```
DDDDDDDDDDDD     RRRRRRRRRRRR        IIIIIIIII     VVV         VVV  EEEEEEEEEEEEEEE  RRRRRRRRRRRR
DDDDDDDDDDDD     RRRRRRRRRRRR        IIIIIIIII     VVV         VVV  EEEEEEEEEEEEEEE  RRRRRRRRRRRR
DDDDDDDDDDDD     RRRRRRRRRRRR        IIIIIIIII     VVV         VVV  EEEEEEEEEEEEEEE  RRRRRRRRRRRR
DDD          DDD RRR          RRR       III       VVV         VVV  EEE              RRR          RRR
DDD          DDD RRR          RRR       III       VVV         VVV  EEE              RRR          RRR
DDD          DDD RRR          RRR       III       VVV         VVV  EEE              RRR          RRR
DDD          DDD RRR          RRR       III       VVV         VVV  EEE              RRR          RRR
DDD          DDD RRR          RRR       III       VVV         VVV  EEE              RRR          RRR
DDD          DDD RRR          RRR       III       VVV         VVV  EEE              RRR          RRR
DDD          DDD RRRRRRRRRRRR           III       VVV         VVV  EEEEEEEEEEE      RRRRRRRRRRRR
DDD          DDD RRRRRRRRRRRR           III       VVV         VVV  EEEEEEEEEEE      RRRRRRRRRRRR
DDD          DDD RRRRRRRRRRRR           III       VVV         VVV  EEEEEEEEEEE      RRRRRRRRRRRR
DDD          DDD RRR   RRR              III       VVV         VVV  EEE              RRR   RRR
DDD          DDD RRR   RRR              III       VVV         VVV  EEE              RRR   RRR
DDD          DDD RRR   RRR              III       VVV         VVV  EEE              RRR   RRR
DDD          DDD RRR      RRR           III          VVV   VVV     EEE              RRR      RRR
DDD          DDD RRR      RRR           III          VVV   VVV     EEE              RRR      RRR
DDD          DDD RRR      RRR           III          VVV   VVV     EEE              RRR      RRR
DDDDDDDDDDDD     RRR         RRR     IIIIIIIII          VVV       EEEEEEEEEEEEEEE  RRR         RRR
DDDDDDDDDDDD     RRR         RRR     IIIIIIIII          VVV       EEEEEEEEEEEEEEE  RRR         RRR
DDDDDDDDDDDD     RRR         RRR     IIIIIIIII          VVV       EEEEEEEEEEEEEEE  RRR         RRR
```

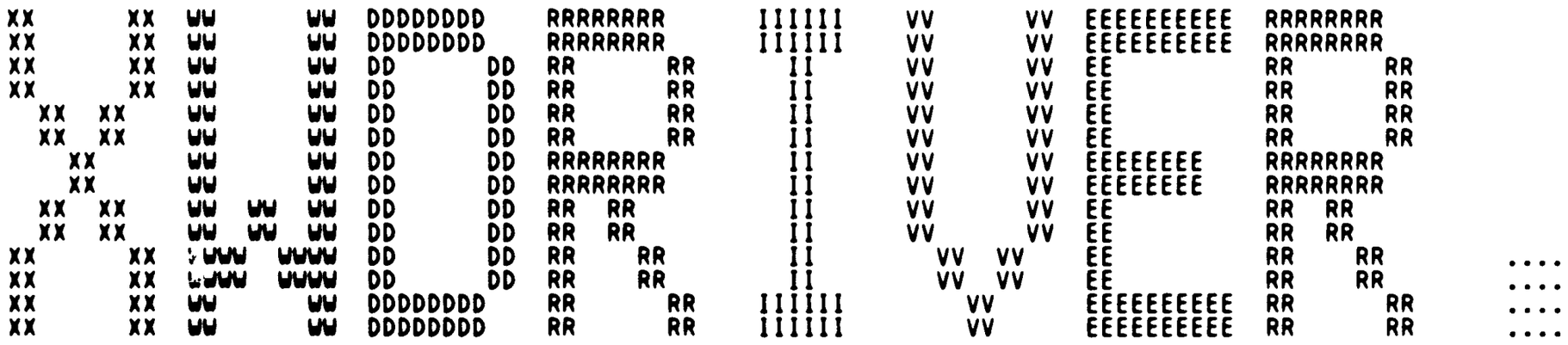
XX XX WW WW DDDDDDDD RRRRRRRR IIIIII VV VV EEEEEEEEEE RRRRRRRR
XX XX WW WW DDDDDDDD RRRRRRRR IIIIII VV VV EEEEEEEEEE RRRRRRRR
XX XX WW WW DD DD RR RR II VV VV EE RR RR
XX XX WW WW DD DD RR RR II VV VV EE RR RR
XX XX WW WW DD DD RR RR II VV VV EE RR RR
XX XX WW WW DD DD RR RR II VV VV EE RR RR
XX XX WW WW DD DD RRRRRRRR II VV VV EEEEEEEE RRRRRRRR
XX XX WW WW DD DD RRRRRRRR II VV VV EEEEEEEE RRRRRRRR
XX XX WW WW WW DD DD RR RR II VV VV EE RR RR
XX XX WW WW WW DD DD RR RR II VV VV EE RR RR
XX XX WWWW WWWW DD DD RR RR II VV VV EE RR RR ....
XX XX WWWW WWWW DD DD RR RR II VV VV EE RR RR ....
XX XX WW WW DDDDDDDD RR RR IIIIII VV EEEEEEEEEE RR RR ....
XX XX WW WW DDDDDDDD RR RR IIIIII VV EEEEEEEEEE RR RR ....
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          .SBTTL  MODULE - XWDRIVER
0000     2          .TITLE  XWDRIVER - BSC DUP11 Device Driver
0000     3          .IDENT  'V04-000'
0000     4
0000     5  ;
0000     6  ;*****************************************************************************
0000     7  ;*                                                                           *
0000     8  ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
0000     9  ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000    10  ;*  ALL RIGHTS RESERVED.                                                     *
0000    11  ;*                                                                           *
0000    12  ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED     *
0000    13  ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE    *
0000    14  ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER     *
0000    15  ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY     *
0000    16  ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY     *
0000    17  ;*  TRANSFERRED.                                                             *
0000    18  ;*                                                                           *
0000    19  ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE     *
0000    20  ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
0000    21  ;*  CORPORATION.                                                             *
0000    22  ;*                                                                           *
0000    23  ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS     *
0000    24  ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
0000    25  ;*                                                                           *
0000    26  ;*                                                                           *
0000    27  ;*****************************************************************************
0000    28  ;
0000    29  ; Modified by:
0000    30  ;
0000    31  ;       V03-003 KDM0002         Kathleen D. Morse       28-Jun-1982
0000    32  ;               Added $DCDEF, $DYNDEF, $PRDEF, and $SSDEF.
0000    33  ;
0000    34  ;       V03-002 MMD0002         Meg Dumont,     4-Jun-1982  11:16
0000    35  ;               Fix UCB$W_INT to be UCB$M_INT.
0000    36  ;
0000    37  ;       V03-001 MMD0001         Meg Dumont,     19-May-1982  14:46
0000    38  ;               Add known fixes from XJDRIVER to this driver
0000    39  ;
0000    40  ;
0000    41  ;
0000    42  ; MACRO LIBRARY CALLS
0000    43  ;
0000    44
0000    45          $XWDEF
0000    46          $EBCDEF
0000    47          $CRBDEF                         ;DEFINE CRB OFFSETS
0000    48          $DCDEF                          ;DEFINE DATA TYPES
0000    49          $DDBDEF                         ;DEFINE DDB OFFSETS
0000    50          $DPTDEF                         ;DEFINE DPT OFFSETS
0000    51          $DYNDEF                         ;DEFINE DYNAMIC DATA STRUCTURE TYPES
0000    52          $IDBDEF                         ;DEFINE IDB OFFSETS
0000    53          $IODEF                          ;DEFINE I/O FUNCTION CODES
0000    54          $IRPDEF                         ;DEFINE IRP OFFSETS
0000    55          $JIBDEF
0000    56          $MSGDEF                         ;DEFINE SYSTEM MESSAGE TYPES
0000    57          $PCBDEF                         ;DEFINE PCB OFFSETS
```

```
                     0000     58          $PRDEF                          ;DEFINE PROCESSOR REGISTERS
                     0000     59          $UCBDEF                         ;DEFINE UCB OFFSETS
                     0000     60          $VADEF                          ;DEFINE VIRTUAL ADDRESS FIELDS
                     0000     61          $VECDEF                         ;DEFINE VEC OFFSETS
                     0000     62
                     0000     63 ;
                     0000     64 ; LOCAL SYMBOLS
                     0000     65 ;
                     0000     66 ; ARGUMENT LIST OFFSET DEFINITIONS
                     0000     67 ;
                     0000     68
00000000             0000     69 P1=0                                    ;FIRST FUNCTION DEPENDENT PARAMETER
00000004             0000     70 P2=4                                    ;SECOND FUNCTION DEPENDENT PARAMETER
00000008             0000     71 P3=8                                    ;THIRD FUNCTION DEPENDENT PARAMETER
0000000C             0000     72 P4=12                                   ;FOURTH FUNCTION DEPENDENT PARAMETER
00000010             0000     73 P5=16                                   ;FIFTH FUNCTION DEPENDENT PARAMETER
00000014             0000     74 P6=20                                   ;SIZTH FUNCTION DEPENDENT PARAMETER
                     0000     75
                     0000     76 ;
                     0000     77 ; Error recording macro
                     0000     78 ;
                     0000     79          .MACRO  PROTLO  ERRNO
                     0000     80          MOVZBL  #ERRNO,-(SP)               ; Put the error on the stack
                     0000     81          BSBW    ERR_RECORD                 ; Call the error recorder
                     0000     82          .ENDM   PROTLO
                     0000     83
00000002             0000     84          REP=1*2                            ; Out of phase ACK after REP
00000004             0000     85          NAK=2*2                            ; RECEIVED NAKS
00000007             0000     86          NAKOUT=3*2!1                       ; Fatal nak retry
00000009             0000     87          BADACK=4*2!1                       ; Out-of-phase ACK other than as
                     0000     88                                             ;  a response to a ENQ
0000000B             0000     89          DSR_LOST=5*2!1                     ; DSR lost durring a buffer
0000000C             0000     90          GARBLED_RESP=6*2                   ; Unrecognizable response to text
0000000E             0000     91          CON_RESP=7*2                       ; Conversational response received
00000010             0000     92          REC_BKTOLONG=8*2                   ; Receive block to long for buffer
00000012             0000     93          REC_REP=9*2                        ; Reply lost (ENQ received)
00000014             0000     94          REC_FORABORT=10*2                  ; Forward abort received (ENQ in text)
00000016             0000     95          REC_TXTFORMAT=11*2                 ; Block format undecipherable
00000019             0000     96          REC_DISCON=12*2!1                  ; Disconnect received
0000001A             0000     97          ABEOT=13*2                         ; Abnormal EOT (durring xmt text)
0000001D             0000     98          REPOUT=14*2!1                      ; Transmit retry threshold
0000001F             0000     99          SEN_RES_TIMOUT=15*2!1              ; Timout durring response
00000020             0000    100          TMO_XMT_TXTBLK=16*2                ; Timout sending a text block
00000022             0000    101          NAK_SENT=17*2                      ; Nak transmitted
00000024             0000    102          REP_SENT=18*2                      ; Rep (ENQ) sent
                     0000    103
00000012             0000    104          NO_ERRORS=18                       ; Highest error number+3 rounded up
                     0000    105 ;        .LIST   MEB
                     0000    106 ;
                     0000    107 ;Define Device Dependent Unit Control Block Offsets
                     0000    108 ;
                     0000    109          $DEFINI UCB
0000009C             0000    110                  .=UCB$L_DPC
                     009C    111          $DEF    UCB$L_XQ_SPBF    .BLKL 1           ; Short protocol buffer
                     00A0    112          $DEF    UCB$L_XW_SYCNT .BLKL 1             ; Sync count (also offline ct)
                     00A4    113          $DEF    UCB$L_XW_ITBPTR .BLKL 1            ; Internal CRC pointer
                     00A8    114          $DEF    UCB$B_XW_ITBCNT .BLKB 1            ; Internal CRC count
```

```
                     00A9   115          $DEF    UCB$B_XW_TISTA  .BLKB 1          ; Interrupt state
                     00AA   116          $DEF    UCB$B_XW_RISTA  .BLKB 1          ; Receive interrupt state
                     00AB   117          $DEF    UCB$W_XW_LSTCSR .BLKW 1          ; CSR at end of last transfer
                     00AD   118          $DEF    UCB$B_XW_ERRORS .BLKB NO_ERRORS+2 ; Error talleys + 2 bytes
                     00C1   119                                                   ;   for current err & 0 spare
000000C1             00C1   120 UCB$K_SIZE=.
                     00C1   121
                     00C1   122          $DEFEND UCB
                     0000   123
                     0000   124 ;
                     0000   125 ; Device specific flags definition (for UCB$W_DEVSTS)
                     0000   126 ;
00000000             0000   127          UCB$V_XW_REP=0                          ; Retry state
00000001             0000   128          UCB$M_XW_REP=^O1                        ;...
                     0000   129
00000001             0000   130          UCB$V_XW_AK1=1                          ; ACK1 next
00000002             0000   131          UCB$M_XW_AK1=2                          ; ACK1 next
                     0000   132
00000002             0000   133          UCB$V_XW_NAK=2                          ; NAK sent last flag
00000004             0000   134          UCB$M_XW_NAK=^O4                        ; ...
                     0000   135                                                  ;  to a long word
                     0000   136
00000008             0000   137          UCB$M_XW_PIPE=^o10                      ; Cancel pipeline marker
00000003             0000   138          UCB$V_XW_PIPE=3
                     0000   139 ;
                     0000   140 ; DUP-11 CSR Offset And Bit Definitions
                     0000   141 ;
00000002             0000   142          DUP_CSR_RDBF=2                          ; Receiver data buffer
00000002             0000   143          DUP_CSR_PARM=2                          ; Parameter register
00000004             0000   144          DUP_CSR_TXCSR=4                         ; Transmitter command register
00000006             0000   145          DUP_CSR_TXDBF=6                         ; Transmitter data buffer
00000002             0000   146          DUP_CSR_M_DTR=^O2                       ; Data terminal ready
00000004             0000   147          DUP_CSR_M_RTS=^O4                       ; Request to send
00000010             0000   148          DUP_CSR_M_REN=^O20                      ; Receiver enable
00000040             0000   149          DUP_CSR_M_RIE=^O100                     ; Receiver interrupt enable
00000080             0000   150          DUP_CSR_M_RDN=^O200                     ; Receiver done
00000100             0000   151          DUP_CSR_M_SSY=^O400                     ; Strip SYN
00000200             0000   152          DUP_CSR_M_DSR=^O1000                    ; Data set ready
00002000             0000   153          DUP_CSR_M_CTS=^O20000                   ; Clear to send
00008000             0000   154          DUP_CSR_M_BYM=^O100000                  ; Byte mode
00000010             0000   155          DUP_CSR_M_SND=^O20                      ; Send
00000040             0000   156          DUP_CSR_M_TIE=^O100                     ; Transmit interrupt enable
00000080             0000   157          DUP_CSR_M_TDN=^O200                     ; Transmit done
00000100             0000   158          DUP_CSR_M_RES=^O400                     ; Reset
00000100             0000   159          DUP_CSR_M_SOM=^O400                     ; Transmit start of message
00000200             0000   160          DUP_CSR_M_EOM=^O1000                    ; Transmit end of message
00000200             0000   161          DUP_CSR_M_CRI=^O1000                    ; CRC inhibit bit
00000008             0000   162          DUP_CSR_M_HDX=^O10                      ; Half duplex bit
00001000             0000   163          DUP_CSR_M_ETM=^O10000                   ; Maintenance clock bit
00000800             0000   164          DUP_CSR_M_STM=^O4000                    ; System (internal)loop mode
                     0000   165 ;
                     0000   166 ; Transmit interrupt state definition
                     0000   167 ;
00000000             0000   168          TSTA$IDLE=0                             ; No interrupt expected
00000001             0000   169          TSTA$CTS=1                              ; Wait forCTS
00000002             0000   170          TSTA$SYN=2                              ; Transmit SYN
00000003             0000   171          TSTA$TEXT=3                             ; Transmit text
```

```
00000004  0000   172          TSTA$ENDPAD=4                   ; Transmit trailing pad
00000005  0000   173          TSTA$DIAG=5                     ; Diagnostic (no CTS wait) state
00000006  0000   174          TSTA$DROPRTS=6                  ; Drop RTS state
          0000   175
          0000   176 ;
          0000   177 ; Receive interrupt state definition
          0000   178 ;
00000000  0000   179          RST$IDLE=0                      ; No interrupts expected
00000001  0000   180          RST$BINARY=1                    ; Read until byte count runout
00000002  0000   181          RST$FIRST=2                     ; Starting state
00000003  0000   182          RST$NTTX=3                      ; Non-transparent text state
00000004  0000   183          RST$STOP1=4                     ; Last CRC byte
00000005  0000   184          RST$STOP=5                      ; Stop after next byte
00000006  0000   185          RST$ITB=6                       ; First byte of internal CRC next
00000007  0000   186          RST$ITB1=7                      ; Second byte of internal CRC next
00000008  0000   187          RST$XPRTEXT=8                   ; Transparent text state
00000009  0000   188          RST$DLE=9                       ; Control character or stuffed DLE next
0000000A  0000   189          RST$XPRITB=10                   ; First byte of internal CRC next
0000000B  0000   190          RST$XPRITB1=11                  ; Second byte of internal CRC next
0000000C  0000   191          RST$DIAG=12                     ; Diagnostic receive interrupt state
0000000D  0000   192          RST$LOOP_TEST=13                ; Diagnostic slave loop state
          0000   193 ;
          0000   194 ;DRIVER PROLOG TABLE
          0000   195 ;
          0000   196          DPTAB   -                       ; Define driver prolog table
          0000   197                  END=XW_END,-            ; End of driver
          0000   198                  ADAPTER=UBA,-           ; Adapter type
          0000   199                  UCBSIZE=UCB$K_SIZE,-    ; UCB size
          0000   200                  NAME=XWDRIVER           ; Driver name
          0038   201          DPT_STORE       INIT            ; Control block init values
          0038   202          DPT_STORE       UCB,-
          0038   203                          UCB$B_ERTMAX,B,8 ; Retry count
          003C   204          DPT_STORE       UCB,-           ; Fork IPL
          003C   205                          UCB$B_FIPL,B,8
          0040   206          DPT_STORE       UCB,UCB$L_DEVCHAR,L,- ; Device characteristics bits
          0040   207                          <DEV$M_AVL-     ; Device avalable
          0040   208                          !DEV$M_IDV-     ; Input device
          0040   209                          !DEV$M_ODV>     ; Output device
          0047   210          DPT_STORE       UCB,UCB$W_DEVBUFSIZ,- ; Device buffer size
          0047   211                          W,520
          004C   212          DPT_STORE       UCB,UCB$B_DIPL,B,21 ; Device IPL
          0050   213          DPT_STORE       REINIT          ; Fields to reinit. at driver reload
          0050   214          DPT_STORE       DDB,DDB$L_DDT,- ; Device dispatch table
          0050   215                          D,XW$DDT
          0055   216          DPT_STORE       CRB,CRB$L_INTD+4,- ; Receive interrupt entry point
          0055   217                          D,XW_RINT
          005A   218          DPT_STORE       CRB,CRB$L_INTD2+4,- ; Transmit interrupt entry point
          005A   219                          D,XW_TINT
          005F   220          DPT_STORE       CRB,CRB$L_INTD+VEC$L_UNITINIT,- ; Unit initalization
          005F   221                          D,XW_INIT
          0064   222          DPT_STORE       END
          0000   223 ;
          0000   224 ; Driver Dispatch Table
          0000   225 ;
          0000   226          DDTAB   XW,-                    ; Driver dispatch table
          0000   227                  XW_STARTIO,-            ; Start IO entry point
          0000   228                  0,-                     ; Unexpected interrupt
```

