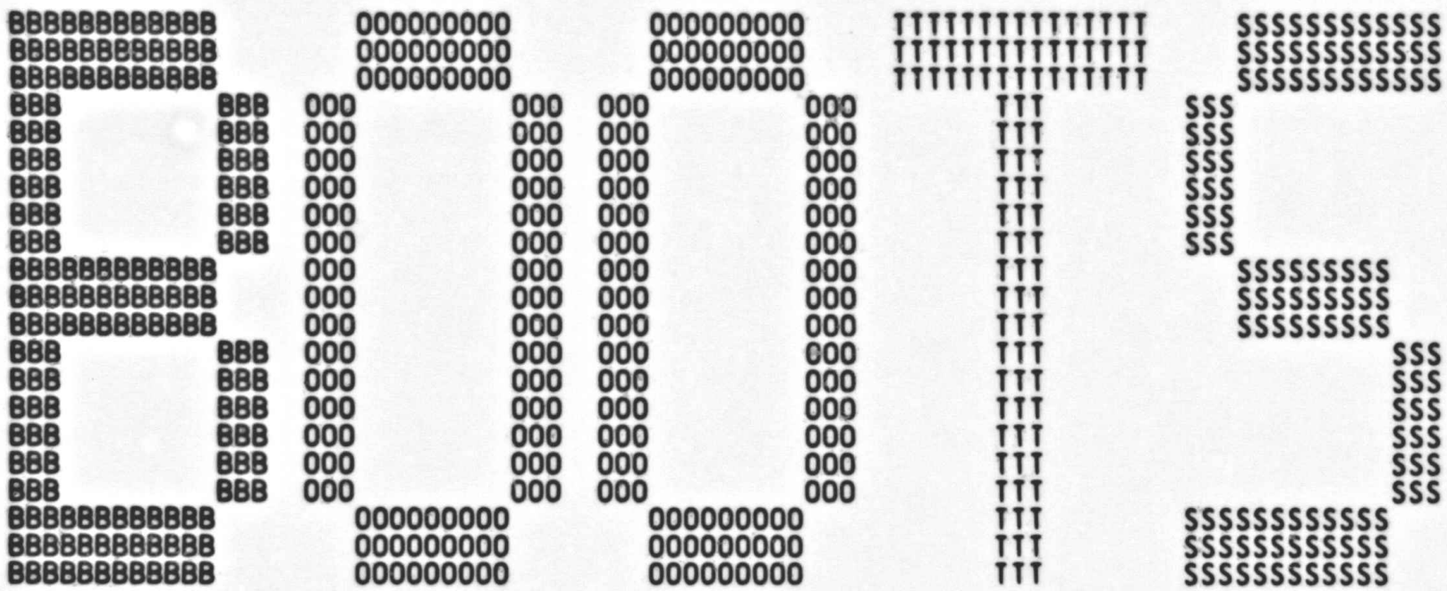
BBBBBBBBBBB OOOOOOOOO OOOOOOOOO TTTTTTTTTTTTTTT SSSSSSSSSSS
BBB BBB OOO OOO OOO OOO TTT SSS
BBBBBBBBBBB OOO OOO OOO OOO TTT SSSSSSSSS
BBB BBB OOO OOO OOO OOO TTT SSS
BBBBBBBBBBB OOOOOOOOO OOOOOOOOO TTT SSSSSSSSSSS

```
DDDDDDDD    DDDDDDDD    BBBBBBBB    TTTTTTTTTT  DDDDDDDD    RRRRRRRR      IIIIII    VV      VV  RRRRRRRR
DDDDDDDD    DDDDDDDD    BBBBBBBB    TTTTTTTTTT  DDDDDDDD    RRRRRRRR      IIIIII    VV      VV  RRRRRRRR
DD      DD  DD      DD  BB      BB      TT      DD      DD  RR      RR      II      VV      VV  RR      RR
DD      DD  DD      DD  BB      BB      TT      DD      DD  RR      RR      II      VV      VV  RR      RR
DD      DD  DD      DD  BB      BB      TT      DD      DD  RR      RR      II      VV      VV  RR      RR
DD      DD  DD      DD  BB      BB      TT      DD      DD  RR      RR      II      VV      VV  RR      RR
DD      DD  DD      DD  BBBBBBBB        TT      DD      DD  RRRRRRRR        II      VV      VV  RRRRRRRR
DD      DD  DD      DD  BBBBBBBB        TT      DD      DD  RRRRRRRR        II      VV      VV  RRRRRRRR
DD      DD  DD      DD  BB      BB      TT      DD      DD  RR   RR         II      VV      VV  RR   RR
DD      DD  DD      DD  BB      BB      TT      DD      DD  RR   RR         II      VV      VV  RR   RR
DD      DD  DD      DD  BB      BB      TT      DD      DD  RR    RR        II        VV  VV    RR    RR      ....
DD      DD  DD      DD  BB      BB      TT      DD      DD  RR    RR        II        VV  VV    RR    RR      ....
DDDDDDDD    DDDDDDDD    BBBBBBBB        TT      DDDDDDDD    RR     RR    IIIIII        VV      RR     RR     ....
DDDDDDDD    DDDDDDDD    BBBBBBBB        TT      DDDDDDDD    RR     RR    IIIIII        VV      RR     RR     ....

