-         ; Extract datapath number.
52   50  05              02AD   860                  #VEC$S_DATAPATH,R0,R2     ;
  50   14 B1    0F   02B0   861          REMQUE  @ADP$L_DPQFL(R1),R0       ; R0 => next driver fork block
       26       1D   02B4   862          BVS     20$                       ; If VS no driver process waiting
                     02B6   863
     7E   53    7D   02B6   864          MOVQ    R3,-(SP)                  ; Save R3, R4, R5
          55    DD   02B9   865          PUSHL   R5
     55   50    D0   02BB   866          MOVL    R0,R5                     ; R5 => driver fork block.
          10    91   02BE   867          CMPB    #DYN$C_UCB,-              ; See if we dequeued a UCB or a CDRP.
        0A A5           02C0   868                  UCB$B_TYPE(R5)
       22       12   02C2   869          BNEQ    30$                       ; NEQ implies a CDRP.
                     02C4   870
                     02C4   871 ; Here we have R5 => UCB.
                     02C4   872
  51   24 A5    D0   02C4   873          MOVL    UCB$L_CRB(R5),R1          ; R1 => CRB.
                     02C8   874
       52       F0   02C8   875          INSV    R2,-                      ; Store assigned datapath #
```

```
                                                H 4
                02CA   876                      #VEC$V_DATAPATH,-       ;  in CRB.
          05    00     02CA   877                      #VEC$S_DATAPATH,-
            37  A1     02CC   878                      CRB$L_INTD+VEC$B_DATAPATH(R1)
                       02CE   879
    53    10    A5  7D 02CE   880              MOVQ     UCB$L_FR3(R5),R3        ; Restore driver context.
          0C    B5  16 02D2   881              JSB      @UCB$L_FPC(R5)          ; Call back waiting driver.
                       02D5   882 5$:
          55 8ED0      02D5   883              POPL     R5                      ; Restore deallocator's R5,R4,R3
          53    8E  7D 02D8   884              MOVQ     (SP)+,R3                ;
                05     02DB   885 10$:         RSB                              ; Return to deallocator.
FA 60 A1  52    E3     02DC   886 20$:         BBCS     R2,-
                       02E1   887                      ADP$W_DPBITMAP(R1),10$  ; Set datapath bit and exit
                       02E1   888              BUG_CHECK INCONSTATE             ; Inconsistent state.
                05     02E5   889              RSB                              ;
                       02E6   890
                       02E6   891 ; Here we have R5 => CDRP.
                       02E6   892
                       02E6   893 30$:
          52    F0     02E6   894              INSV     R2,-                    ; Store assigned datapath #
                       02E8   895                      #VEC$V_DATAPATH,-       ;  in CDRP field.
          05    00     02E8   896                      #VEC$S_DATAPATH,-
            3F  A5     02EA   897                      CDRP$L_UBARSRCE+UBMD$B_DATAPATH(R5)
                       02EC   898
00000000'EF       16   02EC   899              JSB      SCS$RESUMEWAITR         ; Resume waiting thread and any backed
                       02F2   900                                               ;  up IRP's.
          E1    11     02F2   901              BRB      5$                      ; Branch back to resume deallocator's
                       02F4   902                                               ;  thread.
```

```
                                                          I 4
                02F4   904          .SBTTL  REQUEST AND ALLOCATE UNIBUS MAP REGISTERS FOR CLASS DRIVER
                02F4   905 ;+
                02F4   906 ; IOC$REQMAPUDA - REQUEST AND ALLOCATE UNIBUS MAP REGISTERS FOR CLASS DRIVER
                02F4   907 ;
                02F4   908 ; THIS ROUTINE IS CALLED TO ALLOCATE UBA MAP REGISTERS AND TO MARK THE ALLOCATION
                02F4   909 ; IN THE UBA MAP REGISTER ALLOCATION DATA STRUCTURES.
                02F4   910 ;
                02F4   911 ; INPUTS:
                02F4   912 ;
                02F4   913 ;       R4 = ADDRESS OF PORT DESCRIPTOR TABLE.
                02F4   914 ;       R5 = ADDRESS OF CLASS DRIVER REQUEST PACKET (CDRP).
                02F4   915 ;
                02F4   916 ; OUTPUTS:
                02F4   917 ;
                02F4   918 ;       IF MAP REGISTERS ARE ALLOCATED FOR THE CDRP, THE APPROPRIATE FIELDS
                02F4   919 ;       IN THE CDRP ARE MODIFIED TO INDICATE WHICH REGISTERS, AND THE NUMBER
                02F4   920 ;       OF REGISTERS THAT HAVE BEEN ALLOCATED.  ALSO THE ALLOCATION DATA
                02F4   921 ;       STRUCTURE IN THE ADP IS MODIFIED.
                02F4   922 ;
                02F4   923 ;       IF MAP REGISTERS CANNOT BE ALLOCATED AT THIS TIME, THE CDRP IS
                02F4   924 ;       QUEUED ONTO THE RESOURCE WAIT LIST AND THE UCB$W_RWAITCNT IS
                02F4   925 ;       INCREMENTED.
                02F4   926 ;
                02F4   927 ;-
                02F4   928
                02F4   929 IOC$REQMAPUDA::                          ; Allocate UBA map registers for class drive
     25   10    02F4   930          BSBB    IOC$ALOMAPUDA          ; Call to allocate map registers if availabl
                02F6   931                                         ;  Returns R2 => ADP.
                02F6   932
                02F6   933 ; If here, low bit of R0 tells us whether we were successful in the allocation
                02F6   934 ;       attempt.
                02F6   935
          0F 50   E8   02F6   936          BLBS    R0,10$                 ; Branch around if successful.
  10 A5    53   7D     02F9   937          MOVQ    R3,CDRP$L_FR3(R5)      ; Save driver process context
        28 B5   B6     02FD   938          INCW    @CDRP$L_RWCPTR(R5)     ; One more CDRP, on this UCB, awaiting
                       0300   939                                         ;  resources.
        0C A5 8ED0     0300   940          POPL    CDRP$L_FPC(R5)         ; Save map register wait return address
           65   0E     0304   941          INSQUE  CDRP$L_FQFL(R5),-
        34 B2          0306   942                  @ADP$L_MRQBL(R2)       ; Insert process in map register wait queue
                 05    0308   943 10$:     RSB                            ;
```

```
                                                    J 4
                     0309   945          .SBTTL  REQUEST UNIBUS MAP REGISTERS
                     0309   946 ;+
                     0309   947 ; IOC$REQMAPREG - REQUEST UNIBUS MAP REGISTERS
                     0309   948 ;
                     0309   949 ; THIS ROUTINE IS CALLED TO REQUEST UNIBUS MAP REGISTERS TO PERFORM AN
                     0309   950 ; I/O TRANSFER.
                     0309   951 ;
                     0309   952 ; INPUTS:
                     0309   953 ;
                     0309   954 ;       R5 = UCB ADDRESS OF DEVICE UNIT.
                     0309   955 ;       04(SP) = RETURN ADDRESS OF CALLER'S CALLER.
                     0309   956 ;
                     0309   957 ;       IT IS ASSUMED THAT THE CALLER OWNS THE I/O CHANNEL ON WHICH THE
                     0309   958 ;       TRANSFER IS TO OCCUR ON.
                     0309   959 ;
                     0309   960 ; OUTPUTS:
                     0309   961 ;
                     0309   962 ;       IF MAP REGISTERS HAVE BEEN PERMANENTLY ASSIGNED TO THE ASSOCIATED
                     0309   963 ;       I/O CHANNEL, THEN CONTROL IS IMMEDIATELY RETURNED TO THE CALLER.
                     0309   964 ;       ELSE AN ATTEMPT IS MADE TO ALLOCATE THE REQUESTED NUMBER OF MAP REG-
                     0309   965 ;       ISTERS. IF SUFFICIENT CONTIGUOUS MAP REGISTERS ARE FOUND, THEN THEY
                     0309   966 ;       ARE ASSIGNED TO THE ASSOCIATED I/O CHANNEL AND CONTROL IS RETURNED
                     0309   967 ;       TO THE CALLER. ELSE THE DRIVER PROCESS CONTEXT IS SAVED IN ITS FORK
                     0309   968 ;       BLOCK, THE FORK BLOCK IS INSERTED IN THE MAP REGISTER WAIT QUEUE,
                     0309   969 ;       AND A RETURN TO THE DRIVER PROCESS' CALLER IS EXECUTED.
                     0309   970 ;-
                     0309   971
                     0309   972 IOC$REQMAPREG::                                ;REQUEST UNIBUS MAP REGISTERS
          3A   10    0309   973          BSBB    IOC$ALOUBAMAP                 ; ALLOCATE UBA MAP REGISTER
       0C 50   E8    030B   974          BLBS    R0,10$                        ;IF LBS SUCCESSFUL ALLOCATION
10 A5     53   7D    030E   975          MOVQ    R3,UCB$L_FR3(R5)              ;SAVE DRIVER PROCESS CONTEXT
       0C A5 8ED0    0312   976          POPL    UCB$L_FPC(R5)                 ;SAVE MAP REGISTER WAIT RETURN ADDRESS
34 B2     65   0E    0316   977          INSQUE  UCB$L_FQFL(R5),@ADP$L_MRQBL(R2) ;INSERT PROCESS IN MAP REGISTER WAIT
                05    031A   978 10$:     RSB                                   ;
```

```
                                                K 4
                                .SBTTL  ALLOCATE UNIBUS MAP REGISTERS
                        ;+
                        ; IOC$ALOUBAMAP - ALLOCATE UBA MAP REGISTERS (CRB DATABASE SPECIFIED)
                        ; IOC$ALOUBAMAPN - ALLOCATE UBA MAP REGISTERS (ARGUMENT SPECIFIED)
                        ; IOC$ALOMAPUDA - ALLOCATE UBA MAP REGISTERS (FOR CLASS DRIVER(S))
                        ;
                        ; This routine is called to allocate uba map registers and to mark the allocation
                        ;       in the map register allocation structure located in the ADP.  The state
                        ;       of the UNIBUS map registers is maintained in a set of descriptors
                        ;       that describe contiguous extents of allocatable (i.e. free) map
                        ;       registers.  A map register descriptor consists of the
                        ;       corresponding elements of two distinct arrays (of one word items)
                        ;       located in the ADP.  These arrays, ADP$W_MRNREGARY and ADP$W_MRFREGARY,
                        ;       contain the number of map registers and the first map register in each
                        ;       contiguous extent of free map registers.  These arrays are each
                        ;       preceeded by a one word field containing all 1's (-1) so that compares
                        ;       made against the "previous" descriptor fail when the current desciptor
                        ;       is the one whose index is zero.
                        ;
                        ;       ADP$L_MRACTMDRS maintains the number of active descriptors, i.e. the
                        ;       number of elements of each array which contain valid data.
                        ;
                        ; INPUTS: (FOR IOC$ALOUBAMAP AND ALOUBAMAPN)
                        ;       R3 = NUMBER OF MAP REGISTERS TO ALLOCATE (IOC$ALOUBAMAPN only).
                        ;       R5 = DEVICE UNIT UCB ADDRESS.
                        ;
                        ; INPUT:  (FOR IOC$ALOMAPUDA)
                        ;       R4 => PDT
                        ;       R5 => CDRP
                        ;
                        ; OUTPUTS:
                        ;       R0 = SUCCESS INDICATION.
                        ;       R2 => ADP
                        ;-
                                .enabl  lsb
IOC$ALOMAPUDA:
        7E  53  7D      MOVQ    R3,-(SP)                ; Save R3,R4,R5
            55  DD      PUSHL   R5                      ;
                        MOVL    PDT$L_ADP(R4),R2        ; R2 => ADP before we modify R4.
                        MOVL    CDRP$L_BCNT(R5),R3      ; Get transfer byte count
                        MOVZWL  CDRP$W_BOFF(R5),R4      ; Get byte offset in page
                        MOVAB   ^X3FF(R3)[R4],R3        ; Calculate highest relative byte and round
                        ASHL    #-9,R3,R3               ; Calculate number of map registers required
                        MOVAB   CDRP$L_UBARSRCE(R5),R1  ; R1 => UBMD.
                        BRB     COMMON_ALOUBAMAP        ; Branch to common code.
IOC$ALOUBAMAPN::                                        ;ALLOCATE UBA MAP REGISTERS ARGUMENT SPECIFI
        7E  53  7D      MOVQ    R3,-(SP)                ; Save R3,R4,R5
            55  DD      PUSHL   R5                      ;
            18  11      BRB     5$                      ;
IOC$ALOUBAMAP::                                         ;ALLOCATE UBA MAP REGISTERS CRB SPECIFIED
        7E  53  7D      MOVQ    R3,-(SP)                ; Save R3,R4,R5
            55  DD      PUSHL   R5                      ;
```

```
                     034A  1037
     53   7E A5   3C 034A  1038          MOVZWL  UCB$W_BCNT(R5),R3       ;GET TRANSFER BYTE COUNT
     54   7C A5   3C 034E  1039          MOVZWL  UCB$W_BOFF(R5),R4       ;GET BYTE OFFSET IN PAGE
 53   03FF C344   9E 0352  1040          MOVAB   ^X3FF(R3)[R4],R3        ;CALCULATE HIGHEST RELATIVE BYTE AND ROUND
53   53   F7 8F   78 0358  1041          ASHL    #-9,R3,R3               ;CALCULATE NUMBER OF MAP REGISTERS REQUIRED
                     035D  1042 5$:
     51   24 A5   D0 035D  1043          MOVL    UCB$L_CRB(R5),R1                ; R1 => CRB.
     52   38 A1   D0 0361  1044          MOVL    CRB$L_INTD+VEC$L_ADP(R1),R2     ; R2 => ADP.
     51   34 A1   9E 0365  1045          MOVAB   CRB$L_INTD+VEC$W_MAPREG(R1),R1  ; R1 => UBMD.
               0F   E0 0369  1046          BBS     #VEC$V_MAPLOCK,-                ; If SET, already permanently
            38 61        036B  1047                  UBMD$W_MAPREG(R1),40$           ;  allocated, so branch around.
                     036D  1048
                     036D  1049 ; Here:
                     036D  1050 ;       R1 => UBMD - caller's structure where we record registers allocated
                     036D  1051 ;       R2 => ADP
                     036D  1052 ;       R3 =  number of map registers to allocate
                     036D  1053 ;
                     036D  1054
                     036D  1055 COMMON_ALOUBAMAP:
          5C A2   D5 036D  1056          TSTL    ADP$L_MRACTMDRS(R2)     ; Test for zero active descriptors.
             13   13 0370  1057          BEQL    15$                     ; EQL implies no registers available.
             53   D6 0372  1058          INCL    R3                      ; Round up request to next multiple
        53   01   8A 0374  1059          BICB    #1,R3                   ;  of 2.
             55   D4 0377  1060          CLRL    R5                      ; Establish loop variable.
                     0379  1061 10$:
  64 A245    53   B1 0379  1062          CMPW    R3,ADP$W_MRNREGARY(R2)[R5]      ; See if enough regs described here.
             09   15 037E  1063          BLEQ    20$                             ; LEQ implies YES.
                     0380  1064
F4 55    5C A2   F2 0380  1065          AOBLSS  ADP$L_MRACTMDRS(R2),R5,10$      ; Else branch back and continue
                     0385  1066 15$:
             50   D4 0385  1067          CLRL    R0                              ; If here, allocation failure.
             1F   11 0387  1068          BRB     50$                             ; Branch around to return.
                     0389  1069 20$:
61   015E C245   B0 0389  1070          MOVW    ADP$W_MRFREGARY(R2)[R5],-       ; Allocate from low end of extent
                     038F  1071                  UBMD$W_MAPREG(R1)               ;  by copying 1st map reg. #.
    02 A1   53   90 038F  1072          MOVB    R3,UBMD$B_NUMREG(R1)            ; Set # of map regs allocated.
  64 A245   53   A2 0393  1073          SUBW    R3,ADP$W_MRNREGARY(R2)[R5]      ; Subtract out # regs allocated.
             05   12 0398  1074          BNEQ    30$                             ; NEQ implies extent not empty,
                     039A  1075                                                  ;   branch around deallocate.
          0129   30 039A  1076          BSBW    DEALLOC_DESCRIP                 ; Call to deallocate descriptor.
             06   11 039D  1077          BRB     40$                             ;  And branch back to return.
                     039F  1078 30$:
015E C245    53   A0 039F  1079          ADDW    R3,ADP$W_MRFREGARY(R2)[R5]      ; Bump descriptor past
                     03A5  1080                                                  ;  allocated registers.
        50   01   D0 03A5  1081 40$:    MOVL    S^#SS$_NORMAL,R0        ; Indicate success.
             55 8ED0 03A8  1082 50$:    POPL    R5                      ; Restore R5,R4,R3
        53   8E   7D 03AB  1083          MOVQ    (SP)+,R3                ;
             05   03AE  1084          RSB
                     03AF  1085          .dsabl  lsb
```

```
                                                M 4
                     03AF  1087          .SBTTL  Allocate a specific set of UNIBUS Map Registers
                     03AF  1088 ;+
                     03AF  1089 ; IOC$ALOUBAMAPSP
                     03AF  1090 ;
                     03AF  1091 ; This routine is called to allocate a specific set of UNIBUS Map Registers.
                     03AF  1092 ;
                     03AF  1093 ; INPUTS:
                     03AF  1094 ;       R3 =  # of map registers to allocate
                     03AF  1095 ;       R4 =  # of first map register to allocate
                     03AF  1096 ;       R5 => UCB
                     03AF  1097 ;
                     03AF  1098 ; OUTPUTS:
                     03AF  1099 ;       R0 =  Success or failure indication
                     03AF  1100 ;             Note R0, R1 and R2 modified.
                     03AF  1101 ;-
                     03AF  1102
                     03AF  1103 IOC$ALOUBAMAPSP::
        7E   53  7D  03AF  1104          MOVQ    R3,-(SP)                        ; Save R3,R4,R5
             55  DD  03B2  1105          PUSHL   R5                              ;
                     03B4  1106
   50  24 A5  D0  03B4  1107          MOVL    UCB$L_CRB(R5),R0                ; R0 => CRB.
   52  38 A0  D0  03B8  1108          MOVL    CRB$L_INTD+VEC$L_ADP(R0),R2     ; R2 => ADP.
   51  34 A0  9E  03BC  1109          MOVAB   CRB$L_INTD+VEC$W_MAPREG(R0),R1  ; R1 => UBA mapping descriptor.
                     03C0  1110
      5C A2  D5  03C0  1111          TSTL    ADP$L_MRACTMDRS(R2)             ; Test for zero active descriptors.
         2C  13  03C3  1112          BEQL    30$                             ; EQL implies no registers available.
      05 54  E9  03C5  1113          BLBC    R4,10$                          ; Prepare to round DOWN to even boundary.
      54 01  8A  03C8  1114          BICB    #1,R4                           ; Clear low bit if set and
         53  D6  03CB  1115          INCL    R3                              ;  then increment # of registers to allocate
                     03CD  1116 10$:
         53  D6  03CD  1117          INCL    R3                              ; Prepare to round UP to even # of registers
            53  01  8A  03CF  1118          BICB    #1,R3                           ;
                     03D2  1119
            55  D4  03D2  1120          CLRL    R5                              ; R5 will be index register.
                     03D4  1121 20$:
      015E C245  54  B1  03D4  1122          CMPW    R4,ADP$W_MRFREGARY(R2)[R5]      ; Are registers we want in
                     03DA  1123                                                  ;  current extent?
            15  19  03DA  1124          BLSS    30$                             ; LSS means current extent is beyond the
                     03DC  1125                                                  ;  desired registers. Therefore they are
                     03DC  1126                                                  ;  not availiable and we have failed.
            46  13  03DC  1127          BEQL    50$                             ; EQL means they are at the beginning
                     03DE  1128                                                  ;  of the current extent.
                     03DE  1129
                     03DE  1130 ; Here the registers we want are either within the middle of the current
                     03DE  1131 ;       extent or else beyond the current extent.
                     03DE  1132
50 64 A245  015E C245  A1  03DE  1133          ADDW3   ADP$W_MRFREGARY(R2)[R5],-       ; R0 = 1st register beyond
                     03E7  1134                  ADP$W_MRNREGARY(R2)[R5],R0      ;  current extent.
            50  54  B1  03E7  1135          CMPW    R4,R0                           ; Are we in current extent?
                09  19  03EA  1136          BLSS    40$                             ; LSS means YES, in current.
         E3 55   5C A2  F2  03EC  1137          AOBLSS  ADP$L_MRACTMDRS(R2),R5,20$      ; Loop thru all extents.
                     03F1  1138 30$:                                                ; Failure if we fall thru.
            50  D4  03F1  1139          CLRL    R0                              ; Set failure code.
            52  11  03F3  1140          BRB     80$                             ; And branch to return.
                     03F5  1141 40$:
                     03F5  1142
                     03F5  1143 ; Here the first register we want is greater than the first register of
```