```
0000    229                     XW_FUNCTABLE,-          ; FDT lable
0000    230                     XW_CANCELIO,-           ; Cancel IO label
0000    231                     REGDUMP,-               ; Register dump routine
0000    232                     170,-                   ; Size of diagnostic buffer
0000    233                     170                     ; Size of errlog buffer
0038    234 ;
0038    235 ;FDT tables for BSC DUP-11 driver
0038    236 ;
0038    237 XW_FUNCTABLE:
0038    238             FUNCTAB ,-                      ; Legal functions
0038    239                     <READLBLK,-             ; Read logical block
0038    240                     READPBLK,-              ; Read physical block
0038    241                     WRITELBLK,-             ; Write logical block
0038    242                     WRITEPBLK,-             ; Write physical block
0038    243                     SETMODE,-               ; Set mode
0038    244                     SENSEMODE,-
0038    245                     >
0040    246 ;
0040    247             FUNCTAB ,-                      ; Legal functions
0040    248                     <READLBLK,-             ; Read logical block
0040    249                     READPBLK,-              ; Read  block
0040    250                     WRITELBLK,-             ; Write logical block
0040    251                     WRITEPBLK,-             ; Write physical block
0040    252                     SETMODE,-               ; Set mode
0040    253                     SENSEMODE,-
0040    254                     >
0048    255 ;
0048    256             FUNCTAB XW_WRITE,-              ; Write entry
0048    257                     <WRITEPBLK-             ; Write physical block
0048    258                     WRITELBLK-              ; Write logical block
0048    259                     >
0054    260 ;
0054    261             FUNCTAB XW_READ,-               ; Read processing entry
0054    262                     <READPBLK-              ; Read physical block
0054    263                     READLBLK-               ; Read logical block
0054    264                     >
0060    265 ;
0060    266             FUNCTAB +EXE$SETMODE,-          ; Set mode
0060    267                     SETMODE
006C    268 ;
006C    269             FUNCTAB +EXE$ZEROPARM,-
006C    270                     SENSEMODE
```

```
                     0078   272          .SBTTL  XWDRIVER Write FDT Routines
                     0078   273 ;+
                     0078   274 ; XW_WRITE - WRITE FUNCTION PROCESSING
                     0078   275 ;
                     0078   276 ; THIS ROUTINE IS CALLED FROM THE FUNCTION DECISION TABLE DISPATCHER TO PROCESS
                     0078   277 ; A WRITE LOGICAL OR WRITE PHYSICAL FUNCTION TO A DUP-11.
                     0078   278 ;
                     0078   279 ; INPUTS:
                     0078   280 ;
                     0078   281 ;       R0 = SCRATCH.
                     0078   282 ;       R1 = SCRATCH.
                     0078   283 ;       R2 = SCRATCH.
                     0078   284 ;       R3 = ADDRESS OF I/O REQUEST PACKET.
                     0078   285 ;       R4 = CURRENT PROCESS PCB ADDRESS.
                     0078   286 ;       R5 = ASSIGNED DEVICE UCB ADDRESS.
                     0078   287 ;       R6 = ADDRESS OF CCB.
                     0078   288 ;       R7 = I/O FUNCTION CODE.
                     0078   289 ;       R8 = FUNCTION DECISION TABLE DISPATCH ADDRESS.
                     0078   290 ;       R9 = SCRATCH.
                     0078   291 ;       R10 = SCRATCH.
                     0078   292 ;       R11 = SCRATCH.
                     0078   293 ;       AP = ADDRESS OF FIRST FUNCTION DEPENDENT PARAMETER.
                     0078   294 ;
                     0078   295 ; OUTPUTS:
                     0078   296 ;
                     0078   297 ;       THE FUNCTION PARAMETERS ARE CHECKED AND A BUFFER IS ALLOCATED FOR THE
                     0078   298 ;       DUP-11 DRIVER TO BUILD A MESSAGE INTO.
                     0078   299 ;-
                     0078   300
                     0078   301          .ENABL  LSB
                     0078   302 XW_WRITE:
    03    0D    EF   0078   303          EXTZV   #XW$V_IOMOD,#XW$S_IOMOD,- ; Extract the IO modifier field
  50    20 A3        007B   304                  IRP$W_FUNC(R3),R0
                     007E   305          CASE    R0,-                      ; Dispatch on the modifier
                     007E   306                  <XMT_BINARY,-             ; Send until count runout
                     007E   307                  XMT_PTPBSC,-              ; Add point to point BSC control
                     007E   308                  BAD_MODIFIER,-
                     007E   309                  BAD_MODIFIER,-
                     007E   310                  BAD_MODIFIER,-
                     007E   311                  BAD_MODIFIER,-
                     007E   312                  BAD_MODIFIER,-
                     007E   313                  XMT_DIAG,-                ; Full duplex diagnostic processing
                     007E   314                  >                         ; Last modifier
                     0092   315 ;
                     0092   316 ; Modifier was out of range so abort the request
                     0092   317 ;
                     0092   318 BAD_MODIFIER:
 51    0080 8F    3C   0092   319          MOVZWL  #XW$M_ILLMOD,R1           ; Get device dependant status
  00000000'GF    17   0097   320          JMP     G^EXE$ABORTIO             ; Abort the request
                     009D   321 ;
                     009D   322 ; Send the users buffer with no processing
                     009D   323 ;
                     009D   324 XMT_BINARY:
    50    6C    D0   009D   325          MOVL    P1(AP),R0                 ; Get the user buffer pointer
  51    04 AC    D0   00A0   326          MOVL    P2(AP),R1                 ; Get the user count
          2E    13   00A4   327          BEQL    10$                       ; Complete now if zero count
  00000000'GF    16   00A6   328          JSB     G^EXE$WRITECHK            ; Check the user buffer
```

```
                00000231'EF   16   00AC   329          JSB     ALLOC_SETUP                ; Allocate and set up a buffer
              32 A3    04 AC   B0   00B2   330          MOVW    P2(AP),IRP$W_BCNT(R3)      ; Store the byte count
                 52    2C B3   D0   00B7   331          MOVL    @IRP$L_SVAPTE(R3),R2       ; Get the system buffer pointer
                       38      BB   00BB   332          PUSHR   #^M<R3,R4,R5>              ; MOVC uses these
        62    00 BC    04 AC   28   00BD   333          MOVC    P2(AP),@P1(AP),(R2)                ; Move the buffer
                       38      BA   00C3   334          POPR    #^M<R3,R4,R5>              ; Restore pointers
                00000000'GF   17   00C5   335          JMP     G^EXE$QIODRVPKT            ; Post the packet to the driver
```

```
                        00CB    337 ;
                        00CB    338 ; Format the users list of records into a bisync block
                        00CB    339 ;
                        00CB    340 XMT_PTPBSC:
          50    6C    D0  00CB    341          MOVL    P1(AP),R0               ; Get user buffer pointer
        51    04 AC    D0  00CE    342          MOVL    P2(AP),R1               ; Get user buffer count
                  0D    12  00D2    343          BNEQ    20$                     ; BR if not zero count
     50    0000'8F    3C  00D4    344 10$:     MOVZWL  #SS$_NORMAL,R0          ; Get normal completion status
                  51    D4  00D9    345          CLRL    R1                      ; No device dependant status
      00000000'GF    17  00DB    346          JMP     G^EXE$FINISHIOC         ; Complete the IO request
      00000000'GF    16  00E1    347 20$:     JSB     G^EXE$WRITECHK          ; Check the user buffer descripter
                        00E7    348 ;
                        00E7    349 ; Check the buffer chain
                        00E7    350 ;
     59    50    51    C1  00E7    351          ADDL3   R1,R0,R9                ; Point to the end of the user buffer
                  52    D4  00EB    352          CLRL    R2                      ; Set zero record count
                  59    02  00ED    353          SUBL2   #2,R9                   ; Point before last count
                  58    80  00F0    354 30$:     MOVZWL  (R0)+,R8                ; Get the the count
     58    FFFF    8F    B1  00F3    355          CMPW    #-1,R8                  ; Last count ?
                  13    13  00F8    356          BEQL    50$                     ; yes - branch out
                  52    D6  00FA    357          INCL    R2                      ; Step record count
FFEE 50    58    59    F1  00FC    358          ACBL    R9,R8,R0,30$            ; BR if next count is in buffer
     51    0040    8F    3C  0102    359          MOVZWL  #XW$M_BADCHAIN,R1       ; Get bad buffer status
      00000000'GF    17  0107    360 40$:     JMP     G^EXE$ABORTIO           ; Abort the operation
                        010D    361
                  01    E0  010D    362 50$:     BBS     #XW$V_CHA_XPR,-         ; Is transparency called for
           03 44 A5          010F    363                  UCB$L_DEVDEPEND(R5),XPRTXT
               00A9    31  0112    364          BRW     NXPRTXT                 ; Go to nontransparent code
                        0115    365          .DSABL  LSB
                        0115    366 ; Build a transparent text block
                        0115    367 ;
                        0115    368 XPRTXT:
        51    04 AC    02    C5  0115    369          MULL3   #2,P2(AP),R1            ; Get the count
                  52    03    C4  011A    370          MULL2   #3,R2                   ; Each record needs 7 extra bytes
                  51    52    C0  011D    371          ADDL    R2,R1                   ; Get the total size
                     010E    30  0120    372          BSBW    ALLOC_SETUP             ; Get a system buffer
                        0123    373 ;       R9 -From data pointer
                        0123    374 ;       R10 -To data pointer
                        0123    375 ;       R11 -Record count
                  0138    8F    BB  0123    376          PUSHR   #^M<R3,R4,R5,R8>
                  58    20 A3    3C  0127    377          MOVZWL  IRP$W_FUNC(R3),R8       ; Save the function
                        7E    D4  012B    378          CLRL    -(SP)                   ; No starting CRC
                  5A    2C B3    D0  012D    379          MOVL    @IRP$L_SVAPTE(R3),R10   ; Set the to pointer
                  59    6C    D0  0131    380          MOVL    P1(AP),R9               ; Set the from pointer
                  5B    89    3C  0134    381          MOVZWL  (R9)+,R11               ; Get the first record,s count
               8A    0210    8F    B0  0137    382 10$:     MOVW    #EBCDLE!<EBCSTX*256>,(R10)+ ; All records start with DLE STX
 69    5B    6E     00000CAC'EF    0B  013C    383          CRC     POLY_TABLE,(SP),R11,(R9) ; Get the CRC on the Record
                  6E    50    B0  0145    384          MOVW    R0,(SP)                 ; Save the CRC until end of record
                  69    5B    10  3A  0148    385 20$:     LOCC    #EBCDLE,R11,(R9)        ; Find any DLE in the data
                        07    13  014C    386          BEQL    30$                     ; BR if end of record and no CRC
                        51    D6  014E    387          INCL    R1                      ; Include the DLE in the "found" data
                     02 50    F5  0150    388          SOBGTR  R0,30$                  ;  ...
                        50    D7  0153    389          DECL    R0                      ; Flag a DLE ended the record
                  5B    50    D0  0155    390 30$:     MOVL    R0,R11                  ; Set remaining data count
               50    51    59    C3  0158    391          SUBL3   R9,R1,R0                ; Get scaned data count
               6A    69    50    28  015C    392          MOVC    R0,(R9),(R10)           ; And move the data
                  59    51    D0  0160    393          MOVL    R1,R9                   ; Set new from pointer
```

```
                    5A   53   D0  0163   394          MOVL    R3,R10                  ; Set new to pointer
                    8A   10   90  0166   395          MOVB    #EBCDLE,(R10)+          ; Stuff the DLE or end with DLE
                         5B   D5  0169   396          TSTL    R11                     ; Is this the end of block
                         DB   14  016B   397          BGTR    20$                     ; NO -find more DLE,s
                         03   13  016D   398          BEQL    40$                     ; If eql, was the end of the block
                    8A   10   90  016F   399          MOVB    #EBCDLE,(R10)+          ; Else block ended with DLE, so stuff
                                   0172   400                                          ; an extra DLE in for control character
                    5B   89   B0  0172   401 40$:     MOVW    (R9)+,R11               ; Get the next record count
                5B  FFFF 8F   B1  0175   402          CMPW    #-1,R11                 ; Is it the sentenal
                         11   13  017A   403          BEQL    50$                     ; BR if it is
                    6A   1F   90  017C   404          MOVB    #EBCITB,(R10)           ; Store a record seperator
                         2D   10  017F   405          BSB     70$                     ; Add the trcord end char. to CRC
                8A  3232 8F   B0  0181   406          MOVW    #EBCSYN!<EBCSYN*256>,(R10)+ ; Followed by SYN SYN sin
                6E  018C'8F   3C  0186   407          MOVZWL  #DLE_STX_CRC,(SP)       ; Assume the next partial CRC
                         AA   11  018B   408          BRB     10$                     ; Get another record
                    6A   26   90  018D   409 50$:     MOVB    #EBCETB,(R10)           ; Assume not last block
                         0A   E1  0190   410          BBC     #IO$V_LASTBLOCK,-       ; Is it the last block
                           03 58        0192   411                                  R8,60$
                    6A   03   90  0194   412          MOVB    #EBCETX,(R10)           ; Set end of last block
                         15   10  0197   413 60$:     BSB     70$                     ; Include the block end chsr. in CRC
                    5E   04   C0  0199   414          ADDL    #4,SP                   ; Remove the partial CRC
                    0138 8F   BA  019C   415          POPR    #^M<R3,R4,R5,R8>
              5A    2C   B3   C2  01A0   416          SUBL2   @IRP$L_SVAPTE(R3),R10   ; Get the block count
            32 A3   5A   B0   01A4   417          MOVW    R10,IRP$W_BCNT(R3)      ; Save the byte count
         00000000'GF   17   01A8   418          JMP     G^EXE$QIODRVPKT         ; Post the IRP to the driver
                                   01AE   419
6A  01  04 AE  00000CAC'EF   0B   01AE   420 70$:     CRC     POLY_TABLE,4(SP),#1,(R10) ; include the last chara. in the CRC
                         5A   D6   01B8   421          INCL    R10                     ; Step by the last byte
                    8A   50   B0   01BA   422          MOVW    R0,(R10)+               ; And store the last byte
                         05   01BD   423          RSB
                              01BE   424 ;
                              01BE   425 ; Nontransparent record blocking and CRC generation
                              01BE   426 ;
                              01BE   427 NXPRTXT:
                    52   03   C4   01BE   428          MULL2   #3,R2                   ; Need count space +3 byte for each rec
                    52   04   C0   01C1   429          ADDL    #4,R2                   ; Room for STX and luck
51  52   04 AC   A1   01C4   430          ADDW3   P2(AP),R2,R1            ; get eventual size
              0065 30   01C9   431          BSBW    ALLOC_SETUP             ; allocate and set up a system buffer
                    0078 8F   BB   01CC   432          PUSHR   #^M<R3,R4,R5,R6>                ; save the FDT call reg. setup
                56   20 A3   3C   01D0   433          MOVZWL  IRP$W_FUNC(R3),R6       ; Get the IO function & mod.
                53   2C B3   D0   01D4   434          MOVL    @IRP$L_SVAPTE(R3),R3    ; Get the system buffer pointer
                    83   02   90   01D8   435          MOVB    #EBCSTX,(R3)+           ; Start the block with a stx
                    5B   6C   D0   01DB   436          MOVL    P1(AP),R11              ; Get the user buffer pointer
                    52   8B   3C   01DE   437 10$:     MOVZWL  (R11)+,R2               ; And the first record count
            52  FFFF 8F   B1   01E1   438          CMPW    #-1,R2                  ; Is it the sentenal
                         33   13   01E6   439          BEQL    40$                     ; BR if it is
                    59   52   7D   01E8   440          MOVQ    R2,R9                   ; Save the start of rec. ptr. &cnt.
            63  6B   52   28   01EB   441          MOVC    R2,(R11),(R3)           ; Move the record into the sys. buf.
                              01EF   442          ; R1- start of next record, R3- end of this record
                    5B   51   D0   01EF   443          MOVL    R1,R11                  ; Save a pointer to next record
                         59   D6   01F2   444          INCL    R9                      ; Include the record end character
                              01F4   445                                          ;  (not yet added) in the count
                    63   1F   90   01F4   446          MOVB    #EBCITB,(R3)            ; Add an ITB to the block
            6B  FFFF 8F   B1   01F7   447          CMPW    #-1,(R11)               ; Is this to be end of block
                         0A   12   01FC   448          BNEQ    30$                     ; BR- if it is ITB CRC CRC
                    63   26   90   01FE   449          MOVB    #EBCETB,(R3)            ; Assume intermediate block
                         0A   E1   0201   450          BBC     #IO$V_LASTBLOCK,-       ; BR if not last block
```

```
              03 56      0203   451              R6,30$
           63    03  90  0205   452      MOVB    #EBCETX,(R3)          ; Add an ETX to the block
6A  59  00  00000CAC'EF  0B  0208   453 30$: CRC  POLY_TABLE,#0,R9,(R10) ; Get the crc for the whole record
                         0211   454                                    ;  after the CRC instruction r0=CRC
                         0211   455                                    ;  and R3 points after the last byte
           83    50  B0  0211   456      MOVW    R0,(R3)+              ; Store the CRC in the block
       83  3232  8F  B0  0214   457      MOVW    #EBCSYN!<EBCSYN*256>,(R3)+ ; Start each record with syn-syn
                 C3  11  0219   458      BRB     10$                   ; Do next record
       51  53    02  C3  021B   459 40$: SUBL3   #2,R3,R1              ; Point after last CRC (before syn)
           0078  8F  BA  021F   460      POPR    #^M<R3,R4,R5,R6>
        51  2C   B3  C2  0223   461      SUBL2   @IRP$L_SVAPTE(R3),R1  ; Calculate the count
        32 A3    51  B0  0227   462      MOVW    R1,IRP$W_BCNT(R3)     ; And save it in the IRP
        00000000'GF  17  022B   463      JMP     G^EXE$QIODRVPKT       ; Queue the packet to the driver
                         0231   464
```

