

|              |     |           |     |           |     |              |            |            |
|--------------|-----|-----------|-----|-----------|-----|--------------|------------|------------|
| BBBBBBBBBBBB |     | 000000000 |     | 000000000 |     | TTTTTTTTTTTT |            | SSSSSSSSSS |
| BBBBBBBBBBBB |     | 000000000 |     | 000000000 |     | TTTTTTTTTTTT |            | SSSSSSSSSS |
| BBBBBBBBBBBB |     | 000000000 |     | 000000000 |     | TTTTTTTTTTTT |            | SSSSSSSSSS |
| BBB          | BBB | 000       | 000 | 000       | 000 | TTT          | SSS        |            |
| BBB          | BBB | 000       | 000 | 000       | 000 | TTT          | SSS        |            |
| BBB          | BBB | 000       | 000 | 000       | 000 | TTT          | SSS        |            |
| BBB          | BBB | 000       | 000 | 000       | 000 | TTT          | SSS        |            |
| BBB          | BBB | 000       | 000 | 000       | 000 | TTT          | SSS        |            |
| BBB          | BBB | 000       | 000 | 000       | 000 | TTT          | SSS        |            |
| BBB          | BBB | 000       | 000 | 000       | 000 | TTT          | SSS        |            |
| BBBBBBBBBBBB |     | 000       | 000 | 000       | 000 | TTT          | SSS        |            |
| BBBBBBBBBBBB |     | 000       | 000 | 000       | 000 | TTT          | SSS        |            |
| BBBBBBBBBBBB |     | 000       | 000 | 000       | 000 | TTT          | SSS        |            |
| BBB          | BBB | 000       | 000 | 000       | 000 | TTT          | SSS        |            |
| BBB          | BBB | 000       | 000 | 000       | 000 | TTT          | SSS        |            |
| BBB          | BBB | 000       | 000 | 000       | 000 | TTT          | SSS        |            |
| BBB          | BBB | 000       | 000 | 000       | 000 | TTT          | SSS        |            |
| BBB          | BBB | 000       | 000 | 000       | 000 | TTT          | SSS        |            |
| BBB          | BBB | 000       | 000 | 000       | 000 | TTT          | SSS        |            |
| BBBBBBBBBBBB |     | 000000000 |     | 000000000 |     | TTT          | SSSSSSSSSS |            |
| BBBBBBBBBBBB |     | 000000000 |     | 000000000 |     | TTT          | SSSSSSSSSS |            |
| BBBBBBBBBBBB |     | 000000000 |     | 000000000 |     | TTT          | SSSSSSSSSS |            |

```

LL          IIIII
LL          IIIII
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LL          II
LLLLLLLLLLL IIIII
LLLLLLLLLLL IIIII
SSSSSSSSS
SSSSSSSSS
SS
SS
SS
SS
SSSSSS
SSSSSS
SS
SS
SS
SS
SSSSSSSSS
SSSSSSSSS

