```
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**DDDRIVER

```
DDDDDDDD    DDDDDDDD    DDDDDDDD    RRRRRRRR      IIIIII    VV      VV  EEEEEEEEEE  RRRRRRRR
DDDDDDDD    DDDDDDDD    DDDDDDDD    RRRRRRRR      IIIIII    VV      VV  EEEEEEEEEE  RRRRRRRR
DD      DD  DD      DD  DD      DD  RR      RR      II      VV      VV  EE          RR      RR
DD      DD  DD      DD  DD      DD  RR      RR      II      VV      VV  EE          RR      RR
DD      DD  DD      DD  DD      DD  RR      RR      II      VV      VV  EE          RR      RR
DD      DD  DD      DD  DD      DD  RR      RR      II      VV      VV  EE          RR      RR
DD      DD  DD      DD  DD      DD  RRRRRRRR        II      VV      VV  EEEEEEEE    RRRRRRRR
DD      DD  DD      DD  DD      DD  RRRRRRRR        II      VV      VV  EEEEEEEE    RRRRRRRR
DD      DD  DD      DD  DD      DD  RR  RR          II      VV      VV  EE          RR  RR
DD      DD  DD      DD  DD      DD  RR  RR          II      VV      VV  EE          RR  RR
DD      DD  DD      DD  DD      DD  RR    RR        II        VV  VV    EE          RR    RR        ....
DD      DD  DD      DD  DD      DD  RR    RR        II        VV  VV    EE          RR    RR        ....
DDDDDDDD    DDDDDDDD    DDDDDDDD    RR      RR    IIIIII        VV      EEEEEEEEEE  RR      RR      ....
DDDDDDDD    DDDDDDDD    DDDDDDDD    RR      RR    IIIIII        VV      EEEEEEEEEE  RR      RR      ....

LL            IIIIII      SSSSSSSS
LL            IIIIII      SSSSSSSS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II        SSSSSS
LL              II        SSSSSS
LL              II              SS
LL              II              SS
LL              II              SS
LL              II              SS
LLLLLLLLLL    IIIIII    SSSSSSSS
LLLLLLLLLL    IIIIII    SSSSSSSS
```

Table of contents


```
    (1)      131      EXTERNAL DEFINITIONS
    (1)      153      LOCAL DEFINITONS
    (1)      330      LOCAL MACRO DEFINITIONS
    (1)      418      DRIVER PROLOGUE TABLE
    (1)      472      DRIVER DISPATCH TABLE
    (1)      492      FUNCTION DECISION TABLE
    (1)      590      TU58 UNIT INITIALIZATION
    (1)      680      INIT_UNIT
    (1)      772      SND_PKT_NOINT
    (1)      823      DUMP_RECVR_BUF
    (1)      867      STARTIO ROUTINE
    (1)     1228      COMMON EXIT ROUTINE
    (1)     1247      Controller error handler
    (1)     1279      TIMEOUT and POWERFAIL recovery
    (1)     1324      RESET_CONTROLLER
    (1)     1391      REGISTER DUMP ROUTINE
    (1)     1425      INTERRUPT SERVICE ROUTINE
    (1)     1465      LOCAL SUBROUTINES
    (1)     1506      GET_PACKET_C
    (1)     1636      SEND_PACKET_C
    (1)     1746      GET_PACKET_U
    (1)     1876      SEND_PACKET_U
```

```
0000        .TITLE  DDDRIVER - TU58 DC100 TAPE CARTRIDGE 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    ;
0000    ;++
0000    ;
0000    ; FACILITY:
0000    ;
0000    ;       VAX/VMS  TU58 DATA CARTRIDGE DRIVER
0000    ;
0000    ;
0000    ; ABSTRACT:
0000    ;
0000    ;       This module contains the tables and routines necessary to
0000    ;       perfrom all device-dependent processing of an I/O request
0000    ;       for the TU58 on a VAX/VMS operating system.  This device
0000    ;       operates using a RADIAL SERIAL PROTOCOL.  For a spec of the
0000    ;       TU58 and its RSP, see the "TU58 DECtape II USER'S GUIDE",
0000    ;       document # EK-0TU58-UG-001.
0000    ;
0000    ;       This driver is desinged to work with a DL-11 serial interface
0000    ;       via the UBA registers, or as a console device via the internal
0000    ;       processor registers.
0000    ;
0000    ; AUTHOR:
0000    ;
0000    ;       S. T. Jeffreys          31-OCT-1979     (with thanks to Mark Grossman)
0000    ;
0000    ; MODIFIED BY:
0000    ;
0000    ;       V03-007 RAS0300         Ron Schaefer            19-Jun-1984
0000    ;               Add DEV$M_NNM characteristic to DECHAR2 so that these
0000    ;               devices will have the "node$" prefix.
```

```
0000    ;
0000    ;       V03-006 PRD0044         Paul R DeStefano        10-Oct-1983
0000    ;               Made UNLOAD, AVAILABLE, and PACKACK legal functions and
0000    ;               added necessary code to handle these functions within
0000    ;               the driver.
0000    ;
0000    ;       V03-005 ROW0211         Ralph O. Weber          16-AUG-1983
0000    ;               Change device-dependent UCB definition base from UCB$W_BCR+2
0000    ;               to UCB$K_LCL_DISK_LENGTH.
0000    ;
0000    ;       V03-004 STJ3060         Steven T. Jeffreys      02-Mar-1983
0000    ;               - Do not use MRSP protocol when the TU58 is a console
0000    ;                 device on 11/750 processors.
0000    ;               - Check the DD_V_USEMRSP bit instead of DD_V_MRSP when
0000    ;                 deciding whether or not to use MRSP for this request.
0000    ;
0000    ;       V03-003 STJ48858        Steven T. Jeffreys      14-Sep-1982
0000    ;               - Add some sanity checks on the DATA packet byte count.
0000    ;
0000    ;       V03-002 KTA0100         Kerbey T. Altmann       07-Jun-1982
0000    ;               Add code to set UCB$L_MEDIA_ID.
0000    ;
0000    ;       V03-001 STJ0260         Steven T. Jeffreys      15-Apr-1982
0000    ;               - A slight change for 11/730 initialization code.
0000    ;                 Relax some of the sanity checking to allow for
0000    ;                 variations between processors.
0000    ;
0000    ;       V02-008 STJ0232         Steven T. Jeffreys      02-Mar-1982
0000    ;               - Set the number of retries per request down to 4 (from 8).
0000    ;               - Modify the reset_controller routine to always dump the
0000    ;                 reciever buffer immediately after clearing the BREAK.
0000    ;               - Modify dump_recvr_buf to return an error status if
0000    ;                 the DONE bit never gets set.
0000    ;               - Modify snd_pkt_noint to return an error status if
0000    ;                 the READY bit never gets set.
0000    ;
0000    ;       V02-007 STJ0107         Steven T. Jeffreys      19-Aug-1981
0000    ;               - Modified SND_PKT_NOINT to spin on the tranmitter ready bit.
0000    ;                 This is necessary for 11/750 console TU58s to init properly.
0000    ;               - Always return SS$_DATAOVERUN from GET_PACKET_C and GET_PACKET_U
0000    ;                 so that in the event of an error, R0 will contain the correct
0000    ;                 status code.  For non-error conditions, R0 is ingored.
0000    ;
0000    ;       V02-006 TCM0001         Trudy C. Matthews       29-Jul-1981
0000    ;               Changed all "7ZZ"s to "730"s.
0000    ;
0000    ;       V02-005 JLV0025         Jake VanNoy             11-Jun-1981
0000    ;               Added DEFUNITS to DPT.
0000    ;
0000    ;       V02-004 STJ0027         Steven  T. Jeffreys,    24-Mar-1981
0000    ;               - Modify the driver so that the device dependent status info
0000    ;                 is not returned in R1 (second longord of the IOSB).
0000    ;                 Register R1 will be zeroed before completing the I/O.
0000    ;               - Return SS$_DATAOVERUN for data over run errors.
0000    ;               - Return SS$_PARITY for checksum errors.
0000    ;               - Alter errorlogging scheme to log all device errors.
0000    ;                 Old scheme was to only log errors that aborted the request.
```

```
0000  ;                - Modify the controller error/reset routine so that it will
0000  ;                  return the original reason for the controller reset.
0000  ;                - Added support for multiple console TU58s. (eg. CSA1, CSA2)
0000  ;                - Added support for MRSP.
0000  ;                - Optimized interrupt service routines to minimize interrupt
0000  ;                  latentcy and processor usage.
0000  ;
0000  ;        V02-003 ACG0179         Andrew C. Goldstein,    23-Jul-1980  18:17
0000  ;                Fix ACP class code in DPT
0000  ;
0000  ;        V02-002 STJ0001         Steve Jeffreys,         15-July-1980
0000  ;                Fix access violation on IO$_SEARCH requests.
0000  ;
0000  ;--
0000
```

```
0000            .SBTTL  EXTERNAL DEFINITIONS
0000    ;
0000    ; EXTERNAL SYMBOLS
0000    ;
0000            $ADPDEF                         ;Define adapter control block
0000            $CRBDEF                         ;Define channel request block
0000            $DCDEF                          ;Define device class
0000            $DDBDEF                         ;Define device data block
0000            $DEVDEF                         ;Define device characteristics
0000            $DPTDEF                         ;Define driver prologue table
0000            $DYNDEF                         ;Define data structure types
0000            $EMBDEF                         ;Define error message buffer
0000            $IDBDEF                         ;Define interrupt data block
0000            $IODEF                          ;Define I/O function codes
0000            $IRPDEF                         ;Define I/O request packet
0000            $PRDEF                          ;Define processor registers
0000            $PTEDEF                         ;Page table entry definitions
0000            $SSDEF                          ;Define system status codes
0000            $UBADEF                         ;Define unibus adapter
0000            $UCBDEF                         ;Define urit control block
0000            $VECDEF                         ;Define interrupt vector block
```

```
                 0000            .SBTTL  LOCAL DEFINITONS
                 0000    ;
                 0000    ; Local symbols
                 0000    ;
                 0000
                 0000    ;
                 0000    ; TU58 register definitions
                 0000    ;
                 0000
                 0000            $DEFINI DD                              ;Start of register definitions
                 0000
                 0000    $DEF    DD_RCSR         .BLKW   1               ;Receiver status register
                 0002            _VIELD  DD_RCSR,0,<-                    ;Start of bit definitions
                 0002                    <,6>,-                          ;Unused bits
                 0002                    <RIE,,M>,-                      ;Receiver interrupt enable
                 0002                    <RDONE,,M>,-                    ;Receiver done
                 0002                    >                               ;
                 0002    $DEF    DD_RBUF         .BLKW   1               ;Receiver data buffer register
                 0004            _VIELD  DD_RBUF,0,<-                    ;Start of bit definitions
                 0004                    <,15>,-                         ;0-7 data , 8-14 unused
                 0004                    <ERROR,,M>,-                    ;Composite error bit
                 0004                    >
                 0004    $DEF    DD_XCSR         .BLKW   1               ;Transmitter status register
                 0006            _VIELD  DD_XCSR,0,<-                    ;Start of bit definitions
                 0006                    <BREAK,,M>,-                    ;Causes a framing error-break
                 0006                    <,5>,-                          ;Unused
                 0006                    <TIE,,M>,-                      ;Transmitter interrupt enable
                 0006                    <TRDY,,M>,-                     ;Transmitter ready
                 0006                    >                               ;
                 0006    $DEF    DD_XBUF         .BLKW   1               ;TRANSMITTER DATA BUFFER REGISTER
                 0008
                 0008            $DEFEND DD
                 0000
                 0000    ;
                 0000    ;       Device function codes.
                 0000    ;
00000001         0000    DD_CODE_DATACHECK = 1                           ;Datacheck READ/WRITE modifier
00000002         0000    DD_CODE_READ      = 2                           ;Read request
00000003         0000    DD_CODE_WRITE     = 3                           ;Write request
00000005         0000    DD_CODE_SEARCH    = 5                           ;Search Request
00000007         0000    DD_CODE_DIAG      = 7                           ;Diagnostic request
0000000A         0000    DD_CODE_DETECT    = 10                          ;Detect TU58 type
                 0000    ;
                 0000    ;       Radial serial protocol flags
                 0000    ;
00000001         0000    DD_FLAG_DATA      = 1                           ;Data packet flag
00000002         0000    DD_FLAG_CNTL      = 2                           ;Command packet flag
00000004         0000    DD_FLAG_INIT      = 4                           ;Initialize controller flag
00000010         0000    DD_FLAG_CONT      = 16                          ;Continue flag
00000013         0000    DD_FLAG_XOFF      = 19                          ;Suspend device action flag
00000010         0000    ACK_CHAR          = DD_FLAG_CONT                ;ACK character for MRSP
00000003         0000    MRSP_V_SWITCH     = 3                           ;Switch vield for MRSP
00000008         0000    MRSP_M_SWITCH     = 8                           ;Switch mask for MRSP
00000A02         0000    CMD_PKT_HDR       = ^X0A02                      ;Command packet header switch
000A0A02         0000    DTECT_PKT_HDR     = ^X000A0A02                  ;Special command packet header
00000A0C         0000    DTECT_PKT_CHKSM   = ^X0A0C                      ;Special command packet checksum
00000404         0000    TWO_INITS         = ^X0404                      ;Two INIT characters
```

```
                0000  ;
                0000  ;       Device dependent status codes returned in the END PACKET.
                0000  ;
    00000000    0000  DD_STS_NORMAL  = 0                        ;Normal success
    00000001    0000  DD_STS_RETRY   = 1                        ;Success but with retries
    FFFFFFFF    0000  DD_STS_BADDIAG = -1                       ;Failed self test (diagnostics)
    FFFFFFFE    0000  DD_STS_PARTOP  = -2                       ;Partial operation (end of tape)
    FFFFFFF8    0000  DD_STS_BADUNIT = -8                       ;Bad unit #
    FFFFFFF7    0000  DD_STS_NOCART  = -9                       ;No cartridge
    FFFFFFF5    0000  DD_STS_WRTLCK  = -11                      ;Write locked
    FFFFFFEF    0000  DD_STS_DATACHK = -17                      ;Data check error
    FFFFFFE0    0000  DD_STS_BLKNF   = -32                      ;Seek error (block not found)
    FFFFFFDF    0000  DD_STS_MTRSTOP = -33                      ;Motor stopped
    FFFFFFD0    0000  DD_STS_BADOPCD = -48                      ;Bad op code
    FFFFFFC9    0000  DD_STS_BADRCD  = -55                      ;Bad record #
                0000  ;
                0000  ;       The remainder of the status codes are internal
                0000  ;       to this driver, and are used only for debugging.
                0000  ;
    FFFFFFC7    0000  DD_STS_BADINIT = -57                      ;Failed init sequence
    FFFFFFC6    0000  DD_STS_BADCKSM = -58                      ;Failed checksum comparison
    FFFFFFC5    0000  DD_STS_DATAOVR = -59                      ;Data over-run
                0000  ;
                0000  ;       Define various and sundry symbols.
                0000  ;
    0000000E    0000  DD_CP_LEN        = 14                     ;Command packet buffer length in bytes
    0000000E    0000  DD_EP_LEN        = 14                     ;End packet buffer length in bytes
    00000004    0000  DD_NUM_REGS      = 4;                     ;Number of device registers
    0000002C    0000  DD_ERRBUF_LEN    = <<DD_NUM_REGS * 4> + DD_CP_LEN + DD_EP_LEN>
    0000000B    0000  DD_ERRBUF_LEN_L  = 11                     ;Length of error buffer in longwords
    0000000A    0000  DD_CP_BCNT       = 10                     ;Command packet byte count
    00000002    0000  DD_CHKSM_LEN     = 2                      ;Packet checksum length in bytes
    00000080    0000  DD_MAX_PKT_BCNT  = 128                    ;Maximum data packet byte count
    00000001    0000  DD_DPBCNT        = 1                      ;Offset to data packet byte count
    00000002    0000  DD_DATA          = 2                      ;Offset to data w/i data packet
    00000084    0000  DD_DPBUF_LEN     = <DD_MAX_PKT_BCNT + 4>  ;Data packet buffer length
    00000034    0000  LONG_WAIT        = 52                     ;# of seconds before read timeout
    00000002    0000  SHORT_WAIT       = 2                      ;# of seconds to wait after first byte read
    00000002    0000  WRITE_WAIT       = 2                      ;# of seconds before write timeout
    00000017    0000  HIGH_DIPL        = 23                     ;Special DIPL for COMET
    00004E20    0000  MAXCHRTIM        = 20000                  ;Maximum character time, in 10 u-sec
                0000                                            ;units, assuming a 50 baud line.
                0000  ;
                0000  ;       Define the bits for device dependent status word in the UCB.
                0000  ;       The field that contains these bits is UCB$W_TU58STS.
                0000  ;
                0000          _VIELD  DD,0,<-                   ;Start of bit definitions
                0000                  <CONSOLE,,M>,-            ;Console device
                0000                  <PASS1,,M>,-              ;Determines where to call buffer routines
                0000                  <MRSP,,M>,-               ;This TU58 speaks MRSP
                0000                  <USEMRSP,,M>,-            ;Use MRSP for this operation
                0000                  >                         ;End of bit definitions
```

```
          0000  ;
          0000  ;       Define extensions to UCB
          0000  ;
          0000          $DEFINI UCB                     ;Start of UCB extensions
          0000
000000CC  0000  .=UCB$K_LCL_DISK_LENGTH                 ;Begin definitions at end of UCB
          00CC  $DEF    UCB$L_DD_BUFWIN .BLKL   1       ;Window to user's buffer
          00D0  $DEF    UCB$L_DD_DPTR   .BLKL   1       ;Pointer to data packet buffer
          00D4  $DEF    UCB$L_DD_MSGPKT .BLKL   1       ;Current message packet address
          00D8  $DEF    UCB$W_DD_MSGLEN .BLKW   1       ;Current message packet length
          00DA  ;
          00DA  ; Define buffer for device registers.
          00DA  ;
          00DA  $DEF    UCB$K_DD_REGBUF                 ;Register buffer address
          00DA  $DEF    UCB$W_DD_XCSR   .BLKW   1       ;Transmitter CSR
          00DC  $DEF    UCB$W_DD_XBUF   .BLKW   1       ;Transmitter buffer
          00DE  $DEF    UCB$W_DD_RCSR   .BLKW   1       ;Receiver CSR
          00E0  $DEF    UCB$W_DD_RBUF   .BLKW   1       ;Receiver buffer
          00E2  ;
          00E2  ; Define the Command Packet buffer.
          00E2  ;
          00E2  $DEF    UCB$K_DD_CPADDR                 ;Command packet buffer address
          00E2  $DEF    UCB$B_DD_CPFLAG .BLKB   1       ;Command packet flag byte
          00E3  $DEF    UCB$B_DD_CPBCNT .BLKB   1       ;Command packet byte count
          00E4  $DEF    UCB$B_DD_OPCODE .BLKB   1       ;Op code
          00E5  $DEF    UCB$B_DD_OPMOD  .BLKB   1       ;Op code modifier
          00E6  $DEF    UCB$B_DD_UNIT   .BLKB   1       ;Unit number
          00E7  $DEF    UCB$B_DD_SWITCH .BLKB   1       ;Switches
          00E8  $DEF    UCB$B_DD_SEQL   .BLKB   1       ;Sequence number - low byte
          00E9  $DEF    UCB$B_DD_SEQH   .BLKB   1       ;Sequence number - high byte
          00EA  $DEF    UCB$B_DD_BCNTL  .BLKB   1       ;Byte count - low byte
          00EB  $DEF    UCB$B_DD_BCNTH  .BLKB   1       ;Byte count - high byte
          00EC  $DEF    UCB$B_DD_BLKL   .BLKB   1       ;Block number - low byte
          00ED  $DEF    UCB$B_DD_BLKH   .BLKB   1       ;Block number - high byte
          00EE  $DEF    UCB$W_DD_CPCKSM .BLKW   1       ;Command packet checksum
          00F0  ;
          00F0  ;       Define the End Packet buffer
          00F0  ;
          00F0  $DEF    UCB$K_DD_EPADDR                 ;End packet buffer address
          00F0  $DEF    UCB$B_DD_EPFLAG .BLKB   1       ;End packet flag
          00F1  $DEF    UCB$B_DD_EPBCNT .BLKB   1       ;Packet byte count
          00F2  $DEF    UCB$B_DD_EPCODE .BLKB   1       ;Op code
          00F3  $DEF    UCB$B_DD_EPSTS  .BLKB   1       ;Success code
          00F4  $DEF    UCB$B_DD_EPUNIT .BLKB   4       ;Unit # (next 3 bytes not used)
          00F8  $DEF    UCB$W_DD_ABCNT  .BLKW   1       ;Actual byte count (bytes transfered)
          00FA  $DEF    UCB$W_DD_SMSTS  .BLKW   1       ;Summary status (more status info)
          00FC  $DEF    UCB$W_DD_EPCKSM .BLKW   1       ;Packet checksum
          00FE
          00FE  $DEF    UCB$K_DD_UCBLEN                 ;Last loc. in UCB
          00FE          $DEFEND UCB
          0000
          0000  ;
          0000  ; The following are psuedonyms for existing but unused fields in the UCB.
          0000  ;
000000CA  0000  UCB$W_TU58STS   = UCB$B_OFFNDX          ;Device specific status
          0000                                          ;This field also uses UCB$B_OFFRTC
          0000  ;
```