```
                                                     N 4
                     03F5  1144 ;       current extent (defined by R5 = index) and is less than or equal to
                     03F5  1145 ;       the last register of the extent. R0 contains the # of the register just
                     03F5  1146 ;       beyond the current extent.  In other words,
                     03F5  1147 ;
                     03F5  1148 ;       ADP$W_MRFREGARY(R2)[R5] < R4 < R0
                     03F5  1149
          50   54   A2 03F5  1150       SUBW    R4,R0                   ; R0 = length of subextent based at R4.
          53   50   B1 03F8  1151       CMPW    R0,R3                   ; Compare to # of registers needed.
               F4   19 03FB  1152       BLSS    30$                     ; LSS means failure.
                    03FD  1153
          61   54   B0 03FD  1154       MOVW    R4,UBMD$W_MAPREG(R1)    ; Success. Fill in user's descriptor
       02 A1   53   90 0400  1155       MOVB    R3,UBMD$B_NUMREG(R1)    ;  with base register and # of registers.
                    0404  1156
                    0404  1157 ;       SUBW3   ADP$W_MRFREGARY(R2)[R5],R4,-    ; Distance from beginning of
                    0404  1158 ;               ADP$W_MRNREGARY(R2)[R5]         ;  extent to R4 is new length.
    64 A245   50   A2 0404  1159       SUBW    R0,ADP$W_MRNREGARY(R2)[R5]      ; Equivalent result.
                    0409  1160
          50   53   A2 0409  1161       SUBW    R3,R0                   ; R0 = # regs. left in sub-extent.
               36   13 040C  1162       BEQL    70$                     ; EQL means we do not have to allocate
                    040E  1163                                           ;  and fill a new extent descriptor.
               55   D6 040E  1164       INCL    R5                      ; R5 = index of new extent descriptor.
          7E   50   B0 0410  1165       MOVW    R0,-(SP)                ; Save length of new extent.
             00C9   30 0413  1166       BSBW    ALLOC_DESCRIP           ; Call to allocate a new descriptor.
                    0416  1167
 015E C245   53   54   A1 0416  1168       ADDW3   R4,R3,ADP$W_MRFREGARY(R2)[R5]   ; Fill in new descriptor with
    64 A245   8E   B0 041D  1169       MOVW    (SP)+,ADP$W_MRNREGARY(R2)[R5]   ;  1st register and # registers.
               20   11 0422  1170       BRB     70$                     ; Branch around to success.
                    0424  1171 50$:
                    0424  1172
                    0424  1173 ; Here the first register we want is equal to the first register of the current
                    0424  1174 ;       extent (defined by index register R5).  In other words,
                    0424  1175 ;
                    0424  1176 ;       R4 = ADP$W_MRFREGARY(R2)[R5]
                    0424  1177
    64 A245   53   B1 0424  1178       CMPW    R3,ADP$W_MRNREGARY(R2)[R5]      ; See if we have enough registers.
               C6   14 0429  1179       BGTR    30$                     ; GTR implies failure.
                    042B  1180
          61   54   B0 042B  1181       MOVW    R4,UBMD$W_MAPREG(R1)    ; Success.  Fill in user's descriptor
       02 A1   53   B0 042E  1182       MOVW    R3,UBMD$B_NUMREG(R1)    ;  with 1st register and # allocated.
                    0432  1183
    64 A245   53   A2 0432  1184       SUBW    R3,ADP$W_MRNREGARY(R2)[R5]      ; Update current descriptor.
               08   13 0437  1185       BEQL    60$                     ; EQL means current extent now
                    0439  1186                                           ;  empty. Go to deallocate.
 015E C245   53   A0 0439  1187       ADDW    R3,ADP$W_MRFREGARY(R2)[R5]      ; If not empty, update 1st register.
               03   11 043F  1188       BRB     70$                     ; Branch around deallocate.
                    0441  1189 60$:
             0082   30 0441  1190       BSBW    DEALLOC_DESCRIP         ; Deallocate system descriptor.
          50   01   D0 0444  1191 70$:  MOVL    S^#SS$_NORMAL,R0        ; Set success indicator.
             55 8ED0 0447  1192 80$:  POPL    R5                      ; Restore R5,R4,R3
          53   8E   7D 044A  1193       MOVQ    (SP)+,R3                ;
                    05 044D  1194       RSB                             ; And return to caller.
```

```
                                                B 5
                                044E  1196          .SBTTL  Permanently Allocate UNIBUS Map Registers
                                044E  1197 ;+
                                044E  1198 ; IOC$ALOUBAMAPRM  - Permanently Allocate UBA Map Registers (CRB Database Specified)
                                044E  1199 ; IOC$ALOUBAMAPRMN - Permanently Allocate UBA Map Registers (Argument Specified)
                                044E  1200 ;
                                044E  1201 ; This routine is called to permanently allocate UNIBUS map registers.
                                044E  1202 ;       Here we allocate the map registers from the highest numbered
                                044E  1203 ;       available registers.
                                044E  1204 ;
                                044E  1205 ; INPUTS:
                                044E  1206 ;       R3 =  # Registers to allocate (IOC$ALOUBAMAPRMN only)
                                044E  1207 ;       R5 => UCB
                                044E  1208 ;
                                044E  1209 ; OUTPUTS:
                                044E  1210 ;       R0 = Success indication
                                044E  1211 ;
                                044E  1212 ;-
                                044E  1213
                                044E  1214          .enabl  LSB
                                044E  1215 IOC$ALOUBMAPRMN::                        ;ALLOCATE UBA MAP REGISTERS ARGUMENT SPECIFI
          7E    53    7D      044E  1216          MOVQ    R3,-(SP)                  ; Save R3,R4,R5
                55    DD      0451  1217          PUSHL   R5                        ;
                              0453  1218
                18    11      0453  1219          BRB     5$                        ;
                              0455  1220 IOC$ALOUBMAPRM::                         ;ALLOCATE UBA MAP REGISTERS CRB SPECIFIED
          7E    53    7D      0455  1221          MOVQ    R3,-(SP)                  ; Save R3,R4,R5
                55    DD      0458  1222          PUSHL   R5                        ;
                              045A  1223
       53   7E A5    3C      045A  1224          MOVZWL  UCB$W_BCNT(R5),R3         ;GET TRANSFER BYTE COUNT
       54   7C A5    3C      045E  1225          MOVZWL  UCB$W_BOFF(R5),R4         ;GET BYTE OFFSET IN PAGE
 53   03FF C344    9E      0462  1226          MOVAB   ^X3FF(R3)[R4],R3          ;CALCULATE HIGHEST RELATIVE BYTE AND ROUND
53   53   F7 8F    78      0468  1227          ASHL    #-9,R3,R3                 ;CALCULATE NUMBER OF MAP REGISTERS REQUIRED
                              046D  1228 5$:
       51   24 A5    D0      046D  1229          MOVL    UCB$L_CRB(R5),R1                  ; R1 => CRB
       52   38 A1    D0      0471  1230          MOVL    CRB$L_INTD+VEC$L_ADP(R1),R2       ; R2 => ADP
       51   34 A1    9E      0475  1231          MOVAB   CRB$L_INTD+VEC$W_MAPREG(R1),R1    ; R1 => UBMD.
                0F    E0      0479  1232          BBS     #VEC$V_MAPLOCK,-                  ; If SET, already permanently
                38 61         047B  1233                  UBMD$W_MAPREG(R1),30$             ;  allocated, so branch around.
                              047D  1234
                53    D6      047D  1235          INCL    R3                        ; Round up request to next multiple
             53   01    8A      047F  1236          BICB    #1,R3                     ;  of 2.
       55    5C A2    D0      0482  1237          MOVL    ADP$L_MRACTMDRS(R2),R5    ; R5 = index beyond last MRD.
                0A    13      0486  1238          BEQL    15$                       ; EQL implies no registers available.
                              0488  1239 10$:
          62 A245    53    B1      0488  1240          CMPW    R3,ADP$W_MRNREGARY-2(R2)[R5]      ; See if enough regs described here.
                07    15      048D  1241          BLEQ    20$                               ; LEQ implies YES.
                              048F  1242
                F6 55    F5      048F  1243          SOBGTR  R5,10$                    ; Else branch back and continue
                              0492  1244 15$:
                50    D4      0492  1245          CLRL    R0                        ; If here, allocation failure.
                22    11      0494  1246          BRB     40$                       ; Branch around to return.
                              0496  1247 20$:
50   62 A245    015C C245    A1      0496  1248          ADDW3   ADP$W_MRFREGARY-2(R2)[R5],-        ; Calculate register # beyond
                              049F  1249                  ADP$W_MRNREGARY-2(R2)[R5],R0      ;  last extent.
             50   53    A2      049F  1250          SUBW    R3,R0                     ; We allocate from high end. R0
                              04A2  1251                                            ;  contains 1st reg. to alloc.
             61   50    B0      04A2  1252          MOVW    R0,UBMD$W_MAPREG(R1)      ; Record 1st register allocated.
```

```
                                                C 5
61   8000 8F   A8  04A5  1253          BISW    #VEC$M_MAPLOCK,UBMD$W_MAPREG(R1);  and mark it permanent.
  02 A1   53   90  04AA  1254          MOVB    R3,UBMD$B_NUMREG(R1)               ; Set # of map regs allocated.
62 A245   53   A2  04AE  1255          SUBW    R3,ADP$W_MRNREGARY-2(R2)[R5]       ; Subtract out # regs allocated.
          0A   13  04B3  1256          BEQL    50$                                ; EQL implies descriptor not
               04B5  1257                                                         ;  valid; branch to deallocate.
               04B5  1258 30$:
     50   01   D0  04B5  1259          MOVL    S^#SS$_NORMAL,R0           ; Indicate success.
               04B8  1260 40$:
          55 8ED0  04B8  1261          POPL    R5                         ; Restore R5,R4,R3
     53   8E   7D  04BB  1262          MOVQ    (SP)+,R3                   ;
               05  04BE  1263          RSB
               04BF  1264 50$:
          55   D7  04BF  1265          DECL    R5                         ; R5 = index of descriptor to dealloc.
        0002   30  04C1  1266          BSBW    DEALLOC_DESCRIP            ; Call to deallocate descriptor.
          EF   11  04C4  1267          BRB     30$                        ;  And branch back to return.
               04C6  1268          .dsabl  lsb
```

```
                                                     D 5
                                04C6  1270 ;+
                                04C6  1271 ; DEALLOC_DESCRIP - Common internal subroutine called to deallocate
                                04C6  1272 ;       a UBA Map Register descriptor.
                                04C6  1273 ;
                                04C6  1274 ; INPUTS:
                                04C6  1275 ;       R2 => ADP
                                04C6  1276 ;       R5 =  index of descriptor to deallocate.
                                04C6  1277 ; OUTPUTS:
                                04C6  1278 ;       The UBA Map Allocation structures are updated by contracting
                                04C6  1279 ;             descriptors over the deallocated one.
                                04C6  1280 ;       Register R5 is modified.
                                04C6  1281 ;-
                                04C6  1282
                                04C6  1283 DEALLOC_DESCRIP:
                     5C A2   D7 04C6  1284         DECL    ADP$L_MRACTMDRS(R2)     ; Decrement # active descriptors.
                                04C9  1285 10$:
          64 A245   66 A245   B0 04C9  1286         MOVW    ADP$W_MRNREGARY+2(R2)[R5],-     ; Move data towards lower index
                                04D0  1287                 ADP$W_MRNREGARY(R2)[R5]         ;  to fill up hole.
      015E C245   0160 C245   B0 04D0  1288         MOVW    ADP$W_MRFREGARY+2(R2)[R5],-     ;
                                04D9  1289                 ADP$W_MRFREGARY(R2)[R5]         ;
                EB 55   5C A2   F2 04D9  1290         AOBLSS  ADP$L_MRACTMDRS(R2),R5,10$      ; Loop thru rest of active MDRS.
                                05 04DE  1291         RSB
                                04DF  1292
                                04DF  1293 ;+
                                04DF  1294 ; ALLOC_DESCRIP - Common internal subroutine to allocate a UBA map register
                                04DF  1295 ;       descriptor in the middle of the range of descriptors.
                                04DF  1296 ;
                                04DF  1297 ; INPUTS:
                                04DF  1298 ;       R2 => ADP
                                04DF  1299 ;       R5 =  index of where we must allocate descriptor
                                04DF  1300 ; OUTPUTS:
                                04DF  1301 ;       Allocation is accomplished by creating a hole in each of the arrays
                                04DF  1302 ;             by moving descriptor items to the next higher element.
                                04DF  1303 ;       Note R0 is modified.
                                04DF  1304 ;-
                                04DF  1305
                                04DF  1306 ALLOC_DESCRIP:
                     50  5C A2  D0 04DF  1307         MOVL    ADP$L_MRACTMDRS(R2),R0          ; R0 = # active descriptors.
                                04E3  1308 10$:
                        55  50  D1 04E3  1309         CMPL    R0,R5                           ; Have we finished?
                            13  15 04E6  1310         BLEQ    20$                             ; LEQ implies YES.
          64 A240   62 A240   B0 04E8  1311         MOVW    ADP$W_MRNREGARY-2(R2)[R0],-     ; Starting from ends of arrays,
                                04EF  1312                 ADP$W_MRNREGARY(R2)[R0]         ;  copy # register items.
                                04EF  1313
      015E C240   015C C240   B0 04EF  1314         MOVW    ADP$W_MRFREGARY-2(R2)[R0],-     ;
                                04F8  1315                 ADP$W_MRFREGARY(R2)[R0]         ;
                        E8 50   F5 04F8  1316         SOBGTR  R0,10$                          ; And loop back until we reach
                                04FB  1317                                                 ;  the hole we have created.
                     5C A2   D6 04FB  1318 20$:     INCL    ADP$L_MRACTMDRS(R2)             ; Increment # active descriptors.
                                05 04FE  1319         RSB                                     ; Return to caller
```

```
                                        .SBTTL  Release UNIBUS Map Registers
                                ;+
                                ; IOC$RELMAPUDA - RELEASE UNIBUS MAP REGISTERS (CALLED FROM UDA PORT DRIVER)
                                ; IOC$RELMAPREG - RELEASE UNIBUS MAP REGISTERS
                                ;
                                ; This routine is called to release UNIBUS map registers that were previously
                                ; assigned for an I/O transfer.
                                ;
                                ; INPUTS:
                                ;   (For IOC$RELMAPUDA only)
                                ;
                                ;       R4 => PDT
                                ;       R5 => CDRP
                                ;
                                ;   (For IOC$RELMAPREG call only)
                                ;
                                ;       R5 = UCB ADDRESS OF DEVICE UNIT.
                                ;
                                ;       It is assumed that the caller still owns the I/O channel on which
                                ;       the transfer took place.
                                ;
                                ; OUTPUTS:
                                ;
                                ;       If the mapping registers have been permanently assigned to the asso-
                                ;       ciated I/O channel (only possible for IOC$RELMAPREG), then control
                                ;       is immediately returned to the caller.  Else the mapping registers are
                                ;       released (via a call to IOC$DALOCUBAMAP) and we then go into a loop
                                ;       removing waiting driver processes from the Map Register Wait Queue
                                ;       until either the Queue is completely drained or we run out of map
                                ;       registers to satisfy the needs of a given waiting driver process.
                                ;       Driver processes waiting here have their context stored in either
                                ;       a UCB fork block or a CDRP fork block and the processing required to
                                ;       resume each of these types of driver process is slightly different.
                                ;       What is done for each is to allocate the required map registers
                                ;       (via a call to IOC$ALOUBAMPA for UCB threads and via a call to
                                ;       IOC$ALOUBAMAP for CDRP threads) and to resume the waiting driver
                                ;       process.  Resuming a UCB thread is done by restoring register
                                ;       context and JSB'ing to the saved PC.  Resuming a CDRP thread is
                                ;       accomplished by calling SCS$RESUMEWAITR.
                                ;-
                                        .enabl  lsb
                                IOC$RELMAPUDA::
      7E    53    7D                  MOVQ    R3,-(SP)                ; Save R3-R6
      7E    55    7D                  MOVQ    R5,-(SP)                ;
                                ;
52    00E0  C4    D0                  MOVL    PDT$L_ADP(R4),R2        ; R2 => ADP.
      56    52    D0                  MOVL    R2,R6                   ; R6 => ADP also.
                                ;
  53    3C  A5    9E                  MOVAB   CDRP$L_UBARSRCE(R5),R3  ; R3 => UBMD.
      54    63    3C                  MOVZWL  UBMD$W_MAPREG(R3),R4    ; R4 has 1st mapreg #.
  53    02  A3    9A                  MOVZBL  UBMD$B_NUMREG(R3),R3    ; R3 has # of mapregs.
            1E    11                  BRB     10$                     ; Branch to common code.
                                ;
                                IOC$RELMAPREG::                         ; Release unibus map registers
  51    24  A5    D0                  MOVL    UCB$L_CRB(R5),R1        ; R1 => CRB.
            0F    E0                  BBS     #VEC$V_MAPLOCK,-        ; If SET, permanent allocation so branch.
        3D  34  A1                              CRB$L_INTD+VEC$W_MAPREG(R1),50$
```

```
                                                F 5
                     7E    53   7D  0523  1378          MOVQ    R3,-(SP)                ; Save R3-R6
                     7E    55   7D  0526  1379          MOVQ    R5,-(SP)                ;
                                 0529  1380
                  52   38 A1   D0  0529  1381          MOVL    CRB$L_INTD+VEC$L_ADP(R1),R2 ;GET ADDRESS OF ADP
                     56    52   D0  052D  1382          MOVL    R2,R6                   ;SAVE ADDRESS OF ADP
                  54   34 A1   3C  0530  1383          MOVZWL  CRB$L_INTD+VEC$W_MAPREG(R1),R4 ;GET STARTING MAP REGISTER NUMBER
                  53   36 A1   9A  0534  1384          MOVZBL  CRB$L_INTD+VEC$B_NUMREG(R1),R3 ;GET NUMBER OF REGISTERS TO DEALLOC
                                 0538  1385 10$:
                        0038   30  0538  1386          BSBW    IOC$DALOCUBAMAP         ; Free up UBA map resources.
                                 053B  1387 20$:
                  55   30 B6   0F  053B  1388          REMQUE  @ADP$L_MRQFL(R6),R5     ;GET ADDRESS OF NEXT DRIVER FORK BLOCK
                        19   1D  053F  1389          BVS     40$                     ;IF VS NO DRIVER PROCESS WAITING
                                 0541  1390
                        10   91  0541  1391          CMPB    #DYN$C_UCB,-            ; See if we dequeued a UCB or a CDRP.
                     0A A5        0543  1392                  UCB$B_TYPE(R5)
                        1A   12  0545  1393          BNEQ    REALLOC_CD_MAPREGS      ; NEQ implies a CDRP.
                                 0547  1394
                      FDFB   30  0547  1395          BSBW    IOC$ALOUBAMAP           ;SEARCH MAP REGISTER BITMAP AND ALLOCATE
                     09 50   E9  054A  1396          BLBC    R0,30$                  ;IF LBC ALLOCATION FAILURE
                  53   10 A5   7D  054D  1397          MOVQ    UCB$L_FR3(R5),R3        ;RESTORE DRIVE PROCESS CONTEXT
                     0C B5   16  0551  1398          JSB     @UCB$L_FPC(R5)          ;CALL DRIVER AT MAP REGISTER WAIT RETURN ADD
                        E5   11  0554  1399          BRB     20$                     ;
               30 A6    65   0E  0556  1400 30$:     INSQUE  UCB$L_FQFL(R5),ADP$L_MRQFL(R6) ;REINSERT DRIVER PROCESS AT FRONT OF
                  55    8E   7D  055A  1401 40$:     MOVQ    (SP)+,R5                ; Restore R3-R6
                  53    8E   7D  055D  1402          MOVQ    (SP)+,R3                ;
                        05   05  0560  1403 50$:     RSB                             ;
                                 0561  1404
                                 0561  1405 REALLOC_CD_MAPREGS:                      ; Reallocate mapregs to a class driver
                                 0561  1406                                         ;  process.
                  54   14 A5   D0  0561  1407          MOVL    CDRP$L_FR4(R5),R4       ; Restore saved fork register.
                       FDB3   30  0565  1408          BSBW    IOC$ALOMAPUDA           ; Allocate map registers if we can.
                     EB 50    E9  0568  1409          BLBC    R0,30$                  ; LBC implies allocation failure, branch
                                 056B  1410
          00000000'EF    16   056B  1411          JSB     SCS$RESUMEWAITR         ; Resume waiting thread and any backed
                                 0571  1412                                         ;  up IRP's.
                        C8   11  0571  1413          BRB     20$                     ; Branch back to try and allocate more
                                 0573  1414                                         ;  UNIBUS map registers.
                                 0573  1415          .dsabl  lsb
```

