

[illegible]


```
IIIIII  NN  NN  IIIII  BBBB BBBB  AAAAA  DDDDDDD
IIIIII  NN  NN  IIIII  BBBB BBBB  AAAAA  DDDDDDD
  II    NN  NN  II     BB      AA      AA  DD      DD
  II    NN  NN  II     BB      AA      AA  DD      DD
  II    NNNN NN  II     BB      AA      AA  DD      DD
  II    NNNN NN  II     BBBB BBBB AA      AA  DD      DD
  II    NN  NN  II     BBBB BBBB AA      AA  DD      DD
  II    NN  NN  II     BB      BB  AAAAAAAAAA DD      DD
  II    NN  NN  II     BB      BB  AAAAAAAAAA DD      DD
  II    NN  NN  II     BB      BB  AA      AA  DD      DD
  II    NN  NN  IIIII  BBBB BBBB  AA      AA  DDDDDDD
IIIIII  NN  NN  IIIII  BBBB BBBB  AA      AA  DDDDDDD
```

....
....
....
....

```
LL      IIIII  SSSSSSS
LL      IIIII  SSSSSSS
LL      II     SS
LL      II     SS
LL      II     SS
LL      II     SS
LL      II     SSSSS
LL      II     SSSSS
LL      II     SS
LL      II     SS
LL      II     SS
LL      II     SS
LLLLLLLLLL IIIII  SSSSSSS
LLLLLLLLLL IIIII  SSSSSSS
```



```
1 0001 0 MODULE INIBAD (
2 0002 0
3 0003 0 LANGUAGE (BLISS32),
4 0004 0 IDENT = 'V04-000'
5 0005 1 BEGIN
6 0006 1
7 0007 1
8 0008 1 *****
9 0009 1 *
10 0010 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
11 0011 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
12 0012 1 * ALL RIGHTS RESERVED.
13 0013 1 *
14 0014 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
15 0015 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
16 0016 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
17 0017 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
18 0018 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
19 0019 1 * TRANSFERRED.
20 0020 1 *
21 0021 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
22 0022 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
23 0023 1 * CORPORATION.
24 0024 1 *
25 0025 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
26 0026 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
27 0027 1 *
28 0028 1 *
29 0029 1 *****
30 0030 1
31 0031 1 ++
32 0032 1
33 0033 1 FACILITY: INIT Utility Structure Level 1
34 0034 1
35 0035 1 ABSTRACT:
36 0036 1
37 0037 1 This module contains the routines that do the bad block processing.
38 0038 1
39 0039 1 ENVIRONMENT:
40 0040 1
41 0041 1 STARLET operating system, including privileged system services
42 0042 1 and internal exec routines.
43 0043 1
44 0044 1 --
45 0045 1
46 0046 1
47 0047 1 AUTHOR: Andrew C. Goldstein, CREATION DATE: 10-Nov-1977 19:21
48 0048 1
49 0049 1 MODIFIED BY:
50 0050 1
51 0051 1 V03-004 ACG0361 Andrew C. Goldstein, 21-Sep-1983 16:45
52 0052 1 Eliminate use of physical read operations
53 0053 1
54 0054 1 V03-003 LMP0060 L. Mark Pilant, 24-Nov-1982 14:39
55 0055 1 Correct a problem that caused the software badblock information
56 0056 1 to be ignored on a device whose sector size was not 512 bytes
57 0057 1 (RL02's).
```


INIBAD
V04-000

G 16
16-Sep-1984 01:43:03
14-Sep-1984 12:35:13

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[INIT.SRC]INIBAD.B32;1 Page 2
(1)

```
: 58      0058 1 |
: 59      0059 1 |
: 60      0060 1 |
: 61      0061 1 |
: 62      0062 1 |
: 63      0063 1 |
: 64      0064 1 |
: 65      0065 1 |
: 66      0066 1 |
: 67      0067 1 |
: 68      0068 1 |
: 69      0069 1 |
: 70      0070 1 |
: 71      0071 1 |
: 72      0072 1 |
: 73      0073 1 |
: 74      0074 1 |
: 75      0075 1 |
: 76      0076 1 |
: 77      0077 1 |
: 78      0078 1 |
: 79      0079 1 |
: 80      0370 1 |
: 81      0502 1 |
: 82      0503 1 |
: 83      0504 1 |
: 84      0505 1 |
: 85      0506 1 |
: 86      0507 1 |
: 87      0508 1 |
: 88      0509 1 |

V03-002 LMP50629      L. Mark Pilant,      1-Nov-1982 15:52
Correct a problem that caused an invalid media address error
to be returned when the number of sectors in a cylinder was
not a multiple of the device blocking factor.

V03-001 ACG0283      Andrew C. Goldstein,      8-Apr-1982 10:15
Clean up use of VERIFIED options

V0102   ACG0075      Andrew C. Goldstein,      19-Oct-1979 17:48
Add pack serial number to home block

V0101   ACG0069      Andrew C. Goldstein,      9-Oct-1979 16:44
Remove device data table; always look for DEC-144 data

V0100   ACG00001      Andrew C. Goldstein,      10-Oct-1978 21:27
Previous revision history moved to [INIT.SRC]INIT.REV

**

LIBRARY 'SYSS$LIBRARY:LIB.L32';
REQUIRE 'SRC$:INIDEF.B32';
REQUIRE 'LIBD$: [VMSLIB.OBJ]INITMSG.B32';

FORWARD ROUTINE
INIT_BADBLOCKS : NOVALUE,      ! main level bad block processing
GET_FACTBAD,   ! process factory bad block data
GET_SOFTBAD   : NOVALUE,      ! process bad block scan program data
GET_USERBAD   : NOVALUE,      ! process user specified data
MARK_BAD      : NOVALUE;      ! enter bad block in allocation table
```



```

90 0510 1 GLOBAL ROUTINE INIT_BADBLOCKS : NOVALUE =
91 0511 1
92 0512 1 ++
93 0513 1
94 0514 1 FUNCTIONAL DESCRIPTION:
95 0515 1
96 0516 1 This is the main bad block processing routine. It calls the software
97 0517 1 data, factory data, and manually entered bad block routines as
98 0518 1 is appropriate.
99 0519 1
100 0520 1
101 0521 1 CALLING SEQUENCE:
102 0522 1 INIT_BADBLOCKS ()
103 0523 1
104 0524 1 INPUT PARAMETERS:
105 0525 1 NONE
106 0526 1
107 0527 1 IMPLICIT INPUTS:
108 0528 1 parser data base
109 0529 1 data base in INIT_DISK
110 0530 1
111 0531 1 OUTPUT PARAMETERS:
112 0532 1 NONE
113 0533 1
114 0534 1 IMPLICIT OUTPUTS:
115 0535 1 bad block area in allocation table
116 0536 1
117 0537 1 ROUTINE VALUE:
118 0538 1 NONE
119 0539 1
120 0540 1 SIDE EFFECTS:
121 0541 1 disk bad block data read
122 0542 1
123 0543 1 --
124 0544 1
125 0545 2 BEGIN
126 0546 2
127 0547 2 EXTERNAL
128 0548 2 INIT_OPTIONS : BITVECTOR, ! command options
129 0549 2 DEVICE_CHAR : BBLOCK, ! device characteristics
130 0550 2 VOLUME_SIZE, ! size of volume rounded to cluster
131 0551 2 SMALL_DISK; ! maximum size of a "small" disk
132 0552 2
133 0553 2
134 0554 2 ! Establish whether the volume has factory bad block data or not and
135 0555 2 ! call the appropriate routine. Then, if user data has been entered,
136 0556 2 ! call the routine to process it.
137 0557 2
138 0558 2
139 0559 2 IF .INIT_OPTIONS[OPT_VERIFIED]
140 0560 2 THEN
141 0561 2 BEGIN
142 0562 2 IF NOT GET_FACTBAD ()
143 0563 2 THEN GET_SOFTBAD ();
144 0564 2 END
145 0565 2 ELSE
146 0566 2 BEGIN
```