```
0000  ; Note that the following fields in the UCB are currently not
0000  ; used by the TU58 driver, and may be used as the need arises.
0000  ;
0000  ; UCB$B_FEX
0000  ; UCB$B_CEX
0000  ; UCB$x_xxx              (unused, WORD sized field contiguous with UCB$B_CEX)
0000  ; UCB$W_EC1
0000  ; UCB$W_EC2
```

```
0000            .SBTTL  LOCAL MACRO DEFINITIONS
0000    ;
0000    ;       LOCAL MACRO DEFINITIONS
0000    ;
0000
0000    ;
0000    ; GET_PACKET    - Get a packet of data from the TU58
0000    ;
0000    .MACRO  GET_PACKET      ?L1,?L2
0000            BBC     #DD_V_CONSOLE,-                 ;Branch if not a console device
0000                    UCB$W_TU58STS(R5),L1            ;
0000            BSBW    GET_PACKET_C                    ;Get packet (console)
0000            BRB     L2                              ;Rejoin common code
0000    L1:     BSBW    GET_PACKET_U                    ;Get packet (unibus)
0000    L2:
0000    .ENDM   GET_PACKET
0000
0000    ;
0000    ; SEND_PACKET   - Send a packet of data to the TU58
0000    ;
0000    .MACRO  SEND_PACKET     ?L1,?L2
0000            BBC     #DD_V_CONSOLE,-                 ;Branch if not a console device
0000                    UCB$W_TU58STS(R5),L1            ;
0000            BSBW    SEND_PACKET_C                   ;Send packet (console)
0000            BRB     L2                              ;Rejoin common code
0000    L1:     BSBW    SEND_PACKET_U                   ;Send packet (unibus)
0000    L2:
0000    .ENDM   SEND_PACKET
0000
0000
0000    ;
0000    ; GET_BYTE      - Get a byte of data from the device.
0000    ;
0000    .MACRO  GET_BYTE
0000            MOVAB   UCB$B_SPR(R5),-                 ;Get buffer address
0000                    UCB$L_DD_MSGPKT(R5)             ;
0000            MOVW    #1,UCB$W_DD_MSGLEN(R5)          ;Get buffer length
0000            GET_PACKET                              ;Get the data
0000    .ENDM   GET_BYTE
0000
0000    ;
0000    ; SEND_BYTE     - Send a byte of data to the device.
0000    ;
0000    .MACRO  SEND_BYTE
0000            MOVAB   UCB$B_SPR(R5),-                 ;Get buffer address
0000                    UCB$L_DD_MSGPKT(R5)             ;
0000            MOVW    #1,UCB$W_DD_MSGLEN(R5)          ;Get buffer length
0000            SEND_PACKET                             ;Send data
0000    .ENDM   SEND_BYTE
0000
0000
0000    ;
0000    ; MOVTOUSER
0000    ;
0000    ; Move a string of data to the user's buffer.
0000    ; Assume all inputs have been set up.
0000    ; The different entry points are needed to avoid problems
```

```
0000     ; with page  aligned user buffers.
0000     ;
0000     .MACRO  MOVTOUSER ?L1,?L2
0000             BBSC    #DD_V_PASS1,-                  ;Is this the first data transfer?
0000                     UCB$W_TU58STS(R5),L1           ;Branch if yes
0000             MOVL    UCB$L_DD_BUFWIN(R5),R0         ;Get window into user's buffer
0000             JSB     G^IOC$MOVTOUSER2               ;Move data
0000             BRB     L2                             ;Rejoin common code
0000     L1:     JSB     G^IOC$MOVTOUSER                ;Move data
0000     L2:     MOVL    R0,UCB$L_DD_BUFWIN(R5)         ;Save new window
0000     .ENDM   MOVTOUSER
0000
0000
0000     ;
0000     ; MOVFROMUSER
0000     ;
0000     ; Move a string of data from the user's buffer.
0000     ; Assume all inputs have been set up.
0000     ; The different entry points are needed to avoid problems
0000     ; with page  aligned user buffers.
0000     ;
0000     .MACRO  MOVFROMUSER ?L1,?L2
0000             BBSC    #DD_V_PASS1,-                  ;Is this the first data transfer?
0000                     UCB$W_TU58STS(R5),L1           ;Branch if yes
0000             MOVL    UCB$L_DD_BUFWIN(R5),R0         ;Get window into user's buffer
0000             JSB     G^IOC$MOVFRUSER2               ;Move data
0000             BRB     L2                             ;Rejoin common code
0000     L1:     JSB     G^IOC$MOVFRUSER                ;Move data
0000     L2:     MOVL    R0,UCB$L_DD_BUFWIN(R5)         ;Save new window
0000     .ENDM   MOVFROMUSER
```

```
0000          .SBTTL  DRIVER PROLOGUE TABLE
0000     ;
0000     ; DRIVER PROLOGUE TABLE
0000     ;
0000     ;        The DPT describes driver parameters and I/O databse fields
0000     ;        that are initialized during driver LOADing and RELOADing.
0000     ;
0000     DPTAB    -                                    ;DPT creation macro
0000              END=DD_END,-                         ;End of driver
0000              ADAPTER=UBA,-                        ;Adapter type = UNIBUS
0000              FLAGS=DPT$M_SVP,-                    ;System page table entry required
0000              UCBSIZE=<UCB$K_DD_UCBLEN>,-          ;Length of UCB
0000              NAME=DDDRIVER,-                      ;Driver name
0000              DEFUNITS=2                           ;Number of units to autoconfigure.
0038
0038     DPT_STORE INIT                                ;START CONTROL BLOCK INIT VALUES
0038     DPT_STORE DDB,DDB$L_ACPD,L,<^A\F11\>          ;Default ACP name
003F     DPT_STORE DDB,DDB$L_ACPD+3,B,DDB$K_TAPE       ;ACP class
0043     DPT_STORE UCB,UCB$B_FIPL,B,8                  ;Fork IPL
0047
0047     DPT_STORE UCB,UCB$L_DEVCHAR,L,-               ;DEVICE CHARACTERISTICS
0047                      <DEV$M_FOD-                  ;- File oriented
0047                      !DEV$M_DIR-                  ;- Directory structured
0047                      !DEV$M_AVL-                  ;- Available
0047                      !DEV$M_ELG-                  ;- Error logging
0047                      !DEV$M_SHR-                  ;- Shareable
0047                      !DEV$M_IDV-                  ;- Input device
0047                      !DEV$M_ODV-                  ;- Output device
0047                      !DEV$M_RND>                  ;- Random access
004E     DPT_STORE UCB,UCB$L_DEVCHAR2,L,-              ; DEVICE CHARACTERISTICS
004E                      <DEV$M_NNM>                  ; Prefix name with "node$"
0055
0055     DPT_STORE UCB,UCB$B_DEVCLASS,B,DC$_DISK       ;Device class
0059     DPT_STORE UCB,UCB$B_DEVTYPE,B,DT$_TU58        ;Device type
005D     DPT_STORE UCB,UCB$W_DEVBUFSIZ,W,512           ;Default buffer size
0062     DPT_STORE UCB,UCB$B_SECTORS,B,128             ;Number of sectors/track
0066     DPT_STORE UCB,UCB$B_TRACKS,B,4                ;Number of tracks/cylinder
006A     DPT_STORE UCB,UCB$W_CYLINDERS,W,1             ;Number of cylinders
006F     DPT_STORE UCB,UCB$B_DIPL,B,20                 ;Device IPL
0073     DPT_STORE UCB,UCB$B_ERTMAX,B,4                ;Maximum error retry count
0077     DPT_STORE UCB,UCB$L_MAXBLOCK,L,512            ;Maximum # of blocks
007E     DPT_STORE UCB,UCB$L_MEDIA_ID,-
007E              L,<^X2129503A>                       ;Media ident "DD TU58"
0085     DPT_STORE REINIT                              ;START CONTROL BLOCK RE-INIT VALUES
0085     DPT_STORE CRB,CRB$L_INTD+4,D,DD_INT           ;Interrupt service routine address
008A     DPT_STORE CRB,CRB$L_INTD2+4,D,DD_INT          ;Interrupt service routine address
008F     DPT_STORE CRB,-                               ;Unit init
008F              CRB$L_INTD+VEC$L_UNITINIT,D,-        ;
008F              DD_UNIT_INIT                         ;
0094     DPT_STORE DDB,DDB$L_DDT,D,DD$DDT              ;DDT address
0099
0099     DPT_STORE END                                 ;End of prologue table
0000
```

```
0000            .SBTTL  DRIVER DISPATCH TABLE
0000    ;
0000    ;DRIVER DISPATCH TABLE
0000    ;
0000    ;       The DDT list entry points for the driver subroutines which are
0000    ;       called by the operating system.
0000    ;
0000
0000            DPTAB   -                               ;DDT creation macro
0000                    DEVNAM=DD,-                     ;Generic device name
0000                    START=DD_STARTIO,-              ;STARTIO routine address
0000                    UNSOLIC=0,-                     ;Unsolicited interrupt handler
0000                    FUNCTB=DD_FUNCTABLE,-           ;FDT address
0000                    CANCEL=0,-                      ;CANCEL I/O handler
0000                    REGDMP=DD_REGDUMP,-             ;Register dump routine address
0000                    DIAGBF=<DD_ERRBUF_LEN+<<1+3+5>*4>>,- ;Byte size of diab buffer
0000                    ERLGBF=<DD_ERRBUF_LEN+<1*4>+<EMB$L_DV_REGSAV>>  ;Byte size of
0038                                                                    ;errorlog buffer
0038
```

```
0038             .SBTTL  FUNCTION DECISION TABLE
0038    ;
0038    ; FUNCTION DECISION TABLE
0038    ;
0038    ;        The FDT lists valid function codes, specifies which codes
0038    ;        are buffered, and designates subroutines to perform I/O
0038    ;        preprocessing for particular functions.
0038    ;
0038
0038    DD_FUNCTABLE:
0038
0038             FUNCTAB,-                            ;LIST LEGAL FUNCTIONS
0038                     <DIAGNOSE,-                  ;RUN INTERNAL DIAGNOSTICS
0038                     SEARCH,-                     ;SEARCH FOR BLOCK
0038                     SENSECHAR,-                  ;SENSE CHARACTERISTICS
0038                     SETCHAR,-                    ;SET CHARACTERISTICS
0038                     SENSEMODE,-                  ;SENSE MODE
0038                     SETMODE,-                    ;SET MODE
0038                     READLBLK,-                   ;READ LOGICAL BLOCK
0038                     WRITELBLK,-                  ;WRITE LOGICAL BLOCK
0038                     READPBLK,-                   ;READ PHYSICAL BLOCK
0038                     WRITEPBLK,-                  ;WRITE PHYSICAL BLOCK
0038                     READVBLK,-                   ;READ VIRTUAL BLOCK
0038                     WRITEVBLK,-                  ;WRITE VIRTUAL BLOCK
0038                     ACCESS,-                     ;ACCESS FILE / FIND DIRECTORY ENTRY
0038                     ACPCONTROL,-                 ;ACP CONTROL FUNCTION
0038                     CREATE,-                     ;CREATE FILE AND /OR DIRECTORY ENTRY
0038                     DEACCESS,-                   ;DEACCESS FILE
0038                     DELETE,-                     ;DELETE FILE AND / OR DIRECTORY ENTRY
0038                     MODIFY,-                     ;MODIFY FILE ATTRIBUTES
0038                     MOUNT,-                      ;MOUNT VOLUME
0038                     UNLOAD,-                     ;UNLOAD VOLUME
0038                     AVAILABLE,-                  ;AVAILABLE
0038                     PACKACK-                     ;PACK ACKNOWLEDGE
0038                     >
0040
0040             FUNCTAB,-                            ;BUFFERED FUNCTIONS
0040                     <SENSECHAR,-                 ;SENSE CHARACTERISTICS
0040                     SETCHAR,-                    ;SET CHARACTERISTICS
0040                     SENSEMODE,-                  ;SENSE MODE
0040                     SETMODE,-                    ;SET MODE
0040                     ACCESS,-                     ;ACCESS FILE / FIND DIRECTORY ENTRY
0040                     ACPCONTROL,-                 ;ACP CONTROL FUNCTION
0040                     CREATE,-                     ;CREATE FILE AND /OR DIRECTORY ENTRY
0040                     DEACCESS,-                   ;DEACCESS FILE
0040                     DELETE,-                     ;DELETE FILE AND / OR DIRECTORY ENTRY
0040                     MODIFY,-                     ;MODIFY FILE ATTRIBUTES
0040                     MOUNT,-                      ;MOUNT VOLUME
0040                     UNLOAD,-                     ;UNLOAD VOLUME
0040                     AVAILABLE,-                  ;AVAILABLE
0040                     PACKACK-                     ;PACK ACKNOWLEDGE
0040                     >
0048             FUNCTAB +ACP$READBLK,-               ;READ FUNCTIONS
0048                     <READLBLK,-
0048                     READPBLK,-
0048                     READVBLK,-
0048                     >
```

```
0054   FUNCTAB +ACP$WRITEBLK,-           ;WRITE FUNCTIONS
0054           <WRITELBLK,-
0054           WRITEPBLK,-
0054           WRITEVBLK,-
0054           >
0060   FUNCTAB +ACP$ACCESS,-
0060           <ACCESS,-
0060           CREATE-
0060           >
006C   FUNCTAB +ACP$DEACCESS,-
006C           <DEACCESS-
006C           >
0078   FUNCTAB +ACP$MODIFY,-
0078           <ACPCONTROL,-
0078           DELETE,-
0078           MODIFY-
0078           >
0084   FUNCTAB +EXE$LCLDSKVALID,-        ;LOCAL DISK VALID FUNCTIONS
0084           <UNLOAD,-                 ;UNLOAD VOLUME
0084           AVAILABLE,-               ;AVAILABLE
0084           PACKACK,-                 ;PACK ACKNOWLEDGE
0084           >
0090   FUNCTAB +EXE$ONEPARM,-            ;ONE PARAMETER FUNCTIONS
0090           <SEARCH,-
0090           >
009C   FUNCTAB +EXE$ZEROPARM,-           ;ZERO PARAMETER FUNCTIONS
009C           <DIAGNOSE,-               ;DIAGNOSE THE DRIVE
009C           >
00A8   FUNCTAB +ACP$MOUNT,-
00A8           <MOUNT-
00A8           >
00B4   FUNCTAB +EXE$SENSEMODE,-
00B4           <SENSECHAR,-
00B4           SENSEMODE-
00B4           >
00C0   FUNCTAB +EXE$SETCHAR,-
00C0           <SETCHAR,-
00C0           SETMODE-
00C0           >
00CC
```

```
                             00CC        .SBTTL  TU58 UNIT INITIALIZATION
                             00CC  ;++
                             00CC  ; DD_UNIT_INIT - TU58 TAPE CARTRIDGE UNIT INITIALIZATION
                             00CC  ;
                             00CC  ; The operating system calls this routine at system startup,
                             00CC  ; during driver loading and during recovery from power failure.
                             00CC  ;
                             00CC  ; INPUTS:
                             00CC  ;
                             00CC  ;                 R4 = ADDRESS OF CSR
                             00CC  ;                 R5 = ADDRESS OF UNIT CONTROL BLOCK
                             00CC  ;
                             00CC  ;         All interrupts are locked out.
                             00CC  ;
                             00CC  ; OUTPUTS:
                             00CC  ;
                             00CC  ;         Allocate a data packet buffer out of nonpaged pool, if the buffer
                             00CC  ;         has not already been allocated.  This is done to reduce the size
                             00CC  ;         of the UCB.  If the buffer cannot be allocated, exit immediately.
                             00CC  ;         In addition to the buffer itself, a 12 byte header must be allocated,
                             00CC  ;         and certain fields be initilaized so that the buffer can be self-
                             00CC  ;         identifying.
                             00CC  ;
                             00CC  ;         The unit status is set to online and a determination is made
                             00CC  ;         as to whether the unit interfaces via unibus adapter registers(UBA)
                             00CC  ;         or internal processor registers(IPR). If the unit uses IPR's, then
                             00CC  ;         it is a console device.  The device is a console device if
                             00CC  ;         the CSR address is not a system address.  All system addresses
                             00CC  ;         have bit 31 = 1, which makes them appear negative.
                             00CC  ;
                             00CC  ; Side Effects:
                             00CC  ;
                             00CC  ;         If the processor type is unknown, or the sector buffer could not
                             00CC  ;         be allocated, mark the device (UCB) offline to prevent it use.
                             00CC  ;--
                             00CC  DD_UNIT_INIT:                                   ;TU58 controller intialization
                             00CC  ;
                             00CC  ;         Allocate the buffer if it has not been allocated yet.
                             00CC  ;
             00CA C5    B4   00CC        CLRW    UCB$W_TU58STS(R5)               ;Reset device status flags
                  04    E3   00D0        BBCS    #UCB$V_ONLINE,-                 ;Set device online
             00 64 A5        00D2                UCB$W_STS(R5),10$               ;
             00D0 C5    D5   00D5  10$:  TSTL    UCB$L_DD_DPTR(R5)               ;Is pointer to buffer nonzero?
                  2F    12   00D9        BNEQ    20$                             ;Branch if yes
                  54    DD   00DB        PUSHL   R4                              ;Save R4
    54    00000000'GF   DE   00DD        MOVAL   G^EXE$GL_NONPAGED,R4            ;Put address of nonpaged pool
                             00E4                                                ;list head in R4
                  64    DD   00E4        PUSHL   (R4)                            ;Save IPL in pool list head
              64  12    DB   00E6        MFPR    #PR$_IPL,(R4)                   ;Save IPL in pool list head
           51  90 8F    9A   00E9        MOVZBL  #<DD_DPBUF_LEN+12>,R1           ;Set buffer length in R1
          00000000'GF   16   00ED        JSB     G^EXE$ALONONPAGED               ;Allocate data packet buffer
                  64  8ED0   00F3        POPL    (R4)                            ;Restore IPL in list head
                  54  8ED0   00F6        POPL    R4                              ;Restore R4
              27  50    E9   00F9        BLBC    R0,13$                          ;Branch if error
           08 A2  51    B0   00FC        MOVW    R1,8(R2)                        ;Put size into buffer header
           0A A2  13    90   0100        MOVB    #DYN$C_BUFIO,10(R2)             ;Put type into buffer header
      00D0 C5  52  0C   C1   0104        ADDL3   #12,R2,-                        ;Save pointer to start of actual
```

