B 4

```
UUU         UUU  VVV         VVV        111        RRRRRRRRRRRR        OOOOOOOOO     MMM         MMM
UUU         UUU  VVV         VVV        111        RRRRRRRRRRRR        OOOOOOOOO     MMM         MMM
UUU         UUU  VVV         VVV        111        RRRRRRRRRRRR        OOOOOOOOO     MMM         MMM
UUU         UUU  VVV         VVV     111111        RRR         RRR  OOO         OOO  MMMMMM   MMMMMM
UUU         UUU  VVV         VVV     111111        RRR         RRR  OOO         OOO  MMMMMM   MMMMMM
UUU         UUU  VVV         VVV     111111        RRR         RRR  OOO         OOO  MMMMMM   MMMMMM
UUU         UUU  VVV         VVV        111        RRR         RRR  OOO         OOO  MMM   MMM   MMM
UUU         UUU  VVV         VVV        111        RRR         RRR  OOO         OOO  MMM   MMM   MMM
UUU         UUU  VVV         VVV        111        RRR         RRR  OOO         OOO  MMM   MMM   MMM
UUU         UUU  VVV         VVV        111        RRRRRRRRRRRR     OOO         OOO  MMM         MMM
UUU         UUU  VVV         VVV        111        RRRRRRRRRRRR     OOO         OOO  MMM         MMM
UUU         UUU  VVV         VVV        111        RRRRRRRRRRRR     OOO         OOO  MMM         MMM
UUU         UUU  VVV         VVV        111        RRR   RRR        OOO         OOO  MMM         MMM
UUU         UUU  VVV         VVV        111        RRR   RRR        OOO         OOO  MMM         MMM
UUU         UUU  VVV         VVV        111        RRR   RRR        OOO         OOO  MMM         MMM
UUU         UUU     VVV   VVV           111        RRR      RRR     OOO         OOO  MMM         MMM
UUU         UUU     VVV   VVV           111        RRR      RRR     OOO         OOO  MMM         MMM
UUU         UUU     VVV   VVV           111        RRR      RRR     OOO         OOO  MMM         MMM
UUUUUUUUUUUUUUU        VVV           111111111     RRR         RRR     OOOOOOOOO     MMM         MMM
UUUUUUUUUUUUUUU        VVV           111111111     RRR         RRR     OOOOOOOOO     MMM         MMM
UUUUUUUUUUUUUUU        VVV           111111111     RRR         RRR     OOOOOOOOO     MMM         MMM
```

