C 8

```
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  EEEEEEEEEEEE     RRRRRRRRRRRR
DDD         DDD  RRRRRRRRRRRR           III        VVV         VVV  EEEEEEEEEEEE     RRRRRRRRRRRR
DDD         DDD  RRRRRRRRRRRR           III        VVV         VVV  EEEEEEEEEEEE     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
```

FILEID**XWDRIVER


```
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      WW      WW  DD      DD  RRRRRRRR        II      VV      VV  EEEEEEEE    RRRRRRRR
    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
```

XWDR
V04-

Table of contents

```
0000          .SBTTL  MODULE - XWDRIVER
0000          .TITLE  XWDRIVER - BSC DUP11 Device Driver
0000          .IDENT  'V04-000'
0000
0000  ;
0000  ;****************************************************************************
0000  ;*                                                                          *
0000  ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
0000  ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
0000  ;*  ALL RIGHTS RESERVED.                                                    *
0000  ;*                                                                          *
0000  ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
0000  ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000  ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
0000  ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
0000  ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
0000  ;*  TRANSFERRED.                                                            *
0000  ;*                                                                          *
0000  ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
0000  ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
0000  ;*  CORPORATION.                                                            *
0000  ;*                                                                          *
0000  ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
0000  ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
0000  ;*                                                                          *
0000  ;*                                                                          *
0000  ;****************************************************************************
0000  ;
0000  ; Modified by:
0000  ;
0000  ;       V03-003 KDM0002         Kathleen D. Morse       28-Jun-1982
0000  ;               Added $DCDEF, $DYNDEF, $PRDEF, and $SSDEF.
0000  ;
0000  ;       V03-002 MMD0002         Meg Dumont,     4-Jun-1982  11:16
0000  ;               Fix UCB$W_INT to be UCB$M_INT.
0000  ;
0000  ;       V03-001 MMD0001         Meg Dumont,     19-May-1982  14:46
0000  ;               Add known fixes from XJDRIVER to this driver
0000  ;
0000  ;
0000  ;
0000  ; MACRO LIBRARY CALLS
0000  ;
0000
0000          $XWDEF
0000          $EBCDEF
0000          $CRBDEF                         ;DEFINE CRB OFFSETS
0000          $DCDEF                          ;DEFINE DATA TYPES
0000          $DDBDEF                         ;DEFINE DDB OFFSETS
0000          $DPTDEF                         ;DEFINE DPT OFFSETS
0000          $DYNDEF                         ;DEFINE DYNAMIC DATA STRUCTURE TYPES
0000          $IDBDEF                         ;DEFINE IDB OFFSETS
0000          $IODEF                          ;DEFINE I/O FUNCTION CODES
0000          $IRPDEF                         ;DEFINE IRP OFFSETS
0000          $JIBDEF
0000          $MSGDEF                         ;DEFINE SYSTEM MESSAGE TYPES
0000          $PCBDEF                         ;DEFINE PCB OFFSETS
```

```
                   0000          $PRDEF                                  ;DEFINE PROCESSOR REGISTERS
                   0000          $UCBDEF                                 ;DEFINE UCB OFFSETS
                   0000          $VADEF                                  ;DEFINE VIRTUAL ADDRESS FIELDS
                   0000          $VECDEF                                 ;DEFINE VEC OFFSETS
                   0000
                   0000  ;
                   0000  ; LOCAL SYMBOLS
                   0000  ;
                   0000  ; ARGUMENT LIST OFFSET DEFINITIONS
                   0000  ;
                   0000
          00000000 0000  P1=0                                            ;FIRST FUNCTION DEPENDENT PARAMETER
          00000004 0000  P2=4                                            ;SECOND FUNCTION DEPENDENT PARAMETER
          00000008 0000  P3=8                                            ;THIRD FUNCTION DEPENDENT PARAMETER
          0000000C 0000  P4=12                                           ;FOURTH FUNCTION DEPENDENT PARAMETER
          00000010 0000  P5=16                                           ;FIFTH FUNCTION DEPENDENT PARAMETER
          00000014 0000  P6=20                                           ;SIZTH FUNCTION DEPENDENT PARAMETER
                   0000
                   0000  ;
                   0000  ; Error recording macro
                   0000  ;
                   0000          .MACRO  PROTLO  ERRNO
                   0000          MOVZBL  #ERRNO,-(SP)                    ; Put the error on the stack
                   0000          BSBW    ERR_RECORD                      ; Call the error recorder
                   0000          .ENDM   PROTLO
                   0000
          00000002 0000          REP=1*2                                 ; Out of phase ACK after REP
          00000004 0000          NAK=2*2                                 ; RECEIVED NAKS
          00000007 0000          NAKOUT=3*2!1                            ; Fatal nak retry
          00000009 0000          BADACK=4*2!1                            ; Out-of-phase ACK other than as
                   0000                                                  ;  a response to a ENQ
          0000000B 0000          DSR_LOST=5*2!1                          ; DSR lost durring a buffer
          0000000C 0000          GARBLED_RESP=6*2                        ; Unrecognizable response to text
          0000000E 0000          CON_RESP=7*2                            ; Conversational response received
          00000010 0000          REC_BKTOLONG=8*2                        ; Receive block to long for buffer
          00000012 0000          REC_REP=9*2                             ; Reply lost (ENQ received)
          00000014 0000          REC_FORABORT=10*2                       ; Forward abort received (ENQ in text)
          00000016 0000          REC_TXTFORMAT=11*2                      ; Block format undecipherable
          00000019 0000          REC_DISCON=12*2!1                       ; Disconnect received
          0000001A 0000          ABEOT=13*2                              ; Abnormal EOT (durring xmt text)
          0000001D 0000          REPOUT=14*2!1                           ; Transmit retry threshold
          0000001F 0000          SEN_RES_TIMOUT=15*2!1                   ; Timout durring response
          00000020 0000          TMO_XMT_TXTBLK=16*2                     ; Timout sending a text block
          00000022 0000          NAK_SENT=17*2                           ; Nak transmitted
          00000024 0000          REP_SENT=18*2                           ; Rep (ENQ) sent
                   0000
          00000012 0000          NO_ERRORS=18                            ; Highest error number+3 rounded up
                   0000          .LIST   MEB
                   0000  ;
                   0000  ;
                   0000  ;Define Device Dependent Unit Control Block Offsets
                   0000  ;
                   0000          $DEFINI UCB
          0000009C 0000                  .=UCB$L_DPC
                   009C          $DEF    UCB$L_XW_SPBF   .BLKL 1         ; Short protocol buffer
                   00A0          $DEF    UCB$L_XW_SYCNT  .BLKL 1         ; Sync count (also offline ct)
                   00A4          $DEF    UCB$L_XW_ITBPTR .BLKL 1         ; Internal CRC pointer
                   00A8          $DEF    UCB$B_XW_ITBCNT .BLKB 1         ; Internal CRC count
```

```
          00A9          $DEF    UCB$B_XW_TISTA  .BLKB 1          ; Interrupt state
          00AA          $DEF    UCB$B_XW_RISTA  .BLKB 1          ; Receive interrupt state
          00AB          $DEF    UCB$W_XW_LSTCSR .BLKW 1          ; CSR at end of last transfer
          00AD          $DEF    UCB$B_XW_ERRORS .BLKB NO_ERRORS+2 ; Error talleys + 2 bytes
          00C1                                                   ;   for current err & 0 spare
000000C1  00C1  UCB$K_SIZE=.
          00C1
          00C1          $DEFEND UCB
          0000
          0000  ;
          0000  ; Device specific flags definition (for UCB$W_DEVSTS)
          0000  ;
00000000  0000          UCB$V_XW_REP=0                           ; Retry state
00000001  0000          UCB$M_XW_REP=^01                         ;...
          0000
00000001  0000          UCB$V_XW_AK1=1                           ; ACK1 next
00000002  0000          UCB$M_XW_AK1=2                           ; ACK1 next
          0000
00000002  0000          UCB$V_XW_NAK=2                           ; NAK sent last flag
00000004  0000          UCB$M_XW_NAK=^04                         ; ...
          0000                                                   ;  to a long word
          0000
00000008  0000          UCB$M_XW_PIPE=^o10                       ; Cancel pipeline marker
00000003  0000          UCB$V_XW_PIPE=3
          0000  ;
          0000  ; DUP-11 CSR Offset And Bit Definitions
          0000  ;
00000002  0000          DUP_CSR_RDBF=2                           ; Receiver data buffer
00000002  0000          DUP_CSR_PARM=2                           ; Parameter register
00000004  0000          DUP_CSR_TXCSR=4                          ; Transmitter command register
00000006  0000          DUP_CSR_TXDBF=6                          ; Transmitter data buffer
00000002  0000          DUP_CSR_M_DTR=^02                        ; Data terminal ready
00000004  0000          DUP_CSR_M_RTS=^04                        ; Request to send
00000010  0000          DUP_CSR_M_REN=^020                       ; Receiver enable
00000040  0000          DUP_CSR_M_RIE=^0100                      ; Receiver interrupt enable
00000080  0000          DUP_CSR_M_RDN=^0200                      ; Receiver done
00000100  0000          DUP_CSR_M_SSY=^0400                      ; Strip SYN
00000200  0000          DUP_CSR_M_DSR=^01000                     ; Data set ready
00002000  0000          DUP_CSR_M_CTS=^020000                    ; Clear to send
00008000  0000          DUP_CSR_M_BYM=^0100000                   ; Byte mode
00000010  0000          DUP_CSR_M_SND=^020                       ; Send
00000040  0000          DUP_CSR_M_TIE=^0100                      ; Transmit interrupt enable
00000080  0000          DUP_CSR_M_TDN=^0200                      ; Transmit done
00000100  0000          DUP_CSR_M_RES=^0400                      ; Reset
00000100  0000          DUP_CSR_M_SOM=^0400                      ; Transmit start of message
00000200  0000          DUP_CSR_M_EOM=^01000                     ; Transmit end of message
00000200  0000          DUP_CSR_M_CRI=^01000                     ; CRC inhibit bit
00000008  0000          DUP_CSR_M_HDX=^010                       ; Half duplex bit
00001000  0000          DUP_CSR_M_ETM=^010000                    ; Maintenance clock bit
00000800  0000          DUP_CSR_M_STM=^04000                     ; System (internal)loop mode
          0000  ;