```
                          010A              UCB$L_DD_DPTR(R5)          ;data packet buffer.
                          010A      ;
                          010A      ;       Check the CSR address to determine if this is a console or unibus
                          010A      ;       TU58.  Unibus TU58s will have a CSR address that is a system
                          010A      ;       address, and the number will appear negative.  Console TU58s will
                          010A      ;       have a CSR address that is actually a processor internal register
                          010A      ;       number, and will appear to be a positive number.
                          010A      ;
                54   D5   010A      20$:    TSTL    R4                          ;Test CSR address
                15   13   010C              BEQL    13$                         ;Sanity check
                23   19   010E              BLSS    50$                         ;Branch if unibus TU58
                          0110      ;
                          0110      ;       The device is a console TU58.  Mark it as such.
                          0110      ;       Determine the CPU type and set the device IPL field
                          0110      ;       in the UCB accordingly.
                          0110      ;
 50     00000000'GF  90   0110              MOVB    G^EXE$GB_CPUTYPE,R0         ;Get CPU type code
                0A   15   0117              BLEQ    13$                         ;Sanity check
           50   03   91   0119              CMPB    #PR$_SID_TYP730,R0          ;Is this a 730 cpu?
                0F   13   011C              BEQL    40$                         ;Branch if yes
           50   02   91   011E              CMPB    #PR$_SID_TYP750,R0          ;Is this a 750 cpu?
                06   13   0121              BEQL    30$                         ;Branch if yes
                04   E4   0123      13$:    BBSC    #UCB$V_ONLINE,-             ;Set device offline
          00 64 A5        0125                      UCB$W_STS(R5),14$           ;
                     05   0128      14$:    RSB                                 ;Just in case
                17   90   0129      30$:    MOVB    #HIGH_DIPL,-                ;Set high device IPL for 750 cpu
             5E A5        012B                      UCB$B_DIPL(R5)              ;
                00   E2   012D      40$:    BBSS    #DD_V_CONSOLE,-             ; Mark this unit as a console device
        00 00CA C5        012F                      UCB$W_TU58STS(R5),50$       ;
                01   10   0133      50$:    BSBB    INIT_UNIT                   ;Do second half of unit initialization
                          0135      ;       BLBC    R0,13$                      ;Branch if error
                     05   0135              RSB                                 ;Return
```

```
                           0136          .SBTTL  INIT_UNIT
                           0136  ;++
                           0136  ; INIT_UNIT
                           0136  ;
                           0136  ; This routine is the second half of the TU58 unit initialization
                           0136  ; routine.  This routine will initialize the controller and determine
                           0136  ; if the controller speaks MRSP.  The algorithm for determining the
                           0136  ; TU58 controller type is given below.
                           0136  ;
                           0136  ; Determining the TU58 controller type:
                           0136  ;
                           0136  ;        Initialize the controller, and then send it a command packet with
                           0136  ;        the flow control bit set (MRSP_M_SWITCH in the SWITCH field of the
                           0136  ;        command packet), and get the first byte of the controller's response.
                           0136  ;        If the controller speaks MRSP, the response will be 02 (which is the
                           0136  ;        first byte of an end packet).  If the controller does not speak MRSP,
                           0136  ;        then the first byte will be 01 (first byte of a data packet).  Note
                           0136  ;        that a data overrun condition will also indicate that the controller
                           0136  ;        does not speak MRSP.  If an overrun condition did not occur, and the
                           0136  ;        response was not 01 or 02, then assume that there is a hard controller
                           0136  ;        error, and exit with the appropriate error status.  After getting the
                           0136  ;        response, ignore the rest of the data sent by the TU58 and reinitialize
                           0136  ;        the controller.
                           0136  ;
                           0136  ; Input:
                           0136  ;        IPL       = IPL$_POWER
                           0136  ;        R0 .. R3 = scratch
                           0136  ;        R4        = CSR address
                           0136  ;        R5        = UCB address
                           0136  ;
                           0136  ; Output:
                           0136  ;        R0 is even if error, odd if success
                           0136  ;--
                           0136
                           0136  INIT_UNIT:                                     ;Second half of unit initialization
            047F    30     0136          BSBW    RESET_CONTROLLER               ;Initialize the controller
           68 50    E9     0139          BLBC    R0,13$                         ;Branch if error
                           013C  ;
                           013C  ;       Build the special command packet used to detect the controller type.
                           013C  ;       Note that since the only variables in the command packet will be the
                           013C  ;       device unit number (which will also cause the checksum to vary) the
                           013C  ;       command packet can be built quickly, and the checksum computed easily.
                           013C  ;
     52    00E2 C5    DE     013C          MOVAL   UCB$K_DD_CPADDR(R5),R2         ;Get address of command packet buffer
82    000A0A02 8F    D0     0141          MOVL    #DTECT_PKT_HDR,(R2)+           ;Set FLAG, PKT_BCNT, OPCODE, and OPMOD
     50      54 A5    3C     0148          MOVZWL  UCB$W_UNIT(R5),R0              ;Get device unit #
        00 50    0B    E2     014C          BBSS    #<MRSP_V_SWITCH+8>,R0,10$;Set MRSP switch bit to prevent data
                           0150                                                 ;overruns on 730 processors.
              00    E1     0150  10$:    BBC     #DD_V_CONSOLE,-                ;Branch if not console device
         02 00CA C5          0152                  UCB$W_TU58STS(R5),20$          ;
              50    B7     0156          DECW    R0                             ;Decrement the unit #
           82 50    D0     0158  20$:    MOVL    R0,(R2)+                       ;Set the UNIT, SWITCH, and SEQUENCE #
              82    D4     015B          CLRL    (R2)+                          ;Clear TRANSFER COUNT and BLOCK #
62    50    0A0C 8F    A1     015D          ADDW3   #DTECT_PKT_CHKSM,R0,(R2);Compute and set the CHECKSUM
                           0163  ;
                           0163  ;       Send the command packet to the controller.
                           0163  ;
```

```
52   00D4 C5     9E  0163           MOVAB   UCB$L_DD_MSGPKT(R5),R2   ;Get address of packet pointer
82   00E2 C5     DE  0168           MOVAL   UCB$K_DD_CPADDR(R5),(R2)+;Get command packet address
        62    0E     B0  016D           MOVW    #DD_CP_LEN,(R2)          ;Get command packet length
              40     10  0170           BSBB    SND_PKT_NOINT            ;Send command packet
           2F 50     E9  0172           BLBC    R0,13$                   ;Branch if error
                         0175       ;
                         0175       ;       Get the controller's response.  The response
                         0175       ;       (in R1) will determine the controller type.
                         0175       ;
                         0175
            009E     30  0175           BSBW    DUMP_RECVR_BUF           ;Get controller response
           29 50     E9  0178           BLBC    R0,13$                   ;Branch if error
     1F 51    0F     E0  017B           BBS     #DD_RBUF_V_ERROR,R1,30$  ;Branch if data overrun
        51    01     91  017F           CMPB    #DD_FLAG_DATA,R1         ;Was it a data packet flag?
              1A     13  0182           BEQL    30$                      ;Branch if equal
        51    02     91  0184           CMPB    #DD_FLAG_CNTL,R1         ;Was it a command packet flag?
              1B     12  0187           BNEQ    13$                      ;Branch if not
   00000000'GF       91  0189           CMPB    G^EXE$GB_CPUTYPE,-       ;Is this a 750 cpu?
              02         018F                   #PR$_SID_TYP750          ;
              06     12  0190           BNEQ    25$                      ;Branch if not
              00     E0  0192           BBS     #DD_V_CONSOLE,-          ;Branch if 11/750 console device
      06 00CA C5         0194                   UCB$W_TU58STS(R5),30$    ; (ie. do not set the MRSP bit)
              02     E2  0198   25$:    BBSS    #DD_V_MRSP,-             ;Set MRSP bit
      00 00CA C5         019A                   UCB$W_TU58STS(R5),30$    ;
            0417     30  019E   30$:    BSBW    RESET_CONTROLLER         ;Ignore the rest of the response
           0D 50     E8  01A1           BLBS    R0,40$                   ;Branch if success
                         01A4       ;
                         01A4       ;       Something went wrong.
                         01A4       ;
              51     D4  01A4   13$:    CLRL    R1                       ;
            06F1     30  01A6           BSBW    RCSR                     ;Clear receiver CSR
            06DF     30  01A9           BSBW    SCSR                     ;Clear transmitter CSR
50   0054 8F         B0  01AC           MOVW    #SS$_CTRLERR,R0          ;Set error status
                  05     01B1   40$:    RSB                              ;Return
```

```
                                    01B2          .SBTTL  SND_PKT_NOINT
                                    01B2  ;++
                                    01B2  ; SND_PKT_NOINT
                                    01B2  ;
                                    01B2  ; This routine will send a packet of bytes to the TU58 controller.
                                    01B2  ; This routine is special in that it does not use interrupts, and
                                    01B2  ; expects IPL to be at IPL_POWER.
                                    01B2  ;
                                    01B2  ; Input:  R4 = CSR address
                                    01B2  ;         R5 = UCB address
                                    01B2  ;
                                    01B2  ; Output: R0 .. R3 are destroyed
                                    01B2  ;         Note that if R0 is nonzero, then not all the bytes were sent.
                                    01B2  ;--
                                    01B2
                                    01B2  SND_PKT_NOINT:                                         ;Send packet w/o interrupts
                    50   00D8 C5  3C 01B2          MOVZWL  UCB$W_DD_MSGLEN(R5),R0               ;Get # of bytes to send
                    52   00D4 C5  D0 01B7          MOVL    UCB$L_DD_MSGPKT(R5),R2               ;Get address of message packet
                                    01BC  ;
                                    01BC  ; Wait for one character time until the TU58 is ready for another character.
                                    01BC  ;
53   00000000'GF   00004E20 8F   C5 01BC  10$:     MULL3   #MAXCHRTIM,G^EXE$GL_TENUSEC,R3;Calculate wait loop count
                          00   E1 01C8          BBC     #DD_V_CONSOLE,-                      ;Branch if not a console device
                     0C 00CA C5     01CA                  UCB$W_TU58STS(R5),20$                ;
                  00000000'GF   91 01CE          CMPB    G^EXE$GB_CPUTYPE,-                   ;Is this a 730 cpu?
                          03      01D4                  #PR$_SID_TYP730                      ;
                          03   12 01D5          BNEQ    20$                                  ;Branch if not
                     53   28   C6 01D7          DIVL2   #40,R3                               ;Reduce wait time for 730 consoles
                          00   E1 01DA  20$:    BBC     #DD_V_CONSOLE,-                      ;Branch if not a console device
                     18 00CA C5     01DC                  UCB$W_TU58STS(R5),40$                ;
                     51   1E   DB 01E0          MFPR    #PR$_CSTS,R1                         ;Get console device CSR
                  17 51   07   E0 01E3  30$:    BBS     #DD_XCSR_V_TRDY,R1,50$               ;Branch if transmitter ready
     01'01'01'01'01'01'01'01'     01E7          .BYTE   1[8]                                 ;NOPs to slow loop to allow DMA
                     E8 53    F5 01EF          SOBGTR  R3,20$                               ;Loop
                50   0054 8F   3C 01F2          MOVZWL  #SS$_CTRLERR,R0                      ;ERROR -- transmitter never ready
                          05      01F7          RSB
                  51   04 A4   3C 01F8  40$:    MOVZWL  DD_XCSR(R4),R1                       ;Get UNIBUS device CSR
                          E5   11 01FC          BRB     30$                                  ;Rejoin common code
                                    01FE  ;
                                    01FE  ; Send a byte the TU58.
                                    01FE  ;
                     51   82   9A 01FE  50$:    MOVZBL  (R2)+,R1                             ;Get byte to send
                          00   E1 0201          BBC     #DD_V_CONSOLE,-                      ;Branch if not console device
                     09 00CA C5     0203                  UCB$W_TU58STS(R5),70$                ;
                     1F   51   DA 0207          MTPR    R1,#PR$_CSTD                         ;Send byte (console)
                     AF   50   F5 020A  60$:    SOBGTR  R0,10$                               ;Branch if more bytes to send
                          50   D6 020D          INCL    R0                                   ;Set success status
                          05      020F          RSB                                          ;Return
                  06 A4   51   B0 0210  70$:    MOVW    R1,DD_XBUF(R4)                       ;Send byte (unibus)
                          F4   11 0214          BRB     60$                                  ;Rejoin common code
```

```
                                     0216         .SBTTL  DUMP_RECVR_BUF
                                     0216  ;++
                                     0216  ; DUMP_RECVR_BUF
                                     0216  ;
                                     0216  ; Dump the contents of the receiver buffer to R1.
                                     0216  ; This routine will wait (via a delay loop) for
                                     0216  ; approximately 1 character time before giving up.
                                     0216  ; Note that the receiver buffer is dumped no matter what.
                                     0216  ; This routine executes at IPL equal to IPL$_POWER.
                                     0216  ;
                                     0216  ; Input:    R4 = CSR address
                                     0216  ;           R5 = UCB address
                                     0216  ;
                                     0216  ; Output:   R1 = contents of receiver buffer
                                     0216  ;           R0 is destroyed.
                                     0216  ;--
                                     0216
                                     0216  DUMP_RECVR_BUF:                                          ;Dump receiver buffer
50   00000000'GF   00004E20 8F   C5  0216          MULL3   #MAXCHRTIM,G^EXE$GL_TENUSEC,R0;Calc wait interval
                         00      E1  0222          BBC     #DD_V_CONSOLE,-                  ;Branch if not a console device
                   0C 00CA C5        0224                  UCB$W_TU58STS(R5),10$            ;
                   00000000'GF   91  0228          CMPB    G^EXE$GB_CPUTYPE,-               ;Is this a 730 cpu?
                         03          022E                  #PR$_SID_TYP730                  ;
                         03      12  022F          BNEQ    10$                              ;Branch if not
                   50    28      C6  0231          DIVL2   #40,R0                           ;Reduce wait time for 730 consoles
                         00      E1  0234  10$:    BBC     #DD_V_CONSOLE,-                  ;Branch if not console device
                   2E 00CA C5        0236                  UCB$W_TU58STS(R5),50$            ;
                      51 1C      DB  023A          MFPR    #PR$_CSRS,R1                     ;Get CSR (console)
               51    0080 8F     B3  023D  20$:    BITW    #DD_RCSR_M_RDONE,R1              ;Test DONE bit
                         12      12  0242          BNEQ    30$                              ;Branch if transfer done
        01'01'01'01'01'01'01'01'     0244          .BYTE   1[8]                             ;NOP's to slow loop to allow DMA I/O
                      E5 50      F5  024C          SOBGTR  R0,10$                           ;Branch if time not up
               50    0054 8F     3C  024F          MOVZWL  #SS$_CTRLERR,R0                  ;ERROR -- done bit never set
                         03      11  0254          BRB     35$                              ;Read receiver buffer anyway
                   50    01      3C  0256  30$:    MOVZWL  #SS$_NORMAL,R0                   ;Set success status
                         00      E1  0259  35$:    BBC     #DD_V_CONSOLE,-                  ;Branch if not a console device
                   04 00CA C5        025B                  UCB$W_TU58STS(R5),40$            ;
                      51 1D      DB  025F          MFPR    #PR$_CSRD,R1                     ;Dump receiver buffer (console)
                         05      05  0262          RSB                                      ;Return
                51    02 A4      3C  0263  40$:    MOVZWL  DD_RBUF(R4),R1                   ;Dump receiver buffer (unibus)
                         05      05  0267          RSB                                      ;Return
                   51    64      B0  0268  50$:    MOVW    DD_RCSR(R4),R1                   ;Get receiver CSR (unibus)
                         D0      11  026B          BRB     20$                              ;Rejoin common code
```

```
026D          .SBTTL  STARTIO ROUTINE
026D    ;++
026D    ; DD_STARTIO - START I/O OPERATION ON TU58
026D    ;
026D    ; This routine is entered when the associated unit is idle and a packet is
026D    ; available for processing.
026D    ;
026D    ; INPUTS:
026D    ;
026D    ;       R3 = ADDRESS OF I/O REQUEST PACKET.
026D    ;       R5 = ADDRESS OF DEVICE UNIT UCB.
026D    ;
026D    ; OUTPUTS:
026D    ;
026D    ;       The I/O request will be completed, barring errors, and the driver
026D    ;       returns with some extra information in the IOSB.  The format of the
026D    ;       IOSB is:
026D    ;                                 IOSB
026D    ;        31                      16 15                     0
026D    ;       +-----------------------+-----------------------+
026D    ;       :    TRANSFER COUNT     :        STATUS         :
026D    ;       +-----------------------+-----------------------+
026D    ;       :                   <unused>                    :
026D    ;       +-----------------------+-----------------------+
026D    ;
026D    ;       Where TRANSFER COUNT is the number of bytes transfered,
026D    ;       STATUS is the I/O status code.  The second longword is
026D    ;       unused, and is zero.
026D    ;
026D    ; ERROR LOGGING:
026D    ;
026D    ;       The information copied to the error logger buffer includes the 4
026D    ;       device registers as well as the COMMAND and END packet buffers.
026D    ;       The register dump routine assumes the information is in the UCB,
026D    ;       and that the register and packet buffers are contiguous.
026D    ; NOTE:
026D    ;       Data is sent/received to/from the TU58 one byte at a time.
026D    ;
026D    ;       All exchanges of data between the TU58 and this driver are done by
026D    ;       "message packets" that have a special format that is recognized by
026D    ;       the TU58 controller.  Each message packet has a checksum associated
026D    ;       with it.  The checksum is (for lack of a better word) checked upon
026D    ;       receipt of the message packet.  If the checksum that was sent with
026D    ;       the packet does not agree with checksum calculated by the receiver,
026D    ;       this is considered a controller error, and the driver attempts to
026D    ;       retry the entire I/O request.
026D    ;
026D    ;       To complete a given I/O request, several message packets may be sent
026D    ;       back and forth between the driver and the device.  For a full spec of
026D    ;       driver - device interaction and message packet design, see the
026D    ;       "TU58 DECtape II USER'S GUIDE", document # EK-0TU58-UG-001.
026D    ;
026D    ;
026D    ;--
026D    DD_STARTIO:                                     ;Start I/O operation
026D    ;
026D            REQPCHAN                                ;Request channel-prevent access to
```