```
                     0231   466 ;
                     0231   467 ; Allocate and set up a system buffer utility
                     0231   468 ;
                     0231   469 ALLOC_SETUP:
          53   DD    0231   470          PUSHL   R3                          ; Save the IRP pointer
      51   0C   C0    0233   471          ADDL    #12,R1                      ; include overhead size
00000000'GF   16    0236   472          JSB     G^EXE$BUFQUOPRC             ; Does the user have quota
        06   50   E9 023C   473          BLBC    R0,10$                      ; BR if not
00000000'GF   16    023F   474          JSB     G^EXE$ALLOCBUF              ; Get the buffer
          53 8ED0   0245   475 10$:      POPL    R3                          ; Get back the IRP pointer
        1E   50   E9 0248   476          BLBC    R0,20$                     ;BR if allocation failure
   2C A3     52   D0 024B   477          MOVL    R2,IRP$L_SVAPTE(R3)        ;Save the new buffer pointer
   30 A3     51   B0 024F   478          MOVW    R1,IRP$W_BOFF(R3)          ; Save the no. bytes allocated and charged
 50    0080 C4   D0 0253   479          MOVL    PCB$L_JIB(R4),R0           ; GET JIB ADDRESS
   20 A0     51   C2 0258   480          SUBL    R1,JIB$L_BYTCNT(R0)        ; Charge process for buffer
   82    0C A2   9E 025C   481          MOVAB   12(R2),(R2)+               ;Set up pointer to first byte of buffer
      62   6C   D0 0260   482          MOVL    P1(AP),(R2)                ; Set up pointer to start of user buffer
40 A3   04 AC   B0 0263   483          MOVW    P2(AP),IRP$W_ABCNT(R3)     ; Save the users transfer count
                05 0268   484          RSB
                     0269   485
      5E   04   C0 0269   486 20$:     ADDL    #4,SP                       ; Pop the return off the stack
00000000'GF   17    026C   487          JMP     G^EXE$ABORTIO               ; Abort the operation
```

```
                        0272   489 ;
                        0272   490 ; Diagnostic full duplex read/write FDT processing
                        0272   491 ;
                        0272   492 XMT_DIAG:
          50    08 AC   7D   0272   493          MOVQ    P3(AP),R0                       ; Check the read buffer
    00000000'GF   16   0276   494          JSB     G^EXE$READCHK                   ; ...
          50    6C   7D   027C   495          MOVQ    P1(AP),R0                       ; Check the write buffer
    00000000'GF   16   027F   496          JSB     G^EXE$WRITECHK                  ; ...
51  0C AC   04 AC   C1   0285   497          ADDL3   P2(AP),P4(AP),R1                ; Get the total buffer count
          A3 AF   16   028B   498          JSB     ALLOC_SETUP                     ; Allocate a system buffer
          62    08 AC   D0   028E   499          MOVL    P3(AP),(R2)                     ; Set receive buffer addr. for
                        0292   500                                                  ;  IO completion code
    32 A3   0C AC   3C   0292   501          MOVZWL  P4(AP),IRP$W_BCNT(R3)           ; Set the rec. count
50  0C AC   2C B3   C1   0297   502          ADDL3   @IRP$L_SVAPTE(R3),P4(AP),R0 ; Get the transmit pointer
                38   BB   029D   503          PUSHR   #^M<R3,R4,R5>
60  00 BC   04 AC   28   029F   504          MOVC    P2(AP),@P1(AP),(R0)             ; Move the transmit data
                38   BA   02A5   505          POPR    #^M<R3,R4,R5>
    00000000'GF   17   02A7   506          JMP     G^EXE$QIODRVPKT                 ; Queue the packet to the driver
```

```
                        02AD    508          .SBTTL  XW/DUP-11 Read FDT Routines
                        02AD    509 ;+
                        02AD    510 ; XW_READ - READ FUNCTION PROCESSING
                        02AD    511 ;
                        02AD    512 ; THIS ROUTINE IS CALLED FROM THE FUNCTION DECISION TABLE DISPATCHER TO PROCESS
                        02AD    513 ; A READ LOGICAL OR READ PHYSICAL FUNCTION TO A DUP11 DEVICE.
                        02AD    514 ;
                        02AD    515 ; INPUTS:
                        02AD    516 ;
                        02AD    517 ;       R0 = SCRATCH.
                        02AD    518 ;       R1 = SCRATCH.
                        02AD    519 ;       R2 = SCRATCH.
                        02AD    520 ;       R3 = ADDRESS OF I/O REQUEST PACKET.
                        02AD    521 ;       R4 = CURRENT PROCESS PCB ADDRESS.
                        02AD    522 ;       R5 = ASSIGNED DEVICE UCB ADDRESS.
                        02AD    523 ;       R6 = ADDRESS OF CCB.
                        02AD    524 ;       R7 = I/O FUNCTION CODE.
                        02AD    525 ;       R8 = FUNCTION DECISION TABLE DISPATCH ADDRESS.
                        02AD    526 ;       R9 = SCRATCH.
                        02AD    527 ;       R10 = SCRATCH.
                        02AD    528 ;       R11 = SCRATCH.
                        02AD    529 ;       AP = ADDRESS OF FIRST FUNCTION DEPENDENT PARAMETER.
                        02AD    530 ;
                        02AD    531 ; OUTPUTS:
                        02AD    532 ;
                        02AD    533 ;       THE FUNCTION PARAMETERS ARE CHECKED AND A BUFFER IS ALLOCATED FOR THE
                        02AD    534 ;       DUP-11 DRIVER TO  RECEIVE A MESSAGE INTO.
                        02AD    535 ;-
                        02AD    536
                        02AD    537 XW_READ:
    03    0D    EF      02AD    538          EXTZV   #XW$V_IOMOD,#XW$S_IOMOD,- ; Extract the IO modifier field
 50    20 A3            02B0    539                  IRP$W_FUNC(R3),R0
                        02B3    540          CASE    R0,-                      ; Dispatch on the modifier
                        02B3    541                  <REC_BINARY,-             ; receive until count runout
                        02B3    542                  REC_PTPBSC,-              ; Add point to point BSC control
                        02B3    543                  >                         ; Last modifier
       FDD4    31       02BB    544          BRW     BAD_MODIFIER              ; Out of range function
                        02BE    545 ;
                        02BE    546 ; Receive until byte count run out- no line protocol or data processing
                        02BE    547 ;
                        02BE    548          .ENABL  LSB
                        02BE    549 REC_BINARY:
                        02BE    550 ;
                        02BE    551 ; Receive data and manage point-to-point BSC line protocol
                        02BE    552 ;
                        02BE    553 REC_PTPBSC:
    50    6C    D0      02BE    554          MOVL    P1(AP),R0                 ; Get the user pointer
 51    04 AC    D0      02C1    555          MOVL    P2(AP),R1                 ; Get the users count
          0D    12      02C5    556          BNEQ    20$                       ; Complete the request now if zero
          51    D4      02C7    557 10$:     CLRL    R1                        ; No device dependant status
50    0000'8F    3C      02C9    558          MOVZWL  #SS$_NORMAL,R0            ; Get normal completion status
 00000000'GF    17      02CE    559          JMP     G^EXE$FINISHIOC           ; Return the packet
 00000000'GF    16      02D4    560 20$:     JSB     G^EXE$READCHK             ; Check the user buffer
    03    0D    ED      02DA    561          CMPZV   #XW$V_IOMOD,#XW$S_IOMOD,- ; Is this a binary operation
 00    20 A3            02DD    562                  IRP$W_FUNC(R3),#IO$K_SRRUNOUT
          07    13      02E0    563          BEQL    30$                       ; BR if it is
 51    42 A5    3C      02E2    564          MOVZWL  UCB$W_DEVBUFSIZ(R5),R1    ; Get the block size
```

```
                51    20   C0  02E6   565           ADL     #8*4,R1                         ; Room for ITB pointers
                  FF45   30  02E9   566 30$:        BSB     ALLOC_SETUP                     ; Get a system buffer and set it up
          32 A3   51    0C   A3  02EC   567           SUBW3   #12,RT,IRP$W_BCNT(R3)           ; Save the transfer count
              00000000'GF   17  02F1   568 40$:        JMP     G^EXE$QIODRVPKT                 ; Post the packet to the driver
```

```
                     02F7    570          .SBTTL  Start IO on the DUP-11
                     02F7    571 ;+
                     02F7    572 ; XW_STARTIO - START I/O OPERATION ON THE DUP-11
                     02F7    573 ;
                     02F7    574 ; THIS ROUTINE IS ENTERED WHEN THE ASSOCIATED UNIT IS IDLE AND A PACKET IS
                     02F7    575 ; AVAILABLE FOR PROCESSING.
                     02F7    576 ;
                     02F7    577 ; INPUTS:
                     02F7    578 ;
                     02F7    579 ;       R3 = ADDRESS OF I/O REQUEST PACKET.
                     02F7    580 ;       R5 = ADDRESS OF DEVICE UNIT UCB.
                     02F7    581 ;
                     02F7    582 ; OUTPUTS:
                     02F7    583 ;
                     02F7    584 ; LOTSATEXT
                     02F7    585 ;
                     02F7    586 ;-
                     02F7    587
                     02F7    588 XW_STARTIO:
                     02F7    589          .ENABL  LSB
54   24 A5    D0     02F7    590          MOVL    UCB$L_CRB(R5),R4        ; Get the DUP csr pointer
54   2C B4    D0     02FB    591          MOVL    @CRB$L_INTD+VEC$L_IDB(R4),R4 ; ...
          00   ED     02FF    592          CMPZV   #IRP$V_FCODE,-          ;Is this a set function
20 A3     06          0301    593                  #IRP$S_FCODE,IRP$W_FUNC(R3),-
               23     0304    594                  #IO$_SETMODE
               4D  12 0305    595          BNEQ    50$                     ; BR if not set
                     0307    596 ;
                     0307    597 ; Set mode
                     0307    598 ;
42 A5    3A A3    B0  0307    599          MOVW    IRP$L_MEDIA+2(R3),UCB$W_DEVBUFSIZ(R5) ; Set block size
44 A5    3C A3    D0  030C    600          MOVL    IRP$L_MEDIA+4(R3),UCB$L_DEVDEPEND(R5) ; Save characteristics
               07  E1  0311    601          BBC     #IO$V_SHUTDOWN,-        ; Is dorp line requested
          08 20 A3          0313    602                  IRP$W_FUNC(R3),10$
04 A4   0100 8F    A8  0316    603          BISW    #DUP_CSR_M_RES,DUP_CSR_TXCSR(R4) ;Reset the DUP
               33  11  031C    604          BRB     40$                     ; Complete the request
                     031E    605
               06  E1  031E    606 10$:     BBC     #IO$V_STARTUP,-         ; Is enable line requested
          2E 20 A3          0320    607                  IRP$W_FUNC(R3),40$
            64    02  A8  0323    608          BISW    #DUP_CSR_M_DTR,(R4)     ; Assert DTR
 64   0200 8F    B3  0326    609 20$:     BITW    #DUP_CSR_M_DSR,(R4)     ; Is DSR up ?
               24  12  032B    610          BNEQ    40$                     ; BR if it is
               08  E0  032D    611          BBS     #IO$V_NODSRWAIT,-       ; Is a non wait diagnostic function here
          1F 20 A3          032F    612                  IRP$W_FUNC(R3),40$
             0097   30  0332    613          BSBW    110$                    ; Send a line off message
                     0335    614          DSBINT
                     033B    615          WFIKPCH 30$,#2
                     0345    616 30$:     SETIPL  UCB$B_FIPL(R5)
D8 64 A5     03     E1  0349    617          BBC     #UCB$V_CANCEL,UCB$W_STS(R5),20$ ; Cancel requested ?
             0465   31  034E    618          BRW     CANCEL_CUR              ; Return this packet
             0439   31  0351    619 40$:     BRW     COMP_GOOD               ; Complete the request
                     0354    620
     06      00     ED  0354    621 50$:     CMPZV   #IRP$V_FCODE,#IRP$S_FCODE,-     ; Is it a sense mode ?
   27   20 A3          0357    622                  IRP$W_FUNC(R3),#IO$_SENSEMODE
               5C  13  035A    623          BEQL    90$                             ; BR if it is
                     035C    624 ;
                     035C    625 ; Send/receive request
                     035C    626 ;
```

```
                68 A5    05    CA  035C    627          BICL    #UCB$M_XW_NAK!UCB$M_XW_REP,UCB$W_DEVSTS(R5) ; Reset status bits
          0080 C5    0081 C5    90  0360    628          MOVB    UCB$B_ERTMAX(R5),UCB$B_ERTCNT(R5) ; Init the retry count
                                    0367    629 SND_REC_REQ:
             03 64 A5    03    E1  0367    630          BBC     #UCB$V_CANCEL,UCB$W_STS(R5),60$ ; BR if not canceled
                      0447    31  036C    631          BRW     CANCEL_CUR                      ; Cancel this IRP
           009A C5    20 A3    B0  036F    632 60$:     MOVW    IRP$W_FUNC(R3),UCB$W_FUNC(R5) ; Save the function code for ELG.
             78 A5    2C B3    D0  0375    633          MOVL    @IRP$L_SVAPTE(R3),UCB$L_SVAPTE(R5) ; copy buffer desc. to UCB
             7E A5    32 A3    B0  037A    634          MOVW    IRP$W_BCNT(R3),UCB$W_BCNT(R5) ; ...
              64    0200 8F    B3  037F    635          BITW    #DUP_CSR_M_DSR,(R4)     ; Is the line up
                         0B    12  0384    636          BNEQ    70$                     ; BR if it is
                         08    E0  0386    637          BBS     #IO$V_NODSRWAIT,-       ; Is it a diagnostic bypass????????
                   06 20 A3          0388    638                  IRP$W_FUNC(R3),70$
                                     038B    639 ;
                                     038B    640 ; The line is not connected (no DSR) abort the request
                                     038B    641 ;
                                     038B    642 ABORT_NO_DSR:
                      51    04    3C  038B    643          MOVZWL  #XW$M_NODSR,R1          ; Get no DSR status
                         042E    31  038E    644          BRW     ABORT_CUR               ; Conplete this IRP with abort status
                                     0391    645
                      03    0D    EF  0391    646 70$:     EXTZV   #XW$V_IOMOD,#XW$S_IOMOD,- ; Extract the IO modifier field
                50    20 A3          0394    647                  IRP$W_FUNC(R3),R0
             08 2A A3    01    E1  0397    648          BBC     #IRP$V_FUNC,IRP$W_STS(R3),80$ ; BR if a write
                                     039C    649          CASE    R0,-                    ; READ dispatch on the modifier
                                     039C    650                  <REC_INI_BINARY,-       ; Send until count runout
                                     039C    651                  REC_INI_PTPBSC,-        ; Add point to point BSC control
                                     039C    652                  >                       ; Last modifier
                                     03A4    653 80$:     CASE    R0,-                    ; WRITE dispatch on the modifier
                                     03A4    654                  <XMT_INI_BINARY,-               ; Send until count runout
                                     03A4    655                  XMT_INI_PTPBSC,-                ; Add point to point BSC control
                                     03A4    656                  ABORT_CUR,-
                                     03A4    657                  ABORT_CUR,-
                                     03A4    658                  ABORT_CUR,-
                                     03A4    659                  ABORT_CUR,-
                                     03A4    660                  ABORT_CUR,-
                                     03A4    661                  XMT_INI_DIAG,-                  ; Start a diagnostic QIO
                                     03A4    662                  >                       ; Last modifier
                                     03B8    663 ;
                                     03B8    664 ; Sense mode
                                     03B8    665 ;
                   51    44 A5    D0  03B8    666 90$:     MOVL    UCB$L_DEVDEPEND(R5),R1  ; Get the device char.
                      51    04    CA  03BC    667          BICL    #XW$M_CHA_DSR,R1        ; Assume no DSR
                64    0200 8F    B3  03BF    668          BITW    #DUP_CSR_M_DSR,(R4)     ; Is DSR up ?
                         03    13  03C4    669          BEQL    100$                    ; BR if not
                      51    04    C8  03C6    670          BISL    #XW$M_CHA_DSR,R1        ; Mark DSR active
                         03C3    31  03C9    671 100$:    BRW     COMP_GOOD_AB            ; Return the IRP with status
                                     03CC    672
                                     03CC    673 ;
                                     03CC    674 ; Send a message to the operator
                                     03CC    675 ;
          001B 00A0 C5    01    0F  9D  03CC    676 110$:    ACBB    #15,#1,UCB$L_XW_SYCNT(R5),130$ ; Time for a message
                         00A0 C5    94  03D4    677          CLRB    UCB$L_XW_SYCNT(R5)      ; Reset the message counter
                      50    05    9A  03D8    678          MOVZBL  #MSG$_DEVOFFLIN,R0      ; Get a device off line message
                         18          BB  03DB    679 120$:    PUSHR   #^M<R3,R4>
                      54    50    D0  03DD    680          MOVL    R0,R4                   ; Set up the message no.
                53    00000000'GF    9E  03E0    681          MOVAB   G^SYS$GL_OPRMBX,R3      ; Get the operators box no.
                      00000000'GF    16  03E7    682          JSB     G^EXE$SNDEVMSG          ; And send it
                         18          BA  03ED    683          POPR    #^M<R3,R4>
```

05 03EF 684 130$: RSB

```
                     03F0   686          .DSABL  LSB
                     03F0   687 ;
                     03F0   688 ;Receive until byte count runout with no protocol or processing set up code
                     03F0   689 ;
                     03F0   690 REC_INI_BINARY:
     68 A5    08     AA   03F0   691          BICW    #UCB$M_XW_PIPE,UCB$W_DEVSTS(R5) ; Reset any abort pipeline mark
  00AA C5    01     90   03F4   692          MOVB    #RST$BINARY,UCB$B_XW_RISTA(R5) ; Set interrupt processor state
05 20 A3    07     E1   03F9   693          BBC     #IO$V_SLAVLOOP,IRP$W_FUNC(R3),10$ ; Is this a diagnostic loop
  00AA C5    0D     90   03FE   694          MOVB    #RST$LOOP_TEST,UCB$B_XW_RISTA(R5) ; Set loop state
          04EE     30   0403   695 10$:      BSBW    RCV_START                       ; Start the receiver
                        0406   696 REC_BIN_WAIT:
                        0406   697          WFIKPCH REC_BIN_TMO,#2                  ; Wait for the receive to complete
                        0410   698          IOFORK                                  ; Create a fork process
        2C B3    C3     0416   699          SUBL3   @IRP$L_SVAPTE(R3),-             ; Get the actual transfer size
     50    78 A5        0419   700                  UCB$L_SVAPTE(R5),R0
     32 A3    50     B0   041C   701          MOVW    R0,IRP$W_BCNT(R3)               ; and set it for the iocompletion code
          036A     31   0420   702          BRW     COMP_GOOD                       ; Return the packet to the user
```