```
                     0573  1417 ;+
                     0573  1418 ; IOC$DALOCUBAMAP - Common internal subroutine to update the UBA Map allocation
                     0573  1419 ;       structures to include the map registers specified here among the
                     0573  1420 ;       available map registers.
                     0573  1421 ;
                     0573  1422 ; INPUTS:
                     0573  1423 ;       R2 => ADP
                     0573  1424 ;       R3 =  # map registers to free.
                     0573  1425 ;       R4 =  first map register to free.
                     0573  1426 ;
                     0573  1427 ; OUTPUTS:
                     0573  1428 ;       The UBA Map Allocation structures are updated.
                     0573  1429 ;
                     0573  1430 ;       Registers R0, R1 and R5 are modified.
                     0573  1431 ;
                     0573  1432 ;-
                     0573  1433
                     0573  1434 IOC$DALOCUBAMAP:
          55   D4   0573  1435         CLRL    R5                              ; Initialize loop variable.
51   53   54   C1   0575  1436         ADDL3   R4,R3,R1                        ; R1 = map register beyond extent.
          53   D5   0579  1437         TSTL    R3                              ; Is the # of regs. to deallocate zero?
          62   13   057B  1438         BEQL    90$                             ; Branch to bugcheck if zero.
        5C A2  D5   057D  1439         TSTL    ADP$L_MRACTMDRS(R2)             ; Test for zero active descriptors.
          4E   13   0580  1440         BEQL    50$                             ; EQL implies no registers available.
                     0582  1441 10$:
015E C245    51   B1   0582  1442         CMPW    R1,ADP$W_MRFREGARY(R2)[R5]      ; See if map registers to free
                     0588  1443                                                 ;  are before those described
                     0588  1444                                                 ;  by current descriptor.
          07   15   0588  1445         BLEQ    20$                             ; LEQ implies yes.
                     058A  1446
 F3 55    5C A2   F2   058A  1447         AOBLSS  ADP$L_MRACTMDRS(R2),R5,10$      ; Else branch back and try next.
          2B   11   058F  1448         BRB     40$                             ; If here, registers to free
                     0591  1449                                                 ;  beyond those described by
                     0591  1450                                                 ;  last descriptor. So branch
                     0591  1451                                                 ;  to try and absorb at end of
                     0591  1452                                                 ;  last descriptor.
                     0591  1453 20$:
          29   12   0591  1454         BNEQ    40$                     ; NEQ implies that although we alloca-
                     0593  1455                                         ;  registers before the current des-
                     0593  1456                                         ;  criptor, we are not contiguous with
                     0593  1457                                         ;  it.  So we branch to try and absorb
                     0593  1458                                         ;  these registers in the previous one.
                     0593  1459
                     0593  1460 ; Here we can absorb the registers in the current descriptor.
                     0593  1461
50   015C C245  62 A245  A1   0593  1462         ADDW3   ADP$W_MRNREGARY-2(R2)[R5],-     ; Calculate end of previous
                     059C  1463                 ADP$W_MRFREGARY-2(R2)[R5],R0     ;  extent and move to R0.
          54   50   B1   059C  1464         CMPW    R0,R4                           ; Does it coincide with start
                     059F  1465                                                 ;  of this extent?
          0C   13   059F  1466         BEQL    30$                             ; EQL implies yes.
                     05A1  1467
                     05A1  1468 ; Here we have the most likely case. The map registers that we are freeing can
                     05A1  1469 ;       be absorbed into the top of the current descriptor but not also in the
                     05A1  1470 ;       previous descriptor.
                     05A1  1471
     015E C245   54   B0   05A1  1472         MOVW    R4,ADP$W_MRFREGARY(R2)[R5]      ; First register freed becomes
                     05A7  1473                                                 ;  first register of current
```

```
                                                H 5
                     05A7  1474                                             ;  descriptor.
          64 A245    53   A0 05A7  1475          ADDW    R3,ADP$W_MRNREGARY(R2)[R5]      ; Number of registers is sum of
                     05AC  1476                                             ;  registers freed and registers
                     05AC  1477                                             ;  previously described here.
                          05 05AC  1478          RSB
                     05AD  1479
                     05AD  1480 ; Here we have the case where the map registers being freed fall between two
                     05AD  1481 ;       discontiguous blocks and exactly span the difference.  We then can
                     05AD  1482 ;       describe the entire group with one descriptor, and so we also
                     05AD  1483 ;       deallocate the current descriptor.  Note new combined descriptor
                     05AD  1484 ;       will still begin at same map register number so we do NOT alter
                     05AD  1485 ;       this item.
                     05AD  1486
                     05AD  1487 30$:
          62 A245    53   A0 05AD  1488          ADDW    R3,ADP$W_MRNREGARY-2(R2)[R5]    ; Partial sum of registers
                     05B2  1489                                             ;  being freed and previous ones.
     62 A245   64 A245   A0 05B2  1490          ADDW    ADP$W_MRNREGARY(R2)[R5],-       ; Now add in registers described
                     05B9  1491                         ADP$W_MRNREGARY-2(R2)[R5]       ;  in current descriptor.
                     05B9  1492
               FF0A   31   05B9  1493          BRW     DEALLOC_DESCRIP                 ; BRW to subroutine and let it
                     05BC  1494                                             ;  return to our caller.
                     05BC  1495
                     05BC  1496 ; Here we cannot absorb the freed map registers in the current descriptor.
                     05BC  1497 ;       We test to see if we can absorb them in the previous descriptor.
                     05BC  1498
                     05BC  1499 40$:
50   015C C245   62 A245   A1 05BC  1500          ADDW3   ADP$W_MRNREGARY-2(R2)[R5],-     ; Calculate end of previous
                     05C5  1501                         ADP$W_MRFREGARY-2(R2)[R5],R0     ;  extent and move to R0.
                    54   50 B1 05C5  1502          CMPW    R0,R4                           ; See if contigous with previous.
                       06   12 05C8  1503          BNEQ    50$                             ; NEQ implies NO.
                     05CA  1504
          62 A245    53   A0 05CA  1505          ADDW    R3,ADP$W_MRNREGARY-2(R2)[R5]    ; Sum # of registers in extent.
                          05 05CF  1506          RSB
                     05D0  1507
                     05D0  1508 ; Here we must allocate a new descriptor to describe the map registers we
                     05D0  1509 ;       are freeing.  Conditions at this time are as follows:
                     05D0  1510 ;
                     05D0  1511 ;               R2 => ADP
                     05D0  1512 ;               R3 =  # registers to free
                     05D0  1513 ;               R4 =  first register to free
                     05D0  1514 ;               R5 =  index of where we must allocate descriptor
                     05D0  1515 ;
                     05D0  1516 ;       Allocation is accomplished by calling subroutine ALLOC_DESCRIP
                     05D0  1517 ;
                     05D0  1518
                     05D0  1519 50$:
               FF0C   30   05D0  1520          BSBW    ALLOC_DESCRIP                   ; Alloc R5 = index of descriptor.
          64 A245    53   B0 05D3  1521          MOVW    R3,ADP$W_MRNREGARY(R2)[R5]      ; Fill in allocated descriptor.
     015E C245    54   B0 05D8  1522          MOVW    R4,ADP$W_MRFREGARY(R2)[R5]      ;
                          05 05DE  1523          RSB
                     05DF  1524
                     05DF  1525 90$:    BUG_CHECK INCONSTATE                    ; Non-fatal bugcheck on zero map
                     05E3  1526                                             ; registers deallocation attempts.
                          05 05E3  1527          RSB                             ; Then ignore deallocate request.
```

```
                        .SBTTL  RETURN TO CALLER
                ;+
                ; IOC$RETURN - RETURN TO CALLER
                ;
                ; THIS ROUTINE IS CALLED AS A RESULT OF A DDT DISPATCH TO A NULL ENTRY. ITS
                ; FUNCTION IS MERELY TO RETURN TO ITS CALLER.
                ;
                ; INPUTS:
                ;
                ;       NONE.
                ;
                ; OUTPUTS:
                ;
                ;       NONE.
                ;-
                
05E4  1545 IOC$RETURN::                         ;RETURN TO CALLER
05  05E4  1546         RSB                     ;
```

```
                                05E5  1548          .SBTTL  WAITFOR INTERRUPT OR TIMEOUT AND KEEP CHANNEL
                                05E5  1549 ;+
                                05E5  1550 ; IOC$WFIKPCH - WAITFOR INTERRUPT OR TIMEOUT AND KEEP CHANNEL
                                05E5  1551 ;
                                05E5  1552 ; THIS ROUTINE IS CALLED TO SOFTWARE ENABLE INTERRUPTS AND TIMEOUT ON
                                05E5  1553 ; A DEVICE UNIT AND TO KEEP THE CHANNEL. THIS ROUTINE CAN BE CALLED AT
                                05E5  1554 ; EITHER FORK OR DEVICE INTERRUPT LEVEL.
                                05E5  1555 ;
                                05E5  1556 ; INPUTS:
                                05E5  1557 ;
                                05E5  1558 ;       00(SP) = RETURN ADDRESS OF CALLER.
                                05E5  1559 ;       04(SP) = TIMEOUT VALUE IN SECONDS.
                                05E5  1560 ;       08(SP) = IPL TO LOWER TO AFTER SETTING WAIT.
                                05E5  1561 ;       12(SP) = RETURN ADDRESS OF CALLER'S CALLER.
                                05E5  1562 ;
                                05E5  1563 ;       R5 = UCB ADDRESS OF DEVICE UNIT.
                                05E5  1564 ;
                                05E5  1565 ; OUTPUTS:
                                05E5  1566 ;
                                05E5  1567 ;       THE TIMEOUT VALUE IS COMPUTED AND STORED IN DUE TIME, REGISTERS R3 AND
                                05E5  1568 ;       R4 ALONG WITH THE RETURN PC ARE SAVED IN THE FORK BLOCK, INTERRUPTS AND
                                05E5  1569 ;       TIMEOUT ARE ENABLED, AND A RETURN TO THE CALLER'S CALLER IS EXECUTED.
                                05E5  1570 ;-
                                05E5  1571
                                05E5  1572 IOC$WFIKPCH::                                ;WAITFOR INTERRUPT/TIMEOUT AND KEEP CHANNEL
             6E    02    C0   05E5  1573          ADDL    #2,(SP)                         ;CALCULATE OFFSET TO NORMAL RETURN
          10 A5    53    7D   05E8  1574          MOVQ    R3,UCB$L_FR3(R5)                ;SAVE REGISTERS R3 AND R4
             0C A5  8ED0   05EC  1575          POPL    UCB$L_FPC(R5)                   ;SAVE INTERRUPT RETURN ADDRESS
          64 A5    03    A8   05F0  1576          BISW    #UCB$M_INT!UCB$M_TIM,UCB$W_STS(R5) ;ENABLE INTERRUPT AND TIMEOUT
6C A5  00000000'EF    8E    C1   05F4  1577          ADDL3   (SP)+,G^EXE$GL_ABSTIM,UCB$L_DUETIM(R5) ;SET TIMEOUT TIME
          64 A5  0040 8F    AA   05FD  1578          BICW    #UCB$M_TIMOUT,UCB$W_STS(R5) ;CLEAR UNIT TIMED OUT
                                0603  1579          ENBINT                          ;ENABLE INTERRUPTS
                          05    0606  1580          RSB                             ;
```

```
                                                    K 5
                     0607  1582                .SBTTL  WAITFOR INTERRUPT OR TIMEOUT AND RELEASE CHANNEL
                     0607  1583 ;+
                     0607  1584 ; IOC$WFIRLCH - WAITFOR INTERRUPT OR TIMEOUT AND RELEASE CHANNEL
                     0607  1585 ;
                     0607  1586 ; THIS ROUTINE IS CALLED TO SOFTWARE ENABLE INTERRUPTS AND TIMEOUT ON A DEVICE
                     0607  1587 ; UNIT AND TO RELEASE THE CHANNEL. THIS ROUTINE CAN ONLY BE CALLED AT FORK LEVEL.
                     0607  1588 ;
                     0607  1589 ; INPUTS:
                     0607  1590 ;
                     0607  1591 ;       00(SP) = RETURN ADDRESS OF CALLER.
                     0607  1592 ;       04(SP) = TIMEOUT VALUE IN SECONDS.
                     0607  1593 ;       08(SP) = IPL TO LOWER TO AFTER SETTING WAIT.
                     0607  1594 ;       12(SP) = RETURN ADDRESS OF CALLER'S CALLER.
                     0607  1595 ;
                     0607  1596 ;       R5 = UCB ADDRESS OF DEVICE UNIT.
                     0607  1597 ;
                     0607  1598 ; OUTPUTS:
                     0607  1599 ;
                     0607  1600 ;       THE TIMEOUT VALUE IS COMPUTED AND STORED IN DUE TIME, REGISTERS R3 AND
                     0607  1601 ;       R4 ALONG WITH THE RETURN PC ARE SAVED IN THE FORK BLOCK, INTERRUPTS AND
                     0607  1602 ;       TIMEOUT ARE ENABLED, THE CHANNEL IS RELEASED, AND A RETURN TO THE CALLER'S
                     0607  1603 ;       CALLER IS EXECUTED.
                     0607  1604 ;-
                     0607  1605
                     0607  1606 IOC$WFIRLCH::                                ;WAITFOR INTERRUPT/TIMEOUT AND RELEASE CHANN
          6E   02    C0  0607  1607          ADDL    #2,(SP)                 ;CALCULATE OFFSET TO NORMAL RETURN
        10 A5  53    7D  060A  1608          MOVQ    R3,UCB$L_FR3(R5)        ;SAVE REGISTERS R3 AND R4
           0C A5 8ED0  060E  1609          POPL    UCB$L_FPC(R5)           ;SAVE INTERRUPT RETURN ADDRESS
        64 A5  03    A8  0612  1610          BISW    #UCB$M_INT!UCB$M_TIM,UCB$W_STS(R5) ;ENABLE INTERRUPT AND TIMEOUT
6C A5   00000000'EF  8E  C1  0616  1611          ADDL3   (SP)+,@#EXE$GL_ABSTIM,UCB$L_DUETIM(R5) ;SET TIMEOUT TIME
        64 A5  0040  8F  AA  061F  1612          BICW    #UCB$M_TIMOUT,UCB$W_STS(R5) ;CLEAR UNIT TIMED OUT
                             0625  1613          ENBINT                          ;ENABLE INTERRUPTS
                FA5F    31  0628  1614          BRW     IOC$RELCHAN             ;RELEASE ALL CHANNELS AND RETURN TO CALLER
                             062B  1615
                             062B  1616
```

```
                                                L 5
                     062B  1618          .SBTTL  ALLOCATE SYSTEM PAGE TABLE
                     062B  1619 ;+
                     062B  1620 ; IOC$ALLOSPT - ALLOCATE SYSTEM PAGE TABLE
                     062B  1621 ;
                     062B  1622 ; THIS ROUTINE ALLOCATES SYSTEM PAGE TABLE (SPT) ENTRIES.
                     062B  1623 ;
                     062B  1624 ; INPUTS:
                     062B  1625 ;
                     062B  1626 ;       R1 = NUMBER OF SPT ENTRIES TO BE ALLOCATED
                     062B  1627 ;
                     062B  1628 ;       BOO$GL_SPTFREL = LOWEST FREE VPN
                     062B  1629 ;       BOO$GL_SPTFREH = HIGHEST FREE VPN
                     062B  1630 ;
                     062B  1631 ;       IT IS ASSUMED THAT THE CALLER IS RUNNING AT IPL$_SYNCH.
                     062B  1632 ;
                     062B  1633 ; OUTPUTS:
                     062B  1634 ;
                     062B  1635 ;       R0 = SUCCESS INDICATION.
                     062B  1636 ;       R2 = STARTING PAGE NUMBER ALLOCATED (SVPN).
                     062B  1637 ;       R3 = ADDRESS OF BASE OF SYSTEM PAGE TABLE (MMG$GL_SPTBASE).
                     062B  1638 ;
                     062B  1639 ;       R1 IS PRESERVED ACROSS CALL.
                     062B  1640 ;-
                     062B  1641 IOC$ALLOSPT::                           ;ALLOCATE SYSTEM PAGE TABLE
           50    D4  062B  1642          CLRL    R0                    ;ASSUME FAILURE
52 00000000'EF    D0  062D  1643          MOVL    L^BOO$GL_SPTFREL,R2   ;GET NEXT AVAILABLE SYSTEM VPN
   53    52    51  C1  0634  1644          ADDL3   R1,R2,R3              ;COMPUTE NEXT WITH THIS ALLOCATION
00000000'EF    53  D1  0638  1645          CMPL    R3,L^BOO$GL_SPTFREH   ;ARE THERE ENOUGH AVAILABLE?
           10    1E  063F  1646          BGEQU   10$                   ;BR IF NO
00000000'EF    53  D0  0641  1647          MOVL    R3,L^BOO$GL_SPTFREL   ;MARK THE ENTRIES ALLOCATED
53 00000000'EF    D0  0648  1648          MOVL    L^MMG$GL_SPTBASE,R3   ;GET ADDR OF BASE OF SPT
           50    D6  064F  1649          INCL    R0                    ;SET SUCCESS
                     0651  1650 10$:
           05        0651  1651          RSB                           ;
```

```
                                M 5
0652  1653          .SBTTL  CONVERT DEVICE NAME AND UNIT
0652  1654 ;+
0652  1655 ; IOC$CVT_DEVNAM - Convert device name and unit
0652  1656 ;
0652  1657 ; This routine is called to convert a device name and unit number to a physical
0652  1658 ; device name string.
0652  1659 ;
0652  1660 ; Inputs:
0652  1661 ;
0652  1662 ;       The caller is assumed to have PROBEd the output buffer for write access.
0652  1663 ;       The I/O data base is locked for read access.  This means that the caller
0652  1664 ;       owns the I/O data base mutex and/or is at IPL SYNCH or higher.
0652  1665 ;
0652  1666 ;       R0 = length of output buffer.
0652  1667 ;       R1 = address of output buffer.
0652  1668 ;       R4 = name string formation mode, one of:
0652  1669 ;           -1 (DVI$_DEVNAM) -- a name suitable for displays
0652  1670 ;               for non-local devices, return node$ddcn
0652  1671 ;               for local devices:
0652  1672 ;                   if in cluster and file oriented device, return node$ddcn
0652  1673 ;                   otherwise, return ddcn
0652  1674 ;               0 (DVI$_FULLDEVNAM) -- a name with appropriate node information
0652  1675 ;                 if allocation class not zero and file oriented device, return
0652  1676 ;                       $alloclass$ddcn
0652  1677 ;                 otherwise, return node$ddcn
0652  1678 ;               1 (DVI$_ALLDEVNAM) -- a name with allocation class information
0652  1679 ;                 if allocation class not zero, return $alloclass$ddcn
0652  1680 ;                 otherwise, return node$ddcn
0652  1681 ;               2 (no GETDVI item code) -- an old fashioned name
0652  1682 ;                 return ddcn
0652  1683 ;               3 (no GETDVI item code) -- a secondary path name for displays
0652  1684 ;                 same as -1 except secondary path name returned
0652  1685 ;               4 (no GETDVI item code) -- path controller name for displays
0652  1686 ;                 same as -1 except no unit number is appended
0652  1687 ;               Note: if the node name string is null, node$ is not returned.
0652  1688 ;       R5 = address of device UCB.
0652  1689 ;
0652  1690 ; Outputs:
0652  1691 ;
0652  1692 ;       The device name and unit number are converted and stored in the specified
0652  1693 ;       output buffer. The following register values are returned:
0652  1694 ;
0652  1695 ;               R0 = Final conversion status.
0652  1696 ;                       SS$_NORMAL or
0652  1697 ;                       SS$_BUFFEROVF (an alternate success status which
0652  1698 ;                                      indicates that the supplied buffer could not
0652  1699 ;                                      hold the device name string)
0652  1700 ;               R1 = Length of conversion string.  R1 = 0 if the alternate
0652  1701 ;                    path name was requested but none exists.
0652  1702 ;-
0652  1703
0652  1704 ;
0652  1705 ; Working storage (offsets from R7)
0652  1706 ;
0652  1707          $OFFSET 0,POSITIVE,< -
0652  1708                  <BINNUM,8>, -           ;Binary value to convert to ASCII
0652  1709                  -       ; add new working storage cells before this line
```