```
                        0273                                            ;controller by two units at same time
                        0273
                        0273 DD_RESTARTIO:                              ;Restart the I/O operation after
                        0273                                            ;a powerfail recovery.  We start
                        0273                                            ;here to aviod having to release
                        0273                                            ;and then request a new channel.
                        0273
     0081 C5      90    0273          MOVB    UCB$B_ERTMAX(R5),-        ;Set error retry count
     0080 C5            0277                  UCB$B_ERTCNT(R5)          ;
       02         AA    027A          BICW    #UCB$M_DIAGBUF,-          ;Clear diag. status bit
     68 A5              027C                  UCB$W_DEVSTS(R5)          ;
       07         E1    027E          BBC     #IRP$V_DIAGBUF,-          ;Branch if not diag. request
  04 2A A3              0280                  IRP$W_STS(R3),10$         ;
       02         A8    0283          BISW    #UCB$M_DIAGBUF,-          ;Set diag. status bit
     68 A5              0285                  UCB$W_DEVSTS(R5)          ;
54   24 A5        D0    0287 10$:     MOVL    UCB$L_CRB(R5),R4          ;Get CRB address
                        028B          ASSUME  IDB$L_CSR EQ 0            ;The following assumes this
54   2C B4        D0    028B          MOVL    @CRB$L_INTD+VEC$L_IDB(R4),R4   ;Get CSR address
                        028F ;
                        028F ;        Buffer up the command packet.
                        028F ;
52  00E2 C5       9E    028F          MOVAB   UCB$K_DD_CPADDR(R5),R2    ;Get buffer address
82  0A02 8F       B0    0294          MOVW    #CMD_PKT_HDR,(R2)+        ;Set packet flag byte and byte count
       062C       30    0299          BSBW    SET_FUNCTION              ;Determine device function
     82   50      90    029C          MOVB    R0,(R2)+                  ;Put opcode into buffer
     82   51      90    029F          MOVB    R1,(R2)+                  ;Put opcode modifier into buffer
                        02A2 ;
                        02A2 ;        For console TU58s, the physical device unit number is one
                        02A2 ;        less than the unit number stored in the UCB.  This is a
                        02A2 ;        necessary to be compatible with existing software.
                        02A2 ;
50   54 A5        3C    02A2          MOVZWL  UCB$W_UNIT(R5),R0         ;Get device unit #
       00         E1    02A6          BBC     #DD_V_CONSOLE,-           ;Branch if not a console device
  02 00CA C5            02A8                  UCB$W_TU58STS(R5),20$     ;
       50         97    02AC          DECB    R0                        ;Decrement device unit
     82   50      90    02AE 20$:     MOVB    R0,(R2)+                  ;Put unit number into command packet
       50         D4    02B1          CLRL    R0                        ;Assume not using MSRP
       03         E1    02B3          BBC     #DD_V_USEMRSP,-           ;Branch if not using MRSP
  03 00CA C5            02B5                  UCB$W_TU58STS(R5),30$     ;
     50   08      90    02B9          MOVB    #MRSP_M_SWITCH,R0         ;Set switch to use MRSP
     82   50      90    02BC 30$:     MOVB    R0,(R2)+                  ;Set command switch
          82      B4    02BF          CLRW    (R2)+                     ;Set sequence # (unused)
62   7E A5        B0    02C1          MOVW    UCB$W_BCNT(R5),(R2)       ;Put byte count into buffer
00C0 C5   82      B0    02C5          MOVW    (R2)+,UCB$W_BCR(R5)       ;Save # bytes to transfer
                        02CA ;
                        02CA ;        Convert the physical media address in IRP$L_MEDIA to an LBN.
                        02CA ;        This is necessary because the TU58 controller expects an LBN,
                        02CA ;        not a physical media address.  The LBN is given by the formula:
                        02CA ;
                        02CA ;        LBN = (CYLINDER*(TRACKS/CYLINDER)+TRACK)*(SECTORS/TRACK))+SECTOR
                        02CA ;
     38 A3        D0    02CA          MOVL    IRP$L_MEDIA(R3),-         ;Copy media address to UCB
   00BC C5              02CD                  UCB$L_MEDIA(R5)           ;
53 00BC C5        DE    02D0          MOVAL   UCB$L_MEDIA(R5),R3        ;Get address of media address
   50   83        9A    02D5          MOVZBL  (R3)+,R0                  ;Get SECTOR
   51   83        9A    02D8          MOVZBL  (R3)+,R1                  ;Get TRACK
   7E   83        3C    02DB          MOVZWL  (R3)+,-(SP)               ;Get CYLINDER
```

```
   53   45 A5   9A  02DE          MOVZBL  UCB$B_TRACKS(R5),R3        ;Get TRACKS/CYLINDER
        53 8E   C4  02E2          MULL2   (SP)+,R3                   ;R3 = C*(T/C)
        51 53   C0  02E5          ADDL2   R3,R1                      ;R1 = C*(T/C)+T
   53   44 A5   9A  02E8          MOVZBL  UCB$B_SECTORS(R5),R3       ;Get SECTORS/TRACK
        51 53   C4  02EC          MULL2   R3,R1                      ;R1 = (C*(T/C)+T)*(S/T)
   82   50 51   A1  02EF          ADDW3   R1,R0,(R2)+                ;Put LBN into command packet buffer
                    02F3  ;
                    02F3  ;       Compute the checksum for the command packet, and then
                    02F3  ;       insert it at the end of the packet.
                    02F3  ;
   51   00E2 C5 9E  02F3          MOVAB   UCB$K_DD_CPADDR(R5),R1   ;Get command packet address
        52   0C 3C  02F8          MOVZWL  #<DD_CP_LEN - DD_CHKSM_LEN>,R2  ;Get command packet length
           05AA 30  02FB          BSBW    CHECKSUM                   ;Compute the checksum
   00EE C5   50 B0  02FE          MOVW    R0,UCB$W_DD_CPCKSM(R5)   ;Insert the checksum into packet
                    0303  ;
                    0303  DD_TRYAGN:
                    0303  ;
                    0303  ;       Clear the End Packet buffer
                    0303  ;
             30 BB  0303          PUSHR   #^M<R4,R5>                 ;Save R4 and R5
             00 2C  0305          MOVC5   #0,-                       ;Source length
      00000000'EF   0307                  0,-                        ;Source address
             00     030C                  #0,-                       ;Fill character
             0E     030D                  #DD_EP_LEN,-               ;Destination length
        00F0 C5     030E                  UCB$K_DD_EPADDR(R5)        ;Destination address
             30 BA  0311          POPR    #^M<R4,R5>                 ;Restore R4 and R5
                    0313  ;
                    0313  ;       In case the I/O request has been restarted, some of the
                    0313  ;       fields in the UCB must be reset to their initial values.
                    0313  ;
   53   58 A5   D0  0313          MOVL    UCB$L_IRP(R5),R3           ;Get IRP address
        2C A3   7D  0317          MOVQ    IRP$L_SVAPTE(R3),-         ;Restore UCB$L_SVAPTE,
        78 A5       031A                  UCB$L_SVAPTE(R5)           ;UCB$W_BCNT and UCB$W_BOFF
        7E A5   B0  031C          MOVW    UCB$W_BCNT(R5),-           ;Restore UCB$W_BCR
      00C0 C5       031F                  UCB$W_BCR(R5)              ;
          02    A8  0322          BISW    #DD_M_PASS1,-              ;Reset bit in TU58STS
      00CA C5       0324                  UCB$W_TU58STS(R5)          ;
                    0327  ;
                    0327  ;       Send the command packet to the device.
                    0327  ;
      00E2 C5   9E  0327          MOVAB   UCB$K_DD_CPADDR(R5),-    ;Get command packet address
      00D4 C5       032B                  UCB$L_DD_MSGPKT(R5)      ;
        0E      B0  032E          MOVW    #DD_CP_LEN,-               ;Get command packet length.
      00D8 C5       0330                  UCB$W_DD_MSGLEN(R5)      ;
                    0333          SEND_PACKET                        ;Send the command packet
                    0341  ;
                    0341  ;       Drop thru to GET_RESPONSE
                    0341  ;
```

```
                       0341  ;
                       0341  ; This sequence of code is executed after each successful
                       0341  ; transmission/reception of a message packet.  GET_RESPONSE
                       0341  ; merely gets the next byte sent from the device, and uses
                       0341  ; that byte to decide what to do next.
                       0341  ;
                       0341  GET_RESPONSE:                                        ;Get the TU58's resonse
                       0341            GET_BYTE                                   ;Get a byte
51   0091 C5    90     035B            MOVB    UCB$B_SPR(R5),R1                   ;Move to R1 for convenience
51     01       91     0360            CMPB    #DD_FLAG_DATA,R1                   ;Data flag?
       18       13     0363            BEQL    READ_DATA                          ;Branch if data flag
51     10       91     0365            CMPB    #DD_FLAG_CONT,R1                   ;Continue flag?
       0D       13     0368            BEQL    10$                                ;Branch if continue flag
51     02       91     036A            CMPB    #DD_FLAG_CNTL,R1                   ;Control flag?
       0B       13     036D            BEQL    20$                                ;If control flag then branch
50   0054 8F    3C     036F            MOVZWL  #SS$_CTRLERR,R0                    ;Assume controller error
     01ED       31     0374            BRW     CONT_ERROR                         ;Controller protocol error
     00C0       31     0377  10$:      BRW     WRITE_DATA                         ;Write to the device
     0143       31     037A  20$:      BRW     END_PACKET                         ;Get the end packet
```

```
                                 037D  ;
                                 037D  ; The following code sequence services READ requests by
                                 037D  ; transferring a data packet from the device to the user's buffer.
                                 037D  ;
                                 037D  READ_DATA:                                          ;Service READ reauest
          00D0 D5    51    90    037D            MOVB    R1,@UCB$L_DD_DPTR(R5)           ;Buffer data flag
                                 0382            GET_BYTE                                ;Get the packet byte count
               50    01    9A    039C            MOVZBL  #DD_DPBCNT,R0                   ;Put offset in R0
             0091 C5       90    039F            MOVB    UCB$B_SPR(R5),-                 ;Buffer the byte count
           00D0 D540             03A3                    @UCB$L_DD_DPTR(R5)[R0]          ;
                                 03A7  ;
                                 03A7  ;         Before getting the data from the device, ensure the data packet
                                 03A7  ;         byte count is correct.  Failure to do so can cause the driver to
                                 03A7  ;         overwrite nonpaged pool should the byte count be incorrect.  Note
                                 03A7  ;         that if the number of bytes remaining to be transfered is greater
                                 03A7  ;         than 128 (decimal), then the packet byte count must be 128.  If the
                                 03A7  ;         number of bytes remaining to be transfered is less than or equal to
                                 03A7  ;         128, then the packet byte count must equal that number.
                                 03A7  ;
          52    0091 C5    9A    03A7            MOVZBL  UCB$B_SPR(R5),R2                ;Get the PBC
          51    0080 8F    3C    03AC            MOVZWL  #128,R1                         ;Set up R1 for the comparisons
          51    00C0 C5    B1    03B1            CMPW    UCB$W_BCR(R5),R1                ;Is the BCR less than 128?
                    05     1E    03B6            BGEQU   100$                            ;Branch if not
          51    00C0 C5    3C    03B8            MOVZWL  UCB$W_BCR(R5),R1                ;Get the BCR
               51    52    D1    03BD  100$:     CMPL    R2,R1                           ;Is the PBR what it ought to be?
                    63     12    03C0            BNEQ    69$                             ;Branch if not (error)
                                 03C2  ;
                                 03C2  ;         Get the rest of the data packet, including the checksum.
                                 03C2  ;
                    50     96    03C2            INCB    R0                              ;Set offset to point to next byte
  00D4 C5    00D0 D540     9E    03C4            MOVAB   @UCB$L_DD_DPTR(R5)[R0],-        ;Get pointer into packet buffer
                                 03CC                    UCB$L_DD_MSGPKT(R5)             ;
          50    0091 C5    9B    03CC            MOVZBW  UCB$B_SPR(R5),R0                ;Get packet byte count
  00D8 C5    50     02     A1    03D1            ADDW3   #DD_CHKSM_LEN,R0,-              ;Allow room for the packet checksum
                                 03D7                    UCB$W_DD_MSGLEN(R5)             ;
                                 03D7            GET_PACKET                              ;Get the data packet
                                 03E5  ;
                                 03E5  ;         Now transfer the data in the data packet buffer to
                                 03E5  ;         the users buffer, and then await the TU58's next byte.
                                 03E5  ;         On return from MOVTOUSER, R1 points to the end of the packet buffer.
                                 03E5  ;
               50    02    9A    03E5            MOVZBL  #DD_DATA,R0                     ;Put offset into R0
       51  00D0 D540       9E    03E8            MOVAB   @UCB$L_DD_DPTR(R5)[R0],R1       ;Get address of data
          52    0091 C5    9A    03EE            MOVZBL  UCB$B_SPR(R5),R2                ;Get # of data bytes
                                 03F3            MOVTOUSER                               ;Move data
                                 0411  ;
                                 0411  ;         Calculate the packet checksum and compare it with the checksum
                                 0411  ;         received from the device.  If no match then the data was garbled
                                 0411  ;         during transmission and the I/O operation must be retried.
                                 0411  ;         On return from CHECKSUM, R1 points to the packet buffer checksum.
                                 0411  ;
               50    51    D0    0411            MOVL    R1,R0                           ;Save pointer into buffer
          51    00D0 C5    D0    0414            MOVL    UCB$L_DD_DPTR(R5),R1            ;Get buffer address
          52    50    51   C3    0419            SUBL3   R1,R0,R2                        ;Calculate the buffer length
                  0488     30    041D            BSBW    CHECKSUM                        ;Calculate the checksum
               61    50    B1    0420            CMPW    R0,(R1)                         ;Compare checksums
                    08     13    0423            BEQL    10$                             ;Branch if no errors
```

```
50   01F4 8F   B0 0425  69$:   MOVW    #SS$_PARITY,R0          ;Put error code in R0
          0137   31 042A         BRW     CONT_ERROR              ;Attempt to retry operation
                    042D  ;
                    042D  ;      Subtract the packet byte count from the total byte count and continue.
                    042D  ;
50   0091 C5   9B 042D  10$:   MOVZBW  UCB$B_SPR(R5),R0        ;Get data packet byte count
00C0 C5   50   A2 0432         SUBW    R0,UCB$W_BCR(R5)        ;Update byte transfer count
          FF07   31 0437         BRW     GET_RESPONSE            ;Await next byte from device
```

```
                         043A  ;
                         043A  ; The following code services a WRITE request by transfering data
                         043A  ; from the user's buffer to the TU58.  The transfer is made by
                         043A  ; sending data packets of <= 128 bytes.  When sending data we do
                         043A  ; not have to worry about data over-run.
                         043A  ;
                         043A  WRITE_DATA:                                  ;Send data to the device
            01      90   043A          MOVB    #DD_FLAG_DATA,-              ;Buffer data flag
       00D0 D5           043C                  @UCB$L_DD_DPTR(R5)           ;
                         043F  ;
                         043F  ;       Calculate the packet byte count.
                         043F  ;
       00C0 C5      B1   043F          CMPW    UCB$W_BCR(R5),-              ;Can we send a full packet?
       0080 8F           0443                  #DD_MAX_PKT_BCNT             ;
            07      1A   0446          BGTRU   10$                          ;Branch if yes
  51   00C0 C5      9B   0448          MOVZBW  UCB$W_BCR(R5),R1             ;Set byte count <= 128
            04      11   044D          BRB     20$                          ;Rejoin common code
   51    80 8F      9B   044F  10$:    MOVZBW  #DD_MAX_PKT_BCNT,R1          ;Set byte count = 128
        50  01      9A   0453  20$:    MOVZBL  #DD_DPBCNT,R0                ;Put offset into R0
 00D0 D540  51      90   0456          MOVB    R1,@UCB$L_DD_DPTR(R5)[R0] ;Put byte count into data packet
  00C0 C5   51      A2   045C          SUBW    R1,UCB$W_BCR(R5)             ;Calc new UCB$W_BCR
                         0461  ;
                         0461  ;       Transfer the data from the user's buffer to the packet buffer.
                         0461  ;       On return from IOC$MOVFRUSER, R1 = pointer into packet buffer.
                         0461  ;
        52  51      9A   0461          MOVZBL  R1,R2                        ;Get byte count
        50  02      9A   0464          MOVZBL  #DD_DATA,R0                  ;Put offset into R0
 51  00D0 D540      9E   0467          MOVAB   @UCB$L_DD_DPTR(R5)[R0],R1 ;Get address of data w/i data packet
                         046D          MOVFROMUSER                          ;Move data
                         048B  ;
                         048B  ;       Create the data packet checksum.  On return from CHECKSUM,
                         048B  ;       R1 points to the end of the data buffer.
                         048B  ;
        50  51      D0   048B          MOVL    R1,R0                        ;Save pointer into buffer
  51   00D0 C5      D0   048E          MOVL    UCB$L_DD_DPTR(R5),R1         ;Get buffer address
  52    50  51      C3   0493          SUBL3   R1,R0,R2                     ;Calculate buffer length
          040E      30   0497          BSBW    CHECKSUM                     ;Calculate checksum
        81  50      B0   049A          MOVW    R0,(R1)+                     ;Put checksum into buffer
                         049D  ;
                         049D  ;       Send the data packet to the device.
                         049D  ;
       00D0 C5      D0   049D          MOVL    UCB$L_DD_DPTR(R5),-          ;Get buffer address
       00D4 C5           04A1                  UCB$L_DD_MSGPKT(R5)          ;
50  51 00D4 C5      C3   04A4          SUBL3   UCB$L_DD_MSGPKT(R5),R1,R0 ;Calculate buffer length
  00D8 C5   50      B0   04AA          MOVW    R0,UCB$W_DD_MSGLEN(R5)       ;Move to input cell
                         04AF          SEND_PACKET                          ;Send data packet
          FE81      31   04BD          BRW     GET_RESPONSE                 ;Await next byte from device
```