```



(2) 53  
(3) 119

DECLARATIONS  
Console Floppy Bootstrap Driver Code

```
0000 1 .TITLE DXBTDRIVR - CONSOLE RX01 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 RX01 console floppy.
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 02-04 TCM0001 Trudy C. Matthews 12-Jun-1981
0000 42 Correct bug in DO_MAPPING routine.
0000 43
0000 44
0000 45 02-03 SRB0001 Steve Beckhardt 10-Jul-1980
0000 46 Fix bug in change CAS0001
0000 47
0000 48 02-02 CAS0001 C.A. Samuelson 30-Apr-1980
0000 49 Change interface to BOOTDRIVR for UBA purge datapath
0000 50
0000 51 --
```



```
0000 53      .SBTTL  DECLARATIONS
0000 54      :
0000 55      : INCLUDE FILES:
0000 56      :
0000 57      :
0000 58      $BTDDDEF      : Boot device types
0000 59      $IODEF       : I/O function codes
0000 60      $PRDEF       : Processor registers
0000 61      $PTEDEF      : PTE definitions
0000 62      $RPBDEF      : RPB offsets
0000 63      $SSDEF       : Status codes
0000 64      $VADEF       : Virtual address fields
0000 65      :
0000 66      :
0000 67      : MACROS:
0000 68      :
0000 69      :
0000 70      :
0000 71      : EQUATED SYMBOLS:
0000 72      :
0000 73      :
0000 74      :
0000 75      : 11/780 CONSOLE FLOPPY DEFINITIONS
0000 76      :
0000 77      :
0000 78      $DEFINI DX      : START OF REGISTER DEFINITIONS
0000 79      :
0000 80      _VIELD  RXCS,6,<-      : START OF PROC. REG. RXCS DEFS.
0000 81      <IE,,M>,-      : INTERRUPT ENABLE
0000 82      <DONE,,M>,-      : DONE
0000 83      >
0000 84      :
0000 85      _VIELD  RXDB,0,<-      : START OF PROC. REG. RXDB DEFS
0000 86      <DATA,8>,-      : DATA
0000 87      <SEL,4>,-      : SELECT
0000 88      >
0000 89      :
0000 90      _VIELD  TXCS,6,<-      : START OF PROC. REG. TXCS DEFS
0000 91      <IE,,M>,-      : INTERRUPT ENABLE
0000 92      <READY,,M>,-      : READY
0000 93      >
0000 94      :
0000 95      _VIELD  TXDB,0,<-      : START OF PROC. REG. TXDB DEFS
0000 96      <DATA,8>,-      : DATA
0000 97      <SEL,4>,-      : SELECT
0000 98      >
0000 99      :
00000900 0000 100 DX_READSECTOR = ^X900      : FUNCTION READ SECTOR
00000901 0000 101 DX_WRITESECTOR = ^X901      : FUNCTION WRITE SECTOR
00000200 0000 102 DX_FUNCCOMPLETE = ^X200      : STATUS FUNCTION COMPLETE
0000 103      :
0000 104      $DEFEND DX      : END OF 11/780 FLOPPY DEFINITIONS
0000 105      :
0000 106      : OWN STORAGE:
0000 107      :
0000 108      :
0000 109      :
```

DXBTDRIVR  
V04-000

- CONSOLE RX01 BOOT DRIVER  
DECLARATIONS

M 10

15-SEP-1984 23:51:54 VAX/VMS Macro V04-00  
4-SEP-1984 23:04:18 [BOOTS.SRC]DXBTDRIVR.MAR;1

Page 3  
(2)

0000 110 ; Boot driver table entry  