```
                                                N 5
                     0652  1710                      <RESR0>, -              ;Result R0
                     0652  1711                      <RESR1>, -              ;Result R1
                     0652  1712                      <SCRLEN,0> -            ;amount of working storage
                     0652  1713                      <RESR2>, -              ;saved R2
                     0652  1714                      <RESR3>, -              ;saved R3
                     0652  1715                      <RESR4>, -              ;saved R4
                     0652  1716                      >
                     0000  1717  BINNUM:
                     0008  1718  RESR0:
                     000C  1719  RESR1:
                     0010  1720  SCRLEN:
                     0010  1721  RESR2:
                     0014  1722  RESR3:
                     0018  1723  RESR4:
                     0652  1724
                     0652  1725  IOC$CVT_DEVNAM::                          ;Convert device name and unit
                     0652  1726
 00FC 8F    BB       0652  1727          PUSHR   #^M<R2,R3,R4,R5,R6,R7>    ;Save registers
                     0656  1728  ;
                     0656  1729  ; Push a quadword onto the stack.  The quadword will land
                     0656  1730  ; on the stack so that when the POPR at the end of the routine
                     0656  1731  ; is executed, R0 will contain the routine value, and R1 will
                     0656  1732  ; contain the length of the formatted device name.
                     0656  1733  ;
 7E    01    7D       0656  1734          MOVQ    #SS$_NORMAL,-(SP)        ;Put a 1 and a 0 on the stack
       7E    7C       0659  1735          CLRQ    -(SP)                    ;Init binary number working area.
                     065B  1736          ASSUME  SCRLEN EQ 16
 57    5E    D0       065B  1737          MOVL    SP, R7                   ;Setup result R0 and R1 pointer in R7.
                     065E  1738  ;
                     065E  1739  ; Precede the device name with a "_" (underscore character) to
                     065E  1740  ; indicate that this is a physical device name.
                     065E  1741  ;
 53    5F  8F    9A       065E  1742          MOVZBL  #^A/_/,R3                ;Put underscore character in R3
         00B4    30       0662  1743          BSBW    PUTCHAR                  ;Put it in the output buffer
                     0665  1744  ;
                     0665  1745  ; Check for a possible nodename. If it exists, determine which format
                     0665  1746  ; of name was requested by the caller.
                     0665  1747  ;
 56    28  A5    D0       0665  1748          MOVL    UCB$L_DDB(R5),R6         ;Get DDB address
 52    34  A6    D0       0669  1749          MOVL    DDB$L_SB(R6),R2          ;Get System Block address
             5D    13       066D  1750          BEQL    LOCAL_NAME               ;None, leave
             09    E1       066F  1751          BBC     #DEV$V_NNM,-             ;Branch if nodename not wanted
       58 3C  A5              0671  1752          UCB$L_DEVCHAR2(R5),LOCAL_NAME
                     0674  1753          CASE    R4, -                    ;Dispatch on type of output requested:
                     0674  1754                  limit=#-1, displist=< -
                     0674  1755                  DISPLAY_NAME, -          ; -1 ==> node$dev: for disks, else dev:
                     0674  1756                  FULL_NAME, -             ;  0 ==> $allocls$dev: or node$dev:
                     0674  1757                  ALLOC_NAME, -            ;  1 ==> $allocls$dev: or node$dev:
                     0674  1758                  LOCAL_NAME, -            ;  2 ==> just dev:
                     0674  1759                  SECONDARY_NAME, -        ;  3 ==> secondary path
                     0674  1760                  DISPLAY_NAME -           ;  4 ==> same as -1 sans unit number
                     0674  1761                  >
             5B    11       0686  1762          BRB     EXDVNM                   ; All others are NOPs.
                     0688  1763
                     0688  1764  FULL_NAME:
 33 38  A5    0E    E1       0688  1765          BBC     #DEV$V_FOD, -            ;A file oriented device?
                     068D  1766                  UCB$L_DEVCHAR(R5), -
```

```
                                                B 6
                     068D  1760                    ADD_NODE                ;Branch if not file oriented device.
                     068D  1761
                     068D  1762 ALLOC_NAME:
                     068D  1763
           67 3C A6  9A 068D  1764          MOVZBL  DDB$L_ALLOCLS(R6), -    ;Setup allocation class value
                     0691  1765                  BINNUM(R7)              ; for conversion.
                  2D 13    0691  1766          BEQL    ADD_NODE            ;If none return node+device name.
                0080 30    0693  1767          BSBW    PUTDOLLAR           ;Prepend allocation class with a '$'
                  58 10    0696  1768          BSBB    PUTNUM              ;Convert allocation class number to
                     0698  1769                                              ;ASCII and put it in the buffer
                  30 11    0698  1770          BRB     ADD_DOLLAR          ;Append dollar sign to alloc. class
                     069A  1771                                              ; and add device name to buffer.
                     069A  1772
                     069A  1773 SECONDARY_NAME:
                  04 E1    069A  1774          BBC     #DEV$V_2P,-         ;Branch if device not dual-pathed.
                3C A5      069C  1775                  UCB$L_DEVCHAR2(R5),-    ; (I.E. there is no secondary path to
                4C         069E  1776                  NO_SECONDARY            ; return.)
           56 00A0 C5 D0   069F  1777          MOVL    UCB$L_DP_DDB(R5),R6 ;Get secondary DDB.
                  45 13    06A4  1778          BEQL    NO_SECONDARY        ;Branch to no sec. path if none.
              52 34 A6 D0   06A6  1779          MOVL    DDB$L_SB(R6),R2     ;Get alternate SB.
                     06AA  1780
                     06AA  1781 DISPLAY_NAME:
     00000000'8F  52 D1    06AA  1782          CMPL    R2,#SCS$GA_LOCALSB  ;Is it the perm local system block?
                  0D 12    06B1  1783          BNEQ    ADD_NODE            ;Return node+devnam for non-local devs.
                     06B3  1784          IFNOCLSTR LOCAL_NAME              ;Return devnam if not part of a cluster.
           0C 38 A5 0E E1   06BB  1785          BBC     #DEV$V_FOD, -       ;A file oriented device?
                     06C0  1786                  UCB$L_DEVCHAR(R5), -
                     06C0  1787                  LOCAL_NAME              ;Branch if not a file oriented device.
                     06C0  1788                                          ;Its a local disk in a cluster: return
                     06C0  1789                                          ;node+device name format.
                     06C0  1790 ;
                     06C0  1791 ; Return node name plus device name.  Copy node name to buffer and
                     06C0  1792 ; suffix with a "$" before moving in rest of device name.
                     06C0  1793 ;
                     06C0  1794 ADD_NODE:
              52 44 A2  9E 06C0  1795          MOVAB   SB$T_NODENAME(R2),R2    ;Point to name field
                  62 95    06C4  1796          TSTB    (R2)                ;Is the node name null?
                  04 13    06C6  1797          BEQL    LOCAL_NAME          ;Skip inserting node name, if its null.
                  3E 10    06C8  1798          BSBB    PUTASCIC            ;Copy counted ASCII str. to output buf.
                     06CA  1799 ADD_DOLLAR:
                  4A 10    06CA  1800          BSBB    PUTDOLLAR           ;Append dollar sign to node name
                     06CC  1801 ;
                     06CC  1802 ; Copy device name to buffer.
                     06CC  1803 ;
                     06CC  1804 LOCAL_NAME:
              52 14 A6  9E 06CC  1805          MOVAB   DDB$T_NAME(R6),R2   ;Get address of ASCIC device name.
                  36 10    06D0  1806          BSBB    PUTASCIC            ;Copy counted ASCII str. to output buf.
           04 18 A7 B1     06D2  1807          CMPW    RESR4(R7),#4        ;Do we want the unit number?
                  0B 13    06D6  1808          BEQL    EXDVNM              ;Nope
           67 54 A5 3C     06D8  1809          MOVZWL  UCB$W_UNIT(R5), -   ;Setup device unit number for
                     06DC  1810                  BINNUM(R7)              ; converstion to ASCII.
                  12 10    06DC  1811          BSBB    PUTNUM              ;Convert unit number to ASCII.
                     06DE  1812 ;
                     06DE  1813 ; Terminate the device name with a ":" (colon).
                     06DE  1814 ;
                  53 3A 9A 06DE  1815          MOVZBL  #^A/:/,R3           ;Put a ":" in R3
                  36 10    06E1  1816          BSBB    PUTCHAR             ;Put the ":" in output buffer
```

```
                     06E3  1817 ;
                     06E3  1818 ; Clean up the stack and exit.  The stack has been set up so that
                     06E3  1819 ; the proper values will be stored in R0 and R1 by the POPR.
                     06E3  1820 ;
5E   08   C0         06E3  1821 EXDVNM: ADDL    #RESR0,SP                ;Remove everything upto result R0
                     06E6  1822                                          ;from the stack
00FF 8F   BA         06E6  1823         POPR    #^M<R0,R1,R2,R3,R4,R5,R6,R7> ;Restore registers
          05         06EA  1824         RSB                              ;Return
                     06EB  1825
                     06EB  1826 ;
                     06EB  1827 ; Come here when the secondary device name was requested but none exists.
                     06EB  1828 ;
                     06EB  1829 NO_SECONDARY:
   0C A7   D4        06EB  1830         CLRL    RESR1(R7)                ;Clear count of characters
      F3   11        06EE  1831         BRB     EXDVNM                   ;and return.
                     06F0  1832
                     06F0  1833
                     06F0  1834 ;++
                     06F0  1835 ; The following code is a local subroutine to convert binary to ASCII and
                     06F0  1836 ; put the ASCII equivalent in the output name buffer.
                     06F0  1837 ;
                     06F0  1838 ; Inputs:
                     06F0  1839 ;
                     06F0  1840 ;       BINNUM(R7)      binary number to be converted (a quadword with high
                     06F0  1841 ;                       longword zeroed
                     06F0  1842 ;
                     06F0  1843 ; Outputs:
                     06F0  1844 ;       The number at BINNUM(R7) is converted to ASCII and stored in the
                     06F0  1845 ;       device name buffer.
                     06F0  1846 ;--
                     06F0  1847 PUTNUM:
         53   01   8E 06F0  1848         MNEGB   #1, R3                   ;Get end-of-number marker.
         7E   53   90 06F3  1849 10$:    MOVB    R3, -(SP)                ;Move digit/marker to scratch.
53   67  67   0A   7B 06F6  1850         EDIV    #10, BINNUM(R7), -       ;Divide number by 10, overwrite number
                     06FB  1851                 BINNUM(R7), R3           ;with quotient, put remainder in R3.
              F6   12 06FB  1852         BNEQ    10$                      ;If quotient not zero, go save this
                     06FD  1853                                          ; digit and get the next one.
                     06FD  1854 ;
                     06FD  1855 ; Get digits -- most significant first (then saved ones), convert them to
                     06FD  1856 ; ASCII, and put them in the output buffer
                     06FD  1857 ;
         53   30   80 06FD  1858 50$:    ADDB    #^A/0/, R3               ;Convert binary digit to ASCII
              17   10 0700  1859         BSBB    PUTCHAR                  ;Copy digit to output buffer
         53   8E   90 0702  1860         MOVB    (SP)+, R3                ;Get another digit
              F6   18 0705  1861         BGEQ    50$                      ;Branch if the end
                   05 0707  1862         RSB
                     0708  1863
                     0708  1864 ;++
                     0708  1865 ; The following code is a local subroutine to copy a counted ASCII string
                     0708  1866 ; to the output name buffer.
                     0708  1867 ;
                     0708  1868 ; Inputs:
                     0708  1869 ;
                     0708  1870 ;       R2      Beginning address of a counted ASCII string
                     0708  1871 ;
                     0708  1872 ; Outputs:
                     0708  1873 ;       The counted ASCII string pointed to by R2 is copied to the device
```

```
                                                D 6
                     0708  1874 ;       name buffer.
                     0708  1875 ;--
                     0708  1876 PUTASCIC:
54   82   9A         0708  1877         MOVZBL  (R2)+, R4               ;Get counted string length.
     08   13         070B  1878         BEQL    90$                     ;If no characters, leave.
53   82   90         070D  1879 5$:      MOVB    (R2)+, R3               ;Move one byte to output buffer.
     07   10         0710  1880         BSBB    PUTCHAR                 ;Put the character in the output buffer.
  F8 54   F5         0712  1881         SOBGTR  R4, 5$                  ;Branch if more to copy.
          05         0715  1882 90$:     RSB                             ;All done, return.
                     0716  1883
                     0716  1884 ;++
                     0716  1885 ;
                     0716  1886 ; The following code is a local subroutine to place a given
                     0716  1887 ; byte in the output buffer.  A count is kept of all characters
                     0716  1888 ; placed in the output buffer.  If the output buffer is full,
                     0716  1889 ; the byte is not copied, the count is not increased, and the
                     0716  1890 ; return status for IOC$CVT_DEVNAM is changed to SS$_BUFFEROVF
                     0716  1891 ; (an alternate success status).
                     0716  1892 ;
                     0716  1893 ; Inputs:
                     0716  1894 ;       R0      Count of unstored character slots remaining in output buffer
                     0716  1895 ;       R1      Address of next unused character slot in output buffer
                     0716  1896 ;       R3      Character to be placed in the buffer
                     0716  1897 ;
                     0716  1898 ; Implicit inputs:
                     0716  1899 ;       RESR0(R7)       longword holding final IOC$CVT_DEVNAM status
                     0716  1900 ;       RESR1(R7)       longword holding final IOC$CVT_DEVNAM count of
                     0716  1901 ;                               characters stored in the buffer (to be
                     0716  1902 ;                               returned in R1
                     0716  1903 ;
                     0716  1904 ; Outputs:
                     0716  1905 ;       None.
                     0716  1906 ;
                     0716  1907 ; Implicit outputs:
                     0716  1908 ;       If R0 >= zero:
                     0716  1909 ;               R0              <== R0 - 1
                     0716  1910 ;               (R1)            <== R3
                     0716  1911 ;               R1              <== R1 + 1
                     0716  1912 ;               RESR1(R7)       <== RESR1(R7) + 1
                     0716  1913 ;       otherwise:
                     0716  1914 ;               RESR0(R7)       <== SS$_BUFFEROVF
                     0716  1915 ;++
                     0716  1916 ; PUTDOLLAR is an internal routine which is the equivalent of:
                     0716  1917 ;
                     0716  1918 ;       MOVB    #^A/$/, R3
                     0716  1919 ;       BSBB    PUTCHAR
                     0716  1920 ;--
                     0716  1921 PUTDOLLAR:
          53   24    90   0716  1922         MOVB    #^A/$/, R3              ;Setup to put "$" in output buffer.
                     0719  1923 PUTCHAR:
          50   D7    0719  1924         DECL    R0                      ;Decrease characters remaining count.
          07   19    071B  1925         BLSS    90$                     ;Branch if no more characters remaining.
81        53   90    071D  1926         MOVB    R3, (R1)+               ;Copy character to output buffer
  0C      A7   D6    0720  1927         INCL    RESR1(R7)               ;Count characters stored
               05    0723  1928         RSB                             ;Return
                     0724  1929
08 A7    0601  8F    3C    0724  1930 90$:     MOVZWL  #SS$_BUFFEROVF, -       ;Set buffer overflow status
```

```
                072A  1931                    RESRO(R7)
 05             072A  1932          RSB
```

```
                                F 6
072B  1934          .SBTTL  BROADCAST TO A TERMINAL
072B  1935 ;++
072B  1936 ; IOC$BROADCAST
072B  1937 ;
072B  1938 ;       This routine will allow driver fork processes to broadcast a
072B  1939 ;       given message to given terminal.  The broadcast request is
072B  1940 ;       dispatched to the proper terminal and control returns immediately
072B  1941 ;       to the caller.  Some time later the broadcast will complete, and
072B  1942 ;       at that time all the necessary post-processing will be done.
072B  1943 ;
072B  1944 ;       This routine does not implement all the features of the $BRDCST system
072B  1945 ;       service, but only the bare minimum necessary to send a message to a
072B  1946 ;       single terminal.  For more information about the terminal broadcast
072B  1947 ;       mechanism, see the module SYSBRDCST.
072B  1948 ;
072B  1949 ; Input:
072B  1950 ;
072B  1951 ;       R1 = Message length
072B  1952 ;       R2 = Message address
072B  1953 ;       R5 = Address of target terminal's UCB
072B  1954 ;
072B  1955 ; Implicit input:
072B  1956 ;
072B  1957 ;       IPL$_ASTDEL  <=  CURRENT_IPL  <=  UCB$B_FIPL(R5)
072B  1958 ;
072B  1959 ; Output:
072B  1960 ;
072B  1961 ;       None.  The contents of R1 .. R5 are preserved across the call.
072B  1962 ;
072B  1963 ; Routine value:
072B  1964 ;
072B  1965 ;       SS$_NORMAL      - The broadcast completed successfully.
072B  1966 ;       SS$_INSFMEM     - Insufficient dynamic nonpaged pool for the request.
072B  1967 ;       SS$_DEVOFFLINE  - The target terminal has rejected the request.
072B  1968 ;       SS$_ILLIOFUNC   - The specified UCB does not belong to a terminal.
072B  1969 ;                         (Therefore a BROADCAST is an illegal I/O function.)
072B  1970 ;--
072B  1971
00000000  072B  1972 SAVED_R0 = 0                                ;.
00000004  072B  1973 SAVED_R1 = 4                                ; .
00000008  072B  1974 SAVED_R2 = 8                                ; .
0000000C  072B  1975 SAVED_R3 = 12                               ; . Symbolic offsets to saved registers
00000010  072B  1976 SAVED_R4 = 16                               ; .
00000014  072B  1977 SAVED_R5 = 20                               ;.
072B  1978
072B  1979 IOC$BROADCAST::                                       ; Broadcast to a terminal
50   00F4 8F   3C  072B  1980          MOVZWL  #SS$_ILLIOFUNC,R0          ; Assume device not a terminal
          02   E1  0730  1981          BBC     #DEV$V_TRM,-               ; Branch if not a terminal
     56 38 A5       0732  1982                  UCB$L_DEVCHAR(R5),14$      ;
       3F   BB      0735  1983          PUSHR   #^M<R0,R1,R2,R3,R4,R5>     ; Save R0 .. R5
51     30   C0      0737  1984          ADDL2   #TTY$K_WB_LENGTH,R1        ; Calculate the total pool required
6E   0124 8F   3C  073A  1985          MOVZWL  #SS$_INSFMEM,SAVED_R0(SP); Assume allocation failure
        F8BE'  30  073F  1986          BSBW    EXE$ALONONPAGED            ; Allocate the pool
       44 50   E9  0742  1987          BLBC    R0,13$                     ; Exit if error
                   0745  1988          ;
                   0745  1989          ; Fill in the Terminal Write Packet (TWP).
                   0745  1990          ; Note that EXE$ALONONPAGED the pool size
```

