

```
UUU      UUU VVV      VVV      111      RRRRRRRRRRRR      0000000000      MMM      MMM
UUU      UUU VVV      VVV      111      RRRRRRRRRRRR      0000000000      MMM      MMM
UUU      UUU VVV      VVV      111      RRRRRRRRRRRR      0000000000      MMM      MMM
UUU      UUU VVV      VVV      111111      RRR      RRR      000      000      MMMMMM      MMMMMM
UUU      UUU VVV      VVV      111111      RRR      RRR      000      000      MMMMMM      MMMMMM
UUU      UUU VVV      VVV      111111      RRR      RRR      000      000      MMMMMM      MMMMMM
UUU      UUU VVV      VVV      111      RRR      RRR      000      000      MMM      MMM      MMM
UUU      UUU VVV      VVV      111      RRR      RRR      000      000      MMM      MMM      MMM
UUU      UUU VVV      VVV      111      RRR      RRR      000      000      MMM      MMM      MMM
UUU      UUU VVV      VVV      111      RRR      RRR      000      000      MMM      MMM      MMM
UUU      UUU VVV      VVV      111      RRR      RRR      000      000      MMM      MMM      MMM
UUU      UUU VVV      VVV      111      RRR      RRR      000      000      MMM      MMM      MMM
UUU      UUU VVV      VVV      111      RRR      RRR      000      000      MMM      MMM      MMM
UUU      UUU VVV      VVV      111      RRR      RRR      000      000      MMM      MMM      MMM
UUU      UUU VVV      VVV      111      RRR      RRR      000      000      MMM      MMM      MMM
UUU      UUU VVV      VVV      111      RRR      RRR      000      000      MMM      MMM      MMM
UUU      UUU VVV      VVV      111      RRR      RRR      000      000      MMM      MMM      MMM
UUU      UUU VVV      VVV      111      RRR      RRR      000      000      MMM      MMM      MMM
UUU      UUU VVV      VVV      111      RRR      RRR      000      000      MMM      MMM      MMM
UUU      UUU VVV      VVV      111      RRR      RRR      000      000      MMM      MMM      MMM
UUU      UUU VVV      VVV      111      RRR      RRR      000      000      MMM      MMM      MMM
UUUUUUUUUUUUUUUUUU      VVV      1111111111      RRR      RRR      0000000000      MMM      MMM
UUUUUUUUUUUUUUUUUU      VVV      1111111111      RRR      RRR      0000000000      MMM      MMM
UUUUUUUUUUUUUUUUUU      VVV      1111111111      RRR      RRR      0000000000      MMM      MMM
```

```

NN      NN      EEEEEEEEEEE TTTTTTTTTT BBBB88888 000000 000000 TTTTTTTTTT
NN      NN      EEEEEEEEEEE TTTTTTTTTT BBBB88888 000000 000000 TTTTTTTTTT
NN      NN      EE          TT          BB      BB 00      00 00      00 00      00
NN      NN      EE          TT          BB      BB 00      00 00      00 00      00
NNNN    NN      EE          TT          BB      BB 00      00 00      00 00      00
NNNN    NN      EE          TT          BB      BB 00      00 00      00 00      00
NN      NN      EEEEEEEEEEE TT          BBBB88888 00      00 00      00 00      00
NN      NN      EEEEEEEEEEE TT          BBBB88888 00      00 00      00 00      00
NN      NN      EE          TT          BB      BB 00      00 00      00 00      00
NN      NN      EE          TT          BB      BB 00      00 00      00 00      00
NN      NN      EE          TT          BB      BB 00      00 00      00 00      00
NN      NN      EE          TT          BB      BB 00      00 00      00 00      00
NN      NN      EEEEEEEEEEE TT          BBBB88888 000000 000000 TTTT
NN      NN      EEEEEEEEEEE TT          BBBB88888 000000 000000 TTTT

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
LLLLLLLLLLLL IIIIII  SSSSSSSS
LLLLLLLLLLLL IIIIII  SSSSSSSS