INIBAD
V04-000

I 16
16-Sep-1984 01:43:03
14-Sep-1984 12:35:13

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[INIT.SRC]INIBAD.B32;1 Page 4
(2)

```
: 147      0567 3      IF .DEVICE_CHAR[DIB$ MAXBLOCK] LSSU .VOLUME_SIZE
: 148      0568 3      THEN MARK_BAD (1, .DEVICE_CHAR[DIB$ MAXBLOCK]);
: 149      0569 2      END;
: 150      0570 2
: 151      0571 2      IF .INIT_OPTIONS[OPT_BADBLOCKS]
: 152      0572 2      THEN GET_USERBAD ();
: 153      0573 2
: 154      0574 1      END;
```

! end of routine INIT_BADBLOCKS

```
.TITLE INIBAD
.IDENT \V04-000\

.EXTRN INIT_OPTIONS, DEVICE_CHAR
.EXTRN VOLUME_SIZE, SMALL_DISK

.PSECT $CODE$,NOWRT,2

.ENTRY INIT_BADBLOCKS, Save nothing
BBC #6, INIT_OPTIONS, 1$
CALLS #0, GET_FACTBAD
BLBS R0, 2$
CALLS #0, GET_SOFTBAD
BRB 2$
CMPL DEVICE_CHAR+112, VOLUME_SIZE
BGEQU 2$
PUSHL DEVICE_CHAR+112
PUSHL #1
CALLS #2, MARK_BAD
BBC #1, INIT_OPTIONS+1, 3$
CALLS #0, GET_USERBAD
RET
```

```
: 0510
: 0559
: 0562
: 0563
: 0559
: 0567
: 0568
: 0571
: 0572
: 0574
```

; Routine Size: 55 bytes, Routine Base: \$CODE\$ + 0000


```
156 0575 1 ROUTINE GET_FACTBAD =
157 0576 1
158 0577 1 ++
159 0578 1
160 0579 1 FUNCTIONAL DESCRIPTION:
161 0580 1
162 0581 1 This routine processes the factory bad block data found on the last
163 0582 1 track of the disk.
164 0583 1
165 0584 1
166 0585 1 CALLING SEQUENCE:
167 0586 1 GET_FACTBAD ()
168 0587 1
169 0588 1 INPUT PARAMETERS:
170 0589 1 NONE
171 0590 1
172 0591 1 IMPLICIT INPUTS:
173 0592 1 device table in INIT_DISK
174 0593 1
175 0594 1 OUTPUT PARAMETERS:
176 0595 1 NONE
177 0596 1
178 0597 1 IMPLICIT OUTPUTS:
179 0598 1 allocation table in INIT_DISK
180 0599 1 SERIAL_NUMBER: pack serial number from bad block data
181 0600 1
182 0601 1 ROUTINE VALUE:
183 0602 1 LBS if factory data found
184 0603 1 LBC if factory data not found
185 0604 1
186 0605 1 SIDE EFFECTS:
187 0606 1 disk blocks read
188 0607 1
189 0608 1 --
190 0609 1
191 0610 2 BEGIN
192 0611 2
193 0612 2 LABEL
194 0613 2 SEARCH_TRACK; ! main loop to search last track of disk
195 0614 2
196 0615 2 LOCAL
197 0616 2 LBN, ! LBN to mark bad
198 0617 2 BLOCKFACT, ! blocking factor of disk
199 0618 2 FIRST_TIME, ! first time through flag
200 0619 2 FIRST_BUFFER, ! first buffer flag
201 0620 2 NOGOOD, ! no blocks read without errors
202 0621 2 STATUS, ! return status
203 0622 2 P : REF BBLOCK, ! pointer into bad block descriptors
204 0623 2 DATA_LBN; ! LBN of current block in last track
205 0624 2
206 0625 2 OWN
207 0626 2 BUFFER2 : BBLOCK [512]; ! buffer for second copy of data
208 0627 2
209 0628 2 EXTERNAL
210 0629 2 INIT_OPTIONS : BITVECTOR, ! command options
211 0630 2 SERIAL_NUMBER, ! pack serial number
212 0631 2 DEVICE_CHAR : BBLOCK, ! device characteristics
```



```
213 0632 2      BUFFER      : BBLOCK;      ! I/O buffer
214 0633 2
215 0634 2  EXTERNAL ROUTINE
216 0635 2      READ_BLOCK;      ! read disk block
217 0636 2
218 0637 2
219 0638 2  ! First mark the entire last track of the disk bad to prevent its use.
220 0639 2  !
221 0640 2
222 0641 3  BLOCKFACT = (.DEVICE_CHAR[DIB$B_SECTORS]
223 0642 3      * .DEVICE_CHAR[DIB$B_TRACKS]
224 0643 3      * .DEVICE_CHAR[DIB$W_CYLINDERS])
225 0644 2      / .DEVICE_CHAR[DIB$L_MAXBLOCK];
226 0645 2
227 0646 2  DATA_LBN = .DEVICE_CHAR[DIB$L_MAXBLOCK] - .DEVICE_CHAR[DIB$B_SECTORS]/.BLOCKFACT;
228 0647 2  MARK_BAD (.DEVICE_CHAR[DIB$B_SECTORS]/.BLOCKFACT, .DATA_LBN);
229 0648 2
230 0649 2  ! Now, if automatic bad block processing is not inhibited, find a good
231 0650 2  ! block on the last track and process the bad block list in it. Do this
232 0651 2  ! twice, once on the first good block and once on the first good block
233 0652 2  ! after sector 10 (if not redundant) to get both factory and software
234 0653 2  ! detected bad block data.
235 0654 2  !
236 0655 2
237 0656 2  IF NOT .INIT_OPTIONS[OPT_VERIFIED] THEN RETURN 1;
238 0657 2
239 0658 2  FIRST_TIME = 1;
240 0659 2  NOGOOD = 1;
241 0660 2
242 0661 3  SEARCH_TRACK: BEGIN
243 0662 3  WHILE T DO
244 0663 4      BEGIN
245 0664 4
246 0665 4      FIRST_BUFFER = 1;
247 0666 4      WHILE 1 DO
248 0667 5          BEGIN
249 0668 5              STATUS = READ_BLOCK (.DATA_LBN, (IF .FIRST_BUFFER THEN BUFFER ELSE BUFFER2));
250 0669 5              IF .STATUS
251 0670 5                  THEN
252 0671 6                  BEGIN
253 0672 6                      NOGOOD = 0;
254 0673 6                      IF .FIRST_BUFFER
255 0674 6                          THEN
256 0675 7                          BEGIN
257 0676 7                              IF .BUFFER[BBD$L_LASTWORD] EQL -1
258 0677 7                                  THEN FIRST_BUFFER = 0;
259 0678 7                              END
260 0679 6                          ELSE
261 0680 7                              BEGIN
262 0681 7                                  IF CH$EQL (512, BUFFER, 512, BUFFER2, 0)
263 0682 7                                      THEN EXITLOOP;
264 0683 6                                  END;
265 0684 6                              END
266 0685 5                          ELSE IF .STATUS NEQ SSS_PARITY
267 0686 5                              THEN ERR_EXIT (.STATUS);
268 0687 5
269 0688 5      DATA_LBN = .DATA_LBN + 1;
```



```

: 270      0689 5      IF .DATA_LBN GEQ .DEVICE_CHAR[DIB$L_MAXBLOCK]
: 271      0690 5      THEN LEAVE SEARCH_TRACK;
: 272      0691 4      END;                                ! end of block search loop
: 273      0692 4
: 274      0693 4 ! We have a good bad block list. Process its entries.
: 275      0694 4 !
: 276      0695 4
: 277      0696 4      IF .FIRST TIME
: 278      0697 4      THEN SERIAL_NUMBER = .BUFFER[BBD$L_SERIAL];
: 279      0698 4
: 280      0699 4      IF .BUFFER[BBD$W_FLAGS] EQL 65535
: 281      0700 4      THEN ERR_EXIT (INIT$_DIAGPACK);
: 282      0701 4
: 283      0702 4      P = BUFFER + BBD$C_DESCRIPTOR;
: 284      0703 4      DO
: 285      0704 5          BEGIN
: 286      0705 5              IF .P[BBD$V_CYLINDER] EQL 32767
: 287      0706 5              THEN EXITLOOP;
: 288      0707 7              LBN = ((.P[BBD$V_CYLINDER] * .DEVICE_CHAR[DIB$B_TRACKS]
: 289      0708 6                  + .P[BBD$V_TRACK]) * .DEVICE_CHAR[DIB$B_SECTORS]
: 290      0709 5                  + .P[BBD$V_SECTOR]) / .BLOCKFACT;
: 291      0710 5              MARK_BAD (1, LBN);
: 292      0711 5              P = .P + BBD$C_ENTRY;
: 293      0712 5          END
: 294      0713 4      UNTIL .P GEQA BUFFER+512;
: 295      0714 4
: 296      0715 4 ! If we are not yet into the user data, position to it and try again.
: 297      0716 4 !
: 298      0717 4
: 299      0718 4      FIRST_TIME = 0;
: 300      0719 4      IF DATA_LBN GEQU .DEVICE_CHAR[DIB$L_MAXBLOCK]
: 301      0720 4          - .DEVICE_CHAR[DIB$B_SECTORS] / .BLOCKFACT + 10 THEN EXITLOOP;
: 302      0721 4      DATA_LBN = .DEVICE_CHAR[DIB$L_MAXBLOCK]
: 303      0722 4          - .DEVICE_CHAR[DIB$B_SECTORS] / .BLOCKFACT + 10;
: 304      0723 3      END;                                ! end of outer loop
: 305      0724 2 END;                                ! end of block SEARCH_TRACK
: 306      0725 2
: 307      0726 2 ! If we found no good data at all, complain.
: 308      0727 2 !
: 309      0728 2
: 310      0729 2 IF .NOGOOD
: 311      0730 2 THEN ERR_EXIT (INIT$_FACTBAD);
: 312      0731 2
: 313      0732 2 RETURN NOT .FIRST_TIME;
: 314      0733 2
: 315      0734 1 END;                                ! end of routine GET_FACTBAD
```