```
                                                G 6
                     0745  1991          ; in R1 and the pool address in R2.
                     0745  1992          ;
08 A2    51    B0    0745  1993          MOVW    R1,TTY$W_WB_SIZE(R2)    ; Set TWP size
         30    90    0749  1994          MOVB    #DYN$C_TWP,-            ; Set TWP structure type
                     074B  1995                  TTY$B_WB_TYPE(R2)       ;
         06    90    074D  1996          MOVB    #IPL$_QUEUEAST,-        ; Set the TWP fork IPL (for later use)
      0B A2          074F  1997                  TTY$B_WB_FIPL(R2)       ;
10 A2    01    D0    0751  1998          MOVL    #1,TTY$L_WB_FR3(R2)     ; Request refresh of read prompt
      30 A2    9E    0755  1999          MOVAB   TTY$L_WB_DATA(R2),-     ; Set address of message start
      1C A2          0758  2000                  TTY$L_WB_NEXT(R2)       ;
      04 AE    C1    075A  2001          ADDL3   SAVED_R1(SP),-          ; Set address of message end
      1C A2          075D  2002                  TTY$L_WB_NEXT(R2),-     ;
      20 A2          075F  2003                  TTY$L_WB_END(R2)        ;
      96'AF    9E    0761  2004          MOVAB   B^END_BROADCAST,-       ; Set callback address
      2C A2          0764  2005                  TTY$L_WB_RETADDR(R2)    ;
      24 A2    D4    0766  2006          CLRL    TTY$L_WB_IRP(R2)        ; Clear pointer to associated IRP
         52    DD    0769  2007          PUSHL   R2                      ; Save TWP address
      08 AE    28    076B  2008          MOVC3   4+SAVED_R1(SP),-        ; Copy the message text to the TWP
      0C BE          076E  2009                  @4+SAVED_R2(SP),-       ; (note the stack depth changed)
      30 A2          0770  2010                  TTY$L_WB_DATA(R2)       ;
                     0772  2011          ;
                     0772  2012          ; Queue the broadcast request to the terminal.
                     0772  2013          ; The disposition of the broadcast request will be determined
                     0772  2014          ; by the contents of TTY$L_WB_END.  Note that if the request is
                     0772  2015          ; accepted by a remote terminal, or is rejected outright, the
                     0772  2016          ; TWP is no longer needed, and may be deallocated.  The TTY$L_WB_END
                     0772  2017          ; field of the TWP will contain one of the following values:
                     0772  2018          ;
                     0772  2019          ;       System address: request accepted by TTDRIVER
                     0772  2020          ;       1:              request accepted by RTTDRIVER
                     0772  2021          ;       2:              request rejected
                     0772  2022          ;
   53    6E    D0    0772  2023                  MOVL    (SP),R3                 ; Put TWP address in R3
55    18 AE    D0    0775  2024                  MOVL    4+SAVED_R5(SP),R5       ; Restore UCB address
      F884'    30    0779  2025                  BSBW    EXE$ALTQUEPKT           ; Queue the request to the terminal
         50 8ED0    077C  2026                  POPL    R0                      ; Remove TWP address from the stack
   6E    01    3C    077F  2027                  MOVZWL  #SS$_NORMAL,SAVED_R0(SP); Assume success
      20 A0    D5    0782  2028                  TSTL    TTY$L_WB_END(R0)        ; Check for rejection
         05    13    0785  2029                  BEQL    69$                     ; Branch if request rejected
         08    14    0787  2030                  BGTR    80$                     ; Branch if remote terminal accepted
         3F    BA    0789  2031  13$:            POPR    #^M<R0,R1,R2,R3,R4,R5>  ; Restore the registers
               05    078B  2032  14$:            RSB                             ; Return
   0084 8F    3C    078C  2033  69$:            MOVZWL  #SS$_DEVOFFLINE,-       ; Set broadcast rejection status
         6E          0790  2034                  SAVED_R0(SP)            ;
      F86C'    30    0791  2035  80$:            BSBW    COM$DRVDEALMEM          ; Deallocate the TWP
         F3    11    0794  2036                  BRB     13$                     ; Take common exit path
                     0796  2037
                     0796  2038  ;
                     0796  2039  ; The following code performs all of the necessary broadcast post-processing.
                     0796  2040  ; This entry point is FORKed to at IPL IPL$_QUEUEAST from the terminal driver.
                     0796  2041  ; The fork block is the TWP.
                     0796  2042  ;
                     0796  2043  END_BROADCAST:                               ; Post-processor for broadcast requests
   50    55    D0    0796  2044          MOVL    R5,R0                       ; Copy TWP address
      F864'    31    0799  2045          BRW     EXE$DEANONPAGED             ; Deallocate the TWP and return
```

```
                     079C  2047          .SBTTL  BROADCAST EMERGENCY MESSAGE TO CONSOLE
                     079C  2048 ;++
                     079C  2049 ; IOC$CONBRDCST
                     079C  2050 ;
                     079C  2051 ;       This routine will allow emergency messages to be put on the console
                     079C  2052 ;       terminal.  Some time later the broadcast will complete, and
                     079C  2053 ;       at that time all the necessary post-processing will be done.
                     079C  2054 ;
                     079C  2055 ; Input:
                     079C  2056 ;
                     079C  2057 ;       R1 = Message length
                     079C  2058 ;       R2 = Message address
                     079C  2059 ;
                     079C  2060 ; Implicit input:
                     079C  2061 ;
                     079C  2062 ;       IPL$_ASTDEL  <=  CURRENT_IPL  <=  UCB$B_FIPL(R5)
                     079C  2063 ;
                     079C  2064 ;       A dedicated TWP block must immediately preced the message.
                     079C  2065 ;       The low bit of the first byte of the TWP is assumed to remain clear
                     079C  2066 ;       while it is in use.
                     079C  2067 ;
                     079C  2068 ; Output:
                     079C  2069 ;
                     079C  2070 ;       None.  The contents of R1 .. R5 are preserved across the call.
                     079C  2071 ;
                     079C  2072 ; Routine value:
                     079C  2073 ;
                     079C  2074 ;       SS$_NORMAL      - The broadcast completed successfully.
                     079C  2075 ;--
                     079C  2076
                00000000  079C  2077 SAVED_R0 = 0                                    ;.
                00000004  079C  2078 SAVED_R1 = 4                                    ; .
                00000008  079C  2079 SAVED_R2 = 8                                    ; .
                0000000C  079C  2080 SAVED_R3 = 12                                   ; . Symbolic offsets to saved registers
                00000010  079C  2081 SAVED_R4 = 16                                   ; .
                00000014  079C  2082 SAVED_R5 = 20                                   ;.
                          079C  2083
                          079C  2084 IOC$CONBRDCST::                                 ; Broadcast to a terminal
                       3F   BB  079C  2085         PUSHR   #^M<R0,R1,R2,R3,R4,R5>   ; Save R0 .. R5
55   00000000'EF   9E  079E  2086         MOVAB   @PA$UCB0,R5              ; Get the console terminal UCB
     52   30   C2  07A5  2087         SUBL2   #TTY$K_WB_LENGTH,R2      ; Retreat to the start of the TWP
                          07A8  2088         ;
                          07A8  2089         ; Fill in the Terminal Write Packet (TWP).
                          07A8  2090         ;
     08 A2   51   B0  07A8  2091         MOVW    R1,TTY$W_WB_SIZE(R2)     ; Set TWP size
             30   90  07AC  2092         MOVB    #DYN$C_TWP,-             ; Set TWP structure type
          0A A2       07AE  2093                 TTY$B_WB_TYPE(R2)        ;
             06   90  07B0  2094         MOVB    #IPL$_QUEUEAST,-         ; Set the TWP fork IPL (for later use)
          0B A2       07B2  2095                 TTY$B_WB_FIPL(R2)        ;
     10 A2   01   D0  07B4  2096         MOVL    #1,TTY$L_WB_FR3(R2)      ; Request refresh of read prompt
          30 A2   9E  07B8  2097         MOVAB   TTY$L_WB_DATA(R2),-      ; Set address of message start
          1C A2       07BB  2098                 TTY$L_WB_NEXT(R2)        ;
          04 AE   C1  07BD  2099         ADDL3   SAVED_R1(SP),-           ; Set address of message end
          1C A2       07C0  2100                 TTY$L_WB_NEXT(R2),-      ;
          20 A2       07C2  2101                 TTY$L_WB_END(R2)         ;
          EC'AF   9E  07C4  2102         MOVAB   B^END_CONBRDCST,-        ; Set callback address
          2C A2       07C7  2103                 TTY$L_WB_RETADDR(R2)     ;
```

```
                                                I 6
                     24 A2    D4  07C9  2104          CLRL    TTY$L_WB_IRP(R2)          ; Clear pointer to associated IRP
                        52    DD  07CC  2105          PUSHL   R2                        ; Save TWP address
                              07CE  2106          ;
                              07CE  2107          ; Queue the broadcast request to the terminal.
                              07CE  2108          ;
                     53 52    D0  07CE  2109          MOVL    R2,R3                     ; Put TWP address in R3
                        F82C'  30  07D1  2110          BSBW    EXE$ALTQUEPKT             ; Queue the request to the terminal
                        50 8ED0  07D4  2111          POPL    R0                        ; Remove TWP address from the stack
                     6E 01    3C  07D7  2112          MOVZWL  #SS$_NORMAL,SAVED_R0(SP); Assume success
                     20 A0    D5  07DA  2113          TSTL    TTY$L_WB_END(R0)          ; Check for rejection
                        03    13  07DD  2114          BEQL    69$                       ; Branch if request rejected
                        3F    BA  07DF  2115 13$:     POPR    #^M<R0,R1,R2,R3,R4,R5>    ; Restore the registers
                        05    7E1 2116 14$:     RSB                               ; Return
                     0084 8F  3C  07E2  2117 69$:     MOVZWL  #SS$_DEVOFFLINE,-         ; Set broadcast rejection status
                        6E        07E6  2118                  SAVED_R0(SP)              ;
                     60 01    CE  07E7  2119 80$:     MNEGL   #1,(R0)                   ; Mark the TWP free
                        F3    11  07EA  2120          BRB     13$                       ; Take common exit path
                              07EC  2121
                              07EC  2122 ;
                              07EC  2123 ; The following code performs all of the necessary broadcast post-processing.
                              07EC  2124 ; This entry point is FORKed to at IPL IPL$_QUEUEAST from the terminal driver.
                              07EC  2125 ; The fork block is the TWP.
                              07EC  2126 ;
                              07EC  2127 END_CONBRDCST:                             ; Post-processor for broadcast requests
                     65 01    CE  07EC  2128          MNEGL   #1,(R5)                   ; Mark the TWP free
                        05    07EF  2129          RSB
```

```
                     07F0  2131          .SBTTL  SCAN THE I/O DATA BASE
                     07F0  2132 ;+
                     07F0  2133 ; IOC$SCAN_IODB - Scan the I/O data base and return next block.
                     07F0  2134 ;
                     07F0  2135 ; This routine is called to scan the device lists in the IO data base and
                     07F0  2136 ; return a pointer to the next block in the list.  Context is kept in R11
                     07F0  2137 ; and by using back pointers.
                     07F0  2138 ;
                     07F0  2139 ; Inputs:
                     07F0  2140 ;
                     07F0  2141 ;       The I/O data base is locked for read access.  This means that the caller
                     07F0  2142 ;       owns the I/O data base mutex and/or is at IPL SYNCH or higher.
                     07F0  2143 ;
                     07F0  2144 ;               R11 =  0 implies first call
                     07F0  2145 ;               R11 <> 0 indicates that R11 is pointer to current DDB
                     07F0  2146 ;               R10 =  0 implies end of UCB chain
                     07F0  2147 ;               R10 <> 0 indicates that R10 is pointer to current UCB
                     07F0  2148 ;
                     07F0  2149 ; Outputs:
                     07F0  2150 ;
                     07F0  2151 ;               R0  = Success status.
                     07F0  2152 ;               R10 = Pointer to UCB
                     07F0  2153 ;               R11 = Pointer to DDB
                     07F0  2154 ;
                     07F0  2155 ;       All other registers preserved.
                     07F0  2156 ;
                     07F0  2157 ;-
                     07F0  2158
                     07F0  2159 IOC$SCAN_IODB::
                     07F0  2160
          50   01   D0  07F0  2161          MOVL    #1,R0                   ; Success
          5B   D5  07F3  2162          TSTL    R11                     ; Initial condition?
               2C  13  07F5  2163          BEQL    50$                     ; Yes
               5A  D5  07F7  2164          TSTL    R10                     ; End of chain?
               07  13  07F9  2165          BEQL    10$                     ; Yes
     5A   30   AA  D0  07FB  2166          MOVL    UCB$L_LINK(R10),R10     ; Get next UCB
               01  13  07FF  2167          BEQL    10$                     ; None
                   05  0801  2168          RSB
                       0802  2169
               6B  D5  0802  2170 10$:     TSTL    DDB$L_LINK(R11)         ; At end of DDB chain?
               0A  13  0804  2171          BEQL    30$                     ; Yes
          5B   6B  D0  0806  2172          MOVL    DDB$L_LINK(R11),R11     ; No, get next one
     5A   04   AB  D0  0809  2173 20$:     MOVL    DDB$L_UCB(R11),R10      ; Pick up first UCB
               F3  13  080D  2174          BEQL    10$                     ; None, get next DDB
                   05  080F  2175          RSB
                       0810  2176
     5B   34   AB  D0  0810  2177 30$:     MOVL    DDB$L_SB(R11),R11       ; Get back to parent system block
          5B   6B  D0  0814  2178 40$:     MOVL    SB$L_FLINK(R11),R11     ; Get next system block
00000000'8F   5B  D1  0817  2179          CMPL    R11,#SCS$GQ_CONFIG      ; End of chain?
               0A  12  081E  2180          BNEQ    60$                     ; No
               50  D7  0820  2181          DECL    R0
                   05  0822  2182          RSB
                       0823  2183
5B   00000000'9F   D0  0823  2184 50$:     MOVL    @#SCS$GQ_CONFIG,R11     ; Pick up first system block
          54   AB  D5  082A  2185 60$:     TSTL    SB$L_DDB(R11)           ; Is there a DDB chain?
               E5  13  082D  2186          BEQL    40$                     ; No, go try next SB
     5B   54   AB  D0  082F  2187          MOVL    SB$L_DDB(R11),R11       ; Yes, get the first DDB
```

```
D4   11  0833  2188          BRB     20$
         0835  2189
```

```
                     0835  2191          .SBTTL  SCAN THE I/O DATA BASE BOTH PRIMARY & SECONDARY PATHS
                     0835  2192 ;++
                     0835  2193 ; IOC$SCAN_IODB_2P
                     0835  2194 ;
                     0835  2195 ; This routine is called to scan the device lists in the IO data base and
                     0835  2196 ; return a pointer to the next block in the list.  Context is kept in R10
                     0835  2197 ; and R11 and by using back pointers.
                     0835  2198 ;
                     0835  2199 ; SCAN_IODB_2P differs from SCAN_IODB in that it will scan both the primary
                     0835  2200 ; and secondary UCB chain for each DDB.  This means that if a device is
                     0835  2201 ; dual-pathed, SCAN_IODB_2P will return the address of its UCB twice, once in
                     0835  2202 ; the context of the primary controller and once in the context of the
                     0835  2203 ; secondary.
                     0835  2204 ;
                     0835  2205 ; Inputs and Outputs are identical to IOC$SCAN_IODB.
                     0835  2206 ;--
                     0835  2207
                     0835  2208 IOC$SCAN_IODB_2P::
                     0835  2209
          50   01    D0   0835  2210          MOVL    #1,R0                   ; Success
               5B    D5   0838  2211          TSTL    R11                     ; Initial condition?
               41    13   083A  2212          BEQL    60$                     ; Yes
               5A    D5   083C  2213          TSTL    R10                     ; Caller signalled end of chain?
               1C    13   083E  2214          BEQL    30$                     ; Yes, done with this DDB
                     0840  2215 ;
                     0840  2216 ; At this point we must decide if we're following the primary or secondary
                     0840  2217 ; chain of UCBs on this DDB.
                     0840  2218 ;
     5B    28 AA    D1   0840  2219          CMPL    UCB$L_DDB(R10),R11      ; Are we traversing the primary chain?
               07    12   0844  2220          BNEQ    10$                     ; Branch if we're following secondary
     5A    30 AA    D0   0846  2221          MOVL    UCB$L_LINK(R10),R10     ; Get next UCB on primary chain
               09    13   084A  2222          BEQL    20$                     ; Branch if none to try secondary chain
               05    13   084C  2223          RSB                             ; Else return UCB address to caller
                     084D  2224 ;
                     084D  2225 ; Get next UCB on secondary chain.
                     084D  2226 ;
 5A    00A4 CA    D0   084D  2227 10$:     MOVL    UCB$L_DP_LINK(R10),R10  ; Get next UCB on secondary chain
               08    13   0852  2228          BEQL    30$                     ; Branch if none left
               05    13   0854  2229          RSB                             ; Else return UCB address to caller
                     0855  2230 ;
                     0855  2231 ; No UCBs left on primary chain; traverse secondary chain if present.
                     0855  2232 ;
     5A    40 AB    D0   0855  2233 20$:     MOVL    DDB$L_DP_UCB(R11),R10   ; Get first UCB on secondary chain
               01    13   0859  2234          BEQL    30$                     ; Branch if none to try next DDB
               05    13   085B  2235          RSB                             ; Else return UCB address to caller
                     085C  2236 ;
                     085C  2237 ; Step to next DDB.
                     085C  2238 ;
               6B    D5   085C  2239 30$:     TSTL    DDB$L_LINK(R11)         ; At end of DDB chain?
               0A    13   085E  2240          BEQL    40$                     ; Yes, try next system block
          5B   6B    D0   0860  2241          MOVL    DDB$L_LINK(R11),R11     ; No, get next one
     5A    04 AB    D0   0863  2242 35$:     MOVL    DDB$L_UCB(R11),R10      ; Pick up first UCB on primary chain
               EC    13   0867  2243          BEQL    20$                     ; None, try for UCB on secondary chain
               05    13   0869  2244          RSB                             ; Else return UCB address to caller
                     086A  2245
                     086A  2246 ;
                     086A  2247 ; Step to next system block.
```

```
                                086A  2248 ;
          5B   34 AB    D0      086A  2249 40$:    MOVL    DDB$L_SB(R11),R11       ; Get back to parent system block
            5B   6B    D0      086E  2250 50$:    MOVL    SB$L_FLINK(R11),R11     ; Get next system block
00000000'8F     5B    D1      0871  2251          CMPL    R11,#SCS$GQ_CONFIG      ; End of chain?
                0A    12      0878  2252          BNEQ    70$                     ; No
                50    D7      087A  2253          DECL    R0                      ; Signal end of IO scan
                05    05      087C  2254          RSB
                              087D  2255
5B   00000000'9F    D0      087D  2256 60$:    MOVL    @#SCS$GQ_CONFIG,R11     ; Pick up first system block
              54 AB    D5      0882  2257 70$:    TSTL    SB$L_DDB(R11)           ; Is there a DDB chain?
                E5    13      0887  2258          BEQL    50$                     ; No, go try next SB
          5B   54 AB    D0      0889  2259          MOVL    SB$L_DDB(R11),R11       ; Yes, get the first DDB
                D4    11      088D  2260          BRB     35$                     ; Try for UCB on primary chain
```