B00
V03-

**FILE**ID**VMBUVAX1

```
VV      VV  MM      MM  BBBBBBBB    UU      UU  VV      VV    AAAAAA    XX      XX      11
VV      VV  MM      MM  BBBBBBBB    UU      UU  VV      VV    AAAAAA    XX      XX      11
VV      VV  MMMM  MMMM  BB      BB  UU      UU  VV      VV  AA      AA  XX      XX    1111
VV      VV  MMMM  MMMM  BB      BB  UU      UU  VV      VV  AA      AA  XX      XX    1111
VV      VV  MM  MM  MM  BB      BB  UU      UU  VV      VV  AA      AA    XX  XX        11
VV      VV  MM  MM  MM  BB      BB  UU      UU  VV      VV  AA      AA    XX  XX        11
VV      VV  MM      MM  BBBBBBBB    UU      UU  VV      VV  AA      AA      XX          11
VV      VV  MM      MM  BBBBBBBB    UU      UU  VV      VV  AA      AA      XX          11
VV      VV  MM      MM  BB      BB  UU      UU  VV      VV  AAAAAAAAAA    XX  XX        11
VV      VV  MM      MM  BB      BB  UU      UU  VV      VV  AAAAAAAAAA    XX  XX        11
  VV  VV    MM      MM  BB      BB  UU      UU    VV  VV    AA      AA  XX      XX      11          ....
  VV  VV    MM      MM  BB      BB  UU      UU    VV  VV    AA      AA  XX      XX      11          ....
    VV      MM      MM  BBBBBBBB    UUUUUUUUUU      VV      AA      AA  XX      XX    111111        ....
    VV      MM      MM  BBBBBBBB    UUUUUUUUUU      VV      AA      AA  XX      XX    111111        ....

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          .title  VMB_MICROVAX_I
0000          .ident /V01.0-00/
0000  ;
0000  ;*****************************************************************************
0000  ;*                                                                           *
0000  ;*  Copyright (c) 1984                                                       *
0000  ;*  by DIGITAL Equipment Corporation, Maynard, Mass.                         *
0000  ;*                                                                           *
0000  ;*  This software is furnished under a license and may be used and  copied   *
0000  ;*  only  in  accordance  with  the  terms  of  such  license and with the   *
0000  ;*  inclusion of the above copyright notice.  This software or  any  other   *
0000  ;*  copies  thereof may not be provided or otherwise made available to any   *
0000  ;*  other person.  No title to and ownership of  the  software  is  hereby   *
0000  ;*  transferred.                                                             *
0000  ;*                                                                           *
0000  ;*  The information in this software is subject to change  without  notice   *
0000  ;*  and  should  not  be  construed  as  a commitment by DIGITAL Equipment   *
0000  ;*  Corporation.                                                             *
0000  ;*                                                                           *
0000  ;*  DIGITAL assumes no responsibility for the use or  reliability  of  its   *
0000  ;*  software on equipment which is not supplied by DIGITAL.                  *
0000  ;*                                                                           *
0000  ;*****************************************************************************
0000  ;
0000  ; Facility:
0000  ;
0000  ;       Bootstrap ROM code for the MicroVAX I.
0000  ;
0000  ; Abstract:
0000  ;
0000  ;       This module contains a hybrid bootstrap for VMS, ELN, diagnostics
0000  ;       and other systems that is a combination of the other VAX hardware
0000  ;       boot functions and VMB's traditional function. It was edited down
0000  ;       from the VMS VMB and then enhanced to perform the priority ordered
0000  ;       boot.
0000  ;
0000  ; Author: R. Heinen
0000  ;
0000  ; Date: July 1983
0000  ;--
0000
0000          $bdtdef                         ;define boot driver descriptor
0000          $bqodef                         ;define boot driver offsets
0000          $btddef                         ;define boot device types
0000          $ihddef                         ;define VMS image header
0000          $iodef                          ;define I/O function codes
0000          $ipldef                         ;ipl's
0000          $ndtdef                         ;define adapter types
0000          $prdef                          ;processor registers
0000          $pruvldef                       ;processor registers for MicroVAX I
0000          $rpbdef                         ;RPB
0000          $ssdef                          ;define VMS status codes
0000          $vmbargdef                      ;define VMB arguments
0000
0000
0000  ;
0000  ; define some new btd symbols
```

```
                0000   ;
                0000
00000008        0000           btd$k_prom = 8
00000060        0000           btd$k_qna = 96
                0000
                0000   ;***********************************************************************
                0000
                0000   ;
                0000   ; define a macro to define boot driver names etc.
                0000   ;
                0000
                0000           .macro boot_device name,h_unit,pcsr,type,rtn,?l1
                0000   l1:     .asciz  /name/
                0000           .byte   h_unit
                0000           .byte   type
                0000           .long   pcsr+phy_a_io_space
                0000           .long   rtn-l1
                0000           .endm
                0000
                0000   ;
                0000   ; define macros to aid with error message printing
                0000   ;
                0000
                0000           .macro  fatal_message   code
                0000           .if     nb,code
                0000           movzwl  #ss$_'code,r0
                0000           .endc
                0000           brw     fatal_error
                0000           .endm
                0000
                0000           .macro  msg_def mname,txt
                0000           .word   ss$_'mname
                0000           .if     nb,<txt>
                0000           .word   a_'mname-.
                0000           .save_psect
                0000           .psect  $$$$10boot,byte
                0000   last_msg = .
                0000   a_'mname:
                0000           .asciz  \txt \
                0000           .restore_psect
                0000           .iff
                0000           .word   last_msg-.
                0000           .endc
                0000           .endm
                0000
                0000   ;
                0000   ; define local data structure offsets
                0000   ;
                0000
                0000   ;
                0000   ; define boot device desc structure
                0000   ;
                0000
                0000           $defini bd
00000004        0000   bd_l_name:      .blkl   1
00000005        0004   bd_b_high_unit: .blkb   1
00000006        0005   bd_b_type:      .blkb   1
```

```
0000000A  0006  bd_a_csr:            .blkl   1
0000000E  000A  bd_a_routine:        .blkl   1
          000E  bd_s_bd:
          000E          $defend bd
          0000
          0000  ;
          0000  ; define local data constants
          0000  ;
          0000
20000000  0000  phy_a_io_space            = ^x20000000    ;physical address of I/O space
          0000
          0000  ;
          0000  ; define extents
          0000  ;
          0000
00002000  0000  k_max_memory_pages        = 8192          ;max number of pages
0000007F  0000  k_max_io_pages            = 127           ;max pages in one I/O transfer
          0000
          0000  ;
          0000  ; define addresses in 64K segment
          0000  ;
          0000
00000000  0000  k_rpb_addr                = 0
00000200  0000  k_rom_code_addr           = ^x200         ;allow for 16K
00004200  0000  k_scb_addr                = ^x4200        ;
00004600  0000  k_pfn_map_addr            = ^x4600        ;2 pages
          0000                                            ;stack area of three pages
00005000  0000  k_secondary_boot_addr     = ^x5000
          0000
          0000  ;
          0000  ; define MicroVAX I machine check codes used here
          0000  ;
          0000
00000001  0000  k_parity.error            = 1
00000002  0000  k_bus.timeout             = 2
          0000
          0000  ;
          0000  ; define scb vectors used here
          0000  ;
          0000
00000004  0000  scb_a_mcheck              = 4
00000060  0000  scb_a_write_timeout       = ^x60
0000002C  0000  scb_a_breakpoint          = ^x2c
00000028  0000  scb_a_trace_trap          = ^x28
          0000
          0000  ;
          0000  ; define bits in MicroVAX I switch pack
          0000  ;
          0000
00000006  0000  switch_v_QVSS             = 6             ; 1 if normal, 0 if QVSS
00000007  0000  switch_v_disk_boot        = 7             ; 1 if normal, 0 if disable disk search
          0000
          0000  ;
          0000  ; define MSV-11 Memory controller values
          0000  ;
          0000
20001440  0000  msv11_csr_base = ^x1440 + phy_a_io_space
```

```
00000001  0000  msv11_csr_parity_enable = 1
          0000
          0000  ;
          0000  ; define led values
          0000  ;
          0000
00000F0D  0000  led_memory_ok = ^xf0d
00000F0E  0000  led_boot_inprogress = ^xf0e
00000F0F  0000  led_transfer_control = ^xf0f
          0000
          0000  ;
          0000  ; define console halt code
          0000  ;
          0000
00000F05  0000  console_halt = ^xf05
```

```
                                     0000          .sbttl  read write data
                                     0000
                                 00000000          .psect  $$$$04boot,long
                                     0000
                                     0000    ;
                                     0000    ; strings used for file opens
                                     0000    ;
                                     0000
                                     0000    vmsfile:                                  ;Name of standard secondary
58 45 53 59 53 2E 30 53 59 53 5B 00' 0000          .ASCIC  /[SYS0.SYSEXE]SYSBOOT.EXE/ ;bootstrap image file.
58 45 2E 54 4F 4F 42 53 59 53 5D 45  000C
                                  45 0018
                                  18 0000
                                     0019
                                     0019    diagfile:                                 ;Name of standard diagnostic
41 4D 53 59 53 2E 30 53 59 53 5B 00' 0019          .ASCIC  /[SYS0.SYSMAINT]DIAGBOOT.EXE/ ;secondary bootstrap image.
54 4F 4F 42 47 41 49 44 5D 54 4E 49  0025
                         45 58 45 2E 0031
                                  1B 0019
                                     0035
                                     0035    nameprompt:                               ;Prompt string for secondary
00 3A 65 6C 69 66 74 6F 6F 42 0A 0D  0035          .ASCIZ  <13><10>/Bootfile:/     ;boot file name.
                                     0041
                                     0041    ;
                                     0041    ; define two boot device priority lists
                                     0041    ;
                                     0041
                                     0041    boot_device_list:
                                     0041
                                     0041          boot_device     DUA,3,<^X1468>,btd$k_uda,disk_boot
                                     004F
                                     004F    no_disk_boot_device_list:
                                     004F
                                     004F          boot_device     PRA,0,<^x0000>,btd$k_prom,prom_boot
                                     005D          boot_device     XQA,0,<^X1920>,btd$k_qna,network_boot
                                0000 006B          .word   0                       ;implant a zero name
                                     006D
                                     006D    ;
                                     006D    ; define text to correspond to ss$_ values.
                                     006D    ;
                                     006D
                                     006D    message_header:
52 45 2D 46 2D 54 4F 4F 42 25 0A 0D  006D          .asciz  <13><10>/%BOOT-F-ERROR, /
                   00 20 2C 52 4F 52 0079
                                     007F
                                     007F    message_base:
                                     007F
                                     007F    ;
                                     007F    ; define some ss$_ codes that are only used here
                                     007F    ;
                                     007F
                            00008000 007F    ss$_memerr = ^x8000
                            00008008 007F    ss$_scbint = ^x8008
                            00008010 007F    ss$_2ndint = ^x8010
                            00008018 007F    ss$_norom  = ^x8018
                                     007F
                                     007F    msg_def nosuchdev,<None of the bootable devices contain a program image>
```

```
              0083  msg_def devassign,<Device is not present>
              0087  msg_def nosuchfile,<Program image not found>
              008B  msg_def filestruct,<Invalid boot device file structure>
              008F          msg_def badchksum
              0093          msg_def badfilehdr
              0097          msg_def badirectory
              009B  msg_def filnotcntg,<Invalid program image format>
              009F  msg_def endoffile
              00A3  msg_def badfilename,<Invalid filename>
              00A7  msg_def bufferovf,<Program image does not fit in available memory>
              00AB  msg_def ctrlerr,<Boot device I/O error>
              00AF  msg_def devinact,<Failed to initialize boot device>
              00B3  msg_def devoffline,<Device is offline>
              00B7  msg_def memerr,<Memory initialization error>
              00BB  msg_def scbint,<Unexpected SCB exception or machine check>
              00BF  msg_def 2ndint,<Unexpected exception after starting program image>
              00C3  msg_def norom,<No valid ROM image found>
              00C7  msg_def nosuchnode,<No response from load server>
        0000  00CB          .word   0                       ;terminate list
              00CD
              00CD  ;
              00CD  ; writable data
              00CD  ;
              00CD          .ALIGN  LONG
              00D0
              00D0  ;
              00D0  ; Parameter list handed from primary boot to secondary boot
              00D0  ; The first location contains the argument count.  It is intended
              00D0  ; that the secondary boot will know what is in the list based on
              00D0  ; the argument count and the VMB version number.  This means that
              00D0  ; new information should be placed at new offsets even if older
              00D0  ; stuff becomes obsolete.  The VMB version number can be used to
              00D0  ; totally change the argument meanings if necessary.
              00D0  ;
              00D0
              00D0  second_param:
    000000D4  00D0          fil$gq_cache    == .+vmb$q_filecache ;FILEREAD cache descriptor
    000000F4  00D0          boo$gb_systemid == .+vmb$b_systemid  ;SCS system id
    0000000E  00D0          .long   <vmb$c_argbytcnt-4>/4   ;Size of argument list
              00D4          .rept   vmb$c_argbytcnt-4       ;Reserve space for the arguments
              00D4          .byte   0
          00  00D4          .endr
              010C
              010C  file_cache_desc:                        ;saved cache desc
    00000000  010C          .long   0                       ;to re-init the cache after error
    00000000  0110          .long   0
              0114
              0114  ;
              0114  ; address of the RPB as a global
              0114  ;
              0114
              0114  boo$gl_rpbbase::
    00000000  0114          .long   0
              0118
              0118  ;
              0118  ; machine check support
              0118  ;
```

```
                   0118
                   0118   machine_check_continue:              ;contains 0 or that address to
         0000011C  0118           .blkl   1                    ;transfer to after a machine check
                   011C
                   011C   ;
                   011C   ; error device name
                   011C   ;
                   011C
                   011C   boot_device_name:
         00000000  011C           .long   0
               00  0120           .byte   0
```

```
0121            .sbttl  boot code
0121    ;++
0121    ; ROM_START
0121    ;
0121    ; functional description:
0121    ;
0121    ; This code is entered after the MicroVAX I microcode has completed its
0121    ; restart/boot/halt sequence. It runs at IPL 31, in Kernel mode on the
0121    ; interrupt stack. The action is to initialize an RPB, setup a bitmap of
0121    ; useable memory pages and load the next part of the system boot based on
0121    ; the input flag settings.
0121    ;
0121    ; If the inputs include a specific boot device name that device and only
0121    ; that device is booted. On the otherhand, if no specific boot device
0121    ; is specified then a priority ordered sequence of boot devices is tried.
0121    ; (In this case R0 will be 0 or contain all blanks.
0121    ;
0121    ; As follows:
0121    ;
0121    ;       DU units 0,1,2,3
0121    ;       Other disks
0121    ;       ROM (See below for an explanation of how the ROM is found.)
0121    ;       QNA
0121    ;
0121    ;       If none of these devices provide a bootstrap then a message is
0121    ;       displayed followed by a HALT.
0121    ;
0121    ; ROM systems are recognized by the boot memory search. A ROM system must
0121    ; be aligned on a 4KB boundary and contain a foot print which is the same
0121    ; as the second part of the boot block described below.
0121    ;