.PSECT \$OWNS,NOEXE,2

00000 BUFFER2:.BLKB 512

.EXTRN SERIAL_NUMBER, BUFFER
.EXTRN READ_BLOCK

.PSECT \$CODE\$,NOWRT,2


```
OFFC 00000 GET_FACTBAD:
      50      0000G CF 9A 00002      .WORD      Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11      : 0575
      51      0000G CF 9A 00007      MOVZBL     DEVICE_CHAR+8, R0      : 0642
      52      0000G CF 3C 0000F      MOVZBL     DEVICE_CHAR+9, R1
      53      0000G CF 3C 0000F      MULL2      R1, R0
      54      0000G CF 3C 0000F      MOVZWL     DEVICE_CHAR+10, R2      : 0643
      55      0000G CF 3C 0000F      MULL2      R2, R0
      56      0000G CF C7 00017      DIVL3      DEVICE_CHAR+112, R0, BLOCKFACT      : 0644
      57      0000G CF 9A 0001D      MOVZBL     DEVICE_CHAR+8, R0      : 0646
      58      0000G CF 57 C6 00022      DIVL2      BLOCKFACT, R0
      59      0000G CF 50 C3 00025      SUBL3      R0, DEVICE_CHAR+112, DATA_LBN
      7E      0000G CF 6E DD 0002B      PUSHL     DATA_LBN      : 0647
      04      0000V CF 02 FB 0002F      CALLS      #2, MARK_BAD
      05      0000G CF 06 E0 00034      BRB      #6, INIT_OPTIONS, 1$      : 0656
      06      0000G CF 01 D0 0003A      MOVL      #1, R0
      07      0000G CF 04 0003D      RET
      08      0000G CF 01 D0 0003E 1$:      MOVL      #1, FIRST_TIME      : 0658
      09      0000G CF 01 D0 00041      MOVL      #1, NOGOOD      : 0659
      10      0000G CF 01 D0 00044 2$:      MOVL      #1, FIRST_BUFFER      : 0665
      11      0000G CF 6E D0 00047      MOVL      DATA_LBN, R6      : 0668
      12      0000G CF 5A E9 0004A 3$:      BLBC      FIRST_BUFFER, 4$
      13      0000G CF 05 11 00052      MOVAB     BUFFER, R0
      14      0000G CF 05 11 00052      BRB      5$
      15      0000G CF 50 9E 00054 4$:      MOVAB     BUFFER2, R0
      16      0000G CF 50 DD 00059 5$:      PUSHL     R0
      17      0000G CF 56 DD 0005B      PUSHL     R6
      18      0000G CF 02 FB 0005D      CALLS      #2, READ_BLOCK
      19      0000G CF 50 D0 0C062      MOVL      R0, STATUS
      20      0000G CF 58 E9 00065      BLBC      STATUS, 7$      : 0669
      21      0000G CF 59 D4 00068      CLRL      NOGOOD      : 0672
      22      0000G CF 5A E9 0006A      BLBC      FIRST_BUFFER, 6$      : 0673
      23      0000G CF 8F 0006D      CMPL      BUFFER+508, #-1      : 0676
      24      0000G CF 24 12 00076      BNEQ      8$
      25      0000G CF 5A D4 00078      CLRL      FIRST_BUFFER      : 0677
      26      0000G CF 20 11 0007A      BRB      8$      : 0673
      27      0000G CF 8F 29 0007C 6$:      CMPC3     #512, BUFFER, BUFFER2      : 0681
      28      0000G CF 14 12 00086      BNEQ      8$
      29      0000G CF 21 11 00088      BRB      9$      : 0682
      30      0000G CF 58 D1 0008A 7$:      CMPL      STATUS, #500      : 0685
      31      0000G CF 09 13 00091      BEQL      8$
      32      0000G CF 58 DD 00093      PUSHL     STATUS      : 0686
      33      0000G CF 01 FB 00095      CALLS      #1, LIB$STOP
      34      0000G CF 6E D6 0009C 8$:      INCL      DATA_LBN      : 0688
      35      0000G CF 6E D0 0009E      MOVL      DATA_LBN, R6      : 0689
      36      0000G CF 56 D1 000A1      CMPL      R6, DEVICE_CHAR+112
      37      0000G CF A2 19 000A6      BLSS      3$
      38      0000G CF 0092 31 000A8      BRW      14$      : 0690
      39      0000G CF 07 55 E9 000AB 9$:      BLBC      FIRST_TIME, 10$      : 0696
      40      0000G CF 0000G CF D0 000AE      MOVL      BUFFER, SERIAL_NUMBER      : 0697
      41      0000G CF FFFF 8F 0000G CF B1 000B5 10$:      CMPW      BUFFER+6, #65535      : 0699
      42      0000G CF 0D 12 000BC      BNEQ      11$
      43      0000G CF 8F DD 000BE      PUSHL     #7700644
      44      0000G CF 01 FB 000C4      CALLS      #1, LIB$STOP      : 0700
      45      0000G CF 00 9E 000CB 11$:      MOVAB     BUFFER+8, P      : 0702
      46      0000G CF 00 ED 000D0 12$:      CMPZV     #0, #15, (P), #32767      : 0705
```