```



```

1 0 module netboot;
2 0
3 0 {*****
4 0 {*
5 0 {* Copyright (c) 1984
6 0 {* by DIGITAL Equipment Corporation, Maynard, Mass.
7 0 {*
8 0 {* This software is furnished under a license and may be used and copied
9 0 {* only in accordance with the terms of such license and with the
10 0 {* inclusion of the above copyright notice. This software or any other
11 0 {* copies thereof may not be provided or otherwise made available to any
12 0 {* other person. No title to and ownership of the software is hereby
13 0 {* transferred.
14 0 {*
15 0 {* The information in this software is subject to change without notice
16 0 {* and should not be construed as a commitment by DIGITAL Equipment
17 0 {* Corporation.
18 0 {*
19 0 {* DIGITAL assumes no responsibility for the use or reliability of its
20 0 {* software on equipment which is not supplied by DIGITAL.
21 0 {*
22 0 {*****
23 0 {
24 0
25 0 {++
26 0 * FACILITY:
27 0 *
28 0 * VAXELN and MicroVAX I
29 0 *
30 0 * ABSTRACT:
31 0 *
32 0 * This module contains a network primary bootstrap. It handles the
33 0 * DNA Low-level Maintenance Protocol (MOP) messages to down-line
34 0 * load a VAX processor with a system memory image.
35 0 *
36 0 * AUTHOR:
37 0 *
38 0 * Len Kawell, 19-Nov-1981
39 0 *
40 0 * V1.0-01 Len Kawell 23-May-1984
41 0 * Call a new driver entry point to stop the device when the
42 0 * load is complete. This makes sure that the device does not
43 0 * continue receiving messages once the O/S is loaded.
44 0 --}
45 0

```

```
46 0      include $datalink;
47 0
48 0
49 0      const
50 0          memload_code      = 2;      { Memory load without xfer addr }
51 0          request_program_code = 8;    { Request program }
52 0          request_load_code  = 10;    { Request memory load }
53 0          parameters_code    = 20;    { Parameters with xfer addr }
54 0
55 0          opsys_pgmtype      = 2;      { Operating system program type }
56 0
57 0          transfer_parameter  = 0;      { Transfer address parameter }
58 0          node_name_parameter = 1;      { Node name parameter }
59 0          node_address_parameter = 2;   { Node address parameter }
60 0          host_name_parameter = 3;      { Host name parameter }
61 0          host_address_parameter = 4;   { Host address parameter }
62 0
63 0          maximum_retries     = 8;      { Maximum retries }
64 0
65 0          ss$_normal          = 1;      { Success }
66 0          ss$_bufferovf      = %x600;  { Buffer overflow error }
67 0          ss$_ctrlerr        = %x54;   { Device controller error }
68 0          ss$_devinact       = %x20d4; { Device initialization error }
69 0          ss$_nosuchnode     = %x28c;   { No response from load server }
70 0
71 0      type
72 0          byte = 0..255;      { Byte }
73 0          word = 0..65535;   { Word }
74 0          buffer = array[0..255] of char; { I/O buffer }
75 0
76 0          xmtmsg = packed record { Transmit message format}
77 0              code: byte;        { Code }
78 0              case byte of
79 0                  request_program_code: (
80 0                      devtype: byte; { Device type }