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

Table of contents

```
0000     1          .TITLE  DDBTDRIVR - CONSOLE TU58 BOOT DRIVER
0000     2          .IDENT  'V04-000'
0000     3
0000     4 ;
0000     5 ;*****************************************************************************
0000     6 ;*                                                                           *
0000     7 ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
0000     8 ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000     9 ;*  ALL RIGHTS RESERVED.                                                     *
0000    10 ;*                                                                           *
0000    11 ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
0000    12 ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000    13 ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
0000    14 ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
0000    15 ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
0000    16 ;*  TRANSFERRED.                                                             *
0000    17 ;*                                                                           *
0000    18 ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
0000    19 ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
0000    20 ;*  CORPORATION.                                                             *
0000    21 ;*                                                                           *
0000    22 ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
0000    23 ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
0000    24 ;*                                                                           *
0000    25 ;*                                                                           *
0000    26 ;*****************************************************************************
0000    27 ;
0000    28
0000    29 ;++
0000    30 ; FACILITY:     BOOTS
0000    31 ;
0000    32 ; ABSTRACT:
0000    33 ;       This module contains the bootstrap device driver for the
0000    34 ;       console TU58.
0000    35 ;
0000    36 ; ENVIRONMENT:  IPL 31, kernel mode, code must be PIC
0000    37 ;
0000    38 ; AUTHOR:  Steve Beckhardt,     Creation Date:  1-Nov-1979
0000    39 ;
0000    40 ; MODIFIED BY:
0000    41 ;
0000    42 ;       V03-002 RLRCPUDISP      Robert L. Rappaport     15-Jun-1983
0000    43 ;               Recode CPUDISP macros to use new format.
0000    44 ;
0000    45 ;       V03-001 TCM0003         Trudy C. Matthews       15-Feb-1983
0000    46 ;               Update CPUDISP to include 790 path.
0000    47 ;
0000    48 ;--
```

```
                     50          .SBTTL  DECLARATIONS
                     51 ;
                     52 ; INCLUDE FILES:
                     53 ;
                     54
                     55          $BTDDEF                         ; Boot device types
                     56          $IODEF                          ; I/O function codes
                     57          $PRDEF                          ; Processor registers
                     58          $PTEDEF                         ; PTE definitions
                     59          $RPBDEF                         ; RPB offsets
                     60          $SSDEF                          ; Status codes
                     61          $VADEF                          ; Virtual address fields
                     62
                     63 ;
                     64 ; MACROS:
                     65 ;
                     66
                     67 ;
                     68 ; EQUATED SYMBOLS:
                     69 ;
                     70
                     71 ;
                     72 ; CONSOLE TU58 REGISTER DEFINITIONS
                     73 ;
                     74
                     75          $DEFINI DD                      ;START OF REGISTER DEFINITIONS
                     76
                     77          _VIELD  CSRS,0,<-               ;START OF PROC. REG. CSRS DEFS.
                     78                  <,6>,-                  ; UNUSED
                     79                  <IE,,M>,-               ; INTERRUPT ENABLE
                     80                  <DONE,,M>,-             ; DONE
                     81          >
                     82
                     83          _VIELD  CSRD,0,<-               ;START OF PROC. REG. CSRD DEFS.
                     84                  <DATA,8>,-              ; DATA
                     85                  <,7>,-                  ; UNUSED
                     86                  <ERROR,,M>,-            ; ERROR
                     87          >
                     88
                     89          _VIELD  CSTS,0,<-               ;START OF PROC. REG. CSTS DEFS.
                     90                  <BREAK,,M>,-            ; BREAK
                     91                  <,5>,-                  ; UNUSED
                     92                  <IE,,M>,-               ; INTERRUPT ENABLE
                     93                  <READY,,M>,-            ; READY
                     94          >
                     95
                     96          _VIELD  CSTD,0,<-               ;START OF PROC. REG. CSTD DEFS.
                     97                  <DATA,8>,-              ; DATA
                     98          >
                     99
                    100          ;
                    101          ; PROTOCOL FLAGS
                    102          ;
                    103
00000001   0000     104 DD_DATA = 1
00000002   0000     105 DD_CNTRL = 2
00000004   0000     106 DD_INIT = 4
```

```
00000010  0000   107 DD_CONTINUE = 16
          0000   108
          0000   109          ;
          0000   110          ; FUNCTION CODES
          0000   111          ;
          0000   112
00000002  0000   113 DD_READ = 2
00000003  0000   114 DD_WRITE = 3
00000040  0000   115 DD_ENDPKT = 64
          0000   116
          0000   117          ;
          0000   118          ; SWITCH BITS
          0000   119          ;
          0000   120
00000003  0000   121 MRSP_SWITCH = 3
          0000   122
          0000   123          $DEFEND DD
          0000   124
          0000   125 ;
          0000   126 ; Boot driver table entry
          0000   127 ;
          0000   128
          0000   129          $BOOT_DRIVER    DEVTYPE = BTD$K_CONSOLE,-  ; Device type (console)
          0000   130                          ACTION = DD_SELECT,-       ; Action routine
          0000   131                          SIZE = CONSDD_DRVSIZ,-     ; Driver size
          0000   132                          ADDR = CONSDD_START,-      ; Driver start
          0000   133                          ENTRY = CONSDD_DRIVER,-    ; Driver entry
          0000   134                          DRIVRNAME = DDNAME         ; Driver file name
          0000   135
```

```
                        0000   137          .SBTTL  DD_SELECT - Select correct CPUs for this driver
                        0000   138
                        0000   139 ;++
                        0000   140 ; FUNCTIONAL DESCRIPTION:
                        0000   141 ;
                        0000   142 ;       This routine is an action routine called by the boot driver
                        0000   143 ;       select code to determine if this is a cpu that has a TU58
                        0000   144 ;       for a console.