          0000  ; Transmit interrupt state definition
          0000  ;
00000000  0000          TSTA$IDLE=0                              ; No interrupt expected
00000001  0000          TSTA$CTS=1                               ; Wait forCTS
00000002  0000          TSTA$SYN=2                               ; Transmit SYN
00000003  0000          TSTA$TEXT=3                              ; Transmit text
```

```
00000004  0000        TSTA$ENDPAD=4                     ; Transmit trailing pad
00000005  0000        TSTA$DIAG=5                       ; Diagnostic (no CTS wait) state
00000006  0000        TSTA$DROPRTS=6                    ; Drop RTS state
          0000
          0000   ;
          0000   ; Receive interrupt state definition
          0000   ;
00000000  0000        RST$IDLE=0                        ; No interrupts expected
00000001  0000        RST$BINARY=1                      ; Read until byte count runout
00000002  0000        RST$FIRST=2                       ; Starting state
00000003  0000        RST$NTTX=3                        ; Non-transparent text state
00000004  0000        RST$STOP1=4                       ; Last CRC byte
00000005  0000        RST$STOP=5                        ; Stop after next byte
00000006  0000        RST$ITB=6                         ; First byte of internal CRC next
00000007  0000        RST$ITB1=7                        ; Second byte of internal CRC next
00000008  0000        RST$XPRTEXT=8                     ; Transparent text state
00000009  0000        RST$DLE=9                         ; Control character or stuffed DLE next
0000000A  0000        RST$XPRITB=10                     ; First byte of internal CRC next
0000000B  0000        RST$XPRITB1=11                    ; Second byte of internal CRC next
0000000C  0000        RST$DIAG=12                       ; Diagnostic receive interrupt state
0000000D  0000        RST$LOOP_TEST=13                  ; Diagnostic slave loop state
          0000   ;
          0000   ;DRIVER PROLOG TABLE
          0000   ;
          0000        DPTAB   -                         ; Define driver prolog table
          0000                END=XW_END,-              ; End of driver
          0000                ADAPTER=UBA,-             ; Adapter type
          0000                UCBSIZE=UCB$K_SIZE,-      ; UCB size
          0000                NAME=XWDRIVER             ; Driver name
          0038        DPT_STORE       INIT              ; Control block init values
          0038        DPT_STORE       UCB,-
          0038                        UCB$B_ERTMAX,B,8 ; Retry count
          003C        DPT_STORE       UCB,-             ; Fork IPL
          003C                        UCB$B_FIPL,B,8
          0040        DPT_STORE       UCB,UCB$L_DEVCHAR,L,- ; Device characteristics bits
          0040                        <DEV$M_AVL-       ; Device avalable
          0040                        !DEV$M_IDV-       ; Input device
          0040                        !DEV$M_ODV>       ; Output device
          0047        DPT_STORE       UCB,UCB$W_DEVBUFSIZ,- ; Device buffer size
          0047                        W,520
          004C        DPT_STORE       UCB,UCB$B_DIPL,B,21 ; Device IPL
          0050        DPT_STORE       REINIT            ; Fields to reinit. at driver roload
          0050        DPT_STORE       DDB,DDB$L_DDT,- ; Device dispatch table
          0050                        D,XW$DDT
          0055        DPT_STORE       CRB,CRB$L_INTD+4,- ; Receive interrupt entry point
          0055                        D,XW_RINT
          005A        DPT_STORE       CRB,CRB$L_INTD2+4,- ; Transmit interrupt entry point
          005A                        D,XW_TINT
          005F        DPT_STORE       CRB,CRB$L_INTD+VEC$L_UNITINIT,- ; Unit initalization
          005F                        D,XW_INIT
          0064        DPT_STORE       END
          0000   ;
          0000   ; Driver Dispatch Table
          0000   ;
          0000        DDTAB   XW,-                      ; Driver dispatch table
          0000                XW_STARTIO,-              ; Start IO entry point
          0000                0,-                       ; Unexpected interrupt
```

```
0000                          XW_FUNCTABLE,-          ; FDT lable
0000                          XW_CANCELIO,-           ; Cancel IO label
0000                          REGDUMP,-               ; Register dump routine
0000                          170,-                   ; Size of diagnostic buffer
0000                          170                     ; Size of errlog buffer
0038    ;
0038    ;FDT tables for BSC DUP-11 driver
0038    ;
0038    XW_FUNCTABLE:
0038            FUNCTAB ,-                      ; Legal functions
0038                    <READLBLK,-             ; Read logical block
0038                    READPBLK,-              ; Read physical block
0038                    WRITELBLK,-             ; Write logical block
0038                    WRITEPBLK,-             ; Write physical block
0038                    SETMODE,-               ; Set mode
0038                    SENSEMODE,-
0038                    >
0040    ;
0040            FUNCTAB ,-                      ; Legal functions
0040                    <READLBLK,-             ; Read logical block
0040                    READPBLK,-              ; Read  block
0040                    WRITELBLK,-             ; Write logical block
0040                    WRITEPBLK,-             ; Write physical block
0040                    SETMODE,-               ; Set mode
0040                    SENSEMODE,-
0040                    >
0048    ;
0048            FUNCTAB XW_WRITE,-              ; Write entry
0048                    <WRITEPBLK-             ; Write physical block
0048                    WRITELBLK-              ; Write logical block
0048                    >
0054    ;
0054            FUNCTAB XW_READ,-               ; Read processing entry
0054                    <READPBLK-              ; Read physical block
0054                    READLBLK-               ; Read logical block
0054                    >
0060    ;
0060            FUNCTAB +EXE$SETMODE,-  ; Set mode
0060                    SETMODE
006C    ;
006C            FUNCTAB +EXE$ZEROPARM,-
006C                    SENSEMODE
```

```
                        0078          .SBTTL  XWDRIVER Write FDT Routines
                        0078  ;+
                        0078  ; XW_WRITE - WRITE FUNCTION PROCESSING
                        0078  ;
                        0078  ; THIS ROUTINE IS CALLED FROM THE FUNCTION DECISION TABLE DISPATCHER TO PROCESS
                        0078  ; A WRITE LOGICAL OR WRITE PHYSICAL FUNCTION TO A DUP-11.
                        0078  ;
                        0078  ; INPUTS:
                        0078  ;
                        0078  ;       R0 = SCRATCH.
                        0078  ;       R1 = SCRATCH.
                        0078  ;       R2 = SCRATCH.
                        0078  ;       R3 = ADDRESS OF I/O REQUEST PACKET.
                        0078  ;       R4 = CURRENT PROCESS PCB ADDRESS.
                        0078  ;       R5 = ASSIGNED DEVICE UCB ADDRESS.
                        0078  ;       R6 = ADDRESS OF CCB.
                        0078  ;       R7 = I/O FUNCTION CODE.
                        0078  ;       R8 = FUNCTION DECISION TABLE DISPATCH ADDRESS.
                        0078  ;       R9 = SCRATCH.
                        0078  ;       R10 = SCRATCH.
                        0078  ;       R11 = SCRATCH.
                        0078  ;       AP = ADDRESS OF FIRST FUNCTION DEPENDENT PARAMETER.
                        0078  ;
                        0078  ; OUTPUTS:
                        0078  ;
                        0078  ;       THE FUNCTION PARAMETERS ARE CHECKED AND A BUFFER IS ALLOCATED FOR THE
                        0078  ;       DUP-11 DRIVER TO BUILD A MESSAGE INTO.
                        0078  ;-
                        0078
                        0078          .ENABL  LSB
                        0078  XW_WRITE:
       03  0D      EF   0078          EXTZV   #XW$V_IOMOD,#XW$S_IOMOD,- ; Extract the IO modifier field
   50   20 A3           007B                  IRP$W_FUNC(R3),R0
                        007E          CASE    R0,-                    ; Dispatch on the modifier
                        007E                  <XMT_BINARY,-           ; Send until count runout
                        007E                  XMT_PTPBSC,-            ; Add point to point BSC control
                        007E                  BAD_MODIFIER,-
                        007E                  BAD_MODIFIER,-
                        007E                  BAD_MODIFIER,-
                        007E                  BAD_MODIFIER,-
                        007E                  BAD_MODIFIER,-
                        007E                  XMT_DIAG,-              ; Full duplex diagnostic processing
                        007E                  >                       ; Last modifier
                        0092  ;
                        0092  ; Modifier was out of range so abort the request
                        0092  ;
                        0092  BAD_MODIFIER:
 51   0080 8F      3C   0092          MOVZWL  #XW$M_ILLMOD,R1         ; Get device dependant status
   00000000'GF     17   0097          JMP     G^EXE$ABORTIO           ; Abort the request
                        009D  ;
                        009D  ; Send the users buffer with no processing
                        009D  ;
                        009D  XMT_BINARY:
       50   6C     D0   009D          MOVL    P1(AP),R0               ; Get the user buffer pointer
    51   04 AC     D0   00A0          MOVL    P2(AP),R1               ; Get the user count
            2E     13   00A4          BEQL    10$                     ; Complete now if zero count
   00000000'GF     16   00A6          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   ;
                                 00CB   ; Format the users list of records into a bisync block
                                 00CB   ;
                                 00CB   XMT_PTPBSC:
            50    6C    D0       00CB           MOVL    P1(AP),R0                       ; Get user buffer pointer
         51    04 AC    D0       00CE           MOVL    P2(AP),R1                       ; Get user buffer count
                  0D    12       00D2           BNEQ    20$                             ; BR if not zero count
       50    0000'8F    3C       00D4   10$:    MOVZWL  #SS$_NORMAL,R0                  ; Get normal completion status
                  51    D4       00D9           CLRL    R1                              ; No device dependant status
         00000000'GF    17       00DB           JMP     G^EXE$FINISHIOC                 ; Complete the IO request
         00000000'GF    16       00E1   20$:    JSB     G^EXE$WRITECHK                  ; Check the user buffer descripter
                                 00E7   ;
                                 00E7   ; Check the buffer chain
                                 00E7   ;
      59    50    51    C1       00E7           ADDL3   R1,R0,R9                        ; Point to the end of the user buffer
                  52    D4       00EB           CLRL    R2                              ; Set zero record count
            59    02    C2       00ED           SUBL2   #2,R9                           ; Point before last count
            58    80    3C       00F0   30$:    MOVZWL  (R0)+,R8                        ; Get the the count