```
                        04C0  ;
                        04C0  ; This code will get the end packet and interpret the device
                        04C0  ; status code.
                        04C0  ;
                        04C0  END_PACKET:                                    ;Intrepret the end packet
00F0 C5    51    90     04C0          MOVB    R1,UCB$B_DD_EPFLAG(R5)        ;Buffer flag byte
                        04C5  ;
                        04C5  ;       Read in the rest of the end packet, including the checksum.
                        04C5  ;
     00F1 C5    9E      04C5          MOVAB   UCB$B_DD_EPBCNT(R5),-         ;Get buffer address
     00D4 C5            04C9                  UCB$L_DD_MSGPKT(R5)           ;
          0D    B0      04CC          MOVW    #<DD_EP_LEN-1>,-              ;Get buffer length
     00D8 C5            04CE                  UCB$W_DD_MSGLEN(R5)           ;
                        04D1          GET_PACKET                            ;Get end packet
                        04DF  ;
                        04DF  ;       Calculate the packet checksum and compare it to the checksum sent
                        04DF  ;       by the device.  If no match, then the data was garbled during
                        04DF  ;       transmission, and the I/O request must be retried.
                        04DF  ;
51   00F0 C5    9E      04DF          MOVAB   UCB$K_DD_EPADDR(R5),R1        ;Get Command packet buffer address
52        0C    3C      04E4          MOVZWL  #<DD_CP_LEN - DD_CHKSM_LEN>,R2  ;Get buffer length
        03BE    30      04E7          BSBW    CHECKSUM                      ;Calculate checksum
00FC C5   50    B1      04EA          CMPW    R0,UCB$W_DD_EPCKSM(R5)        ;Compare checksums
          07    13      04EF          BEQL    10$                           ;Branch if match
50   01F4 8F    B0      04F1          MOVW    #SS$_PARITY,R0                ;Put error code in R0
          6C    11      04F6          BRB     CONT_ERROR                    ;Attempt to retry request
                        04F8  ;
                        04F8  ;       Get the device status and see how things went.
                        04F8  ;
51   00F3 C5    9A      04F8  10$:    MOVZBL  UCB$B_DD_EPSTS(R5),R1         ;Get device status code
50        01    B0      04FD          MOVW    #SS$_NORMAL,R0                ;Assume success
00        51    91      0500          CMPB    R1,#DD_STS_NORMAL             ;Test status
          37    18      0503          BGEQ    DD_EXIT                       ;Branch if success or success w/ retries
                        0505                                                ;- assumes value of 0 and 1 respectively
                        0505  ;
                        0505  ;       If we get this far, something went wrong.
                        0505  ;       Log a device error, and see if the device-specific
                        0505  ;       error code can be mapped directly to a VMS error code.
                        0505  ;       If not, then map the error into SS$_CTRLERR.  Note that
                        0505  ;       the operation will not be retried for this type of error.
                        0505  ;
 00000000'GF    16      0505          JSB     G^ERL$DEVICERR                ;Log the error
50   01C4 8F    B0      050B          MOVW    #SS$_NONEXDRV,R0              ;Assume bad unit #
 51     F8 8F   91      0510          CMPB    #DD_STS_BADUNIT,R1            ;
          26    13      0514          BEQL    DD_EXIT                       ;
50   01A4 8F    B0      0516          MOVW    #SS$_MEDOFL,R0                ;Assume no cartridge
 51     F7 8F   91      051B          CMPB    #DD_STS_NOCART,R1             ;
          1B    13      051F          BEQL    DD_EXIT                       ;
50   025C 8F    B0      0521          MOVW    #SS$_WRITLCK,R0               ;Assume writelock error
 51     F5 8F   91      0526          CMPB    #DD_STS_WRTLCK,R1             ;
          10    13      052A          BEQL    DD_EXIT                       ;
50   005C 8F    B0      052C          MOVW    #SS$_DATACHECK,R0             ;Assume datacheck error
 51     EF 8F   91      0531          CMPB    #DD_STS_DATACHK,R1            ;
          05    13      0535          BEQL    DD_EXIT                       ;
                        0537  ;
                        0537  ;       Map all other errors into CONTROLLER ERRORS, but
                        0537  ;       return device status code in device dependent
```

```
                        0537  1221 ;        field of the IOSB.
                        0537  1222 ;
        50  0054 8F  B0 0537  1223          MOVW    #SS$_CTRLERR,R0         ;Assume controller error
                        053C  1224 ;
                        053C  1225 ;        Fall through to DD_EXIT
                        053C  1226 ;
```

```
                        053C          .SBTTL  COMMON EXIT ROUTINE
                        053C      ;
                        053C      ;       Common exit routine.  Exit status is in the low word of R0.
                        053C      ;
                        053C      DD_EXIT:                                   ;Common exit routine
          7E    50   3C 053C              MOVZWL  R0,-(SP)                   ;Save I/O status
     00000000'GF     16 053F              JSB     G^IOC$DIAGBUFILL           ;Fill diagnostic buffer if present.
                        0545              RELCHAN                            ;Release channel
                        054B      ;
                        054B      ;       Put the IOSB info into R0 and R1 and exit the driver.
                        054B      ;
                50 8ED0 054B              POPL    R0                         ;Restore I/O status
        00C0 C5      A3 054E              SUBW3   UCB$W_BCR(R5),-            ;Calc bytes transferred
      52   7E A5        0552                      UCB$W_BCNT(R5),R2          ;
   52    52   10     78 0555              ASHL    #16,R2,R2                  ;Shift count to high word
         50   52     C8 0559              BISL    R2,R0                      ;Merge count into status longword
              51     D4 055C              CLRL    R1                         ;Clear second status longword
                        055E              REQCOM                             ;That's all folks!
```

```
                        0564          .SBTTL  Controller error handler
                        0564  ;
                        0564  ; This code attempts to recover from errors encountered during
                        0564  ; a data transfer.  Control is transferred to DD_RETRY only on
                        0564  ; device timeout recovery.  All other errors trap to CONT_ERROR.
                        0564  ;
                        0564  ; The only way to recover from an error is to set the controller to
                        0564  ; a known state, via the TU58 INIT sequence, and retry the entire I/O
                        0564  ; operation.  If the INIT sequence fails, then exit with a controller
                        0564  ; error.
                        0564  ;
                        0564  CONT_ERROR:                                      ;Recover from controller error
   00CC C5   50    3C   0564          MOVZWL  R0,UCB$L_DD_BUFWIN(R5)   ;Save the error status code
        00F4       30   0569          BSBW    DD_SAVEREG               ;Save device registers
   00000000'GF     16   056C          JSB     G^ERL$DEVICERR           ;Log the error
                        0572  ;
                        0572  ; This entry point is used by the timeout routine.  It is
                        0572  ; necessary to enter here to avoid overwriting the register
                        0572  ; save area in the UCB that was written by the timeout routine.
                        0572  ;
                        0572  DD_RETRY:
          44       10   0572          BSBB    RESET_CONTROLLER         ;Reset the device controller
       0F 50       E9   0574          BLBC    R0,10$                   ;Branch if controller init failure
     0080 C5       97   0577          DECB    UCB$B_ERTCNT(R5)         ;Decrement error retry count
          09       15   057B          BLEQ    10$                      ;Branch if no retries left
          0F       E0   057D          BBS     #IO$V_INHRETRY,-         ;Branch if no retry requested
   03 009A C5           057F                  UCB$W_FUNC(R5),10$       ;
        FD7D       31   0583          BRW     DD_TRYAGN                ;Retry I/O function
50   00CC C5       B0   0586  10$:    MOVW    UCB$L_DD_BUFWIN(R5),R0   ;Restore the error status code
          AF       11   058B          BRB     DD_EXIT                  ;Exit the driver
                        058D
```

```
                          058D          .SBTTL TIMEOUT and POWERFAIL recovery
                          058D  ;++
                          058D  ; This routine is called in the event of a device timeout.
                          058D  ; The timeout could be caused by a failure of the TU58 hardware
                          058D  ; or the hardware connecting the TU58 to the host machine, or
                          058D  ; the host machine could have undergone powerfail recovery.
                          058D  ;
                          058D  ; Input:
                          058D  ;
                          058D  ;       IPL     = Device IPL
                          058D  ;       (SP)    = Return address
                          058D  ;       R0-R3   = Scratch
                          058D  ;       R4      = CSR address
                          058D  ;       R5      = UCB address
                          058D  ;--
                          058D
                          058D  DD_TIME_OUT:
           00D0   30      058D          BSBW    DD_SAVEREG              ;Move registers into UCB
             51   D4      0590          CLRL    R1                      ;Prepare to disable interrupts
           0305   30      0592          BSBW    RCSR                    ;Disable receive interrupts
           02F3   30      0595          BSBW    SCSR                    ;Disable transmit interrupts
                          0598          SETIPL  UCB$B_FIPL(R5)          ;Lower IPL to FIPL
             05   E4      059C          BBSC    #UCB$V_POWER,-          ;Power failure?
        14 64 A5          059E                  UCB$W_STS(R5),DD_PWF    ;Branch if yes
      00000000'GF 16      05A1          JSB     G^ERL$DEVICTMO          ;Log timeout error
             06   E5      05A7          BBCC    #UCB$V_TIMOUT,-         ;Reset status
        00 64 A5          05A9                  UCB$W_STS(R5),10$       ;
        022C 8F   3C      05AC  10$:    MOVZWL  #SS$_TIMEOUT,-          ;Save the timeout status
        00CC C5           05B0                  UCB$L_DD_BUFWIN(R5)     ;
             BD   11      05B3          BRB     DD_RETRY                ;Init the controller and retry the I/O
                          05B5
                          05B5
                          05B5
                          05B5  ;
                          05B5  ; DD_PWF
                          05B5  ;
                          05B5  ; This routine is branched after the device timeout caused by
                          05B5  ; a powerfail.  The I/O operation is retried.  Note that the
                          05B5  ; controller has already been reset by the unit initialization
                          05B5  ; routines, which are called by the operating system as part
                          05B5  ; of its normal powerfail recovery.
                          05B5  ;
                          05B5  DD_PWF:
           FCBB   31      05B5          BRW     DD_RESTARTIO            ;Start function over again
```

```
                        05B8           .SBTTL  RESET_CONTROLLER
                        05B8   ;++
                        05B8   ; RESET_CONTROLLER
                        05B8   ;
                        05B8   ;       This routine will set the device controller to a known state.
                        05B8   ;       To initialize the TU58 controller, it is necessary to perform
                        05B8   ;       the following steps:
                        05B8   ;
                        05B8   ;               - Set the BREAK bit in the transmitter CSR and hold
                        05B8   ;                 it for at least two character times.
                        05B8   ;               - Clear the Break bit.
                        05B8   ;               - Dump the receiver buffer.  The contents are unimportant.
                        05B8   ;               - Send two INIT characters.
                        05B8   ;               - Dump the reciever buffer.  The response must be equal to
                        05B8   ;                 DD_FLAG_CONT.  If not, the initialization sequence failed.
                        05B8   ;
                        05B8   ;       The initialization sequence must be done with IPL equal to IPL_POWER
                        05B8   ;       to lock out all interrupts.
                        05B8   ;
                        05B8   ; Input:
                        05B8   ;
                        05B8   ;       R0-R3   = Scratch
                        05B8   ;       R4      = CSR address
                        05B8   ;       R5      = UCB addres
                        05B8   ;
                        05B8   ; Output:
                        05B8   ;
                        05B8   ;       R0 = SS$_NORMAL if the reset worked, SS$_CTRLERR if not.
                        05B8   ;--
                        05B8
                        05B8   RESET_CONTROLLER:
                        05B8           ;
                        05B8           ; Set the BREAK condition.  Clear it after a short wait.
                        05B8           ;
         51    01   3C  05B8           MOVZWL  #DD_XCSR_M_BREAK,R1      ;Put BREAK into R1
                        05BB           DSBINT                           ;Disable interrupts
             02C7   30  05C1           BSBW    SCSR                     ;Set BREAK condition
                        05C4           TIMEWAIT #<2*MAXCHRTIM>,#1,#0,W ;Loop for 2 character times
               51   D4  05EB           CLRL    R1                       ;Clear R1
             029B   30  05ED           BSBW    SCSR                     ;Clear break condition
                        05F0           ;
                        05F0           ; Dump the receiver buffer.
                        05F0           ;
             FC23   30  05F0           BSBW    DUMP_RECVR_BUF           ;Dump buffer and discard contents
                        05F3           ;
                        05F3           ; Send two INIT characters.
                        05F3           ;
   50   00D4 C5     DE  05F3           MOVAL   UCB$L_DD_MSGPKT(R5),R0   ;Get address of buffer pointer
   60   00CE C5     3E  05F8           MOVAW   2+UCB$L_DD_BUFWIN(R5),(R0);Get address of message buffer
00 B0   0404 8F     B0  05FD           MOVW    #TWO_INITS,@(R0)         ;Put INIT characters into buffer
   04 A0   02       B0  0603           MOVW    #2,4(R0)                 ;Save # of bytes to send
             FBA8   30  0607           BSBW    SND_PKT_NOINT            ;Send the packet
            10 50   E9  060A           BLBC    R0,20$                   ;Branch if error
                        060D           ;
                        060D           ; The controller should respond with a DD_FLAG_CONT.
                        060D           ; If it does not, then exit with a controller error.
                        060D           ; R0 will contain a meaningful status code on return
```

```
                      060D           ; from dump_recvr_buf.
                      060D           ;
          FC06   30   060D           BSBW    DUMP_RECVR_BUF           ;Get controller response
       50   01   3C   0610           MOVZWL  #SS$_NORMAL,R0           ;Assume success status
       51   10   91   0613           CMPB    #DD_FLAG_CONT,R1         ;Is it the correct response?
            05   13   0616           BEQL    20$                      ;Branch if true
50    0054 8F    3C   0618           MOVZWL  #SS$_CTRLERR,R0          ;Set error status
                      061D      20$: ENBINT                           ;Enable interrupts
                 05   0620           RSB                              ;Return
```

```
                          0621             .SBTTL  REGISTER DUMP ROUTINE
                          0621      ;++
                          0621      ; DD_REGDUMP - REGISTER DUMP ROUTINE
                          0621      ;
                          0621      ; FUNCTIONAL DESCRIPTION:
                          0621      ;
                          0621      ; This routine is called to save the device registers in a specified
                          0621      ; buffer. It is called from the device error logging routine and from
                          0621      ; the diagnostic buffer fill routine.
                          0621      ;
                          0621      ; INPUTS:
                          0621      ;       R0 = ADDRESS OF REGISTER SAVE BUFFER
                          0621      ;       R4 = ADDRESS OF DEVICE CONTROL STATUS REGISTER
                          0621      ;       R5= ADDRESS OF UNIT CONTROL BLOCK
                          0621      ; OUTPUTS:
                          0621      ;       DEVICE REGISTERS ARE MOVED INTO THE SPECIFIED BUFFER.
                          0621      ;       R0 CONTAINS THE ADDRESS OF THE NEXT EMPTY LONGWORD IN THE BUFFER.
                          0621      ;       ALL REGISTERS EXCEPT R1 AND R2 ARE PRESERVED.
                          0621      ;
                          0621      ;--
                          0621      DD_REGDUMP:
       80    0B     D0    0621             MOVL    #DD_ERRBUF_LEN_L,(R0)+    ;Insert # of longwords dumped
       52    04     D0    0624             MOVL    #DD_NUM_REGS,R2           ;Store number of registers
 51    00DA  C5     3E    0627             MOVAW   UCB$W_DD_XCSR(R5),R1      ;Get address of first reg.
       80    81     3C    062C      10$:   MOVZWL  (R1)+,(R0)+               ;Dump registers into buffer
          FA 52     F5    062F             SOBGTR  R2,10$                    ;If GTR - more to go
                          0632      ;
                          0632      ;       Now dump the Command and End packets to the error buffer.
                          0632      ;
       52    07     D0    0632             MOVL    #7,R2                     ;Get length of combined packets
       80    81     D0    0635      20$:   MOVL    (R1)+,(R0)+               ;Dump longword to buffer
          FA 52     F5    0638             SOBGTR  R2,20$                    ;If GTR - more to go
                    05    063B             RSB                               ;Return
```

```
                        063C         .SBTTL  INTERRUPT SERVICE ROUTINE
                        063C  ;
                        063C  ;++
                        063C  ; DD$INT - TU58 INTERRUPT ROUTINE
                        063C  ;
                        063C  ; This routine is entered via a JSB instruction when an
                        063C  ; interrupt occurs.  The state of the stack on entry is:
                        063C  ;
                        063C  ;          00(SP) = ADDRESS OF IDB ADDRESS.
                        063C  ;          04(SP) = SAVED R0.
                        063C  ;          08(SP) = SAVED R1.
                        063C  ;          12(SP) = SAVED R2.
                        063C  ;          16(SP) = SAVED R3.
                        063C  ;          20(SP) = SAVED R4.
                        063C  ;          24(SP) = SAVED R5.
                        063C  ;          28(SP) = SAVED INTERRUPT PC
                        063C  ;          32(SP) = INTERRUPT PSL.
                        063C  ;
                        063C  ; Interrupt dispatching occurs as follows:
                        063C  ;--
                        063C DD_INT::
       53   9E    D0    063C         MOVL    @(SP)+,R3               ;Get IDB address
       54   63    7D    063F         MOVQ    (R3),R4                 ;Get CSR and UCB address
    53    10 A5   7D    0642         MOVQ    UCB$L_FR3(R5),R3        ;Restore R3 and R4
          01      E1    0646         BBC     #UCB$V_INT,-            ;If clr -interrupt not expected
    03 64 A5            0648                 UCB$W_STS(R5),DD_UNSOLNT
       0C B5      17    064B         JMP     @UCB$L_FPC(R5)          ;Jump to interrupt ret. address
                        064E DD_UNSOLNT:
                        064E  ;
                        064E  ;          In order to dismiss the interrupt, the receiver buffer must be read.
                        064E  ;
          00      E1    064E         BBC     #DD_V_CONSOLE,-         ;Branch if not console device
   06 00CA C5           0650                 UCB$W_TU58STS(R5),20$   ;
       51   1D    DB    0654         MFPR    #PR$_CSRD,R1            ;Read buffer for CONSOLE TU58
            3F    BA    0657 10$:    POPR    #^M<R0,R1,R2,R3,R4,R5>  ;Restore R0-R5
            02          0659         REI                             ;Dismiss the interrupt
    51    02 A4   B0    065A 20$:    MOVW    DD_RBUF(R4),R1          ;Read buffer for UNIBUS TU58
            F7    11    065E         BRB     10$                     ;Dismiss the interrupt
                        0660
```

```
                         0660            .SBTTL  LOCAL SUBROUTINES
                         0660
                         0660      ;++
                         0660      ;
                         0660      ; DD_SAVEREG
                         0660      ;
                         0660      ; ABSTRACT:     Store the device registers in the UCB, so they may
                         0660      ;               later be passed to the error logger.
                         0660      ;
                         0660      ; INPUT:        R4 = CSR Address
                         0660      ;               R5 = UCB address
                         0660      ;
                         0660      ; OUTPUT:       Registers saved in UCB.
                         0660      ;
                         0660      ; SIDE EFFECTS: R0,R2 are over-written.
                         0660      ;
                         0660      ;               Note that for CONSOLE devices, the xmitter buffer is
                         0660      ;               a write-only register.  Code around this by saving the
                         0660      ;               image of the buffer as it is loaded in the routine
                         0660      ;               SEND_PACKET.  Also note that the MFPR instruction has
                         0660      ;               a longword context, so the CONSOLE devices registers
                         0660      ;               must be saved carefully, so other locations in the UCB
                         0660      ;               are not overwritten.
                         0660      ;
                         0660      ;--
                         0660      DD_SAVEREG:                                          ;Save device registers.
   52   00DA C5    3E    0660                 MOVAW   UCB$K_DD_REGBUF(R5),R2         ;Get address of buffer
              00   E1    0665                 BBC     #DD_V_CONSOLE,-                ;Branch if not console device
      12 00CA C5         0667                         UCB$W_TU58STS(R5),10$          ;
         50   1E   DB    066B                 MFPR    #PR$_CSTS,R0                   ;Get xmitter CSR
         82   50   B0    066E                 MOVW    R0,(R2)+                       ;Save xmitter CSR
      02 A2   1C   DB    0671                 MFPR    #PR$_CSRS,2(R2)                ;Save receiver CSR
         50   1D   DB    0675                 MFPR    #PR$_CSRD,R0                   ;Get receiver buffer
      04 A2   50   B0    0678                 MOVW    R0,4(R2)                       ;Save receiver buffer
              05         067C                 RSB                                    ;Return
   82    04 A4     B0    067D      10$:       MOVW    DD_XCSR(R4),(R2)+              ;Save xmitter CSR
   82    06 A4     B0    0681                 MOVW    DD_XBUF(R4),(R2)+              ;Save xmitter buffer
      82    64     B0    0685                 MOVW    DD_RCSR(R4),(R2)+              ;Save receiver CSR
   82    02 A4     B0    0688                 MOVW    DD_RBUF(R4),(R2)+              ;Save receiver buffer
              05         068C                 RSB                                    ;Return
```