INIBAD
V04-000

B 1
16-Sep-1984 01:43:03
14-Sep-1984 12:35:13

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[INIT.SRC]INIBAD.B32;1
Page 9
(3)

50	64	0F		3F 13 000D9	BEQL	13\$		
		51	0000G	00 EF 000DB	EXTZV	#0, #15, (P), R0		0707
		50		CF 9A 000E0	MOVZBL	DEVICE_CHAR+9, R1		
52	03	07		51 C4 000E5	MULL2	R1, R0		
		50		00 EF 000E8	EXTZV	#0, #7, 3(P), R2		0708
		51	0000G	52 C0 000EE	ADDL2	R2, R0		
		50		CF 9A 000F1	MOVZBL	DEVICE_CHAR+8, R1		
		52	02	51 C4 000F6	MULL2	R1, R0		
		50		A4 9A 000F9	MOVZBL	2(P), R2		0709
	5B	50		52 C0 000FD	ADDL2	R2, R0		
		50		57 C7 00100	DIVL3	BLOCKFACT, R0, LBN		
				5B DD 00104	PUSHL	LBN		0710
				01 DD 00106	PUSHL	#1		
		0000V		02 FB 00108	CALLS	#2, MARK_BAD		
		54		04 C0 0010D	ADDL2	#4, P		0711
		50	0000G	CF 9E 00110	MOVAB	BUFFER+512, R0		0713
		50		54 D1 00115	CMPL	P, R0		
				B6 1F 00118	BLSSU	12\$		
		50	0000G	55 D4 0011A 13\$:	CLRL	FIRST TIME		0718
		50		CF 9A 0011C	MOVZBL	DEVICE_CHAR+8, R0		0720
		50	0000G	57 C6 00121	DIVL2	BLOCKFACT, R0		
		50		CF C2 00124	SUBL2	DEVICE_CHAR+112, R0		
		50		0A C2 00129	SUBL2	#10, R0		
		52		50 CE 0012C	MNEGL	R0, R2		
		51		6E 9E 0012F	MOVAB	DATA_LBN, R1		0719
		51		52 D1 00132	CMPL	R2, R1		
				06 1B 00135	BLEQU	14\$		
		6E		50 CE 00137	MNEGL	R0, DATA_LBN		0722
				FF07 31 0013A	BRW	2\$		0662
		0D		59 E9 0013D 14\$:	BLBC	NOGOOD, 15\$		0729
			007580AC	8F DD 00140	PUSHL	#7700652		0730
		00000000G		01 FB 00146	CALLS	#1, LIB\$STOP		
		55		55 D2 0014D 15\$:	MCOML	FIRST TIME, R5		0732
		50		55 D0 00150	MOVL	R5, R0		
				04 00153	RET			0734

; Routine Size: 340 bytes, Routine Base: \$CODE\$ + 0037


```
317 0735 1 ROUTINE GET_SOFTBAD : NOVALUE =
318 0736 1
319 0737 1 ++
320 0738 1
321 0739 1 FUNCTIONAL DESCRIPTION:
322 0740 1
323 0741 1 This routine processes the data left by the bad block scan program
324 0742 1 somewhere near the end of the disk.
325 0743 1
326 0744 1
327 0745 1 CALLING SEQUENCE:
328 0746 1 GET_SOFTBAD ()
329 0747 1
330 0748 1 INPUT PARAMETERS:
331 0749 1 NONE
332 0750 1
333 0751 1 IMPLICIT INPUTS:
334 0752 1 device table in INIT_DISK
335 0753 1
336 0754 1 OUTPUT PARAMETERS:
337 0755 1 NONE
338 0756 1
339 0757 1 IMPLICIT OUTPUTS:
340 0758 1 allocation table in INIT_DISK
341 0759 1
342 0760 1 ROUTINE VALUE:
343 0761 1 NONE
344 0762 1
345 0763 1 SIDE EFFECTS:
346 0764 1 disk blocks read
347 0765 1
348 0766 1 --
349 0767 1
350 0768 2 BEGIN
351 0769 2
352 0770 2 LOCAL
353 0771 2 LBN,
354 0772 2 STATUS,
355 0773 2 P : REF BBLOCK; ! LBN to mark bad
356 0774 2 ! return status
357 0775 2 ! pointer into bad block map
358 0776 2 EXTERNAL
359 0777 2 INIT_OPTIONS : BITVECTOR, ! command options
360 0778 2 DEVICE_CHAR : BBLOCK, ! device characteristics
361 0779 2 BUFFER : BBLOCK; ! I/O buffer
362 0780 2 EXTERNAL ROUTINE
363 0781 2 READ_BLOCK, ! read block by LBN
364 0782 2 CHECKSUM2; ! compute block checksum
365 0783 2
366 0784 2
367 0785 2 ! Scan from the end of the volume forward to find the bad block data.
368 0786 2 ! If none is found, output a warning and proceed.
369 0787 2 !
370 0788 2
371 0789 2 LBN = .DEVICE_CHAR[DIB$L_MAXBLOCK];
372 0790 2 IF
373 0791 3 BEGIN
```



```

374 0792 3  DECR J FROM 32 TO 1 DO
375 0793 4  BEGIN
376 0794 4  LBN = .LBN - 1;
377 0795 4  STATUS = READ_BLOCK (.LBN, BUFFER);
378 0796 4
379 0797 4  IF .STATUS
380 0798 4  THEN
381 0799 5  BEGIN
382 0800 5  IF CHECKSUM2 (BUFFER, $BYTEOFFSET (BBM$W_CHECKSUM))
383 0801 5  AND .BUFFER[BBM$B_COUNTSIZE] EQL 1
384 0802 5  AND .BUFFER[BBM$B_LBNSIZE] EQL 3
385 0803 5  AND .BUFFER[BBM$B_INUSE] LEQ (512 - BBM$C_POINTERS - 2) / 2
386 0804 5  THEN EXITLOOP 0;
387 0805 5  END
388 0806 4  ELSE IF .STATUS NEQ SSS_PARITY
389 0807 4  THEN ERR_EXIT (.STATUS);
390 0808 4  END
391 0809 3  END
392 0810 2  THEN
393 0811 3  BEGIN
394 0812 3  ERR_MESSAGE (INIT$NOBADDATA);
395 0813 3  RETURN;
396 0814 2  END;
397 0815 2
398 0816 2  ! Found a good bad block descriptor. Enter it in the bad block map and
399 0817 2  ! then process its contents.
400 0818 2  !
401 0819 2
402 0820 2  MARK_BAD (.DEVICE_CHAR[DIB$L_MAXBLOCK] - .LBN, .LBN);
403 0821 2
404 0822 2  P = BUFFER + BBM$C_POINTERS;
405 0823 2  DECR J FROM .BUFFER[BBM$B_INUSE]/2 TO 1 DO
406 0824 3  BEGIN
407 0825 3  LBN = .P[BBM$W_LOWLBN];
408 0826 3  LBN<16,8> = .P[BBM$B_HIGHLBN];
409 0827 3  MARK_BAD (.P[BBM$B_COUNT]+1, .LBN);
410 0828 3  P = .P + 4;
411 0829 2  END;
412 0830 2
413 0831 1  END;

! end of routine GET_SOFTBAD
```

```

                                .EXTRN CHECKSUM2
                                003C 00000 GET_SOFTBAD:
                                .WORD Save R2,R3,R4,R5
                                55 0000G CF 9E 00002 MOVAB BUFFER, R5
                                54 0000G CF D0 00007 MOVL DEVICE_CHAR+112, LBN
                                52 20 D0 0000C MOVL #32, J
                                55 DD 0000F 1$: PUSHL R5
                                74 9F 00011 PUSHAB -(LBN)
                                0000G CF 02 FB 00013 CALLS #2, READ_BLOCK
                                53 50 D0 00018 MOVL R0, STATUS
                                23 53 E9 0001B BLBC STATUS, 2$
                                7E 01FE 8F 3C 0001E MOVZWL #510, -(SP)
                                55 DD 00023 PUSHL R5
```

```

: 0735
:
: 0789
: 0792
: 0795
:
:
: 0797
: 0800
:
```