                        0000   145 ;
                        0000   146 ; CALLING SEQUENCE:
                        0000   147 ;
                        0000   148 ;       JSB     DD_SELECT
                        0000   149 ;
                        0000   150 ; INPUT PARAMETERS:
                        0000   151 ;
                        0000   152 ;       EXE$GB_CPUTYPE          Global variable containing cpu type
                        0000   153 ;
                        0000   154 ; OUTPUT PARAMETERS:
                        0000   155 ;
                        0000   156 ;       R0      0 = Do not use this driver
                        0000   157 ;               1 = Use this driver
                        0000   158 ;
                        0000   159 ;       MRSP    1 if this CPU has a TU58 that speaks MRSP, 0 if not.
                        0000   160 ;
                        0000   161 ;               It is assumed that certain CPUs must have MRSP TU58s.
                        0000   162 ;               Currently, only the 11/730 must have an MRSP TU58.
                        0000   163 ;               The 11/780 and 11/730 may have MRSP TU58s, but it is
                        0000   164 ;               not essential that MRSP be used.
                        0000   165 ;--
                        0000   166
                        0000   167
                        0000   168 CONSDD_START:                                ; Label to mark start of driver
                        0000   169
                        0000   170 DD_SELECT:
00000044'GF   94        0000   171         CLRB    G^MRSP                      ; Assume TU58 does not speak MRSP
         50   D4        0006   172         CLRL    R0                          ; Initialize R0
                        0008   173         CPUDISP <<780,CPU_780>,-            ; 11/780
                        0008   174                  <750,CPU_750>,-            ; 11/750
                        0008   175                  <730,CPU_730>,-            ; 11/730
                        0008   176                  <790,CPU_790>>,-           ; 11/790
                        0008   177                 ENVIRON=VMB;
                        0035   178 CPU_780:
                        0035   179 CPU_790:
              05        0035   180         RSB                                 ; Do not use this driver
                        0036   181
                        0036   182 CPU_730:
00000044'GF   D6        0036   183         INCL    G^MRSP                      ; Use MRSP (and fall through to common code)
                        003C   184
                        003C   185 CPU_750:
         50   D6        003C   186         INCL    R0                          ; Use this driver
              05        003E   187         RSB
```

```
                        003F    189          .SBTTL  Console TU58 Driver
                        003F    190
                        003F    191 ;++
                        003F    192 ;
                        003F    193 ; Inputs:
                        003F    194 ;
                        003F    195 ;       R1      Address of page table for virtual -> physical mapping
                        003F    196 ;       R2      Base VPN of transfer (Bits 29:9 of R10)
                        003F    197 ;       R5      LBN for current piece of transfer
                        003F    198 ;       R8      Size of transfer in bytes
                        003F    199 ;       R9      Address of the RPB
                        003F    200 ;       R10     Starting address of transfer
                        003F    201 ;
                        003F    202 ;       FUNC(AP)  I/O operation (IO$_READLBLK or IO$_WRITELBLK only)
                        003F    203 ;       MODE(AP)  Address interpretation mode:
                        003F    204 ;                    0 -> Physical,  1 -> Virtual
                        003F    205 ;
                        003F    206 ; Outputs:
                        003F    207 ;
                        003F    208 ;       R0      Status code:
                        003F    209 ;
                        003F    210 ;               SS$_NORMAL      Successful transfer
                        003F    211 ;               SS$_BUFBYTALI   Odd byte count
                        003F    212 ;               SS$_CTRLERR     Fatal controller error
                        003F    213 ;--
00000010                003F    214 FUNC = 16
00000014                003F    215 MODE = 20
                        003F    216
                        003F    217
                        003F    218 ;
                        003F    219 ; OWN STORAGE:
                        003F    220 ;
                        003F    221
                00000000 003F   222 STKPTR: .LONG   0                       ; Saved stack pointer
                      00 0043   223 INIT:   .BYTE   0                       ; TU58 initialized flag
                      00 0044   224 MRSP:   .BYTE   0                       ; MRSP protocol flag
58 45 2E 52 45 56 49 52 44 44 44 00' 0045   225 DDNAME: .ASCIC  /DDDRIVER.EXE/          ; Driver file name
                      45 0051
                      0C 0045
                         0052   226
                         0052   227 CONSDD_DRIVER:
                0D06 8F    BB 0052   228         PUSHR   #^M<R1,R2,R8,R10,R11>   ; Save input registers
                  0A 58    E9 0056   229         BLBC    R8,5$                   ; Branch if even byte count
          50    030C 8F    3C 0059   230         MOVZWL  #SS$_BUFBYTALI,R0       ; Error - odd byte count
                0D06 8F    BA 005E   231         POPR    #^M<R1,R2,R8,R10,R11>   ; Restore registers
                      05   0062   232         RSB
            D8 AF    5E    D0 0063   233 5$:     MOVL    SP,STKPTR               ; Save stack pointer
                         0067   234
                         0067   235 ;
                         0067   236 ; There are 4 possibilities concerning mapping:  the I/O can be
                         0067   237 ; done virtual or physical (MODE(AP)) and we can be executing virtual
                         0067   238 ; or physical (contents of processor register PR$_MAPEN).  If both
                         0067   239 ; modes match, then we can just copy data to/from the user buffer.
