

BBBBBBBBBBBB		00000000		00000000		TTTTTTTTTTTT		SSSSSSSSSS
BBBBBBBBBBBB		00000000		00000000		TTTTTTTTTTTT		SSSSSSSSSS
BBBBBBBBBBBB		00000000		00000000		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		00000000		00000000		TTT	SSSSSSSSSS	
BBBBBBBBBBBB		00000000		00000000		TTT	SSSSSSSSSS	
BBBBBBBBBBBB		00000000		00000000		TTT	SSSSSSSSSS	

[illegible]

(2) 68
(3) 105

DECLARATIONS
11/790 Console RL02 Bootstrap Driver Code

```
0000 1      .TITLE CVBTDRIVR - 11/790 CONSOLE RL02 BOOTDRIVER
0000 2      .IDENT 'V04-000'
0000 3
0000 4
0000 5 *****
0000 6
0000 7      *
0000 8      *  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 9      *  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 10     *  ALL RIGHTS RESERVED.
0000 11
0000 12     *  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 13     *  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 14     *  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 15     *  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 16     *  OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 17     *  TRANSFERRED.
0000 18
0000 19     *  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 20     *  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 21     *  CORPORATION.
0000 22
0000 23     *  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 24     *  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 25     *
0000 26 *****
0000 27
0000 28
0000 29 ++
0000 30 FACILITY:      BOOTS
0000 31
0000 32 ABSTRACT:
0000 33      This module contains the bootstrap device driver for the
0000 34      11/790 RL02 console disk.
0000 35
0000 36 ENVIRONMENT:  IPL 31, kernel mode, code must be PIC
0000 37
0000 38 AUTHOR:  Trudy Matthews,      CREATION DATE:  14-Nov-1982
0000 39
0000 40 MODIFIED BY:
0000 41
0000 42 V03-005 TCM0005      Trudy C. Matthews      13-Aug-1984
0000 43      Fix DO_MAPPING routine so that it doesn't overwrite the
0000 44      STXCS status in R0. Fix GETWORD and PUTWORD so they call
0000 45      DO_MAPPING at every page boundary.
0000 46
0000 47 V03-004 TCM0004      Trudy C. Matthews      03-Jan-1984
0000 48      Correct some bugs.
0000 49
0000 50 V03-003 TCM0003      Trudy C. Matthews      17-Oct-1983
0000 51      Make sure routines READ_1_BLK and WRITE_1_BLK exit with
0000 52      correct status in R0.
0000 53
0000 54 V03-002 TCM0002      Trudy C. Matthews      31-May-1983
0000 55      Restore R4 correctly from stack on error paths in READ_1_BLK
0000 56      and WRITE_1_BLK. Also add correct name of standard console
0000 57      RL02 driver (CVDRIVER), and change module name from VENBTDRVR
```


0000 58 :
0000 59 :
0000 60 :
0000 61 :
0000 62 :
0000 63 :
0000 64 :
0000 65 :
0000 66 :--

to CVBTDRIVR.