81 0                      mopver: byte;  { MOP version }
82 0                      pgmtype: byte; { Program type }
83 0                  );
84 0                  request_load_code: (
85 0                      loadnum: byte; { Load number }
86 0                      error: byte;  { Error flag }
87 0                  );
88 0              end;
89 0
90 0          rcvmsg = packed record { Receive message format}
91 0              code: byte;        { Code }
92 0              loadnum: byte;     { Load number }
93 0              case byte of
94 0                  memload_code: ( { Memory load }
95 0                      loadaddr: ^buffer; { Load address }
96 0                      image: buffer; { Image data }
97 0                  );
98 0                  parameters_code: ( { Parameters }
99 0                      param: packed array[1..255] of char;
100 0                  );
101 0
102 0
```



```
103 0          end;
104 0
105 0          bitmap = packed array[0..8192] of boolean; { PFN bitmap }
106 0
107 0          function net_init(var devtype: integer; data_buffer: ^anytype):boolean;
108 1          external;
109 0
110 0          function net_transmit(var buff: byte_data(buff_size); buff_size: word): boolean;
111 1          external;
112 0
113 0          function net_receive(var buff: byte_data(buff_size); buff_size: word): boolean;
114 1          external;
115 0
116 0          procedure net_stop;
117 1          external;
118 0
119 0
```



```
120 0      function boo$downline_load(load_address: integer;  
121 1      data_buffer: ^anytype;  
122 1      pfn_bitmap: ^bitmap;  
123 1      var transfer_address: integer;  
124 1      var node_address: datalink_address;  
125 1      var node_name: varying_string(6)): integer;  
126 1      {++  
127 1      boo$downline_load - Downline load a system image  
128 1      {  
129 1      Inputs:  
130 1      {  
131 1      load_address - Starting physical load address  
132 1      data_buffer - Pointer to an 8KB buffer for use by the load driver  
133 1      pfn_bitmap - Pointer to the Page Frame Number bitmap  
134 1      {  
135 1      Outputs:  
136 1      {  
137 1      transfer_address - Received transfer address  
138 1      node_address - Received node address  
139 1      node_name - Received node name  
140 1      {  
141 1      Function value -  
142 1      {  
143 1      ss$_normal - Downline load completed successfully  
144 1      ss$_bufferovf - Load buffer overflow (ie. PFN invalid)  
145 1      ss$_devinact - Device initialization error  
146 1      ss$_ctrlerr - Device I/O error  
147 1      ss$_nosuchnode - No response from a load server  
148 1      {--}  
149 1      var  
150 1      status: boolean;  
151 1      retries: integer := maximum_retries;  
152 1      request_opsys: boolean := true;  
153 1      current_loadnum: integer := 0;  
154 1      current_devtype: integer;  
155 1      xmtbuff: xmtmsg;  
156 1      rcvbuff: rcvmsg;  
157 1      pi: integer;  
158 1      begin  
159 1  
160 1      rcvbuff.code := 0;  
161 1  
162 1      { Initialize the driver }  
163 1
```