                         0067   240 ; If the I/O is to be done virtual and we are executing physical
                         0067   241 ; then the buffer address has to be translated using the page table
                         0067   242 ; pointed to by R1.  If the I/O is to be done physical and we are
                         0067   243 ; executing virtual then we have to double map the buffer using a spare
```

```
                        0067   244 ; PTE.  At this point, we just compute a mapping switch in R11 as follows:
                        0067   245 ;
                        0067   246 ;       0       Both modes match, just copy the data
                        0067   247 ;       1       Do virtual -> physical translation using page table
                        0067   248 ;       -1      Do physical -> virtual mapping using a spare PTE
                        0067   249 ;
      5B    38    DB    0067   250         MFPR    #PR$_MAPEN,R11          ; Get mapping enabled switch
      5B    5B    CE    006A   251         MNEGL   R11,R11                 ; Negate it
   5B   14   AC    C0    006D   252         ADDL    MODE(AP),R11            ; Add I/O mode switch
                        0071   253
      56    51    7D    0071   254         MOVQ    R1,R6                   ; R6 = Addr. pf page tbl, R7 = Base VPN
         0180    30    0074   255         BSBW    DO_MAPPING              ; Initialize mapping if required
        C9 AF    95    0077   256         TSTB    INIT                    ; Is it necessary to initialize TU58?
           21    12    007A   257         BNEQ    10$                     ; No
                        007C   258
                        007C   259         ;
                        007C   260         ; Initialize TU58.  This code is executed the first time the
                        007C   261         ; driver is called and the next time the driver is called
                        007C   262         ; after exiting with an error.
                        007C   263         ;
                        007C   264
      1E    01    DA    007C   265         MTPR    #CSTS_M_BREAK,#PR$_CSTS ; Set BREAK bit in trans. csr
            52    D4    007F   266         CLRL    R2                      ; R2 contains null characters
         0112    30    0081   267         BSBW    XMIT_TWO_CHARS          ; Send two nulls
      1E    00    DA    0084   268         MTPR    #0,#PR$_CSTS            ; Clear BREAK bit
      50    1D    DB    0087   269         MFPR    #PR$_CSRD,R0            ; Clear receive buffer
52    0404 8F     3C    008A   270         MOVZWL  #DD_INIT@8+DD_INIT,R2   ; R2 contains two INIT characters
         0104    30    008F   271         BSBW    XMIT_TWO_CHARS          ; Send two INIT characters
         012B    30    0092   272         BSBW    RECV_ONE_CHAR           ; Receive a character
      10    52    91    0095   273         CMPB    R2,#DD_CONTINUE         ; Is it continue?
            4C    12    0098   274         BNEQ    32$                     ; No, error
        A6 AF     96    009A   275         INCB    INIT                    ; So that we don't execute this code again
                        009D   276
                        009D   277 ;
                        009D   278 ; Perform the I/O transfer.  Register usage is:
                        009D   279 ;
                        009D   280 ;       R0              Scratch
                        009D   281 ;       R1              Loop counter
                        009D   282 ;       R2              Character to/from TU58
                        009D   283 ;       R3              Character received from TU58
                        009D   284 ;       R4              Checksum
                        009D   285 ;       R5              Logical block number
                        009D   286 ;       R6              Address of page table
                        009D   287 ;       R7              Virtual page number of buffer
                        009D   288 ;       R8              Size of remaining buffer (in bytes)
                        009D   289 ;       R9              Address of RPB
                        009D   290 ;       R10             Address of current spot in buffer
                        009D   291 ;       R11             Mapping switch
                        009D   292 ;
                        009D   293 ;
                        009D   294         ASSUME  DD_WRITE  EQ  DD_READ+1
                        009D   295         ASSUME  DD_DATA  EQ  1
                        009D   296
            54    D4    009D   297 10$:    CLRL    R4                      ; Clear checksum
52    0A02 8F     3C    009F   298         MOVZWL  #10@8+DD_CNTRL,R2       ; Command packet and byte count
         00E9    30    00A4   299         BSBW    XMIT_TWO_UPDSUM         ; Send them
            52    9A    00A7   300         MOVZBL  #DD_READ,R2             ; Assume read command
```

```
                21    10 AC    D1  00AA    301          CMPL    FUNC(AP),#IO$_READLBLK  ; Is it a read?
                        02    13  00AE    302          BEQL    20$                     ; Yes
                        52    D6  00B0    303          INCL    R2                      ; No, convert to write command
                      00DB    30  00B2    304 20$:     BSBW    XMIT_TWO_UPDSUM         ; Send command and modifier (0)
                                  00B5    305 ;
                                  00B5    306 ;       The unit number of the console boot device is always zero.
                                  00B5    307 ;       This is a result of the 11/750 console implementation, where
                                  00B5    308 ;       logical device CSA1: is actually physical device DDA0:.
                                  00B5    309 ;
                        52    D4  00B5    310          CLRL    R2                      ; Set Unit # and Switch fields
                     8A AF    95  00B7    311          TSTB    MRSP                    ; Does this TU58 speak MRSP?