0121    ; If the boot is from a mass storage device then for each valid volume
0121    ; that is found, the volume is searched as a Files-11 volume and then
0121    ; the secondary boot image is found. If the volume is not a Files-11 volume
0121    ; then block 0 of the volume is read and checked to see if it meets the
0121    ; standard for the boot block format. If not, the volume is not used and the
0121    ; next volume is tried unless a specific device was specified by the user.
0121    ;
0121    ; The boot block format is:
0121    ;
0121    ;                           +-------+-------+-------+-------+
0121    ;       BB+0:               !     1 !     n !     any value !
0121    ;                           +-------+-------+-------+-------+
0121    ;                           ! low LBN       ! High LBN      !
0121    ;                           +-------+-------+-------+-------+
0121    ;
0121    ; This second part is used for both the boot block and the ROM system.
0121    ;
0121    ;                           +-------+-------+-------+-------+
0121    ;       BB+(2*n)+0:         !  Chk  !     k !       18(Hex) !
0121    ;                           +-------+-------+-------+-------+
0121    ;                           ! any value, most likely 0      !
0121    ;                           +-------+-------+-------+-------+
0121    ;       BB+(2*n)+8:         ! size in blocks of the image   !
0121    ;                           +-------+-------+-------+-------+
0121    ;       BB+(2*n)+12:        ! load offset                   !
0121    ;                           +-------+-------+-------+-------+
```

```
0121 ;       BB+(2*n)+16:    | offset into image to start    |
0121 ;                       +-------+-------+-------+-------+
0121 ;       BB+(2*n)+20:    | sum of the previous three LW's|
0121 ;                       +-------+-------+-------+-------+
0121 ;
0121 ; The input bits in R5 can contain a bit that disables the Files-11 search.
0121 ;
0121 ; If a Files-11 boot is done then the file booted is either:
0121 ;
0121 ;       SYSBOOT.EXE - default
0121 ;       DIAGBOOT.EXE - R5 bit setting
0121 ;       solicited from the console
0121 ;
0121 ; Details of how the memory look and the register settings when the
0121 ; secondary bootstraps are entered are documented where the exits occur.
0121 ;
0121 ; inputs:
0121 ;
0121 ;       r0 = boot device name in ASCII or 0 if none specified
0121 ;       r1 = switch pack settings 1 is "ON", 0 is "OFF"
0121 ;
0121 ;               Bit     Meaning
0121 ;               ---     -------
0121 ;
0121 ;               7       Enable disk search during bootstrap
0121 ;
0121 ;               6       1 if VT100/VT200 console, 0 if QVSS video option
0121 ;
0121 ;               4-5     Halt action
0121 ;               3       Console Break enabled
0121 ;
0121 ;               2       Reserved
0121 ;
0121 ;               0-1     Console baud rate
0121 ;
0121 ;       R5 = software boot control flags from the /N boot command qualifier.
0121 ;
0121 ;       The following bits are used by this boot ROM code:
0121 ;
0121 ;               Bit     Meaning
0121 ;               ---     -------
0121 ;
0121 ;                3      RPB$V_BBLOCK.
0121 ;                       If set, the attempt to Files-11 boot is skipped
0121 ;                       and only the boot block type boot is done.
0121 ;
0121 ;                4      RPB$V_DIAG.
0121 ;                       Diagnostic boot. Secondary bootstrap is image
0121 ;                       called [SYSMAINT]DIAGBOOT.EXE.
0121 ;
0121 ;                6      RPB$V_HEADER.
0121 ;                       Image header. Takes the transfer address of the
0121 ;                       secondary bootstrap image from that file's
0121 ;                       image header. If RPB$V_HEADER is not set,
0121 ;                       transfers control to the first byte of the
0121 ;                       secondary boot file.
0121 ;
```

```
0121  ;         8        RPB$V_SOLICT.
0121  ;                  File name. Prompt for the name of a
0121  ;                  secondary bootstrap file.
0121  ;
0121  ;         9        RPB$V_HALT.
0121  ;                  Halt before transfer. Executes a HALT
0121  ;                  instruction before transferring control to the
0121  ;                  secondary bootstrap.
0121  ;
0121  ;         <31:28>  RPB$V_TOPSYS
0121  ;                  Specifies the top level directory number for system
0121  ;                  disks with multiple systems
0121  ;
0121  ;    The following bits are NOT used by this boot ROM code:
0121  ;
0121  ;         Bit      Meaning
0121  ;         ---      -------
0121  ;
0121  ;         0        RPB$V_CONV.
0121  ;                  Conversational boot. At various points in the
0121  ;                  system boot procedure, the bootstrap code
0121  ;                  solicits parameters and other input from the
0121  ;                  console terminal. If the DIAG is also on, then
0121  ;                  the diagnostic supervisor should enter "MENU"
0121  ;                  mode and prompt user for devices to test.
0121  ;
0121  ;         1        RPB$V_DEBUG.
0121  ;                  Debug. If this flag is set, VMS maps the code
0121  ;                  for the XDELTA debugger into the system page
0121  ;                  tables of the running system.
0121  ;
0121  ;         2        RPB$V_INIBPT.
0121  ;                  Initial breakpoint. If RPB$V_DEBUG is set, VMS
0121  ;                  executes a BPT instruction immediately after
0121  ;                  enabling mapping.
0121  ;
0121  ;         5        RPB$V_BOOBPT.
0121  ;                  Bootstrap breakpoint. Stops the primary
0121  ;                  and secondary bootstraps with a breakpoint
0121  ;                  instruction before testing memory.
0121  ;
0121  ;         7        RPB$V_NOTEST.
0121  ;                  Memory test inhibit. Sets a bit in the PFN bit
0121  ;                  map for each page of memory present. Does not
0121  ;                  test the memory.
0121  ;
0121  ;         10       RPB$V_NOPFND.
0121  ;                  No PFN deletion (not implemented; intended to
0121  ;                  tell VMB not to read a file from the boot device
0121  ;                  that identifies bad or reserved memory pages,
0121  ;                  so that VMB does not mark these pages as valid
0121  ;                  in the PFN bitmap).
0121  ;
0121  ;         11       RPB$V_MPM.
0121  ;                  Specifies that multi-port memory is to be used
0121  ;                  for the total exec memory requirement.  No local
0121  ;                  memory is to be used.  This is for tightly-coupled
```

```
0121 ;                       multi-processing.  If the DIAG is also on, then
0121 ;                       the diagnostic supervisor enters "AUTOTEST" mode.
0121 ;
0121 ;              12       RPB$V_USEMPM.
0121 ;                       Specifies that multi-port memory should be used in
0121 ;                       addition to local memory, as though both were one
0121 ;                       single pool of pages.
0121 ;
0121 ;              13       RPB$V_MEMTEST
0121 ;                       Specifies that a more extensive algorithm be used
0121 ;                       when testing main memory for hardware uncorrectable
0121 ;                       (RDS) errors.
0121 ;
0121 ;              14       RPB$V_FINDMEM
0121 ;                       Requests use of MA780 memory if MS780 is insufficient
0121 ;                       for booting.  Used for 11/782 installations.
0121 ;
0121 ;      r10 = original PC
0121 ;      r11 = original PSL
0121 ;      AP = halt code
0121 ;      SP = address of 64K memory block + 200 hex
0121 ;
0121 ; implicit inputs:
0121 ;
0121 ;      IPL is 31, interrupt stack.
0121 ;      The first instruction of this code is at SP.
0121 ;
0121 ;      All of the system's memory controllers have been
0121 ;      initialized to have parity error detect ON. This
0121 ;      means that the 64K memory block that contains this
0121 ;      code has correct parity. The cache is enabled and
0121 ;      will continue to be enabled throughout.
0121 ;
0121 ; When the secondary bootstrap code gains control memory will look like:
0121 ;
0121 ; 0    +---------------------------------------+
0121 ;      + RPB                                   +
0121 ; 200  +---------------------------------------+
0121 ;      + 8K of Boot Code                       +
0121 ;      + boot driver preamble starts at 200    +
0121 ; 4200 +---------------------------------------+ (PR$_SCBB value)
0121 ;      + 2 Pages of SCB                        +
0121 ; 4600 +---------------------------------------+
0121 ;      + 2 Pages of PFN Bit Map described by   +
0121 ;      + RPB fields                            +
0121 ; 4A00 +---------------------------------------+
0121 ;      + available for stack (3 Pages)         +
0121 ; 5000 +---------------------------------------+
0121 ;      + Secondary boot code image             +
0121 ;      .
0121 ;      .
0121 ;      .
0121 ;
0121 ; The register contents when control is passed to the secondary
0121 ; bootstrap are:
0121 ;
0121 ;      R11 = base address of RPB
```

```
                        0121  ;       AP = address of the secondary boot parameter block alla VMB
                        0121  ;       SP = current stack pointer
                        0121  ;       PR$_SCBB = SCB address
                        0121  ;--
                        0121
                        0121  ;
                        0121  ; this will be the first location in the boot ROM
                        0121  ;
                        0121
                    00000000          .psect  $$$$00boot,long
                        0000  ROM_BASE:
           011E'  31    0000          brw     rom_start                ;transfer control to actual code
  83 82 81 80 06        0003          .byte   6,^x80,^x81,^x82,^x83   ;footprint
                        0008
                    00000121          .psect  $$$$04boot,long          ;
                        0121
                        0121  ROM_START:
                        0121          .default        displacement,word
```

```
                               0121          .sbttl  SCB initialization and XDELTA breakpoint
                               0121
              26   0F    DA    0121          mtpr    #^xf,#pr$_mcesr          ;reset any machine checks
                               0124
                               0124  ;
                               0124  ; setup SCB for the duration of this execution
                               0124  ;
                               0124
        57   4400 CE    9E     0124          movab   k_scb_addr+1024-k_rom_code_addr(sp),r7
                               0129                                           ;address scb plus two pages
          59    FF 8F   9A     0129          movzbl  #255,r9                  ;setloop count DIV 2
        77   06D9'CF    DE     012D  10$:    moval   unfielded_scb_int+1,-(r7) ;address general error routine
                F8 59   F4     0132          sobgeq  r9,10$                   ;continue in loop
                E0 AF   D4     0135          clrl    machine_check_continue ;init machine check continue address
             0615'CF    9E     0138          movab   machine_check_detect+1,- ;init machine check vector
                04 A7          013C                  scb_a_mcheck(r7)         ;
             0735'CF    9E     013E          movab   write_timeout_int+1,-  ;init write timeout vector
                60 A7          0142                  scb_a_write_timeout(r7);
              11   57    DA    0144          mtpr    r7,#pr$_scbb             ;insert scb address in PR
                               0147
                               0147  ;
                               0147  ; Read the system identification processor register to discover which
                               0147  ; kind of VAX is to be booted.
                               0147  ;
                               0147
              58   3E    DB    0147          mfpr    #pr$_sid,r8              ;Read the CPU identification
                               014A                                           ;processor register.
     58   58    E8 8F   78     014A          ashl    #-pr$v_sid_type,r8,r8    ;Get CPU identification code.
      0000'CF   58       90    014F          movb    r8,exe$gb_cputype        ;Save processor code globally
                               0154                                           ;in boot driver desc table
                               0154
                               0154  ;
                               0154  ; If the DEBUG flag is defined (meaning that XDELTA has been linked
                               0154  ; with this primary bootstrap), set up 2 XDELTA handlers in the SCB --
                               0154  ; one for breakpoints and one for tbit traps. Then initialize the
                               0154  ; XDELTA breakpoint table, allocate 3 pages of stack, and, if requested,
                               0154  ; execute a breakpoint before proceeding with the bootstrap.
                               0154  ;
                               0154
                               0154          .if     df,debug
                               0154          movab   xdt$breakpoint+1,scb_a_breakpoint(r7) ;Set up BPT handler.
                               0154          movab   xdt$trace_trap+1,scb_a_trace_trap(r7) ;Set up TBIT handler.
                               0154          movab   ini$brk,xdt$initial_break ;Store the initial breakpoint.
                               0154          movl    sp,r6                    ;Save current top of stack.
                               0154          movab   k_secondary_boot_addr-k_rom_code_addr(sp),sp ;Address a stack,
                               0154                                           ;add 1 page for the SCB
                               0154          bbc     #rpb$v_boobpt,r5,nobrk   ;If no BPT was requested in the
                               0154                                           ;boot flags, just proceed.
                               0154
                               0154  ;
                               0154  ; Initial breakpoint.
                               0154  ;
                               0154  ; Current register status is as follows:
                               0154  ;
                               0154  ;       R0-R5   - initial input values
                               0154  ;       R6      - SP value at start of ROM code
                               0154  ;       R7      - address of the SCB
```

```
0154  ;         R8      - processor identification code
0154  ;         R9      - destroyed
0154  ;         R10-FP  - initial input values
0154  ;         SP      - address of a 3-page stack
0154  ;
0154  ; Code following the breakpoint is going to restore SP to its original
0154  ; value. If you want to modify SP in XDELTA, modify R6 instead.
0154  ;
0154
0154  ini$brk::                                    ;Debugging breakpoint.
0154
0154          bpt                                  ;Stop in XDELTA.
0154
0154  NOBRK:                                       ;Proceed with bootstrapping.
0154          movl    r6,sp                        ;restore stack pointer
0154          .ENDC
```

```
                        0154         .sbttl  rpb initialization
                        0154    ;
                        0154    ; initialize and address the RPB
                        0154    ;
                        0154
      56   FE00 CE  9E  0154         movab   k_rpb_addr-k_rom_code_addr(sp),r6 ;address rpb with temp reg
        1C A6   50  7D  0159         movq    r0,rpb$l_bootr0(r6)       ;save registers
        24 A6   52  7D  015D         movq    r2,rpb$l_bootr2(r6)       ;
        2C A6   54  7D  0161         movq    r4,rpb$l_bootr4(r6)       ;
        10 A6   5A  7D  0165         movq    r10,rpb$l_haltpc(r6)      ;save halt PC and PSL
           66   56  D0  0169         movl    r6,rpb$l_base(r6)         ;address of RPB
        A4 AF   56  D0  016C         movl    r6,boo$gl_rpbbase         ;also globally
        18 A6   5C  D0  0170         movl    ap,rpb$l_haltcode(r6)     ;save halt code
           04 A6    D4  0174         clrl    rpb$l_restart(r6)         ;init header fields
           0C A6    D4  0177         clrl    rpb$l_rstrtflg(r6)        ;
        08 A6   01  CE  017A         mnegl   #1,rpb$l_chksum(r6)       ;
   34 A6   0000'CF  9E  017E         movab   boo$al_vector,rpb$l_iovec(r6) ;insert address of driver
   00   6E      00  2C  0184         movc5   #0,(sp),#0,-              ;init remainder of RPB
   38 A6   00D0 8F      0188                 #rpb$c_length-rpb$l_iovecsz,rpb$l_iovecsz(r6)
           5B   56  D0  018D         movl    r6,r11                    ;set future RPB address
      00B0 CB   57  D0  0190         movl    r7,rpb$l_scbb(r11)        ;save scbb address in RPB
      0090 CB   28  90  0195         movb    #ndt$_ub0,rpb$b_confreg(r11) ;one Qbus on Micro-VAX I.
      00A1 CB   28  90  019A         movb    #ndt$_ub0,rpb$b_bootndt(r11) ;Pretend this is UNIBUS.
                        019F
                        019F    ;
                        019F    ; init the secondary bootstrap parameter block
                        019F    ;
                        019F
      5C   FF2D CF  9E  019F         movab   second_param,ap           ;load its base address
        0C AC   01  CE  01A4         mnegl   #1,vmb$l_lo_pfn(ap)       ;set pfn data
        10 AC   01  CE  01A8         mnegl   #1,vmb$l_hi_pfn(ap)       ;set pfn data
                        01AC
                        01AC    ;
                        01AC    ; address larger stack and setup free memory pointer
                        01AC    ;
                        01AC
      5E   5000 CB  9E  01AC    15$: movab   k_secondary_boot_addr(r11),sp; address target for I/O
                        01B1                                           ;and create a three page stack