		0008	
		0008	BOO\$DOWNLINE_LOAD:
		0008	.entry BOO\$DOWNLINE_LOAD, ^m<dv,iv,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11>
SE	FEE8 CE 9E	000A	movab -0118(sp),sp
6E	04 AC D0	000F	movl 04(ap),(sp)
5B	0C AC D0	0013	movl 0C(ap),r11
	57 08 D0	0017	movl #8,r7
5A	01 90	001A	movb #1,r10
	58 D4	001D	clrl r8

		001F	
FEFA	CD 94	001F	clrb -0106(fp)


```
164 1      if not net_init(current_devtype, data_buffer)
           0023
           08 AC DD 0023      pushl 08(ap)
           FE F0 CD DF 0026      pushal -0110(fp)
00000000* EF 02 FB 002A      calls #2,NET_INIT
           08 50 E8 0031      blbs r0,vcg.1

165 1      then
166 1          begin
           0034
167 2          boo$downline_load := ss$_devinact;
           0034
           59 20D4 8F 3C 0034      movzwl #20D4,r9
168 2          goto return;
           0039
           016C 31 0039      brw RETURN
169 2          end;
           003C
           003C      vcg.1:

170 1      repeat
171 1          003C

172 2          { Create a Request Program or Request Load message }
173 2          if request_opsys
174 2              003C
175 2              18 5A E9 003C      blbc r10,vcg.2

176 2          then
177 2              begin
           003F
178 3              xmtbuff.code := request_program_code;
           003F
           FEF6 CD 08 90 003F      movb #8,-010A(fp)
179 3              xmtbuff.devtype := current_devtype;
           0044
           FEF7 CD FE F0 CD 90 0044      movb -0110(fp),-0109(fp)
180 3              xmtbuff.mopver := 1;
           004B
           FEF8 CD 01 90 004B      movb #1,-0108(fp)
181 3              xmtbuff.pgmttype := opsys_pgmttype;
           0050
           FEF9 CD 02 90 0050      movb #2,-0107(fp)
182 3              end
           0055
           0E 11 0055      brb vcg.3
```

```
                                0057  vcg.2:
183  2      else
184  2          begin
                                0057
185  3          xmtbuff.code := request_load_code;
                                0057
FEF6 CD 0A 90 0057          movb    #A,-010A(fp)
186  3          xmtbuff.loadnum := current_loadnum;
                                005C
FEF7 CD 58 90 005C          movb    r8,-0109(fp)
187  3          xmtbuff.error := 0;
                                0061
FEF8 CD 94 0061          clrb     -0108(fp)
188  3          end;
                                0065
                                0065  vcg.3:
189  2      { Transmit the message and retry if it fails. }
190  2      while not net_transmit(xmtbuff, size(xmtbuff)) do
191  2
192  2          0065
                                0065          pushl    #4
FEF6 CD 9F 0067          pushab   -010A(fp)
00000000* EF 02 FB 0068          calls   #2,NET_TRANSMIT
                                0072          blbs     r0,vcg.4
                                0072          blbs     r0,vcg.4
193  2          begin
                                0075
194  3          retries := retries - 1;
                                0075
57 D7 0075          decl     r7
195  3          if retries <= 0
                                0077
EC 14 0077          bgtr     vcg.3
196  3          then
197  3              begin
                                0079
198  4              boo$downline_load := ss$_ctrlerr;
                                0079
59 54 8F 9A 0079          movzbl   #54,r9
199  4              goto return;
                                007D
0128 31 007D          brw      RETURN
200  4          end;
```



```
                                0080
201 3      end;                                0080
                                0080 vcg.4:

202 2      { If last message received was a Parameters message, then load done }
203 2
204 2
205 2      if rcvbuff.code = parameters_code
                                0080
52 FEFA CD 9A 0080      movzbl -0106(fp),r2
   14 52 D1 0085      cmpl r2,#14
   03 12 0088      bneq gen.1
   0087 31 008A      brw DONE
                                008D gen.1:

206 2      then
207 2          goto done; 008D

208 2
209 2      { Receive a response message }
210 2
211 2      status := net_receive(rcvbuff, size(rcvbuff));
                                008D
7E 0106 8F 3C 008D      movzwl #106, -(sp)