0000 111 ;  
0000 112 ;  
0000 113 \$BOOT\_DRIVER  
0000 114  
0000 115  
0000 116  
0000 117

DEVTYPE = BTD\$K\_CONSOLE,- ; Device type (console)  
CPUTYPE = PR\$\_SID\_TYP780,- ; Cpu type (11/780)  
SIZE = CONSDX\_DRV\$IZ,- ; Driver size  
ADDR = CONSDX\_DRIVER,- ; Driver address  
DRIVNAME = DXNAME ; Driver file name



```
0000 119 .SBTTL Console Floppy Bootstrap Driver Code
0000 120
0000 121 :++
0000 122 :
0000 123 : Inputs:
0000 124 :
0000 125 : R1 Address of page table for virtual -> physical mapping
0000 126 : R2 Base VPN of transfer (Bits 29:9 of R10)
0000 127 : R5 LBN for current piece of transfer
0000 128 : R8 Size of transfer in bytes
0000 129 : R9 Address of the RPB
0000 130 : R10 Starting address of transfer
0000 131 :
0000 132 : FUNC(AP) I/O operation (IO$_READLBLK or IO$_WRITELBLK only)
0000 133 : MODE(AP) Address interpretation mode:
0000 134 : 0 -> Physical, 1 -> Virtual
0000 135 :
0000 136 : Outputs:
0000 137 :
0000 138 : R0 Status code:
0000 139 :
0000 140 : $$$_NORMAL Successful transfer
0000 141 : $$$_CTRLERR Fatal controller error
0000 142 : --
0000 143 :
00000010 0000 144 FUNC = 16
00000014 0000 145 MODE = 20
0000 146
0000 147 CONSDX_DRIVER:
0000 148 PUSH R1,R2,R8,R10,R11 ; Save input registers
0004 149
0004 150 :
0004 151 : Perform initialization: Set up a mapping switch in R11, read RXDB
0004 152 : if done is set in RXCS, and set up registers.
0004 153 : There are 4 possibilities concerning mapping: the I/O can be
0004 154 : done virtual or physical (MODE(AP)) and we can be executing virtual
0004 155 : or physical (contents of processor register PR$_MAPEN). If both
0004 156 : modes match, then we can just copy data to/from the user buffer.
0004 157 : If the I/O is to be done virtual and we are executing physical
0004 158 : then the buffer address has to be translated using the page table
0004 159 : pointed to by R1. If the I/O is to be done physical and we are
0004 160 : executing virtual then we have to double map the buffer using a spare
0004 161 : PTE. At this point, we just compute a mapping switch in R11 as follows:
0004 162 :
0004 163 : 0 Both modes match, just copy the data
0004 164 : 1 Do virtual -> physical translation using page table
0004 165 : -1 Do physical -> virtual mapping using a spare PTE
0004 166 :
0004 167 5$: MFPR #PR$_MAPEN,R11 ; Get mapping enabled switch
0007 168 MNEGL R11,R11 ; Negate it
000A 169 ADDL MODE(AP),R11 ; Add I/O mode switch
000E 170
000E 171 MFPR #PR$_RXCS,R0 ; Get RXCS
0011 172 BBC #RXCS_V_DONE,R0,10$ ; Branch if done is clear
0015 173 MFPR #PR$_RXDB,R0 ; Read RXDB and ignore
0018 174 10$: MOVQ R1,R6 ; R6 = Addr. of page tbl, R7 = Base VPN
001B 175
```