           5A   5E  D0  01B1         movl    sp,r10                    ;copy to address of free memory
```

```
                                  01B4         .sbttl  memory initialization
                                  01B4  ;
                                  01B4  ; initialize parity memory
                                  01B4  ;
                                  01B4  ; allocate and init RPB PFN bit map
                                  01B4  ;
                                  01B4
           44 AB    01    0A  78  01B4         ashl    #10,#1,rpb$q_pfnmap(r11) ;set size to 1024 bytes
           48 AB  4600    CB  9E  01B9         movab   k_pfn_map_addr(r11),rpb$q_pfnmap+4(r11) ;set up pfn desc addr
4600 CB    44 AB    00    6E  00  2C  01BF     movc5   #0,(sp),#0,rpb$q_pfnmap(r11),k_pfn_map_addr(r11) ;init to zeros
          FF4A CF    E4'AF    9E  01C8         movab   b^5$,machine_check_continue ;setup continue address
           59    00001FFF 8F  D0  01CE         movl    #k_max_memory_pages-1,r9 ;set page number of last memory page
           07 20 AB    06     E0  01D5         bbs     #switch_v_qvss,rpb$l_bootr1(r11),3$;if set console not QVSS
           59    00000200 8F  C2  01DA         subl    #512,r9                     ;else - last 256K used for QVSS video
                                  01E1  3$:    setipl  #^x1d-1                     ;lower IPL to allow write timeout
                                  01E4
                                  01E4  ;
                                  01E4  ; sweep all of memory to set parity and establish bit map
                                  01E4  ;
                                  01E4
        FF2D CF    029F'CF    9E  01E4  5$:    movab   nxm_memory,machine_check_continue ;enable machine check
                          56  7C  01EB         clrq    r6                          ;set first physical address
                                  01ED                                             ;zero page count
           58     00BC    CB  9E  01ED         movab   rpb$l_memdsc(r11),r8        ;address first memory descriptor
                          68  D4  01F2         clrl    (r8)                        ;init page count field
           04 A8    01        CE  01F4         mnegl   #1,4(r8)                    ;set very low PFN
                                  01F8
                                  01F8  ;
                                  01F8  ; write 0's to a page
                                  01F8  ;
                                  01F8
                                  01F8  page_boundary:
                                  01F8
           56    57    09     78  01F8         ashl    #9,r7,r6                    ;compute page address
                                  01FC
                                  01FC  ;
                                  01FC  ; don't write 0's in the 64k where this code is
                                  01FC  ;
                                  01FC
              56    5B        D1  01FC         cmpl    r11,r6                      ;compare addresses to find out
                    42        12  01FF         bneq    20$                         ;br if not in the 64KB of good memory
              0C AC           D5  0201         tstl    vmb$l_lo_pfn(ap)            ;low pfn inited?
                    04        18  0204         bgeq    10$                         ;br if yes
           0C AC    57        D0  0206         movl    r7,vmb$l_lo_pfn(ap)         ;insert base of area as lowest PFN
              04 A8           D5  020A  10$:   tstl    4(r8)                       ;memory desc PFN set yet?
                    04        18  020D         bgeq    15$                         ;br if yes
           04 A8    57        D0  020F         movl    r7,4(r8)                    ;insert this low PFN
              50    01        CE  0213  15$:   mnegl   #1,r0                       ;set all bits in a register
   4600 CB    20    57    50  F0  0216         insv    r0,r7,#32,k_pfn_map_addr(r11) ;enable 128 pages
   4604 CB    20    57    50  F0  021D         insv    r0,r7,#32,k_pfn_map_addr+4(r11) ;of good memory
   4608 CB    20    57    50  F0  0224         insv    r0,r7,#32,k_pfn_map_addr+8(r11) ;
   460C CB    20    57    50  F0  022B         insv    r0,r7,#32,k_pfn_map_addr+12(r11) ;
           50    7F    8F     9A  0232         movzbl  #127,r0                     ;load page count in r0
              57    6740      9E  0236         movab   (r7)[r0],r7                 ;adjust PFN to last page tested
           4C AB    50        C0  023A         addl    r0,rpb$l_pfncnt(r11)        ;adjust good page count < 8000
              68    50        C0  023E         addl    r0,(r8)                     ;adjust desc size
                    3D        11  0241         brb     40$                         ;continue in common
```

```
                              0243
                              0243   ;
                              0243   ; write and then read a single memory page
                              0243   ;
                              0243
                              0243   ;
                              0243   ; loop to check for correct parity and verify contents
                              0243   ;
                              0243
                    66   7C   0243   20$:    clrq    (r6)                        ;write memory to zero
             50     02   D0   0245           movl    #k_bus.timeout,r0           ;initialize for error exit
             51     86   7D   0248           movq    (r6)+,r1                    ;read memory back for parity detect
                              024B                                               ;first read checks for page present
                              024B                                               ;skip out if not present to nxm_memory
                              024B                                               ;skip out on parity error
                    52   12   024B           bneq    nxm_memory                  ;if eql then correct read back
                              024D                                               ;if neq then odd case of PROM
                              024D
                              024D   ;
                              024D   ; check for page cross
                              024D   ;
                              024D
         56    01FF 8F   B3   024D           bitw    #^x1ff,r6                   ;page cross?
                    EF   12   0252           bneq    20$                         ;br if no, keep going
                              0254
                              0254   ;
                              0254   ; try to write a non-zero value and verify that it can be done
                              0254   ;
                              0254   ; This is to detect pages in ROM's that are all zeros
                              0254   ;
                              0254
     0FFFFFFC E6    3F   B0   0254           movw    #63,<1@28>-4(r6)            ;write memory, page is present
                              025B                                               ;with don't cache bit
     0FFFFFFC E6    3F   B1   025B           cmpw    #63,<1@28>-4(r6)            ;read back correct?
                    3B   12   0262           bneq    nxm_memory                  ;if neq then some non RAM memory
                 FC A6   B4   0264           clrw    -4(r6)                      ;reset to zero
                              0267
                              0267   ;
                              0267   ; page is written - parity appears correct
                              0267   ;
                              0267
4600 CB   01   57   01   F0   0267   30$:    insv    #1,r7,#1,k_pfn_map_addr(r11) ;insert bit in PFN map
                 0C AC   D5   026E           tstl    vmb$l_lo_pfn(ap)            ;low pfn inited?
                    04   18   0271           bgeq    35$                         ;br if yes
            0C AC   57   D0   0273           movl    r7,vmb$l_lo_pfn(ap)         ;insert lowest PFN
                 04 A8   D5   0277   35$:    tstl    4(r8)                       ;memory desc PFN set yet?
                    04   18   027A           bgeq    40$                         ;br if yes
            04 A8   57   D0   027C           movl    r7,4(r8)                    ;insert this low PFN
            10 AC   57   D0   0280   40$:    movl    r7,vmb$l_hi_pfn(ap)         ;insert highest
                    68   D6   0284           incl    (r8)                        ;count in current memory desc
                 4C AB   D6   0286           incl    rpb$l_pfncnt(r11)           ;count as good page
                              0289
                              0289   ;
                              0289   ; come here to move to next page
                              0289   ;
                              0289
                              0289   next_page:
```

```
                              0289
  FF69 57   01   59   F1      0289           acbl    r9,#1,r7,page_boundary  ; continue until end of memory
                              028F
                              028F      ;
                              028F      ; restore IPL and setup SCB for booting
                              028F      ;
                              028F
                              028F           setipl  #ipl$_power             ;reset IPL
           FE82 CF   D4       0292           clrl    machine_check_continue  ;reset machine check continue addr
  23   00000F0D 8F   DA       0296           mtpr    #Led_memory_ok,#pr$_txdb; set lights
                39   11       029D           brb     begin_boot
                              029F
                              029F      ;
                              029F      ; come here when a page does not exist
                              029F      ;
                              029F
                              029F      nxm_memory:
                              029F
           50   01   D1       029F           cmpl    #k_parity.error,r0      ;expected error?
                12   12       02A2           bneq    20$                     ;parity is ok
                              02A4
                              02A4      ;
                              02A4      ; reset the memory controllers to clear parity error
                              02A4      ;
                              02A4
           51   0F   D0       02A4           movl    #15,r1                  ;set loop count of controllers
  52   20001440 8F   D0       02A7           movl    #msv11_csr_base,r2      ;address base of controlller CSR's
           82   01   B0       02AE      10$: movw    #msv11_csr_parity_enable,(r2)+;blast all possible CSR's
             FA 51   F4       02B1           sobgeq  r1,10$                  ;continue until done
                D3   11       02B4           brb     next_page               ;no check for PROM needed on parity
                                                                             ;problem
                              02B6
                              02B6      ;
                              02B6      ; bus timeout means non existant memory on a read, the page is not present
                              02B6      ;
                              02B6      ; But, page could be PROM memory
                              02B6      ;
                              02B6
           50   02   D1       02B6      20$: cmpl    #k_bus.timeout,r0       ;expected error?
                15   12       02B9           bneq    fatal_memory_error      ;br if unexpected
                              02BB
                              02BB      ;
                              02BB      ; record holes in memory via descriptors
                              02BB      ;
                              02BB
                68   D5       02BB           tstl    (r8)                    ;this desc in use?
                CA   13       02BD           beql    next_page               ;br if no, don't move to next
      50   00FC CB   9E       02BF           movab   <rpb$c_nmemdsc*rpb$c_memdscsiz>+rpb$l_memdsc(r11),r0
           58   50   D1       02C4           cmpl    r0,r8                   ;overrun area?
                C0   12       02C7           bneq    next_page               ;br if yes
           58   08   C0       02C9           addl    #rpb$c_memdscsiz,r8     ;address next memory desc
                68   D4       02CC           clrl    (r8)                    ;set count to zero
                B9   11       02CE           brb     next_page               ;
                              02D0
                              02D0      ;
                              02D0      ; memory initialization error
                              02D0      ;
                              02D0
```

```
                02D0   839 fatal_memory_error:
                02D0   840
                02D0   841         fatal_message   memerr
```

```
                        02D8  ;++
                        02D8  ; begin_boot - start the booting process
                        02D8  ;
                        02D8  ; functional description:
                        02D8  ;
                        02D8  ; This sequence is entered after the RPB and PFN bitmap are set up.
                        02D8  ;
                        02D8  ; The process of selecting a boot device and type of boot operation starts
                        02D8  ; here.
                        02D8  ;
                        02D8  ; inputs:
                        02D8  ;
                        02D8  ;       r11 = address of the RPB
                        02D8  ;       ap = address of the secondary parameter block
                        02D8  ;
                        02D8  ;--
                        02D8
                        02D8  begin_boot:
                        02D8
                        02D8  ;
                        02D8  ; If the "solicit for secondary bootstrap file" flag is not set,
                        02D8  ; just use a predefined file specification.
                        02D8  ;
                        02D8
          03      E0    02D8          bbs     #rpb$v_bblock,-         ;br if not files-11 boot
       30 AB            02DA                  rpb$l_bootr5(r11),-     ;
          43            02DC                  25$                     ;
          08      E1    02DD          bbc     #rpb$v_solict,-         ;If "solicit" flag is not
       30 AB            02DF                  rpb$l_bootr5(r11),-     ;set, just use a default file
          13            02E1                  10$                     ;specification.
                        02E2
                        02E2  ;
                        02E2  ; To solicit a file name, call a device-independent subroutine that
                        02E2  ; writes a prompt string to the console terminal, and then reads the
                        02E2  ; user typed file name. All device specifications are ignored.
                        02E2  ;
                        02E2
       20 AB      DD    02E2          pushl   rpb$l_bootr1(r11)       ;Pass options switch settings
       68 AB      9F    02E5          pushab  rpb$t_file(r11)         ;Set address of input buffer.
          27      DD    02E8          pushl   #39                     ;Set maximum character count.
     FD47 CF      9F    02EA          pushab  nameprompt              ;Set address of prompt string.
 0000'CF  04      FB    02EE          calls   #4,boo$readprompt       ;Prompt and read string.
          2B      11    02F3          brb     25$                     ;Go try to read the file.
                        02F5
                        02F5  ;
                        02F5  ; If the solicit boot flag was not set, use a default file name string.
                        02F5  ; Usually, this file name is [SYSEXE]SYSBOOT.EXE. However, if the
                        02F5  ; diagnostic boot flag is set, the file name is [SYSMAINT]DIAGBOOT.EXE.
                        02F5  ;
                        02F5
 57  FD07 CF      9E    02F5  10$:    movab   vmsfile,r7              ;Assume SYSBOOT.EXE.
          04      E1    02FA          bbc     #rpb$v_diag,-           ;If the diagnostic flag is not
       30 AB            02FC                  rpb$l_bootr5(r11),-     ;set, SYSBOOT is correct.
          05            02FE                  15$
 57  FD16 CF      9E    02FF          movab   diagfile,r7             ;Otherwise, use predefined
                        0304                                          ;name of diagnostic boot.
                        0304
```

```
                          0304   900 ;
                          0304   901 ; Copy the file name to the RPB.
                          0304   902 ;
                          0304   903
          50    67    9A  0304   904 15$:    movzbl  (r7),r0                     ;Size of name string
                50    D6  0307   905         incl    r0                          ;Include the byte count character
    68 AB       67    50    28  0309   906         movc3   r0,(r7),rpb$t_file(r11)     ;Move name into RPB
          04    1C    EF  030E   907         extzv   #rpb$v_topsys,#rpb$s_topsys,-
      50    30  AB        0311   908                 rpb$l_bootr5(r11),r0        ;Value of 0-F means top level
                          0314   909                                             ;system directory "SYS0" - "SYSF"
          09    50    D1  0314   910         cmpl    r0,#9                       ;0 - 9 ?
                03    15  0317   911         bleq    20$                         ;Branch if yes
          50    07    C0  0319   912         addl    #<<^A/A/>-<^A/9/>-1>,r0     ;Add bias to make A - F
       6D AB    50    80  031C   913 20$:    addb    r0,rpb$t_file+5(r11)        ;Form "SYSn"
                          0320   914
                          0320   915 ;
                          0320   916 ; extract and stabilize device name info