```
                                                J 6
                     0423    704 ;
                     0423    705 ; Receive a buffer then check its' CRC and do the proper PTP BSC line protocol
                     0423    706 ;
                     0423    707         .ENABL  LSB
                     0423    708 REC_INI_PTPBSC:
            03   E1  0423    709         BBC     #UCB$V_XW_PIPE,-        ; is the cancel pipe active
        06 68 A5     0425    710                 UCB$W_DEVSTS(R5),5$
                     0428    711 PIPE_MARK:
          51   20 3C 0428    712         MOVZWL  #XW$M_PIPE_MARK,R1      ; Get the pipe mark error
               0391 31 042B    713         BRW     ABORT_CUR               ; And abort the current IRP
        50   7E A5 3C 042E    714 5$:     MOVZWL  UCB$W_BCNT(R5),R0       ; Get the rec buffer size
          50   20 A2 0432    715         SUBW    #8+4,R0                 ; Remove the CRC pointers from count
    51    50   78 A5 C1 0435    716         ADDL3   UCB$L_SVAPTE(R5),R0,R1  ; Point to the first CRC pointer
     00A4 C5   51 D0 043A    717         MOVL    R1,UCB$L_XW_ITBPTR(R5)  ; Set up the ITB pointer pointer
     00A8 C5   07 90 043F    718         MOVB    #7,UCB$B_XW_ITBCNT(R5) ; Init. the ITB count
               50 D7 0444    719         DECL    R0                      ; Save a byte for a count
        7E A5  50 B0 0446    720         MOVW    R0,UCB$W_BCNT(R5)       ; Set the receive buffer count
           78 A5 D6 044A    721         INCL    UCB$L_SVAPTE(R5)        ; Save the count byte
              81 7C 044D    722         CLRQ    (R1)+                   ; Init. the ITB pointer area
              81 7C 044F    723         CLRQ    (R1)+                   ; ...
              81 7C 0451    724         CLRQ    (R1)+                   ; ...
              81 7C 0453    725         CLRQ    (R1)+                   ; ...
     00AA C5   02 90 0455    726         MOVB    #RST$FIRST,UCB$B_XW_RISTA(R5) ; Set the starting int. state
               0497 30 045A    727         BSBW    RCV_START               ; Start the device
                     045D    728 REC_TEXT_WAIT:
                     045D    729         WFIKPCH REC_TEXT_TMO,#4         ; Wait for the buffer to complete
                     0467    730         IOFORK                          ; Create a fork process
        51   2C B3 D0 046D    731         MOVL    @IRP$L_SVAPTE(R3),R1    ; Get the start of the buffer
              51 D6 0471    732         INCL    R1                      ; Skip the hidden byte
    50   78 A5   51 C3 0473    733         SUBL3   R1,UCB$L_SVAPTE(R5),R0  ; Get the no. bytes received
           7E A5 B5 0478    734         TSTW    UCB$W_BCNT(R5)          ; Did the byte count run out
              09 14 047B    735         BGTR    10$                     ; BR if not
                     047D    736         PROTLO  REC_BKTOLONG
              FEE1 31 0483    737         BRW     SND_REC_REQ             ; Reassign the buffer
                     0486    738
              50 D7 0486    739 10$:    DECL    R0                      ; Redice the buff. count by one
        52    81 90 0488    740         MOVB    (R1)+,R2                ; Get the first char. of the block
        52    2D 91 048B    741         CMPB    #EBCENQ,R2              ; Is the last response requested
              1C 12 048E    742         BNEQ    20$                     ; BR if not
                     0490    743         PROTLO  REC_REP
              00BC 30 0496    744         BSBW    SEND_RESP               ; Set up the last response
                     0499    745         WFIKPCH REC_REP_TMO,#4
                     04A3    746         IOFORK                          ; Create a fork process
              FEBB 31 04A9    747         BRW     SND_REC_REQ             ; Reassign the buffer
                     04AC    748
        52    37 91 04AC    749 20$:    CMPB    #EBCEOT,R2              ; Is it an EOT
              06 12 04AF    750         BNEQ    30$                     ; BR if not
        51    01 3C 04B1    751         MOVZWL  #XW$M_EOT,R1            ; Get an EOT status
              0308 31 04B4    752         BRW     ABORT_CUR               ; Abort the packet
                     04B7    753
        52    02 91 04B7    754 30$:    CMPB    #EBCSTX,R2              ; Is it a STX
              54 12 04BA    755         BNEQ    70$                     ; BR if not
  FF A140     2D 91 04BC    756         CMPB    #EBCENQ,-1(R1)[R0]      ; Is it a TTD or abort
              3A 12 04C1    757         BNEQ    50$                     ; BR if not
  FD A140     03 91 04C3    758         CMPB    #EBCETX,-3(R1)[R0]      ; Is the ENQ a CRC char. ?
              33 13 04C8    759         BEQL    50$                     ; BR if it is
  FD A140     26 91 04CA    760         CMPB    #EBCETB,-3(R1)[R0]      ; ...
```

```
                                                K 6
                2C   13  04CF   761          BEQL    50$                     ; BR if it is
                         04D1   762 ;
                         04D1   763 ; TTD or Forward Abort
                         04D1   764 ;
          51     3D   9A  04D1   765          MOVZBL  #EBCNAK,R1              ; Get a NAK
             03CD       30  04D4   766          BSBW    XMT_START_PRO1          ; And send it
                         04D7   767          WFIKPCH REC_FABO_TMO,#4
                         04E1   768          IOFORK                          ; Create a fork process
    50    2C B3   D0  04E7   769          MOVL    @IRP$L_SVAPTE(R3),R0    ; Get the old buffer pointer
                50   D6  04EB   770          INCL    R0                      ; Point to first received byte
60     2D02 8F   B1  04ED   771          CMPW    #EBCSTX!<EBCENQ*256>,(R0) ; Was it a TTD
                06   13  04F2   772          BEQL    40$                     ; Yes- only log aborts
                         04F4   773          PROTLO  REC_FORABORT
              FE6A   31  04FA   774 40$:     BRW     SND_REC_REQ             ; Reassign the buffer
            3C A3   D4  04FD   775 50$:     CLRL    IRP$L_MEDIA+4(R3)       ; Start with no status
FD A140     03   91  0500   776          CMPB    #EBCETX,-3(R1)[R0]      ; Is it an ETX block
                06   12  0505   777          BNEQ    60$                     ; BR if not
          2000 8F   3C  0507   778          MOVZWL  #XW$M_ETXEND,-          ; Set the completion status
            3C A3       050B   779                  IRP$L_MEDIA+4(R3)
             0404   31  050D   780 60$:     BRW     BLK_CHECK               ; Go do the CRC check
                         0510   781
    52    FF A1   B0  0510   782 70$:     MOVW    -1(R1),R2               ; Get the first two bytes from the buff.
                51   D6  0514   783          INCL    R1                      ; Step the buffer pointer
                50   D7  0516   784          DECL    R0                      ; And count
52     371U 8F   B1  0518   785          CMPW    #EBCDLE!<EBCEOT*256>,R2 ; Is it a disconnect
                0C   12  051D   786          BNEQ    80$                     ; Br in not
                         051F   787          PROTLO  REC_DISCON
          51    08   3C  0525   788          MOVZWL  #XW$M_DISCON,R1         ; Get a disconnect status
             0294   31  0528   789          BRW     ABORT_CUR               ; Return the packet to the user
                         052B   790
52     0210 8F   B1  052B   791 80$:     CMPW    #EBCDLE!<EBCSTX*256>,R2 ; Is it xpr text
                09   13  0530   792          BEQL    90$                     ; BR if it is
                         0532   793 85$:     PROTLO  REC_TXTFORMAT
              FE2C   31  0538   794          BRW     SND_REC_REQ             ; Reassign the buffer
                         053B   795
52     1000 8F   3C  053B   796 90$:     MOVZWL  #XW$M_XPR,R2            ; Get transparent status
FD A140     03   91  0540   797          CMPB    #EBCETX,-3(R1)[R0]      ; Is it the last block
                07   12  0545   798          BNEQ    100$                    ; Br if not
52    00002000 8F   C8  0547   799          BISL    #XW$M_ETXEND,R2         ; Save the status for the completion
  3C A3    52   D0  054E   800 100$:    MOVL    R2,IRP$L_MEDIA+4(R3)    ; ...
             03BF   31  0552   801          BRW     BLK_CHECK               ; Check transparent text block
                         0555   802          .DSABL  LSB
                         0555   803 ;
                         0555   804 ; Set up response utility
                         0555   805 ;
                         0555   806 SEND_RESP:
                02   E1  0555   807          BBC     #UCB$V_XW_NAK,-         ; Is a NAK called for
          06 68 A5       0557   808                  UCB$W_DEVSTS(R5),10$
            51   3D   9A  055A   809          MOVZBL  #EBCNAK,R1              ; Get a NAK message
             0344   31  055D   810          BRW     XMT_START_PRO1          ; Start the device and return to caller
                01   E1  0560   811 10$:     BBC     #UCB$V_XW_AK1,-         ; BR if an AK1 is needed
          07 68 A5       0562   812                  UCB$W_DEVSTS(R5),20$
            51   7010 8F   3C  0565   813          MOVZWL  #EBCDLE!<EBCAK0*256>,R1 ; Get an AK0
                05   11  056A   814          BRB     30$                     ; And send it
            51   6110 8F   3C  056C   815 20$:     MOVZWL  #EBCDLE!<EBCAK1*256>,R1 ; Get an AK1
             0336   31  0571   816 30$:     BRW     XMT_START_PRO2          ; Start the device
```

```
                     0574   818 ;
                     0574   819 ; Timout during protocol response processing
                     0574   820 ;
                     0574   821          .ENABL  LSB
                     0574   822 REC_BIN_TMO:
                  0E   10    0574   823          BSB     10$                     ; Call common timout processing
                FE8D   31    0576   824          BRW     REC_BIN_WAIT            ; Cont. waiting
                     0579   825 REC_TEXT_TMO:
                  09   10    0579   826          BSB     10$                     ; Join common timout processing
                FEDF   31    057B   827          BRW     REC_TEXT_WAIT           ; Continue waiting
                     057E   828 REC_REP_TMO:
                     057E   829 REC_SNAK_TMO:
                     057E   830 REC_FABO_TMO:
                FDE6   31    057E   831          BRW     SND_REC_REQ             ; Try again
                     0581   832 REC_GACK_TMO:
                049C   31    0581   833          BRW     REC_DONE                ; Complete it
                     0584   834
08 64 A5          03   E1    0584   835 10$:     BBC     #UCB$V_CANCEL,UCB$W_STS(R5),20$ ; Is a cancel requested ?
                    28   10    0589   836          BSB     TMO_RESET               ; Reset the device
                  50 8ED0    058B   837          POPL    R0                      ; Remove the return
                  0225   31    058E   838          BRW     CANCEL_CUR              ; Cancel this IRP
                     0591   839
 64   0200 8F     B3    0591   840 20$:     BITW    #DUP_CSR_M_DSR,(R4)     ; Is the line connected ?
                    0E   12    0596   841          BNEQ    30$                     ; BR if it is
                    19   10    0598   842          BSB     TMO_RESET               ; Reset the device
                     059A   843          PROTLO  DSR_LOST
                  50 8ED0    05A0   844          POPL    R0                      ; Remove the return
                FDE5   31    05A3   845          BRW     ABORT_NO_DSR            ; Abort this IRP
                     05A6   846
          50     8E    D0    05A6   847 30$:     MOVL    (SP)+,R0                ; Get the return
                     05A9   848          DSBINT                          ; Disable interrupts
          7E     50    D0    05AF   849          MOVL    R0,-(SP)                ; Put the return back
                     05    05B2   850          RSB
                     05B3   851
                     05B3   852 TMO_RESET:
00AB C5     64     B0    05B3   853          MOVW    (R4),UCB$W_XW_LSTCSR(R5) ; Save the modem lines at end of trans.
          64     02    B0    05B8   854          MOVW    #DUP_CSR_M_DTR,(R4)     ; Turn off the receiver
            04   A4    B4    05BB   855          CLRW    DUP_CSR_TXCSR(R4)       ; Turn off the transmiter
                     05BE   856          SETIPL  UCB$B_FIPL(R5)          ; Drop to fork
00AA C5     00     90    05C2   857          MOVB    #RST$IDLE,UCB$B_XW_RISTA(R5) ; Set receiver to idle
00A9 C5     00     90    05C7   858          MOVB    #TSTA$IDLE,UCB$B_XW_TISTA(R5) ; Set transmiter to idle
                     05    05CC   859          RSB
                     05CD   860          .DSABL  LSB
```

```
                     05CD    862 ;
                     05CD    863 ; Send as binary adding only SYN and trailing PAD characters
                     05CD    864 ;
                     05CD    865 XMT_INI_BINARY:
68 A5    02    A8    05CD    866             BISW    #UCB$M_XW_AK1,UCB$W_DEVSTS(R5)
68 A5    08    AA    05D1    867             BICW    #UCB$M_XW_PIPE,UCB$W_DEVSTS(R5)
      02E1    30    05D5    868             BSBW    XMT_START                       ; Start up the transmitter
51   47 A5    9A    05D8    869             MOVZBL  UCB$L_DEVDEPEND+3(R5),R1 ; Get the time to wait in a long word
                     05DC    870             WFIKPCH 10$,RT                  ; Wait for interrupt or timeout
                     05E6    871             IOFORK                          ; Create a fork process
      019E    31    05EC    872             BRW     COMP_GOOD               ; Complete the request with good status
      FFC1    30    05EF    873 10$:        BSBW    TMO_RESET               ; Turn off the device
   51   10    3C    05F2    874             MOVZWL  #XW$M_TRABINTMO,R1      ; Get a timout sending binary error
      01C7    31    05F5    875             BRW     ABORT_CUR               ; And abort the request
                     05F8    876
```

```
                     05F8   878 ;
                     05F8   879 ; Send as point-to-point BSC
                     05F8   880 ;
                     05F8   881         .ENABL  LSB
                     05F8   882 XMT_INI_PTPBSC:
          03   E1    05F8   883         BBC     #UCB$V_XW_PIPE,-        ; Is the cancel pipe active
 03 68 A5              05FA   884                 UCB$W_DEVSTS(R5),5$
      FE28   31      05FD   885         BRW     PIPE_MARK
      02B6   30      0600   886 5$:     BSBW    XMT_START               ; Start up the transmitter
                     0603   887 XMT_TEXT_WAIT:  WFIKPCH 150$,#4              ; Wait for interrupt or timeout
                     060D   888         IOFORK                          ; Create a fork process
                     0613   889 GET_RESPONSE:
      02CF   30      0613   890         BSBW    RCV_START_PRO           ; Start a protocol response receive
                     0616   891         WFIKPCH 160$,#4                 ; Wait for a response
                     0620   892         IOFORK                          ; Create a fork process
   51 009C C5   D0    0626   893         MOVL    UCB$L_XW_SPBF(R5),R1    ; Get the response in R1
      51   10   91    062B   894         CMPB    #EBCDLE,R1              ; Is it a DLE response
           03   13    062E   895         BEQL    6$                      ; BR if DLE
      008F   31      0630   896         BRW     100$                    ; BR if not DLE
51 51   F8 8F   78    0633   897 6$:     ASHL    #-8,R1,R1               ; Set up the next byte
      51   37   91    0638   898         CMPB    #EBCEOT,R1              ; Is it a disconnect
           06   12    063B   899         BNEQ    10$                     ; BR if not
      51   08   3C    063D   900         MOVZWL  #XW$M_DISCON,R1         ; Get disconnect status
      017C   31      0640   901         BRW     ABORT_CUR               ; Complete the IRP
   51   7C 8F   91    0643   902 10$:    CMPB    #EBCRVI,R1              ; Was it an RVI
           08   12    0647   903         BNEQ    20$                     ; BR if not
   51 0400 8F   3C    0649   904         MOVZWL  #XW$M_RVI,R1            ; Get RVI status
      013E   31      064E   905         BRW     COMP_GOOD_AB            ; Complete the IRP
   51   6B 8F   91    0651   906 20$:    CMPB    #EBCWACK,R1             ; Is it a WACK
           24   12    0655   907         BNEQ    50$                     ; BR if not WACK
                     0657   908         DSBINT                          ; Lock out interrupts for the wait
                     065D   909         WFIKPCH 30$,#2                  ; Wait for 2 seconds
                     0667   910 30$:     SETIPL  UCB$B_FIPL(R5)          ; Drop IPL to fork
03 64 A5   03   E1    066B   911         BBC     #UCB$V_CANCEL,UCB$W_STS(R5),40$ ; BR if no user cancel
      0143   31      0670   912         BRW     CANCEL_CUR              ; Return the packet to the user
   51   2D   9A    0673   913 40$:     MOVZBL  #EBCENQ,R1              ; Get an ENQ
      022B   30      0676   914         BSBW    XMT_START_PRO1          ; And send it
        88   11    0679   915         BRB     XMT_TEXT_WAIT           ; Wait for a response
                     067B   916 ;
                     067B   917 ; Check ACKs
                     067B   918 ;
12 68 A5   01   E1    067B   919 50$:     BBC     #UCB$V_XW_AK1,UCB$W_DEVSTS(R5),60$ ; Which ACK is needed
   51   61 8F   91    0680   920         CMPB    #EBCAK1,R1              ; Is it an ACK 1
           03   12    0684   921         BNEQ    52$                     ; BR if not
      00F6   31      0686   922         BRW     COMP_GOOD_TGL           ; Complete the IRP and toggle ACKs
   51   70 8F   91    0689   923 52$:     CMPB    #EBCAK0,R1              ; Is it the other ACK
           15   13    068D   924         BEQL    80$                     ; BR if it is
      0094   31      068F   925         BRW     170$                    ; NO- consider it garbled
   51   70 8F   91    0692   926 60$:     CMPB    #EBCAK0,R1              ; Is it an ACK 0
           03   12    0696   927         BNEQ    62$                     ; BR if not
      00E4   31      0698   928         BRW     COMP_GOOD_TGL           ; Complete the IRP and toggle the ACKs
   51   61 8F   91    069B   929 62$:     CMPB    #EBCAK1,R1              ; Is it the other ACK
           03   13    069F   930         BEQL    80$                     ; BR if it is
      0082   31      06A1   931         BRW     170$                    ; No- consider it garbled
                     06A4   932 ;
                     06A4   933 ; Out of phase ACK
                     06A4   934 ;
```