5B 38 DB 0004 167 5\$: MFPR #PR\$\_MAPEN,R11 ; Get mapping enabled switch  
5B 5B CE 0007 168 MNEGL R11,R11 ; Negate it  
5B 14 AC CO 000A 169 ADDL MODE(AP),R11 ; Add I/O mode switch  
03 50 20 DB 000E 170  
50 07 E1 0011 172 BBC #RXCS\_V\_DONE,R0,10\$ ; Branch if done is clear  
50 21 DB 0015 173 MFPR #PR\$\_RXDB,R0 ; Read RXDB and ignore  
56 51 7D 0018 174 10\$: MOVQ R1,R6 ; R6 = Addr. of page tbl, R7 = Base VPN  
001B 175



```
001B 176 :  
001B 177 : Convert logical block number in R5 to a logical sector and cylinder  
001B 178 : on floppy. This is saved in R4 and R5 and converted to a physical  
001B 179 : sector and cylinder as needed. Note that the console floppy has  
001B 180 : 26 sectors per track, 1 track per cylinder, and 77 cylinders.  
001B 181 : Also note that each sector is 128 bytes so the LBN must be  
001B 182 : multiplied by 4 to convert to sectors.  
001B 183 :  
50 00 55 04 7A 001B 184 EMUL #4,R5,#0,R0 : Mult. LBN by 4, Quad result to R0, R1  
54 55 50 1A 7B 0020 185 EDIV #26,R0,R5,R4 : Divide by # of sectors, quotient  
0025 186 : (R5) is cyl. Rem. (R4) is sector  
0025 187 :  
0025 188 :  
0025 189 : Set up mapping if required  
0025 190 :  
00C4 30 0025 191 BSBW DO_MAPPING  
0028 192 :  
0028 193 :  
0028 194 : This is the main loop to read or write to the floppy and to get  
0028 195 : or store in memory. Register usage is:  
0028 196 :  
0028 197 : R0 - R3 Scratch  
0028 198 : R4 Logical sector  
0028 199 : R5 Logical cylinder  
0028 200 : R6 Address of page table  
0028 201 : R7 Virtual page number of buffer  
0028 202 : R8 Size of remaining buffer (in bytes)  
0028 203 : R9 Address of RPB  
0028 204 : R10 Address of current spot in buffer  
0028 205 : R11 Mapping switch  
0028 206 :  
0028 207 : First convert logical sector and cylinder to physical sector and  
0028 208 : cylinder. Then send command, sector, and cylinder to floppy. Then  
0028 209 : read or write 128 bytes of data. Repeat until byte count goes to zero.  
0028 210 :  
0028 211 :  
0028 212 : ASSUME DX_WRITESECTOR EQ DX_READSECTOR+1  
0028 213 :  
53 0900 8F 3C 0028 214 30$: BSBB TRKSEC : Convert to physical sector and  
21 10 AC D1 002A 215 : cylinder in R1 and R2  
002A 216 MOVZWL #DX_READSECTOR,R3 : Assume read  
002F 217 CMPL FUNC(AP),#IOS_READLBLK : Is it read?  
0033 218 BEQL 40$ : Yes  
53 02 13 0035 219 INCL R3 : No, convert to write  
53 72 10 0037 220 40$: BSBB DXOUT : Output function  
53 51 D0 0039 221 MOVL R1,R3 :  
53 6D 10 003C 222 BSBB DXOUT : Output sector  
53 52 D0 003E 223 MOVL R2,R3 :  
68 10 0041 224 BSBB DXOUT : Output cylinder  
0043 225 :  
52 0080 8F 3C 0043 226 MOVZWL #128,R2 : Loop counter (128 bytes)  
21 10 AC D1 0048 227 CMPL FUNC(AP),#IOS_READLBLK : Read or write?  
004C 228 BEQL 60$ : Read  
004E 229 :  
004E 230 :  
004E 231 : Do a Write to the floppy  
004E 232 :
```



```

004E 233
7D 10 004E 234 50$: BSBB GETBYTE ; Get a byte from memory in R3
59 10 0050 235 BSBB DXOUT ; Output it
F9 52 F5 0052 236 SOBGTR R2,50$ ; Repeat
0055 237
0200 8F 64 10 0055 238 BSBB DXIN ; Get floppy status in R3
53 B1 0057 239 CMPW R3,#DX_FUNCCOMPLETE ; Error?
1A 13 005C 240 BEQL 80$ ; No
005E 241
005E 242 55$: ; Error from floppy
005E 243
005E 244
50 0D06 8F BA 005E 245 POPR #^M<R1,R2,R8,R10,R11> ; Restore registers