0000G	CF		02	FB	00025	CALLS	#2, CHECKSUM2	:	
	26		50	E9	0002A	BLBC	R0, 3\$	:	
	01		65	91	0002D	CMPB	BUFFER, #1	:	0801
			21	12	00030	BNEQ	3\$	:	
	03	01	A5	91	00032	CMPB	BUFFER+1, #3	:	0802
			1B	12	00036	BNEQ	3\$	:	
FD	8F	02	A5	91	00038	CMPB	BUFFER+2, #253	:	0803
			14	1A	0003D	BGTRU	3\$	:	
			23	11	0003F	BRB	4\$	:	0804
000001F4	8F		53	D1	00041	CMPL	STATUS, #500	:	0806
			09	13	00048	BEQL	3\$	:	
			53	DD	0004A	PUSHL	STATUS	:	0807
00000000G	00		01	FB	0004C	CALLS	#1, LIB\$STOP	:	
	B9		52	F5	00053	SOBGTR	J, 1\$	:	0792
		00759008	8F	DD	00056	PUSHL	#7704584	:	0812
00000000G	00		01	FB	0005C	CALLS	#1, LIB\$SIGNAL	:	
				04	00063	RET		:	0811
			54	DD	00064	PUSHL	LBN	:	0820
7E	0000G	CF	54	C3	00066	SUBL3	LBN, DEVICE CHAR+112, -(SP)	:	
	0000V	CF	02	FB	0006C	CALLS	#2, MARK_BAD	:	
		52	04	A5	9E	MOVAB	BUFFER+4, P	:	0822
		53	02	A5	9A	MOVZBL	BUFFER+2, R3	:	0823
		53	02	C6	00079	DIVL2	#2, R3	:	
			53	D6	0007C	INCL	J	:	
			19	11	0007E	BRB	6\$	:	
		54	02	A2	3C	MOVZWL	2(P), LBN	:	0825
54		10		62	F0	INSV	(P), #16, #8, LBN	:	0826
			54	DD	00089	PUSHL	LBN	:	0827
		7E	01	A2	9A	MOVZBL	1(P), -(SP)	:	
			6E	D6	0008F	INCL	(SP)	:	
	0000V	CF	02	FB	00091	CALLS	#2, MARK_BAD	:	
		52	04	C0	00096	ADDL2	#4, P	:	0828
		E4	53	F5	00099	SOBGTR	J, 5\$	:	0823
			04	0009C	RET			:	0831

; Routine Size: 157 bytes, Routine Base: \$CODE\$ + 018B


```

415 0832 1 ROUTINE GET_USERBAD : NOVALUE =
416 0833 1
417 0834 1 ++
418 0835 1
419 0836 1 FUNCTIONAL DESCRIPTION:
420 0837 1
421 0838 1 This routine processes the bad block data entered by the user in the
422 0839 1 command line.
423 0840 1
424 0841 1
425 0842 1 CALLING SEQUENCE:
426 0843 1 GET_USERBAD ()
427 0844 1
428 0845 1 INPUT PARAMETERS:
429 0846 1 NONE
430 0847 1
431 0848 1 IMPLICIT INPUTS:
432 0849 1 device table in INIT_DISK
433 0850 1 parser output database
434 0851 1
435 0852 1 OUTPUT PARAMETERS:
436 0853 1 NONE
437 0854 1
438 0855 1 IMPLICIT OUTPUTS:
439 0856 1 allocation table in INIT_DISK
440 0857 1
441 0858 1 ROUTINE VALUE:
442 0859 1 NONE
443 0860 1
444 0861 1 SIDE EFFECTS:
445 0862 1 disk blocks read
446 0863 1
447 0864 1 --
448 0865 1
449 0866 2 BEGIN
450 0867 2
451 0868 2 LOCAL
452 0869 2 BLOCKFACT, ! blocking factor of disk
453 0870 2 LBN; ! LBN to mark bad
454 0871 2
455 0872 2 EXTERNAL
456 0873 2 DEVICE_CHAR : BBLOCK, ! device characteristics
457 0874 2 BADBLOCK_TABLE : BBLOCKVECTOR [BAD_LENGTH], ! user entered bad block table
458 0875 2
459 0876 2 BADBLOCK_COUNT; ! count of entries
460 0877 2
461 0878 2
462 0879 2 ! Pick up each entry in the bad block table. If it was entered in
463 0880 2 sector - track - cylinder form, convert it to LBN. Enter it in the
464 0881 2 allocation table.
465 0882 2
466 0883 2
467 0884 2 BLOCKFACT = (.DEVICE_CHAR[DIB$B_SECTORS]
468 0885 2 * .DEVICE_CHAR[DIB$B_TRACKS]
469 0886 2 * .DEVICE_CHAR[DIB$W_CYLINDERS])
470 0887 2 / .DEVICE_CHAR[DIB$L_MAXBLOCK];
471 0888 2
```



```

: 472      0889  2 INCR J FROM 0 TO .BADBLOCK_COUNT-1 DO
: 473      0890      BEGIN
: 474      0891      IF .BADBLOCK_TABLE[J, BAD_STC_FORM]
: 475      0892      THEN
: 476      0893          LBN = ((.BADBLOCK_TABLE[J, BAD_CYLINDER] * .DEVICE_CHAR[DIB$B_TRACKS]
: 477      0894          + .BADBLOCK_TABLE[J, BAD_TRACK]) * .DEVICE_CHAR[DIB$B_SECTORS]
: 478      0895          + .BADBLOCK_TABLE[J, BAD_SECTOR]) / .BLOCKFACT
: 479      0896      ELSE
: 480      0897          LBN = .BADBLOCK_TABLE[J, BAD_LBN];
: 481      0898      MARK_BAD (.BADBLOCK_TABLE[J, BAD_COUNT], .LBN);
: 482      0899      END;
: 483      0900
: 484      0901  1 END;