       58    FFFF 8F    B1       00F3           CMPW    #-1,R8                          ; Last count ?
                  13    13       00F8           BEQL    50$                             ; yes - branch out
                  52    D6       00FA           INCL    R2                              ; Step record count
 FFEE 50    58    59    F1       00FC           ACBL    R9,R8,R0,30$                    ; BR if next count is in buffer
       51    0040 8F    3C       0102           MOVZWL  #XW$M_BADCHAIN,R1               ; Get bad buffer status
         00000000'GF    17       0107   40$:    JMP     G^EXE$ABORTIO                   ; Abort the operation
                                 010D
                  01    E0       010D   50$:    BBS     #XW$V_CHA_XPR,-                 ; Is transparency called for
            03 44 A5             010F                   UCB$L_DEVDEPEND(R5),XPRTXT
                00A9    31       0112           BRW     NXPRTXT                         ; Go to nontransparent code
                                 0115           .DSABL  LSB
                                 0115   ; Build a transparent text block
                                 0115   ;
                                 0115   XPRTXT:
   51    04 AC    02    C5       0115           MULL3   #2,P2(AP),R1                    ; Get the count
            52    03    C4       011A           MULL2   #3,R2                           ; Each record needs 7 extra bytes
            51    52    C0       011D           ADDL    R2,R1                           ; Get the total size
                010E    30       0120           BSBW    ALLOC_SETUP                     ; Get a system buffer
                                 0123   ;               R9 -From data pointer
                                 0123   ;               R10 -To data pointer
                                 0123   ;               R11 -Record count
             0138 8F    BB       0123           PUSHR   #^M<R3,R4,R5,R8>
         58    20 A3    3C       0127           MOVZWL  IRP$W_FUNC(R3),R8               ; Save the function
                  7E    D4       012B           CLRL    -(SP)                           ; No starting CRC
         5A    2C B3    D0       012D           MOVL    @IRP$L_SVAPTE(R3),R10           ; Set the to pointer
            59    6C    D0       0131           MOVL    P1(AP),R9                       ; Set the from pointer
            5B    89    3C       0134           MOVZWL  (R9)+,R11                       ; Get the first record,s count
       8A    0210 8F    B0       0137   10$:    MOVW    #EBCDLE!<EBCSTX*256>,(R10)+ ; All records start with DLE STX
69   5B    6E   00000CAC'EF    0B       013C           CRC     POLY_TABLE,(SP),R11,(R9) ; Get the CRC on the Record
            6E    50    B0       0145           MOVW    R0,(SP)                         ; Save the CRC until end of record
      69    5B    10    3A       0148   20$:    LOCC    #EBCDLE,R11,(R9)                ; Find any DLE in the data
                  07    13       014C           BEQL    30$                             ; BR if end of record and no CRC
                  51    D6       014E           INCL    R1                              ; Include the DLE in the "found" data
            02    50    F5       0150           SOBGTR  R0,30$                          ; ...
                  50    D7       0153           DECL    R0                              ; Flag a DLE ended the record
            5B    50    D0       0155   30$:    MOVL    R0,R11                          ; Set remaining data count
      50    51    59    C3       0158           SUBL3   R9,R1,R0                        ; Get scaned data count
      6A    69    50    28       015C           MOVC    R0,(R9),(R10)                   ; And move the data
            59    51    D0       0160           MOVL    R1,R9                           ; Set new from pointer
```

```
                  5A    53   D0 0163            MOVL    R3,R10                       ; Set new to pointer
                  8A    10   90 0166            MOVB    #EBCDLE,(R10)+               ; Stuff the DLE or end with DLE
                        5B   D5 0169            TSTL    R11                          ; Is this the end of block
                        DB   14 016B            BGTR    20$                          ; NO -find more DLE,s
                        03   13 016D            BEQL    40$                          ; If eql, was the end of the block
                  8A    10   90 016F            MOVB    #EBCDLE,(R10)+               ; Else block ended with DLE, so stuff
                                0172                                                 ; an extra DLE in for control character
                  5B    89   B0 0172    40$:    MOVW    (R9)+,R11                    ; Get the next record count
            5B    FFFF  8F   B1 0175            CMPW    #-1,R11                      ; Is it the sentenal
                        11   13 017A            BEQL    50$                          ; BR if it is
                  6A    1F   90 017C            MOVB    #EBCITB,(R10)                ; Store a record seperator
                        2D   10 017F            BSB     70$                          ; Add the trcord end char. to CRC
            8A    3232  8F   B0 0181            MOVW    #EBCSYN!<EBCSYN*256>,(R10)+ ; Followed by SYN SYN sin
            6E    018C' 8F   3C 0186            MOVZWL  #DLE_STX_CRC,(SP)            ; Assume the next partial CRC
                        AA   11 018B            BRB     10$                          ; Get another record
                  6A    26   90 018D    50$:    MOVB    #EBCETB,(R10)                ; Assume not last block
                        0A   E1 0190            BBC     #IO$V_LASTBLOCK,-            ; Is it the last block
                     03 58      0192                    R8,60$
                  6A    03   90 0194            MOVB    #EBCETX,(R10)                ; Set end of last block
                        15   10 0197    60$:    BSB     70$                          ; Include the block end chsr. in CRC
                  5E    04   C0 0199            ADDL    #4,SP                        ; Remove the partial CRC
                  0138  8F   BA 019C            POPR    #^M<R3,R4,R5,R8>
            5A    2C    B3   C2 01A0            SUBL2   @IRP$L_SVAPTE(R3),R10        ; Get the block count
            32    A3    5A   B0 01A4            MOVW    R10,IRP$W_BCNT(R3)           ; Save the byte count
            00000000'   GF   17 01A8            JMP     G^EXE$QIODRVPKT              ; Post the IRP to the driver
                                01AE
6A   01   04 AE   00000CAC' EF   0B 01AE    70$:    CRC     POLY_TABLE,4(SP),#1,(R10) ; include the last chara. in the CRC
                        5A   D6 01B8            INCL    R10                          ; Step by the last byte
                  8A    50   B0 01BA            MOVW    R0,(R10)+                    ; And store the last byte
                        05      01BD            RSB
                                01BE    ;
                                01BE    ; Nontransparent record blocking and CRC generation
                                01BE    ;
                                01BE    NXPRTXT:
                  52    03   C4 01BE            MULL2   #3,R2                        ; Need count space +3 byte for each rec
                  52    04   C0 01C1            ADDL    #4,R2                        ; Room for STX and luck
      51    52    04    AC   A1 01C4            ADDW3   P2(AP),R2,R1                 ; get eventual size
                  0065       30 01C9            BSBW    ALLOC_SETUP                  ; allocate and set up a system buffer
                  0078  8F   BB 01CC            PUSHR   #^M<R3,R4,R5,R6>             ; save the FDT call reg. setup
            56    20    A3   3C 01D0            MOVZWL  IRP$W_FUNC(R3),R6            ; Get the IO function & mod.
            53    2C    B3   D0 01D4            MOVL    @IRP$L_SVAPTE(R3),R3         ; Get the system buffer pointer
                  83    02   90 01D8            MOVB    #EBCSTX,(R3)+                ; Start the block with a stx
                  5B    6C   D0 01DB            MOVL    P1(AP),R11                   ; Get the user buffer pointer
                  52    8B   3C 01DE    10$:    MOVZWL  (R11)+,R2                    ; And the first record count
            52    FFFF  8F   B1 01E1            CMPW    #-1,R2                       ; Is it the sentenal
                        33   13 01E6            BEQL    40$                          ; BR if it is
                  59    52   7D 01E8            MOVQ    R2,R9                        ; Save the start of rec. ptr. &cnt.
            63    6B    52   28 01EB            MOVC    R2,(R11),(R3)                ; Move the record into the sys. buf.
                                01EF            ; R1- start of next record, R3- end of this record
                  5B    51   D0 01EF            MOVL    R1,R11                       ; Save a pointer to next record
                        59   D6 01F2            INCL    R9                           ; Include the record end character
                                01F4                                                 ;  (not yet added) in the count
                  63    1F   90 01F4            MOVB    #EBCITB,(R3)                 ; Add an ITB to the block
            6B    FFFF  8F   B1 01F7            CMPW    #-1,(R11)                    ; Is this to be end of block
                        0A   12 01FC            BNEQ    30$                          ; BR- if it is ITB CRC CRC
                  63    26   90 01FE            MOVB    #EBCETB,(R3)                 ; Assume intermediate block
                        0A   E1 0201            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  ;
                                  0272  ; Diagnostic full duplex read/write FDT processing
                                  0272  ;
                                  0272  XMT_DIAG:
          50     08 AC     7D     0272          MOVQ    P3(AP),R0                       ; Check the read buffer
        00000000'GF        16     0276          JSB     G^EXE$READCHK                   ; ...
          50        6C     7D     027C          MOVQ    P1(AP),R0                       ; Check the write buffer
        00000000'GF        16     027F          JSB     G^EXE$WRITECHK                  ; ...
   51  0C AC     04 AC     C1     0285          ADDL3   P2(AP),P4(AP),R1                ; Get the total buffer count
                 A3 AF     16     028B          JSB     ALLOC_SETUP                     ; Allocate a system buffer
          62     08 AC     D0     028E          MOVL    P3(AP),(R2)                     ; Set receive buffer addr. for
                                  0292                                                  ;  IO completion code
       32 A3     0C AC     3C     0292          MOVZWL  P4(AP),IRP$W_BCNT(R3)           ; Set the rec. count
   50  0C AC     2C B3     C1     0297          ADDL3   @IRP$L_SVAPTE(R3),P4(AP),R0 ; Get the transmit pointer
                    38     BB     029D          PUSHR   #^M<R3,R4,R5>
   60  00 BC     04 AC     28     029F          MOVC    P2(AP),@P1(AP),(R0)             ; Move the transmit data
                    38     BA     02A5          POPR    #^M<R3,R4,R5>
        00000000'GF        17     02A7          JMP     G^EXE$QIODRVPKT                 ; Queue the packet to the driver
```