0054 8F 3C 0062 246 MOVZWL #SS$_CTRLERR,R0 ; Set failure status
05 0067 247 RSB ; Return and retry
0068 248
0068 249
0068 250 ; Do a Read from floppy
0068 251
0200 8F 51 10 0068 252 60$: BSBB DXIN ; Get floppy status in R3
53 B1 006A 253 CMPW R3,#DX_FUNCCOMPLETE ; Error?
ED 12 006F 254 BNEQ 55$ ; Yes
0071 255
48 10 0071 256 70$: BSBB DXIN ; Get a byte from floppy
63 10 0073 257 BSBB PUTBYTE ; Store in memory
F9 52 F5 0075 258 SOBGTR R2,70$ ; Repeat
0078 259
0078 260
0078 261 ; Done with this sector. Repeat loop if byte count is non-zero
0078 262
58 D5 0078 263 80$: TSTL R8 ; Test remaining byte count
0D 13 007A 264 BEQL 90$ ; Done
54 D6 007C 265 INCL R4 ; Not done, incr. sector
1A 54 D1 007E 266 CMPL R4,#26 ; Overflow to next cylinder?
A5 1F 0081 267 BLSSU 30$ ; No, do next sector
54 D4 0083 268 CLRL R4 ; Yes, clear sector
55 D6 0085 269 INCL R5 ; Increment cylinder
9F 11 0087 270 BRB 30$ ; Do next sector
0089 271
50 01 3C 0089 272 90$: MOVZWL #SS$_NORMAL,R0 ; Successful completion
0D06 8F BA 008C 273 POPR #^M<R1,R2,R8,R10,R11> ; Restore registers
05 0090 274 RSB
0091 275
0091 276
0091 277
0091 278 ;++
0091 279 ; TRKSEC - Subroutine to convert logical sector/cylinder to physical
0091 280 ; sector/cylinder on floppy by applying sector interleave
0091 281 ; and track-to-track skew (6 sectors).
0091 282
0091 283 ; Inputs:
0091 284 ; R4 Logical sector
0091 285 ; R5 Logical cylinder
0091 286
0091 287 ; Outputs:
0091 288 ; R1 Physical sector
0091 289 ; R2 Physical cylinder
```

```
0091 290 :
0091 291 : Credits:
0091 292 : Thanks to Chuck Monia for this one.
0091 293 :--
0091 294 :
50 51 51 54 D0 0091 295 TRKSEC: MOVL R4,R1 ; Get logical sector
51 51 51 0C 91 0094 296 CMPB #12,R1 ; Set C-bit if 12 < sector <= 26
51 51 51 51 D8 0097 297 ADWC R1,R1 ; Double sector number, add interleave
50 51 55 06 7A 009A 298 EMUL #6,R5,R1,R0 ; Compute skew (6 * cyl + sector)
51 50 50 1A 7B 009F 299 EDIV #26,R0,R0,R1 ; Modulo sectors per track (26)
51 51 51 51 D6 00A4 300 INCL R1 ; Add 1 to sector because floppy
52 55 01 C1 00A6 301 ADDL3 #1,R5,R2 ; sectors start at 1 not 0
00A6 302 ; Add 1 to cylinder because first
00AA 303 ; floppy cylinder is unused
05 00AA 304 RSB
00AB 305
00AB 306
00AB 307
00AB 308 :++
00AB 309 : DXOUT - Subroutine to output a byte to the floppy.
00AB 310 :
00AB 311 : Inputs:
00AB 312 : R3 Byte to output
00AB 313 :
00AB 314 : Outputs:
00AB 315 : None
00AB 316 :--
00AB 317 :
00AB 318 ASSUME TXDB_V_SEL EQ 8
00AB 319 :
50 22 DB 00AB 320 DXOUT: MFPR #PR$ TXCS,R0 ; Get status
F9 50 07 E1 00AE 321 BBC #TXCS_V_READY,R0,DXOUT ; Branch back if not ready
53 0100 8F A8 00B2 322 BISW #^X100,R3 ; Set floppy unit number
23 53 DA 00B7 323 MTPR R3,#PR$ TXDB ; Send data byte
05 00BA 324 RSB
00BB 325
00BB 326
00BB 327 :++
00BB 328 : DXIN - Subroutine to read a byte from the floppy
00BB 329 :
00BB 330 : Inputs:
00BB 331 : None
00BB 332 :
00BB 333 : Outputs:
00BB 334 : R3 Byte read from floppy (in low byte)
00BB 335 : (Second byte contains select field)
00BB 336 :--
00BB 337 :
50 20 DB 00BB 338 DXIN: MFPR #PR$ RXCS,R0 ; Get status
F9 50 07 E1 00BE 339 BBC #RXCS_V_DONE,R0,DXIN ; Branch back if not done
53 53 21 DB 00C2 340 MFPR #PR$ RXDB,R3 ; Get data
00 53 04 08 ED 00C5 341 CMPZV #RXDB_V_SEL,#RXDB_S_SEL,R3,#0 ; Ignore if from
EF 13 00CA 342 BEQL DXIN ; console terminal
05 00CC 343 RSB