```
                00    E4  06A4   935 80$:    BBSC    #UCB$V_XW_REP,-          ; Are we in a rep state
          09 68 A5        06A6   936         UCB$W_DEVSTS(R5),90$
                          06A9   937         PROTLO  BADACK
              0080    31  06AF   938         BRW     180$                     ; Retry without rep state
            0081 C5   90  06B2   939 90$:    MOVB    UCB$B_ERTMAX(R5),-       ; Reset the retry counter
            0080 C5       06B6   940         UCB$B_ERTCNT(R5)
                          06B9   941         PROTLO  REP                      ; Record a retry
               FCA5   31  06BF   942         BRW     SND_REC_REQ              ; Resend the last block
            51    3D  91  06C2   943 100$:   CMPB    #EBCNAK,R1               ; Is it a NAK
                  1F  12  06C5   944         BNEQ    120$                     ; BR if not
                          06C7   945         PROTLO  NAK                      ; Report a nak sent
        68 A5     01  AA  06CD   946         BICW    #UCB$M_XW_REP,UCB$W_DEVSTS(R5) ; Reset possiable rep state
            0080 C5   97  06D1   947         DECB    UCB$B_ERTCNT(R5)         ; Dec. the retry count
                  03  15  06D5   948         BLEQ    110$                     ; Retry if threshold not exceeded
               FC8D   31  06D7   949         BRW     SND_REC_REQ              ; Try again
            51    02  3C  06DA   950 110$:   MOVZWL  #XW$M_DATACK,R1          ; Get retry threshold exceeded status
                          06DD   951         PROTLO  NAKOUT
               00D9   31  06E3   952         BRW     ABORT_CUR                ; And abort the request and queue
            51    37  91  06E6   953 120$:   CMPB    #EBCEOT,R1               ; Is it an EOT
                  0C  12  06E9   954         BNEQ    130$                     ; BR if not
                          06EB   955         PROTLO  ABEOT
            51    01  3C  06F1   956         MOVZWL  #XW$M_EOT,R1             ; Get EOT status
               00C8   31  06F4   957         BRW     ABORT_CUR                ; Abort the IRP and queue
            51    02  91  06F7   958 130$:   CMPB    #EBCSTX,R1               ; Is it a conversational response
                  0E  12  06FA   959         BNEQ    140$                     ; Consider it a garbled response
                          06FC   960         PROTLO  CON_RESP
    51    0800 8F   3C  0702   961         MOVZWL  #XW$M_CONACK,R1          ; Get conversational response status
                          0707   962         ; Toggle ACKs here ?
               0085   31  0707   963         BRW     COMP_GOOD_AB             ; Complete the request with status
                          070A   964 140$:   PROTLO  GARBLED_RESP
                  14  11  0710   965         BRB     170$                     ; Go to garbled response processing
                          0712   966 ;
                          0712   967 ; Response timout processing- send an ENQ to see what happened
                          0712   968 ;
               FE9E   30  0712   969 150$:   BSBW    TMO_RESET                ; Reset the transmitter
                          0715   970         PROTLO  TMO_XMT_TXTBLK
               FE95   30  071B   971 160$:   BSBW    TMO_RESET                ; Reset the device and go to fork
    03 64 A5     03  E1  071E   972         BBC     #UCB$V_CANCEL,UCB$W_STS(R5),170$ ;If user cancel get out now
               0090   31  0723   973         BRW     CANCEL_CUR               ; Gancet the IRP
                 00   E2  0726   974 170$:   BBSS    #UCB$V_XW_REP,-          ; Is this a new rep state
          07 68 A5        0728   975         UCB$W_DEVSTS(R5),180$
            0081 C5   90  072B   976         MOVB    UCB$B_ERTMAX(R5),-       ; Reset the retry count
            0080 C5       072F   977         UCB$B_ERTCNT(R5)
            0080 C5   97  0732   978 180$:   DECB    UCB$B_ERTCNT(R5)         ; Retry error ?
                  0C  14  0736   979         BGTR    190$                     ; BR if not
            51    02  3C  0738   980         MOVZWL  #XW$M_DATACK,R1          ; Get a threshold exceeded
                          073B   981         PROTLO  REPOUT
               007B   31  0741   982         BRW     ABORT_CUR                ; Complete the request
            51    2D  9A  0744   983 190$:   MOVZBL  #EBCENQ,R1               ; Get retry protocol
                          0747   984         PROTLO  REP_SENT
  64    0200 8F   B3  074D   985         BITW    #DUP_CSR_M_DSR,(R4)      ; Is DSR up
                  09  12  0752   986         BNEQ    195$                     ; Br if not
                          0754   987         PROTLO  DSR_LOST
               FC2E   31  075A   988         BRW     ABORT_NO_DSR             ; Get out
               0144   30  075D   989 195$:   BSBW    XMT_START_PRO1           ; Start up a protocol transmission
                          0760   990         WFIKPCH 200$,#3                  ; And wait for it to complete
                          076A   991         IOFORK                           ; Create a fork process
```

```
FEA0   31  0770   992          BRW     GET_RESPONSE            ; Join common response processing
           0773   993                                          ;  with rep flag set
FE3D   30  0773   994 200$:    BSBW    TMO_RESET               ; Stop the device
           0776   995          PROTLO  SEN_RES_TIMOUT
FE94   31  077C   996          BRW     GET_RESPONSE            ; Try again
           077F   997          .DSABL  LSB
           077F   998
```

```
                     077F  1000 ;
                     077F  1001 ; Complete the request succesfully
                     077F  1002 ;
                     077F  1003          .ENABL  LSB
                     077F  1004 COMP_GOOD_TGL:
32 A3   40 A3   B0   077F  1005          MOVW    IRP$W_ABCNT(R3),IRP$W_BCNT(P3) ; Set the origional user size
           01   E3   0784  1006          BBCS    #UCB$V_XW_AK1,-
     04 68 A5         0786  1007                  UCB$W_DEVSTS(R5),10$    ; Toggel the ack switch
   68 A5   02   CA   0789  1008          BICL    #UCB$M_XW_AK1,UCB$W_DEVSTS(R5)
                     078D  1009 10$:
                     078D  1010 COMP_GOOD:
           51   D4   078D  1011          CLRL    R1                      ; No device dependant status
                     078F  1012 COMP_GOOD_AB:
   50   32 A3   B0   078F  1013          MOVW    IRP$W_BCNT(R3),R0       ; Set the transfer size in status
 50   50   10   78   0793  1014          ASHL    #16,R0,R0               ; Move it to the high byte
 50   0000'8F   B0   0797  1015          MOVW    #SS$_NORMAL,R0          ; Get success code
                     079C  1016 COMPLETE:
      40 A3   B4   079C  1017 20$:       CLRW    IRP$W_ABCNT(R3)         ; Clean up unused field
0C 2A A3      07   E1   079F  1018          BBC     #IRP$V_DIAGBUF,IRP$W_STS(R3),25$ ; Is a diagnostic buffer here
      7E   50   7D   07A4  1019          MOVQ    R0,-(SP)                ; Save the completion status
  00000000'GF   16   07A7  1020          JSB     G^IOC$DIAGBUFILL        ; Fill the diagnostic buffer if any
      50   8E   7D   07AD  1021          MOVQ    (SP)+,R0                ; Get the status back
                     07B0  1022 25$:     REQCOM                          ; Complete the request
                     07B6  1023
                     07B6  1024 CANCEL_CUR:
           51   D4   07B6  1025          CLRL    R1                      ; No device dependant status
 50   0000'8F   3C   07B8  1026          MOVZWL  #SS$_CANCEL,R0          ; Get cancel status
           09   11   07BD  1027          BRB     30$                     ; And complete the request
                     07BF  1028
                     07BF  1029 ABORT_CUR:
   68 A5   08   A8   07BF  1030          BISW    #UCB$M_XW_PIPE,UCB$W_DEVSTS(R5) ; set pipeline marker
50   0000'8F   3C   07C3  1031          MOVZWL  #SS$_ABORT,R0           ; Get aborted status (device dependant
                     07C8  1032                                          ;  status is already set
                     07C8  1033 30$:
      32 A3   B4   07C8  1034 BAD_COMP:       CLRW    IRP$W_BCNT(R3)          ; Zero byte count
         CF   11   07CB  1035          BRB     20$                     ; Complete the request
                     07CD  1036          .DSABL  LSB
```

```
                     07CD  1038 ;
                     07CD  1039 ; Start a full duplex diagnostic read/write
                     07CD  1040 ;
                     07CD  1041 XMT_INI_DIAG:
   50    32 A3    3C  07CD  1042          MOVZWL  IRP$W_BCNT(R3),R0       ; Get the receive count
   78 A5    50    C0  07D1  1043          ADDL2   R0,UCB$L_SVAPTE(R5)     ; Point to transmit buffer
7E A5    40 A3    B0  07D5  1044          MOVW    IRP$W_ABCNT(R3),UCB$W_BCNT(R5) ; Set xmt. buffer count
0090 C5    50    B0  07DA  1045          MOVW    R0,UCB$B_SLAVE(R5)      ; Set receive count
      2C B3       D0  07DF  1046          MOVL    @IRP$L_SVAPTE(R3),-     ; Set receive buffer pointer
     009C C5          07E2  1047                  UCB$L_XW_SPBF(R5)
                      07E5  1048 ; UCB$L_SVAPTE = transmit buffer pointer
                      07E5  1049 ; UCB$W_BCNT = transmit buffer count
                      07E5  1050 ; UCB$L_XW_SPBF = receive buffer pointer
                      07E5  1051 ; UCB$B_SLAVE = receive buffer count
00A9 C5    01    90  07E5  1052          MOVB    #TSTA$CTS,UCB$B_XW_TISTA(R5) ;set transmit int. state
00AA C5    0C    90  07EA  1053          MOVB    #RST$DIAG,UCB$B_XW_RISTA(R5) ; Set receive state
00A0 C5    04    9A  07EF  1054          MOVZBL  #4,UCB$L_XW_SYCNT(R5)   ; Set the sync count
                     07F4  1055          DSBINT                          ; Disable interrupts
   50    46 A5    9A  07FA  1056          MOVZBL  UCB$L_DEVDEPEND+2(R5),R0 ; Get the sync character
      8200 8F       A1  07FE  1057          ADDW3   #DUP_CSR_M_BYM!DUP_CSR_M_CRI,- ; Set up the param. csr
   02 A4    50          0802  1058                  R0,DUP_CSR_PARM(R4)
      64    10       AA  0805  1059          BICW    #DUP_CSR_M_REN,(R4)     ; Force a resync
64    0154 8F       A8  0808  1060          BISW    #DUP_CSR_M_SSY!DUP_CSR_M_REN!- ; Set RTS,REN,SSY,RIE
                     080D  1061                  DUP_CSR_M_RIE!DUP_CSR_M_RTS,(R4)
      0050 8F       A8  080D  1062          BISW    #DUP_CSR_M_TIE!DUP_CSR_M_SND,- ; Set TIE and SEND
        04 A4          0811  1063                  DUP_CSR_TXCSR(R4)
      01FF 8F       B0  0813  1064          MOVW    #DUP_CSR_M_SOM!PAD,-    ; Set start of message and a PAD
        06 A4          0817  1065                  DUP_CSR_TXDBF(R4)
           09       E1  0819  1066          BBC     #IO$V_MAINTLOOP,-       ; Is it a internal loop
    06 20 A3          081B  1067                  IRP$W_FUNC(R3),2$
      0800 8F       A8  081E  1068          BISW    #DUP_CSR_M_STM,-        ; Set the system test mode bit
        04 A4          0822  1069                  DUP_CSR_TXCSR(R4)
           06       E1  0824  1070 2$:      BBC     #IO$V_NOCTSWAIT,-       ; Is the no CTS wait state requested
    05 20 A3          0826  1071                  IRP$W_FUNC(R3),5$
00A9 C5    05    90  0829  1072          MOVB    #TSTA$DIAG,UCB$B_XW_TISTA(R5) ; Set the no CTS wait state
           0C       E1  082E  1073 5$:      BBC     #IO$V_INTCLOCK,-        ; Is this a internal clock test
    06 20 A3          0830  1074                  IRP$W_FUNC(R3),10$
04 A4    1000 8F       A8  0833  1075          BISW    #DUP_CSR_M_ETM,DUP_CSR_TXCSR(R4) ; Yes turn it on
                     0839  1076 10$:     WFIKPCH 30$,#2                  ; Wait for request to finish
                     0843  1077          DSBINT
                     0849  1078 15$:     WFIKPCH 40$,#2                  ; Wait for receiver to finish
                     0853  1079          IOFORK                          ; Go to fork level
     FF3F CF       DF  0859  1080          PUSHAL  COMPLETE                ; Set up a normal cpmpletion
 50    0000'8F       3C  085D  1081          MOVZWL  #SS$_NORMAL,R0          ; Get normal status
      0090 C5       A3  0862  1082 20$:     SUBW3   UCB$B_SLAVE(R5),-       ; Get the receive size
   51    32 A3          0866  1083                  IRP$W_BCNT(R3),R1
   32 A3    51    B0  0869  1084          MOVW    R1,IRP$W_BCNT(R3)       ; Fake transfer size
51    51    10    78  086D  1085          ASHL    #16,R1,R1               ; Put it in the high byte
        7E A5       A3  0871  1086          SUBW3   UCB$W_BCNT(R5),-        ; Get the transmit count
   51    40 A3          0874  1087                  IRP$W_ABCNT(R3),R1
   2A A3    02    A8  0877  1088          BISW    #IRP$M_FUNC,IRP$W_STS(R3) ; Set read for completion code
                     05  087B  1089          RSB                             ; Join the common completion code
                     087C  1090
15 64 A5    03    E0  087C  1091 30$:     BBS     #UCB$V_CANCEL,UCB$W_STS(R5),50$ ; Is a cancel requested
                     0881  1092          DSBINT
           B0    11  0887  1093          BRB     10$                     ; NO keep waiting
08 64 A5    03    E0  0889  1094 40$:     BBS     #UCB$V_CANCEL,UCB$W_STS(R5),50$ ; Is a cancel requested
```

```
                        088E  1095          DSBINT
                B3   11 0894  1096          BRB     15$
                        0896  1097
              FD1A   30 0896  1098 50$:     BSBW    TMO_RESET
50      0000'8F      3C 0899  1099          MOVZWL  #SS$_CANCEL,R0          ; Get cancel status
        FF26 CF      DF 089E  1100          PUSHAL  BAD_COMP                ; and complete the request
                BE   11 08A2  1101          BRB     20$                     ; In the common code
```

```
                        08A4  1103 ;
                        08A4  1104 ; Transmit startup utility
                        08A4  1105 ;
                        08A4  1106 XMT_START_PRO1:
                        08A4  1107          .ENABL  LSB
      7E A5   01   B0   08A4  1108          MOVW    #1,UCB$W_BCNT(R5)       ; Set the transfer count at one
              04   11   08A8  1109          BRB     10$                     ; Now load the buffer with the message
                        08AA  1110 XMT_START_PRO2:
      7E A5   02   B0   08AA  1111          MOVW    #2,UCB$W_BCNT(R5)       ; Set the transfer count at two
 009C C5   51   D0   08AE  1112 10$:      MOVL    R1,UCB$L_XW_SPBF(R5)    ; Load the buffer with the protocol mes.
      009C C5   DE   08B3  1113          MOVAL   UCB$L_XW_SPBF(R5),-     ; Set the protocol buffer pointer
        78 A5        08B7  1114                  UCB$L_SVAPTE(R5)
                        08B9  1115 XMT_START:
 00A9 C5   01   90   08B9  1116          MOVB    #TSTA$CTS,UCB$B_XW_TISTA(R5) ; Set the starting state to CTS
 00A0 C5   08   9A   08BE  1117          MOVZBL  #8,UCB$L_XW_SYCNT(R5)   ; set the sync count
      64   04   A8   08C3  1118          BISW    #DUP_CSR_M_RTS,(R4)     ; Set request to send
      50   8E   D0   08C6  1119          MOVL    (SP)+,R0                ; Save the return
                        08C9  1120          DSBINT                          ; Disable interrupts
      7E   50   D0   08CF  1121          MOVL    R0,-(SP)                ; Put the return back
      8200 8F   B0   08D2  1122          MOVW    #DUP_CSR_M_CRI!DUP_CSR_M_BYM,-
        02 A4        08D6  1123                  DUP_CSR_PARM(R4)
      0058 8F   A8   08D8  1124          BISW    #DUP_CSR_M_TIE!DUP_CSR_M_HDX!DUP_CSR_M_SND,- ; Turn on interrupt
        04 A4        08DC  1125                  DUP_CSR_TXCSR(R4)
06 A4 01FF 8F   B0   08DE  1126          MOVW    #DUP_CSR_M_SOM!PAD,DUP_CSR_TXDBF(R4) ; Set SOM and PAD
                        08E4  1127          ; Is the TSOM and SEND order important ?
              05        08E4  1128          RSB                             ;Return from subroutine
                        08E5  1129          .DSABL  LSB
                        08E5  1130 ;
                        08E5  1131 ;Receive startup utility
                        08E5  1132 ;
                        08E5  1133 RCV_START_PRO:
   00AA C5   02   90   08E5  1134          MOVB    #RST$FIRST,UCB$B_XW_RISTA(R5) ; Set starting receive state
78 A5 009C C5   DE   08EA  1135          MOVAL   UCB$L_XW_SPBF(R5),UCB$L_SVAPTE(R5) ; Set the protocol buffer
      7E A5   02   B0   08F0  1136          MOVW    #2,UCB$W_BCNT(R5)       ; Set the protocol response
                        08F4  1137 RCV_START:
      50   46 A5   9A   08F4  1138          MOVZBL  UCB$L_DEVDEPEND+2(R5),R0 ; Set the SYN character in R0
      8200 8F   A1   08F8  1139          ADDW3   #DUP_CSR_M_BYM!DUP_CSR_M_CRI,- ; Set SYN char. and no crc
 02 A4   50        08FC  1140                  R0,DUP_CSR_PARM(R4)
      64   10   AA   08FF  1141          BICW    #DUP_CSR_M_REN,(R4)     ; Force a resync
      50   8E   D0   0902  1142          MOVL    (SP)+,R0                ; Save the return
                        0905  1143          DSBINT                          ; Disable interrupts
```

```
                7E   50    D0  090B  1145          MOVL    R0,-(SP)                        ; Put the return back
            64  0150 8F    A8  090E  1146          BISW    #DUP_CSR_M_SSY!DUP_CSR_M_REN!DUP_CSR_M_RIE,(R4) ; Start the rcv.
                           05  0913  1147          RSB                                     ; Return to caller
```