```
                                                N 6
088F  2262          .SBTTL  IOC$CTRLINIT - Call driver controller init. routine
088F  2263 ;++
088F  2264 ;  FUNCTIONAL DESCRIPTION:
088F  2265 ;
088F  2266 ;       For UNIBUS devices, the device CSR is tested for existance.  If this
088F  2267 ;       test fails, a no routine call occurs and failure status is returned in
088F  2268 ;       R0.  Input values for a device driver's controller initialization
088F  2269 ;       routine are loaded into the proper registers, the routine starting
088F  2270 ;       address is located, and if a routine exists, it is called.
088F  2271 ;
088F  2272 ;  INPUTS:
088F  2273 ;       R1      CSR address to use if IDB contains zero
088F  2274 ;       R8      CRB address (primary)
088F  2275 ;       R11     DDB address
088F  2276 ;
088F  2277 ;  OUTPUTS:
088F  2278 ;       R0      Status (success, or failure ==> UNIBUS CSR non-existant)
088F  2279 ;       R1-R6   Destroyed
088F  2280 ;--
088F  2281
088F  2282 ;++
088F  2283 ;  Controller initialization routine parameters:
088F  2284 ;
088F  2285 ;  INPUTS:
088F  2286 ;       R4      CSR address (for UNIBUS and MASSBUS devices)
088F  2287 ;               SCSSYSTEMID address (for class drivers during SYSGEN driver
088F  2288 ;                   loading)
088F  2289 ;               zero for all others, including class drivers during power
088F  2290 ;                   failure recovery
088F  2291 ;       R5      IDB address (or zero if none exists)
088F  2292 ;       R6      DDB address
088F  2293 ;       R8      CRB address
088F  2294 ;
088F  2295 ;  OUTPUTS:
088F  2296 ;       Must preserve all registers except R0 through R4.
088F  2297 ;
088F  2298 ;--
088F  2299
088F  2300
088F  2301 IOC$CTRLINIT::
088F  2302
55   2C A8    D0  088F  2303          MOVL    CRB$L_INTD+VEC$L_IDB(R8), R5    ; Get IDB address.
          05    18  0893  2304          BGEQ    10$                             ; Branch if none.
     54   65    D0  0895  2305          MOVL    IDB$L_CSR(R5), R4               ; Get CSR address.
          05    19  0898  2306          BLSS    20$                             ; Branch if really a CSR.
     54   51    D0  089A  2307 10$:     MOVL    R1, R4                          ; Else, use supplied value,
          1B    11  089D  2308          BRB     40$                             ; and skip CSR testing.
                    089F  2309
56   14 A5    D0  089F  2310 20$:     MOVL    IDB$L_ADP(R5), R6               ; Get ADP address.
          15    18  08A3  2311          BGEQ    40$                             ; If none, skip CSR test.
0E A6    01    B1  08A5  2312          CMPW    #AT$_UBA, ADP$W_ADPTYPE(R6)     ; Is this a UBA?
          0F    12  08A9  2313          BNEQ    40$                             ; If not a UBA, skip CSR test.
     56   66    D0  08AB  2314          MOVL    ADP$L_CSR(R6), R6               ; Get adapter config reg addr.
     50   54    D0  08AE  2315          MOVL    R4, R0                          ; Setup CSR for test.
00000000'GF    16  08B1  2316          JSB     G^EXE$TEST_CSR                  ; Test UNIBUS CSR.
     0E   50    E9  08B7  2317          BLBC    R0, 90$                         ; Branch if no CSR present.
                    08BA  2318
```

```
                                                B 7
50   30 A8   D0  08BA  2319 40$:    MOVL    CRB$L_INTD+VEC$L_INITIAL(R8), R0 ; Get ctrl init rout addr.
        05   18  08BE  2320          BGEQ    80$                               ; Branch if none.
   56   5B   D0  08C0  2321          MOVL    R11, R6                           ; Get DDB address.
        60   16  08C3  2322          JSB     (R0)                              ; Call ctrl init routine.
                 08C5  2323
   50   01   D0  08C5  2324 80$:    MOVL    #1, R0                            ; Set success status.
             05  08C8  2325 90$:    RSB                                       ; Return w/ status.
```

```
                        08C9  2327          .SBTTL  IOC$UNITINIT - Call driver unit init. routine
                        08C9  2328 ;++
                        08C9  2329 ;  FUNCTIONAL DESCRIPTION:
                        08C9  2330 ;
                        08C9  2331 ;       Input values for a device driver's unit initialization routine are
                        08C9  2332 ;       loaded into the proper registers, the routine starting address is
                        08C9  2333 ;       located, and if a routine exists, it is called.
                        08C9  2334 ;
                        08C9  2335 ;  INPUTS:
                        08C9  2336 ;       R5      UCB address
                        08C9  2337 ;       R8      CRB address (primary)
                        08C9  2338 ;
                        08C9  2339 ;  OUTPUTS:
                        08C9  2340 ;       R0-R4   Destroyed
                        08C9  2341 ;
                        08C9  2342 ;  NOTES:
                        08C9  2343 ;
                        08C9  2344 ;       There are two unit initialization routine addresses in the I/O data
                        08C9  2345 ;       base; CRB$L_INTD_VEC$L_UNITINIT and DDT$L_UNITINIT.  Normally, only
                        08C9  2346 ;       one of these two places should contain a unit initialization routine
                        08C9  2347 ;       address.  However, for the console block storage device, the both
                        08C9  2348 ;       locations contain an address, and the DDT contains the address which
                        08C9  2349 ;       must be used.
                        08C9  2350 ;
                        08C9  2351 ;       In this case, the CRB is shared by the console terminal and console
                        08C9  2352 ;       block storage devices.  The CRB contains the address of the unit
                        08C9  2353 ;       initialization routine for the console terminal, and the console
                        08C9  2354 ;       terminal DDT contains no unit initialization routine address.  Thus
                        08C9  2355 ;       the console terminal device "fits" the "normal" case.  However, the
                        08C9  2356 ;       console block storage device DDT contains a unit initialization
                        08C9  2357 ;       routine which differs from the console terminal unit initialization
                        08C9  2358 ;       routine and whose address is stored in the DDT.
                        08C9  2359 ;
                        08C9  2360 ;       Since the CRB is shared and contains the wrong unit initialization
                        08C9  2361 ;       routine address for the console block storage device, the DDT must be
                        08C9  2362 ;       inspected first.  Initialization for the console terminal will be
                        08C9  2363 ;       accomplished correctly regardless of which address is checked first.
                        08C9  2364 ;
                        08C9  2365 ;--
                        08C9  2366
                        08C9  2367 ;++
                        08C9  2368 ;  Unit initialization routine parameters:
                        08C9  2369 ;
                        08C9  2370 ;  INPUTS:
                        08C9  2371 ;       R3      CSR address (primary)
                        08C9  2372 ;       R4      CSR address (secondary, same as primary if no secondary exists)
                        08C9  2373 ;       R5      UCB address
                        08C9  2374 ;
                        08C9  2375 ;  OUTPUTS:
                        08C9  2376 ;       Must preserve all registers except R0 through R4.
                        08C9  2377 ;
                        08C9  2378 ;--
                        08C9  2379
                        08C9  2380
                        08C9  2381 IOC$UNITINIT::
                        08C9  2382
     50   0088 C5   D0  08C9  2383          MOVL    UCB$L_DDT(R5), R0               ; Get DDT address.
```

```
                                                D 7
                50    18 A0    D0  08CE  2384          MOVL    DDT$L_UNITINIT(R0), R0                  ; Get DDT unit init rout addr.
000005E4'8F       50       D1  08D2  2385          CMPL    R0, #IOC$RETURN                         ; Null unit init routine?
                     06    12  08D9  2386          BNEQ    10$                                     ; Branch if real unit init rout.
                50    3C A8    D0  08DB  2387          MOVL    CRB$L_INTD+VEC$L_UNITINIT(R8), R0 ; Get CRB unit init rout addr.
                     1A    18  08DF  2388          BGEQ    90$                                     ; Branch if no unit init rout.
                                08E1  2389
                     54    D4  08E1  2390 10$:     CLRL    R4                                      ; Assume no IDB exists.
                53    2C A8    D0  08E3  2391          MOVL    CRB$L_INTD+VEC$L_IDB(R8), R3    ; Get IDB address.
                     10    18  08E7  2392          BGEQ    50$                                     ; Branch if none.
                   53    63    D0  08E9  2393          MOVL    IDB$L_CSR(R3), R3                   ; Get primary CSR.
                   54    53    D0  08EC  2394          MOVL    R3, R4                              ; Assume no sec. CRB exists.
                51    20 A8    D0  08EF  2395          MOVL    CRB$L_LINK(R8), R1                  ; Get secondary CRB addr.
                     04    18  08F3  2396          BGEQ    50$                                     ; Branch if none.
                                08F5  2397          ASSUME  IDB$L_CSR EQ 0
                54    2C B1    D0  08F5  2398          MOVL    @CRB$L_INTD+VEC$L_IDB(R1), R4   ; Get secondary CSR addr.
                                08F9  2399
                     60    17  08F9  2400 50$:     JMP     (R0)                                    ; Call unit init routine, and
                                08FB  2401                                                          ; return to caller.
                                08FB  2402
                     05    08FB  2403 90$:     RSB                                                 ; No unit init routine to call:
                                08FC  2404                                                          ; return to caller.
```

```
                                                E 7
                                .SBTTL  Parse Device Name String

                                ;+
                                ;
                                ; IOC$PARSDEVNAM - parse device name string
                                ;
                                ; This routine parses a device name string, checking syntax and
                                ; extracting node name, allocation class number, and unit number.
                                ; If device type format is specified, it is converted into the internal
                                ; compressed format.
                                ;
                                ; INPUTS:
                                ;
                                ;       R8 = size of name string
                                ;       R9 = address of name string
                                ;       R10 = flags
                                ;
                                ; OUTPUTS:
                                ;
                                ;       R0 = SS$_NORMAL - valid name string
                                ;          = SS$_IVDEVNAM - invalid device name string
                                ;       R2 = unit number
                                ;       R3 = length of SCS node name at head of name string
                                ;            or allocation class number
                                ;            or device type code
                                ;       R8 = size of name string
                                ;       R9 = address of name string
                                ;       R10 = flags
                                ;       R4 - R7, R11 preserved
                                ;
                                ;-

                                        .ENABLE LSB

                                IOC$PARSDEVNAM::
        0070 8F     BB  08FC 2441           PUSHR   #^M<R4,R5,R6>           ; save working registers
             58     D5  0900 2442           TSTL    R8                      ; check name string length
             28     13  0902 2443           BEQL    30$                     ; branch if null - error
        54   58     7D  0904 2444           MOVQ    R8,R4                   ; copy name string descriptor
56      59   01     C3  0907 2445           SUBL3   #1,R9,R6                ; default is no node no allocation
                        090B 2446                                           ; class, set pointer before beginning
                        090B 2447                                           ; of the string
69      58   24     3A  090B 2448           LOCC    #^A'$',R8,(R9)          ; scan name for a "$"
             03     13  090F 2449           BEQL    10$                     ; failed to find one - no nodename
        56   51     D0  0911 2450           MOVL    R1,R6                   ; found it, save pointer
             52     7C  0914 2451 10$:      CLRQ    R2                      ; init unit number and node name
        50   65     9A  0916 2452 20$:      MOVZBL  (R5),R0                 ; get next character
   11   50   06     E1  0919 2453           BBC     #6,R0,40$               ; br if code 0-^X3F - numeric or $
        50   20     8A  091D 2454           BICB    #^X20,R0                ; collapse lower case to upper case
   5A 8F     50     91  0920 2455           CMPB    R0,#^A'Z'               ; possible alphabetic?
             77     1A  0924 2456           BGTRU   150$                    ; br if not
   41 8F     50     91  0926 2457           CMPB    R0,#^A'A'               ; possible alphabetic?
             37     1E  092A 2458           BGEQU   70$                     ; branch if OK - store it
             6F     11  092C 2459 30$:      BRB     150$                    ; no - error
                        092E 2460 ;
                        092E 2461 ; Non alphabetic - may be numeric or "$"
                        092E 2462 ;
```

```
                56   55   D1   092E  2463 40$:    CMPL    R5,R6                   ; hit the "$" yet?
                     0E   13   0931  2464         BEQL    50$                     ; yes, deal with it
                     34   1A   0933  2465         BGTRU   80$                     ; past it, digits are unit number
                39   50   91   0935  2466         CMPB    R0,#^A'9'               ; legal?
                     63   1A   0938  2467         BGTRU   150$                    ; no, error
                30   50   91   093A  2468         CMPB    R0,#^A'0'               ; legal?
                     24   1E   093D  2469         BGEQU   70$                     ; yes, accept it as alpha
                     5C   11   093F  2470         BRB     150$                    ; no, error
                               0941  2471 ;
                               0941  2472 ; $ in device name - either node name or allocation class.
                               0941  2473 ;
            53   55   59   C3   0941  2474 50$:    SUBL3   R9,R5,R3                ; compute length of node name
                     1C   12   0945  2475         BNEQ    70$                     ; branch if non-null - process the $
                               0947  2476 ;
                               0947  2477 ; Process allocation class number.
                               0947  2478 ;
                     55   D6   0947  2479 60$:    INCL    R5                      ; move over "$" to allocation
                     54   D7   0949  2480         DECL    R4                      ; class digits.
                     6A   10   094B  2481         BSBB    GETNUMBER               ; convert allocation class.
                53   52   D0   094D  2482         MOVL    R2,R3                   ; store requested allocation class.
                     4B   15   0950  2483         BLEQ    150$                    ; leq zero is not legal.
                5A   04   88   0952  2484         BISB    #IOC$M_CLASS,R10        ; set allocation class flag
                50   24   91   0955  2485         CMPB    #^A'$',R0               ; was terminator a "$"?
                     43   12   0958  2486         BNEQ    150$                    ; if not, invalid device name.
                58   54   7D   095A  2487         MOVQ    R4,R8                   ; reset device name - unit size.
                     54   D5   095D  2488         TSTL    R4                      ; check remaining string count
                     B5   14   095F  2489         BGTR    20$                     ; if characters remain, process them.
                     3A   11   0961  2490         BRB     150$                    ; else, invalid device name.
                               0963  2491
                85   50   90   0963  2492 70$:    MOVB    R0,(R5)+                ; store potentially upcased character
                  AD 54   F5   0966  2493         SOBGTR  R4,20$                  ; any more characters to scan?
                               0969  2494 ;
                               0969  2495 ; End of alpha scan. Make sure we actually got a non-null device name.
                               0969  2496 ;
                58   54   C2   0969  2497 80$:    SUBL    R4,R8                   ; subtract unit number from string
                     2F   13   096C  2498         BEQL    150$                    ; if eql no device name specified
                     56   D6   096E  2499         INCL    R6                      ; point past $ in node name
                55   56   D1   0970  2500         CMPL    R6,R5                   ; see if we've processed any more chars
                     09   1F   0973  2501         BLSSU   90$                     ; branch if yes
                  25 5A   E8   0975  2502         BLBS    R10,150$                ; branch if physical - not valid
            21 5A    06   E1   0978  2503         BBC     #IOC$V_ANY,R10,150$     ; or if not generic search for any
                     0D   11   097C  2504         BRB     100$                    ; node name only - verify end of string
                               097E  2505 ;
                               097E  2506 ; Process unit number and make sure there's no trailing junk.
                               097E  2507 ;
                     52   D4   097E  2508 90$:    CLRL    R2                      ; init unit number to 0
                     54   D5   0980  2509         TSTL    R4                      ; see if there's anything left
                     0B   15   0982  2510         BLEQ    110$                    ; branch if not
                5A   01   88   0984  2511         BISB    #IOC$M_PHY,R10          ; set physical flag
                     2E   10   0987  2512         BSBB    GETNUMBER               ; convert unit number
                     54   D6   0989  2513         INCL    R4                      ; return terminator to string count
                     54   D5   098B  2514 100$:   TSTL    R4                      ; reached end of string?
                     0E   14   098D  2515         BGTR    150$                    ; branch if not - error
            37 5A    01   E0   098F  2516 110$:   BBS     #IOC$V_TYPE,R10,190$    ; branch if name is a device type
                50   01   D0   0993  2517 120$:   MOVL    #SS$_NORMAL,R0          ; successful parse
                0070 8F   BA   0996  2518 130$:   POPR    #^M<R4,R5,R6>           ; restore registers
                               05   099A  2519         RSB                     ; and return
```

```
                                099B  2520 ;
                                099B  2521 ; Invalid device name
                                099B  2522 ;
                          8E    D5  099B  2523 140$:   TSTL    (SP)+                   ; pop GETNUMBER return address from stack
                    50    0144  8F  3C  099D  2524 150$:   MOVZWL  #SS$_IVDEVNAM,R0     ; set invalid device name
                          F2    11  09A2  2525           BRB     130$
                                09A4  2526
                                09A4  2527 ;+
                                09A4  2528 ; Routine to convert ASCII to integer
                                09A4  2529 ;
                                09A4  2530 ; Inputs:
                                09A4  2531 ;
                                09A4  2532 ;       R2      assumed zero
                                09A4  2533 ;       R4      size of string
                                09A4  2534 ;       R5      starting address of string
                                09A4  2535 ;
                                09A4  2536 ; Outputs:
                                09A4  2537 ;
                                09A4  2538 ;       R0      terminator character
                                09A4  2539 ;       R2      converted number
                                09A4  2540 ;       R4      size of string with number and terminator character removed
                                09A4  2541 ;       R5      address of first character after number terminator character
                                09A4  2542 ;-
                                09A4  2543
                          50    85  9A  09A4  2544 160$:   MOVZBL  (R5)+,R0             ; get next character.
                          50    30  82  09A7  2545         SUBB    #^A'0',R0           ; base it at decimal digits.
                                10  1F  09AA  2546         BLSSU   170$                ; branch if not a decimal digit.
                          09    50  91  09AC  2547         CMPB    R0,#9               ; is it a digit?
                                0B  1A  09AF  2548         BGTRU   170$                ; branch if not a decimal digit.
                          52    0A  C4  09B1  2549         MULL    #10,R2              ; scale current unit number by 10
                          52    50  C0  09B4  2550         ADDL    R0,R2               ; add new digit to accumulation.
                                09B7  2551 GETNUMBER:
                             EA 54  F4  09B7  2552         SOBGEQ  R4,160$             ; count off a character
                                04  11  09BA  2553         BRB     180$                ; branch if no more characters
                                09BC  2554
                          50    FF  A5  9A  09BC  2555 170$:   MOVZBL  -1(R5),R0       ; restore terminator character.
                    00008000  8F    52  D1  09C0  2556 180$:   CMPL    R2,#32768       ; check number value
                                D2  1E  09C7  2557         BGEQU   140$                ; branch if not valid
                                05  09C9  2558         RSB                         ; return to caller.
                                09CA  2559
                                09CA  2560 ;
                                09CA  2561 ; Parse device type name. We do this last because all the regular device
                                09CA  2562 ; name validation is necessary anyway. Now we just have to do the
                                09CA  2563 ; additional checks and pack the characters.
                                09CA  2564 ;
                                53  D5  09CA  2565 190$:   TSTL    R3                  ; check if we saw node or alloc class
                                CF  12  09CC  2566         BNEQ    150$                ; branch if so - not valid
                          50    55  59  C3  09CE  2567         SUBL3   R9,R5,R0            ; compute total length of string
                                50  58  C2  09D2  2568         SUBL    R8,R0               ; compute length of unit number string
                                02  50  D1  09D5  2569         CMPL    R0,#2               ; must be two digits
                                C3  12  09D8  2570         BNEQ    150$                ; branch if not - not valid
                                55  59  D0  09DA  2571         MOVL    R9,R5               ; copy name address again
                                02  58  D1  09DD  2572         CMPL    R8,#2               ; check minimum name length
                                BB  1F  09E0  2573         BLSSU   150$                ; too short - out
                    50    85    40  8F  83  09E2  2574         SUBB3   #^A'A'-1,(R5)+,R0   ; get char and convert to compressed
              53    05    11    50  F0  09E7  2575         INSV    R0,#17,#5,R3        ; store compressed code
                    50    85    40  8F  83  09EC  2576         SUBB3   #^A'A'-1,(R5)+,R0   ; get char and convert to compressed
```

```
                53   05   0C   50   F0  09F1  2577          INSV    R0,#12,#5,R3            ; store compressed code
                          03   58   D1  09F6  2578          CMPL    R8,#3                   ; check how many chars left
                               A2   1A  09F9  2579          BGTRU   150$                    ; string was longer than 5 - error
                               0A   1F  09FB  2580          BLSSU   200$                    ; short - don't take 3rd alpha
                     50   85   40   8F  83  09FD  2581          SUBB3   #^A'A'-1,(R5)+,R0       ; get char and convert to compressed
                53   05        07   50  F0  0A02  2582          INSV    R0,#7,#5,R3             ; store compressed code
                               53   52  C0  0A07  2583 200$:     ADDL    R2,R3                   ; add in unit number
                               5A   01  8A  0A0A  2584          BICB    #IOC$M_PHY,R10          ; clear physical flag
                                  FF83  31  0A0D  2585          BRW     120$                    ; done
```