                        04    13  00BA    312          BEQL    25$                     ; Branch if not
                00 52   0B    E2  00BC    313          BBSS    #<MRSP_SWITCH+8>,R2,25$ ; Set MRSP switch
                      00CD    30  00C0    314 25$:     BSBW    XMIT_TWO_UPDSUM         ; Send unit # and switches (0)
                        52    D4  00C3    315          CLRL    R2                      ; Set sequence number (unused field)
                      00C8    30  00C5    316          BSBW    XMIT_TWO_UPDSUM         ; Send sequence number (0)
                   52   58    3C  00C8    317          MOVZWL  R8,R2                   ; Byte count
                      00C2    30  00CB    318          BSBW    XMIT_TWO_UPDSUM         ; Send it
                   52   55    3C  00CE    319          MOVZWL  R5,R2                   ; Block number
                      00BC    30  00D1    320          BSBW    XMIT_TWO_UPDSUM         ; Send it
                   52   54    3C  00D4    321          MOVZWL  R4,R2                   ; Checksum
                      00BC    30  00D7    322          BSBW    XMIT_TWO_CHARS          ; Send it
                21    10 AC    D1  00DA    323          CMPL    FUNC(AP),#IO$_READLBLK  ; Is it a read command?
                        40    13  00DE    324          BEQL    50$
                                  00E0    325
                                  00E0    326 ;
                                  00E0    327 ; Do a write to the TU58
                                  00E0    328 ;
                      00DD    30  00E0    329 30$:     BSBW    RECV_ONE_CHAR           ; Receive one character
                   10   52    91  00E3    330          CMPB    R2,#DD_CONTINUE         ; Is it CONTINUE?
                        40    12  00E6    331 32$:     BNEQ    52$                     ; No, error
                        54    D4  00E8    332          CLRL    R4                      ; Clear checksum
                 51    80 8F    9A  00EA    333          MOVZBL  #128,R1                 ; Number of bytes to send
                   58   51    D1  00EE    334          CMPL    R1,R8                   ; Is it less than remaining byte count?
                        03    1F  00F1    335          BLSSU   35$                     ; Yes
                   51   58    D0  00F3    336          MOVL    R8,R1                   ; No, use remaining byte count instead
                52 51   08    78  00F6    337 35$:     ASHL    #8,R1,R2                ; Put byte count in second byte
                        52    D6  00FA    338          INCL    R2                      ; Flag byte = DD_DATA
                      0091    30  00FC    339          BSBW    XMIT_TWO_UPDSUM         ; Send flag byte and byte count
                   51   02    C6  00FF    340          DIVL    #2,R1                   ; Convert byte count to word count
                      00D3    30  0102    341 40$:     BSBW    GETBYTE                 ; Get a byte from memory in R3
                   52   53    D0  0105    342          MOVL    R3,R2                   ; Save byte in R2
                      00CD    30  0108    343          BSBW    GETBYTE                 ; Get another byte from memory in R3
             52 08   08   53    F0  010B    344          INSV    R3,#8,#8,R2             ; Put it in second byte of R2
                        7E    10  0110    345          BSBB    XMIT_TWO_UPDSUM         ; Send two bytes
                     ED 51    F5  0112    346          SOBGTR  R1,40$                  ; Repeat until byte count is 0
                   52   54    3C  0115    347          MOVZWL  R4,R2                   ; Get checksum
                        7C    10  0118    348          BSBB    XMIT_TWO_CHARS          ; Send it
                        58    D5  011A    349          TSTL    R8                      ; Any more data to send?
                        C2    12  011C    350          BNEQ    30$                     ; Yes, send another packet
                        2C    11  011E    351          BRB     END_OF_DATA             ; No, get End packet
                                  0120    352
                                  0120    353 ;
                                  0120    354 ; Do a read from TU58
                                  0120    355 ;
                        54    D4  0120    356 50$:     CLRL    R4                      ; Clear checksum
                      0088    30  0122    357          BSBW    RECV_TWO_UPDSUM         ; Get flag byte in R3, byte count in R2
```

```
                01   53   91   0125   358                CMPB    R3,#DD_DATA             ; Is this a data packet?
                     53   12   0128   359 52$:           BNEQ    DD_ERROR                ; No, error
           51    52   02   C7   012A   360                DIVL3   #2,R2,R1                ; Convert byte count to word count in R1
                      7D   10   012E   361 55$:           BSBB    RECV_TWO_UPDSUM         ; Receive next two data bytes in R3, R2
                   00B0   30   0130   362                BSBW    PUTBYTE                 ; Store first byte in memory
                53    52   9A   0133   363                MOVZBL  R2,R3                   ; Move second byte to R3
                   00AA   30   0136   364                BSBW    PUTBYTE                 ; Store second byte in memory
                   F2 51   F5   0139   365                SOBGTR  R1,55$                  ; Repeat until byte count is 0
                      7D   10   013C   366                BSBB    RECV_TWO_CHARS          ; Get checksum in R3, R2
      53   08    08   52   F0   013E   367                INSV    R2,#8,#8,R3             ; Assemble checksum into one word
                54    53   B1   0143   368                CMPW    R3,R4                   ; Is checksum correct?
                      35   12   0146   369                BNEQ    DD_ERROR                ; No, error
                      58   D5   0148   370                TSTL    R8                      ; Anymore data to receive?
                      D4   12   014A   371                BNEQ    50$                     ; Yes, receive next data packet
                           014C   372
                           014C   373 ;
                           014C   374 ; We've sent or received all the data.  Make sure we receive an end
                           014C   375 ; packet with a success code.