```
                     0914  1149          .SBTTL  Block Checking
                     0914  1150 ;+
                     0914  1151 ;
                     0914  1152 ; Remove all control characters, check all CRCs, and compact the data into
                     0914  1153 ; the same buffer.
                     0914  1154 ;
                     0914  1155 ;
                     0914  1156 ; input
                     0914  1157 ;       R0- count from R1 to end of buffer
                     0914  1158 ;       R1- pointer to byte after STX
                     0914  1159 ;
                     0914  1160 ; output
                     0914  1161 ;       R0, R1- destroyed
                     0914  1162 ;       control is passed to either the NAK or ACK routine after checking
                     0914  1163 ;
                     0914  1164 ; Register conventions
                     0914  1165 ;       R6-ITB pointer
                     0914  1166 ;       R7-count pointer
                     0914  1167 ;       R8-ITB count
                     0914  1168 ;       R9- pointer to beginning of raw data
                     0914  1169 ;       R10-count to CRC
                     0914  1170 ;
                     0914  1171 ;
                     0914  1172 ;-
          0000018C   0914  1173          DLE_STX_CRC=^X18C                       ; DLE, STX string CRC
          0000C181   0914  1174          STX_CRC=^XC181                          ; Partial CRC for STX
                     0914  1175
                     0914  1176 BLK_CHECK:
                     0914  1177          .ENABL  LSB
                06F8 0914  1178          PUSHR   #^M<R3,R4,R5,R6,R7,R9,R10>
                59   51    D0  0918  1179          MOVL    R1,R9                   ; Save the input pointer
             57   2C B3    D0  091B  1180          MOVL    @IRP$L_SVAPTE(R3),R7    ; Get the output pointer
          56   00A4 C5    D0  091F  1181          MOVL    UCB$L_XW_ITBPTR(R5),R6  ; Get the ITB table base
                56   1C    C0  0924  1182          ADDL    #7*4,R6                 ; Point to tne first ITB
          7E   51   50    C1  0927  1183          ADDL3   R0,R1,-(SP)             ; Point to end of block
                     7E    D4  092B  1184          CLRL    -(SP)                   ; No partial CRC
                5A   66    D0  092D  1185          MOVL    (R6),R10                ; Are there any internal CRC
                     3A    12  0930  1186          BNEQ    20$                     ; BR if there are
             2C B3   79    3E  0932  1187          MOVAW   -(R9),@IRP$L_SVAPTE(R3) ; Fake the system buffer pointer
          89   50   03    A3  0936  1188          SUBW3   #3,R0,(R9)+             ; Set count for user
                5A   50    D0  093A  1189          MOVL    R0,R10                  ; Set up the count
69   5A   6E    00000CAC'EF    0B  093D  1190 5$:       CRC     POLY_TABLE,(SP),R10,(R9) ; DO the CRC on the whole block
                     6C    12  0946  1191          BNEQ    BAD_CRC                 ; BR if not zero
          FD A3    FFFF  8F    B0  0948  1192          MOVW    #-1,-3(R3)              ; Store the sentenal
          50   53   01    C3  094E  1193          SUBL3   #1,R3,R0                ; Point to end of block
                     0086  31    0952  1194          BRW     GOOD_TEXT               ; Send a ACk
                5A   76    D0  0955  1195 10$:      MOVL    -(R6),R10               ; Get count to end of CRC
                     12    12  0958  1196          BNEQ    20$                     ; Not end so branch
          5A   04 AE   59    C3  095A  1197          SUBL3   R9,4(SP),R10            ; Get the remaining count
             87   5A   03    A3  095F  1198          SUBW3   #3,R10,(R7)+            ; Set the last count
             67   69   5A    28  0963  1199          MOVC    R10,(R9),(R7)           ; Move the last record
                59   57    D0  0967  1200          MOVL    R7,R9                   ; Fake the last CRC
                     D1    11  096A  1201          BRB     5$                      ; Do the last CRC
                5A   59    C2  096C  1202 20$:      SUBL    R9,R10                  ; Get count to next ITB CRC
69   5A   6E    00000CAC'EF    0B  096F  1203 30$:      CRC     POLY_TABLE,(SP),R10,(R9) ; Do the partial CRC
                     3A    12  0978  1204          BNEQ    BAD_CRC                 ; BR if bad CRC
             87   5A   03    A3  097A  1205          SUBW3   #3,R10,(R7)+            ; store the count in the buffer
```

```
                67   69   5A   28  097E  1206          MOVC    R10,(R9),(R7)           ; And compact the data
                57   53   03   C3  0982  1207          SUBL3   #3,R3,R7                ; Point to next count
             5A   04 AE   51   C3  0986  1208          SUBL3   R1,4(SP),R10            ; Get the count remaining
                61   5A   32   3B  098B  1209          SKPC    #EBCSYN,R10,(R1)        ; Skip the SYN
                          23   13  098F  1210          BEQL    BAD_CRC                 ; BR if bad CRC
             6E   0000018C 8F   D0  0991  1211          MOVL    #DLE_STX_CRC,(SP)       ; start the next CRC
                81   0210 8F   B1  0998  1212          CMPW    #EBCDLE!^EBCSTX*256>,(R1)+ ; The DLE STX are in the CRC
                          10   13  099D  1213          BEQL    40$                     ; BR if  here
                          6E   D4  099F  1214          CLRL    (SP)                    ; Set no partial CRC
                          71   B5  09A1  1215          TSTW    -(R1)                   ; Back up
                     61   02   91  09A3  1216          CMPB    #EBCSTX,(R1)            ; Dies the next record start with STX
                          07   12  09A6  1217          BNEQ    40$                     ; Br if not
                6E   C181 8F   3C  09A8  1218          MOVZWL  #STX_CRC,(SP)           ; Get the STX partial crc
                          51   D6  09AD  1219          INCL    R1                      ; Remove the STX
                     59   51   D0  09AF  1220 40$:     MOVL    R1,R9                   ; Set the new from pointer
                          A1   11  09B2  1221          BRB     10$                     ; Get next record
                               09B4  1222 BAD_CRC:
                     5E   08   C0  09B4  1223          ADDL    #2*4,SP                 ; Pop the CRC and end
                     06F8 8F   BA  09B7  1224          POPR    #^M<R3,R4,R5,R6,R7,R9,R10>
                          04   C8  09BB  1225          BISL    #UCB$M_XW_NAK,-  ; Mark a NAK sent
                            68 A5        09BD  1226                          UCB$W_DEVSTS(R5)
                               09BF  1227          PROTLO  NAK_SENT
                       FB8D   30  09C5  1228          BSBW    SEND_RESP               ; Set up and start the NAK
                               09C8  1229          WFIKPCH REC_SNAK_TMO,#4
                               09D2  1230          IOFORK                          ; Create a fork process
                       F98C   31  09D8  1231          BRW     SND_REC_REQ             ; Restart the receive buffer
                               09DB  1232 GOOD_TEXT:
                     5E   08   C0  09DB  1233          ADDL    #2*4,SP                 ; Pop the CRC and end
                     06F8 8F   BA  09DE  1234          POPR    #^M<R3,R4,R5,R6,R7,R9,R10>
                   50   2C B3   C2  09E2  1235          SUBL    @IRP$L_SVAPTE(R3),R0    ; Get buffer size
                   40 A3   50   B1  09E6  1236          CMPW    R0,IRP$W_ABCNT(R3)      ; Is the users buffer large enough
                          08   19  09EA  1237          BLSS    55$                     ; Br if it is
                50   0000'8F   3C  09EC  1238          MOVZWL  #SS$_TOOMUCHDATA,R0     ; Get a error message for the user
                       FDD4   31  09F1  1239          BRW     BAD_COMP                ; And return the buffer with a fatal err
                   32 A3   50   B0  09F4  1240 55$:     MOVW    R0,IRP$W_BCNT(R3)       ; Set the transfer size for the iocomp.
                          01   E3  09F8  1241          BBCS    #UCB$V_XW_AK1,-         ; Toggel the ACK switch
                            04 68 A5        09FA  1242                          UCB$W_DEVSTS(R5),60$    ; ...
                      68 A5   02   CA  09FD  1243          BICL    #UCB$M_XW_AK1,UCB$W_DEVSTS(R5) ;...
                          04   CA  0A01  1244 60$:     BICL    #UCB$M_XW_NAK,-         ; Reset NAK sent last switch
                            68 A5        0A03  1245                          UCB$W_DEVSTS(R5)
             03 64 A5   03   E1  0A05  1246          BBC     #UCB$V_CANCEL,UCB$W_STS(R5),70$ ; If canceled the higher
                               0A0A  1247                                          ;  levels will want to send an EOT
                               0A0A  1248                                          ;  Rather than this ACK
                       FDA9   31  0A0A  1249          BRW     CANCEL_CUR              ; Return this packet cancled
                       FB45   30  0A0D  1250 70$:     BSBW    SEND_RESP               ; Set up the correct response
                               0A10  1251          WFIKPCH REC_GACK_TMO,#4         ; And wait for it to be sent
                               0A1A  1252          IOFORK                          ; Create a fork process
                               0A20  1253 REC_DONE:
                   51   3C A3   D0  0A20  1254          MOVL    IRP$L_MEDIA+4(R3),R1    ; Get the dev. dep. status
                       FD68   31  0A24  1255          BRW     COMP_GOOD_AB            ; Complete the request
                               0A27  1256
                               0A27  1257          .DSABL  LSB
```

```
                     0A27  1259          .SBTTL  DUP_11 Receive Interrupt Processing
                     0A27  1260 ;+
                     0A27  1261 ;XW_RINT - DUP-11 RECEIVE INTERRUPT PROCESSING
                     0A27  1262 ;
                     0A27  1263 ; THIS ROUTINE IS ENTERED VIA A JSB INSTRUCTION WHEN AN INTERRUPT OCCURS ON A
                     0A27  1264 ; DUP-11 LINE INTERFACE. THE STATE OF THE STACK ON ENTRY IS:
                     0A27  1265 ;
                     0A27  1266 ;       00(SP) = ADDRESS OF IDB ADDRESS.
                     0A27  1267 ;       04(SP) = SAVED R3.
                     0A27  1268 ;       08(SP) = SAVED R4.
                     0A27  1269 ;       12(SP) = SAVED R5.
                     0A27  1270 ;       16(SP) = INTERRUPT PC.
                     0A27  1271 ;       20(SP) = INTERRUPT PSL.
                     0A27  1272 ;
                     0A27  1273 ; Interrupt dispatching occurs according to received data.
                     0A27  1274 ;
                     0A27  1275 ;-
                     0A27  1276 XW_RINT::
                     0A27  1277          .ENABL  LSB
    53    9E   D0    0A27  1278          MOVL    @(SP)+,R3                   ; Get the IDB pointer
    54    63   7D    0A2A  1279          MOVQ    IDB$L_CSR(R3),R4            ; Set CSR and UCB pointer
64 A5     02   AA    0A2D  1280          BICW    #UCB$M_INT,UCB$W_STS(R5) ; Clear expected bit
50   02 A4   9A      0A31  1281          MOVZBL  DUP_CSR_RDBF(R4),R0         ; Get the received byte
                     0A35  1282          CASE    UCB$B_XW_RISTA(R5),-        ; Vector the interrupt
                     0A35  1283                  TYPE=B,-                    ; Byte data type
                     0A35  1284                  <RST_IDLE,-                 ; Idle state
                     0A35  1285                  RST_BINARY,-                ; Binary reception
                     0A35  1286                  RST_FIRST,-                 ; First character state
                     0A35  1287                  RST_NTTX,-                  ; Nontransparent text
                     0A35  1288                  RST_STOP1,-                 ; First byte of last CRC
                     0A35  1289                  RST_STOP,-                  ; Store last byte and complete request
                     0A35  1290                  RST_ITB,-                   ; Store internal CRC
                     0A35  1291                  RST_ITB1,-                  ; Store last byte of internal CRC
                     0A35  1292                  RST_XPRTEXT,-               ; Transparent text
                     0A35  1293                  RST_DLE,-                   ; transparent DLE processing
                     0A35  1294                  RST_XPRITB,-                ; Transparent internal CRC
                     0A35  1295                  RST_XPRITB1,-               ; Second byte of internal CRC
                     0A35  1296                  RST_DIAG,-                  ; Diagnostic function
                     0A35  1297                  RST_LOOP_TEST,-             ; Slave loop diagnostic function
                     0A35  1298                  >
                     0A57  1299
                     0A57  1300 ;
                     0A57  1301 ; Second byte of internal CRC state
                     0A57  1302 ;
                     0A57  1303 RST_XPRITB1:
00AA C5    08    90  0A57  1304          MOVB    #RST$XPRTEXT,UCB$B_XW_RISTA(R5) ;Go back to transparent text
           05    11  0A5C  1305          BRB     100$                        ; Go to common internal CRC code
                     0A5E  1306 ;
                     0A5E  1307 ; Store the second internal CRC
                     0A5E  1308 ;
                     0A5E  1309 RST_ITB1:
00AA C5    03    90  0A5E  1310          MOVB    #RST$NTTX,UCB$B_XW_RISTA(R5) ; Go back to text state
                     0A63  1311 ;
                     0A63  1312 ; Internal CRC processing
                     0A63  1313 ;
  78 B5    50    90  0A63  1314 100$:   MOVB    R0,@UCB$L_SVAPTE(R5)        ; Store the last CRC byte
        78 A5    D6  0A67  1315          INCL    UCB$L_SVAPTE(R5)            ; Step pointer
```

```
                7E A5   B7  0A6A  1316          DECW    UCB$W_BCNT(R5)              ; And reduce count
                   5B   15  0A6D  1317          BLEQ    50$                         ; The end of the buffer!!
50           00A8 C5   9A  0A6F  1318          MOVZBL  UCB$B_XW_ITBCNT(R5),R0      ; Get the index
                78 A5   D0  0A74  1319          MOVL    UCB$L_SVAPTE(R5),-          ; Save the pointer
           00A4 D540        0A77  1320                  @UCB$L_XW_ITBPTR(R5)[R0]
                   50   97  0A7B  1321          DECB    R0                          ; One less CRC
                   4B   15  0A7D  1322          BLEQ    50$                         ; More than 8 ??
00A8 C5            50   90  0A7F  1323          MOVB    R0,UCB$B_XW_ITBCNT(R5)      ; Save the new count
                   71   11  0A84  1324          BRB     25$                         ; get out
                            0A86  1325
                            0A86  1326 ;
                            0A86  1327 ; First character state
                            0A86  1328 ;
                            0A86  1329 RST_FIRST:
64         0100 8F   AA  0A86  1330          BICW    #DUP_CSR_M_SSY,(R4)         ; Reset strip sync
00AA C5         03   90  0A8B  1331          MOVB    #RST$NTTX,UCB$B_XW_RISTA(R5) ; Assume non-transparent text
      50        10   91  0A90  1332          CMPB    #EBCDLE,R0                  ; Is it non-transparent
                4E   12  0A93  1333          BNEQ    RST_NTTX                    ; Process it if not
00AA C5         08   90  0A95  1334          MOVB    #RST$XPRTEXT,UCB$B_XW_RISTA(R5) ; Set state to transparent text
                4F   11  0A9A  1335          BRB     RIDONE                      ; Clean up stack and exit
                            0A9C  1336
      50        1F   91  0A9C  1337 30$:     CMPB    #EBCITB,R0                  ; Will an internal CRC be next
                07   12  0A9F  1338          BNEQ    35$                         ; BR if not
00AA C5         06   90  0AA1  1339          MOVB    #RST$ITB,UCB$B_XW_RISTA(R5) ; Set state to store CRC
                43   11  0AA6  1340          BRB     RIDONE                      ; Complete interrupt
      50        32   91  0AA8  1341 35$:     CMPB    #EBCSYN,R0                  ; Is it a SYN in text
                4A   13  0AAB  1342          BEQL    25$                         ; If it is BR and ignore it
      50        2D   91  0AAD  1343          CMPB    #EBCENQ,R0                  ; Is it a forward abort
                0E   13  0AB0  1344          BEQL    RST_STOP                    ; Complete the block now if it is
00AA C5         04   90  0AB2  1345          MOVB    #RST$STOP1,UCB$B_XW_RISTA(R5) ; Set state to get last CRC
                32   11  0AB7  1346          BRB     RIDONE                      ; Complete the interrupt
                            0AB9  1347 ;
                            0AB9  1348 ; First byte of last CRC state
                            0AB9  1349 ;
                            0AB9  1350 RST_STOP1:
      00AA C5         05   90  0AB9  1351          MOVB    #RST$STOP,UCB$B_XW_RISTA(R5) ; Set state for stop after next
                      28   11  0ABE  1352          BRB     RIDONE                      ; Complete the interrupt
                            0AC0  1353 ;
                            0AC0  1354 ; End of buffer state
                            0AC0  1355 ;
                            0AC0  1356 RST_STOP:
        78 B5           50   90  0AC0  1357          MOVB    R0,@UCB$L_SVAPTE(R5)        ; Store it in the buffer
              78 A5     D6   0AC4  1358          INCL    UCB$L_SVAPTE(R5)            ; Step the buffer pointer
              7E A5     B7   0AC7  1359          DECW    UCB$W_BCNT(R5)              ; Include the byte in the count
      00AA C5           00   90  0ACA  1360 50$:     MOVB    #RST$IDLE,UCB$B_XW_RISTA(R5) ; Set state to idle
      64        0050 8F   AA  0ACF  1361          BICW    #DUP_CSR_M_RIE!DUP_CSR_M_REN,(R4) ; Turn interrupts off
        53        10 A5   D0  0AD4  1362          MOVL    UCB$L_FR3(R5),R3            ; Restore the fork R3
      00AB C5           64   B0  0AD8  1363          MOVW    (R4),UCB$W_XW_LSTCSR(R5) ; Save the ending CSR
              0C B5     16   0ADD  1364          JSB     @UCB$L_FPC(R5)              ; Call back in line to complete request
               0014     31   0AE0  1365          BRW     25$                         ; Clean up and exit the interrupt
                            0AE3  1366
                            0AE3  1367 ;
                            0AE3  1368 ; Non-transparent text state
                            0AE3  1369 ;
                            0AE3  1370 RST_NTTX:
B1 00000C8C'EF              50   E0  0AE3  1371          BBS     R0,CCHRMASK,30$         ; BR if a control char.
                            0AEB  1372 RIDONE:
```