                          0320   917 ;
                          0320   918
          57  FD1D CF   9E  0320   919 25$:    movab   boot_device_list,r7         ;address descriptor list
  7E    1C AB   80A0A0A0 8F CB  0325   920         bicl3   #^x80A0A0A0,rpb$l_bootr0(r11),-(sp) ;make name uppercase
                          032E   921                                             ;remove possible parity bit
          20    6E    91  032E   922         cmpb    (sp),#^a/ /                 ;special non-name?
                0E    14  0331   923         bgtr    35$                         ;br if gtr then specific device
                          0333   924
                          0333   925 ;
                          0333   926 ; non-specific device name
                          0333   927 ;
                          0333   928
                6E    D4  0333   929         clrl    (sp)                        ;specify non name
                07    E0  0335   930         bbs     #switch_v_disk_boot,-       ;br if entire list is to be searched
          10 20 AB        0337   931                 rpb$l_bootr1(r11),40$       ;
          57  FD11 CF   9E  033A   932         movab   no_disk_boot_device_list,r7;address alternate descriptor list
                09    11  033F   933         brb     40$                         ;continue
                          0341   934
                          0341   935 ;
                          0341   936 ; specific device name
                          0341   937 ;
                          0341   938
      FDD6 CF   6E    D0  0341   939 35$:    movl    (sp),boot_device_name       ;save specified name
         03 AE  30    82  0346   940         subb    #^a/0/,3(sp)                ;reduce unit number
                          034A   941
                          034A   942 ;
                          034A   943 ; start with first entry in boot device list and try each one until a
                          034A   944 ; boot occurs or the list is empty
                          034A   945 ;
                          034A   946
         67 AB        94  034A   947 40$:    clrb    rpb$b_slave(r11)            ;no slave or
         64 AB        B4  034D   948         clrw    rpb$w_unit(r11)             ;unit info
      5A  5000 CB     9E  0350   949         movab   k_secondary_boot_addr(r11),r10 ;set nominal load address
   66 AB    05 A7     90  0355   950         movb    bd_b_type(r7),rpb$b_devtyp(r11) ;load device type
                6E    95  035A   951         tstb    (sp)                        ;special non-name?
                18    13  035C   952         beql    45$                         ;br if yes, no specific device
    67    6E    18    00    ED  035E   953         cmpzv   #0,#24,(sp),bd_l_name(r7); compare three characters for equal
                2C    12  0363   954         bneq    50$                         ;br if no match
    04 A7    03 AE    91  0365   955         cmpb    3(sp),bd_b_high_unit(r7) ;unit in range?
                25    1A  036A   956         bgtru   50$                         ;br if no
```

```
      04 A7    03 AE   90  036C   957          movb    3(sp),bd_b_high_unit(r7);boot specific unit only
      64 AB    03 AE   9B  0371   958          movzbw  3(sp),rpb$w_unit(r11)   ;
       FDA1 CF    67   D0  0376   959 45$:     movl    bd_l_name(r7),boot_device_name;build error device name
FD9D CF   30   64 AB   81  037B   960          addb3   rpb$w_unit(r11),#^a/0/,boot_device_name+3;
         0A B747       16  0382   961          jsb     @bd_a_routine(r7)[r7]   ;use device specific routine
            2B 50      E8  0386   962          blbs    r0,100$                 ;br if success, transfer control
      24 50    01      E1  0389   963          bbc     #1,r0,55$               ;severe or fatal error?
                           038D   964                                          ;br if fatal error
               6E      95  038D   965          tstb    (sp)                    ;special non-name?
               20      12  038F   966          bneq    55$                     ;br if specific device
         57    0E      C0  0391   967 50$:     addl2   #bd_s_bd,r7             ;try next device
   23    00000F0D 8F   DA  0394   968          mtpr    #led_memory_ok,#pr$_txdb; tell operator
               67      B5  039B   969          tstw    bd_l_name(r7)           ;is there another?
               AB      12  039D   970          bneq    40$                     ;continue if not end of list
                           039F   971
                           039F   972 ;
                           039F   973 ; device data base search done without a match or valid boot device
                           039F   974 ;
                           039F   975
      50    0848 8F    3C  039F   976          movzwl  #ss$_devassign,r0       ;list end, specific name error
               6E      95  03A4   977          tstb    (sp)                    ;special non-name?
               09      12  03A6   978          bneq    55$                     ;br if specific device
      50    0908 8F    3C  03A8   979          movzwl  #ss$_nosuchdev,r0       ;list end, generic error
         FD6B CF       D4  03AD   980          clrl    boot_device_name        ;no specific name
                           03B1   981 55$:     fatal_message                   ;issue error in r0
                           03B4   982
                           03B4   983 ;
                           03B4   984 ; secondary image in place, transfer control to it
                           03B4   985 ;
                           03B4   986
      5E    5000 CB    9E  03B4   987 100$:    movab   k_secondary_boot_addr(r11),sp ;load fresh sp
                           03B9   988
                           03B9   989 ;
                           03B9   990 ; restore SCBB values
                           03B9   991 ;
                           03B9   992
   23    00000F0F 8F   DA  03B9   993          mtpr    #led_transfer_control,#pr$_txdb; tell user
      57    4600 CB    9E  03C0   994          movab   k_scb_addr+1024(r11),r7 ;address scb plus two pages
         59    FF 8F   9A  03C5   995          movzbl  #255,r9                 ;setloop count DIV 2
      77    06E1'CF    DE  03C9   996 110$:    moval   secondary_scb_int+1,-(r7) ;address general error routine
            F8 59      F4  03CE   997          sobgeq  r9,110$                 ;continue in loop
                           03D1   998
                           03D1   999 ;
                           03D1  1000 ; recompute size of bitmap
                           03D1  1001 ;
                           03D1  1002
   50    10 AC    FD 8F    78  03D1  1003          ashl    #-3,vmb$l_hi_pfn(ap),r0 ;get last valid PFN
     44 AB    50    01     C1  03D7  1004          addl3   #1,r0,rpb$q_pfnmap(r11) ;set size of map in bytes
      14 AC    44 AB       7D  03DC  1005          movq    rpb$q_pfnmap(r11),vmb$q_pfnmap(ap);copy pfnmap desc
                           03E1  1006
                           03E1  1007 ;
                           03E1  1008 ; halt system prior to entering secondary boot if requested
                           03E1  1009 ;
                           03E1  1010
               09      E1  03E1  1011          bbc     #rpb$v_halt,-           ;If boot flags don't call for
            30 AB          03E3  1012                  rpb$l_bootr5(r11),-     ;halt, just transfer to new
               07          03E5  1013                  120$                    ;bootstrap image.
```

```
      23    00000F05 8F    DA  03E6  1014          mtpr    #console_halt,#pr$_txdb ;Otherwise, HALT.
                     65    17  03ED  1015 120$:    jmp     (r5)                    ;Execute JUMP.
```

```
                                03EF          .sbttl  specific device boot subroutines
                                03EF  ;++
                                03EF  ; prom_boot
                                03EF  ;
                                03EF  ; functional description:
                                03EF  ;
                                03EF  ; This routine tries to boot from a PROM system image that may be in
                                03EF  ; memory. Each bad page 16KB boundary is tested to see if it is readable.
                                03EF  ;
                                03EF  ; inputs:
                                03EF  ;
                                03EF  ;       r7 = address of the internal boot device description
                                03EF  ;       r10 = address of the secondary boot's memory
                                03EF  ;       r11 = RPB address
                                03EF  ;       ap = address of the secondary parameter block
                                03EF  ;
                                03EF  ; outputs:
                                03EF  ;
                                03EF  ;       r0 = ss$_norom - no rom present, severe error
                                03EF  ;
                                03EF  ;       or
                                03EF  ;
                                03EF  ;       r0 = 1 if success
                                03EF  ;       r5 = transfer address
                                03EF  ;
                                03EF  ;       r7,r11 are preserved
                                03EF  ;--
                                03EF
                                03EF  prom_boot:
                                03EF
                                03EF  ;
                                03EF  ; cycle up through memory
                                03EF  ;
                                03EF
FD23 CF    15'AF    9E  03EF          movab   b^20$,machine_check_continue;implant for read timeout
             53    D4  03F5          clrl    r3                      ;initial page address
18 4600 CB   53    E0  03F7  10$:    bbs     r3,k_pfn_map_addr(r11),20$ ;br if that boundary is not bad
   51   53   09    78  03FD          ashl    #9,r3,r1                ;compute address
        18   61    B1  0401          cmpw    (r1),#^x18              ;try to read that memory
                       0404                                          ;may machine check
             0F    12  0404          bneq    20$                     ;br if not key
           02DF    30  0406          bsbw    verify_boot_block       ;verify the boot block
          09 50    E9  0409          blbc    r0,20$                  ;br if not correct
                       040C
                       040C  ;
                       040C  ; PROM found, boot from it
                       040C  ;
                       040C
55    10 A1  51    C1  040C          addl3   r1,16(r1),r5            ;compute starting address
                       0411
                       0411  ;
                       0411  ; reset machine state
                       0411  ;
                       0411
          34 AB    7C  0411          clrq    rpb$l_iovec(r11)        ;no driver
                   05  0414          rsb                             ;done
                       0415
```

```
                                    0415  1074 ;
                                    0415  1075 ; move onto next 16KB boundary
                                    0415  1076 ;
                                    0415  1077
   FFD8 53   20   00001FFF 8F   F1  0415  1078 20$:     acbl    #k_max_memory_pages-1,#32,r3,10$; continue until done
                  50   801A 8F   3C  041F  1079           movzwl  #ss$_norom!2,r0          ;set severe error code
                        FCF0 CF   D4  0424  1080           clrl    machine_check_continue   ;
                                05  0428  1081           rsb
```

```
                      0429  ;++
                      0429  ; network_boot
                      0429  ;
                      0429  ; functional description:
                      0429  ;
                      0429  ; This routine tries to boot from a network device
                      0429  ;
                      0429  ; inputs:
                      0429  ;
                      0429  ;         r7 = address of the internal boot device description
                      0429  ;         r10 = address of the secondary boot's memory
                      0429  ;         r11 = RPB address
                      0429  ;         ap = address of the secondary parameter block
                      0429  ;
                      0429  ; outputs:
                      0429  ;
                      0429  ;         r0 = 1 if success
                      0429  ;         r5 = transfer address
                      0429  ;
                      0429  ;         or
                      0429  ;
                      0429  ;         r0 = ss$_nosuchdev        - CSR does not exist - severe
                      0429  ;            = ss$_bufferovf        - secondary bootstrap does not fit - fatal
                      0429  ;            = ss$_devinact         - device could not be inited - fatal
                      0429  ;            = ss$_ctrlerr          - I/O error during operation - fatal
                      0429  ;            = ss$_devoffline       - device is offline - severe
                      0429  ;
                      0429  ;         r7,r11 are preserved
                      0429  ;--
                      0429
                      0429  network_boot:
                      0429
        0281   30     0429          bsbw    validate_csr                    ;test CSR of device
                      042C                                                  ;return implies success
                      042C  ;
                      042C  ; boot via the Ethernet
                      042C  ;
                      042C
     52   7E   DE     042C          moval   -(sp),r2                        ;address target for transfer address
       68 AB   9F     042F          pushab  rpb$t_file(r11)                 ;address to store node name
       28 AB   9F     0432          pushab  rpb$l_bootr3(r11)               ;address to store node address
          62   9F     0435          pushab  (r2)                            ;address to store transfer address
     4600 CB   9F     0437          pushab  k_pfn_map_addr(r11)             ;address of bit map
     5000 CB   9F     043B          pushab  k_secondary_boot_addr(r11)  ;buffer space
 53  7000 CB   9E     043F          movab   k_secondary_boot_addr+<16*512>(r11),r3 ;image load address
          63   9F     0444          pushab  (r3)                            ;image load address
 0000'CF  06   FB     0446          calls   #6,boo$downline_load            ;try QNA boot
 55   53  8E   C1     044B          addl3   (sp)+,r3,r5                     ;compute transfer address
       03 50   E9     044F          blbc    r0,10$                          ;br if not success
       34 AB   7C     0452          clrq    rpb$l_iovec(r11)                ;no driver
          05          0455  10$:    rsb                                     ;done
```

```
                    0456  ;++
                    0456  ; disk_boot
                    0456  ;
                    0456  ; functional description:
                    0456  ;
                    0456  ; This routine tries to boot from a disk device
                    0456  ;
                    0456  ; inputs:
                    0456  ;
                    0456  ;       r7 = address of the internal boot device description
                    0456  ;       r10 = address of the secondary boot's memory
                    0456  ;       r11 = RPB address
                    0456  ;       ap = address of the secondary parameter block
                    0456  ;
                    0456  ; outputs:
                    0456  ;
                    0456  ;       r0 = 1 if success
                    0456  ;       r5 = transfer address
                    0456  ;
                    0456  ;       or
                    0456  ;
                    0456  ;       r0 = ss$_nosuchdev        - CSR does not exist - severe
                    0456  ;          = ss$_nosuchfile       - file is not on the volume - fatal
                    0456  ;          = ss$_filnotcntg       - boot file is not contiguous - fatal
                    0456  ;          = ss$_bufferovf        - secondary bootstrap does not fit - fatal
                    0456  ;          = ss$_devinact         - device could not be inited - fatal
                    0456  ;          = ss$_ctrlerr          - I/O error during operation - fatal
                    0456  ;          = ss$_devoffline       - device is offline - severe
                    0456  ;
                    0456  ;       r7,r11 are preserved
                    0456  ;--
                    0456
                    0456  disk_boot:
                    0456
        0254   30   0456          bsbw    validate_csr            ;check CSR and return if success
                    0459
                    0459  ;
                    0459  ; move and initialize the disk driver
                    0459  ;
                    0459
 52    34 AB   D0   0459          movl    rpb$l_iovec(r11),r2     ;address boot driver
    59    6B   9E   045D          movab   k_rpb_addr(r11),r9      ;load move parameter
     18 B242   16   0460          jsb     @bqo$l_move(r2)[r2]     ;call move code
                    0464
                    0464  ;
                    0464  ; try low to high units, removable first, non-removable second
                    0464  ;
                    0464  ; Build a mask with two sets of 8 bits. The first 8 bits are the available
                    0464  ; "soft" disk units and the second 8 are the available "hard". The mask
                    0464  ; starts with rpb$w_unit to bd_b_high_unit set in each set.
                    0464  ;
                    0464
          58   D4   0464          clrl    r8                      ;no units to search