                           014C   376 ;
                           014C   377 END_OF_DATA:
                      54   D4   014C   378                CLRL    R4                      ; Clear checksum
                      5D   10   014E   379                BSBB    RECV_TWO_UPDSUM         ; Get flag byte and byte count
                02    53   91   0150   380                CMPB    R3,#DD_CNTRL            ; Is it a command packet?
                      28   12   0153   381                BNEQ    DD_ERROR                ; No, error
                      56   10   0155   382                BSBB    RECV_TWO_UPDSUM         ; Get opcode and success/failure byte
             40 8F    53   91   0157   383                CMPB    R3,#DD_ENDPKT           ; Is it an end packet?
                      20   12   015B   384                BNEQ    DD_ERROR                ; No, error
                      52   D5   015D   385                TSTL    R2                      ; Is it success?
                      1C   19   015F   386                BLSS    DD_ERROR                ; No, error
                51    04   D0   0161   387                MOVL    #4,R1                   ; Read remainder of packet
                      47   10   0164   388 10$:           BSBB    RECV_TWO_UPDSUM         ;
                   FB 51   F5   0166   389                SOBGTR  R1,10$                  ;
                      50   10   0169   390                BSBB    RECV_TWO_CHARS          ; Read checksum in R3, R2
      53   08    08   52   F0   016B   391                INSV    R2,#8,#8,R3             ; Form checksum in R3
                54    53   B1   0170   392                CMPW    R3,R4                   ; Is checksum correct?
                      08   12   0173   393                BNEQ    DD_ERROR                ; No, error
                50    01   3C   0175   394                MOVZWL  #SS$_NORMAL,R0          ; Return success
                0D06  8F   BA   0178   395                POPR    #^M<R1,R2,R8,R10,R11>   ; Restore registers
                           05   017C   396                RSB
                           017D   397
                           017D   398 DD_ERROR:
                FEC2  CF   94   017D   399                CLRB    INIT                    ; Initialize TU58 on next entry
             5E FEBA  CF   D0   0181   400                MOVL    STKPTR,SP               ; Restore stack pointer
                0D06  8F   BA   0186   401                POPR    #^M<R1,R2,R8,R10,R11>   ; Restore registers
             50 0054  8F   3C   018A   402                MOVZWL  #SS$_CTRLERR,R0         ; Set failure status
                           05   018F   403                RSB                             ; Return and retry
                           0190   404
                           0190   405
                           0190   406 ;++
                           0190   407 ; XMIT_TWO_UPDSUM - Transmit two characters and update checksum
                           0190   408 ; XMIT_TWO_CHARS - Transmit two characters
                           0190   409 ; XMIT_ONE_CHAR - Transmit one character
                           0190   410 ;
                           0190   411 ; Inputs:
                           0190   412 ;       R2      Contains one or two characters in low bytes
                           0190   413 ;       R4      Checksum so far
                           0190   414 ;
```

```
                     0190   415 ; Outputs:
                     0190   416 ;       R4      Updated checksum
                     0190   417 ;--
                     0190   418
                     0190   419 XMIT_TWO_UPDSUM:
          54   52    A0   0190   420         ADDW    R2,R4                   ; Add two characters to checksum
          54   00    D8   0193   421         ADWC    #0,R4                   ; Add carry
                     0196   422
                     0196   423 XMIT_TWO_CHARS:
               52    DD   0196   424         PUSHL   R2                      ; Save input on stack.
          52   52    9A   0198   425         MOVZBL  R2,R2                   ; Zero high bytes of data.
               05    10   019B   426         BSBB    XMIT_ONE_CHAR           ; Transmit one character
     52   8E   F8  8F    78   019D   427         ASHL    #-8,(SP)+,R2            ; Retrieve second character
                     01A2   428                                             ; and fall through to transmit it
                     01A2   429
                     01A2   430 XMIT_ONE_CHAR:
          50   1E    DB   01A2   431         MFPR    #PR$_CSTS,R0            ; Get transmit status
     F9   50   07    E1   01A5   432         BBC     #CSTS_V_READY,R0,XMIT_ONE_CHAR  ; Loop until ready
          1F   52    DA   01A9   433         MTPR    R2,#PR$_CSTD            ; Transmit character
               05    05   01AC   434         RSB
                     01AD   435
                     01AD   436
                     01AD   437 ;++
                     01AD   438 ; RECV_TWO_UPDSUM - Receive two characters and update checksum
                     01AD   439 ; RECV_TWO_CHARS - Receive two characters
                     01AD   440 ; RECV_ONE_CHAR - Receive one character
                     01AD   441 ;
                     01AD   442 ; Inputs:
                     01AD   443 ;       R4      Checksum so far
                     01AD   444 ;
                     01AD   445 ; Outputs:
                     01AD   446 ;       R2      Most recently received character
                     01AD   447 ;       R3      Previous character in low byte
                     01AD   448 ;--
                     01AD   449
                     01AD   450 RECV_TWO_UPDSUM:
               0C    10   01AD   451         BSBB    RECV_TWO_CHARS          ; Receive two characters in R3,  R2
53   08   08   52    F0   01AF   452         INSV    R2,#8,#8,R3             ; Put second character into R3
          54   53    A0   01B4   453         ADDW    R3,R4                   ; Add two characters to checksum
          54   00    D8   01B7   454         ADWC    #0,R4                   ; Add carry to checksum
               05    05   01BA   455         RSB
                     01BB   456
                     01BB   457 RECV_TWO_CHARS:
               03    10   01BB   458         BSBB    RECV_ONE_CHAR           ; Get one character in R2
          53   52    9A   01BD   459         MOVZBL  R2,R3                   ; Save first character in R3 and
                     01C0   460                                             ; fall through to get second character
                     01C0   461
                     01C0   462 RECV_ONE_CHAR:
          50   1C    DB   01C0   463         MFPR    #PR$_CSRS,R0            ; Get receive status
     F9   50   07    E1   01C3   464         BBC     #CSRS_V_DONE,R0,RECV_ONE_CHAR   ; Loop until done
          52   1D    DB   01C7   465         MFPR    #PR$_CSRD,R2                    ; Get next character
     FE76 CF   95    13   01CA   466         TSTB    MRSP                            ; Does this TU58 speak MRSP?