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

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

```
                          02F7           .SBTTL  Start IO on the DUP-11
                          02F7  ;+
                          02F7  ; XW_STARTIO - START I/O OPERATION ON THE DUP-11
                          02F7  ;
                          02F7  ; THIS ROUTINE IS ENTERED WHEN THE ASSOCIATED UNIT IS IDLE AND A PACKET IS
                          02F7  ; AVAILABLE FOR PROCESSING.
                          02F7  ;
                          02F7  ; INPUTS:
                          02F7  ;
                          02F7  ;        R3 = ADDRESS OF I/O REQUEST PACKET.
                          02F7  ;        R5 = ADDRESS OF DEVICE UNIT UCB.
                          02F7  ;
                          02F7  ; OUTPUTS:
                          02F7  ;
                          02F7  ; LOTSATEXT
                          02F7  ;
                          02F7  ;-
                          02F7
                          02F7  XW_STARTIO:
                          02F7          .ENABL  LSB
       54   24 A5   D0    02F7          MOVL    UCB$L_CRB(R5),R4            ; Get the DUP csr pointer
       54   2C B4   D0    02FB          MOVL    @CRB$L_INTD+VEC$L_IDB(R4),R4 ; ...
               00   ED    02FF          CMPZV   #IRP$V_FCODE,-              ;Is this a set function
    20 A3      06         0301                  #IRP$S_FCODE,IRP$W_FUNC(R3),-
               23         0304                  #IO$_SETMODE
               4D   12    0305          BNEQ    50$                         ; BR if not set
                          0307  ;
                          0307  ; Set mode
                          0307  ;
 42 A5      3A A3   B0    0307          MOVW    IRP$L_MEDIA+2(R3),UCB$W_DEVBUFSIZ(R5) ; Set block size
 44 A5      3C A3   D0    030C          MOVL    IRP$L_MEDIA+4(R3),UCB$L_DEVDEPEND(R5) ; Save characteristics
               07   E1    0311          BBC     #IO$V_SHUTDOWN,-            ; Is dorp line requested
         08 20 A3         0313                  IRP$W_FUNC(R3),10$
 04 A4    0100 8F   A8    0316          BISW    #DUP_CSR_M_RES,DUP_CSR_TXCSR(R4) ;Reset the DUP
               33   11    031C          BRB     40$                         ; Complete the request
                          031E
               06   E1    031E  10$:    BBC     #IO$V_STARTUP,-             ; Is enable line requested
         2E 20 A3         0320                  IRP$W_FUNC(R3),40$
         64    02   A8    0323          BISW    #DUP_CSR_M_DTR,(R4)         ; Assert DTR
    64    0200 8F   B3    0326  20$:    BITW    #DUP_CSR_M_DSR,(R4)         ; Is DSR up ?
               24   12    032B          BNEQ    40$                         ; BR if it is
               08   E0    032D          BBS     #IO$V_NODSRWAIT,-           ; Is a non wait diagnostic function here
         1F 20 A3         032F                  IRP$W_FUNC(R3),40$
             0097   30    0332          BSBW    110$                        ; Send a line off message
                          0335          DSBINT
                          033B          WFIKPCH 30$,#2
                          0345  30$:    SETIPL  UCB$B_FIPL(R5)
 D8 64 A5      03   E1    0349          BBC     #UCB$V_CANCEL,UCB$W_STS(R5),20$ ; Cancel requested ?
             0465   31    034E          BRW     CANCEL_CUR                  ; Return this packet
             0439   31    0351  40$:    BRW     COMP_GOOD                   ; Complete the request
                          0354
       06      00   ED    0354  50$:    CMPZV   #IRP$V_FCODE,#IRP$S_FCODE,-     ; Is it a sense mode ?
    27      20 A3         0357                  IRP$W_FUNC(R3),#IO$_SENSEMODE
               5C   13    035A          BEQL    90$                             ; BR if it is
                          035C  ;
                          035C  ; Send/receive request
                          035C  ;
```

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

```
 05  03EF    684 130$:    RSB
```

XWD
V04

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

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

```
            2C   13 04CF          BEQL    50$                     ; BR if it is
                    04D1    ;
                    04D1    ; TTD or Forward Abort
                    04D1    ;
       51   3D   9A 04D1          MOVZBL  #EBCNAK,R1              ; Get a NAK
          03CD   30 04D4          BSBW    XMT_START_PRO1          ; And send it
                    04D7          WFIKPCH REC_FABO_TMO,#4
                    04E1          IOFORK                          ; Create a fork process
   50   2C B3    D0 04E7          MOVL    @IRP$L_SVAPTE(R3),R0    ; Get the old buffer pointer
            50   D6 04EB          INCL    R0                      ; Point to first received byte
   60   2D02 8F  B1 04ED          CMPW    #EBCSTX!<EBCENQ*256>,(R0) ; Was it a TTD
            06   13 04F2          BEQL    40$                     ; Yes- only log aborts
                    04F4          PROTLO  REC_FORABORT
          FE6A   31 04FA  40$:    BRW     SND_REC_REQ             ; Reassign the buffer
         3C A3   D4 04FD  50$:    CLRL    IRP$L_MEDIA+4(R3)       ; Start with no status
   FD A140  03   91 0500          CMPB    #EBCETX,-3(R1)[R0]      ; Is it an ETX block
            06   12 0505          BNEQ    60$                     ; BR if not
       2000 8F   3C 0507          MOVZWL  #XW$M_ETXEND,-          ; Set the completion status
         3C A3      050B                  IRP$L_MEDIA+4(R3)
          0404   31 050D  60$:    BRW     BLK_CHECK               ; Go do the CRC check
                    0510
     52   FF A1  B0 0510  70$:    MOVW    -1(R1),R2               ; Get the first two bytes from the buff.
            51   D6 0514          INCL    R1                      ; Step the buffer pointer
            50   D7 0516          DECL    R0                      ; And count
   52   3710 8F  B1 0518          CMPW    #EBCDLE!<EBCEOT*256>,R2 ; Is it a disconnect
            0C   12 051D          BNEQ    80$                     ; Br in not
                    051F          PROTLO  REC_DISCON
       51   08   3C 0525          MOVZWL  #XW$M_DISCON,R1         ; Get a disconnect status
          0294   31 0528          BRW     ABORT_CUR               ; Return the packet to the user
                    052B
   52   0210 8F  B1 052B  80$:    CMPW    #EBCDLE!<EBCSTX*256>,R2 ; Is it xpr text
            09   13 0530          BEQL    90$                     ; BR if it is
                    0532  85$:    PROTLO  REC_TXTFORMAT
          FE2C   31 0538          BRW     SND_REC_REQ             ; Reassign the buffer
                    053B
   52   1000 8F  3C 053B  90$:    MOVZWL  #XW$M_XPR,R2            ; Get transparent status
   FD A140  03   91 0540          CMPB    #EBCETX,-3(R1)[R0]      ; Is it the last block
            07   12 0545          BNEQ    100$                    ; Br if not
52   00002000 8F C8 0547          BISL    #XW$M_ETXEND,R2         ; Save the status for the completion
    3C A3   52   D0 054E  100$:   MOVL    R2,IRP$L_MEDIA+4(R3)    ; ...
          03BF   31 0552          BRW     BLK_CHECK               ; Check transparent text block
                    0555          .DSABL  LSB
                    0555    ;
                    0555    ; Set up response utility
                    0555    ;
                    0555    SEND_RESP:
            02   E1 0555          BBC     #UCB$V_XW_NAK,-         ; Is a NAK called for
       06 68 A5     0557                  UCB$W_DEVSTS(R5),10$
       51   3D   9A 055A          MOVZBL  #EBCNAK,R1              ; Get a NAK message
          0344   31 055D          BRW     XMT_START_PRO1          ; Start the device and return to caller
            01   E1 0560  10$:    BBC     #UCB$V_XW_AK1,-         ; BR if an AK1 is needed
       07 68 A5     0562                  UCB$W_DEVSTS(R5),20$
   51   7010 8F  3C 0565          MOVZWL  #EBCDLE!<EBCAK0*256>,R1 ; Get an AK0
            05   11 056A          BRB     30$                     ; And send it
   51   6110 8F  3C 056C  20$:    MOVZWL  #EBCDLE!<EBCAK1*256>,R1 ; Get an AK1
          0336   31 0571  30$:    BRW     XMT_START_PRO2          ; Start the device
```