 56    64 AB   3C   0466          movzwl  rpb$w_unit(r11),r6      ;build the basic mask
 51    04 A7   9A   046A          movzbl  bd_b_high_unit(r7),r1   ;get high
 58    51  56  C3   046E          subl3   r6,r1,r8                ;number of units
          58   D6   0472          incl    r8                      ;plus 1
```

```
        58  01  58  78  0474          ashl    r8,#1,r8                ;form mask
                58  D7  0478          decl    r8                      ;
        58  58  56  78  047A          ashl    r6,r8,r8                ;move to correct bit pos
    58  08  08  58  F0  047E          insv    r8,#8,#8,r8             ;duplicate mask
            51  56  91  0483          cmpb    r6,r1                   ;high = low - one unit?
                02  12  0486          bneq    10$                     ;br if yes, enter search
                58  94  0488          clrb    r8                      ;no soft disk search
                        048A
                        048A  ;
                        048A  ; select a unit from the mask
                        048A  ;
                        048A
    56  58  10  00  EA  048A  10$:    ffs     #0,#16,r8,r6            ;get the unit number
     64 AB  56  08  AB  048F          bicw3   #^x8,r6,rpb$w_unit(r11) ;set unit number, less mask flag
  FC84 CF   64 AB  30  81  0494       addb3   #^a/0/,rpb$w_unit(r11),boot_device_name+3; new unit in name
                        049B
                        049B  ;
                        049B  ; now, init that unit on the controller
                        049B  ;
                        049B
        52    34 AB  D0  049B         movl    rpb$l_iovec(r11),r2     ;address boot driver
        51    1C A2  D0  049F         movl    bqo$l_unit_init(r2),r1  ;Pick up device init routine
                20  13  04A3          beql    30$                     ;None
                        04A5
                        04A5  ;
                        04A5  ; init the controller and a specific unit
                        04A5  ;
                        04A5  ; it is OK for the unit to be offline but not for the controller to fail
                        04A5  ;
                        04A5
        6241    6C  FA  04A5          callg   (ap),(r2)[r1]           ;do any necessary unit init
           09 50    E8  04A9          blbs    r0,20$                  ;br if unit is online
                        04AC
                        04AC  ;
                        04AC  ; If the unit is not online, it is a fatal error if the controller failed.
                        04AC  ;
                        04AC
     0084 8F    50  B1  04AC          cmpw    r0,#ss$_devoffline      ;offline?
                36  12  04B1          bneq    50$                     ;br if no, more fatal error
                21  11  04B3          brb     35$                     ;continue with next unit
                        04B5
                        04B5  ;
                        04B5  ; controller is up, unit is online, make removable, non-removable tests
                        04B5  ;
                        04B5  ; success from the online is:
                        04B5  ;
                        04B5  ;       #1 unit is online, can't detect hard or soft
                        04B5  ;       #9 unit is online, hard disk
                        04B5  ;       #25 unit is online, soft disk
                        04B5  ;
                        04B5
        0C 56   03  E0  04B5  20$:    bbs     #3,r6,30$               ;br if hard disk mask, try unit
                        04B9
                        04B9  ;
                        04B9  ; looking for a soft disk - can the controller can tell?
                        04B9  ;
                        04B9
```

```
06 50    03   E1  04B9          bbc     #3,r0,25$               ;br if not detectable soft or hard
                  04BD
                  04BD    ;
                  04BD    ; looking for a soft disk and the controller can tell
                  04BD    ;
                  04BD
04 50    04   E0  04BD          bbs     #4,r0,30$               ;br if soft disk flag set, try unit
         1B   11  04C1          brb     40$                     ;continue in common
                  04C3
                  04C3    ;
                  04C3    ; since the controller can't tell, shut off tests in soft mask
                  04C3    ; but do this unit anyway
                  04C3    ;
                  04C3
         58   94  04C3    25$:  clrb    r8                      ;no more soft disk tests
                  04C5
                  04C5    ;
                  04C5    ; try a boot of this unit
                  04C5    ;
                  04C5
   03C0 8F    BB  04C5    30$:  pushr   #^m<r6,r7,r8,r9>        ;save context values
         1F   10  04C9          bsbb    boot_disk_unit          ;try this unit
   03C0 8F    BA  04CB          popr    #^m<r6,r7,r8,r9>        ;restore context values
      17 50   E8  04CF          blbs    r0,50$                  ;br if success
13 50    01   E1  04D2          bbc     #1,r0,50$               ;br if fatal error
                  04D6                                          ;continue if just severe error
51    56 08   C1  04D6    35$:  addl3   #8,r6,r1                ;clear bit in both masks
 00 58   51   E5  04DA          bbcc    r1,r8,40$               ;hard mask or greater
 00 58   56   E5  04DE    40$:  bbcc    r6,r8,45$               ;and soft or hard
         58   B5  04E2    45$:  tstw    r8                      ;more units?
         A4   12  04E4          bneq    10$                     ;br if yes
                  04E6                                          ;exit with error status of last unit
   50    02   88  04E6          bisb    #1@1,r0                 ;make error semi-success
                  04E9
                  04E9    ;
                  04E9    ; fixup name with real booted device and transfer control
                  04E9    ;
                  04E9
              05  04E9    50$:  rsb
```

```
                              04EA          .sbttl  boot a specific disk unit routine
                              04EA  ;++
                              04EA  ; boot_disk_unit
                              04EA  ;
                              04EA  ; functional description:
                              04EA  ;
                              04EA  ; This routine tries a boot of a particular disk unit. The device and
                              04EA  ; driver are present and verified. This routine is used for each unit on
                              04EA  ; which a boot is to be tried. RPB$B_UNIT contains the unit information.
                              04EA  ;
                              04EA  ; inputs:
                              04EA  ;
                              04EA  ;       r9 = rpb address
                              04EA  ;       r10 = address  of the secondary boot's memory
                              04EA  ;       r11 = RPB address
                              04EA  ;       ap = address of the secondary parameter block
                              04EA  ;
                              04EA  ; outputs:
                              04EA  ;
                              04EA  ;       r0 = ss$_success
                              04EA  ;       r5 = transfer address
                              04EA  ;
                              04EA  ;       or
                              04EA  ;
                              04EA  ;       r0 = ss$_nosuchdev       - CSR does not exits
                              04EA  ;          = ss$_nosuchfile      - file is not on the volume
                              04EA  ;          = ss$_filnotcntg      - boot file is not contiguous
                              04EA  ;          = ss$_bufferovf       - secondary bootstrap does not fit
                              04EA  ;          = ss$_devinact        - device could not be inited
                              04EA  ;          = ss$_ctrlerr         - I/O error during operation
                              04EA  ;          = ss$_devoffline      - device is offline
                              04EA  ;
                              04EA  ;       r10 and r11 are preserved.
                              04EA  ;--
                              04EA
                              04EA  boot_disk_unit:
                              04EA
                              04EA  ;
                              04EA  ; do forced boot block boot
                              04EA  ;
                              04EA  ; If RPB$V_BBLOCK is set then read LBN 0 and transfer control to the
                              04EA  ; block.
                              04EA  ;
                              04EA
                  03    E0    04EA          bbs     #rpb$v_bblock,-          ;br if direct boot block boot
            63 30 AB          04EC                  rpb$l_bootr5(r11),80$    ;
                              04EF
                              04EF  ;
                              04EF  ; init the file read cache if this is a FILES-11 boot
                              04EF  ;
                              04EF
  FBDE CF    FC19 CF    7D    04EF          movq    file_cache_desc,fil$gq_cache ;reload the descriptor
                  11    12    04F6          bneq    10$                      ;br if done
               FB05'    30    04F8          bsbw    boo$cache_alloc          ;allocate the cache
       50    8000 8F    3C    04FB          movzwl  #ss$_memerr,r0           ;assume no memory
  FC05 CF    FBD0 CF    7D    0500          movq    fil$gq_cache,file_cache_desc ;save the descriptor
                  48    13    0507          beql    75$                      ;br if cache not allocated
```

```
                          0509
                          0509  ;
                          0509  ; Call a device-independent routine, FIL$OPENFILE to locate the named
                          0509  ; file on the disk.
                          0509  ;
                          0509  ; the cache open is where the drive is mounted so it can fail if there is
                          0509  ; no physical volume
                          0509  ;
                          0509
           FAF4'   30     0509  10$:    bsbw    boo$cache_open          ;Open the FILEREAD cache
           23 50   E9     050C          blbc    r0,55$                  ;br if error
           69 AB   9F     050F  15$:    pushab  rpb$t_file+1(r11)       ;Address of file name string.
      7E   68 AB   9A     0512          movzbl  rpb$t_file(r11),-(sp)   ;Character count of file name.
              7E   D4     0516          clrl    -(sp)                   ;Allocate scratch for channel
                          0518                                          ;and get adr of scratch storage
           3C AB   DF     0518          pushal  rpb$l_fillbn(r11)       ;RPB fields that receive file
                          051B                                          ;statistics during OPEN.
              6A   DF     051B          pushal  (R10)                   ;File header buffer at end of
                          051D                                          ;memory.
         0200 CA   DF     051D          pushal  512(R10)                ;Index file header buffer at
                          0521                                          ;end of memory.
           10 AE   DF     0521          pushal  16(sp)                  ;Address in file name desc.
           10 AE   DF     0524          pushal  16(sp)                  ;Address of phony channel.
     0000'CF  05   FB     0527          calls   #5,fil$openfile         ;Call FILREAD to locate file.
          5E  0C   C0     052C          addl2   #12,sp                  ;Clean up scratch space
          5B  50   E8     052F          blbs    r0,boot_file            ;Branch on success.
                          0532
                          0532  ;
                          0532  ; the volume is not a files-11 volume, try boot block booting, if the error
                          0532  ; related to a file structure problem
                          0532  ;
                          0532
     08C0 8F  50   B1     0532  55$:    cmpw    r0,#ss$_filestruct      ;test for file structure error code
              19   13     0537          beql    80$                     ;br if that's what it is
     0810 8F  50   B1     0539          cmpw    r0,#ss$_badfilehdr      ;test for file structure error code
              12   13     053E          beql    80$                     ;br if that's what it is
     0828 8F  50   B1     0540          cmpw    r0,#ss$_badirectory     ;test for file structure error code
              0B   13     0545          beql    80$                     ;br if that's what it is
     0808 8F  50   B1     0547          cmpw    r0,#ss$_badchksum       ;test for file structure error code
              04   13     054C          beql    80$                     ;br if that's not what it is
          50  02   C8     054E          bisl    #1@1,r0                 ;make non-fatal
              05          0551  75$:    rsb                             ;and go back to caller
                          0552
                          0552  ;
                          0552  ; read LBN 0 as boot block
                          0552  ;
                          0552
              58   D4     0552  80$:    clrl    r8                      ;block to read
          59  01   D0     0554          movl    #1,r9                   ;size to read
          56  5A   D0     0557          movl    r10,r6                  ;Start of free memory
            0080   30     055A          bsbw    readfile                ;read the block to R10
           F1 50   E9     055D          blbc    r0,75$                  ;br if error
                          0560
                          0560  ;
                          0560  ; validate the boot block
                          0560  ;
                          0560
```

```
50    08C2 8F    3C  0560            movzwl  #ss$_filestruct!2,r0      ;set error code, semi-success
52      02 AA    9A  0565            movzbl  2(r10),r2                 ;get offset to secondary id field
01      03 AA    91  0569            cmpb    3(r10),#1                 ;next field a BR instruction
           E2    12  056D            bneq    75$                       ;br if no
51      6A42     3E  056F            movaw   (r10)[r2],r1              ;address next field
                     0573                                              ;this must be in the same page!
        0172     30  0573            bsbw    verify_boot_block         ;check boot block
        D8 50    E9  0576            blbc    r0,75$                    ;br if not a valid block
58   04 AA 10    9C  0579            rotl    #16,4(r10),r8             ;get secondary image LBN
59      08 A1    D0  057E            movl    8(r1),r9                  ;get image size
5A      0C A1    C0  0582            addl    12(r1),r10                ;compute load address
55   10 A1 5A    C1  0586            addl3   r10,16(r1),r5             ;compute transfer address
           42    11  058B            brb     readin_boot               ;boot block is valid, read file
                     058D
                     058D   ;
                     058D   ; File was located successfully. Make sure that the file is contiguous.
                     058D   ; The file statistics block is the following:
                     058D   ;
                     058D   ;            +-----------------------+
                     058D   ;            !     starting LBN      !  (0 if file not contiguous)
                     058D   ;            +-----------------------+
                     058D   ;            !    size in blocks     !
                     058D   ;            +-----------------------+
                     058D   ;
                     058D
                     058D   boot_file:                                 ;Test for contiguity.
58      3C AB    7D  058D            movq    rpb$l_fillbn(r11),r8      ;Get file statistics.
           58    D5  0591            tstl    r8                        ;Contiguous file?
           06    12  0593            bneq    60$                       ;Yes, continue.
50    02AC 8F    3C  0595            movzwl  #ss$_filnotcntg,r0        ;search fatal error
                 05  059A            rsb
                     059B
                     059B   ;
                     059B   ; If the software boot control flags indicate that that transfer
                     059B   ; address of the secondary bootstrap is stored in the image file's
                     059B   ; header block, read that header block. Otherwise, assume that the
                     059B   ; transfer address is simply the 1st byte in the image file.
                     059B   ;
                     059B
55         5A    D0  059B   60$:     movl    r10,r5                    ;Assume no special transfer address.
           06    E1  059E            bbc     #rpb$v_header,-           ;If no header requested,
        30 AB        05A0                    rpb$l_bootr5(r11),-       ;then just branch past header
           2C        05A2                    readin_boot               ;reading code.
56         5A    D0  05A3            movl    r10,r6                    ;Start of free memory
59         01    D0  05A6            movl    #1,r9                     ;Header is always only 1 block.
           32    10  05A9            bsbb    readfile                  ;Read header block.
        5E 50    E9  05AB            blbc    r0,no_fit                 ;br if error
58      3C AB    7D  05AE            movq    rpb$l_fillbn(r11),r8      ;R8 = 1st LBN, R9 = block count
52         59    7D  05B2            movq    r9,r2                     ;R2 = block count, R3 = hdr adr
        FA48'    30  05B5            bsbw    boo$image_att             ;Get image attributes
                     05B8
                     05B8   ;