FEFA CD 9F 0092      pushab -0106(fp)
00000000* EF 02 FB 0096      calls #2, NET_RECEIVE

212 2
213 2      { If a message successfully received and it is either a Memory Load }
214 2      { or a Parameters message, process it. }
215 2
216 2      if status and
                                009D
5B 50 E9 009D      blbc r0,vcg.9
52 FEFA CD 9A 00A0      movzbl -0106(fp),r2
   02 52 D1 00A5      cmpl r2,#2
   05 13 00A8      beql vcg.5
   14 52 D1 00AA      cmpl r2,#14
   4C 12 00AD      bneq vcg.9
                                00AF vcg.5:

217 2      ((rcvbuff.code = memload_code) or
218 2      (rcvbuff.code = parameters_code))
219 2      then
220 2          begin
                                00AF

221 3              if (rcvbuff.loadnum = current_loadnum)
                                00AF
52 FEFB CD 9A 00AF      movzbl -0105(fp),r2
   58 52 D1 00B4      cmpl r2,r8
   04 13 00B7      beql vcg.6
   52 D5 00B9      tstl r2
   39 12 00BB      bneq vcg.8
```

```
00BD  vcg.6:
222 3      or (rcvbuff.loadnum = 0)
223 3      then
224 3          begin
00BD
225 4          current_loadnum := (rcvbuff.loadnum + 1) mod 256;
00BD
52 FEFB CD 9A 00BD      movzbl -0105(fp),r2
52 D6 00BD      incl r2
58 52 9A 00C2      movzbl r2,r8
00C4

226 4      { If memory data, then move to proper memory location }
227 4
228 4      if rcvbuff.code = memload_code
229 4          if rcvbuff.code = memload_code
00C7
52 FEFA CD 9A 00C7      movzbl -0106(fp),r2
02 52 D1 00CC      cmpl r2,#2
25 12 00CF      bneq vcg.8

230 4      then
231 4          begin
00D1

232 5      { Compute memory location for data and check }
233 5      { that page is valid in page frame bitmap. }
234 5
235 5      rcvbuff.loadaddr::integer :=
236 5          FEFC CD 6E C0 00D1      addl2 (sp),-0104(fp)
00D1
237 5          rcvbuff.loadaddr::integer + load_address;
238 5
239 5      if not pfn_bitmap^
00D6
52 FEFC CD 00000200 8F C7 00D6      divl3 #200,-0104(fp),r2
08 68 52 E0 00E0      bbs r2,(r11),vcg.7

240 5      [rcvbuff.loadaddr::integer div 512]
241 5
242 5      then
00E4      begin
243 6          boo$downline_load := ss$_bufferovf;
00E4
59 0600 8F 3C 00E4      movzwl #600,r9
00E4

244 6          goto return;
00E9
00BC 31 00E9      brw RETURN

245 6          end;
00EC
00EC  vcg.7:
```



```
246 5
247 5      rcvbuff.loadaddr^ := rcvbuff.image;
      FEFC DD FF00 CD 0100 8F 28 00EC
      00EC      movc3 #100,-0100(fp),a-0104(fp)
248 5      end;
      00F6
      00F6      vcg.8:
249 4      end;
      00F6
250 3      request_opsys := false;
      00F6
      5A 94 00F6      clrb    r10
251 3      end
      00F8
      FF41 31 00F8      brw    vcg.1
      00FB      vcg.9:
252 2      else
253 2      begin
      00FB
254 3      { Decrement the retry count. If it is zero quit. }
255 3      retries := retries - 1;
256 3
257 3      00FB
      57 D7 00FB      decl    r7
258 3      if retries <= 0
      00FD
      03 15 00FD      bleq    gen.2
      FF3A 31 00FF      brw    vcg.1
      0102      gen.2:
259 3      then
260 3      begin
      0102
261 4      { Return device error status. If no reponse was ever }
262 4      { received from a load server, return no response }
263 4      { from server status. }
264 4
265 4      boo$downline_load := ss$_ctrlerr;
266 4      0102
      59 54 8F 9A 0102      movzbl #54,r9
267 4      if request_opsys
      0106
      03 5A E8 0106      blbs    r10,gen.3
      009C 31 0109      brw    RETURN
      010C      gen.3:
268 4      then
```

```
269 4      boo$downline_load := ss$_nosuchnode;
          59 028C 8F 3C      010C      movzwl #28C,r9
          010C
270 4      goto return;
          0094 31      0111      brw      RETURN
          0111
271 4      end;
          0114
272 3      end
          0114
273 2      until false;
          0114
274 1
275 1      done:
276 1
          0114
          0114      DONE:
277 1      { Get the load parameters }
278 1
279 1      pi := 1;
          56 01 D0      0114      movl      #1,r6
          0114
280 1      transfer_address := -1;
          5A 10 BC DE      0117
          6A 01 CE      0117      moval      @10(ap),r10
          011B      mnegl      #1,(r10)
281 1      node_name := '';
          58 18 BC DE      011E
          68 B4      011E      moval      @18(ap),r8