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

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

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

```
         00   E4 06A4  80$:    BBSC    #UCB$V_XW_REP,-              ; Are we in a rep state
   09 68 A5      06A6                  UCB$W_DEVSTS(R5),90$
                 06A9          PROTLO  BADACK
       0080   31 06AF          BRW     180$                         ; Retry without rep state
    0081 C5   90 06B2  90$:    MOVB    UCB$B_ERTMAX(R5),-           ; Reset the retry counter
    0080 C5      06B6                  UCB$B_ERTCNT(R5)
                 06B9          PROTLO  REP                          ; Record a retry
       FCA5   31 06BF          BRW     SND_REC_REQ                  ; Resend the last block
     51   3D  91 06C2  100$:   CMPB    #EBCNAK,R1                   ; Is it a NAK
          1F  12 06C5          BNEQ    120$                         ; BR if not
                 06C7          PROTLO  NAK                          ; Report a nak sent
  68 A5   01  AA 06CD          BICW    #UCB$M_XW_REP,UCB$W_DEVSTS(R5) ; Reset possiable rep state
    0080 C5   97 06D1          DECB    UCB$B_ERTCNT(R5)             ; Dec. the retry count
          03  15 06D5          BLEQ    110$                         ; Retry if threshold not exceeded
       FC8D   31 06D7          BRW     SND_REC_REQ                  ; Try again
     51   02  3C 06DA  110$:   MOVZWL  #XW$M_DATACK,R1              ; Get retry threshold exceeded status
                 06DD          PROTLO  NAKOUT
       00D9   31 06E3          BRW     ABORT_CUR                    ; And abort the request and queue
     51   37  91 06E6  120$:   CMPB    #EBCEOT,R1                   ; Is it an EOT
          0C  12 06E9          BNEQ    130$                         ; BR if not
                 06EB          PROTLO  ABEOT
     51   01  3C 06F1          MOVZWL  #XW$M_EOT,R1                 ; Get EOT status
       00C8   31 06F4          BRW     ABORT_CUR                    ; Abort the IRP and queue
     51   02  91 06F7  130$:   CMPB    #EBCSTX,R1                   ; Is it a conversational response
          0E  12 06FA          BNEQ    140$                         ; Consider it a garbled response
                 06FC          PROTLO  CON_RESP
  51  0800 8F 3C 0702          MOVZWL  #XW$M_CONACK,R1              ; Get conversational response status
                 0707          ; Toggle ACKs here ?
       0085   31 0707          BRW     COMP_GOOD_AB                 ; Complete the request with status
                 070A  140$:   PROTLO  GARBLED_RESP
          14  11 0710          BRB     170$                         ; Go to garbled response processing
                 0712  ;
                 0712  ; Response timout processing- send an ENQ to see what happened
                 0712  ;
       FE9E   30 0712  150$:   BSBW    TMO_RESET                    ; Reset the transmitter
                 0715          PROTLO  TMO_XMT_TXTBLK
       FE95   30 071B  160$:   BSBW    TMO_RESET                    ; Reset the device and go to fork
03 64 A5  03  E1 071E          BBC     #UCB$V_CANCEL,UCB$W_STS(R5),170$ ;If user cancel get out now
       0090   31 0723          BRW     CANCEL_CUR                   ; Gancet the IRP
          00  E2 0726  170$:   BBSS    #UCB$V_XW_REP,-              ; Is this a new rep state
   07 68 A5      0728                  UCB$W_DEVSTS(R5),180$
    0081 C5   90 072B          MOVB    UCB$B_ERTMAX(R5),-           ; Reset the retry count
    0080 C5      072F                  UCB$B_ERTCNT(R5)
    0080 C5   97 0732  180$:   DECB    UCB$B_ERTCNT(R5)             ; Retry error ?
          0C  14 0736          BGTR    190$                         ; BR if not
     51   02  3C 0738          MOVZWL  #XW$M_DATACK,R1              ; Get a threshold exceeded
                 073B          PROTLO  REPOUT
       007B   31 0741          BRW     ABORT_CUR                    ; Complete the request
     51   2D  9A 0744  190$:   MOVZBL  #EBCENQ,R1                   ; Get retry protocol
                 0747          PROTLO  REP_SENT
  64  0200 8F B3 074D          BITW    #DUP_CSR_M_DSR,(R4)          ; Is DSR up
          09  12 0752          BNEQ    195$                         ; Br if not
                 0754          PROTLO  DSR_LOST
       FC2E   31 075A          BRW     ABORT_NO_DSR                 ; Get out
       0144   30 075D  195$:   BSBW    XMT_START_PRO1               ; Start up a protocol transmission
                 0760          WFIKPCH 200$,#3                      ; And wait for it to complete
                 076A          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      ;
                              077F      ; Complete the request succesfully
                              077F      ;
                              077F                  .ENABL  LSB
                              077F      COMP_GOOD_TGL:
32 A3    40 A3     B0         077F                  MOVW    IRP$W_ABCNT(R3),IRP$W_BCNT(R3) ; Set the origional user size
            01     E3         0784                  BBCS    #UCB$V_XW_AK1,-
      04 68 A5                0786                          UCB$W_DEVSTS(R5),10$    ; Toggel the ack switch
   68 A5    02     CA         0789                  BICL    #UCB$M_XW_AK1,UCB$W_DEVSTS(R5)
                              078D      10$:
                              078D      COMP_GOOD:
            51     D4         078D                  CLRL    R1                      ; No device dependant status
                              078F      COMP_GOOD_AB:
   50    32 A3     B0         078F                  MOVW    IRP$W_BCNT(R3),R0       ; Set the transfer size in status
50    50    10     78         0793                  ASHL    #16,R0,R0               ; Move it to the high byte
50    0000'8F      B0         0797                  MOVW    #SS$_NORMAL,R0          ; Get success code
                              079C      COMPLETE:
         40 A3     B4         079C      20$:        CLRW    IRP$W_ABCNT(R3)         ; Clean up unused field
0C 2A A3    07     E1         079F                  BBC     #IRP$V_DIAGBUF,IRP$W_STS(R3),25$ ; Is a diagnostic buffer here
      7E    50     7D         07A4                  MOVQ    R0,-(SP)                ; Save the completion status
 00000000'GF       16         07A7                  JSB     G^IOC$DIAGBUFILL        ; Fill the diagnostic buffer if any
      50    8E     7D         07AD                  MOVQ    (SP)+,R0                ; Get the status back
                              07B0      25$:        REQCOM                          ; Complete the request
                              07B6
                              07B6      CANCEL_CUR:
            51     D4         07B6                  CLRL    R1                      ; No device dependant status
50    0000'8F      3C         07B8                  MOVZWL  #SS$_CANCEL,R0          ; Get cancel status
            09     11         07BD                  BRB     30$                     ; And complete the request
                              07BF
                              07BF      ABORT_CUR:
   68 A5    08     A8         07BF                  BISW    #UCB$M_XW_PIPE,UCB$W_DEVSTS(R5) ; set pipeline marker
50    0000'8F      3C         07C3                  MOVZWL  #SS$_ABORT,R0           ; Get aborted status (device dependant
                              07C8                                                  ;  status is already set
                              07C8      30$:
         32 A3     B4         07C8      BAD_COMP:           CLRW    IRP$W_BCNT(R3)          ; Zero byte count
            CF     11         07CB                  BRB     20$                     ; Complete the request
                              07CD                  .DSABL  LSB
```

```
                        07CD    ;
                        07CD    ; Start a full duplex diagnostic read/write
                        07CD    ;
                        07CD    XMT_INI_DIAG:
     50    32 A3    3C  07CD            MOVZWL  IRP$W_BCNT(R3),R0           ; Get the receive count
     78 A5    50    C0  07D1            ADDL2   R0,UCB$L_SVAPTE(R5)         ; Point to transmit buffer
  7E A5    40 A3    B0  07D5            MOVW    IRP$W_ABCNT(R3),UCB$W_BCNT(R5) ; Set xmt. buffer count
  0090 C5    50     B0  07DA            MOVW    R0,UCB$B_SLAVE(R5)          ; Set receive count
        2C B3       D0  07DF            MOVL    @IRP$L_SVAPTE(R3),-         ; Set receive buffer pointer
   009C C5              07E2                    UCB$L_XW_SPBF(R5)
                        07E5    ; UCB$L_SVAPTE = transmit buffer pointer
                        07E5    ; UCB$W_BCNT = transmit buffer count
                        07E5    ; UCB$L_XW_SPBF = receive buffer pointer
                        07E5    ; UCB$B_SLAVE = receive buffer count
  00A9 C5    01     90  07E5            MOVB    #TSTA$CTS,UCB$B_XW_TISTA(R5) ;set transmit int. state
  00AA C5    0C     90  07EA            MOVB    #RST$DIAG,UCB$B_XW_RISTA(R5) ; Set receive state
  00A0 C5    04     9A  07EF            MOVZBL  #4,UCB$L_XW_SYCNT(R5)       ; Set the sync count
                        07F4            DSBINT                              ; Disable interrupts
     50    46 A5    9A  07FA            MOVZBL  UCB$L_DEVDEPEND+2(R5),R0    ; Get the sync character
     8200 8F        A1  07FE            ADDW3   #DUP_CSR_M_BYM!DUP_CSR_M_CRI,- ; Set up the param. csr
  02 A4    50           0802                    R0,DUP_CSR_PARM(R4)
     64    10       AA  0805            BICW    #DUP_CSR_M_REN,(R4)         ; Force a resync
  64    0154 8F     A8  0808            BISW    #DUP_CSR_M_SSY!DUP_CSR_M_REN!- ; Set RTS,REN,SSY,RIE
                        080D                    DUP_CSR_M_RIE!DUP_CSR_M_RTS,(R4)
     0050 8F        A8  080D            BISW    #DUP_CSR_M_TIE!DUP_CSR_M_SND,- ; Set TIE and SEND
        04 A4           0811                    DUP_CSR_TXCSR(R4)
     01FF 8F        B0  0813            MOVW    #DUP_CSR_M_SOM!PAD,-        ; Set start of message and a PAD
        06 A4           0817                    DUP_CSR_TXDBF(R4)
           09       E1  0819            BBC     #IO$V_MAINTLOOP,-           ; Is it a internal loop
     06 20 A3           081B                    IRP$W_FUNC(R3),2$
     0800 8F        A8  081E            BISW    #DUP_CSR_M_STM,-            ; Set the system test mode bit
        04 A4           0822                    DUP_CSR_TXCSR(R4)
           06       E1  0824    2$:     BBC     #IO$V_NOCTSWAIT,-           ; Is the no CTS wait state requested
     05 20 A3           0826                    IRP$W_FUNC(R3),5$
  00A9 C5    05     90  0829            MOVB    #TSTA$DIAG,UCB$B_XW_TISTA(R5) ; Set the no CTS wait state
           0C       E1  082E    5$:     BBC     #IO$V_INTCLOCK,-            ; Is this a internal clock test
     06 20 A3           0830                    IRP$W_FUNC(R3),10$
  04 A4    1000 8F  A8  0833            BISW    #DUP_CSR_M_ETM,DUP_CSR_TXCSR(R4) ; Yes turn it on
                        0839    10$:    WFIKPCH 30$,#2                      ; Wait for request to finish
                        0843            DSBINT
                        0849    15$:    WFIKPCH 40$,#2                      ; Wait for receiver to finish
                        0853            IOFORK                              ; Go to fork level
     FF3F CF        DF  0859            PUSHAL  COMPLETE                    ; Set up a normal cpmpletion
  50    0000'8F     3C  085D            MOVZWL  #SS$_NORMAL,R0              ; Get normal status
     0090 C5        A3  0862    20$:    SUBW3   UCB$B_SLAVE(R5),-           ; Get the receive size
  51    32 A3           0866                    IRP$W_BCNT(R3),R1
  32 A3    51       B0  0869            MOVW    R1,IRP$W_BCNT(R3)           ; Fake transfer size
  51    51    10    78  086D            ASHL    #16,R1,R1                   ; Put it in the high byte
        7E A5       A3  0871            SUBW3   UCB$W_BCNT(R5),-            ; Get the transmit count
  51    40 A3           0874                    IRP$W_ABCNT(R3),R1
  2A A3    02       A8  0877            BISW    #IRP$M_FUNC,IRP$W_STS(R3)   ; Set read for completion code
           05           087B            RSB                                 ; Join the common completion code
                        087C
  15 64 A5    03    E0  087C    30$:    BBS     #UCB$V_CANCEL,UCB$W_STS(R5),50$ ; Is a cancel requested
                        0881            DSBINT
        B0    11        0887            BRB     10$                         ; NO keep waiting
  08 64 A5    03    E0  0889    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  ;
                         08A4  ; Transmit startup utility
                         08A4  ;
                         08A4  XMT_START_PRO1:
                         08A4          .ENABL  LSB
        7E A5    01   B0 08A4          MOVW    #1,UCB$W_BCNT(R5)           ; Set the transfer count at one
                 04   11 08A8          BRB     10$                         ; Now load the buffer with the message
                         08AA  XMT_START_PRO2:
        7E A5    02   B0 08AA          MOVW    #2,UCB$W_BCNT(R5)           ; Set the transfer count at two
      009C C5    51   D0 08AE  10$:    MOVL    R1,UCB$L_XW_SPBF(R5)        ; Load the buffer with the protocol mes.
         009C C5      DE 08B3          MOVAL   UCB$L_XW_SPBF(R5),-         ; Set the protocol buffer pointer
            78 A5        08B7          UCB$L_SVAPTE(R5)
                         08B9  XMT_START:
      00A9 C5    01   90 08B9          MOVB    #TSTA$CTS,UCB$B_XW_TISTA(R5) ; Set the starting state to CTS
      00A0 C5    08   9A 08BE          MOVZBL  #8,UCB$L_XW_SYCNT(R5)       ; set the sync count
        64       04   A8 08C3          BISW    #DUP_CSR_M_RTS,(R4)         ; Set request to send
        50       8E   D0 08C6          MOVL    (SP)+,R0                    ; Save the return
                         08C9          DSBINT                              ; Disable interrupts
        7E       50   D0 08CF          MOVL    R0,-(SP)                    ; Put the return back
      8200 8F         B0 08D2          MOVW    #DUP_CSR_M_CRI!DUP_CSR_M_BYM,-
        02 A4            08D6          DUP_CSR_PARM(R4)
      0058 8F         A8 08D8          BISW    #DUP_CSR_M_TIE!DUP_CSR_M_HDX!DUP_CSR_M_SND,- ; Turn on interrupt
        04 A4            08DC          DUP_CSR_TXCSR(R4)