                     05B8   ; R1 = image header block count
                     05B8   ; R2 = size of file in blocks excluding symbol table and patch text
                     05B8   ;
                     05B8
00A0 CB    51    D0  05B8            movl    r1,rpb$b_hdrpgcnt(r11)    ;Store image header block count
```

```
     59  52  51  C3 05BD        subl3   r1,r2,r9                ;Blocks in image after header block(s)
         58  51  C0 05C1        addl    r1,r8                   ;LBN of first block beyond headr block
     51   02 AA  3C 05C4        movzwl  ihd$w_activoff(r10),r1  ;Get offset to image
                    05C8                                        ;activation data in header.
         51  5A  C0 05C8        addl    r10,r1                  ;Form transfer vector address.
     55    614A  9E 05CB        movab   (r1)[r10],r5            ;Get transfer address.
                    05CF
                    05CF  ;
                    05CF  ; Now read in the file. If the file is too large for the remaining
                    05CF  ; memory space, see if the required additional pages are usable.
                    05CF  ; If they are, use them.  If not issue a fatal diagnostic and HALT.
                    05CF  ;
                    05CF  ; Registers set up now are the following:
                    05CF  ;
                    05CF  ;         R5      - transfer address
                    05CF  ;         R8      - starting LBN of file (after header)
                    05CF  ;         R9      - size of file in blocks
                    05CF  ;         R10     - address of 1st byte in free memory
                    05CF  ;         R11     - address of the RPB
                    05CF  ;         AP      - secondary boot argument list
                    05CF  ;
                    05CF
                    05CF  readin_boot:
                    05CF
  14 AC   44 AB  7D 05CF        movq    rpb$q_pfnmap(r11),vmb$q_pfnmap(ap);setup bitmap desc
         56  5A  D0 05D4        movl    r10,r6                  ;buffer for read
                    05D7
                    05D7  ;
                    05D7  ; Will the desired number of blocks fit in the space remaining in the
                    05D7  ; pre-tested 64kb of memory?  If not, check that the additional pages
                    05D7  ; required are usable.  If they are, then read it all, otherwise quit.
                    05D7  ;
                    05D7
           0136  30 05D7        bsbw    verify_image_memory     ;verify pages for image
          2F 50  E9 05DA        blbc    r0,no_fit               ;br if error
                    05DD
                    05DD  ;
                    05DD  ; Now read the secondary boot code into memory
                    05DD  ;
                    05DD  ;         Calls the device-independent bootstrap QIO routine to read
                    05DD  ;         a file. Divides the file into pieces as large as possible, so
                    05DD  ;         that the read is a small number (like 1) of DMA transfers.
                    05DD  ;
                    05DD  ; Registers:
                    05DD  ;
                    05DD  ;         R5      - secondary boot transfer address
                    05DD  ;         R6      - buffer address
                    05DD  ;         R8      - logical block number (LBN)
                    05DD  ;         R9      - number of blocks in file
                    05DD  ;
                    05DD
                    05DD  readfile:                             ;Read file into memory.
     57   7F 8F  9A 05DD        movzbl  #k_max_io_pages,r7      ;Assume maximum transfer size.
         59  57  D1 05E1        cmpl    r7,r9                   ;Minimize with file size.
             03  15 05E4        bleq    10$                     ;Branch if file larger than
                    05E6                                        ;maximum transfer size.
         57  59  D0 05E6        movl    r9,r7                   ;Set to remaining file size.
```

```
                              05E9  1517 10$:                                 ;Push arguments for QIO.
                  5B    DD    05E9  1518         pushl   r11                  ;Push phony channel number.
                  00    DD    05EB  1519         pushl   #0                   ;Physical read mode.
            7E    21    3C    05ED  1520         movzwl  #io$_readlblk,-(sp)  ;Read logical block function.
                  58    DD    05F0  1521         pushl   r8                   ;Starting LBN.
       7E   57    09    9C    05F2  1522         rotl    #9,r7,-(sp)          ;Transfer size in bytes.
                  56    DD    05F6  1523         pushl   r6                   ;Buffer address
         56    04 AE    C0    05F8  1524         addl    4(sp),r6             ;Update buffer address.
            58    57    C0    05FC  1525         addl    r7,r8                ;Update LBN.
       0000'CF    06    FB    05FF  1526         calls   #6,boo$qio           ;Call a bootstrap QIO routine.
            05    50    E9    0604  1527         blbc    r0,30$               ;Continue on success.
            59    57    C2    0607  1528         subl    r7,r9                ;Decrement blocks remaining.
                  D1    14    060A  1529         bgtr    readfile             ;Continue if not done.
                              060C  1530
                              060C  1531 ;
                              060C  1532 ;       R0      - status
                              060C  1533 ;       R5      - secondary boot transfer address
                              060C  1534 ;       R6      - buffer address updated past last byte read
                              060C  1535 ;       R8      - LBN updated to block after last block read
                              060C  1536 ;       R9      - blocks in file (reduced to number not read)
                              060C  1537 ;
                              060C  1538
                              060C  1539 30$:
                        05    060C  1540 no_fit: rsb                          ;Return to caller when done.
```

```
      060D          .sbttl  scb interrupt routines
      060D   ;++
      060D   ; ignore_scb_int
      060D   ;
      060D   ; functional description:
      060D   ;
      060D   ; This sequence runs via an SCB vectored interrupt.
      060D   ;
      060D   ; inputs:
      060D   ;
      060D   ;       none
      060D   ;
      060D   ; outputs:
      060D   ;
      060D   ;       none
      060D   ;--
      060D
      060D          .align  long
      0610
      0610   ignore_scb_int:
   02 0610          rei                             ;
```

```
                              0611  ;++
                              0611  ; machine_check_detect
                              0611  ;
                              0611  ; functional description:
                              0611  ;
                              0611  ; This sequence runs when it is enabled in the machine check vector.
                              0611  ; The action is to alter the return address to a value in r1 and continue.
                              0611  ;
                              0611  ; inputs:
                              0611  ;
                              0611  ;       machine check stack
                              0611  ;       machine_check_continue  = address of the continuation code or 0
                              0611  ;
                              0611  ; outputs:
                              0611  ;
                              0611  ;       r0 = machine check code
                              0611  ;--
                              0611
                              0611          .align  long
                              0614
                              0614  machine_check_detect:
                              0614
      26  000000FF 8F   DA    0614          mtpr    #^xff,#pr$_mcesr        ;clear machine check error
               FAF9 CF   D5    061B          tstl    machine_check_continue  ;change return PC?
                    0D   13    061F          beql    10$                     ;if eql then no, unexpected
         50    04 AE   D0    0621          movl    4(sp),r0                ;load reason
            5E    8E   C0    0625          addl    (sp)+,sp                ;pop stack
      6E   FAEC CF   D0    0628          movl    machine_check_continue,(sp) ;actually change return PC
                         02    062D          rei                             ;continue
                               062E  10$:    fatal_message   scbint
```

```
                          0636  ;++
                          0636  ; fatal_error
                          0636  ;
                          0636  ; functional description:
                          0636  ;
                          0636  ; This routine is entered when a fatal error is to be displayed.
                          0636  ; The input code is a standard ss$_ value and it is matched to a text
                          0636  ; string by scanning a table of longword entries. The first work of the
                          0636  ; longword is the low word of the ss$ code and the next word is the
                          0636  ; displacement to the message text.
                          0636  ;
                          0636  ; inputs:
                          0636  ;
                          0636  ;       r0 = internal error code
                          0636  ;
                          0636  ; outputs:
                          0636  ;
                          0636  ;       The boot is abandoned, the registers are restored to
                          0636  ;       reflect the initial contents and the system is halted.
                          0636  ;--
                          0636
                          0636  fatal_error:
                          0636
5B    0000'CF     9E      0636          movab   rom_base,r11              ;recalculate RPB address
5B    01FF 8F     AA      063B          bicw    #^xTff,r11                ;
5B    FE00 CB     9E      0640          movab   -k_rom_code_addr(r11),r11;
      FACF CF     D4      0645          clrl    machine_check_continue    ;disable error continue
51    FA32 CF     9E      0649          movab   message_base,r1           ;address message desc
      50   03     CA      064E          bicl    #3,r0                     ;remove severity bits
      50   81     B1      0651  10$:    cmpw    (r1)+,r0                  ;compare code
           06     13      0654          beql    15$                       ;br if found
           81     B5      0656          tstw    (r1)+                     ;advance and test for zero offset?
           F7     12      0658          bneq    10$                       ;continue in not found
           2C     11      065A          brb     20$                       ;if list end then no message
      50   61     32      065C  15$:    cvtwl   (r1),r0                   ;fetch displacement from cell
        20 AB     DD      065F          pushl   rpb$l_bootr1(r11)         ;Pass options switch settings
           7E     7C      0662          clrq    -(sp)                     ;no read data
         6140     9F      0664          pushab  (r1)[r0]                  ;address of message text
                          0667
                          0667  ;
                          0667  ; output the header part followed by the input code's message
                          0667  ;
                          0667
        20 AB     DD      0667          pushl   rpb$l_bootr1(r11)         ;Pass options switch settings
           7E     7C      066A          clrq    -(sp)                     ;setup header
      F9FD CF     9F      066C          pushab  message_header            ;
0000'CF    04     FB      0670          calls   #4,boo$readprompt         ;output header
0000'CF    04     FB      0675          calls   #4,boo$readprompt         ;output message
        20 AB     DD      067A          pushl   rpb$l_bootr1(r11)         ;Pass options switch settings
           7E     7C      067D          clrq    -(sp)                     ;output device name
      FA99 CF     9F      067F          pushab  boot_device_name
0000'CF    04     FB      0683          calls   #4,boo$readprompt         ;
                          0688
                          0688  ;
                          0688  ; reload the input registers
                          0688  ;
                          0688
```

```
        5E    6B   D0  0688  20$:    movl    rpb$l_base(r11),sp      ;load sp
     5E    0200 CE   9E  068B          movab   ^x200(sp),sp            ;
        50    1C AB  7D  0690          movq    rpb$l_bootr0(r11),r0    ;load r0,r1
        52    24 AB  7D  0694          movq    rpb$l_bootr2(r11),r2    ;load r2,r3
        54    2C AB  7D  0698          movq    rpb$l_bootr4(r11),r4    ;load r4,r5
        5C    18 AB  D0  069C          movl    rpb$l_haltcode(r11),ap  ;load halt code
        5A    10 AB  7D  06A0          movq    rpb$l_haltpc(r11),r10   ;restore PC,PSL
                         06A4
                         06A4  ;
                         06A4  ; halt system, continue will restart the boot
                         06A4  ;
                         06A4
  23    00000F05 8F  DA  06A4  25$:    mtpr    #console_halt,#pr$_txdb ;halt processor
                 F7  11  06AB          brb     25$                     ;
```

```
                                06AD  ;++
                                06AD  ; validate_csr - test for present CSR
                                06AD  ;
                                06AD  ; functional description:
                                06AD  ;
                                06AD  ; This routine tests for a device CSR and returns to the caller's caller
                                06AD  ; if the CSR is not present. The CSR address is calculated from the base
                                06AD  ; CSR address and the controller number.
                                06AD  ;
                                06AD  ; inputs:
                                06AD  ;
                                06AD  ;        r7 = boot device descriptor address
                                06AD  ;        r11 = rpb address
                                06AD  ;
                                06AD  ; outputs:
                                06AD  ;
                                06AD  ;        return to caller implies that the device is present
                                06AD  ;        return to caller's caller with r0 = ss$_devassign+2
                                06AD  ;
                                06AD  ;        The RPB$L_PHYCSR value is filled in.
                                06AD  ;
                                06AD  ;        r0,r1 are destroyed
                                06AD  ;--
                                06AD
                                06AD  validate_csr:
                                06AD
   FA65 CF    BB'AF       9E    06AD           movab   b^10$,machine_check_continue ;chance machine check addr
     51    06 A7          D0    06B3           movl    bd_a_csr(r7),r1          ;get csr address
                  50      D4    06B7           clrl    r0                       ;set present flag
                  61      B5    06B9           tstw    (r1)                     ;test
           FA59 CF        D4    06BB  10$:     clrl    machine_check_continue   ;zap machine check address
                  50      D5    06BF           tstl    r0                       ;CSR present?
                  09      13    06C1           beql    20$                      ;continue
     50     084A 8F       3C    06C3           movzwl  #ss$_devassign!2,r0      ;set error but semi-success
                51 8ED0         06C8           popl    r1                       ;pop return to caller
                  05            06CB           rsb                              ;done
     54 AB        51      D0    06CC  20$:     movl    r1,rpb$l_csrphy(r11)     ;save csr address
  23   00000F0E 8F        DA    06D0           mtpr    #led_boot_inprogress,#pr$_txdb; tell operator
                  05            06D7           rsb                              ; return to caller
```

```
06D8   ;++
06D8   ; unfielded_scb_int
06D8   ; secondary_scb_int
06D8   ;
06D8   ; functional description:
06D8   ;
06D8   ; This routine is executed if an unwanted SCB interrupt occurs during
06D8   ; booting. An error message is displayed and the system is halted.
06D8   ;
06D8   ; inputs:
06D8   ;
06D8   ;         scb interrupt stack
06D8   ;
06D8   ; outputs:
06D8   ;
06D8   ;         none
06D8   ;--
06D8
06D8           .align  long
06D8
06D8   unfielded_scb_int:
06D8
06D8           fatal_message   scbint
06E0
06E0   secondary_scb_int:
06E0
06E0           fatal_message   2ndint
```

```
                              06E8   ;++
                              06E8   ; verify_boot_block
                              06E8   ;
                              06E8   ; functional description:
                              06E8   ;
                              06E8   ; This routine verifyes a small memory section as a boot block descriptor.
                              06E8   ; It is used to verify a disk boot block or a ROM id block.
                              06E8   ;
                              06E8   ;
                              06E8   ;
                              06E8   ;                              +-------+-------+-------+-------+
                              06E8   ;        BB+0:                 !     1 !     n !     any value !
                              06E8   ;                              +-------+-------+-------+-------+
                              06E8   ;                              ! Low LBN       ! High LBN      !
                              06E8   ;                              +-------+-------+-------+-------+
                              06E8   ;
                              06E8   ;                              +-------+-------+-------+-------+