00CD 344
00CD 345
00CD 346
```



```
00CD 347 :++
00CD 348 : GETBYTE - Subroutine to get a byte from memory
00CD 349 : PUTBYTE - Subroutine to store a byte in memory
00CD 350 :
00CD 351 :     These two subroutines do two things special:
00CD 352 :
00CD 353 :         1) Since the floppy always reads or writes 128 bytes
00CD 354 :            these routines simply return if the byte count is zero.
00CD 355 :         2) These routines take care of page boundaries if
00CD 356 :            mapping is required.
00CD 357 :
00CD 358 : Inputs:
00CD 359 :     R3      Byte to store (PUTBYTE)
00CD 360 :     R6      Address of page table
00CD 361 :     R7      Virtual page number of buffer
00CD 362 :     R8      Size of remaining buffer (in bytes)
00CD 363 :     R10     Address of current spot in buffer
00CD 364 :     R11     Mapping switch:
00CD 365 :         -1   Do physical -> virtual map
00CD 366 :         0    No mapping required
00CD 367 :         1    Do virtual -> physical translation
00CD 368 :
00CD 369 : Outputs:
00CD 370 :     R3      Byte fetched from memory (GETBYTE)
00CD 371 : --
00CD 372 :
00CD 373 : .ENABL  LSB
00CD 374 :
00CD 375 GETBYTE:
00CD 376 :     CLRL    R3                ; Return 0 if byte count = 0
00CD 377 :     TSTL    R8                ; Is byte count 0?
00CD 378 :     BEQL    90$               ; Yes
00CD 379 :     MOVZBL  (R10)+,R3         ; Get byte
00CD 380 :     BRB     10$               ; Branch to common code
00CD 381 :
00CD 382 :
00CD 383 PUTBYTE:
00CD 384 :     TSTL    R8                ; Is byte count 0?
00CD 385 :     BEQL    90$               ; Yes
00CD 386 :     MOVB    R3,(R10)+        ; Store byte
00CD 387 :
00CD 388 :
00CD 389 10$:  DECL    R8                ; Decr. byte count
00CD 390 :     BEQL    90$               ; Reached zero
00CD 391 :     BITW    #VASM_BYTE,R10   ; Did address overflow onto new page?
00CD 392 :     BNEQ    90$               ; No
00CD 393 :     INCL    R7                ; Yes, increment page number
00CD 394 :
00CD 395 :
00CD 396 : Fall through to ...
00CD 397 :
00CD 398 :
00CD 399 :
00CD 400 :++
00CD 401 : DO_MAPPING - Subroutine to perform necessary mapping
00CD 402 :
00CD 403 : Inputs:
```

```
00EC 404 : R6 Address of page table
00EC 405 : R7 Page number of buffer
00EC 406 : R10 Address to map
00EC 407 : R11 Mapping switch:
00EC 408 : -1 Do physical -> virtual map
00EC 409 : 0 No mapping required
00EC 410 : 1 Do virtual -> physical translation
00EC 411 :
00EC 412 : Outputs:
00EC 413 : R10 Address to use
00EC 414 : --
00EC 415 :
00EC 416 DO_MAPPING:
00EC 417 TSTL R11 ; Any mapping required?
00EE 418 BEQL 90$ ; No
00F0 419 BLSS 100$ ; Yes, map physical to virtual
SA FFFFFE00 8F CA 00F2 420 BICL #^C<VASM_BYTE>,R10 ; Yes, translate virtual to physical
00F9 421 ; Clear everything but byte offset
00F9 422 MOVL (R6)[R7],R0 ; Get PFN in R0
5A 15 50 6647 DO 00FD 423 INSV R0,#VAV_VPN,#PTESS_PFN,R10 ; Insert PFN after byte offset
5A 15 09 50 FO 0102 424 90$: RSB
0103 425
0103 426
0103 427 :
0103 428 : Map physical to virtual
0103 429 :
00 0103 430 100$: HALT ; Not implemented yet
0104 431
0104 432 .DSABL LSB
0104 433
58 45 2E 52 45 56 49 52 44 58 44 00' 0104 434 DXNAME: .ASCIC /DXDRIVER.EXE/
45 0110
0C 0104
00000111 0111 435
0111 436 CONSDX_DRVSIZ=-CONSDX_DRIVER
0111 437
0111 438 .END
```