V03-001 TCM0001 Trudy C. Matthews 27-Apr-1983
Change number of "interrupts" per block of data transferred
from 256 to 257 (1 per word + 1 completion). Use
"READ_PHYSICAL" and "WRITE_PHYSICAL" I/O function codes
to avoid using the VENUS console RL02 driver's bad block
replacement scheme.

```
0000 68      .SBTTL DECLARATIONS
0000 69      :
0000 70      : INCLUDE FILES:
0000 71      :
0000 72      :
0000 73      $BTDDEF      : Boot device types
0000 74      $IODEF      : I/O function codes
0000 75      $PRDEF      : Processor registers
0000 76      $PTEDEF     : PTE definitions
0000 77      $RPBDEF     : RPB offsets
0000 78      $SSDEF      : Status codes
0000 79      $STXCSDEF   : Console storage device CSR
0000 80      $STXDBDEF   : Console storage device data register
0000 81      $VADEF      : Virtual address fields
0000 82      :
0000 83      :
0000 84      : MACROS:
0000 85      :
0000 86      :
0000 87      :
0000 88      : EQUATED SYMBOLS:
0000 89      :
0000 90      :
0000 91      :
0000 92      : OWN STORAGE:
0000 93      :
0000 94      :
0000 95      :
0000 96      : Boot driver table entry
0000 97      :
0000 98      :
0000 99      $BOOT_DRIVER  DEVTYP = BTD$K_CONSOLE,- : Device type (console)
0000 100      CPUYPE = PR$_STD_TYP790,- : Cpu type (11/790)
0000 101      SIZE = CONSRL_DRV$IZ,- : Driver size
0000 102      ADDR = CONSRL_DRIVER,- : Driver address
0000 103      DRVRNAME = CONSRL_NAME : Driver file name
```



```
0000 105 .SBTTL 11/790 Console RL02 Bootstrap Driver Code
0000 106
0000 107 :++
0000 108
0000 109 :Inputs:
0000 110
0000 111 R1 Address of page table for virtual -> physical mapping
0000 112 R2 Base VPN of transfer (Bits 29:9 of R10)
0000 113 R5 LBN for current piece of transfer
0000 114 R8 Size of transfer in bytes
0000 115 R9 Address of the RPB
0000 116 R10 Starting address of transfer
0000 117
0000 118 FUNC(AP) I/O operation (IO$_READLBLK or IO$_WRITELBLK only)
0000 119 MODE(AP) Address interpretation mode:
0000 120 0 -> Physical, 1 -> Virtual
0000 121
0000 122 :Outputs:
0000 123
0000 124 R0 Status code:
0000 125
0000 126 $$$_NORMAL Successful transfer
0000 127 $$$_CTRLERR Fatal controller error
0000 128 :--
0000 129
00000010 0000 130 FUNC = 16
00000014 0000 131 MODE = 20
0000 132
0000 133 CONSRL_DRIVER:
0D2E 8F BB 0000 134 PUSH R1,R2,R3,R5,R8,R10,R11 ; Save some registers
0004 135
0004 136
0004 137 : Perform initialization: Set up a mapping switch in R11.
0004 138 : There are 4 possibilities concerning mapping: the I/O can be
0004 139 : done virtual or physical (MODE(AP)) and we can be executing virtual
0004 140 : or physical (contents of processor register PR$_MAPEN). If both
0004 141 : modes match, then we can just copy data to/from the user buffer.
0004 142 : If the I/O is to be done virtual and we are executing physical
0004 143 : then the buffer address has to be translated using the page table
0004 144 : pointed to by R1. If the I/O is to be done physical and we are
0004 145 : executing virtual then we have to double map the buffer using a spare
0004 146 : PTE. At this point, we just compute a mapping switch in R11 as follows:
0004 147
0004 148 0 Both modes match, just copy the data
0004 149 1 Do virtual -> physical translation using page table
0004 150 -1 Do physical -> virtual mapping using a spare PTE
0004 151
0004 152 MFPR #PR$_MAPEN,R11 ; Get mapping enabled switch
5B 38 DB 0004 153 MNEGL R11,R11 ; Negate it
5B 5B CE 0007 154 ADDL MODE(AP),R11 ; Add I/O mode switch
5B 14 AC CO 000A 155