```
                         068D           .SBTTL  GET_PACKET_C
                         068D   ;++
                         068D   ;
                         068D   ; GET_PACKET_C
                         068D   ;
                         068D   ; ABSTRACT:       Get a message packet from a console TU58.  Because of
                         068D   ;                 the unique nature of the TU58, this routine is optimized
                         068D   ;                 for speed.  The TU58 generates an inordinate number of
                         068D   ;                 interrupts, and can use a large fraction of the processor
                         068D   ;                 during normal operation.  In order to reduce overhead, some
                         068D   ;                 extreme measures have been taken in the coding of this routine.
                         068D   ;
                         068D   ; INPUT:          R4 = CSR address
                         068D   ;                 R5 = UCB address
                         068D   ;                 UCB$L_DD_MSGPKT(R5) = Message packet buffer address
                         068D   ;                 UCB$W_DD_MSGLEN(R5) = Message packet buffer length
                         068D   ;
                         068D   ; OUTPUT:         None.
                         068D   ;
                         068D   ;
                         068D   ; NOTES:          1) This routine enables the device interrupts at the
                         068D   ;                    beginning of the message packet and does not
                         068D   ;                    disable them until the entire message packet has
                         068D   ;                    been transferred.
                         068D   ;
                         068D   ;                 2) The WFIKPCH macro is not used.  Instead, the code
                         068D   ;                    is placed inline to avoid the cost in time inherent
                         068D   ;                    in the macro.
                         068D   ;
                         068D   ;                 3) To save the overhead of IOFORKing after each byte
                         068D   ;                    transferred, the routine will not IOFORK until
                         068D   ;                    the entire message packet has been transferred.
                         068D   ;
                         068D   ;                 4) The large time (40 seconds) spent waiting for a device timeout
                         068D   ;                    is necessary because of the slowness of the TU58.  It takes on
                         068D   ;                    the order of 30 - 40 seconds for the device to position the
                         068D   ;                    tape from one end of the cartridge to another.
                         068D   ;
                         068D   ;--
                         068D
                         068D   GET_PACKET_C:                           ;Get a message packet
                         068D   ;
                         068D   ;       The following code is done only once, to set up
                         068D   ;       for the optimized interrupt service loop.
                         068D   ;
        009C C5    8E  D0  068D         MOVL    (SP)+,UCB$L_DPC(R5)     ;Store return address
           0040 8F     AA  0692         BICW    #UCB$M_TIMOUT,-         ;Clear timeout occurred bit
             64 A5         0696                 UCB$W_STS(R5)           ;
                           0698   ;
                           0698   ; If this is an MRSP TU58 then set the interrupt dispatch address to
                           0698   ; point to the code that will handle the MRSP protocol.  Otherwise,
                           0698   ; set the dipatch address to a routine to handle the normal RSP TU58.
                           0698   ;
 0C A5   000006E4'EF   DE  0698         MOVAL   L^300$,UCB$L_FPC(R5)    ;Assume this is an MRSP TU58.
                  03   E0  06A0         BBS     #DD_V_USEMRSP,-         ;Branch if we are to use MRSP
          08 00CA C5       06A2                 UCB$W_TU58STS(R5),10$   ;
 0C A5   0000070E'EF   DE  06A6         MOVAL   L^600$,UCB$L_FPC(R5)    ;Set address for RSP TU58
```

```
          00D4 C5    D0  06AE  10$:    MOVL    UCB$L_DD_MSGPKT(R5),-     ;Get address of message packet
            10 A5        06B2                  UCB$L_FR3(R5)             ;
          00D8 C5    3C  06B4              MOVZWL  UCB$W_DD_MSGLEN(R5),-     ;Get length of message
            14 A5        06B8                  UCB$L_FR4(R5)             ;
                         06BA              DSBINT  UCB$B_DIPL(R5)            ;Lock out device interrupts
  1C   00000040 8F   DA  06C1              MTPR    #DD_RCSR_M_RIE,#PR$_CSRS;Enable interrupts
               03    A8  06C8              BISW    #UCB$M_TIM!UCB$M_INT,-    ;Set INTERRUPT and TIMEOUT enabled
            64 A5        06CA                  UCB$W_STS(R5)             ;
               34    C1  06CC              ADDL3   #LONG_WAIT,-              ;Compute timeout time for entire transfer.
        00000000'GF      06CE                  G^EXE$GL_ABSTIM,-         ;This will save recomputing it for each
            6C A5        06D3                  UCB$L_DUETIM(R5)          ;byte transferred.
                         06D5      ;
                         06D5      ;       This code is executed for the first byte received from the device.
                         06D5      ;       All succeeding interrupts for bytes bytes received will go through
                         06D5      ;       another path, which is faster.  This is done to save several
                         06D5      ;       instructions that need not be executed again.  Note for this path
                         06D5      ;       the IPL must be restored before returning to the caller's caller.
                         06D5      ;       Failure to do so will cause a BUGCHECK in the fork process dispatcher.
                         06D5      ;       Note that R3 and R4 have already been saved, so it is not necessary
                         06D5      ;       to do so again.  Also note that since interrupts and timeouts are not
                         06D5      ;       disabled until all the data has been transfered, it is not necessary
                         06D5      ;       to set the INTERRUPT and TIMEOUT enable bits for each byte transferred.
                         06D5      ;       For that reason, the timeout time is computed only once, and is large
                         06D5      ;       enough to allow time to transfer the maximum number of bytes possible,
                         06D5      ;       which is 128 bytes of data plus some RSP protocol overhead.
                         06D5      ;
                         06D5  100$:   ENBINT                            ;Allow device interrupts
               05        06D8              RSB                               ;Return
             FEB4        06D9              .WORD   DD_TIME_OUT-.             ;Offset to timeout routine
                         06DB      ;
                         06DB      ;       This is the fast interrupt servic loop for MRSP protocol
                         06DB      ;       data transfers.  This path exists to save testing the MRSP
                         06DB      ;       bit in the UCB for each byte transferred.  Note that the interrupt
                         06DB      ;       is dispatched here to avoid the cost of an RSB back to the device's
                         06DB      ;       interrupt dispatcher.  Of course the interrupt dispatcher JMPs here.
                         06DB      ;
       10 A5   53    7D  06DB  200$:   MOVQ    R3,UCB$L_FR3(R5)          ;Save R3 and R4
               3F    BA  06DF  250$:   POPR    #^M<R0,R1,R2,R3,R4,R5>    ;Restore registers from the interrupt
               02        06E1              REI                               ;Dismiss the interrupt
             FEAB        06E2              .WORD   DD_TIME_OUT-.             ;Offset to timeout routine
                         06E4      ;
                         06E4      ;       Dump the receiver buffer (word sized) into R1.
                         06E4      ;       After saving the data, acknowledge the receipt of the data.
                         06E4      ;       This will cause the TU58 to send the next byte, if any.
                         06E4      ;
          51   1D    DB  06E4  300$:   MFPR    #PR$_CSRD,R1              ;Dump receiver buffer to R1
          1F   10    DA  06E7              MTPR    #ACK_CHAR,#PR$_CSTD       ;Send ack. character
          83   51    90  06EA  400$:   MOVB    R1,(R3)+                  ;Put byte in message buffer
          EB   54    F5  06ED              SOBGTR  R4,200$                   ;Branch if more to go
                         06F0      ;
                         06F0      ;       The data has been transferred.  Disable device interrupts
                         06F0      ;       and create a fork process to continue on with the I/O operation.
                         06F0      ;
          1C   00    DA  06F0  500$:   MTPR    #0,#PR$_CSRS              ;Disable device interrupts
               03    AA  06F3              BICW    #UCB$M_TIM!UCB$M_INT,-    ;Clear INTERRUPT and TIMEOUT enabled
            64 A5        06F5                  UCB$W_STS(R5)             ;
      FFFFFFE2 EF    9F  06F7              PUSHAB  L^250$                    ;Set return address of "caller's caller"
```

```
                              06FD          IOFORK                        ;Get back to fork context
          50    0838 8F    3C 0703          MOVZWL  #SS$_DATAOVERUN,R0    ;Always return overrun status
                009C D5    17 0708          JMP     @UCB$L_DPC(R5)        ;Return
                              070C  ;
                              070C  ;       Check for data overrun.  If an overrun occurred, disable
                              070C  ;       device interrupts and get back to fork context.  Then
                              070C  ;       reset the controller.  Note that no overruns will (should)
                              070C  ;       occur when using MRSP, so this test does not have to be made
                              070C  ;       for MRSP TU58s.
                              070C  ;
                         FE81 070C          .WORD   DD_TIME_OUT-.         ;Offset to timeout routine
                  51 1D    DB 070E  600$:   MFPR    #PR$_CSRD,R1          ;Dump receiver buffer to R1
               D5 51 0F    E1 0711          BBC     #DD_RBUF_V_ERROR,R1,400$;Branch if no overrun
   009C C5   FFFFFE49 EF   DE 0715          MOVAL   L^CONT_ERROR,UCB$L_DPC(R5);Return to controller reset routine
                     D0    11 071E          BRB     500$                  ;Branch to common exit code
```

```
                              0720          .SBTTL  SEND_PACKET_C
                              0720  ;++
                              0720  ;
                              0720  ; SEND_PACKET_C
                              0720  ;
                              0720  ; ABSTRACT:     Send a message packet to a console TU58.  Because of
                              0720  ;               the unique nature of the TU58, this routine is optimized
                              0720  ;               for speed.  The TU58 generates an inordinate number of
                              0720  ;               interrupts, and can use a large fraction of the processor
                              0720  ;               during normal operation.  In order to reduce overhead, some
                              0720  ;               extreme measures have been taken in the coding of this routine.
                              0720  ;
                              0720  ; INPUT:        R4 = CSR address
                              0720  ;               R5 = UCB address
                              0720  ;               UCB$L_DD_MSGPKT(R5) = Message packet buffer address
                              0720  ;               UCB$W_DD_MSGLEN(R5) = Message packet buffer length
                              0720  ;
                              0720  ; OUTPUT:       None.
                              0720  ;
                              0720  ; NOTES:
                              0720  ;
                              0720  ;               1) This routine enables the device interrupts at the
                              0720  ;                  beginning of the message packet and does not
                              0720  ;                  disable them until the entire message packet has
                              0720  ;                  been transferred.
                              0720  ;
                              0720  ;               2) The WFIKPCH macro is not used.  Instead, the code
                              0720  ;                  is placed inline to avoid the cost in time inherent
                              0720  ;                  in the macro.
                              0720  ;
                              0720  ;               3) To save the overhead of IOFORKing after each byte
                              0720  ;                  transferred, the routine will not IOFORK until
                              0720  ;                  the entire message packet has been transferred.
                              0720  ;
                              0720  ;
                              0720  ;--
                              0720  SEND_PACKET_C:                                  ;Send a message packet (console)
                              0720  ;
                              0720  ;       The following code needs to be done only once, to
                              0720  ;       set things up for the interrupt service loop.
                              0720  ;
          009C C5   8E   D0   0720          MOVL    (SP)+,UCB$L_DPC(R5)             ;Store return address
             0040   8F   AA   0725          BICW    #UCB$M_TIMOUT,-                 ;Clear timeout occurred bit
                    64   A5   0729                  UCB$W_STS(R5)                   ;
  0C A5   00000769'EF    DE   072B          MOVAL   L^300$,UCB$L_FPC(R5)            ;Save interrupt dispatch address
             00D4   C5   D0   0733          MOVL    UCB$L_DD_MSGPKT(R5),-           ;Get address of message packet
                    10   A5   0737                  UCB$L_FR3(R5)                   ;
             00D8   C5   3C   0739          MOVZWL  UCB$W_DD_MSGLEN(R5),-           ;Get length of message
                    14   A5   073D                  UCB$L_FR4(R5)                   ;
                              073F          DSBINT  UCB$B_DIPL(R5)                  ;Lock out device interrupts
     1E   00000040   8F  DA   0746          MTPR    #DD_XCSR_M_TIE,#PR$_CSTS        ;Enable transmitter interrupts
                    03   A8   074D          BISW    #UCB$M_TIM!UCB$M_INT,-          ;Set INTERRUPT and TIMEOUT enabled
                    64   A5   074F                  UCB$W_STS(R5)                   ;
                    34   C1   0751          ADDL3   #LONG_WAIT,-                    ;Compute timeout time for entire transfer.
          00000000'GF         0753                  G^EXE$GL_ABSTIM,-               ;This will save recomputing it for each
                    6C   A5   0758                  UCB$L_DUETIM(R5)                ;byte transferred.
                              075A  ;
```

```
                   075A  ;       This code is executed for the first byte received from the device.
                   075A  ;       All succeeding interrupts for bytes bytes received will go through
                   075A  ;       another path, which is faster.  This is done to save several
                   075A  ;       instructions that need not be executed again.  Note for this path
                   075A  ;       the IPL must be restored before returning to the caller's caller.
                   075A  ;       Failure to do so will cause a BUGCHECK in the fork process dispatcher.
                   075A  ;       Note that R3 and R4 have already been saved, so it is not necessary
                   075A  ;       to do so again.  Also note that since interrupts and timeouts are not
                   075A  ;       disabled until all the data has been transfered, it is not necessary
                   075A  ;       to set the INTERRUPT and TIMEOUT enable bits for each byte transferred.
                   075A  ;       For that reason, the timeout time is computed only once, and is large
                   075A  ;       enough to allow time to transfer the maximum number of bytes possible,
                   075A  ;       which is 128 bytes of data plus some RSP protocol overhead.
                   075A  ;
                   075A  100$:   ENBINT                          ;Allow device interrupts
              05   075D          RSB                             ;Return
            FE2F   075E          .WORD   DD_TIME_OUT-.           ;Offset to timeout routine
                   0760  ;
                   0760  ;       This is the second and faster path through the wait-for-
                   0760  ;       interrupt-and-keep-channel logic.  It is reasonable to have
                   0760  ;       two paths, as the first path is only needed for the first
                   0760  ;       byte sent, and there will always be at least 14 bytes (the
                   0760  ;       length of a command packet) sent.  This path saves two instructions,
                   0760  ;       as it would be necessary to save and restore the IPL in the
                   0760  ;       other path.  Note that the interrupt is dismissed here.  This
                   0760  ;       is to save the necessary RSB to return to the interrupt dispatcher.
                   0760  ;       Of course the driver's interrupt dispatch JMPs to the location
                   0760  ;       stored in UCB$L_FPC within the device UCB.
                   0760  ;
 10 A5     53  7D  0760  200$:   MOVQ    R3,UCB$L_FR3(R5)        ;Save R3 and R4
           3F  BA  0764  250$:   POPR    #^M<R0,R1,R2,R3,R4,R5>  ;Restore registers from the interrupt
               02  0766          REI                             ;Dismiss the interrupt
             FE26  0767          .WORD   DD_TIME_OUT-.           ;Offset to timeout routine
                   0769  ;
                   0769  ;       Get the data byte from the packet buffer and send it
                   0769  ;       to the device.  If there is more data to send, go through
                   0769  ;       the second wait-for-interrupt-keep-channel path.
                   0769  ;
     51    83  9A  0769  300$:   MOVZBL  (R3)+,R1                ;Get data byte and point to next byte
     1F    51  DA  076C          MTPR    R1,#PR$_CSTD            ;Send data byte
 00DC C5   51  B0  076F          MOVW    R1,UCB$W_DD_XBUF(R5)    ;Save data byte (register is write-only)
     E9    54  F5  0774          SOBGTR  R4,200$                 ;Loop if more data
                   0777  ;
                   0777  ;       The data has been transferred.  Disable device interrupts
                   0777  ;       and create a fork process to continue on with the I/O operation.
                   0777  ;
           03  AA  0777          BICW    #UCB$M_TIM!UCB$M_INT,-  ;Clear INTERRUPT and TIMEOUT enabled
        64 A5      0779                  UCB$W_STS(R5)           ;
     1E    00  DA  077B          MTPR    #0,#PR$_CSTS            ;Disable device interrupts
  FFFFFFE0 EF  9F  077E          PUSHAB  L^250$                  ;Set return address of "caller's caller"
                   0784          IOFORK                          ;Get back to fork context
   009C D5     17  078A          JMP     @UCB$L_DPC(R5)          ;Return
```

```
                             078E          .SBTTL  GET_PACKET_U
                             078E  ;++
                             078E  ;
                             078E  ; GET_PACKET_U
                             078E  ;
                             078E  ; ABSTRACT:      Get a message packet from a unibus TU58.  Because of
                             078E  ;                the unique nature of the TU58, this routine is optimized
                             078E  ;                for speed.  The TU58 generates an inordinate number of
                             078E  ;                interrupts, and can use a large fraction of the processor
                             078E  ;                during normal operation.  In order to reduce overhead, some
                             078E  ;                extreme measures have been taken in the coding of this routine.
                             078E  ;
                             078E  ; INPUT:         R4 = CSR address
                             078E  ;                R5 = UCB address
                             078E  ;                UCB$L_DD_MSGPKT(R5) = Message packet buffer address
                             078E  ;                UCB$W_DD_MSGLEN(R5) = Message packet buffer length
                             078E  ;
                             078E  ; OUTPUT:        None.
                             078E  ;
                             078E  ;
                             078E  ; NOTES:         1) This routine enables the device interrupts at the
                             078E  ;                   beginning of the message packet and does not
                             078E  ;                   disable them until the entire message packet has
                             078E  ;                   been transferred.
                             078E  ;
                             078E  ;                2) The WFIKPCH macro is not used.  Instead, the code
                             078E  ;                   is placed inline to avoid the cost in time inherent
                             078E  ;                   in the macro.
                             078E  ;
                             078E  ;                3) To save the overhead of IOFORKing after each byte
                             078E  ;                   transferred, the routine will not IOFORK until
                             078E  ;                   the entire message packet has been transferred.
                             078E  ;
                             078E  ;                4) The large time (40 seconds) spent waiting for a device timeout
                             078E  ;                   is necessary because of the slowness of the TU58.  It takes on
                             078E  ;                   the order of 30 - 40 seconds for the device to position the
                             078E  ;                   tape from one end of the cartridge to another.
                             078E  ;
                             078E  ;--
                             078E
                             078E  GET_PACKET_U:                                ;Get a message packet
                             078E  ;
                             078E  ;       The following code is done only once, to set up
                             078E  ;       for the optimized interrupt service loop.
                             078E  ;
         009C C5   8E   D0   078E          MOVL    (SP)+,UCB$L_DPC(R5)      ;Store return address
            0040 8F     AA   0793          BICW    #UCB$M_TIMOUT,-          ;Clear timeout occurred bit
               64 A5         0797                  UCB$W_STS(R5)            ;
                             0799  ;
                             0799  ; If this is an MRSP TU58 then set the interrupt dispatch address to
                             0799  ; point to the code that will handle the MRSP protocol.  Otherwise,
                             0799  ; set the dipatch address to a routine to handle the normal RSP TU58.
                             0799  ;
OC A5   000007E1'EF     DE   0799          MOVAL   L^300$,UCB$L_FPC(R5)     ;Assume this is an MRSP TU58.
                 03     E0   07A1          BBS     #DD_V_USEMRSP,-          ;Branch if we are to use MRSP
         08 00CA C5          07A3                  UCB$W_TU58STS(R5),10$    ;
OC A5   00000810'EF     DE   07A7          MOVAL   L^600$,UCB$L_FPC(R5)     ;Set address for RSP TU58
```