                              06E8   ;        BB+(2*n)+0:           !  Chk  !     k !       18(Hex) !
                              06E8   ;                              +-------+-------+-------+-------+
                              06E8   ;                              ! any value, most likely 0      !
                              06E8   ;                              +-------+-------+-------+-------+
                              06E8   ;        BB+(2*n)+8:           ! size in blocks of the image   !
                              06E8   ;                              +-------+-------+-------+-------+
                              06E8   ;        BB+(2*n)+12:          ! offset to load                !
                              06E8   ;                              +-------+-------+-------+-------+
                              06E8   ;        BB+(2*n)+16:          ! offset into image to start    !
                              06E8   ;                              +-------+-------+-------+-------+
                              06E8   ;        BB+(2*n)+20:          ! sum of the previous three LW's!
                              06E8   ;                              +-------+-------+-------+-------+
                              06E8   ;
                              06E8   ; inputs:
                              06E8   ;
                              06E8   ;        r1 = address of the block
                              06E8   ;
                              06E8   ; outputs:
                              06E8   ;
                              06E8   ;        r0 = true or false
                              06E8   ;        r1 = original address
                              06E8   ;
                              06E8   ;        r2 is destroyed
                              06E8   ;--
                              06E8
                              06E8   verify_boot_block:
                              06E8
                  50    D4    06E8           clrl    r0                    ;assume not a valid block
             18   61    B1    06EA           cmpw    (r1),#^x18            ;VAX instruction set id?
                  20    12    06ED           bneq    10$                   ;br if no
     52   02 A1   18    81    06EF           addb3   #^x18,2(r1),r2        ;get optional value
             52   52    92    06F4           mcomb   r2,r2                 ;ones's complement it
          03 A1   52    91    06F7           cmpb    r2,3(r1)              ;check check sum byte
                  12    12    06FB           bneq    10$                   ;continue if no match
 52   0C A1  08 A1      C1    06FD           addl3   8(r1),12(r1),r2       ;check other words
          52  10 A1     C0    0703           addl    16(r1),r2             ;get augment to load address
          14 A1   52    D1    0707           cmpl    r2,20(r1)             ;match?
                  02    12    070B           bneq    10$                   ;br if no
                  50    D6    070D           incl    r0                    ;success
                  05          070F   10$:    rsb
```

```
                        0710  ; ++
                        0710  ; verify_image_memory
                        0710  ;
                        0710  ; functional description:
                        0710  ;
                        0710  ; This routine checks for n contiguous pages from the established load
                        0710  ; address.
                        0710  ;
                        0710  ; inputs:
                        0710  ;
                        0710  ;       r9 = desired page count
                        0710  ;       r10 = target load address
                        0710  ;       r11 = address of the RPB
                        0710  ;       ap = boot argument list
                        0710  ;
                        0710
                        0710  verify_image_memory:
                        0710
   50   0601 8F    3C   0710          movzwl  #ss$_bufferovf,r0          ;set error code
   52   5B   17    9C   0715          rotl    #<32-9>,r11,r2             ;PFN for RPB
     52    7F A2   DE   0719          moval   127(r2),r2                 ;Last PFN guaranteed to be good
   51   5A   17    9C   071D          rotl    #<32-9>,r10,r1             ;Starting PFN for read
        51   59    C0   0721          addl    r9,r1                      ;Last+1 PFN needed to be good
             05    11   0724          brb     30$                        ;Zero or more iterations
07 18 BC     52    E1   0726  10$:    bbc     r2,@vmb$q_pfnmap+4(ap),40$ ;Branch if cannot
                        072B                                             ;read the entire secondary boot
     F7 52   51    F2   072B  30$:    aoblss  r1,r2,10$                  ;Check the next page
        50   01    D0   072F          movl    #1,r0                      ;correct
             05         0732  40$:    rsb
```

```
                     0733   ;++
                     0733   ; write_timeout
                     0733   ;
                     0733   ; functional description:
                     0733   ;
                     0733   ; This sequence runs when a write timeout interrupt occurs.
                     0733   ;
                     0733   ; inputs:
                     0733   ;
                     0733   ;        PC/PSL are on the stack
                     0733   ;        machine_check_continue = address to continue at or 0
                     0733   ;
                     0733   ; outputs:
                     0733   ;
                     0733   ;        r0 = error code
                     0733   ;--
                     0733
                     0733
                     0733            .align  long
                     0734
                     0734   write_timeout_int:
                     0734
6E    F9E0 CF    D0  0734            movl    machine_check_continue,(sp) ;reset PC
           9D    13  0739            beql    unfielded_scb_int          ;unexpected error if no continue addr
     50    02    D0  073B            movl    #k_bus.timeout,r0          ;set code
                 02  073E            rei                                ;done
                     073F
                     073F            .end    ROM_START
```

```
A_2NDINT                          0000017E R     03
A_BADFILENAME                     000000A8 R     03
A_BUFFEROVF                       000000BA R     03
A_CTRLERR                         000000EA R     03
A_DEVASSIGN                       00000036 R     03
A_DEVINACT                        00000101 R     03
A_DEVOFFLINE                      00000123 R     03
A_FILESTRUCT                      00000066 R     03
A_FILNOTCNTG                      0000008A R     03
A_MEMERR                          00000136 R     03
A_NOROM                           000001B1 R     03
A_NOSUCHDEV                       00000000 R     03
A_NOSUCHFILE                      0000004D R     03
A_NOSUCHNODE                      000001CB R     03
A_SCBINT                          00000153 R     03
BDT$L_ACTION                      00000004
BDT$L_ADDR                        0000000C
BDT$L_AUXDRNAME                   00000018
BDT$L_CPUTYPE                     00000000
BDT$L_DEVNAME                     00000024
BDT$L_DEVTYPE                     00000002
BDT$L_DRIVRNAME                   00000014
BDT$L_ENTRY                       00000010
BDT$L_SIZE                        00000008
BDT$L_UNIT_DISC                   00000020
BDT$L_UNIT_INIT                   0000001C
BD_A_CSR                          00000006
BD_A_ROUTINE                      0000000A
BD_B_HIGH_UNIT                    00000004
BD_B_TYPE                         00000005
BD_L_NAME                         00000000
BD_S_BD                           0000000E
BEGIN_BOOT                        000002D8 R     02
BOO$AL_VECTOR                     ********   X   02
BOO$CACHE_ALLOC                   ********   X   02
BOO$CACHE_OPEN                    ********   X   02
BOO$DOWNLINE_LOAD                 ********   X   02
BOO$GB_SYSTEMID                 = 000000F4 RG    02
BOO$GL_RPBBASE                    00000114 RG    02
BOO$IMAGE_ATT                     ********   X   02
BOO$QIO                           ********   X   02
BOO$READPROMPT                    ********   X   02
BOOT_DEVICE_LIST                  00000041 R     02
BOOT_DEVICE_NAME                  0000011C R     02
BOOT_DISK_UNIT                    000004EA R     02
BOOT_FILE                         0000058D R     02
BQO$L_MOVE                      = 00000018
BQO$L_UNIT_INIT                 = 0000001C
BTD$K_PROM                      = 00000008
BTD$K_QNA                       = 00000060
BTD$K_UDA                       = 00000011
CONSOLE_HALT                    = 00000F05
DIAGFILE                          00000019 R     02
DISK_BOOT                         00000456 R     02
EXE$GB_CPUTYPE                    ********   X   02
FATAL_ERROR                       00000636 R     02
FATAL_MEMORY_ERROR                000002D0 R     02
FIL$GQ_CACHE                    = 000000D4 RG    02
FIL$OPENFILE                      ********   X   02
FILE_CACHE_DESC                   0000010C R     02
IGNORE_SCB_INT                    00000610 R     02
IHD$W_ACTIVOFF                  = 00000002
IO$_READLBLK                    = 00000021
IPL$_POWER                      = 0000001F
K_BUS.TIMEOUT                   = 00000002
K_MAX_IO_PAGES                  = 0000007F
K_MAX_MEMORY_PAGES              = 00002000
K_PARITY.ERROR                  = 00000001
K_PFN_MAP_ADDR                  = 00004600
K_ROM_CODE_ADDR                 = 00000200
K_RPB_ADDR                      = 00000000
K_SCB_ADDR                      = 00004200
K_SECONDARY_BOOT_ADDR           = 00005000
LAST_MSG                        = 000001CB R     03
LED_BOOT_INPROGRESS             = 00000F0E
LED_MEMORY_OK                   = 00000F0D
LED_TRANSFER_CONTROL            = 00000F0F
MACHINE_CHECK_CONTINUE            00000118 R     02
MACHINE_CHECK_DETECT              00000614 R     02
MESSAGE_BASE                      0000007F R     02
MESSAGE_HEADER                    0000006D R     02
MSV11_CSR_BASE                  = 20001440
MSV11_CSR_PARITY_ENABLE         = 00000001
NAMEPROMPT                        00000035 R     02
NDT$_UBO                        = 00000028
NETWORK_BOOT                      00000429 R     02
NEXT_PAGE                         00000289 R     02
NO_DISK_BOOT_DEVICE_LIST          0000004F R     02
NO_FIT                            0000060C R     02
NXM_MEMORY                        0000029F R     02
PAGE_BOUNDARY                     000001F8 R     02
PHY_A_IO_SPACE                  = 20000000
PR$V_SID_TYPE                   = 00000018
PR$_IPL                         = 00000012
PR$_MCESR                       = 00000026
PR$_SCBB                        = 00000011
PR$_SID                         = 0000003E
PR$_TXDB                        = 00000023
PROM_BOOT                         000003EF R     02
READFILE                          000005DD R     02
READIN_BOOT                       000005CF R     02
ROM_BASE                          00000000 R     04
ROM_START                         00000121 R     02
RPB$B_BOOTNDT                   = 000000A1
RPB$B_CONFREG                   = 00000090
RPB$B_DEVTYP                    = 00000066
RPB$B_HDRPGCNT                  = 000000A0
RPB$B_SLAVE                     = 00000067
RPB$C_LENGTH                    = 00000108
RPB$C_MEMDSCSIZ                 = 00000008
RPB$C_NMEMDSC                   = 00000008
RPB$L_BASE                      = 00000000
RPB$L_BOOTR0                    = 0000001C
RPB$L_BOOTR1                    = 00000020
```

```
RPB$L_BOOTR2                     = 00000024
RPB$L_BOOTR3                     = 00000028
RPB$L_BOOTR4                     = 0000002C
RPB$L_BOOTR5                     = 00000030
RPB$L_CHKSUM                     = 00000008
RPB$L_CSRPHY                     = 00000054
RPB$L_FILLBN                     = 0000003C
RPB$L_HALTCODE                   = 00000018
RPB$L_HALTPC                     = 00000010
RPB$L_IOVEC                      = 00000034
RPB$L_IOVECSZ                    = 00000038
RPB$L_MEMDSC                     = 000000BC
RPB$L_PFNCNT                     = 0000004C
RPB$L_RESTART                    = 00000004
RPB$L_RSTRTFLG                   = 0000000C
RPB$L_SCBB                       = 000000B0
RPB$Q_PFNMAP                     = 00000044
RPB$S_TOPSYS                     = 00000004
RPB$T_FILE                       = 00000068
RPB$V_BBLOCK                     = 00000003
RPB$V_DIAG                       = 00000004
RPB$V_HALT                       = 00000009
RPB$V_HEADER                     = 00000006
RPB$V_SOLICT                     = 00000008
RPB$V_TOPSYS                     = 0000001C
RPB$W_UNIT                       = 00000064
SCB_A_BREAKPOINT                 = 0000002C
SCB_A_MCHECK                     = 00000004
SCB_A_TRACE_TRAP                 = 00000028
SCB_A_WRITE_TIMEOUT              = 00000060
SECONDARY_SCB_INT                  000006E0 R      02
SECOND_PARAM                       000000D0 R      02
SS$_2NDINT                       = 00008010
SS$_BADCHKSUM                    = 00000808
SS$_BADFILEHDR                   = 00000810
SS$_BADFILENAME                  = 00000818
SS$_BADIRECTORY                  = 00000828
SS$_BUFFEROVF                    = 00000601
SS$_CTRLERR                      = 00000054
SS$_DEVASSIGN                    = 00000848
SS$_DEVINACT                     = 000020D4
SS$_DEVOFFLINE                   = 00000084
SS$_ENDOFFILE                    = 00000870
SS$_FILESTRUCT                   = 000008C0
SS$_FILNOTCNTG                   = 000002AC
SS$_MEMERR                       = 00008000
SS$_NOROM                        = 00008018
SS$_NOSUCHDEV                    = 00000908
SS$_NOSUCHFILE                   = 00000910
SS$_NOSUCHNODE                   = 0000028C
SS$_SCBINT                       = 00008008
SWITCH_V_DISK_BOOT               = 00000007
SWITCH_V_QVSS                    = 00000006
UNFIELDED_SCB_INT                  000006D8 R      02
VALIDATE_CSR                       000006AD R      02
VERIFY_BOOT_BLOCK                  000006E8 R      02
VERIFY_IMAGE_MEMORY                00000710 R      02
VMB$B_SYSTEMID                     00000024
VMB$C_ARGBYTCNT                    0000003C
VMB$L_CI_HIPFN                     00000030
VMB$L_FLAGS                        0000002C
VMB$L_HI_PFN                       00000010
VMB$L_LO_PFN                       0000000C
VMB$Q_FILECACHE                    00000004
VMB$Q_NODENAME                     00000034
VMB$Q_PFNMAP                       00000014
VMB$Q_UCODE                        0000001C
VMSFILE                            00000000 R      02
WRITE_TIMEOUT_INT                  00000734 R      02
```

```
                                     +----------------+
                                     ! Psect synopsis !
                                     +----------------+

PSECT name                      Allocation          PSECT No.  Attributes
----------                      ----------          ---------  ----------
.  ABS  .                       00000000  (    0.)  00 (  0.)  NOPIC   USR   CON   ABS   LCL NOSHR NOEXE NORD   NOWRT NOVEC BYTE
$ABS$                           0000003C  (   60.)  01 (  1.)  NOPIC   USR   CON   ABS   LCL NOSHR   EXE   RD     WRT NOVEC BYTE
$$$$04BOOT                      0000073F  ( 1855.)  02 (  2.)  NOPIC   USR   CON   REL   LCL NOSHR   EXE   RD     WRT NOVEC LONG
$$$$10BOOT                      000001E9  (  489.)  03 (  3.)  NOPIC   USR   CON   REL   LCL NOSHR   EXE   RD     WRT NOVEC BYTE
$$$$00BOOT                      00000008  (    8.)  04 (  4.)  NOPIC   USR   CON   REL   LCL NOSHR   EXE   RD     WRT NOVEC LONG

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

Phase                     Page faults    CPU Time       Elapsed Time
-----                     -----------    --------       ------------
Initialization                     16    00:00:00.08    00:00:00.24
Command processing                 75    00:00:00.55    00:00:01.06
Pass 1                            356    00:00:14.01    00:00:20.29
Symbol table sort                   0    00:00:01.90    00:00:02.02
Pass 2                            291    00:00:04.39    00:00:06.76
Symbol table output                21    00:00:00.11    00:00:00.25
Psect synopsis output               4    00:00:00.03    00:00:00.23
Cross-reference output              0    00:00:00.00    00:00:00.00
Assembler run totals              763    00:00:21.08    00:00:30.86

The working set limit was 1800 pages.
85253 bytes (167 pages) of virtual memory were used to buffer the intermediate code.
There were 70 pages of symbol table space allocated to hold 1184 non-local and 57 local symbols.
1849 source lines were read in Pass 1, producing 23 object records in Pass 2.
24 pages of virtual memory were used to define 23 macros.

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

Macro library name                               Macros defined
------------------                               --------------
DISK$STARWORK03:[GAMACHE.UV1ROM.VMS]LIBUV1.ML          6
DISK$STARWORK03:[GAMACHE.UV1ROM.OBJ]VMB.MLB;3          4
SYS$SYSROOT:[SYSLIB]STARLET.MLB;2                      6
TOTALS (all libraries)                                16

1209 GETS were required to define 16 macros.

There were no errors, warnings or information messages.

MAC/LIS=LIS$:VMBUVAX1/OBJ=OBJ$:VMBUVAX1 VMB$:VMBUVAX1+OBJ$:VMB/LIB+VMS$:LIBUV1/LIB
```

0430 AH-BT13A-SE DIGITAL EQUIPMENT CORPORATION
VAX/VMS V4.0 CONFIDENTIAL AND PROPRIETARY

UV1ROM

PQBTDRIVR
LIS

QVSS
LIS

SEARCHMSG
LIS

VMBUVAX1
MAP

SETUSER
LIS

VMBUVAX1
LIS

BOOTDRUV1
LIS

FILERDUV1
LIS

BOOTIOUV1
LIS

CONIO
LIS

NETBOOT
LIS

0431 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

WELCOME
TXT

LOGIN
COM

SYSTARTUP
COM

XQBOOT
LIS

EDTINI
EDT

REMOVE
COM

VERIFY
LIS

VERIFY

UVMS

UVSTARTUP
COM

VERIFY
MAP

ADDUSER
COM

MODPARAMS
DAT

RESTUSER
COM

BACKUSER
COM

SUCCESS
TXT

SYLOGIN
COM