```

! end of routine GET_USERBAD

.EXTRN BADBLOCK_TABLE, BADBLOCK_COUNT

01FC 00000 GET_USERBAD:

	58	0000G	CF	9E	00002	.WORD	Save R2,R3,R4,R5,R6,R7,R8	: 0832
	57	0000G	CF	9E	00007	MOVAB	DEVICE_CHAR+8, R8	
	50		68	9A	0000C	MOVZBL	BADBLOCK_TABLE, R7	
	51	01	A8	9A	0000F	MOVZBL	DEVICE_CHAR+8, R0	: 0885
	50		51	C4	00013	MULL2	DEVICE_CHAR+9, R1	
	52	02	A8	3C	00016	MOVZWL	R1, R0	
	50		52	C4	0001A	MULL2	DEVICE_CHAR+10, R2	: 0886
55	50	68	A8	C7	0001D	DIVL3	R2, R0	
	54	0000G	CF	D0	00022	MOVL	DEVICE_CHAR+112, R0, BLOCKFACT	: 0887
	52		01	CE	00027	MOVL	BADBLOCK_COUNT, R4	: 0889
			49	11	0002A	MNEGL	#1, J	
		06	A742	7F	0002C	BRB	4\$	
2D	9E		00	E1	00030	PUSHAQ	BADBLOCK_TABLE+6[J]	: 0891
		02	A742	7F	00034	BBC	#0, @ (SP)+, 2\$	
	50		9E	3C	00038	PUSHAQ	BADBLOCK_TABLE+2[J]	: 0893
	51	01	A8	9A	0003B	MOVZWL	@ (SP)+, R0	
	50		51	C4	0003F	MOVZBL	DEVICE_CHAR+9, R1	
		01	A742	7F	00042	MULL2	R1, R0	
	56		9E	9A	00046	PUSHAQ	BADBLOCK_TABLE+1[J]	: 0894
	50		56	C0	00049	MOVZBL	@ (SP)+, R6	
	51		68	9A	0004C	ADDL2	R6, R0	
	50		51	C4	0004F	MOVZBL	DEVICE_CHAR+8, R1	
		6742	7F	00052	MULL2	R1, R0		
	56		9E	9A	00055	PUSHAQ	BADBLOCK_TABLE[J]	: 0895
	50		56	C0	00058	MOVZBL	@ (SP)+, R6	
53	50		55	C7	0005B	ADDL2	R6, R0	
			06	11	0005F	DIVL3	BLOCKFACT, R0, LBN	
		6742	7F	00061	BRB	3\$		: 0893
	53		9E	D0	00064	PUSHAQ	BADBLOCK_TABLE[J]	: 0897
			53	DD	00067	MOVL	@ (SP)+, LBN	
		04	A742	7F	00069	PUSHL	LBN	: 0898
	7E		9E	3C	0006D	PUSHAQ	BADBLOCK_TABLE+4[J]	
B3	CF		02	FB	00070	MOVZWL	@ (SP)+, = (SP)	
	52		54	F2	00075	CALLS	#2, MARK_BAD	: 0889
			04	00079	AOBLSS	R4, J, 1\$		: 0901
					RET			