```
     00D4 C5    D0  07AF  10$:   MOVL    UCB$L_DD_MSGPKT(R5),-   ;Get address of message packet
       10 A5        07B3                 UCB$L_FR3(R5)           ;
    14 A5   54  D0  07B5         MOVL    R4,UCB$L_FR4(R5)        ;Save CSR address
                    07B9         DSBINT  UCB$B_DIPL(R5)          ;Lock out device interrupts
64    0040 8F   B0  07C0         MOVW    #DD_RCSR_M_RIE,DD_RCSR(R4);Enable receiver interrupts
          03    A8  07C5         BISW    #UCB$M_TIM!UCB$M_INT,-  ;Set INTERRUPT and TIMEOUT enabled
       64 A5        07C7                 UCB$W_STS(R5)           ;
          34    C1  07C9         ADDL3   #LONG_WAIT,-            ;Compute timeout time for entire transfer.
  00000000'GF       07CB                 G^EXE$GL_ABSTIM,-       ;This will save recomputing it for each
       6C A5        07D0                 UCB$L_DUETIM(R5)        ;byte transferred.
                    07D2  ;
                    07D2  ;      This code is executed for the first byte received from the device.
                    07D2  ;      All succeeding interrupts for bytes bytes received will go through
                    07D2  ;      another path, which is faster.  This is done to save several
                    07D2  ;      instructions that need not be executed again.  Note for this path
                    07D2  ;      the IPL must be restored before returning to the caller's caller.
                    07D2  ;      Failure to do so will cause a BUGCHECK in the fork process dispatcher.
                    07D2  ;      Note that R3 and R4 have already been saved, so it is not necessary
                    07D2  ;      to do so again.  Also note that since interrupts and timeouts are not
                    07D2  ;      disabled until all the data has been transfered, it is not necessary
                    07D2  ;      to set the INTERRUPT and TIMEOUT enable bits for each byte transferred.
                    07D2  ;      For that reason, the timeout time is computed only once, and is large
                    07D2  ;      enough to allow time to transfer the maximum number of bytes possible,
                    07D2  ;      which is 128 bytes of data plus some RSP protocol overhead.
                    07D2  ;
                    07D2  100$:  ENBINT                          ;Allow device interrupts
              05    07D5         RSB                             ;Return
            FDB7    07D6         .WORD   DD_TIME_OUT-.           ;Offset to timeout routine
                    07D8  ;
                    07D8  ;      This is the fast interrupt servic loop for MRSP protocol
                    07D8  ;      data transfers.  This path exists to save testing the MRSP
                    07D8  ;      bit in the UCB for each byte transferred.  Note that the interrupt
                    07D8  ;      is dispatched here to avoid the cost of an RSB back to the device's
                    07D8  ;      interrupt dispatcher.  Of course the interrupt dispatcher JMPs here.
                    07D8  ;
    10 A5   53  7D  07D8  200$:  MOVQ    R3,UCB$L_FR3(R5)        ;Save R3 and R4
            3F  BA  07DC  250$:  POPR    #^M<R0,R1,R2,R3,R4,R5>  ;Restore registers from the interrupt
              02    07DE         REI                             ;Dismiss the interrupt
            FDAE    07DF         .WORD   DD_TIME_OUT-.           ;Offset to timeout routine
                    07E1  ;
                    07E1  ;      Dump the receiver buffer (word sized) into R1.
                    07E1  ;      After saving the data, acknowledge the receipt of the data.
                    07E1  ;      This will cause the TU58 to send the next byte, if any.
                    07E1  ;
    51   02 A4  3C  07E1  300$:  MOVZWL  DD_RBUF(R4),R1          ;Dump receiver buffer to R1
    06 A4   10  B0  07E5         MOVW    #ACK_CHAR,DD_XBUF(R4)   ;Send ack. character
       83   51  90  07E9  400$:  MOVB    R1,(R3)+                ;Put byte in message buffer
     00D8 C5    B7  07EC         DECW    UCB$W_DD_MSGLEN(R5)     ;Decrement byte count
          E6    14  07F0         BGTR    200$                    ;Branch if more to go
                    07F2  ;
                    07F2  ;      The data has been transferred.  Disable device interrupts
                    07F2  ;      and create a fork process to continue on with the I/O operation.
                    07F2  ;
       64   00  B0  07F2  500$:  MOVW    #0,DD_RCSR(R4)          ;Disable receiver interrupts
            03  AA  07F5         BICW    #UCB$M_TIM!UCB$M_INT,-  ;Clear INTERRUPT and TIMEOUT enabled
       64 A5        07F7                 UCB$W_STS(R5)           ;
  FFFFFFDD EF   9F  07F9         PUSHAB  L^250$                  ;Set return address of "caller's caller"
```

```
                                07FF          IOFORK                              ;Get back to fork context
          50   0838 8F     3C   0805          MOVZWL  #SS$_DATAOVERUN,R0          ;Always return overrun status
               009C D5     17   080A          JMP     @UCB$L_DPC(R5)              ;Return
                                080E    ;
                                080E    ;     Check for data overrun.  If an overrun occurred, disable
                                080E    ;     device interrupts and get back to fork context.  Then
                                080E    ;     reset the controller.  Note that no overruns will (should)
                                080E    ;     occur when using MRSP, so this test does not have to be made
                                080E    ;     for MRSP TU58s.
                                080E    ;
                         FD7F   080E          .WORD   DD_TIME_OUT-.               ;Offset to timeout routine
          51     02 A4     3C   0810  600$:   MOVZWL  DD_RBUF(R4),R1              ;Dump receiver buffer to R1
             D1 51 0F      E1   0814          BBC     #DD_RBUF_V_ERROR,R1,400$;Branch if no overrun
 009C C5   FFFFFD46 EF     DE   0818          MOVAL   L^CONT_ERROR,UCB$L_DPC(R5);Return to controller reset routine
                   CF      11   0821          BRB     500$                        ;Branch to common exit code
```

```
                                  0823          .SBTTL  SEND_PACKET_U
                                  0823  ;++
                                  0823  ;
                                  0823  ; SEND_PACKET_U
                                  0823  ;
                                  0823  ; ABSTRACT:       Send a message packet to a unibus TU58.  Because of
                                  0823  ;                 the unique nature of the TU58, this routine is optimized
                                  0823  ;                 for speed.  The TU58 generates an inordinate number of
                                  0823  ;                 interrupts, and can use a large fraction of the processor
                                  0823  ;                 during normal operation.  In order to reduce overhead, some
                                  0823  ;                 extreme measures have been taken in the coding of this routine.
                                  0823  ;
                                  0823  ; INPUT:          R4 = CSR address
                                  0823  ;                 R5 = UCB address
                                  0823  ;                 UCB$L_DD_MSGPKT(R5) = Message packet buffer address
                                  0823  ;                 UCB$W_DD_MSGLEN(R5) = Message packet buffer length
                                  0823  ;
                                  0823  ; OUTPUT:         None.
                                  0823  ;
                                  0823  ; NOTES:
                                  0823  ;
                                  0823  ;                 1) This routine enables the device interrupts at the
                                  0823  ;                    beginning of the message packet and does not
                                  0823  ;                    disable them until the entire message packet has
                                  0823  ;                    been transferred.
                                  0823  ;
                                  0823  ;                 2) The WFIKPCH macro is not used.  Instead, the code
                                  0823  ;                    is placed inline to avoid the cost in time inherent
                                  0823  ;                    in the macro.
                                  0823  ;
                                  0823  ;                 3) To save the overhead of IOFORKing after each byte
                                  0823  ;                    transferred, the routine will not IOFORK until
                                  0823  ;                    the entire message packet has been transferred.
                                  0823  ;
                                  0823  ;--
                                  0823  SEND_PACKET_U:                                  ;Send a message packet (unibus)
                                  0823  ;
                                  0823  ;       The following code needs to be done only once, to
                                  0823  ;       set things up for the interrupt service loop.
                                  0823  ;
           009C C5   8E     D0    0823          MOVL    (SP)+,UCB$L_DPC(R5)             ;Store return address
              0040 8F       AA    0828          BICW    #UCB$M_TIMOUT,-                 ;Clear timeout occurred bit
                 64 A5            082C                  UCB$W_STS(R5)                   ;
 0C A5   00000869'EF        DE    082E          MOVAL   L^300$,UCB$L_FPC(R5)            ;Save interrupt dispatch address
              00D4 C5       D0    0836          MOVL    UCB$L_DD_MSGPKT(R5),-           ;Get address of message packet
                 10 A5            083A                  UCB$L_FR3(R5)                   ;
           14 A5   54       D0    083C          MOVL    R4,UCB$L_FR4(R5)                ;Save CSR address
                                  0840          DSBINT  UCB$B_DIPL(R5)                  ;Lock out device interrupts
  04 A4   0040 8F           B0    0847          MOVW    #DD_XCSR_M_TIE,DD_XCSR(R4);Enable transmitter interrupts
                 03         A8    084D          BISW    #UCB$M_TIM!UCB$M_INT,-          ;Set INTERRUPT and TIMEOUT enabled
                 64 A5            084F                  UCB$W_STS(R5)                   ;
                 34         C1    0851          ADDL3   #LONG_WAIT,-                    ;Compute timeout time for entire transfer.
           00000000'GF            0853                  G^EXE$GL_ABSTIM,-               ;This will save recomputing it for each
                 6C A5            0858                  UCB$L_DUETIM(R5)                ;byte transferred.
                                  085A  ;
                                  085A  ;       This code is executed for the first byte received from the device.
                                  085A  ;       All succeeding interrupts for bytes bytes received will go through
```

```
                     085A  ;       another path, which is faster.  This is done to save several
                     085A  ;       instructions that need not be executed again.  Note for this path
                     085A  ;       the IPL must be restored before returning to the caller's caller.
                     085A  ;       Failure to do so will cause a BUGCHECK in the fork process dispatcher.
                     085A  ;       Note that R3 and R4 have already been saved, so it is not necessary
                     085A  ;       to do so again.  Also note that since interrupts and timeouts are not
                     085A  ;       disabled until all the data has been transfered, it is not necessary
                     085A  ;       to set the INTERRUPT and TIMEOUT enable bits for each byte transferred.
                     085A  ;       For that reason, the timeout time is computed only once, and is large
                     085A  ;       enough to allow time to transfer the maximum number of bytes possible,
                     085A  ;       which is 128 bytes of data plus some RSP protocol overhead.
                     085A  ;
                     085A  100$:   ENBINT                          ;Allow device interrupts
                 05  085D          RSB                             ;Return
               FD2F  085E          .WORD   DD_TIME_OUT-.           ;Offset to timeout routine
                     0860  ;
                     0860  ;       This is the second and faster path through the wait-for-
                     0860  ;       interrupt-and-keep-channel logic.  It is reasonable to have
                     0860  ;       two paths, as the first path is only needed for the first
                     0860  ;       byte sent, and there will always be at least 14 bytes (the
                     0860  ;       length of a command packet) sent.  This path saves two instructions,
                     0860  ;       as it would be necessary to save and restore the IPL in the
                     0860  ;       other path.  Note that the interrupt is dismissed here.  This
                     0860  ;       is to save the necessary RSB to return to the interrupt dispatcher.
                     0860  ;       Of course the driver's interrupt dispatch JMPs to the location
                     0860  ;       stored in UCB$L_FPC within the device UCB.
                     0860  ;
  10 A5      53  7D  0860  200$:   MOVQ    R3,UCB$L_FR3(R5)        ;Save R3 and R4
             3F  BA  0864  250$:   POPR    #^M<R0,R1,R2,R3,R4,R5>  ;Restore registers from the interrupt
                 02  0866          REI                             ;Dismiss the interrupt
               FD26  0867          .WORD   DD_TIME_OUT-.           ;Offset to timeout routine
                     0869  ;
                     0869  ;       Get the data byte from the packet buffer and send it
                     0869  ;       to the device.  If there is more data to send, go through
                     0869  ;       the second wait-for-interrupt-keep-channel path.
                     0869  ;
  06 A4      83  9B  0869  300$:   MOVZBW  (R3)+,DD_XBUF(R4)       ;Send the data to the TU58
   00D8 C5       B7  086D          DECW    UCB$W_DD_MSGLEN(R5)     ;Decrement byte count
             ED  14  0871          BGTR    200$                    ;Branch if more to send
                     0873  ;
                     0873  ;       The data has been transferred.  Disable device interrupts
                     0873  ;       and create a fork process to continue on with the I/O operation.
                     0873  ;
             03  AA  0873          BICW    #UCB$M_TIM!UCB$M_INT,-  ;Clear INTERRUPT and TIMEOUT enabled
          64 A5      0875                  UCB$W_STS(R5)           ;
  04 A4      00  B0  0877          MOVW    #0,DD_XCSR(R4)          ;Disable transmitter interrupts
FFFFFFE3 EF      9F  087B          PUSHAB  L^250$                  ;Set return address of "caller's caller"
                     0881          IOFORK                          ;Get back to fork context
   009C D5       17  0887          JMP     @UCB$L_DPC(R5)          ;Return
```

```
                        088B  ;++
                        088B  ;
                        088B  ; SCSR
                        088B  ;
                        088B  ; ABSTRACT:      Update the transmitter CSR with the data in R1.
                        088B  ;
                        088B  ; INPUT:         R1 = CSR image
                        088B  ;                R4 = CSR address
                        088B  ;
                        088B  ; OUTPUT:        CSR updated
                        088B  ;
                        088B  ; SIDE EFFECTS: None
                        088B  ;
                        088B  ;--
                        088B  SCSR:
              00   E1   088B          BBC     #DD_V_CONSOLE,-           ;Branch if not console device
         04 00CA   C5   088D                  UCB$W_TU58STS(R5),10$     ;
          1E  51   DA   0891          MTPR    R1,#PR$_CSTS              ;Update CSR
              05        0894          RSB                               ;Return
       04 A4  51   B0   0895  10$:    MOVW    R1,DD_XCSR(R4)            ;Update CSR
              05        0899          RSB                               ;Return
```

```
                    089A  ;++
                    089A  ;
                    089A  ; RCSR
                    089A  ;
                    089A  ; ABSTRACT:      Update the receiver CSR with the data in R1.
                    089A  ;
                    089A  ; INPUT:         R1 = CSR image
                    089A  ;                R4 = CSR address
                    089A  ;
                    089A  ; OUTPUT:        CSR updated
                    089A  ;
                    089A  ; SIDE EFFECTS: None
                    089A  ;
                    089A  ;--
                    089A  RCSR:
          00    E1  089A             BBC     #DD_V_CONSOLE,-          ;Branch if not console device
  04 00CA C5        089C                     UCB$W_TU58STS(R5),10$    ;
     1C   51    DA  08A0             MTPR    R1,#PR$_CSRS             ;Update CSR
                05  08A3             RSB                              ;Return
     64   51    B0  08A4  10$:       MOVW    R1,DD_RCSR(R4)           ;Update CSR
                05  08A7             RSB                              ;Return
```

```
                              08A8  ;++
                              08A8  ;
                              08A8  ; CHECKSUM
                              08A8  ;
                              08A8  ; ABSTRACT:      Given a message packet buffer address and length,
                              08A8  ;                compute the checksum of the buffer.  The checksum
                              08A8  ;                is formed by adding adjacent words in the buffer and
                              08A8  ;                adding in the carry (if any).
                              08A8  ;
                              08A8  ; INPUT:         R1 = Message packet buffer address.
                              08A8  ;                R2 = Message packet buffer length.
                              08A8  ;                R5 = UCB address.
                              08A8  ;
                              08A8  ; OUTPUT:        R0 = Checksum (low word)
                              08A8  ;
                              08A8  ;
                              08A8  ; SIDE EFFECTS: R1,R2 are overwritten.
                              08A8  ;
                              08A8  ;--
                              08A8  CHECKSUM:                                        ;Create massage packet checksum
                     52    DD 08A8          PUSHL   R2                      ;Save byte count on stack
                     50    D4 08AA          CLRL    R0                      ;Clear output cell
       52    52   FF 8F    78 08AC          ASHL    #-1,R2,R2               ;Convert byte count to word count
             50   81       A0 08B1  10$:    ADDW    (R1)+,R0                ;Add word to current checksum
             50   00       D8 08B4          ADWC    #0,R0                   ;Add in carry, if any
                  F7 52    F5 08B7          SOBGTR  R2,10$                  ;Branch if not done
                              08BA  ;
                              08BA  ;       In order to make CHECKSUM work for odd byte counts, we
                              08BA  ;       have add the last byte on after calculating the checksum
                              08BA  ;       for the rest of the buffer.
                              08BA  ;
                  01 8E    E8 08BA          BLBS    (SP)+,20$               ;Branch if odd  byte count
                           05 08BD          RSB                             ;Return
             52   81       9A 08BE  20$:    MOVZBL  (R1)+,R2                ;Get the last byte in the buffer
             50   52       A0 08C1          ADDW    R2,R0                   ;Add it to the checksum
             50   00       D8 08C4          ADWC    #0,R0                   ;Add in carry, if any
                           05 08C7          RSB                             ;Return
```

```
                         08C8       ;++
                         08C8       ;
                         08C8       ; SET_FUNCTION
                         08C8       ;
                         08C8       ; ABSTRACT:     Use the function code in the IRP to determine which TU58
                         08C8       ;               function to perform.  In effect, map the system function
                         08C8       ;               code into a TU58 function code.  The results are returned
                         08C8       ;               in the low bytes of R0 and R1.
                         08C8       ;
                         08C8       ; INPUT:        R5 = UCB ADDRESS
                         08C8       ;
                         08C8       ; OUTPUT:       R0 = TU58 FUNCTION CODE
                         08C8       ;               R1 = TU58 FUNCTION CODE MODIFIER
                         08C8       ;               Mapping bit in UCB$W_TU58STS is set or cleared, depending
                         08C8       ;               on the function requested.
                         08C8       ;
                         08C8       ; SIDE EFFECTS: R3 = IRP ADDRESS
                         08C8       ;
                         08C8       ; NOTE:         Some functions listed as legal in the FDT are not yet
                         08C8       ;               implemented in the TU58 firmware. These functions will
                         08C8       ;               be trapped here and flagged as illegal function codes.
                         08C8       ;               If an illegal function is requested, exit the driver
                         08C8       ;               without sending a command packet.
                         08C8       ;
                         08C8       ;--
                         08C8       SET_FUNCTION:                                   ;Determine TU58 function
                         08C8       ;
                         08C8       ;       If this TU58 speaks MRSP, assume that the function will
                         08C8       ;       use MRSP unless the user explicitly requests otherwise.
                         08C8       ;
       20 A3     B0      08C8               MOVW    IRP$W_FUNC(R3),-        ;Copy function code to UCB
     009A C5             08CB                       UCB$W_FUNC(R5)          ;
          02     E1      08CE               BBC     #DD_V_MRSP,-            ;Branch if not an MRSP TU58
  11 00CA C5             08D0                       UCB$W_TU58STS(R5),20$   ;
          03     E3      08D4               BBCS    #DD_V_USEMRSP,-         ;Set to use MRSP
  00 00CA C5             08D6                       UCB$W_TU58STS(R5),10$   ;
          06     E1      08DA       10$:    BBC     #IO$V_NOMRSP,-          ;Branch if user wants to use MRSP
    06 20 A3             08DC                       IRP$W_FUNC(R3),20$      ;
          03     E5      08DF               BBCC    #DD_V_USEMRSP,-         ;Do not use MRSP for this operation
  00 00CA C5             08E1                       UCB$W_TU58STS(R5),20$   ;
          51     D4      08E5       20$:    CLRL    R1                      ;Assume no function modifier
  53   58 A5     D0      08E7               MOVL    UCB$L_IRP(R5),R3        ;Get IRP address
          00     EF      08EB               EXTZV   #IRP$V_FCODE,-          ;Extract the function code
          06             08ED                       #IRP$S_FCODE,-          ;
  50   20 A3             08EE                       IRP$W_FUNC(R3),R0       ;Put function code in R0
     50   0C     91      08F1               CMPB    #IO$_READPBLK,R0        ;READ physical block?
          13     13      08F4               BEQL    30$                     ;
     50   0B     91      08F6               CMPB    #IO$_WRITEPBLK,R0       ;WRITE physical block?
          13     13      08F9               BEQL    40$                     ;
     50   09     91      08FB               CMPB    #IO$_SEARCH,R0          ;SEARCH/POSITION request?
          1B     13      08FE               BEQL    50$                     ;
     50   1D     91      0900               CMPB    #IO$_DIAGNOSE,R0        ;DIAGNOSE function?
          1A     12      0903               BNEQ    60$                     ;Branch if not
     50   07     90      0905               MOVB    #DD_CODE_DIAG,R0        ;Set diagnose function
          05             0908               RSB                             ;Return
     50   02     90      0909       30$:    MOVB    #DD_CODE_READ,R0        ;Set READ function
          03     11      090C               BRB     45$                     ;Check for function modifier
```