000E 156 : Set up mapping if required
000E 157
000E 158 BSBW DO_MAPPING
010F 30 000E 159
0011 160
0011 161 : Transfer data, one block at a time.
```

```

57 0073'CF DE 0011 162 ;
20 10 AC B1 0011 163 ; MOVAL W^WRITE_1_BLK,R7 ; Assume we're doing a write.
05 13 0016 164 CMPW FUNC(APT,#10$_WRITEBLK ; Function = write?
57 0047'CF DE 001A 165 BEQL TRANSFER ; Yes, go get started.
001C 166 MOVAL W^READ_1_BLK,R7 ; We're reading.
0021 167
0021 168 TRANSFER:
53 00000200 8F D0 0021 169 MOVL #512,R3 ; Assume transferring 1 block.
53 58 D1 0028 170 CMPL R8,R3 ; Minimize assumed byte count (512
03 18 002B 171 BGEQ 10$ ; bytes) with actual byte count in R8.
53 58 D0 002D 172 MOVL R8,R3 ; R3 <- byte count for this iteration.
0030 173 10$:
67 16 0030 174 JSB (R7) ; Read or write 1 block (or less).
0D 50 E9 0032 175 BLBC R0,20$ ; Return if there was an error.
58 00000200 8F C2 0035 176 SUBL #512,R8 ; Done with another block.
04 15 003C 177 BLEQ 20$ ; Branch if we're all done.
55 D6 003E 178 INCL R5 ; Next LBN.
DF 11 0040 179 BRB TRANSFER ; Go do another block.
0042 180 20$:
0D2E 8F BA 0042 181 POPR #^M<R1,R2,R3,R5,R8,R10,R11> ; Restore registers.
05 0046 182 RSB
```



```
0047 184 :++
0047 185 : READ_1_BLOCK - Subroutine to read 1 block of data
0047 186 : WRITE_1_BLOCK - Subroutine to write 1 block of data
0047 187 :
0047 188 : Inputs:
0047 189 : R1: addr of page table for virtual -> physical mapping
0047 190 : R2: base VPN of transfer
0047 191 : R3: # bytes to transfer (<= 512)
0047 192 : R5: LBN of block to transfer
0047 193 : R10: address of memory buffer
0047 194 : R11: mapping switch
0047 195 :
0047 196 : Outputs:
0047 197 : Specified number of bytes are transferred.
0047 198 : If an error occurs, R0 contains SS$ CTRLERR.
0047 199 : R10: next free byte in memory buffer
0047 200 : All other registers preserved.
0047 201 :--
0047 202 :
0047 203 READ_1_BLK:
0047 204 : PUSHL R4 : Save a register.
54 55 54 DD 0047 204 : PUSHL R4 : Save a register.
54 55 08 78 0049 205 : ASHL #8,R5,R4 : Position LBN of block to transfer.
54 54 06 90 004D 206 : MOVB #STXC$C_READP,R4 : Form console command in R4.
0050 207 :
0050 208 : Move the read data into the memory buffer.
0050 209 :
0050 210 10$:
0050 211 : BSBW DL_READ : Read a word from the console.
00A6 30 0053 212 : BSBW PUTWORD : Put the word in memory data buffer.
18 EC 0056 213 : CMPV #STXC$V_STATUS,- : Check that status = continue
02 50 08 0058 214 : #STXC$S_STATUS,R0,- : transaction.
005B 215 : #STXC$C_CONT
005D 216 : BEQL 10$ : Branch if so.
54 8E DO 005D 217 :
54 18 EF 0060 218 : MOVL (SP)+,R4 : Restore R4.
50 50 08 0062 219 : EXTZV #STXC$V_STATUS,- : Put final status in R0 and check that
01 50 91 0065 220 : #STXC$S_STATUS,R0,R0 : it signals "transaction complete".
01 01 12 0068 221 : CMPB R0,#STXC$C_COMPLT
006A 222 : BNEQ HW_ERROR : If not, report an error.
006B 223 : RSB
006B 224 :
006B 225 :
006B 226 : Come here if there is an error during the transfer.
006B 227 :
006B 228 HW_ERROR:
50 00000054 8F DO 006B 229 : MOVL #SS$_CTRLERR,R0 : Set failure status.
05 0072 230 : RSB : Return and retry.
0073 231 :
0073 232 :
0073 233 : Write data from memory buffer to the console.
0073 234 :
0073 235 WRITE_1_BLK:
54 55 54 DD 0073 236 : PUSHL R4 : Save a register.
54 55 08 78 0075 237 : ASHL #8,R5,R4 : Position LBN of block to transfer.
54 54 05 90 0079 238 : MOVB #STXC$C_WRITEP,R4 : Form console command in R4.
007C 239 :
0062 30 007C 240 10$: BSBW GETWORD : Get a word from memory data buffer.
```