               03    13   01CE   467         BEQL    10$                             ; Branch if not
          1F   10    DA   01D0   468         MTPR    #DD_CONTINUE,#PR$_CSTD          ; Send CONTINUE character
     A6   52   0F    E0   01D3   469 10$:     BBS     #CSRD_V_ERROR,R2,DD_ERROR       ; Branch if data overrun
               05    05   01D7   470         RSB
                     01D8   471
```

```
                        01D8   472
                        01D8   473 ;++
                        01D8   474 ; GETBYTE - Subroutine to get a byte from memory
                        01D8   475 ; PUTBYTE - Subroutine to store a byte in memory
                        01D8   476 ;
                        01D8   477 ;       These two subroutines do two things special:
                        01D8   478 ;
                        01D8   479 ;               1)  Since the floppy always reads or writes 128 bytes
                        01D8   480 ;                   these routines simply return if the byte count is zero.
                        01D8   481 ;               2)  These routines take care of page boundaries if
                        01D8   482 ;                   mapping is required.
                        01D8   483 ;
                        01D8   484 ; Inputs:
                        01D8   485 ;       R3      Byte to store (PUTBYTE)
                        01D8   486 ;       R6      Address of page table
                        01D8   487 ;       R7      Virtual page number of buffer
                        01D8   488 ;       R8      Size of remaining buffer (in bytes)
                        01D8   489 ;       R10     Address of current spot in buffer
                        01D8   490 ;       R11     Mapping switch:
                        01D8   491 ;                       -1   Do physical -> virtual map
                        01D8   492 ;                       0    No mapping required
                        01D8   493 ;                       1    Do virtual -> physical translation
                        01D8   494 ;
                        01D8   495 ; Outputs:
                        01D8   496 ;       R3      Byte fetched from memory (GETBYTE)
                        01D8   497 ;--
                        01D8   498
                        01D8   499          .ENABL  LSB
                        01D8   500
                        01D8   501 GETBYTE:
          53    D4      01D8   502          CLRL    R3                      ; Return 0 if byte count = 0
          58    D5      01DA   503          TSTL    R8                      ; Is byte count 0?
          2F    13      01DC   504          BEQL    90$                     ; Yes
    53    8A    9A      01DE   505          MOVZBL  (R10)+,R3               ; Get byte
          07    11      01E1   506          BRB     10$                     ; Branch to common code
                        01E3   507
                        01E3   508
                        01E3   509 PUTBYTE:
          58    D5      01E3   510          TSTL    R8                      ; Is byte count 0?
          26    13      01E5   511          BEQL    90$                     ; Yes
    8A    53    90      01E7   512          MOVB    R3,(R10)+               ; Store byte
                        01EA   513
                        01EA   514
          58    D7      01EA   515 10$:     DECL    R8                      ; Decr. byte count
          1F    13      01EC   516          BEQL    90$                     ; Reached zero
5A  01FF  8F    B3      01EE   517          BITW    #VA$M_BYTE,R10          ; Did address overflow onto new page?
          18    12      01F3   518          BNEQ    90$                     ; No
          57    D6      01F5   519          INCL    R7                      ; Yes, increment page number
                        01F7   520
                        01F7   521 ;
                        01F7   522 ; Fall through to ...
                        01F7   523 ;
                        01F7   524
                        01F7   525
                        01F7   526 ;++
                        01F7   527 ; DO_MAPPING - Subroutine to perform necessary mapping
                        01F7   528 ;
```

```
                        01F7    529 ; Inputs:
                        01F7    530 ;       R6      Address of page table
                        01F7    531 ;       R7      Page number of buffer
                        01F7    532 ;       R10     Address to map
                        01F7    533 ;       R11     Mapping switch:
                        01F7    534 ;                       -1      Do physical -> virtual map
                        01F7    535 ;                       0       No mapping required
                        01F7    536 ;                       1       Do virtual -> physical translation
                        01F7    537 ;
                        01F7    538 ; Outputs:
                        01F7    539 ;       R10     Address to use
                        01F7    540 ;--
                        01F7    541
                        01F7    542 DO_MAPPING:
                5B  D5  01F7    543         TSTL    R11                             ; Any mapping required?