```
                                        I 7
0A10  2587          .SBTTL  Search I/O Database for Device
0A10  2588
0A10  2589 ;+
0A10  2590 ;
0A10  2591 ; IOC$SEARCHINT - internal I/O database search
0A10  2592 ;
0A10  2593 ; This routine searches the I/O database for the specified device, using
0A10  2594 ; the specified search rules. Depending on the search, a lock may or may
0A10  2595 ; not be taken out on the device when it is found.
0A10  2596 ;
0A10  2597 ; Note! While this routine is non-paged and therefore may be called at
0A10  2598 ; elevated IPL, the device locking code it calls is not. Therefore, only
0A10  2599 ; searches with IOC$V_ANY may be called from elevated IPL.
0A10  2600 ;
0A10  2601 ; INPUTS:
0A10  2602 ;
0A10  2603 ;       R2 = unit number
0A10  2604 ;       R3 = length of SCS node name at head of name string
0A10  2605 ;            or allocation class number
0A10  2606 ;            or device type code
0A10  2607 ;       R8 = size of name string
0A10  2608 ;       R9 = address of name string
0A10  2609 ;       R10 = flags
0A10  2610 ;       R11 = address to store lock value block
0A10  2611 ;       I/O database mutex held, IPL 2
0A10  2612 ;
0A10  2613 ; OUTPUTS:
0A10  2614 ;
0A10  2615 ;       R0 = SS$_NORMAL - device found
0A10  2616 ;          = SS$_NOSUCHDEV - device not found
0A10  2617 ;          = SS$_NODEVAVL - device exists but not available according to rules
0A10  2618 ;          = SS$_DEVALLOC - device allocated to other user
0A10  2619 ;          = SS$_NOPRIV - failed device protection
0A10  2620 ;          = SS$_TEMPLATEDEV - can't allocate template device
0A10  2621 ;          = SS$_DEVMOUNT - device already mounted
0A10  2622 ;          = SS$_DEVOFFLINE - device marked offline
0A10  2623 ;       R5 = UCB
0A10  2624 ;       R6 = DDB
0A10  2625 ;       R7 = system block
0A10  2626 ;       R10 - R4, R8 - R11 preserved
0A10  2627 ;
0A10  2628 ;       Note: If failure, R5 - R7 point to the last structures looked at.
0A10  2629 ;
0A10  2630 ;-
0A10  2631
0A10  2632 ;
0A10  2633 ; Stack use:
0A10  2634 ;
00000000  0A10  2635 $AVR2   = 0
00000004  0A10  2636 $AVR3   = 4
00000008  0A10  2637 $AVR4   = 8
0000000C  0A10  2638 $AVR8   = 12
00000010  0A10  2639 $AVR9   = 16
0A10  2640
0A10  2641
0A10  2642          .ENABLE LSB
0A10  2643
```

```
                                                J 7
                     0A10  2644 IOC$SEARCHINT::
          031C 8F    BB  0A10  2645         PUSHR   #^M<R2,R3,R4,R8,R9>     ; save registers
                     0A14  2646 ;
                     0A14  2647 ; Search the system blocks for a suitable node. If we are doing a search
                     0A14  2648 ; by allocation class, generic device type, or no node name is given,
                     0A14  2649 ; all system blocks qualify.
                     0A14  2650 ;
     57  00000000'EF DE  0A14  2651         MOVAL   SCS$GQ_CONFIG,R7        ; get head of SCS SB list
         50    67    D0  0A1B  2652 10$:    MOVL    SB$L_FLINK(R7),R0       ; get next system block
00000000'8F    50    D1  0A1E  2653         CMPL    R0,#SCS$GQ_CONFIG       ; are we back at list head?
               78    13  0A25  2654         BEQL    50$                     ; branch if yes - all done
                     0A27  2655
         57    50    D0  0A27  2656         MOVL    R0,R7
      56   54 A7    DE  0A2A  2657         MOVAL   SB$L_DDB-DDB$L_LINK(R7),R6 ; pick up DDB listhead
         55    56    D0  0A2E  2658         MOVL    R6,R5                   ; make sure UCB is non-zero
                     0A31  2659                                            ; if allocation class or generic dev,
         5A    06    93  0A31  2660         BITB    #IOC$M_CLASS!IOC$M_TYPE,R10
               27    12  0A34  2661         BNEQ    30$                     ; check every system block
      58    0C AE    7D  0A36  2662         MOVQ    SAVR8(SP),R8            ; get orig dev name descriptor
      53    04 AE    D0  0A3A  2663         MOVL    SAVR3(SP),R3            ; get node name length
               1D    13  0A3E  2664         BEQL    30$                     ; branch if none - go ahead
      44 A7    53    91  0A40  2665         CMPB    R3,SB$T_NODENAME(R7)    ; check node name length
               D5    12  0A44  2666         BNEQ    10$                     ; branch if not
   69 45 A7    53    29  0A46  2667         CMPC3   R3,SB$T_NODENAME+1(R7),(R9) ; node names match?
               CE    12  0A4B  2668         BNEQ    10$                     ; branch if not
                     0A4D  2669 ;
                     0A4D  2670 ; Found a suitable system block. Search its DDB list.
                     0A4D  2671 ;
         50    01    3C  0A4D  2672 20$:    MOVZWL  #SS$_NORMAL,R0
   53    04 AE    01    C1  0A50  2673         ADDL3   #1,SAVR3(SP),R3         ; include the "$"
         59    53    C0  0A55  2674         ADDL    R3,R9                   ; skip over the nodename
         58    53    C2  0A58  2675         SUBL    R3,R8                   ; adjust the length
               52    15  0A5B  2676         BLEQ    60$                     ; if no device name, just return SB
                     0A5D  2677
         50    66    D0  0A5D  2678 30$:    MOVL    DDB$L_LINK(R6),R0       ; get address of next DDB
               5A    13  0A60  2679         BEQL    80$                     ; if eql end of list
         56    50    D0  0A62  2680         MOVL    R0,R6
      55    D4 A6    DE  0A65  2681         MOVAL   <DDB$L_UCB-UCB$L_LINK>(R6),R5 ; initialize primary UCB address
         5A    20    8A  0A69  2682         BICB    #IOC$M_2P,R10           ; new DDB - clear secondary flag
      5E 5A    01    E0  0A6C  2683         BBS     #IOC$V_TYPE,R10,100$    ; branch if generic type search
      07 5A    02    E1  0A70  2684         BBC     #IOC$V_CLASS,R10,40$    ; branch if no class to check
   3C A6    04 AE    D1  0A74  2685         CMPL    SAVR3(SP),DDB$L_ALLOCLS(R6) ; else, is allo. class right?
               E2    12  0A79  2686         BNEQ    30$                     ; branch if not, try next DDB
15 A6    69    58    29  0A7B  2687 40$:    CMPC3   R8,(R9),DDB$T_NAME+1(R6) ; check device name
               DB    12  0A80  2688         BNEQ    30$                     ; if no match, try next DDB
      50    14 A6    9A  0A82  2689         MOVZBL  DDB$T_NAME(R6),R0       ; get length of name in DDB
         50    58    D1  0A86  2690         CMPL    R8,R0                   ; check name lengths
               43    13  0A89  2691         BEQL    100$                    ; if they match - OK
               50    D7  0A8B  2692         DECL    R0                      ; try subtracting out controller letter
         50    58    D1  0A8D  2693         CMPL    R8,R0                   ; and see if this matches
               CB    12  0A90  2694         BNEQ    30$                     ; if not, keep trying
            39 5A    E9  0A92  2695         BLBC    R10,100$                ; branch if not physical search - OK
41 8F    15 A640    91  0A95  2696         CMPB    DDB$T_NAME+1(R6)[R0],#^A'A' ; is this controller A?
               31    13  0A9B  2697         BEQL    100$                    ; if so, search it
               BE    11  0A9D  2698         BRB     30$                     ; if not, keep looking
                     0A9F  2699 ;
                     0A9F  2700 ; End of search - no suitable device has been found
```

```
                                                    K 7
                     0A9F   2701 ;
                     0A9F   2702 50$:    MOVZWL  #SS$_NOSUCHDEV,R0          ; no device found
 4D 5A   04          0AA4   2703         BBC     #IOC$V_EXISTS,R10,140$     ; branch if not seen
50  09B0 8F          0AA8   2704         MOVZWL  #SS$_NODEVAVL,R0           ; otherwise status is not available
        46   11      0AAD   2705         BRB     140$
                     0AAF   2706 ;
                     0AAF   2707 ; To here if we're just returning a system block, with no device specified.
                     0AAF   2708 ;
    56   66   D0      0AAF   2709 60$:    MOVL    (R6),R6                    ; get first DDB
 55   04 A6   D0      0AB2   2710         MOVL    DDB$L_UCB(R6),R5           ; and first UCB
        3D   11      0AB6   2711         BRB     140$                       ; and return
                     0AB8   2712 ;
                     0AB8   2713 ; To here when all UCB's on a DDB have been searched.
                     0AB8   2714 ;
A1 5A   01   E0      0AB8   2715 70$:    BBS     #IOC$V_TYPE,R10,30$        ; if generic type search, try next DDB
                     0ABC   2716 ;
                     0ABC   2717 ; To here when all DDB's on a system block have been searched.
                     0ABC   2718 ;
   5A   06   93      0ABC   2719 80$:    BITB    #IOC$M_CLASS!IOC$M_TYPE,R10 ; if generic type or alloc class
        0A   12      0ABF   2720         BNEQ    90$                        ; keep searching system blocks
   5A   09   93      0AC1   2721         BITB    #IOC$M_PHY!IOC$M_LOCAL,R10 ; if physical or local only
        D9   12      0AC4   2722         BNEQ    50$                        ; we're done
   04 AE   D5        0AC6   2723         TSTL    SAVR3(SP)                  ; if there was an explicit node
        D4   12      0AC9   2724         BNEQ    50$                        ; we're done
    FF4D    31       0ACB   2725 90$:    BRW     10$                        ; else go try next system block
                     0ACE   2726 ;
                     0ACE   2727 ; Found a suitable DDB. Search both its UCB lists for the right UCB.
                     0ACE   2728 ;
    52   6E   7D      0ACE   2729 100$:   MOVQ    SAVR2(SP),R2               ; get unit number and device type
54  00000000'EF   D0  0AD1   2730         MOVL    SCH$GL_CURPCB,R4           ; get PCB address
                     0AD8   2731 NEXTUCB:                                   ; re-entry for next UCB
 07 5A   05   E1      0AD8   2732 110$:   BBC     #IOC$V_2P,R10,120$         ; branch if on primary path
55   00A4 C5   D0      0ADC   2733         MOVL    UCB$L_2P_LINK(R5),R5       ; link to next secondary unit.
        04   11      0AE1   2734         BRB     130$
 55   30 A5   D0      0AE3   2735 120$:   MOVL    UCB$L_LINK(R5),R5          ; link to next primary unit.
        11   13      0AE7   2736 130$:   BEQL    150$                       ; branch if no more units.
        28   10      0AE9   2737         BSBB    IOC$TESTUNIT               ; is this unit ok?
     07 50   E8      0AEB   2738         BLBS    R0,140$                    ; branch if successful
E6 5A   04   E1      0AEE   2739         BBC     #IOC$V_EXISTS,R10,110$     ; keep going if we haven't seen it yet
    E3 5A   E9       0AF2   2740         BLBC    R10,110$                   ; or if not physical search
   031C 8F   BA      0AF5   2741 140$:   POPR    #^M<R2,R3,R4,R8,R9>        ; restore registers
        05   0AF9   2742         RSB                                        ; and return
                     0AFA   2743
BA 5A   05   E2      0AFA   2744 150$:   BBSS    #IOC$V_2P,R10,70$          ; branch if secondary path already searched
55   9C A6   DE      0AFE   2745         MOVAL   <DDB$L_2P_UCB -            ; initialize secondary UCB address.
                     0B02   2746                 -UCB$L_2P_LINK>(R6),R5
        D4   11      0B02   2747         BRB     110$                       ; go search secondary path
                     0B04   2748
                     0B04   2749         .DISABLE LSB
```

```
                                                L 7
0B04  2751          .SBTTL  Continue I/O Database Search
0B04  2752
0B04  2753 ;+
0B04  2754 ;
0B04  2755 ; IOC$SEARCHCONT - internal I/O database search
0B04  2756 ;
0B04  2757 ; This routine continues a search started with a call to IOC$SEARCHINT.
0B04  2758 ; It uses IOC$SEARCHINT's outputs as the starting point at which to
0B04  2759 ; resume.
0B04  2760 ;
0B04  2761 ; INPUTS:
0B04  2762 ;
0B04  2763 ;       R2 = unit number
0B04  2764 ;       R3 = length of SCS node name at head of name string
0B04  2765 ;            or allocation class number
0B04  2766 ;            or device type code
0B04  2767 ;       R5 = last UCB
0B04  2768 ;       R6 = last DDB
0B04  2769 ;       R7 = last system block
0B04  2770 ;       R8 = size of name string
0B04  2771 ;       R9 = address of name string
0B04  2772 ;       R10 = flags
0B04  2773 ;       R11 = address to store lock value block
0B04  2774 ;       I/O database mutex held, IPL 2
0B04  2775 ;
0B04  2776 ; OUTPUTS:
0B04  2777 ;
0B04  2778 ;       R0 = SS$_NORMAL - device found
0B04  2779 ;          = SS$_NOSUCHDEV - device not found
0B04  2780 ;          = SS$_NODEVAVL - device exists but not available according to rules
0B04  2781 ;          = SS$_DEVALLOC - device allocated to other user
0B04  2782 ;          = SS$_NOPRIV - failed device protection
0B04  2783 ;          = SS$_TEMPLATEDEV - can't allocate template device
0B04  2784 ;          = SS$_DEVMOUNT - device already mounted
0B04  2785 ;          = SS$_DEVOFFLINE - device marked offline
0B04  2786 ;       R5 = UCB
0B04  2787 ;       R6 = DDB
0B04  2788 ;       R7 = system block
0B04  2789 ;       R10 - R4, R8 - R11 preserved
0B04  2790 ;
0B04  2791 ;       Note: If failure, R5 - R7 point to the last structures looked at.
0B04  2792 ;
0B04  2793 ;-
0B04  2794
0B04  2795 IOC$SEARCHCONT::
 031C 8F   BB  0B04  2796          PUSHR   #^M<R2,R3,R4,R8,R9>     ; save registers
05 5A   08   E5  0B08  2797          BBCC    #IOC$V_ALT,R10,10$      ; check if alternate UCB in use
55  00A8 C5   D0  0B0C  2798          MOVL    UCB$L_DP_ALTUCB(R5),R5  ; link back to other to continue
      C5   11   0B11  2799 10$:     BRB     NEXTUCB                 ; continue search
```

```
                                                M 7
                     0B13  2801          .SBTTL  Check UCB Against Search Rules
                     0B13  2802
                     0B13  2803 ;+
                     0B13  2804 ;
                     0B13  2805 ; IOC$TESTUNIT - Check UCB Against Search Rules
                     0B13  2806 ;
                     0B13  2807 ; INPUTS:
                     0B13  2808 ;
                     0B13  2809 ;       R2 = unit number
                     0B13  2810 ;       R3 = device type code
                     0B13  2811 ;       R4 = PCB address
                     0B13  2812 ;       R5 = UCB address
                     0B13  2813 ;       R10 = flags
                     0B13  2814 ;       R11= address of lock value block
                     0B13  2815 ;
                     0B13  2816 ; OUTPUTS:
                     0B13  2817 ;
                     0B13  2818 ;       R0 = SS$_NORMAL - eligible for use according to flags
                     0B13  2819 ;          = SS$_NOSUCHDEV - wrong unit number
                     0B13  2820 ;          = SS$_DEVALLOC - device allocated to other user
                     0B13  2821 ;          = SS$_NOPRIV - failed device protection
                     0B13  2822 ;          = SS$_TEMPLATEDEV - can't allocate template device
                     0B13  2823 ;          = SS$_DEVMOUNT - device already mounted
                     0B13  2824 ;          = SS$_DEVOFFLINE - device marked offline
                     0B13  2825 ;
                     0B13  2826 ;-
                     0B13  2827
                     0B13  2828 IOC$TESTUNIT::
 50   0908 8F    3C  0B13  2829          MOVZWL  #SS$_NOSUCHDEV,R0       ; assume wrong device
        06 5A    E9  0B18  2830          BLBC    R10,10$                 ; branch if not physical search
   54 A5    52    B1  0B1B  2831          CMPW    R2,UCB$W_UNIT(R5)       ; is the unit number exactly right?
            56    12  0B1F  2832          BNEQ    70$                     ; branch to error if not right.
                  0B21  2833
   09 5A    01    E1  0B21  2834 10$:     BBC     #IOC$V_TYPE,R10,20$     ; branch if not searching for dev type
            00    ED  0B25  2835          CMPZV   #MSCP$V_MTYP_N,-
            16        0B27  2836                  #MSCP$V_MTYP_D1,-
 53   008C C5        0B28  2837                  UCB$L_MEDIA_ID(R5),R3   ; is this the requested type?
            49    12  0B2C  2838          BNEQ    70$                     ; branch if not
      5A    10    88  0B2E  2839 20$:     BISB    #IOC$M_EXISTS,R10       ; note eligible device seen
0A 3C A5    03    E1  0B31  2840          BBC     #DEV$V_CDP,UCB$L_DEVCHAR2(R5),30$ ; is this served path to a local d
 55   00A8 C5    D0  0B36  2841          MOVL    UCB$L_DP_ALTUCB(R5),R5  ; yes, get local path UCB address.
 5A   0100 8F    A8  0B3B  2842          BISW    #IOC$M_ALT,R10          ; note alternate UCB in use
   03 5A    06    E1  0B40  2843 30$:     BBC     #IOC$V_ANY,R10,40$      ; if SEARCHALL, finish with success.
         0091    31  0B44  2844          BRW     150$
                  0B47  2845 ;
                  0B47  2846 ; Check the device reference count and allocation status.
                  0B47  2847 ;
 50   006C 8F    3C  0B47  2848 40$:     MOVZWL  #SS$_DEVMOUNT,R0        ; check if device is already mounted
55 38 A5    13    E0  0B4C  2849          BBS     #DEV$V_MNT,UCB$L_DEVCHAR(R5),100$
 50   0840 8F    3C  0B51  2850          MOVZWL  #SS$_DEVALLOC,R0
4B 64 A5    09    E0  0B56  2851          BBS     #UCB$V_MOUNTING,UCB$W_STS(R5),100$ ; branch if mount in progress
         5C A5    B5  0B5B  2852          TSTW    UCB$W_REFC(R5)          ; is reference count zero?
            19    13  0B5E  2853          BEQL    80$                     ; branch if reference count is zero.
   0B 5A    07    E1  0B60  2854          BBC     #IOC$V_MOUNT,R10,50$    ; if mounting...
   0A 5A    0A    E0  0B64  2855          BBS     #IOC$V_ALLOC,R10,60$    ; if shared mount
0C 38 A5    17    E1  0B68  2856          BBC     #DEV$V_ALL,UCB$L_DEVCHAR(R5),80$ ; OK if not allocated
            03    11  0B6D  2857          BRB     60$                     ; otherwise check allocation
```