06 A4 01FF 8F         B0 08DE          MOVW    #DUP_CSR_M_SOM!PAD,DUP_CSR_TXDBF(R4) ; Set SOM and PAD
                         08E4          ; Is the TSOM and SEND order inportant ?
                      05 08E4          RSB                                 ;Return from subroutine
                         08E5          .DSABL  LSB
                         08E5  ;
                         08E5  ;Receive startup utility
                         08E5  ;
                         08E5  RCV_START_PRO:
      00AA C5    02   90 08E5          MOVB    #RST$FIRST,UCB$B_XW_RISTA(R5) ; Set starting receive state
78 A5 009C C5         DE 08EA          MOVAL   UCB$L_XW_SPBF(R5),UCB$L_SVAPTE(R5) ; Set the protocol buffer
        7E A5    02   B0 08F0          MOVW    #2,UCB$W_BCNT(R5)           ; Set the protocol response
                         08F4  RCV_START:
        50    46 A5   9A 08F4          MOVZBL  UCB$L_DEVDEPEND+2(R5),R0 ; Set the SYN character in R0
      8200 8F         A1 08F8          ADDW3   #DUP_CSR_M_BYM!DUP_CSR_M_CRI,- ; Set SYN char. and no crc
        02 A4    50      08FC          R0,DUP_CSR_PARM(R4)
        64       10   AA 08FF          BICW    #DUP_CSR_M_REN,(R4)         ; Force a resync
        50       8E   D0 0902          MOVL    (SP)+,R0                    ; Save the return
                         0905          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          .SBTTL  Block Checking
                                    0914  ;+
                                    0914  ;
                                    0914  ; Remove all control characters, check all CRCs, and compact the data into
                                    0914  ; the same buffer.
                                    0914  ;
                                    0914  ;
                                    0914  ; input
                                    0914  ;       R0- count from R1 to end of buffer
                                    0914  ;       R1- pointer to byte after STX
                                    0914  ;
                                    0914  ; output
                                    0914  ;       R0, R1- destroyed
                                    0914  ;       control is passed to either the NAK or ACK routine after checking
                                    0914  ;
                                    0914  ; Register conventions
                                    0914  ;       R6-ITB pointer
                                    0914  ;       R7-count pointer
                                    0914  ;       R8-ITB count
                                    0914  ;       R9- pointer to beginning of raw data
                                    0914  ;       R10-count to CRC
                                    0914  ;
                                    0914  ;
                                    0914  ;-
                     0000018C       0914          DLE_STX_CRC=^X18C                   ; DLE, STX string CRC
                     0000C181       0914          STX_CRC=^XC181                      ; Partial CRC for STX
                                    0914
                                    0914  BLK_CHECK:
                                    0914          .ENABL  LSB
               06F8 8F    BB        0914          PUSHR   #^m<R3,R4,R5,R6,R7,R9,R10>
                 59 51    D0        0918          MOVL    R1,R9                       ; Save the input pointer
             57    2C B3  D0        091B          MOVL    @IRP$L_SVAPTE(R3),R7        ; Get the output pointer
          56    00A4 C5   D0        091F          MOVL    UCB$L_XW_ITBPTR(R5),R6      ; Get the ITB table base
                 56 1C    C0        0924          ADDL    #7*4,R6                     ; Point to tne first ITB
              7E 51 50    C1        0927          ADDL3   R0,R1,-(SP)                 ; Point to end of block
                    7E    D4        092B          CLRL    -(SP)                       ; No partial CRC
                 5A 66    D0        092D          MOVL    (R6),R10                    ; Are there any internal CRC
                    3A    12        0930          BNEQ    20$                         ; BR if there are
           2C B3    79    3E        0932          MOVAW   -(R9),@IRP$L_SVAPTE(R3)     ; Fake the system buffer pointer
              89 50 03    A3        0936          SUBW3   #3,R0,(R9)+                 ; Set count for user
                 5A 50    D0        093A          MOVL    R0,R10                      ; Set up the count
 69  5A  6E  00000CAC'EF  0B        093D  5$:     CRC     POLY_TABLE,(SP),R10,(R9)    ; DO the CRC on the whole block
                    6C    12        0946          BNEQ    BAD_CRC                     ; BR if not zero
          FD A3 FFFF 8F   B0        0948          MOVW    #-1,-3(R3)                  ; Store the sentenal
              50 53 01    C3        094E          SUBL3   #1,R3,R0                    ; Point to end of block
                  0086    31        0952          BRW     GOOD_TEXT                   ; Send a ACk
                 5A 76    D0        0955  10$:    MOVL    -(R6),R10                   ; Get count to end of CRC
                    12    12        0958          BNEQ    20$                         ; Not end so branch
           5A 04 AE 59    C3        095A          SUBL3   R9,4(SP),R10                ; Get the remaining count
              87 5A 03    A3        095F          SUBW3   #3,R10,(R7)+                ; Set the last count
              67 69 5A    28        0963          MOVC    R10,(R9),(R7)               ; Move the last record
                 59 57    D0        0967          MOVL    R7,R9                       ; Fake the last CRC
                    D1    11        096A          BRB     5$                          ; Do the last CRC
                 5A 59    C2        096C  20$:    SUBL    R9,R10                      ; Get count to next ITB CRC
 69  5A  6E  00000CAC'EF  0B        096F  30$:    CRC     POLY_TABLE,(SP),R10,(R9)    ; Do the partial CRC
                    3A    12        0978          BNEQ    BAD_CRC                     ; BR if bad CRC
              87 5A 03    A3        097A          SUBW3   #3,R10,(R7)+                ; store the count in the buffer
```

```
     67   69   5A   28  097E          MOVC    R10,(R9),(R7)            ; And compact the data
     57   53   03   C3  0982          SUBL3   #3,R3,R7                 ; Point to next count
  5A   04 AE   51   C3  0986          SUBL3   R1,4(SP),R10             ; Get the count remaining
     61   5A   32   3B  098B          SKPC    #EBCSYN,R10,(R1)         ; Skip the SYN
               23   13  098F          BEQL    BAD_CRC                  ; BR if bad CRC
  6E   0000018C 8F  D0  0991          MOVL    #DLE_STX_CRC,(SP)        ; start the next CRC
     81   0210 8F   B1  0998          CMPW    #EBCDLE!<EBCSTX*256>,(R1)+ ; The DLE STX are in the CRC
               10   13  099D          BEQL    40$                      ; BR if  here
               6E   D4  099F          CLRL    (SP)                     ; Set no partial CRC
               71   B5  09A1          TSTW    -(R1)                    ; Back up
          61   02   91  09A3          CMPB    #EBCSTX,(R1)             ; Dies the next record start with STX
               07   12  09A6          BNEQ    40$                      ; Br if not
     6E   C181 8F   3C  09A8          MOVZWL  #STX_CRC,(SP)            ; Get the STX partial crc
               51   D6  09AD          INCL    R1                       ; Remove the STX
          59   51   D0  09AF  40$:    MOVL    R1,R9                    ; Set the new from pointer
               A1   11  09B2          BRB     10$                      ; Get next record
                        09B4  BAD_CRC:
          5E   08   C0  09B4          ADDL    #2*4,SP                  ; Pop the CRC and end
          06F8 8F   BA  09B7          POPR    #^M<R3,R4,R5,R6,R7,R9,R10>
               04   C8  09BB          BISL    #UCB$M_XW_NAK,- ; Mark a NAK sent
             68 A5      09BD                  UCB$W_DEVSTS(R5)
                        09BF          PROTLO  NAK_SENT
          FB8D      30  09C5          BSBW    SEND_RESP                ; Set up and start the NAK
                        09C8          WFIKPCH REC_SNAK_TMO,#4
                        09D2          IOFORK                           ; Create a fork process
          F9BC      31  09D8          BRW     SND_REC_REQ              ; Restart the receive buffer
                        09DB  GOOD_TEXT:
          5E   08   C0  09DB          ADDL    #2*4,SP                  ; Pop the CRC and end
          06F8 8F   BA  09DE          POPR    #^M<R3,R4,R5,R6,R7,R9,R10>
     50   2C B3     C2  09E2          SUBL    @IRP$L_SVAPTE(R3),R0     ; Get buffer size
     40 A3   50     B1  09E6          CMPW    R0,IRP$W_ABCNT(R3)       ; Is the users buffer large enough
               08   19  09EA          BLSS    55$                      ; Br if it is
     50   0000'8F   3C  09EC          MOVZWL  #SS$_TOOMUCHDATA,R0      ; Get a error message for the user
          FDD4      31  09F1          BRW     BAD_COMP                 ; And return the buffer with a fatal err
     32 A3   50     B0  09F4  55$:    MOVW    R0,IRP$W_BCNT(R3)        ; Set the transfer size for the iocomp.
               01   E3  09F8          BBCS    #UCB$V_XO_AK1,-          ; Toggel the ACK switch
       04 68 A5         09FA                  UCB$W_DEVSTS(R5),60$     ;...
     68 A5   02     CA  09FD          BICL    #UCB$M_XW_AK1,UCB$W_DEVSTS(R5) ;...
               04   CA  0A01  60$:    BICL    #UCB$M_XW_NAK,-          ; Reset NAK sent last switch
             68 A5      0A03                  UCB$W_DEVSTS(R5)
  03 64 A5     03   E1  0A05          BBC     #UCB$V_CANCEL,UCB$W_STS(R5),70$ ; If canceled the higher
                        0A0A                                           ;  levels will want to send an EOT
                        0A0A                                           ;  Rather than this ACK
          FDA9      31  0A0A          BRW     CANCEL_CUR               ; Return this packet cancled
          FB45      30  0A0D  70$:    BSBW    SEND_RESP                ; Set up the correct response
                        0A10          WFIKPCH REC_GACK_TMO,#4          ; And wait for it to be sent
                        0A1A          IOFORK                           ; Create a fork process
                        0A20  REC_DONE:
     51   3C A3     D0  0A20          MOVL    IRP$L_MEDIA+4(R3),R1     ; Get the dev. dep. status
          FD68      31  0A24          BRW     COMP_GOOD_AB             ; Complete the request
                        0A27
                        0A27          .DSABL  LSB
```

```
                 0A27          .SBTTL  DUP_11 Receive Interrupt Processing
                 0A27  ;+
                 0A27  ;XW_RINT - DUP-11 RECEIVE INTERRUPT PROCESSING
                 0A27  ;
                 0A27  ; THIS ROUTINE IS ENTERED VIA A JSB INSTRUCTION WHEN AN INTERRUPT OCCURS ON A
                 0A27  ; DUP-11 LINE INTERFACE. THE STATE OF THE STACK ON ENTRY IS:
                 0A27  ;
                 0A27  ;       00(SP) = ADDRESS OF IDB ADDRESS.
                 0A27  ;       04(SP) = SAVED R3.
                 0A27  ;       08(SP) = SAVED R4.
                 0A27  ;       12(SP) = SAVED R5.
                 0A27  ;       16(SP) = INTERRUPT PC.
                 0A27  ;       20(SP) = INTERRUPT PSL.
                 0A27  ;
                 0A27  ; Interrupt dispatching occurs according to received data.
                 0A27  ;
                 0A27  ;-
                 0A27  XW_RINT::
                 0A27          .ENABL  LSB
      53   9E  D0 0A27         MOVL    @(SP)+,R3                     ; Get the IDB pointer
      54   63  7D 0A2A         MOVQ    IDB$L_CSR(R3),R4              ; Set CSR and UCB pointer
   64 A5   02  AA 0A2D         BICW    #UCB$M_INT,UCB$W_STS(R5)      ; Clear expected bit
   50   02 A4  9A 0A31         MOVZBL  DUP_CSR_RDBF(R4),R0           ; Get the received byte
                 0A35          CASE    UCB$B_XW_RISTA(R5),-          ; Vector the interrupt
                 0A35                  TYPE=B,-                      ; Byte data type
                 0A35                  <RST_IDLE,-                   ; Idle state
                 0A35                  RST_BINARY,-                  ; Binary reception
                 0A35                  RST_FIRST,-                   ; First character state
                 0A35                  RST_NTTX,-                    ; Nontransparent text
                 0A35                  RST_STOP1,-                   ; First byte of last CRC
                 0A35                  RST_STOP,-                    ; Store last byte and complete request
                 0A35                  RST_ITB,-                     ; Store internal CRC
                 0A35                  RST_ITB1,-                    ; Store last byte of internal CRC
                 0A35                  RST_XPRTEXT,-                 ; Transparent text
                 0A35                  RST_DLE,-                     ; transparent DLE processing
                 0A35                  RST_XPRITB,-                  ; Transparent internal CRC
                 0A35                  RST_XPRITB1,-                 ; Second byte of internal CRC
                 0A35                  RST_DIAG,-                    ; Diagnostic function
                 0A35                  RST_LOOP_TEST,-               ; Slave loop diagnostic function
                 0A35                  >
                 0A57
                 0A57  ;
                 0A57  ; Second byte of internal CRC state
                 0A57  ;
                 0A57  RST_XPRITB1:
 00AA C5   08  90 0A57         MOVB    #RST$XPRTEXT,UCB$B_XW_RISTA(R5) ;Go back to transparent text
           05  11 0A5C         BRB     100$                          ; Go to common internal CRC code
                 0A5E  ;
                 0A5E  ; Store the second internal CRC
                 0A5E  ;
                 0A5E  RST_ITB1:
 00AA C5   03  90 0A5E         MOVB    #RST$NTTX,UCB$B_XW_RISTA(R5) ; Go back to text state
                 0A63  ;
                 0A63  ; Internal CRC processing
                 0A63  ;
   78 B5   50  90 0A63  100$:  MOVB    R0,@UCB$L_SVAPTE(R5)      ; Store the last CRC byte
      78 A5    D6 0A67         INCL    UCB$L_SVAPTE(R5)          ; Step pointer
```