                12  13  01F9    544         BEQL    90$                             ; No
                11  19  01FB    545         BLSS    100$                            ; Yes, map physical to virtual
 5A     FFFFFE00 8F  CA  01FD    546         BICL    #^C<VA$M_BYTE>,R10              ; Yes, translate virtual to physical
                        0204    547                                                 ; Clear everything but byte offset
          50    6647  D0  0204    548         MOVL    (R6)[R7],R0                     ; Get PFN in R0
 5A   15    09    50  F0  0208    549         INSV    R0,#VA$V_VPN,#PTE$S_PFN,R10     ; Insert PFN after byte offset
                05      020D    550 90$:    RSB
                        020E    551
                        020E    552
                        020E    553 ;
                        020E    554 ; Map physical to virtual
                        020E    555 ;
                00      020E    556 100$:   HALT                                    ; Not implemented yet
                        020F    557
                        020F    558         .DSABL  LSB
                        020F    559
            0000020F    020F    560 CONSDD_DRVSIZ=.-CONSDD_START
                        020F    561
                        020F    562         .END
```

```
$$BASE                          = 00000001
$$DISPL                         = 00000005
$$GENSW                         = 00000001
$$HIGH                          = 00000004
$$LIMIT                         = 00000003
$$LOW                           = 00000001
$$MNSW                          = 00000001
$$MXSW                          = 00000001
$TABLE                          = 00000000 R      02
BTD$K_CONSOLE                   = 00000040
CONSDD_DRIVER                     00000052 R      03
CONSDD_DRVSIZ                   = 0000020F
CONSDD_START                      00000000 R      03
CPU_730                           00000036 R      03
CPU_750                           0000003C R      03
CPU_780                           00000035 R      03
CPU_790                           00000035 R      03
CSRD_V_ERROR                    = 0000000F
CSRS_V_DONE                     = 00000007
CSTS_M_BREAK                    = 00000001
CSTS_V_READY                    = 00000007
DDNAME                            00000045 R      03
DD_CNTRL                        = 00000002
DD_CONTINUE                     = 00000010
DD_DATA                         = 00000001
DD_ENDPKT                       = 00000040
DD_ERROR                          0000017D R      03
DD_INIT                         = 00000004
DD_READ                         = 00000002
DD_SELECT                         00000000 R      03
DD_WRITE                        = 00000003
DO_MAPPING                        000001F7 R      03
END_OF_DATA                       0000014C R      03
ERROUT                            ******** X      03
EXE$GB_CPUTYPE                    ******** X      03
FUNC                            = 00000010
GETBYTE                           000001D8 R      03
INIT                              00000043 R      03
IO$_READLBLK                    = 00000021
MODE                            = 00000014
MRSP                              00000044 R      03
MRSP_SWITCH                     = 00000003
PR$_CSRD                        = 0000001D
PR$_CSRS                        = 0000001C
PR$_CSTD                        = 0000001F
PR$_CSTS                        = 0000001E
PR$_MAPEN                       = 00000038
PR$_SID_TYP730                  = 00000003
PR$_SID_TYP750                  = 00000002
PR$_SID_TYP780                  = 00000001
PR$_SID_TYP790                  = 00000004
PTE$S_PFN                       = 00000015
PUTBYTE                           000001E3 R      03
RECV_ONE_CHAR                     000001C0 R      03
RECV_TWO_CHARS                    000001BB R      03
RECV_TWO_UPDSUM                   000001AD R      03
SIZ...                          = 00000008
SS$_BUFBYTALI                   = 0000030C
SS$_CTRLERR                     = 00000054
SS$_NORMAL                      = 00000001
STKPTR                            0000003F R      03
VA$M_BYTE                       = 000001FF
VA$V_VPN                        = 00000009
XMIT_ONE_CHAR                     000001A2 R      03
XMIT_TWO_CHARS                    00000196 R      03
XMIT_TWO_UPDSUM                   00000190 R      03
```

Psect synopsis

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
ABS	00000000 (0.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
BOOTDRIVR_4	00000028 (40.)	02 (2.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC BYTE
BOOTDRIVR_2	0000020F (527.)	03 (3.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC BYTE

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	29	00:00:00.06	00:00:00.45
Command processing	108	00:00:00.63	00:00:02.65
Pass 1	337	00:00:10.88	00:00:26.10
Symbol table sort	0	00:00:01.72	00:00:02.37
Pass 2	108	00:00:02.30	00:00:05.25
Symbol table output	10	00:00:00.09	00:00:00.09
Psect synopsis output	1	00:00:00.03	00:00:00.03
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	595	00:00:15.71	00:00:36.94

The working set limit was 1500 pages.
62296 bytes (122 pages) of virtual memory were used to buffer the intermediate code.
There were 60 pages of symbol table space allocated to hold 1088 non-local and 22 local symbols.
562 source lines were read in Pass 1, producing 16 object records in Pass 2.
28 pages of virtual memory were used to define 24 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA28:[BOOTS.OBJ]BOOTS.MLB;1	1
_255DUA28:[SYS.OBJ]LIB.MLB;1	6
_255DUA28:[SYSLIB]STARLET.MLB;2	8
TOTALS (all libraries)	15

1256 GETS were required to define 15 macros.

There were no errors, warnings or information messages.

```
MACRO/LIS=LIS$:DDBTDRIVR/OBJ=OBJ$:DDBTDRIVR MSRC$:DDBTDRIVR/UPDATE=(ENH$:DDBTDRIVR)+EXECML$/LIB+LIB$:BOOTS.MLB/LIB
```

CVBTDRIVR
LIS

DDBTDRIVR
LIS

DMBTDRIVR
LIS

DQBTDRIVR
LIS

CONIO
LIS

LOADERSUB
LIS

CONFIGUTL
LIS

DLBTDRIVR
LIS

DXBTDRIVR
LIS

INITPGFIL
LIS

LOCKDATA
LIS

LOADDRIV
LIS

LOADER
LIS