```
                                                N 7
                     0B6F  2858
          34 5A   E9 0B6F  2859 50$:    BLBC    R10,100$                        ; allocate: error if not phy
60 A4     2C A5   D1 0B72  2860 60$:    CMPL    UCB$L_PID(R5),PCB$L_PID(R4) ; does this process own the device?
             2D   12 0B77  2861 70$:    BNEQ    100$                            ; branch to error if not our device.
                     0B79  2862 ;
                     0B79  2863 ; Check all the other miscellaneous junk that can make a device not
                     0B79  2864 ; available.
                     0B79  2865 ;
          50   24   3C 0B79  2866 80$:    MOVZWL  #SS$_NOPRIV,R0                  ; check if device is spooled
06 38 A5   06   E1 0B7C  2867             BBC     #DEV$V_SPL,UCB$L_DEVCHAR(R5),90$ ; branch if not
                     0B81  2868             IFNPRIV ALLSPOOL,100$,R4                ; else, process must have ALLSPOOL priv.
  50   0084 8F   3C 0B87  2869 90$:    MOVZWL  #SS$_DEVOFFLINE,R0              ; check if device is available
15 38 A5   12   E1 0B8C  2870             BBC     #DEV$V_AVL,UCB$L_DEVCHAR(R5),100$
10 64 A5   04   E1 0B91  2871             BBC     #UCB$V_ONLINE,UCB$W_STS(R5),100$
  50   21DC 8F   3C 0B96  2872             MOVZWL  #SS$_TEMPLATEDEV,R0             ; check if device is a template
06 64 A5   0D   E0 0B9B  2873             BBS     #UCB$V_TEMPLATE,UCB$W_STS(R5),100$
        F45D'  30 0BA0  2874             BSBW    EXE$CHKRDACCES                  ; check device protection
        0A 50  E8 0BA3  2875             BLBS    R0,120$                         ; continue if accessible
                     0BA6  2876 ;
                     0BA6  2877 ; To here on any error.
                     0BA6  2878 ;
  05 5A   08   E5 0BA6  2879 100$:   BBCC    #IOC$V_ALT,R10,110$             ; check if alternate UCB in use
55   00A8 C5   D0 0BAA  2880             MOVL    UCB$L_DP_ALTUCB(R5),R5  ; link back to other to continue
             05   0BAF  2881 110$:   RSB                                     ; return
                     0BB0  2882 ;
                     0BB0  2883 ; We've passed all the local tests. Now try to take out the appropriate
                     0BB0  2884 ; lock on the device.
                     0BB0  2885 ;
          51   5B   D0 0BB0  2886 120$:   MOVL    R11,R1                          ; value block address
               05   13 0BB3  2887             BEQL    130$                    ; branch if none
               61   7C 0BB5  2888             CLRQ    (R1)                    ; initialize value block
            08 A1   7C 0BB7  2889             CLRQ    8(R1)
19 3C A5       00   E1 0BBA  2890 130$:   BBC     #DEV$V_CLU,UCB$L_DEVCHAR2(R5),150$ ; br. if not cluster visible
          50   05   D0 0BBF  2891             MOVL    #LCK$K_EXMODE,R0        ; assume exclusive lock
     0C 5A   0A   E0 0BC2  2892             BBS     #IOC$V_ALLOC,R10,140$   ; branch if allocation requested
     08 5A   07   E1 0BC6  2893             BBC     #IOC$V_MOUNT,R10,140$   ; branch if not mount mode
03 38 A5       17   E0 0BCA  2894             BBS     #DEV$V_ALL,UCB$L_DEVCHAR(R5),140$ ; br. if allocated
          50   04   D0 0BCF  2895             MOVL    #LCK$K_PWMODE,R0        ; mount, no allocation - use PW
            F42B'  30 0BD2  2896 140$:   BSBW    IOC$LOCK_DEV                    ; and try to take device lock
           CE 50   E9 0BD5  2897             BLBC    R0,100$
          50   01   D0 0BD8  2898 150$:   MOVL    #SS$_NORMAL,R0                  ; indicate success
                     05 0BDB  2899             RSB
```

```
                                                B 8
                                0BDC  2901          .SBTTL  IOC$THREADCRB
                                0BDC  2902
                                0BDC  2903 ;++
                                0BDC  2904 ;
                                0BDC  2905 ; FUNCTIONAL DESCRIPTION:
                                0BDC  2906 ;
                                0BDC  2907 ;       This routine will thread a CRB onto the duetime chain headed by
                                0BDC  2908 ;       IOC$CRBTMOUT.
                                0BDC  2909 ;
                                0BDC  2910 ; CALLING SEQUENCE:
                                0BDC  2911 ;
                                0BDC  2912 ;       JSB     IOC$THREADCRB
                                0BDC  2913 ;
                                0BDC  2914 ; INPUTS:
                                0BDC  2915 ;
                                0BDC  2916 ;       R3 -->  CRB
                                0BDC  2917 ;
                                0BDC  2918 ; OUTPUTS:
                                0BDC  2919 ;
                                0BDC  2920 ;       NONE
                                0BDC  2921 ;
                                0BDC  2922 ;--
                                0BDC  2923
                                0BDC  2924 IOC$THREADCRB::
                         50     DD 0BDC  2925          PUSHL   R0                      ; Save a register
              50  00000000'GF   DE 0BDE  2926          MOVAL   G^IOC$GL_CRBTMOUT, R0   ; Pointer to list head
                         60     D5 0BE5  2927 10$:     TSTL    (R0)                    ; End of the line?
                         05     13 0BE7  2928          BEQL    20$                     ; Yes, go add new one
                      50 60     D0 0BE9  2929          MOVL    (R0), R0                ; No, get next block
                         F7     11 0BEC  2930          BRB     10$                     ; Try, try again
                                0BEE  2931
                   60 14 A3     DE 0BEE  2932 20$:     MOVAL   CRB$L_TIMELINK(R3),(R0) ; Link the new block in
                      50 8ED0   0BF2  2933          POPL    R0                      ; Restore register
                         05     0BF5  2934          RSB                             ; Leave
                                0BF6  2935
                                0BF6  2936
                                0BF6  2937          .END
```

```
$$BASE                  =  00000001
$$DISPL                 =  00000008
$$GENSW                 =  00000001
$$HIGH                  =  00000007
$$LIMIT                 =  00000006
$$LOW                   =  00000001
$$MNSW                  =  00000001
$$MXSW                  =  00000001
ADD_DOLLAR                 00006CA R      02
ADD_NODE                   000006C0 R     02
ADP$C_NUMDATAP          =  00000010
ADP$L_CSR               =  00000000
ADP$L_DPQBL             =  00000018
ADP$L_DPQFL             =  00000014
ADP$L_MRACTMDRS         =  0000005C
ADP$L_MRQBL             =  00000034
ADP$L_MRQFL             =  00000030
ADP$W_ADPTYPE           =  0000000E
ADP$W_DPBITMAP          =  00000060
ADP$W_MRFREGARY         =  0000015E
ADP$W_MRNREGARY         =  00000064
ALLOC_DESCRIP              000004DF R     02
ALLOC_NAME                 0000068D R     02
AT$_UBA                 =  00000001
BINNUM                     00000000
BOO$GL_SPTFREH             ********   X   02
BOO$GL_SPTFREL             ********   X   02
BUG$_INCONSTATE            ********   X   02
BUG$_IVBYTEALGN            ********   X   02
BUG$_UNSUPRTCPU            ********   X   02
CAN$C_AMBXDGN           =  00000002
CAN$C_DASSGN            =  00000001
CDRP$C_BCNT             =  FFFFFFD2
CDRP$L_FPC              =  0000000C
CDRP$L_FQFL             =  00000000
CDRP$L_FR3              =  00000010
CDRP$L_FR4              =  00000014
CDRP$L_IOQFL            =  FFFFFFA0
CDRP$L_RWCPTR           =  00000028
CDRP$L_UBARSRCE         =  0000003C
CDRP$W_BOFF             =  FFFFFFD0
CLU$GL_CLUB                ********   X   02
COM$DRVDEALMEM             ********   X   02
COMMON_ALOUBAMAP           0000036D R     02
CRB$B_MASK              =  0000000E
CRB$L_INTD              =  00000024
CRB$L_LINK              =  00000020
CRB$L_TIMELINK          =  00000014
CRB$L_WQBL              =  00000004
CRB$L_WQFL              =  00000000
CRB$M_BSY               =  00000001
CRB$V_BSY               =  00000000
DC$_DISK                =  00000001
DDB$L_2P_UCB            =  00000040
DDB$L_ALLOCLS           =  0000003C
DDB$L_DP_UCB            =  00000040
DDB$L_LINK              =  00000000
DDB$L_SB                =  00000034
DDB$L_UCB               =  00000004
DDB$T_NAME              =  00000014
DDT$L_CANCEL            =  0000000C
DDT$L_REGDUMP           =  00000010
DDT$L_START             =  00000000
DDT$L_UNITINIT          =  00000018
DEALLOC_DESCRIP            000004C6 R     02
DEV$M_MBX               =  00100000
DEV$M_TRM               =  00000004
DEV$V_2P                =  00000004
DEV$V_ALL               =  00000017
DEV$V_AVL               =  00000012
DEV$V_CDP               =  00000003
DEV$V_CLU               =  00000000
DEV$V_FOD               =  0000000E
DEV$V_MNT               =  00000013
DEV$V_NNM               =  00000009
DEV$V_OPR               =  00000007
DEV$V_SPL               =  00000006
DEV$V_TRM               =  00000002
DIR...                  =  00000001
DISKCHK                    00000198 R     02
DISPLAY_NAME               000006AA R     02
DO_PMS                     000001B0 R     02
DYN$C_TWP               =  00000030
DYN$C_UCB               =  00000010
EMB$B_DV_ERTCNT         =  00000C10
EMB$Q_DV_IOSB           =  00000012
EMB$W_DV_STS            =  0000001A
END_BROADCAST              00000796 R     02
END_CONBRDCST              000007EC R     02
ERL$RELEASEMB              ********   X   02
EXDVNM                     000006E3 R     02
EXE$ALONONPAGED            ********   X   02
EXE$ALTQUEPKT              ********   X   02
EXE$CHKRDACCES             ********   X   02
EXE$DEANONPAGED            ********   X   02
EXE$GB_CPUTYPE             ********   X   02
EXE$GL_ABSTIM              ********   X   02
EXE$GQ_SYSTIME             ********   X   02
EXE$MOUNTVER               ********   X   02
EXE$TEST_CSR               ********   X   02
FULL_NAME                  00000688 R     02
GETNUMBER                  000009B7 R     02
IDB$L_ADP               =  00000014
IDB$L_CSR               =  00000000
IDB$L_OWNER             =  00000004
IOC$ALLOSPT                0000062B RG    02
IOC$ALODATAP               00000268 R     02
IOC$ALOMAPUDA              0000031B R     02
IOC$ALOUBAMAP              00000345 RG    02
IOC$ALOUBAMAPN             0000033E RG    02
IOC$ALOUBAMAPSP            000003AF RG    02
IOC$ALOUBMAPRM             00000455 RG    02
IOC$ALOUBMAPRMN            0000044E RG    02
IOC$ALTREQCOM              00000118 RG    02
```

```
IOC$BROADCAST                   0000072B RG    02
IOC$CANCELIO                    00000000 RG    02
IOC$CONBRDCST                   0000079C RG    02
IOC$CREDIT_UCB                  ********   X   02
IOC$CTRLINIT                    0000088F RG    02
IOC$CVT_DEVNAM                  00000652 RG    02
IOC$DALOCUBAMAP                 00000573 R     02
IOC$DELETE_UCB                  ********   X   02
IOC$DIAGBUFILL                  0000005B RG    02
IOC$GL_CRBTMOUT                 ********   X   02
IOC$GL_PSBL                     ********   X   02
IOC$INITIATE                    000001DB RG    02
IOC$LAST_CHAN                   00000020 RG    02
IOC$LAST_CHAN_AMBX              00000017 RG    02
IOC$LOCK_DEV                    ********   X   02
IOC$MNTVER                      000001D2 RG    02
IOC$M_2P                    =   00000020
IOC$M_ALT                   =   00000100
IOC$M_CLASS                 =   00000004
IOC$M_EXISTS                =   00000010
IOC$M_LOCAL                 =   00000008
IOC$M_PHY                   =   00000001
IOC$M_TYPE                  =   00000002
IOC$PARSDEVNAM                  000008FC RG    02
IOC$RELCHAN                     0000008A RG    02
IOC$RELDATAP                    00000293 RG    02
IOC$RELDATAPUDA                 00000288 RG    02
IOC$RELMAPREG                   0000051A RG    02
IOC$RELMAPUDA                   000004FF RG    02
IOC$RELSCHAN                    00000080 RG    02
IOC$REQCOM                      00000143 RG    02
IOC$REQDATAP                    00000208 RG    02
IOC$REQDATAPNW                  0000021A RG    02
IOC$REQDATAPUDA                 00000228 RG    02
IOC$REQMAPREG                   00000309 RG    02
IOC$REQMAPUDA                   000002F4 RG    02
IOC$REQPCHANH                   000000E1 RG    02
IOC$REQPCHANL                   000000EA RG    02
IOC$REQSCHANH                   000000CD RG    02
IOC$REQSCHANL                   000000D7 RG    02
IOC$RETURN                      000005E4 RG    02
IOC$SCAN_IODB                   000007F0 RG    02
IOC$SCAN_IODB_2P                00000835 RG    02
IOC$SEARCHCONT                  00000B04 RG    02
IOC$SEARCHINT                   00000A10 RG    02
IOC$TESTUNIT                    00000B13 RG    02
IOC$THREADCRB                   00000BDC RG    02
IOC$UNITINIT                    000008C9 RG    02
IOC$V_2P                    =   00000005
IOC$V_ALLOC                 =   0000000A
IOC$V_ALT                   =   00000008
IOC$V_ANY                   =   00000006
IOC$V_CLASS                 =   00000002
IOC$V_EXISTS                =   00000004
IOC$V_MOUNT                 =   00000007
IOC$V_TYPE                  =   00000001
IOC$WFIKPCH                     000005E5 RG    02
IOC$WFIRLCH                     00000607 RG    02
IPL$_ASTDEL                 =   00000002
IPL$_IOPOST                 =   00000004
IPL$_QUEUEAST               =   00000006
IRP$L_DIAGBUF               =   0000004C
IRP$L_IOQFL                 =   00000000
IRP$L_MEDIA                 =   00000038
IRP$L_PID                   =   0000000C
IRP$L_SVAPTE                =   0000002C
IRP$L_UCB                   =   0000001C
IRP$V_DIAGBUF               =   00000007
IRP$W_CHAN                  =   00000028
IRP$W_STS                   =   0000002A
LCK$K_EXMODE                =   00000005
LCK$K_PWMODE                =   00000004
LOCAL_NAME                      000006CC R     02
MMG$GL_SPTBASE                  ********   X   02
MNTVERPNDCHK                    000001B8 R     02
MSCP$V_MTYP_D1              =   00000016
MSCP$V_MTYP_N               =   00000000
NEXTUCB                         00000AD8 R     02
NO_SECONDARY                    000006EB R     02
NXTIRP                          00000189 R     02
OPA$UCB0                        ********   X   02
PCB$L_PID                   =   00000060
PCB$Q_PRIV                  =   00000084
PDT$L_ADP                   =   000000E0
PMS$END_IO                      ********   X   02
PMS$GL_IOPFMPDB                 ********   X   02
PMS$START_IO                    ********   X   02
PMSEND                          0000017A R     02
PR$_IPL                     =   00C00012
PR$_SID_TYP730              =   00000003
PR$_SID_TYP750              =   00000002
PR$_SID_TYP780              =   00000001
PR$_SID_TYP790              =   00000004
PR$_SID_TYP8NN              =   00000006
PR$_SID_TYP8SS              =   00000005
PR$_SID_TYPUV1              =   00000007
PR$_SIRR                    =   00000014
PRV$V_ALLSPOOL              =   00000004
PUTASCIC                        00000708 R     02
PUTCHAR                         00000719 R     02
PUTDOLLAR                       00000716 R     02
PUTNUM                          000006F0 R     02
REALLOC_CD_MAPREGS              00000561 R     02
RELDATAP_COMMON                 0000029F R     02
RELEASE                         00000195 R     02
RESR0                           00000008
RESR1                           0000000C
RESR2                           00000010
RESR3                           00000014
RESR4                           00000018
SAVABS...                   =   0000001C
SAVED_R0                    =   00000000
SAVED_R1                    =   00000004
SAVED_R2                    =   00000008
```

```
SAVED_R3                = 0000000C
SAVED_R4                = 00000010
SAVED_R5                = 00000014
SAVR2                   = 00000000
SAVR3                   = 00000004
SAVR4                   = 00000008
SAVR8                   = 0000000C
SAVR9                   = 00000010
SB$L_DDB                = 00000054
SB$L_FLINK              = 00000000
SB$T_NODENAME           = 00000044
SCH$GL_CURPCB             ********    X   02
SCRLEN                    00000010
SCS$GA_LOCALSB            ********    X   02
SCS$GQ_CONFIG             ********    X   02
SCS$RESUMEWAITR           ********    X   02
SECONDARY_NAME            0000069A R      02
SS$_BUFFEROVF           = 00000601
SS$_DEVALLOC            = 00000840
SS$_DEVMOUNT            = 0000006C
SS$_DEVOFFLINE          = 00000084
SS$_ILLIOFUNC           = 000000F4
SS$_INSFMEM             = 00000124
SS$_IVDEVNAM            = 00000144
SS$_NODEVAVL            = 000009B0
SS$_NOPRIV              = 00000024
SS$_NORMAL              = 00000001
SS$_NOSUCHDEV           = 00000908
SS$_TEMPLATEDEV         = 000021DC
TTY$B_WB_FIPL           = 0000000B
TTY$B_WB_TYPE           = 0000000A
TTY$K_WB_LENGTH         = 00000030
TTY$L_WB_DATA           = 00000030
TTY$L_WB_END            = 00000020
TTY$L_WB_FR3            = 00000010
TTY$L_WB_IRP            = 00000024
TTY$L_WB_NEXT           = 0000001C
TTY$L_WB_RETADDR        = 0000002C
TTY$W_WB_SIZE           = 00000008
UBMD$B_DATAPATH         = 00000003
UBMD$B_NUMREG           = 00000002
UBMD$W_MAPREG           = 00000000
UCB$B_DEVCLASS          = 00000040
UCB$B_ERTCNT            = 00000080
UCB$B_FIPL              = 0000000B
UCB$B_TYPE              = 0000000A
UCB$L_2P_LINK           = 000000A4
UCB$L_CRB               = 00000024
UCB$L_DDB               = 00000028
UCB$L_DDT               = 00000088
UCB$L_DEVCHAR           = 00000038
UCB$L_DEVCHAR2          = 0000003C
UCB$L_DP_ALTUCB         = 000000A8
UCB$L_DP_DDB            = 000000A0
UCB$L_DP_LINK           = 000000A4
UCB$L_DUETIM            = 0000006C
UCB$L_EMB               = 00000094
UCB$L_FPC               = 0000000C
UCB$L_FQFL              = 00000000
UCB$L_FR3               = 00000010
UCB$L_IOQFL             = 0000004C
UCB$L_IRP               = 00000058
UCB$L_LINK              = 00000030
UCB$L_MEDIA_ID          = 0000008C
UCB$L_OPCNT             = 00000070
UCB$L_PID               = 0000002C
UCB$L_STS               = 00000064
UCB$L_SVAPTE            = 00000078
UCB$M_BSY               = 00000100
UCB$M_CANCEL            = 00000008
UCB$M_INT               = 00000002
UCB$M_TIM               = 00000001
UCB$M_TIMOUT            = 00000040
UCB$V_BSY               = 00000008
UCB$V_DELETEUCB         = 00000010
UCB$V_ERLOGIP           = 00000002
UCB$V_MNTVERIP          = 0000000E
UCB$V_MNTVERPND         = 00000013
UCB$V_MOUNTING          = 00000009
UCB$V_ONLINE            = 00000004
UCB$V_TEMPLATE          = 0000000D
UCB$W_BCNT              = 0000007E
UCB$W_BOFF              = 0000007C
UCB$W_REFC              = 0000005C
UCB$W_STS               = 00000064
UCB$W_UNIT              = 00000054
VEC$B_DATAPATH          = 00000013
VEC$B_NUMREG            = 00000012
VEC$L_ADP               = 00000014
VEC$L_IDB               = 00000008
VEC$L_INITIAL           = 0000000C
VEC$L_UNITINIT          = 00000018
VEC$M_MAPLOCK           = 00008000
VEC$S_DATAPATH          = 00000005
VEC$V_DATAPATH          = 00000000
VEC$V_MAPLOCK           = 0000000F
VEC$V_PATHLOCK          = 00000007
VEC$W_MAPREG            = 00000010
```

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	0000001C (28.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
WIONONPAGED	00000BF6 (3062.)	02 (2.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC BYTE

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	29	00:00:00.06	00:00:01.71
Command processing	106	00:00:00.55	00:00:04.30
Pass 1	693	00:00:31.36	00:01:37.67
Symbol table sort	0	00:00:04.39	00:00:11.34
Pass 2	403	00:00:08.26	00:00:26.97
Symbol table output	1	00:00:00.25	00:00:00.66
Psect synopsis output	0	00:00:00.01	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	1234	00:00:44.90	00:02:22.68

The working set limit was 2400 pages.
182054 bytes (356 pages) of virtual memory were used to buffer the intermediate code.
There were 150 pages of symbol table space allocated to hold 2771 non-local and 169 local symbols.
2937 source lines were read in Pass 1, producing 24 object records in Pass 2.
59 pages of virtual memory were used to define 55 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA28:[SYS.OBJ]LIB.MLB;1	35
_255DUA28:[SYSLIB]STARLET.MLB;2	12
TOTALS (all libraries)	47

3009 GETS were required to define 47 macros.

There were no errors, warnings or information messages.

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

IOSUBRAMS
LIS

IPCONTROL
LIS

IOSUBPAGD
LIS

LNMSUB
LIS

IOPERFORM
LIS

IOSUBNPAG
LIS

LINKVEC
LIS