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

```
        78 B5   50   90 0AEB                MOVB    R0,@UCB$L_SVAPTE(R5)      ; Store it in the buffer
           78 A5   D6 0AEF                INCL    UCB$L_SVAPTE(R5)          ; Step the buffer pointer
           7E A5   B7 0AF2                DECW    UCB$W_BCNT(R5)            ; Include the byte in the count
                      0AF5        24$:
              D3   15 0AF5                BLEQ    50$                       ; BR if the buffer is full
                      0AF7        RST_IDLE:
        50   8E   7D 0AF7        25$:    MOVQ    (SP)+,R0                  ; Restore R0-1
        52   8E   7D 0AFA                MOVQ    (SP)+,R2                  ; Restore R2-3
        54   8E   7D 0AFD                MOVQ    (SP)+,R4                  ; Restore R4-5
                  02 0B00                REI                               ; Return from interrupt
                     0B01        ;
                     0B01        ; Store the first internal CRC character
                     0B01        ;
                     0B01        RST_ITB:
   00AA C5   07   90 0B01                MOVB    #RST$ITB1,UCB$B_XW_RISTA(R5) ; Set state to store second CRC
             E3   11 0B06                BRB     RIDONE                    ; Complete the interrupt
                     0B08        ;
                     0B08        ; Transparent text state- if DLE is found drop it and go to DLE state
                     0B08        ;
                     0B08        RST_XPRTEXT:
        50   10   91 0B08                CMPB    #EBCDLE,R0                ; Is it a DLE
             DE   12 0B0B                BNEQ    RIDONE                    ; If not continue
   00AA C5   09   90 0B0D                MOVB    #RST$DLE,UCB$B_XW_RISTA(R5) ; Set state to DLE
             D7   11 0B12                BRB     RIDONE                    ; Complete the interrupt
                     0B14        ;
                     0B14        ; Transparent character after a DLE
                     0B14        ;
                     0B14        RST_DLE:
        50   1F   91 0B14                CMPB    #EBCITB,R0                ; Is it an internal CRC
             0D   12 0B17                BNEQ    40$                       ; Check sone more
   00AA C5   0A   90 0B19                MOVB    #RST$XPRITB,UCB$B_XW_RISTA(R5) ; Set transparent ITB state
           78 A5   D7 0B1E        42$:    DECL    UCB$L_SVAPTE(R5)          ; Remove the DLE
           7E A5   B6 0B21                INCW    UCB$W_BCNT(R5)            ; ...
              C5   11 0B24                BRB     RIDONE                    ; Exit the interrupt
        50   32   91 0B26        40$:    CMPB    #EBCSYN,R0                ; Is it a SYN in text
             0D   12 0B29                BNEQ    45$                       ; BR if not
   00AA C5   08   90 0B2B                MOVB    #RST$XPRTEXT,UCB$B_XW_RISTA(R5) ; Set state to text
           78 A5   D7 0B30                DECL    UCB$L_SVAPTE(R5)          ; Remove the DLE also
           7E A5   B6 0B33                INCW    UCB$W_BCNT(R5)            ; ...
              BF   11 0B36                BRB     25$                       ; And exit the interrupt
0C 00000C8C'EF   50   E0 0B38        45$:    BBS     R0,CCHRMASK,47$           ; If it is a cont. cha. strip the DLE
   00AA C5   08   90 0B40                MOVB    #RST$XPRTEXT,UCB$B_XW_RISTA(R5) ;Go back to transparent text
        50   10   91 0B45                CMPB    #EBCDLE,R0                ; Is it a DLE in text
             AD   13 0B48                BEQL    25$                       ; If so dump it
             9F   11 0B4A                BRB     RIDONE                    ; get out
   00AA C5   04   90 0B4C        47$:    MOVB    #RST$STOP1,UCB$B_XW_RISTA(R5) ; Set state to stop after CRC
             CB   11 0B51                BRB     42$                       ; Complete the interrupt
                     0B53        ;
                     0B53        ; Transparent ITB state
                     0B53        ;
                     0B53        RST_XPRITB:
   00AA C5   0B   90 0B53                MOVB    #RST$XPRITB1,UCB$B_XW_RISTA(R5) ; Go to second byte of CRC
             91   11 0B58                BRB     RIDONE                    ; Complete the interrupt
                     0B5A        RST_BINARY:
   64   0100 8F   AA 0B5A                BICW    #DUP_CSR_M_SSY,(R4)       ; Reset strip sync
           FF89   31 0B5F                BRW     RIDONE
                     0B62
```

```
                            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
```