; Routine Size: 122 bytes, Routine Base: \$CODE\$ + 0228

INIBAD
V04-000

H 1
16-Sep-1984 01:43:03
14-Sep-1984 12:35:13

VAX-11 BLISS-32 V4.0-742
DISK\$VMSMASTER:[INIT.SRC]INIBAD.B32;1 Page 15 (5)


```

486 0902 1 ROUTINE MARK_BAD (BLOCK_COUNT, START_LBN) : NOVALUE =
487 0903 1
488 0904 1 ++
489 0905 1
490 0906 1 FUNCTIONAL DESCRIPTION:
491 0907 1
492 0908 1 This routine enters the indicated block(s) into the bad block part
493 0909 1 of the allocation table. The table is maintained in reverse order
494 0910 1 by LBN, and adjacent or overlapping areas are merged. Reverse order
495 0911 1 is used to make the bad block data appear at the front of the volume's
496 0912 1 bad block file.
497 0913 1
498 0914 1
499 0915 1 CALLING SEQUENCE:
500 0916 1 MARK_BAD (ARG1, ARG2)
501 0917 1
502 0918 1 INPUT PARAMETERS:
503 0919 1 ARG1: count of blocks to mark bad
504 0920 1 ARG2: start LBN of blocks
505 0921 1
506 0922 1 IMPLICIT INPUTS:
507 0923 1 allocation table
508 0924 1
509 0925 1 OUTPUT PARAMETERS:
510 0926 1 NONE
511 0927 1
512 0928 1 IMPLICIT OUTPUTS:
513 0929 1 NONE
514 0930 1
515 0931 1 ROUTINE VALUE:
516 0932 1 NONE
517 0933 1
518 0934 1 SIDE EFFECTS:
519 0935 1 allocation table altered
520 0936 1
521 0937 1 --
522 0938 1
523 0939 2 BEGIN
524 0940 2
525 0941 2 LOCAL
526 0942 2 LBN,
527 0943 2 COUNT,
528 0944 2 J,
529 0945 2 C;
530 0946 2
531 0947 2 EXTERNAL
532 0948 2 CLUSTER,
533 0949 2 VOLUME_SIZE,
534 0950 2 BADBLOCK_TOTAL,
535 0951 2 BADBLOCK_LBN : VECTOR,
536 0952 2 BADBLOCK_CNT : VECTOR;
537 0953 2
538 0954 2 EXTERNAL LITERAL
539 0955 2 BADBLOCK_MAX : UNSIGNED (16); ! length of bad block table
540 0956 2
541 0957 2
542 0958 2 ! Round the start LBN and count out to the cluster boundaries surrounding
```



```
543 0959 2 ! the bad area.
544 0960 2 !
545 0961 2 !
546 0962 2 IF .BADBLOCK_TOTAL GEQ BADBLOCK_MAX
547 0963 2 THEN ERR_EXIT (INIT$_MAXBAD);
548 0964 2
549 0965 2 LBN = .START_LBN / .CLUSTER * .CLUSTER;
550 0966 2 COUNT = (.START_LBN + .BLOCK_COUNT + .CLUSTER - 1) / .CLUSTER * .CLUSTER - .LBN;
551 0967 2
552 0968 2 IF .LBN GEQU .VOLUME_SIZE
553 0969 2 THEN ERR_EXIT (INIT$_BADRANGE);
554 0970 2
555 0971 2 ! Search the allocation table until an entry is found with a start LBN lower
556 0972 2 ! than the new LBN. Shuffle the table down at this point and insert the
557 0973 2 ! new entry.
558 0974 2 !
559 0975 2
560 0976 2 J = 0;
561 0977 2 UNTIL .J GEQ .BADBLOCK_TOTAL DO
562 0978 3 BEGIN
563 0979 3 IF .BADBLOCK_LBN[J] LSSU .LBN THEN EXITLOOP;
564 0980 3 J = .J + 1;
565 0981 2 END;
566 0982 2
567 0983 2 CH$MOVE ((.BADBLOCK_TOTAL-.J)*4, BADBLOCK_LBN[J], BADBLOCK_LBN[J+1]);
568 0984 2 CH$MOVE ((.BADBLOCK_TOTAL-.J)*4, BADBLOCK_CNT[J], BADBLOCK_CNT[J+1]);
569 0985 2 BADBLOCK_TOTAL = .BADBLOCK_TOTAL + 1;
570 0986 2 BADBLOCK_CNT[J] = .COUNT;
571 0987 2 BADBLOCK_LBN[J] = .LBN;
572 0988 2
573 0989 2 ! Now check for adjacencies and merge if they exist. Start with the previous
574 0990 2 ! table entry and compare pairs.
575 0991 2 !
576 0992 2
577 0993 2 IF .J NEQ 0 THEN J = .J-1;
578 0994 2 C = 0;
579 0995 2
580 0996 2 UNTIL .J+1 GEQ .BADBLOCK_TOTAL DO
581 0997 3 BEGIN
582 0998 3 IF .BADBLOCK_LBN[J] LEQ .BADBLOCK_LBN[J+1] + .BADBLOCK_CNT[J+1]
583 0999 3 THEN
584 1000 4 BEGIN
585 1001 4 BADBLOCK_CNT[J+1] = MAXU (.BADBLOCK_LBN[J] + .BADBLOCK_CNT[J],
586 1002 4 .BADBLOCK_LBN[J+1] + .BADBLOCK_CNT[J+1])
587 1003 4 - .BADBLOCK_LBN[J+1];
588 1004 4 BADBLOCK_TOTAL = .BADBLOCK_TOTAL - 1;
589 1005 4 CH$MOVE ((.BADBLOCK_TOTAL-.J)*4, BADBLOCK_LBN[J+1], BADBLOCK_LBN[J]);
590 1006 4 CH$MOVE ((.BADBLOCK_TOTAL-.J)*4, BADBLOCK_CNT[J+1], BADBLOCK_CNT[J]);
591 1007 4 BADBLOCK_CNT[.BADBLOCK_TOTAL] = 0;
592 1008 4 END
593 1009 4
594 1010 3 ELSE
595 1011 4 BEGIN
596 1012 4 J = .J + 1;
597 1013 4 C = .C + 1;
598 1014 4 IF .C GEQ 2 THEN EXITLOOP
599 1015 3 END;
```


INIBAD
V04-000

: 600
: 601
: 602

1016 2 END;
1017 2
1018 1 END;

K 1
16-Sep-1984 01:43:03
14-Sep-1984 12:35:13

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[INIT.SRC]INIBAD.B32;1 Page 18
(6)

! end of merge loop
! end of routine MARK_BAD

				.EXTRN	CLUSTER, BADBLOCK TOTAL				
				.EXTRN	BADBLOCK_LBN, BADBLOCK_CNT				
				.EXTRN	BADBLOCK_MAX				
OFFC 00000 MARK_BAD:									
		5B	0000G	CF	9E	00002	.WORD	Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11	: 0902
	00000000G	8F	0000G	CF	D1	00007	MOVAB	BADBLOCK_LBN, R11	: 0962
				0D	19	00010	CMPL	BADBLOCK_TOTAL, #BADBLOCK_MAX	: 0963
			007580BC	8F	DD	00012	BLSS	1\$	: 0965
	00000000G	00		01	FB	00018	PUSHL	#7700668	: 0966
		51	0000G	CF	D0	0001F	CALLS	#1, LIB\$STOP	: 0968
50	08	AC		51	C7	00024	MOVL	CLUSTER, R1	: 0969
58		50		51	C5	00029	DIVL3	R1, START_LBN, R0	: 0970
50	08	AC	04	AC	C1	0002D	MULL3	R1, R0, LBN	: 0971
		50	FF	A140	9E	00033	ADDL3	BLOCK_COUNT, START_LBN, R0	: 0972
		50		51	C6	00038	MOVAB	-1(R1)[R0], R0	: 0973
		50		51	C4	0003B	DIVL2	R1, R0	: 0974
59		50		58	C3	0003E	MULL2	R1, R0	: 0975
	0000G	CF		58	D1	00042	SUBL3	LBN, R0, COUNT	: 0976
				0D	1F	00047	CMPL	LBN, VOLUME_SIZE	: 0977
			00758094	8F	DD	00049	BLSSU	2\$	: 0978
	00000000G	00		01	FB	0004F	PUSHL	#7700628	: 0979
				56	D4	00056	CALLS	#1, LIB\$STOP	: 0980
	0000G	CF		56	D1	00058	CLRL	J	: 0981
		58		0A	18	0005D	CMPL	J, BADBLOCK_TOTAL	: 0982
				6B46	D1	0005F	BGEQ	4\$	: 0983
				04	1F	00063	CMPL	BADBLOCK_LBN[J], LBN	: 0984
				56	D6	00065	BLSSU	4\$	: 0985
				EF	11	00067	INCL	J	: 0986
57	0000G	CF		56	C3	00069	BRB	3\$	: 0987
		57		04	C4	0006F	SUBL3	J, BADBLOCK_TOTAL, R7	: 0988
				04	AB46	DF	MULL2	#4, R7	: 0989
				6B46	DF	00076	PUSHAL	BADBLOCK_LBN+4[J]	: 0990
9E		9E		57	28	00079	PUSHAL	BADBLOCK_LBN[J]	: 0991
			0000G	CF	46	DF	MOVC3	R7, @ (SP)+, @ (SP)+	: 0992
			0000G	CF	46	DF	PUSHAL	BADBLOCK_CNT+4[J]	: 0993
9E		9E		57	28	00087	PUSHAL	BADBLOCK_CNT[J]	: 0994
			0000G	CF	D6	0008B	MOVC3	R7, @ (SP)+, @ (SP)+	: 0995
	0000G	CF		59	D0	0008F	INCL	BADBLOCK_TOTAL	: 0996
		6B46		58	D0	00095	MOVL	COUNT, BADBLOCK_CNT[J]	: 0997
				56	D5	00099	MOVL	LBN, BADBLOCK_LBN[J]	: 0998
				02	13	0009B	TSTL	J	: 0999
				56	D7	0009D	BEQL	5\$	: 1000
				5A	D4	0009F	DECL	J	: 1001
		50		01	A6	9E	CLRL	C	: 1002
	0000G	CF		50	D1	000A5	MOVAB	1(R6), R0	: 1003
				59	18	000AA	CMPL	R0, BADBLOCK_TOTAL	: 1004
			0000G	CF	40	DE	BGEQ	9\$	: 1005
52		6B40		69	C1	000B2	MOVAL	BADBLOCK_CNT[R0], R9	: 1006
		52		6B46	D1	000B7	ADDL3	(R9), BADBLOCK_LBN[R0], R2	: 1007
							CMPL	BADBLOCK_LBN[J], R2	: 1008