          0122      clrw      (r8)
282 1
283 1      with rcvbuff do
          0124
284 1      repeat
          57 14 BC DE      0124
          0124      vcg.10:      moval      @14(ap),r7
          0128
285 2
286 2      { Scan parameter list for parameters. }
287 2
288 2      case ord(param[pi]) of
          0128
          52 FEFB CD46 9A      0128      movzbl      -0105(fp)[r6],r2
          02 00 52 CF      012E      case1      r2,#0,#2
          0132      case.288:
          0132
```



```

      5A 11      0138 case.288.ow:
                    0138 brb      vcg.14

289 3
290 3      node_name_parameter:
                    013A
                    013A a case label:

291 3      node_name := substr(param, pi + 2, ord(param[pi + 1]));
                    013A
52 FEFC CD46 9A      013A      movzbl -0104(fp)[r6],r2
    68 52 F7      0140      cvtlw  r2,(r8)
    68 06 B1      0143      cmpw   #6,(r8)
    03 1E      0146      bgequ   vcg.11
    68 06 B0      0148      movw   #6,(r8)
02 A8 FEFD CD46 68 28      0148 vcg.11: movc3  (r8),-0103(fp)[r6],02(r8)
    3F 11      0153      brb      vcg.14

292 3
293 3      node_address_parameter:
                    0155
                    0155 a case label:

294 3      begin
                    0155

295 4
296 4      { If only a 2 byte node address sent, pad it using the }
297 4      { DECnet Ethernet high-order address. }
298 4
299 4      if ord(param[pi + 1]) > 2
                    0155
52 FEFC CD46 9A      0155      movzbl -0104(fp)[r6],r2
    02 52 D1      015B      cmpl   r2,#2
    09 15      015E      bleq    vcg.12

300 4
301 4      then
                    node_address :=
                    0160
67 FEFD CD46 06 28      0160      movc3  #6,-0103(fp)[r6],(r7)
    10 11      0167      brb      vcg.13
                    0169 vcg.12:

302 5
303 5      else
                    substr(param, pi + 2, size(node_address))
304 4      begin
                    0169

305 5      node_address := '(170,00,04,00,00,00,00,00)';
                    0169
67 FE92 CF 06 03      0169      movc3  #6,$CODE,(r7)

306 5      substr(node_address, 5, 2) := substr(param, pi + 2, 2);
                    016F
52 FEFD CD46 9E      016F      movab  -0103(fp)[r6],r2
    04 A7 62 B0      0175      movw   (r2),04(r7)
```

```
307 5      end;
          0179
          0179 vcg.13:

308 4
309 4      { If the node address does not have an area address, set }
310 4      { it to area 1. }
311 4
312 4      if ord(substr(node_address, 6, 1)) < 4
          0179
52 05 A7 9A 0179      movzbl 05(r7),r2
   04 52 D1 017D      cmpl   r2,#4
   12 18 0180      bgeq    vcg.14

313 4      then
314 4          substr(node_address, 6, 1) :=
          0182
          0182      addl2   #4,r2
52 05 A7 52 90 0185      movb   r2,05(r7)

315 5          chr(ord(substr(node_address, 6, 1)) + 4);
316 4      end;
          0189
          09 11 0189      brb    vcg.14

317 3
318 3      transfer_parameter:
          018B
          018B      a case label:

319 3
320 3          transfer_address := substr(param, pi + 1, 4)::integer;
          018B
52 FEFC CD46 9E 018B      movab  -0104(fp)[r6],r2
   6A 62 D0 0191      movl   (r2),(r10)
          0194 vcg.14:

321 3      end;
322 2
323 2      pi := pi + ord(param[pi + 1]) + 2;
          0194
52 FEFC CD46 9A 0194      movzbl -0104(fp)[r6],r2
   52 56 C0 019A      addl2   r6,r2
   56 52 02 C1 019D      addl3  #2,r2,r6

324 2
325 2      until transfer_address >= 0;
          01A1
          6A D5 01A1      tstl   (r10)
          83 19 01A3      blss   vcg.10

326 1
327 1      boo$downline_load := ss$_normal;
          01A5
          59 01 D0 01A5      movl   #1,r9

328 1
```


M 11
10-AUG-1984 18:07:00.24 VAXELN PASCAL X2.0-08 Page 13
23-MAY-1984 10:03:40 DISK\$STARWORK03:[GAMACHE.UV1ROM.VMB]NETB(3)

EPASCAL/MACH/LIS=LIS\$: /OBJ=OBJ\$: VMBS:NETBOOT+ELN\$:RTLOBJECT/LIB

PQ
Syl

\$T
BO
BO
BO
BQ
BQ
BT
CD
CM
CM
DE
DS
ER
ER
FU
GO
IN
IO
IO
IO
MO
MS
MS
MS
MS
MS
MS
MS
MS
MS
MS
MS
MS
MS
MS
MS
MS
OW
PQ
PR
PR
RD
RE
RI
RPI
RPI
RPI
RPI
RPI
RS
RS
S1
S4

N 11

PQ
Psc

PSI

SA
BO
BO

Ph

In
Co
Pa
Syn
Pa
Syn
Psc
Cr
As

Th
86
Th
52
24

Ma

DI
DI
SY
TO

17

Th

MA

0430 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