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

EXE$

Modu

TST$
TST$
TST$
TST$
TST$
DTSM
SYS$
SYS$
SYS$
LIBR

```
                         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     00   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          .SBTTL  REGDUMP Dump the errors and device CSRs
                          0C14   ;+
                          0C14   ;      Move the accumulated errors and CSR 0 into an error log buffer
                          0C14   ; or a diagnostic buffer.
                          0C14   ;
                          0C14   ; input
                          0C14   ;      R0- buffer pointer
                          0C14   ;      R5- UCB pointer
                          0C14   ;      R4- CSR pointer
                          0C14   ;
                          0C14   ; output
                          0C14   ;      @R0 is filled with:
                          0C14   ;              # long words filled
                          0C14   ;              last (or current) fatal error
                          0C14   ;              CSR 0
                          0C14   ;              accumulated errors (see UCB definition)
                          0C14   ;
                          0C14   ;-
                          0C14   REGDUMP:
                    06  BB    0C14          PUSHR   #^M<R1,R2>
 51    55   000000AD 8F  C1    0C16          ADDL3   #UCB$B_XW_ERRORS,R5,R1  ; Get a pointer to the first error
               52   13  D0    0C1E          MOVL    #NO_ERRORS+1,R2         ; Get the number of error fields
               80   15  D0    0C21          MOVL    #<NO_ERRORS+3>,(R0)+ ; Store no. long words
               80   81  9A    0C24          MOVZBL  (R1)+,(R0)+             ; Store the last error logged
         80   00AB C5  3C    0C27          MOVZWL  UCB$W_XW_LSTCSR(R5),(R0)+ ; Store the CSR
               80   81  9A    0C2C   10$:   MOVZBL  (R1)+,(R0)+             ; Store a long word
               FA   52  F5    0C2F          SOBGTR  R2,10$                  ; BR if not done
      80   FFFFFFFF 8F  D0    0C32          MOVL    #-1,(R0)+               ; Put in sentinal
                    06  BA    0C39          POPR    #^M<R1,R2>
                    05        0C3B          RSB
                              0C3C
                              0C3C          .SBTTL  ERR_RECORD Save an error in the UCB
                              0C3C   ;+
                              0C3C   ; The error is recorded in the appropriate UCB field, if it
                              0C3C   ; is fatal the error logger is called.
                              0C3C   ;
                              0C3C   ; input
                              0C3C   ;      R5- UCB
                              0C3C   ;      R4- Device CSR
                              0C3C   ;      4(SP)- error
                              0C3C   ;      (SP)- return
                              0C3C   ;
                              0C3C   ; output
                              0C3C   ;      The UCB is updated and the error is logged if fatal
                              0C3C   ;
                              0C3C   ;
                              0C3C   ERR_RECORD:
               7E   50  D0    0C3C          MOVL    R0,-(SP)                ; Save a reg.
 50    08 AE    FF 8F  78    0C3F          ASHL    #-1,2*4(SP),R0          ; Get the error number
       00AD C5    50  90    0C45          MOVB    R0,UCB$B_XW_ERRORS(R5)  ; Save the last error
         00AE C540    96    0C4A          INCB    UCB$B_XW_ERRORS+1(R5)[R0] ; Step the error cell
                 05  12    0C4F          BNEQ    5$                      ; Dont pass 0
         00AE C540    97    0C51          DECB    UCB$B_XW_ERRORS+1(R5)[R0] ; ...
           06 08 AE    E9    0C56   5$:    BLBC    2*4(SP),10$             ; Is this a fatal error
         00000000'GF  16    0C5A          JSB     G^ERL$DEVICERR          ; Yes call the error logger
               50   8E  D0    0C60   10$:   MOVL    (SP)+,R0                ; Restore R0
               6E   8E  D0    0C63          MOVL    (SP)+,(SP)              ; Remove the error
```

_$2
Pse

TST
TST
MSG
MSG
MSG
MSG

```
05 0C66 1584          RSB
```



```
Sym
----
C
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
```

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

```
         0C8C          .SBTTL  Control character table
         0C8C  CCHRMASK:
80000008 0C8C          .LONG   ^B10000000000000000000000000001000      ;ETX,ENQ
00842040 0C90          .LONG   ^B00000000100001000010000001000000      ; EOT, SYN, ETB
00000000 0C94          .LONG   ^B00000000000000000000000000000000      ;
00000000 0C98          .LONG   ^B00000000000000000000000000000000      ;
00000000 0C9C          .LONG   ^B00000000000000000000000000000000      ;
00000000 0CA0          .LONG   ^B00000000000000000000000000000000      ;
00000000 0CA4          .LONG   ^B00000000000000000000000000000000      ;
00000000 0CA8          .LONG   ^B00000000000000000000000000000000      ;
         0CAC          .SBTTL  CRC16 Polynomial Table
         0CAC  POLY_TABLE:
00000000 0CAC          .LONG   0
0000CC01 0CB0          .LONG   52225
0000D801 0CB4          .LONG   55297
00001400 0CB8          .LONG   5120
0000F001 0CBC          .LONG   61441
00003C00 0CC0          .LONG   15360
00002800 0CC4          .LONG   10240
0000E401 0CC8          .LONG   58369
0000A001 0CCC          .LONG   40961
00006C00 0CD0          .LONG   27648
00007800 0CD4          .LONG   30720
0000B401 0CD8          .LONG   46081
00005000 0CDC          .LONG   20480
00009C01 0CE0          .LONG   39937
00008801 0CE4          .LONG   34817
00004400 0CE8          .LONG   17408
         0CEC
         0CEC  XW_END:
         0CEC          .END
```

```
$$$                    = 00000020 R    02      DYN$C_DDB              = 00000006
$$OP                   = 00000002              DYN$C_DPT              = 0000001E
ABEOT                  = 0000001A              DYN$C_UCB              = 00000010
ABORT_CUR                00000 7BF R   03      EBCAK0                 = 00000070
ABORT_NO_DSR             0000038B R    03      EBCAK1                 = 00000061
ALLOC_SETUP              00000231 R    03      EBCDC1                 = 00000011
ASCSP                  = 00000020              EBCDC2                 = 00000012
AT$_UBA                = 00000001              EBCDC3                 = 00000013
BADACK                 = 00000009              EBCDLE                 = 00000010
BAD_COMP                 000007C8 R    03      EBCENQ                 = 0000002D
BAD_CRC                  000009B4 R    03      EBCEOT                 = 00000037
BAD_MODIFIER             00000092 R    03      EBCESC                 = 00000027
BLK_CHECK                00000914 R    03      EBCETB                 = 00000026
CANCEL_CUR               000007B6 R    03      EBCETX                 = 00000003
CCHRMASK                 00000C8C R    03      EBCHT                  = 00000005
COMPLETE                 0000079C R    03      EBCIGS                 = 0000001D
COMP_GOOD                0000078D R    03      EBCIRS                 = 0000001E
COMP_GOOD_AB             0000078F R    03      EBCITB                 = 0000001F
COMP_GOOD_TGL            0000077F R    03      EBCNAK                 = 0000003D
CON_RESP               = 0000000E              EBCNL                  = 00000015
CRB$L_INTD             = 00000024              EBCRVI                 = 0000007C
CRB$L_INTD2            = 00000048              EBCSLH                 = 00000061
DDB$L_DDT              = 0000000C              EBCSOH                 = 00000001
DEV$M_AVL                ********     X  02    EBCSP                  = 00000040
DEV$M_IDV                ********     X  02    EBCSTX                 = 00000002
DEV$M_ODV                ********     X  02    EBCSYN                 = 00000032
DLE_STX_CRC            = 0000018C              EBCWACK                = 0000006B
DPT$C_LENGTH           = 00000038              ERL$DEVICERR             ********     X  03
DPT$C_VERSION          = 00000004              ERR_RECORD               00000C3C R      03
DPT$INITAB               00000038 R    02      EXE$ABORTIO              ********     X  03
DPT$REINITAB             00000050 R    02      EXE$ALLOCBUF             ********     X  03
DPT$TAB                  00000000 R    02      EXE$BUFQUOPRC            ********     X  03
DSR_LOST               = 0000000B              EXE$FINISHIOC            ********     X  03
DUP_CSR_M_BYM          = 00008000              EXE$IOFORK               ********     X  03
DUP_CSR_M_CRI          = 00000200              EXE$QIODRVPKT            ********     X  03
DUP_CSR_M_CTS          = 00002000              EXE$READCHK              ********     X  03
DUP_CSR_M_DSR          = 00000200              EXE$SETMODE              ********     X  03
DUP_CSR_M_DTR          = 00000002              EXE$SNDEVMSG             ********     X  03
DUP_CSR_M_EOM          = 00000200              EXE$WRITECHK             ********     X  03
DUP_CSR_M_ETM          = 00001000              EXE$ZEROPARM             ********     X  03
DUP_CSR_M_HDX          = 00000008              FUNCTAB_LEN            = 00000040
DUP_CSR_M_RDN          = 00000080              GARBLED_RESP           = 0000000C
DUP_CSR_M_REN          = 00000010              GET_RESPONSE             00000613 R      03
DUP_CSR_M_RES          = 00000100              GOOD_TEXT                000009DB R      03
DUP_CSR_M_RIE          = 00000040              IDB$L_CSR              = 00000000
DUP_CSR_M_RTS          = 00000004              IDB$L_OWNER            = 00000004
DUP_CSR_M_SND          = 00000010              IDONE                    00000BCF R      03
DUP_CSR_M_SOM          = 00000100              IO$K_SRRUNOUT          = 00000000
DUP_CSR_M_SSY          = 00000100              IO$V_INTCLOCK          = 0000000C
DUP_CSR_M_STM          = 00000800              IO$V_LASTBLOCK         = 0000000A
DUP_CSR_M_TDN          = 00000080              IO$V_MAINTLOOP         = 00000009
DUP_CSR_M_TIE          = 00000040              IO$V_NOCTSWAIT         = 00000006
DUP_CSR_PARM           = 00000002              IO$V_NODSRWAIT         = 00000008
DUP_CSR_RDBF           = 00000002              IO$V_SHUTDOWN          = 00000007
DUP_CSR_TXCSR          = 00000004              IO$V_SLAVLOOP          = 00000007
DUP_CSR_TXDBF          = 00000006              IO$V_STARTUP           = 00000006
DYN$C_CRB              = 00000005              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$C_MEDIA                     = 00000038
IRP$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
------------------                           --------------
_$255$DUA28:[SYS.OBJ]LIB.MLB;1                           22
_$255$DUA28:[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
```

0122 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

XWDRIVER
LIS

DTGLOBAL
LIS

DTDEFINE
LIS

DTRMAIN
LIS

DTRAST
LIS

DTPREFIX
MAR

DTSDTR

DTCOMMON
LIS

DTRECV
MAP

DTSEND
MAP

DTMACROS
MAR