51	6B46	0000GCF46	3F 14 000BB	BGTR	8\$		
	52	51 D1 000BD		ADDL3	BADBLOCK_CNT[J], BADBLOCK_LBN[J], R1		1001
		03 1E 000C5		CMPL	R1, R2		1002
	51	52 D0 000CA		BGEQU	7\$		
69	51	6B40 C3 000CD 7\$:		MOVL	R2, R1		
		0000G CF D7 000D2		SUBL3	BADBLOCK_LBN[R0], R1, (R9)		1003
	57	0000G CF D0 000D6		DECL	BADBLOCK_TOTAL		1004
58	57	56 C3 000DB		MOVL	BADBLOCK_TOTAL, R7		1005
	58	04 C4 000DF		SUBL3	J, R7, R8		
		6B46 DF 000E2		MULL2	#4, R8		
		6B40 DF 000E5		PUSHAL	BADBLOCK_LBN[J]		
9E	9E	58 28 000E8		PUSHAL	BADBLOCK_LBN[R0]		
		0000GCF46 DF 000EC		MOVC3	R8, @ (SP)+, @ (SP)+		
9E	69	58 28 000F1		PUSHAL	BADBLOCK_CNT[J]		1006
		0000GCF47 D4 000F5		MOVC3	R8, (R9), @ (SP)+		
		A5 11 000FA		CLRL	BADBLOCK_CNT[R7]		1007
		56 D6 000FC 8\$:		BRB	6\$		0998
		5A D6 000FE		INCL	J		1012
	02	5A D1 00100		INCL	C		1013
		9C 19 00103		CMPL	C, #2		1014
		04 00105 9\$:		BLSS	6\$		
				RET			1018

; Routine Size: 262 bytes, Routine Base: \$CODE\$ + 02A2

:	603	1019	1	
:	604	1020	1	END
:	605	1021	0	ELUDOM

.EXTRN LIB\$SIGNAL, LIB\$STOP

PSECT SUMMARY

Name	Bytes	Attributes
\$CODE\$	936 NOVEC, NOWRT, RD	EXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)
\$OWNS	512 NOVEC, WRT, RD	NOEXE, NOSHR, LCL, REL, CON, NOPIC, ALIGN(2)

Library Statistics

File	Total	Symbols Loaded	Percent	Pages Mapped	Processing Time
_\$255\$DUA28:[SYSLIB]LIB.L32;1	18619	26	0	1000	00:01.9

INIBAD
V04-000

M 1
16-Sep-1984 01:43:03
14-Sep-1984 12:35:13

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[INIT.SRC]INIBAD.B32;1 Page 20
(6)

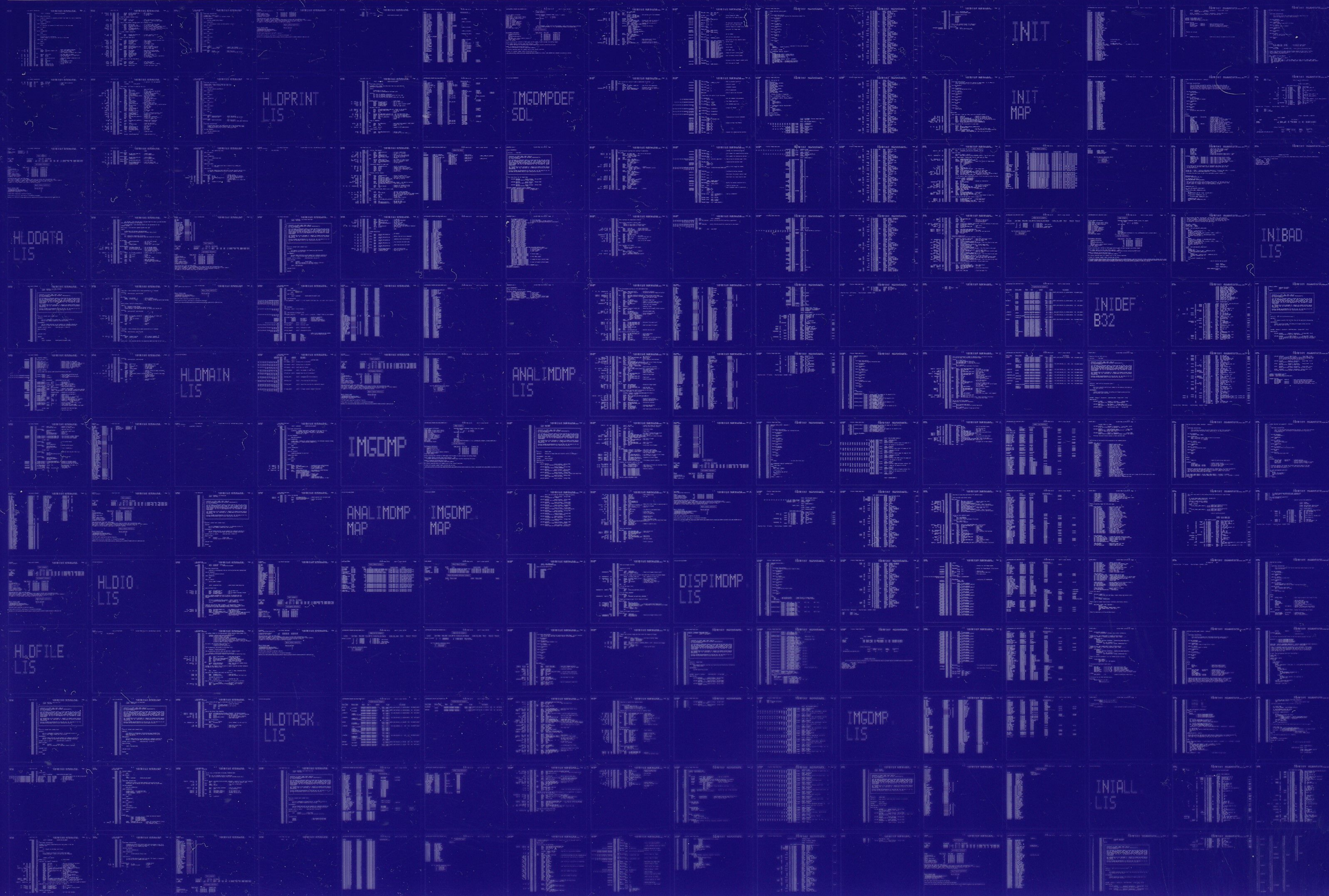
COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:INIBAD/OBJ=OBJ\$:INIBAD MSRC\$:INIBAD/UPDATE=(ENH\$:INIBAD)

: Size: 936 code + 512 data bytes
: Run Time: 00:22.3
: Elapsed Time: 00:46.1
: Lines/CPU Min: 2749
: Lexemes/CPU-Min: 29270
: Memory Used: 151 pages
: Compilation Complete

0186 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY



0187 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