```
                78 B5    50    90  0AEB  1373          MOVB    R0,@UCB$L_SVAPTE(R5)    ; Store it in the buffer
                   78 A5  D6    0AEF  1374          INCL    UCB$L_SVAPTE(R5)        ; Step the buffer pointer
                   7E A5  B7    0AF2  1375          DECW    UCB$W_BCNT(R5)          ; Include the byte in the count
                                0AF5  1376 24$:
                       D3  15   0AF5  1377          BLEQ    50$                     ; BR if the buffer is full
                                0AF7  1378 RST_IDLE:
                    50    8E  7D 0AF7  1379 25$:    MOVQ    (SP)+,R0                ; Restore R0-1
                    52    8E  7D 0AFA  1380         MOVQ    (SP)+,R2                ; Restore R2-3
                    54    8E  7D 0AFD  1381         MOVQ    (SP)+,R4                ; Restore R4-5
                          02  0B00  1382          REI                             ; Return from interrupt
                              0B01  1383 ;
                              0B01  1384 ; Store the first internal CRC character
                              0B01  1385 ;
                              0B01  1386 RST_ITB:
              00AA C5    07   90  0B01  1387          MOVB    #RST$ITB1,UCB$B_XW_RISTA(R5) ; Set state to store second CRC
                         E3   11  0B06  1388          BRB     RIDONE                  ; Complete the interrupt
                              0B08  1389 ;
                              0B08  1390 ; Transparent text state- if DLE is found drop it and go to DLE state
                              0B08  1391 ;
                              0B08  1392 RST_XPRTEXT:
                    50    10   91  0B08  1393          CMPB    #EBCDLE,R0              ; Is it a DLE
                          DE   12  0B0B  1394          BNEQ    RIDONE                  ; If not continue
              00AA C5    09   90  0B0D  1395          MOVB    #RST$DLE,UCB$B_XW_RISTA(R5) ; Set state to DLE
                         D7   11  0B12  1396          BRB     RIDONE                  ; Complete the interrupt
                              0B14  1397 ;
                              0B14  1398 ; Transparent character after a DLE
                              0B14  1399 ;
                              0B14  1400 RST_DLE:
                    50    1F   91  0B14  1401          CMPB    #EBCITB,R0              ; Is it an internal CRC
                          0D   12  0B17  1402          BNEQ    40$                     ; Check sone more
              00AA C5    0A   90  0B19  1403          MOVB    #RST$XPRITB,UCB$B_XW_RISTA(R5) ; Set transparent ITB state
                   78 A5  D7   0B1E  1404 42$:       DECL    UCB$L_SVAPTE(R5)        ; Remove the DLE
                   7E A5  B6   0B21  1405          INCW    UCB$W_BCNT(R5)          ; ...
                          C5   11  0B24  1406          BRB     RIDONE                  ; Exit the interrupt
                    50    32   91  0B26  1407 40$:     CMPB    #EBCSYN,R0              ; Is it a SYN in text
                          0D   12  0B29  1408          BNEQ    45$                     ; BR if not
              00AA C5    08   90  0B2B  1409          MOVB    #RST$XPRTEXT,UCB$B_XW_RISTA(R5) ; Set state to text
                   78 A5  D7   0B30  1410          DECL    UCB$L_SVAPTE(R5)        ; Remove the DLE also
                   7E A5  B6   0B33  1411          INCW    UCB$W_BCNT(R5)          ; ...
                          BF   11  0B36  1412          BRB     25$                     ; And exit the interrupt
        0C 00000C8C'EF    50   E0  0B38  1413 45$:     BBS     R0,CCHRMASK,47$         ; If it is a cont. cha. strip the DLE
              00AA C5    08   90  0B40  1414          MOVB    #RST$XPRTEXT,UCB$B_XW_RISTA(R5) ;Go back to transparent text
                    50    10   91  0B45  1415          CMPB    #EBCDLE,R0              ; Is it a DLE in text
                          AD   13  0B48  1416          BEQL    25$                     ; If so dump it
                          9F   11  0B4A  1417          BRB     RIDONE                  ; get out
              00AA C5    04   90  0B4C  1418 47$:     MOVB    #RST$STOP1,UCB$B_XW_RISTA(R5) ; Set state to stop after CRC
                         CB   11  0B51  1419          BRB     42$                     ; Complete the interrupt
                              0B53  1420 ;
                              0B53  1421 ; Transparent ITB state
                              0B53  1422 ;
                              0B53  1423 RST_XPRITB:
              00AA C5    0B   90  0B53  1424          MOVB    #RST$XPRITB1,UCB$B_XW_RISTA(R5) ; Go to second byte of CRC
                         91   11  0B58  1425          BRB     RIDONE                  ; Complete the interrupt
                              0B5A  1426 RST_BINARY:
              64    0100 8F    AA  0B5A  1427          BICW    #DUP_CSR_M_SSY,(R4)     ; Reset strip sync
                      FF89     31  0B5F  1428          BRW     RIDONE
                              0B62  1429
```

```
                     0B62  1430 ;
                     0B62  1431 ; Diagnostic interrupt service
                     0B62  1432 ;
                     0B62  1433 RST_DIAG:
64   0100 8F    AA   0B62  1434          BICW    #DUP_CSR_M_SSY,(R4)      ; Drop strip sync
009C D5    50    90   0B67  1435          MOVB    R0,@UCB$L_XW_SPBF(R5)    ; Store the received byte
     009C C5    D6   0B6C  1436          INCL    UCB$L_XW_SPBF(R5)        ; Step the pointer
     0090 C5    B7   0B70  1437          DECW    UCB$B_SLAVE(R5)          ; Reduce the count
        FF7E    31   0B74  1438          BRW     24$
                     0B77  1439 ;
                     0B77  1440 ; Diagnostic slave loop function- receive until the first PAD
                     0B77  1441 ;
                     0B77  1442 RST_LOOP_TEST:
  50   FF 8F    91   0B77  1443          CMPB    #PAD,R0                  ; Is it a pad ?
           DD    12   0B7B  1444          BNEQ    RST_BINARY               ; If not handle it like binary
        FF4A    31   0B7D  1445          BRW     50$                      ; If it is the message is done
                     0B80  1446          .DSABL  LSB
```

```
                                B 8
                                        .SBTTL  DUP-11 Transmit Interrupt Processing
                                ;+
                                ; XW_TINIT - DUP-11 TRANSMIT PROCESSING
                                ;
                                ; THIS ROUTINE IS ENTERED VIA A JSB INSTRUCTION WHEN AN INTERRUPT OCCURS ON A
                                ; DUP-11 LINE INTERFACE. THE STATE OF THE STACK ON ENTRY IS:
                                ;
                                ;       00(SP) = ADDRESS OF IDB ADDRESS.
                                ;       04(SP) = SAVED R3.
                                ;       08(SP) = SAVED R4.
                                ;       12(SP) = SAVED R5.
                                ;       16(SP) = INTERRUPT PC.
                                ;       20(SP) = INTERRUPT PSL.
                                ;
                                ; INTERRUPT DISPATCHING OCCURS BASED ON TRANSMIT STATE
                                ;
                                ;-
                                XW_TINT::
      53    9E    D0  0B80  1467          MOVL    @(SP)+,R3               ; Get the IDB pointer
      54    63    7D  0B83  1468          MOVQ    IDB$L_CSR(R3),R4        ; And the UDP csr and the UCB
   64 A5    02    AA  0B86  1469          BICW    #UCB$M_INT,UCB$W_STS(R5) ; Clear expected bit
                      0B8A  1470          CASE    UCB$B_XW_TISTA(R5),-    ; And vector the interrupt
                      0B8A  1471                  TYPE=B,-
                      0B8A  1472                  <IPRO_IDLE,-            ; No interrupt expected
                      0B8A  1473                  IPRO_CTS,-              ; Wait for clear to send
                      0B8A  1474                  IPRO_SYN,-              ; synch state
                      0B8A  1475                  IPRO_TEXT,-             ; Text state
                      0B8A  1476                  IPRO_ENDPAD,-           ; Last pad state
                      0B8A  1477                  IPRO_DIAG,-             ; Diagnostic (no CTS wait state)
                      0B8A  1478                  IPRO_DROPRTS,-          ; Drop RTS state
                      0B8A  1479                  >
                      0B9E  1480
                      0B9E  1481 IPRO_CTS:
 64   2000 8F    B3  0B9E  1482          BITW    #DUP_CSR_M_CTS,(R4)
           07    12  0BA3  1483          BNEQ    10$
06 A4   FF 8F    90  0BA5  1484          MOVB    #PAD,DUP_CSR_TXDBF(R4)  ; send a pad and wait for CTS
           23    11  0BAA  1485          BRB     IDONE                   ; Go exit the interrupt
                      0BAC  1486 10$:
                      0BAC  1487 IPRO_DIAG:
 00A9 C5    02    90  0BAC  1488          MOVB    #TSTA$SYN,UCB$B_XW_TISTA(R5) ; Set state to send synch
                      0BB1  1489 IPRO_SYN:
06 A4    46 A5    90  0BB1  1490          MOVB    UCB$L_DEVDEPEND+2(R5),DUP_CSR_TXDBF(R4) ; send a synch
   14 00A0 C5    F5  0BB6  1491          SOBGTR  UCB$L_XW_SYCNT(R5),IDONE ; Dec SYN count
 00A9 C5    03    90  0BBB  1492          MOVB    #TSTA$TEXT,UCB$B_XW_TISTA(R5) ; set state to text
           0D    11  0BC0  1493          BRB     IDONE                   ; Exit this interrupt
                      0BC2  1494          .ENABL  LSB
                      0BC2  1495 IPRO_TEXT:
      7E A5    B7  0BC2  1496          DECW    UCB$W_BCNT(R5)          ; Reduce the byte count
           12    19  0BC5  1497          BLSS    10$                     ; Change to trailing pad state
      78 B5    90  0BC7  1498          MOVB    @UCB$L_SVAPTE(R5),-     ; Send the next byte
      06 A4                0BCA  1499                  DUP_CSR_TXDBF(R4)
      78 A5    D6  0BCC  1500          INCL    UCB$L_SVAPTE(R5)        ; Step the pointer to next byte
                      0BCF  1501 IPRO_IDLE:
      50    8E    7D  0BCF  1502 IDONE:  MOVQ    (SP)+,R0                ; Restore saved regesters
      52    8E    7D  0BD2  1503          MOVQ    (SP)+,R2                ; ...
      54    8E    7D  0BD5  1504          MOVQ    (SP)+,R4                ; ...
                02    0BD8  1505          REI                             ; Return from the interrupt
```

```
                        0BD9  1506
00A9 C5   04    90      0BD9  1507 10$:     MOVB    #TSTA$ENDPAD,UCB$B_XW_TISTA(R5) ; Set trailing pad state
  7E A5   02    B0      0BDE  1508          MOVW    #2,UCB$W_BCNT(R5)               ; Set the # of trail PADS to 2
                        0BE2  1509 IPRO_ENDPAD:
        7E A5   B7      0BE2  1510          DECW    UCB$W_BCNT(R5)                  ; Decr PAD count
           05   14      0BE5  1511          BGTR    20$                             ; Is it the last ont?
00A9 C5   06    90      0BE7  1512          MOVB    #TSTA$DROPRTS,UCB$B_XW_TISTA(R5) ; Set drop RTS state
06 A4   FF 8F   90      0BEC  1513 20$:     MOVB    #PAD,DUP_CSR_TXDBF(R4)          ; Send the trailing pad
           DC   11      0BF1  1514          BRB     IDONE                           ; Exit the interrupt
                        0BF3  1515          .DSABL  LSB
                        0BF3  1516 IPRO_DROPRTS:
00AB C5   64    B0      0BF3  1517          MOVW    (R4),UCB$W_XW_LSTCSR(R5)        ; Save the last CSR
      0050 8F   AA      0BF8  1518          BICW    #DUP_CSR_M_TIE!DUP_CSR_M_SND,-  ; Turn off interrupt enable
        04 A4           0BFC  1519          DUP_CSR_TXCSR(R4)
03 44 A5    0C  E0      0BFE  1520          BBS     #XW$V_CHA_FDX,UCB$L_DEVDEPEND(R5),10$ ; Is this line HDX
      64    04  AA      0C03  1521          BICW    #DUP_CSR_M_RTS,(R4)             ; Drop request to send
00A9 C5   00    90      0C06  1522 10$:     MOVB    #TSTA$IDLE,UCB$B_XW_TISTA(R5) ; Set state to idle
  53    10 A5   D0      0C0B  1523          MOVL    UCB$L_FR3(R5),R3                ;Get the fork R3
        0C B5   16      0C0F  1524          JSB     @UCB$L_FPC(R5)                  ; Pass control back to where IO was strt
           BB   11      0C12  1525          BRB     IDONE                           ; Exit the interrupt
```

```
                                0C14  1527          .SBTTL  REGDUMP Dump the errors and device CSRs
                                0C14  1528 ;+
                                0C14  1529 ;       Move the accumulated errors and CSR 0 into an error log buffer
                                0C14  1530 ; or a diagnostic buffer.
                                0C14  1531 ;
                                0C14  1532 ; input
                                0C14  1533 ;       R0- buffer pointer
                                0C14  1534 ;       R5- UCB pointer
                                0C14  1535 ;       R4- CSR pointer
                                0C14  1536 ;
                                0C14  1537 ; output
                                0C14  1538 ;       @R0 is filled with:
                                0C14  1539 ;               # long words filled
                                0C14  1540 ;               last (or current) fatal error
                                0C14  1541 ;               CSR 0
                                0C14  1542 ;               accumulated errors (see UCB definition)
                                0C14  1543 ;
                                0C14  1544 ;-
                                0C14  1545 REGDUMP:
                          06    BB  0C14  1546          PUSHR   #^M<R1,R2>
     51   55   000000AD    8F    C1  0C16  1547          ADDL3   #UCB$B_XW_ERRORS,R5,R1  ; Get a pointer to the first error
                    52    13    D0  0C1E  1548          MOVL    #NO_ERRORS+1,R2         ; Get the number of error fields
                    80    15    D0  0C21  1549          MOVL    #<NO_ERRORS+3>,(R0)+ ; Store no. long words
                    80    81    9A  0C24  1550          MOVZBL  (R1)+,(R0)+             ; Store the last error logged
               80   00AB  C5    3C  0C27  1551          MOVZWL  UCB$W_XW_LSTCSR(R5),(R0)+ ; Store the CSR
                    80    81    9A  0C2C  1552 10$:     MOVZBL  (R1)+,(R0)+             ; Store a long word
                      FA  52    F5  0C2F  1553          SOBGTR  R2,10$                  ; BR if not done
               80   FFFFFFFF    8F    D0  0C32  1554          MOVL    #-1,(R0)+               ; Put in sentinal
                          06    BA  0C39  1555          POPR    #^M<R1,R2>
                          05    05  0C3B  1556          RSB
                                0C3C  1557
                                0C3C  1558          .SBTTL  ERR_RECORD Save an error in the UCB
                                0C3C  1559 ;+
                                0C3C  1560 ; The error is recorded in the appropriate UCB field, if it
                                0C3C  1561 ; is fatal the error logger is called.
                                0C3C  1562 ;
                                0C3C  1563 ; input
                                0C3C  1564 ;       R5- UCB
                                0C3C  1565 ;       R4- Device CSR
                                0C3C  1566 ;       4(SP)- error
                                0C3C  1567 ;       (SP)- return
                                0C3C  1568 ;
                                0C3C  1569 ; output
                                0C3C  1570 ;       The UCB is updated and the error is logged if fatal
                                0C3C  1571 ;
                                0C3C  1572 ;
                                0C3C  1573 ERR_RECORD:
                      7E    50    D0  0C3C  1574          MOVL    R0,-(SP)                ; Save a reg.
     50   08 AE      FF    8F    78  0C3F  1575          ASHL    #-1,2*4(SP),R0          ; Get the error number
                00AD C5     50    90  0C45  1576          MOVB    R0,UCB$B_XW_ERRORS(R5)  ; Save the last error
                   00AE C540      96  0C4A  1577          INCB    UCB$B_XW_ERRORS+1(R5)[R0] ; Step the error cell
                          05    12  0C4F  1578          BNEQ    5$                      ; Dont pass 0
                   00AE C540      97  0C51  1579          DECB    UCB$B_XW_ERRORS+1(R5)[R0] ; ...
                    06 08 AE      E9  0C56  1580 5$:      BLBC    2*4(SP),10$             ; Is this a fatal error
                 00000000'GF      16  0C5A  1581          JSB     G^ERL$DEVICERR          ; Yes call the error logger
                      50    8E    D0  0C60  1582 10$:     MOVL    (SP)+,R0                ; Restore R0
                      6E    8E    D0  0C63  1583          MOVL    (SP)+,(SP)              ; Remove the error
```

05 0C66 1584 RSB

_$2

Sym

C
DTS
DTS
DTS
DTS
DTS
DTS
DTS
DTS
DTS
DTS
DTS
DTS
DTS
DTS
IO$
IO$
LIB
NET
R
RA
RT
SS$
SYS
SYS
SYS
SYS
SYS
SYS
SYS
SYS
SYS
SYS
SYS
SYS
TST
TST
TST
TST
TST
TST
TST
TST
TST
TST
TST
TST
TST
TST
TST

```
                   0C67          .SBTTL  CANCEL I/O ON CHANNEL
                   0C67  ;+
                   0C67  ; XW_CANCELIO - CANCEL I/O ON CHANNEL
                   0C67  ;
                   0C67  ; THIS ROUTINE IS CALLED WHEN THE LAST CHANNEL ASSIGNED TO A DEVICE IS DEASSIGNED,
                   0C67  ; THE DEVICE IS DEALLOCATED, AND WHEN THE CANCEL I/O ON CHANNEL SYSTEM SERVICE IS
                   0C67  ; EXECUTED.
                   0C67  ;
                   0C67  ; INPUTS:
                   0C67  ;
                   0C67  ;       R2 = NEGATIVE CHANNEL NUMBER.
                   0C67  ;       R3 = ADDRESS OF CURRENT I/O REQUEST PACKET.
                   0C67  ;       R4 = CURRENT PROCESS PCB ADDRESS.
                   0C67  ;       R5 = DEVICE UCB ADDRESS.
                   0C67  ;
                   0C67  ; OUTPUTS:
                   0C67  ;
                   0C67  ;       THE DEVICE INDEPENDENT CANCEL I/O ROUTINE IS CALLED.
                   0C67  ;
                   0C67  ;-
                   0C67
                   0C67  XW_CANCELIO:
00000000'GF   17   0C67          JMP     G^IOC$CANCELIO
```