```
          50   03  90 090E  40$:    MOVB    #DD_CODE_WRITE,R0       ;Set WRITE function
               0E  E0 0911  45$:    BBS     #IO$V_DATACHECK,-       ;Branch if DATACHECK requested
         01 20 A3     0913                  IRP$W_FUNC(R3),46$      ;
               05     0916          RSB                             ;Return
          51   01  90 0917  46$:    MOVB    #DD_CODE_DATACHECK,R1   ;Set DATACHECK op code modifier
               05     091A          RSB                             ;Return
          50   05  90 091B  50$:    MOVB    #DD_CODE_SEARCH,R0      ;Set SEARCH function
               05     091E          RSB                             ;Return
                      091F  ;
                      091F  ; Trap UNLOAD, AVAILABLE and PACKACK functions here.  These function simply
                      091F  ; set or clear the valid bit in the UCB and return with success status.  A
                      091F  ; command packet is not sent.
                      091F  ;
                      091F
          50   01  91 091F  60$:    CMPB    #IO$_UNLOAD,R0          ; Unload function?
               05  13 0922          BEQL    65$                     ; Branch if so.
          50   11  91 0924          CMPB    #IO$_AVAILABLE,R0       ; Available function?
               08  12 0927          BNEQ    70$                     ; Branch if not.
        0800   8F  AA 0929  65$:    BICW    #UCB$M_VALID,-          ; Clear software valid bit.
            64 A5     092D                  UCB$W_STS(R5)           ;
               0B  11 092F          BRB     80$                     ; Branch to common code.
          50   08  91 0931  70$:    CMPB    #IO$_PACKACK,R0         ; Pack acknowledge function?
               0B  12 0934          BNEQ    100$                    ; Must be illegal function code.
        0800   8F  A8 0936          BISW    #UCB$M_VALID,-          ; Set software volume valid bit.
            64 A5     093A                  UCB$W_STS(R5)           ;
          50   01  D0 093C  80$:    MOVL    #SS$_NORMAL,R0          ; Return success status.
               05  11 093F          BRB     200$                    ; Branch to exit.
                      0941
                      0941  ;
                      0941  ;       If we fall through all the tests, it is an illegal function.
                      0941  ;
   50   00F4   8F  B0 0941  100$:   MOVW    #SS$_ILLIOFUNC,R0       ;Put error status in R0
               8E  D5 0946  200$:   TSTL    (SP)+                   ;Discard return address
             FBF1  31 0948          BRW     DD_EXIT                 ;Exit the driver NOW
                      094B
                      094B  DD_END:                                 ;ADDRESS OF LAST LOCATION IN DRIVER
                      094B
                      094B          .END
```

```
$$$                            = 00000020 R       02
$$OP                           = 00000002
ACK_CHAR                       = 00000010
ACP$ACCESS                       ********  X      03
ACP$DEACCESS                     ********  X      03
ACP$MODIFY                       ********  X      03
ACP$MOUNT                        ********  X      03
ACP$READBLK                      ********  X      03
ACP$WRITEBLK                     ********  X      03
AT$_UBA                        = 00000001
BIT...                         = 00000004
CHECKSUM                         000008A8 R       03
CMD_PKT_HDR                    = 00000A02
CONT_ERROR                       00000564 R       03
CRB$C_INTD                     = 00000024
CRB$L_INTD2                    = 00000048
DC$_DISK                       = 00000001
DD$DDT                           00000000 RG      03
DDB$K_TAPE                     = 00000004
DDB$L_ACPD                     = 00000010
DDB$L_DDT                      = 0000000C
DD_CHKSM_LEN                   = 00000002
DD_CODE_DATACHECK              = 00000001
DD_CODE_DETECT                 = 0000000A
DD_CODE_DIAG                   = 00000007
DD_CODE_READ                   = 00000002
DD_CODE_SEARCH                 = 00000005
DD_CODE_WRITE                  = 00000003
DD_CP_BCNT                     = 0000000A
DD_CP_LEN                      = 0000000E
DD_DATA                        = 00000002
DD_DPBCNT                      = 00000001
DD_DPBUF_LEN                   = 00000084
DD_END                           0000094B R       03
DD_EP_LEN                      = 0000000E
DD_ERRBUF_LEN                  = 0000002C
DD_ERRBUF_LEN_L                = 0000000B
DD_EXIT                          0000053C R       03
DD_FLAG_CNTL                   = 00000002
DD_FLAG_CONT                   = 00000010
DD_FLAG_DATA                   = 00000001
DD_FLAG_INIT                   = 00000004
DD_FLAG_XOFF                   = 00000013
DD_FUNCTABLE                     00000038 R       03
DD_INT                           0000063C RG      03
DD_MAX_PKT_BCNT                = 00000080
DD_M_CONSOLE                   = 00000001
DD_M_MRSP                      = 00000004
DD_M_PASS1                     = 00000002
DD_M_USEMRSP                   = 00000008
DD_NUM_REGS                    = 00000004
DD_PWF                           000005B5 R       03
DD_RBUF                          00000002
DD_RBUF_V_ERROR                = 0000000F
DD_RCSR                          00000000
DD_RCSR_M_RDONE                = 00000080
DD_RCSR_M_RIE                  = 00000040
DD_REGDUMP                       00000621 R       03
DD_RESTARTIO                     00000273 R       03
DD_RETRY                         00000572 R       03
DD_SAVEREG                       00000660 R       03
DD_STARTIO                       0000026D R       03
DD_STS_BADCKSM                 = FFFFFFC6
DD_STS_BADDIAG                 = FFFFFFFF
DD_STS_BADINIT                 = FFFFFFC7
DD_STS_BADOPCD                 = FFFFFFD0
DD_STS_BADRCD                  = FFFFFFC9
DD_STS_BADUNIT                 = FFFFFFF8
DD_STS_BLKNF                   = FFFFFFE0
DD_STS_DATACHK                 = FFFFFFEF
DD_STS_DATAOVR                 = FFFFFFC5
DD_STS_MTRSTOP                 = FFFFFFDF
DD_STS_NOCART                  = FFFFFFF7
DD_STS_NORMAL                  = 00000000
DD_STS_PARTOP                  = FFFFFFFE
DD_STS_RETRY                   = 00000001
DD_STS_WRTLCK                  = FFFFFFF5
DD_TIME_OUT                      0000058D R       03
DD_TRYAGN                        00000303 R       03
DD_UNIT_INIT                     000000CC R       03
DD_UNSOLNT                       0000064E R       03
DD_V_CONSOLE                   = 00000000
DD_V_MRSP                      = 00000002
DD_V_PASS1                     = 00000001
DD_V_USEMRSP                   = 00000003
DD_XBUF                          00000006
DD_XCSR                          00000004
DD_XCSR_M_BREAK                = 00000001
DD_XCSR_M_TIE                  = 00000040
DD_XCSR_V_TRDY                 = 00000007
DEV$M_AVL                      = 00040000
DEV$M_DIR                      = 00000008
DEV$M_ELG                      = 00400000
DEV$M_FOD                      = 00004000
DEV$M_IDV                      = 04000000
DEV$M_NNM                      = 00000200
DEV$M_ODV                      = 08000000
DEV$M_RND                      = 10000000
DEV$M_SHR                      = 00010000
DPT$C_LENGTH                   = 00000038
DPT$C_VERSION                  = 00000004
DPT$INITAB                       00000038 R       02
DPT$M_SVP                      = 00000002
DPT$REINITAB                     00000085 R       02
DPT$TAB                          00000000 R       02
DT$_TU58                       = 0000000E
DTECT_PKT_CHKSM                = 00000A0C
DTECT_PKT_HDR                  = 000A0A02
DUMP_RECVR_BUF                   00000216 R       03
DYN$C_BUFIO                    = 00000013
DYN$C_CRB                      = 00000005
DYN$C_DDB                      = 00000006
DYN$C_DPT                      = 0000001E
DYN$C_UCB                      = 00000010
```

```
EMB$L_DV_REGSAV          = 0000004E
END_PACKET                 000004C0 R      03
ERL$DEVICERR               ********    X   03
ERL$DEVICTMO               ********    X   03
EXE$ALONONPAGED            ********    X   03
EXE$GB_CPUTYPE             ********    X   03
EXE$GL_ABSTIM              ********    X   03
EXE$GL_NONPAGED            ********    X   03
EXE$GL_TENUSEC             ********    X   03
EXE$GL_UBDELAY             ********    X   03
EXE$IOFORK                 ********    X   03
EXE$LCLDSKVALID            ********    X   03
EXE$ONEPARM                ********    X   03
EXE$SENSEMODE              ********    X   03
EXE$SETCHAR                ********    X   03
EXE$ZEROPARM               ********    X   03
FUNCTAB_LEN              = 00000094
GET_PACKET_C               0000068D R      03
GET_PACKET_U               0000078E R      03
GET_RESPONSE               00000341 R      03
HIGH_DIPL                = 00000017
IDB$L_CSR                = 00000000
INIT_UNIT                  00000136 R      03
IO$V_DATACHECK           = 0000000E
IO$V_INHRETRY            = 0000000F
IO$V_NOMRSP              = 00000006
IO$_ACCESS               = 00000032
IO$_ACPCONTROL           = 00000038
IO$_AVAILABLE            = 00000011
IO$_CREATE               = 00000033
IO$_DEACCESS             = 00000034
IO$_DELETE               = 00000035
IO$_DIAGNOSE             = 0000001D
IO$_MODIFY               = 00000036
IO$_MOUNT                = 00000039
IO$_PACKACK              = 00000008
IO$_READLBLK             = 00000021
IO$_READPBLK             = 0000000C
IO$_READVBLK             = 00000031
IO$_SEARCH               = 00000009
IO$_SENSECHAR            = 0000001B
IO$_SENSEMODE            = 00000027
IO$_SETCHAR              = 0000001A
IO$_SETMODE              = 00000023
IO$_UNLOAD               = 00000001
IO$_VIRTUAL              = 0000003F
IO$_WRITELBLK            = 00000020
IO$_WRITEPBLK            = 0000000B
IO$_WRITEVBLK            = 00000030
IOC$DIAGBUFILL             ********    X   03
IOC$MNTVER                 ********    X   03
IOC$MOVFRUSER              ********    X   03
IOC$MOVFRUSER2             ********    X   03
IOC$MOVTOUSER              ********    X   03
IOC$MOVTOUSER2             ********    X   03
IOC$RELCHAN                ********    X   03
IOC$REQCOM                 ********    X   03
IOC$REQPCHANL              ********    X   03
IOC$RETURN                 ********    X   03
IRP$L_MEDIA              = 00000038
IRP$L_SVAPTE             = 0000002C
IRP$S_FCODE              = 00000006
IRP$V_DIAGBUF            = 00000007
IRP$V_FCODE              = 00000000
IRP$W_FUNC               = 00000020
IRP$W_STS                = 0000002A
LONG_WAIT                = 00000034
MASKH                    = 00000008
MASKL                    = 04000000
MAXCHRTIM                = 00004E20
MRSP_M_SWITCH            = 00000008
MRSP_V_SWITCH            = 00000003
PR$_CSRD                 = 0000001D
PR$_CSRS                 = 0000001C
PR$_CSTD                 = 0000001F
PR$_CSTS                 = 0000001E
PR$_IPL                  = 00000012
PR$_SID_TYP730           = 00000003
PR$_SID_TYP750           = 00000002
RCSR                       0000089A R      03
READ_DATA                  0000037D R      03
RESET_CONTROLLER           000005B8 R      03
SCSR                       0000088B R      03
SEND_PACKET_C              00000720 R      03
SEND_PACKET_U              00000823 R      03
SET_FUNCTION               000008C8 R      03
SHORT_WAIT               = 00000002
SIZ...                   = 00000001
SND_PKT_NOINT              000001B2 R      03
SS$_CTRLERR              = 00000054
SS$_DATACHECK            = 0000005C
SS$_DATAOVERUN           = 00000838
SS$_ILLIOFUNC            = 000000F4
SS$_MEDOFL               = 000001A4
SS$_NONEXDRV             = 000001C4
SS$_NORMAL               = 00000001
SS$_PARITY               = 000001F4
SS$_TIMEOUT              = 0000022C
SS$_WRITLCK              = 0000025C
TWO_INITS                = 00000404
UCB$B_DD_BCNTH             000000EB
UCB$B_DD_BCNTL             000000EA
UCB$B_DD_BLKH              000000ED
UCB$B_DD_BLKL              000000EC
UCB$B_DD_CPBCNT            000000E3
UCB$B_DD_CPFLAG            000000E2
UCB$B_DD_EPBCNT            000000F1
UCB$B_DD_EPCODE            000000F2
UCB$B_DD_EPFLAG            000000F0
UCB$B_DD_EPSTS             000000F3
UCB$B_DD_EPUNIT            000000F4
UCB$B_DD_OPCODE            000000E4
UCB$B_DD_OPMOD             000000E5
UCB$B_DD_SEQH              000000E9
```

```
UCB$B_DD_SEQL                    000000E8
UCB$B_DD_SWITCH                  000000E7
UCB$B_DD_UNIT                    000000E6
UCB$B_DEVCLASS                 = 00000040
UCB$B_DEVTYPE                  = 00000041
UCB$B_DIPL                     = 0000005E
UCB$B_ERTCNT                   = 00000080
UCB$B_ERTMAX                   = 00000081
UCB$B_FIPL                     = 0000000B
UCB$B_OFFNDX                   = 000000CA
UCB$B_SECTORS                  = 00000044
UCB$B_SPR                      = 00000091
UCB$B_TRACKS                   = 00000045
UCB$K_DD_CPADDR                  000000E2
UCB$K_DD_EPADDR                  000000F0
UCB$K_DD_REGBUF                  000000DA
UCB$K_DD_UCBLEN                  000000FE
UCB$K_LCL_DISK_LENGTH          = 000000CC
UCB$L_CRB                      = 00000024
UCB$L_DD_BUFWIN                  000000CC
UCB$L_DD_DPTR                    000000D0
UCB$L_DD_MSGPKT                  000000D4
UCB$L_DEVCHAR                  = 00000038
UCB$L_DEVCHAR2                 = 0000003C
UCB$L_DPC                      = 0000009C
UCB$L_DUETIM                   = 0000006C
UCB$L_FPC                      = 0000000C
UCB$L_FR3                      = 00000010
UCB$L_FR4                      = 00000014
UCB$L_IRP                      = 00000058
UCB$L_MAXBLOCK                 = 000000B0
UCB$L_MEDIA                    = 000000BC
UCB$L_MEDIA_ID                 = 0000008C
UCB$L_SVAPTE                   = 00000078
UCB$M_DIAGBUF                  = 00000002
UCB$M_INT                      = 00000002
UCB$M_TIM                      = 00000001
UCB$M_TIMOUT                   = 00000040
UCB$M_VALID                    = 00000800
UCB$V_INT                      = 00000001
UCB$V_ONLINE                   = 00000004
UCB$V_POWER                    = 00000005
UCB$V_TIMOUT                   = 00000006
UCB$W_BCNT                     = 0000007E
UCB$W_BCR                      = 000000C0
UCB$W_CYLINDERS                = 00000046
UCB$W_DD_ABCNT                   000000F8
UCB$W_DD_CPCKSM                  000000EE
UCB$W_DD_EPCKSM                  000000FC
UCB$W_DD_MSGLEN                  000000D8
UCB$W_DD_RBUF                    000000E0
UCB$W_DD_RCSR                    000000DE
UCB$W_DD_SMSTS                   000000FA
UCB$W_DD_XBUF                    000000DC
UCB$W_DD_XCSR                    000000DA
UCB$W_DEVBUFSIZ                = 00000042
UCB$W_DEVSTS                   = 00000068
UCB$W_FUNC                     = 0000009A
UCB$W_STS                      = 00000064
UCB$W_TU58STS                  = 000000CA
UCB$W_UNIT                     = 00000054
VEC$L_IDB                      = 00000008
VEC$L_UNITINIT                 = 00000018
WRITE_DATA                       0000043A R     03
WRITE_WAIT                     = 00000002
```

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	000000FE	(254.)	01	(1.)	NOPIC	USR	CON	ABS	LCL	NOSHR	EXE	RD	WRT	NOVEC	BYTE
$$$105_PROLOGUE	0000009A	(154.)	02	(2.)	NOPIC	USR	CON	REL	LCL	NOSHR	EXE	RD	WRT	NOVEC	BYTE
$$$115_DRIVER	0000094B	(2379.)	03	(3.)	NOPIC	USR	CON	REL	LCL	NOSHR	EXE	RD	WRT	NOVEC	LONG

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	36	00:00:00.03	00:00:02.86
Command processing	130	00:00:00.39	00:00:03.23
Pass 1	614	00:00:19.05	00:01:11.16
Symbol table sort	0	00:00:02.67	00:00:06.95
Pass 2	384	00:00:04.85	00:00:16.80
Symbol table output	19	00:00:00.21	00:00:01.39
Psect synopsis output	0	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	1185	00:00:27.22	00:01:42.41

The working set limit was 2400 pages.
159747 bytes (313 pages) of virtual memory were used to buffer the intermediate code.
There were 130 pages of symbol table space allocated to hold 2399 non-local and 105 local symbols.
2159 source lines were read in Pass 1, producing 22 object records in Pass 2.
51 pages of virtual memory were used to define 48 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA28:[SYS.OBJ]LIB.MLB;1	29
_255DUA28:[SYSLIB]STARLET.MLB;2	10
TOTALS (all libraries)	39

2612 GETS were required to define 39 macros.

There were no errors, warnings or information messages.

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

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

DBDRIVER
LIS

CRDRIVER
LIS

CVDRIVER
LIS

DDDRIVER
LIS

DDCMP
LIS

0109 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

DQDRIVER
LIS

DLDRIVER
LIS

DRDRIVER
LIS

DMDRIVER
LIS