	003A	30	007F	241	BSBW	DL_WRITE		: Write it to the console.
	18	EC	0082	242	CMPV	#STXC\$V_STATUS,-		: Check that status = continue
02	50	08	0084	243		#STXC\$S_STATUS,R0,-		: transaction.
			0087	244		#STXC\$C_CONT		
	F3	13	0087	245	BEQL	10\$		: Loop to continue writing.
			0089	246				
	54	8E	D0	0089	MOVL	(SP)+,R4		: Restore R4.
	18	EF	008C	248	EXTZV	#STXC\$V_STATUS,-		: Put final status in R0 and check that
50	50	08	008E	249		#STXC\$S_STATUS,R0,R0		: it signals "transaction complete".
01	50	91	0091	250	CMPB	R0,#STXC\$C_COMPLT		
	D5	12	0094	251	BNEQ	HW_ERROR		: If not, report an error.
		05	0096	252	RSB			


```
0097 254 :++
0097 255 : DL_READ - Subroutine to read a word from console RL02.
0097 256 : DL_WRITE - Subroutine to write a word of data to console RL02
0097 257 :
0097 258 : Inputs:
0097 259 :         R4      Console command to write to STXCS.
0097 260 :         R6      Word of data to write to RL02 (DL_WRITE only).
0097 261 :
0097 262 : Outputs:
0097 263 :         R6      Word read from console (DL_READ only).
0097 264 :         R0      Contents of STXCS after read completes.
0097 265 : --
0097 266 DL_READ:
50 0000004C 8F DB 0097 267 MFPR #PR$ STXCS,R0 ; Get console status.
   F5 50 07 E1 009E 268 BBC #STXC$$V_READY,R0,DL_READ ; Loop until ready.
0000004C 8F 54 DA 00A2 269 MTPR R4,#PR$ STXCS ; Console command to read a word of data.
50 0000004C 8F DB 00A9 270 10$: MFPR #PR$ STXCS,R0 ; Get console status.
   F5 50 07 E1 00B0 271 BBC #STXC$$V_READY,R0,10$ ; Loop until ready.
56 0000004D 8F DB 00B4 272 MFPR #PR$_STXDB,R6 ; Read a word of data.
   05 00BB 273 RSB
   00BC 274
   00BC 275
   00BC 276 DL_WRITE:
50 0000004C 8F DB 00BC 277 MFPR #PR$ STXCS,R0 ; Get console status.
   F5 50 07 E1 00C3 278 BBC #STXC$$V_READY,R0,DL_WRITE ; Loop until ready.
0000004D 8F 56 DA 00C7 279 MTPR R6,#PR$_STXDB ; Write data to console.
0000004C 8F 54 DA 00CE 280 MTPR R4,#PR$ STXCS ; Notify console that data is present.
50 0000004C 8F DB 00D5 281 10$: MFPR #PR$ STXCS,R0 ; Get console status.
   F5 50 07 E1 00DC 282 BBC #STXC$$V_READY,R0,10$ ; Loop until ready.
   05 00E0 283 RSB
```



```
00E1 285 :++
00E1 286 : GETWORD - Subroutine to get a word from memory
00E1 287 : PUTWORD - Subroutine to store a word in memory
00E1 288 :
00E1 289 :     These two subroutines do two things special:
00E1 290 :
00E1 291 :     1) Since you must always read or write one entire block
00E1 292 :         (512 bytes) through the console interface to the RL02,
00E1 293 :         these routines simply return if the memory buffer is
00E1 294 :         empty/full before the entire block has been transferred.
00E1 295 :
00E1 296 :     2) These routines take care of page boundaries if
00E1 297 :         mapping is required.
00E1 298 :
00E1 299 :     Inputs:
00E1 300 :         R1      Address of page table
00E1 301 :         R2      Virtual page number of buffer
00E1 302 :         R3      Size of remaining buffer (in bytes)
00E1 303 :         R6      Word to store (PUTWORD only)
00E1 304 :         R10     Address of current spot in buffer
00E1 305 :         R11     Mapping switch:
00E1 306 :             -1   Do physical -> virtual map
00E1 307 :             0    No mapping required
00E1 308 :             1    Do virtual -> physical translation
00E1 309 :
00E1 310 :     Outputs:
00E1 311 :         R6      Word fetched from memory (GETWORD only)
00E1 312 :         --
00E1 313 :
00E1 314 : GETWORD:
56      56      D4 00E1 314 CLRL      R6      ; Return 0 if byte count = 0
53      53      D5 00E3 315 TSTL      R3      ; Is byte count 0?
14      14      13 00E5 316 BEQL      10$     ; Yes, return 0 data.
56      8A      9A 00E7 317 MOVZBL   (R10)+,R6 ; Get first byte from memory.
2A      2A      10 00EA 318 BSBB      CHECK_BOUNDS ; Else check for page boundary.
53      53      D7 00EC 319 DECL      R3      ; Decrement byte count.
56      56      0B 13 00EE 320 BEQL      10$     ; If 0 we're done.
56      56      08 9C 00F0 321 ROTL     #8,R6,R6 ; Move 1st byte up to bit 8.
56      56      8A 90 00F4 322 MOVB     (R10)+,R6 ; Get 2nd byte in low-order byte of R6.
53      53      D7 00F7 323 DECL      R3      ; Decrement byte count.
1B      1B      18 00F9 324 BGEQ     CHECK_BOUNDS ; If positive, check for page boundary.
          05      00FB 325 10$: RSB
          00FC 326
          00FC 327
          53      D5 00FC 328 PUTWORD: TSTL      R3      ; Is byte count 0?
          15      13 00FE 329 BEQL      10$     ; Yes, don't write anything in buffer.
8A      8A      90 0100 330 MOVB     R6,(R10)+ ; Put 1st byte in memory buffer.
          11      10 0103 331 BSBB      CHECK_BOUNDS ; Check for page boundary.
          53      D7 0105 332 DECL      R3      ; Decrement byte count.
          0C      13 0107 333 BEQL      10$     ; If 0 we're done.
56      56      F8 8F 78 0109 334 ASHL     #-8,R6,R6 ; Bring 2nd byte down to bit 0.
          8A      56 90 010E 335 MOVB     R6,(R10)+ ; Put 2nd byte into memory buffer.
          53      D7 0111 336 DECL      R3      ; Decrement byte count.
          01      18 0113 337 BGEQ     CHECK_BOUNDS ; If positive, check for page boundary.
          05      05 0115 338 10$: RSB
          0116 339
          5A      01FF 8F B3 0116 340 CHECK_BOUNDS:
          0116 341 BITW      #VASM_BYTE,R10 ; Did address overflow onto new page?
```