DXBTDRIVR  
Symbol table

- CONSOLE RX01 BOOT DRIVER

G 11

15-SEP-1984 23:51:54 VAX/VMS Macro V04-00  
4-SEP-1984 23:04:18 [BOOTS.SRC]DXBTDRIVR.MAR;1

Page 10  
(3)

|                 |            |   |    |
|-----------------|------------|---|----|
| \$TABLE         | = 00000000 | R | 02 |
| BTDSK_CONSOLE   | = 00000040 |   |    |
| CONSDX_DRIVER   | = 00000000 | R | 03 |
| CONSDX_DRVSIZ   | = 00000111 |   |    |
| DO_MAPPING      | = 000000EC | R | 03 |
| DXIN            | = 000000BB | R | 03 |
| DXNAME          | = 00000104 | R | 03 |
| DXOUT           | = 000000AB | R | 03 |
| DX_FUNCCOMPLETE | = 00000200 |   |    |
| DX_READSECTOR   | = 00000900 |   |    |
| DX_WRITESECTOR  | = 00000901 |   |    |
| FUNC            | = 00000010 |   |    |
| GETBYTE         | = 000000CD | R | 03 |
| IOS_READBLK     | = 00000021 |   |    |
| MODE            | = 00000014 |   |    |
| PRS_MAPEN       | = 00000038 |   |    |
| PRS_RXCS        | = 00000020 |   |    |
| PRS_RXDB        | = 00000021 |   |    |
| PRS_SID_TYP780  | = 00000001 |   |    |
| PRS_TXCS        | = 00000022 |   |    |
| PRS_TXDB        | = 00000023 |   |    |
| PTESS_PFN       | = 00000015 |   |    |
| PUTBYTE         | = 000000D8 | R | 03 |
| RXCS_V_DONE     | = 00000007 |   |    |
| RXDB_S_SEL      | = 00000004 |   |    |
| RXDB_V_SEL      | = 00000008 |   |    |
| SIZ...          | = 00000004 |   |    |
| SS\$_CTRLERR    | = 00000054 |   |    |
| SS\$_NORMAL     | = 00000001 |   |    |
| TRKSEC          | = 00000091 | R | 03 |
| TXCS_V_READY    | = 00000007 |   |    |
| TXDB_V_SEL      | = 00000008 |   |    |
| VASM_BYTE       | = 000001FF |   |    |
| VASV_VPN        | = 00000009 |   |    |

! Psect synopsis !

| PSECT name  | Allocation       | PSECT No. | Attributes                                               |
|-------------|------------------|-----------|----------------------------------------------------------|
| . ABS .     | 00000000 ( 0.)   | 00 ( 0.)  | NOPICT USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE |
| \$ABSS      | 00000000 ( 0.)   | 01 ( 1.)  | NOPICT USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE       |
| BOOTDRIVR_4 | 00000028 ( 40.)  | 02 ( 2.)  | NOPICT USR CON REL LCL NOSHR EXE RD WRT NOVEC BYTE       |
| BOOTDRIVR_2 | 00000111 ( 273.) | 03 ( 3.)  | NOPICT USR CON REL LCL NOSHR EXE RD WRT NOVEC BYTE       |

! Performance indicators !

| Phase              | Page faults | CPU Time    | Elapsed Time |
|--------------------|-------------|-------------|--------------|
| Initialization     | 31          | 00:00:00.10 | 00:00:00.47  |
| Command processing | 127         | 00:00:00.63 | 00:00:01.63  |
| Pass 1             | 316         | 00:00:09.87 | 00:00:19.76  |
| Symbol table sort  | 0           | 00:00:01.63 | 00:00:03.35  |
| Pass 2             | 84          | 00:00:01.92 | 00:00:03.86  |

|                        |     |             |             |
|------------------------|-----|-------------|-------------|
| Symbol table output    | 5   | 00:00:00.08 | 00:00:00.08 |
| Psect synopsis output  | 2   | 00:00:00.03 | 00:00:00.03 |
| Cross-reference output | 0   | 00:00:00.00 | 00:00:00.00 |
| Assembler run totals   | 567 | 00:00:14.26 | 00:00:29.19 |

The working set limit was 1500 pages.

56157 bytes (110 pages) of virtual memory were used to buffer the intermediate code.

There were 60 pages of symbol table space allocated to hold 1059 non-local and 13 local symbols.

438 source lines were read in Pass 1, producing 14 object records in Pass 2.

19 pages of virtual memory were used to define 17 macros.

-----  
! Macro library statistics !  
-----

| Macro library name                  | Macros defined |
|-------------------------------------|----------------|
| -----                               | -----          |
| \$255\$DUA28:[BOOTS.OBJ]BOOTS.MLB;1 | 1              |
| \$255\$DUA28:[SYS.OBJ]LIB.MLB;1     | 4              |
| \$255\$DUA28:[SYSLIB]STARLET.MLB;2  | 8              |
| TOTALS (all libraries)              | 13             |

1169 GETS were required to define 13 macros.

There were no errors, warnings or information messages.

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



0038 AH-BT13A-SE  
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION  
CONFIDENTIAL AND PROPRIETARY