```
                        0C6D  1610          .SBTTL  DUP-11 Unit Initalization
                        0C6D  1611 ;+
                        0C6D  1612 ; XW_INIT - DUP-11 LINE INTERFACE UNIT INITIALIZATION
                        0C6D  1613 ;
                        0C6D  1614 ; THIS ROUTINE IS CALLED AT SYSTEM STARTUP AND AFTER A POWER FAILURE. THE
                        0C6D  1615 ; ONLINE BIT IS SET IN THE DEVICE UCB.
                        0C6D  1616 ;
                        0C6D  1617 ; INPUTS:
                        0C6D  1618 ;
                        0C6D  1619 ;       R5 = ADDRESS OF DEVICE UCB.
                        0C6D  1620 ;
                        0C6D  1621 ; OUTPUTS:
                        0C6D  1622 ;
                        0C6D  1623 ;       THE ONLINE BIT IS SET IN THE DEVICE UCB AND THE ADDRESS OF THE UCB
                        0C6D  1624 ;       IS FILLED INTO THE OWNER FIELD OF THE IDB.
                        0C6D  1625 ;-
                        0C6D  1626
                        0C6D  1627 XW_INIT:
    64 A5    10    A8   0C6D  1628          BISW    #UCB$M_ONLINE,UCB$W_STS(R5) ; Mark unit on line
    50   24 A5    D0   0C71  1629          MOVL    UCB$L_CRB(R5),R0            ; Get the CRB pointer
    50   2C A0    D0   0C75  1630          MOVL    CRB$L_INTD+VEC$L_IDB(R0),R0 ; Get the IDB pointer
    04 A0    55    D0   0C79  1631          MOVL    R5,IDB$L_OWNER(R0)          ; Store the UCB pointer
09 64 A5    01    E1   0C7D  1632          BBC     #UCB$V_INT,UCB$W_STS(R5),10$    ; If interrupts not expected
                        0C82  1633                                                  ; then do nothing
04 64 A5    00    E0   0C82  1634          BBS     #UCB$V_TIM,UCB$W_STS(R5),10$    ; If The timeout expected
                        0C87  1635                                                  ; bit is set - no problem
    64 A5    02    AA   0C87  1636          BICW    #UCB$M_INT,UCB$W_STS(R5)        ; If interrupts expected,
                        0C8B  1637                                                  ; but timeoutnot requested
                        0C8B  1638                                                  ; avoid powerfail recover
                        0C8B  1639                                                  ; crash condition
                  05    0C8B  1640 10$:     RSB
                        0C8C  1641
```

```
                                                H  8

                OC8C  1643              .SBTTL  Control character table
                OC8C  1644 CCHRMASK:
80000008        OC8C  1645              .LONG   ^B10000000000000000000000000001000      ;ETX,ENQ
00842040        OC90  1646              .LONG   ^B00000000100001000010000001000000      ; EOT, SYN, ETB
00000000        OC94  1647              .LONG   ^B00000000000000000000000000000000      ;
00000000        OC98  1648              .LONG   ^B00000000000000000000000000000000      ;
00000000        OC9C  1649              .LONG   ^B00000000000000000000000000000000      ;
00000000        OCA0  1650              .LONG   ^B00000000000000000000000000000000      ;
00000000        OCA4  1651              .LONG   ^B00000000000000000000000000000000      ;
00000000        OCA8  1652              .LONG   ^B00000000000000000000000000000000      ;
                OCAC  1653              .SBTTL  CRC16 Polynomial Table
                OCAC  1654 POLY_TABLE:
00000000        OCAC  1655              .LONG   0
0000CC01        OCB0  1656              .LONG   52225
0000D801        OCB4  1657              .LONG   55297
00001400        OCB8  1658              .LONG   5120
0000F001        OCBC  1659              .LONG   61441
00003C00        OCC0  1660              .LONG   15360
00002800        OCC4  1661              .LONG   10240
0000E401        OCC8  1662              .LONG   58369
0000A001        OCCC  1663              .LONG   40961
00006C00        OCD0  1664              .LONG   27648
00007800        OCD4  1665              .LONG   30720
0000B401        OCD8  1666              .LONG   46081
00005000        OCDC  1667              .LONG   20480
00009C01        OCE0  1668              .LONG   39937
00008801        OCE4  1669              .LONG   34817
00004400        OCE8  1670              .LONG   17408
                OCEC  1671
                OCEC  1672 XW_END:
                OCEC  1673              .END
```

```
$$$                     =  00000020  R      02
$$OP                    =  00000002
ABEOT                   =  0000001A
ABORT_CUR                  000007BF  R      03
ABORT_NO_DSR               0000038B  R      03
ALLOC_SETUP                00000231  R      03
ASCSP                   =  00000020
AT$_UBA                 =  00000001
BADACK                  =  00000009
BAD_COMP                   000007C8  R      03
BAD_CRC                    000009B4  R      03
BAD_MODIFIER               00000092  R      03
BLK_CHECK                  00000914  R      03
CANCEL_CUR                 000007B6  R      03
CCHRMASK                   00000C8C  R      03
COMPLETE                   0000079C  R      03
COMP_GOOD                  0000078D  R      03
COMP_GOOD_AB               0000078F  R      03
COMP_GOOD_TGL              0000077F  R      03
CON_RESP                =  0000000E
CRB$L_INTD              =  00000024
CRB$L_INTD2             =  00000048
DDB$L_DDT               =  0000000C
DEV$M_AVL                  ********    X    02
DEV$M_IDV                  ********    X    02
DEV$M_ODV                  ********    X    02
DLE_STX_CRC             =  0000018C
DPT$C_LENGTH            =  00000038
DPT$C_VERSION           =  00000004
DPT$INITAB                 00000038  R      02
DPT$REINITAB               00000050  R      02
DPT$TAB                    00000000  R      02
DSR_LOST                =  0000000B
DUP_CSR_M_BYM           =  00008000
DUP_CSR_M_CRI           =  00000200
DUP_CSR_M_CTS           =  00002000
DUP_CSR_M_DSR           =  00000200
DUP_CSR_M_DTR           =  00000002
DUP_CSR_M_EOM           =  00000200
DUP_CSR_M_ETM           =  00001000
DUP_CSR_M_HDX           =  00000008
DUP_CSR_M_RDN           =  00000080
DUP_CSR_M_REN           =  00000010
DUP_CSR_M_RES           =  00000100
DUP_CSR_M_RIE           =  00000040
DUP_CSR_M_RTS           =  00000004
DUP_CSR_M_SND           =  00000010
DUP_CSR_M_SOM           =  00000100
DUP_CSR_M_SSY           =  00000100
DUP_CSR_M_STM           =  00000800
DUP_CSR_M_TDN           =  00000080
DUP_CSR_M_TIE           =  00000040
DUP_CSR_PARM            =  00000002
DUP_CSR_RDBF            =  00000002
DUP_CSR_TXCSR           =  00000004
DUP_CSR_TXDBF           =  00000006
DYN$C_CRB               =  00000005
DYN$C_DDB               =  00000006
DYN$C_DPT               =  0000001E
DYN$C_UCB               =  00000010
EBCAK0                  =  00000070
EBCAK1                  =  00000061
EBCDC1                  =  00000011
EBCDC2                  =  00000012
EBCDC3                  =  00000013
EBCDLE                  =  00000010
EBCENQ                  =  0000002D
EBCEOT                  =  00000037
EBCESC                  =  00000027
EBCETB                  =  00000026
EBCETX                  =  00000003
EBCHT                   =  00000005
EBCIGS                  =  0000001D
EBCIRS                  =  0000001E
EBCITB                  =  0000001F
EBCNAK                  =  0000003D
EBCNL                   =  00000015
EBCRVI                  =  0000007C
EBCSLH                  =  00000061
EBCSOH                  =  00000001
EBCSP                   =  00000040
EBCSTX                  =  00000002
EBCSYN                  =  00000032
EBCWACK                 =  0000006B
ERL$DEVICERR               ********    X    03
ERR_RECORD                 00000C3C  R      03
EXE$ABORTIO                ********    X    03
EXE$ALLOCBUF               ********    X    03
EXE$BUFQUOPRC              ********    X    03
EXE$FINISHIOC              ********    X    03
EXE$IOFORK                 ********    X    03
EXE$QIODRVPKT              ********    X    03
EXE$READCHK                ********    X    03
EXE$SETMODE                ********    X    03
EXE$SNDEVMSG               ********    X    03
EXE$WRITECHK               ********    X    03
EXE$ZEROPARM               ********    X    03
FUNCTAB_LEN             =  00000040
GARBLED_RESP            =  0000000C
GET_RESPONSE               00000613  R      03
GOOD_TEXT                  000009DB  R      03
IDB$C_CSR               =  00000000
IDB$L_OWNER             =  00000004
IDONE                      00000BCF  R      03
IO$K_SRRUNOUT           =  00000000
IO$V_INTCLOCK           =  0000000C
IO$V_LASTBLOCK          =  0000000A
IO$V_MAINTLOOP          =  00000009
IO$V_NOCTSWAIT          =  00000006
IO$V_NODSRWAIT          =  00000008
IO$V_SHUTDOWN           =  00000007
IO$V_SLAVLOOP           =  00000007
IO$V_STARTUP            =  00000006
IO$_READLBLK            =  00000021
```

```
IO$_READPBLK            =  0000000C
IO$_SENSEMODE           =  00000027
IO$_SETMODE             =  00000023
IO$_VIRTUAL             =  0000003F
IO$_WRITELBLK           =  00000020
IO$_WRITEPBLK           =  0000000B
IOC$CANCELIO            ********    X   03
IOC$DIAGBUFILL          ********    X   03
IOC$MNTVER              ********    X   03
IOC$REQCOM              ********    X   03
IOC$RETURN              ********    X   03
IOC$WFIKPCH             ********    X   03
IPRO_CTS                00000B9E  R     03
IPRO_DIAG               00000BAC  R     03
IPRO_DROPRTS            00000BF3  R     03
IPRO_ENDPAD             00000BE2  R     03
IPRO_IDLE               00000BCF  R     03
IPRO_SYN                00000BB1  R     03
IPRO_TEXT               00000BC2  R     03
IRP$L_MEDIA             =  00000038
IRP$L_SVAPTE            =  0000002C
IRP$M_FUNC              =  00000002
IRP$S_FCODE             =  00000006
IRP$V_DIAGBUF           =  00000007
IRP$V_FCODE             =  00000000
IRP$V_FUNC              =  00000001
IRP$W_ABCNT             =  00000040
IRP$W_BCNT              =  00000032
IRP$W_BOFF              =  00000030
IRP$W_FUNC              =  00000020
IRP$W_STS               =  0000002A
JIB$L_BYTCNT            =  00000020
MASKH                   =  00000080
MASKL                   =  00000000
MSG$_DEVOFFLIN          =  00000005
NAK                     =  00000004
NAKOUT                  =  00000007
NAK_SENT                =  00000022
NO_ERRORS               =  00000012
NXPRTXT                 000001BE  R     03
P1                      =  00000000
P2                      =  00000004
P3                      =  00000008
P4                      =  0000000C
P5                      =  00000010
P6                      =  00000014
PAD                     =  000000FF
PCB$L_JIB               =  00000080
PIPE_MARK               00000428  R     03
POLY_TABLE              00000CAC  R     03
PR$_IPL                 =  00000012
RCV_START               000008F4  R     03
RCV_START_PRO           000008E5  R     03
REC_BINARY              000002BE  R     03
REC_BIN_TMO             00000574  R     03
REC_BIN_WAIT            00000406  R     03
REC_BKTOLONG            =  00000010
REC_DISCON              =  00000019
REC_DONE                00000A20  R     03
REC_FABO_TMO            0000057E  R     03
REC_FORABORT            =  00000014
REC_GACK_TMO            00000581  R     03
REC_INI_BINARY          000003F0  R     03
REC_INI_PTPBSC          00000423  R     03
REC_PTPBSC              000002BE  R     03
REC_REP                 =  00000012
REC_REP_TMO             0000057E  R     03
REC_SNAK_TMO            0000057E  R     03
REC_TEXT_TMO            00000579  R     03
REC_TEXT_WAIT           0000045D  R     03
REC_TXTFORMAT           =  00000016
REGDUMP                 00000C14  R     03
REP                     =  00000002
REPOUT                  =  0000001D
REP_SENT                =  00000024
RIDONE                  00000AEB  R     03
RST$BINARY              =  00000001
RST$DIAG                =  0000000C
RST$DLE                 =  00000009
RST$FIRST               =  00000002
RST$IDLE                =  00000000
RST$ITB                 =  00000006
RST$ITB1                =  00000007
RST$LOOP_TEST           =  0000000D
RST$NTTX                =  00000003
RST$STOP                =  00000005
RST$STOP1               =  00000004
RST$XPRITB              =  0000000A
RST$XPRITB1             =  0000000B
RST$XPRTEXT             =  00000008
RST_BINARY              00000B5A  R     03
RST_DIAG                00000B62  R     03
RST_DLE                 00000B14  R     03
RST_FIRST               00000A86  R     03
RST_IDLE                00000AF7  R     03
RST_ITB                 00000B01  R     03
RST_ITB1                00000A5E  R     03
RST_LOOP_TEST           00000B77  R     03
RST_NTTX                00000AE3  R     03
RST_STOP                00000AC0  R     03
RST_STOP1               00000AB9  R     03
RST_XPRITB              00000B53  R     03
RST_XPRITB1             00000A57  R     03
RST_XPRTEXT             00000B08  R     03
SEND_RESP               00000555  R     03
SEN_RES_TIMOUT          =  0000001F
SND_REC_REQ             00000367  R     03
SS$_ABORT               ********    X   03
SS$_CANCEL              ********    X   03
SS$_NORMAL              ********    X   03
SS$_TOOMUCHDATA         ********    X   03
STX_CRC                 =  0000C181
SYS$GL_OPRMBX           ********    X   03
TMO_RESET               000005B3  R     03
```

```
TMO_XMT_TXTBLK                  =  00000020
TSTA$CTS                        =  00000001
TSTA$DIAG                       =  00000005
TSTA$DROPRTS                    =  00000006
TSTA$ENDPAD                     =  00000004
TSTA$IDLE                       =  00000000
TSTA$SYN                        =  00000002
TSTA$TEXT                       =  00000003
UCB$B_DIPL                      =  0000005E
UCB$B_ERTCNT                    =  00000080
UCB$B_ERTMAX                    =  00000081
UCB$B_FIPL                      =  0000000B
UCB$B_SLAVE                     =  00000090
UCB$B_XW_ERRORS                    000000AD
UCB$B_XW_ITBCNT                    000000A8
UCB$B_XW_RISTA                     000000AA
UCB$B_XW_TISTA                     000000A9
UCB$K_SIZE                      =  000000C1
UCB$L_CRB                       =  00000024
UCB$L_DEVCHAR                   =  00000038
UCB$L_DEVDEPEND                 =  00000044
UCB$L_DPC                       =  0000009C
UCB$L_FPC                       =  0000000C
UCB$L_FR3                       =  00000010
UCB$L_SVAPTE                    =  00000078
UCB$L_XW_ITBPTR                    000000A4
UCB$L_XW_SPBF                      0000009C
UCB$L_XW_SYCNT                     000000A0
UCB$M_INT                       =  00000002
UCB$M_ONLINE                    =  00000010
UCB$M_XW_AK1                    =  00000002
UCB$M_XW_NAK                    =  00000004
UCB$M_XW_PIPE                   =  00000008
UCB$M_XW_REP                    =  00000001
UCB$V_CANCEL                    =  00000003
UCB$V_INT                       =  00000001
UCB$V_TIM                       =  00000000
UCB$V_XW_AK1                    =  00000001
UCB$V_XW_NAK                    =  00000002
UCB$V_XW_PIPE                   =  00000003
UCB$V_XW_REP                    =  00000000
UCB$W_BCNT                      =  0000007E
UCB$W_DEVBUFSIZ                 =  00000042
UCB$W_DEVSTS                    =  00000068
UCB$W_FUNC                      =  0000009A
UCB$W_STS                       =  00000064
UCB$W_XW_LSTCSR                    000000AB
VEC$L_IDB                       =  00000008
VEC$L_UNITINIT                  =  00000018
XMT_BINARY                         0000009D R      03
XMT_DIAG                           00000272 R      03
XMT_INI_BINARY                     000005CD R      03
XMT_INI_DIAG                       000007CD R      03
XMT_INI_PTPBSC                     000005F8 R      03
XMT_PTPBSC                         000000CB R      03
XMT_START                          000008B9 R      03
XMT_START_PRO1                     000008A4 R      03
XMT_START_PRO2                     000008AA R      03
XMT_TEXT_WAIT                      00000603 R      03
XPRTXT                             00000115 R      03
XW$DDT                             00000000 RG     03
XW$M_BADCHAIN                   =  00000040
XW$M_CHA_DSR                    =  00000004
XW$M_CONACK                     =  00000800
XW$M_DATACK                     =  00000002
XW$M_DISCON                     =  00000008
XW$M_EOT                        =  00000001
XW$M_ETXEND                     =  00002000
XW$M_ILLMOD                     =  00000080
XW$M_NODSR                      =  00000004
XW$M_PIPE_MARK                  =  00000020
XW$M_RVI                        =  00000400
XW$M_TRABINTMO                  =  00000010
XW$M_XPR                        =  00001000
XW$S_IOMOD                      =  00000003
XW$V_CHA_FDX                    =  00000000
XW$V_CHA_XPR                    =  00000001
XW$V_IOMOD                      =  0000000D
XW_CANCELIO                        00000C67 R      03
XW_END                             00000CEC R      03
XW_FUNCTABLE                       00000038 R      03
XW_INIT                            00000C6D R      03
XW_READ                            000002AD R      03
XW_RINT                            00000A27 RG     03
XW_STARTIO                         000002F7 R      03
XW_TINT                            00000B80 RG     03
XW_WRITE                           00000078 R      03
```

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	000000C1 (193.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
$$$105_PROLOGUE	00000065 (101.)	02 (2.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC BYTE
$$$115_DRIVER	00000CEC (3308.)	03 (3.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC LONG

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	29	00:00:00.05	00:00:01.38
Command processing	108	00:00:00.52	00:00:02.52
Pass 1	510	00:00:15.08	00:00:53.75
Symbol table sort	0	00:00:02.09	00:00:10.20
Pass 2	310	00:00:03.89	00:00:13.40
Symbol table output	37	00:00:00.19	00:00:00.49
Psect synopsis output	3	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	999	00:00:21.84	00:01:21.76

The working set limit was 2250 pages.
130702 bytes (256 pages) of virtual memory were used to buffer the intermediate code.
There were 110 pages of symbol table space allocated to hold 1930 non-local and 129 local symbols.
1734 source lines were read in Pass 1, producing 25 object records in Pass 2.
38 pages of virtual memory were used to define 35 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA28:[SYS.OBJ]LIB.MLB;1	22
_255DUA28:[SYSLIB]STARLET.MLB;2	9
TOTALS (all libraries)	31

2032 GETS were required to define 31 macros.

There were no errors, warnings or information messages.

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

XWDRIVER
LIS

DTGLOBAL
LIS

DTDEFINE
LIS

DTRMAIN
LIS

DTRAST
LIS

DTPREFIX
MAR

DTSDTR

DTCOMMON
LIS

DTRECV
MAP

DTSEND
MAP

DTMACROS
MAR