CVBTDRIVR
V04-000

- 11/790 CONSOLE RL02 BOOTDRIVER C 6
11/790 Console RL02 Bootstrap Driver Cod

16-SEP-1984 00:16:33 VAX/VMS Macro V04-00
4-SEP-1984 23:03:55 [BOOTS.SRC]CVBTDRIVR.MAR;1

Page 10
(6)

01 13 011B 342 BEQL 10\$
05 011D 343 RSB
011E 344
52 D6 011E 345 10\$: INCL R2
0120 346

; Yes, must update mapping.

; Yes, increment virtual page number
; and fall through to DO_MAPPING.

```
0120 348 :++
0120 349 : DO_MAPPING - Subroutine to perform necessary mapping
0120 350 :
0120 351 : Inputs:
0120 352 :      R1      Address of page table
0120 353 :      R2      Page number of buffer
0120 354 :      R10     Address to map
0120 355 :      R11     Mapping switch:
0120 356 :             -1  Do physical -> virtual map
0120 357 :             0   No mapping required
0120 358 :             1   Do virtual -> physical translation
0120 359 :
0120 360 : Outputs:
0120 361 :      R10     Address to use
0120 362 :      R0      destroyed
0120 363 : --
0120 364 :
0120 365 DO_MAPPING:
0120 366      TSTL    R11                ; Any mapping required?
0122 367      BEQL    90$           ; No
0124 368      BLSS    100$          ; Yes, map physical to virtual
0126 369      BICL    #^C<VASM_BYTE>,R10 ; Yes, translate virtual to physical
012D 370                ; Clear everything but byte offset
012D 371      PUSHL   R0            ; save R0
012F 372      MOVL    (R1)[R2],R0  ; Get PFN in R0
0133 373      INSV    R0,#VASV_VPN,#PTESS_PFN,R10 ; Insert PFN after byte offset
0138 374      MOVL    (SP)+,R0     ; Restore previous value of R0
013B 375 90$:      RSB
013C 376
013C 377
013C 378 :
013C 379 : Map physical to virtual
013C 380 :
013C 381 100$:      HALT                ; Not implemented yet
013D 382
013D 383      .DSABL   LSB
013D 384
013D 385 CONSRL_NAME:      .ASCIC /CVDRIVER.EXE/
0149 386
014A 387 CONSRL_DRVSIZ=.-CONSRL_DRIVER
014A 388
014A 389      .END
```

5B D5 0120 366
17 13 0122 367
16 19 0124 368
5A FFFFFFFE00 8F CA 0126 369
50 DD 012D 370
5A 15 50 6142 DD 012F 372
09 50 FO 0133 373
50 8E DO 0138 374
05 013B 375 90\$:
013C 376
013C 377
013C 378
013C 379
013C 380
00 013C 381
013D 382
013D 383
013D 384
58 45 2E 52 45 56 49 52 44 56 43 00' 013D 385
45 0149 386
0C 013D 387
0000014A 014A 388
014A 389

CVBTDRIVR
Symbol table

- 11/790 CONSOLE RL02 BOOTDRIVER E 6

16-SEP-1984 00:16:33 VAX/VMS Macro V04-00
4-SEP-1984 23:03:55 [BOOTS.SRC]CVBTDRIVR.MAR;1

Page 12
(7)

STABLE	= 00000000	R	02
BTDSK_CONSOLE	= 00000040		
CHECK_BOUNDS	00000116	R	03
CONSRC_DRIVER	00000000	R	03
CONSRL_DRVSIZ	= 0000014A		
CONSRL_NAME	0000013D	R	03
DL_READ	00000097	R	03
DL_WRITE	000000BC	R	03
DO_MAPPING	00000120	R	03
FUNC	= 00000010		
GETWORD	000000E1	R	03
HW_ERROR	0000006B	R	03
IOS_WRITELBLK	= 00000020		
MODE	= 00000014		
PRS_MAPEN	= 00000038		
PRS_SID_TYP790	= 00000004		
PRS_STXCS	= 0000004C		
PRS_STXDB	= 0000004D		
PTSS_PFN	= 00000015		
PUTWORD	000000FC	R	03
READ_1_BLK	00000047	R	03
SSS_CTRLERR	= 00000054		
STXCSSC_COMPLT	= 00000001		
STXCSSC_CONT	= 00000002		
STXCSSC_READP	= 00000006		
STXCSSC_WRITEP	= 00000005		
STXCSSS_STATUS	= 00000008		
STXCSSV_READY	= 00000007		
STXCSSV_STATUS	= 00000018		
TRANSFER	00000021	R	03
VASM_BYTE	= 000001FF		
VASV_VPN	= 00000009		
WRITE_1_BLK	00000073	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
\$ABSS	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	0000014A (330.)	03 (3.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC BYTE

+-----+
! Performance indicators !
+-----+

Phase	Page faults	CPU Time	Elapsed Time
Initialization	29	00:00:00.07	00:00:00.90
Command processing	129	00:00:00.86	00:00:03.76
Pass 1	330	00:00:10.07	00:00:26.03
Symbol table sort	0	00:00:01.63	00:00:02.40
Pass 2	80	00:00:02.00	00:00:03.26
Symbol table output	6	00:00:00.06	00:00:00.06

Psect synopsis output	2	00:00:00.02	00:00:00.06
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	578	00:00:14.71	00:00:36.47

The working set limit was 1650 pages.

55773 bytes (109 pages) of virtual memory were used to buffer the intermediate code.

There were 60 pages of symbol table space allocated to hold 1067 non-local and 11 local symbols.

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

18 pages of virtual memory were used to define 16 macros.

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

Macro library name

Macros defined

-----	-----
\$255\$DUA28:[SHRLIB]790DEF.MLB;1	2
\$255\$DUA28:[BOOTS.OBJ]BOOTS.MLB;1	1
\$255\$DUA28:[SYS.OBJ]LIB.MLB;1	4
\$255\$DUA28:[SYSLIB]STARLET.MLB;2	6
TOTALS (all libraries)	13

1198 GETS were required to define 13 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS\$:CVBTDRIVR/OBJ=OBJ\$:CVBTDRIVR MSRC\$:CVBTDRIVR/UPDATE=(ENH\$:CVBTDRIVR)+EXECML\$/LIB+LIB\$:BOOTS.MLB/LIB+SHRLIB\$:790DEF.ML

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

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY