


```

RRRRRRR      WW      WW      VV      VV      88888888
RRRRRRR      WW      WW      VV      VV      88888888
RR      RR    WW      WW      VV      VV      88      88
RR      RR    WW      WW      VV      VV      88      88
RR      RR    WW      WW      VV      VV      88      88
RR      RR    WW      WW      VV      VV      88      88
RRRRRRR      WW      WW      VV      VV      88888888
RRRRRRR      WW      WW      VV      VV      88888888
RR  RR      WW  WW  WW  VV      VV      88      88
RR  RR      WW  WW  WW  VV      VV      88      88
RR      RR    WWW  WWW  VV  VV      88      88
RR      RR    WWW  WWW  VV  VV      88      88
RR      RR    WW      WW      VV      88888888
RR      RR    WW      WW      VV      88888888

```

```

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

```

```

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

```

```

: 1 0001 0 MODULE RWVB (
: 2 0002 0 LANGUAGE (BLISS32),
:001 3 0003 0 IDENT = 'V04-001',
: 4 -1 0004 0 ) =
: 5 0005 1 BEGIN
: 6 0006 1
: 7 0007 1
: 8 0008 1
: 9 0009 1
:10 0010 1 *
:11 0011 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
:12 0012 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
:13 0013 1 * ALL RIGHTS RESERVED.
:14 0014 1 *
:15 0015 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
:16 0016 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
:17 0017 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
:18 0018 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
:19 0019 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
:20 0020 1 * TRANSFERRED.
:21 0021 1 *
:22 0022 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
:23 0023 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
:24 0024 1 * CORPORATION.
:25 0025 1 *
:26 0026 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
:27 0027 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
:28 0028 1 *
:29 0029 1 *
:30 0030 1 *****
:31 0031 1 ++
:32 0032 1
:33 0033 1 FACILITY: F11ACP Structure Level 1
:34 0034 1
:35 0035 1 ABSTRACT:
:36 0036 1
:37 0037 1 This routine performs the window turn necessary to map a
:38 0038 1 virtual I/O transfer which is not mapped by the current
:39 0039 1 window. It also receives virtual I/O errors for bad block
:40 0040 1 processing.
:41 0041 1
:42 0042 1 ENVIRONMENT:
:43 0043 1
:44 0044 1 STARLET operating system, including privileged system services
:45 0045 1 and internal exec routines.
:46 0046 1
:47 0047 1 --
:48 0048 1
:49 0049 1
:50 0050 1 AUTHOR: Andrew C. Goldstein, CREATION DATE: 7-Jan-1977 00:48
:51 0051 1
:52 0052 1 MODIFIED BY:
:53 0053 1
:001 0054 1 V04-001 CDS0006 Christian D. Saether 10-Nov-1984
:00 0055 1 Deal with non-cluster situation in WRITE_TURN handling.
:00 0056 1
: 0057 1 V03-006 CDS0005 Christian D. Saether 2-Mar-1984

```



```
55      0058 1 |
56      0059 1 |
57      0060 1 |
58      0061 1 |
59      0062 1 |
60      0063 1 |
61      0064 1 |
62      0065 1 |
63      0066 1 |
64      0067 1 |
65      0068 1 |
66      0069 1 |
67      0070 1 |
68      0071 1 |
69      0072 1 |
70      0073 1 |
71      0074 1 |
72      0075 1 |
73      0076 1 |
74      0077 1 |
75      0078 1 |
76      0079 1 |
77      0080 1 |**
78      0081 1 |
79      0082 1 |
80      0083 1 | LIBRARY 'SYSS$LIBRARY:LIB.L32';
81      0084 1 | REQUIRE 'SRC$:FCPDEF.B32';
82      1075 1 |
83      1076 1 |
84      1077 1 | FORWARD ROUTINE
85      1078 1 | READ_WRITEVB : L_NORM, ! main read/write virtual handling
86      1079 1 | MARKBAD_FCB; ! mark bad block in FCB
```

```

: 88      1080 1 GLOBAL ROUTINE READ_WRITEVB : L_NORM =
: 89      1081 1
: 90      1082 1 !++
: 91      1083 1
: 92      1084 1 FUNCTIONAL DESCRIPTION:
: 93      1085 1
: 94      1086 1     This routine performs the window turn necessary to map a
: 95      1087 1     virtual I/O transfer which is not mapped by the current
: 96      1088 1     window. It also receives virtual I/O errors for bad block
: 97      1089 1     processing. These are presently simply returned to the user.
: 98      1090 1
: 99      1091 1 CALLING SEQUENCE:
100      1092 1     READ_WRITEVB ( )
101      1093 1
102      1094 1 INPUT PARAMETERS:
103      1095 1     NONE
104      1096 1
105      1097 1 IMPLICIT INPUTS:
106      1098 1     IO_PACKET: I/O packet of request
107      1099 1
108      1100 1 OUTPUT PARAMETERS:
109      1101 1     NONE
110      1102 1
111      1103 1 IMPLICIT OUTPUTS:
112      1104 1     NONE
113      1105 1
114      1106 1 ROUTINE VALUE:
115      1107 1     1 if request requeued to driver
116      1108 1     0 if error
117      1109 1
118      1110 1 SIDE EFFECTS:
119      1111 1     window turned
120      1112 1     request requeued to driver if mapped
121      1113 1
122      1114 1 --
123      1115 1
124      1116 2 BEGIN
125      1117 2
126      1118 2 LOCAL
127      1119 2     PACKET          : REF BBLOCK,  ! pointer to I/O packet
128      1120 2     WINDOW          : REF BBLOCK,  ! file window
129      1121 2     FCB              : REF BBLOCK,  ! file FCB
130      1122 2     BLOCK_COUNT,    ! number of blocks in transfer
131      1123 2     UNMAPPED,        ! number of blocks not mapped
132      1124 2     MODE,          ! mode (read/write) of transfer
133      1125 2     VBN,          ! starting VBN of transfer
134      1126 2     LBN,          ! translated LBN
135      1127 2     LAST_LBN;      ! highest LBN touched by operation
136      1128 2
137      1129 2 BIND_COMMON;
138      1130 2
:001 :CDS0006 1131 2 EXTERNAL
:002 :CDS0006 1132 2     CLUSGL_CLUB      : ADDRESSING_MODE (GENERAL);
:003 :CDS0006 1133 2
:004 :CDS0006 1134 2 EXTERNAL ROUTINE
:005 :CDS0006 1135 2     KILL_BUFFERS    : L_NORM NOVALUE, ! invalidate buffers
: 140-1 1136 2     UNLOCK_XQP      : L_NORM,      ! unlock xqp and release cache

```



```

: 141 1137 SERIAL_FILE : L_NORM, : serialize file operations
: 142 1138 ALLOCATION_LOCK : L_NORM, : get volume allocation lock
: 143 1139 MAP_VBN : L_NORM, : map and turn window
: 144 1140 REQUEUE_REQ : L_NORM, : requeue request to driver
: 145 1141 SCAN_BADLOG : L_NORM; : scan bad block log file
: 146 1142
:001 :CDS0006 1143 LABEL
:002 :CDS0006 1144 INDEXF;
: 147 1145
: 148 1146 : Extract the request parameters from the I/O packet. Compute VBN and LBN
: 149 1147 : of the next block to be transferred.
: 150 1148
: 151 1149
: 152 1150 PACKET = .IO_PACKET;
: 153 1151 WINDOW = .PACKET[IRPSL_WIND];
: 154 1152 BLOCK_COUNT = (.PACKET[IRPSW_BCNT]+511) / 512;
: 155 1153 VBN = .PACKET[IRPSL_SEGVBN];
: 156 1154
: 157 1155 IF .VBN EQL 0 THEN ERR_EXIT (SS$_BADPARAM);
: 158 1156
: 159 1157 FCB = .WINDOW[WCBSL_FCB];
: 160 1158
: 161 1159 : Serialize further processing on this file.
: 162 1160
: 163 1161
: 164 1162 PRIM_LCKINDX = SERIAL_FILE (FCB [FCBSW_FID]);
: 165 1163
: 166 1164 : Attempt to map the request. If the map fails, report
: 167 1165 : failure. Else requeue the request to the driver.
: 168 1166
: 169 1167
: 170 1168 LBN = MAP_VBN (.VBN, .WINDOW, .BLOCK_COUNT, UNMAPPED);
: 171 1169 IF .LBN EQL -1 THEN ERR_EXIT (SS$_ENDOFFILE);
: 172 1170
: 173 1171 IF .PACKET[IRPSV_VIRTUAL]
: 174 1172 THEN
: 175 1173 BEGIN
: 176 1174 LAST_LBN = .LBN + (.BLOCK_COUNT - .UNMAPPED - 1);
: 177 1175 IF .LBN GEQU .CURRENT_UCB[UCBSL_MAXBLOCK]
: 178 1176 OR .LAST_LBN GEQU .CURRENT_UCB[UCBSL_MAXBLOCK]
: 179 1177 THEN ERR_EXIT (SS$_ILLBLKNUM);
: 180 1178
:001 :CDS0006 1179 : If WRITE_TURN is set, this is a request to write a block into a file
:002 :CDS0006 1180 : that the file system has some interest in. Specifically, those files
:003 :CDS0006 1181 : which may have blocks cached in the file system cache.
:004 :CDS0006 1182 : What will happen depends on whether this is a cluster or not.
:005 :CDS0006 1183 : If we are in a cluster, there are locks backing these buffers and
:006 :CDS0006 1184 : to invalidate them throughout the cluster we must bump the sequence
:007 :CDS0006 1185 : number for the backing lock.
:008 :CDS0006 1186 : If we are not in a cluster, call a routine which will scan the buffer
:009 :CDS0006 1187 : descriptors and toss those buffers that are relevant.
:010 :CDS0006 1188
:011 :CDS0006 1189
:012 :CDS0006 1190 IF .WINDOW [WCBSV_WRITE_TURN]
:013 :CDS0006 1191 THEN
:014 :CDS0006 1192 BEGIN
:015 :CDS0006 1193 LOCAL

```



```

:016 :CDS0006 1194 4      CLUSTER;
:017 :CDS0006 1195 4
:018 :CDS0006 1196 4      CLUSTER = 0;
:019 :CDS0006 1197 4
:020 :CDS0006 1198 4      IF .BBLOCK [CURRENT_UCB [UCB$$_DEVCHAR2], DEV$$_CLU]
:021 :CDS0006 1199 4      AND .CLUSGL_CLUB NEQ 0
:022 :CDS0006 1200 4      THEN
:023 :CDS0006 1201 4      CLUSTER = 1;
:024 :CDS0006 1202 4
:025 :CDS0006 1203 4      IF .FCB [FCB$$_FID_NMX] NEQ 0
:026 :CDS0006 1204 4      OR .FCB [FCB$$_FID_NUM] GTRU 2
:027 :CDS0006 1205 4      THEN
:028 :CDS0006 1206 5      BEGIN
:029 :CDS0006 1207 5      IF .CLUSTER
:030 :CDS0006 1208 5      THEN
186-5 1209 5
187 1210 5 ! This is not the index file or bitmap file, so we assume it
188 1211 5 ! is a directory.  Simply bump the data sequence number.  The
189 1212 5 ! unlock_xqp will store the updated value block.
190 1213 5
191 1214 5
:001 :CDS0006 1215 5      LB_DATASEQ [.PRIM_LCKINDX] = .LB_DATASEQ [.PRIM_LCKINDX] + 1
:002 :CDS0006 1216 5      ELSE
:003 :CDS0006 1217 6      BEGIN
:004 :CDS0006 1218 6
:005 :CDS0006 1219 6 ! Invalidate all directory data blocks and directory index blocks.
:006 :CDS0006 1220 6
:007 :CDS0006 1221 6
:008 :CDS0006 1222 6      KILL_BUFFERS (1, .LB_BASIS [.PRIM_LCKINDX]);
:009 :CDS0006 1223 6      KILL_BUFFERS (3, .LB_BASIS [.PRIM_LCKINDX]);
:010 :CDS0006 1224 5      END;
:011 :CDS0006 1225 5      ELSE
193-1 1226 4
194 1227 5      BEGIN
195 1228 5      IF .FCB [FCB$$_FID_NUM] EQL 1
196 1229 5      THEN
197 1230 5
198 1231 5 ! This is the index file.  Determine if the VBN being written is within
199 1232 5 ! the index file bitmap or in the header area, and increment the appropriate
200 1233 5 ! sequence number to invalidate cached buffers.
201 1234 5
202 1235 5
:001 :CDS0006 1236 6 INDEXF:      BEGIN
:002 :CDS0006 1237 6      LOCAL
:003 :CDS0006 1238 6      HDRBASE;
:004 :CDS0006 1239 6
:005 :CDS0006 1240 5      IF NOT .CLUSTER
:006 :CDS0006 1241 6      THEN
:007 :CDS0006 1242 6
:008 :CDS0006 1243 6 ! This is a bit of the sledgehammer approach.  Wipe out all buffers
:009 :CDS0006 1244 6 ! cached for the index file, period.  This could be refined to
:010 :CDS0006 1245 6 ! selectively hit only those buffers modified, but that routine
:011 :CDS0006 1246 6 ! does not exist right now.  It is also not clear the effort is worth
:012 :CDS0006 1247 6 ! it, because this logic is typically invoked when disk rebuild
:013 :CDS0006 1248 6 ! is rewriting headers, which is does not do very many of, and while
:014 :CDS0006 1249 6 ! it is blocking all disk activity anyway.
:015 :CDS0006 1250 6

```



```
:016 :CDS0006 1251 6
:017 :CDS0006 1252 7
:018 :CDS0006 1253 7
:019 :CDS0006 1254 7
:020 :CDS0006 1255 6
:206-3 1256 6
:207 1257 6
:208 1258 6
:209 1259 6
:210 1260 6
:211 1261 6
:212 1262 6
:213 1263 7
:214 1264 6
:215 1265 7
:216 1266 7
:217 1267 7
:218 1268 7
:219 1269 7
:220 1270 7
:221 1271 7
:222 1272 7
:223 1273 7
:224 1274 7
:225 1275 7
:226 1276 7
:227 1277 7
:228 1278 7
:229 1279 8
:230 1280 8
:231 1281 8
:232 1282 8
:233 1283 7
:234 1284 7
:235 1285 7
:236 1286 7
:237 1287 7
:238 1288 7
:239 1289 7
:240 1290 7
:241 1291 7
:242 1292 7
:243 1293 7
:244 1294 8
:245 1295 8
:246 1296 8
:247 1297 8
:248 1298 8
:249 1299 8
:250 1300 8
:251 1301 8
:252 1302 8
:253 1303 7
:254 1304 6
:255 1305 6
:256 1306 5
:257 1307 5

BEGIN
KILL BUFFERS (2, 0);
LEAVE INDEXF;
END;

HDRBASE = .CURRENT_VCB [VCB$W_CLUSTER]*4
+ .CURRENT_VCB [VCB$B_IBMAPSIZE];

! Loop for all blocks being written.

INCR I FROM 0 TO (.BLOCK_COUNT - 1)
DO
  BEGIN
  LOCAL
  FID : BBLOCK [6];

  UNLOCK_XQP ();

  FID = .VBN + .I;

  IF .FID LEQU .HDRBASE
  THEN
    ! This is not within the header area. Assume it is the index file bitmap.

    BEGIN
    ALLOCATION_LOCK ();
    (LB_DATASEQ [0])<16,16> = .(LB_DATASEQ [0])<16,16> + 1;
    END
  ELSE
    ! This is within the header area. Calculate the file number and bump
    ! it's sequence number. Note that extension headers are locked under
    ! the primary header's file number, and hence this does not directly
    ! invalidate them. However, we will assume that whatever is out there
    ! mucking with headers in the first place will know enough to rewrite
    ! the primary header when screwing with extension headers and get them
    ! in that fashion.

    BEGIN
    LOCAL
    LCKINDX;

    FID = .FID - .HDRBASE;
    FID [FID$B_NMX] = .FID<16,8>;
    FID [FID$B_RVN] = .CURRENT_RVN;
    LCKINDX = SERIAL_FILE (FID);
    LB_HDRSEQ [.LCKINDX] = .LB_HDRSEQ [.LCKINDX] + 1;
    END;
  END;
ELSE
  END;
! of loop through all blocks
! of this is file number 1
```



```

: 258      1308  5  ! This is for file number 2, which is the storage bitmap. Invalidate
: 259      1309  5  ! entries cached for it.
: 260      1310  5  !
: 261      1311  5  !
: 262      1312  6  !
:001 CDS0006 1313  6  !
:002 CDS0006 1314  6  !
:003 CDS0006 1315  6  !
:004 CDS0006 1316  6  !
:005 CDS0006 1317  6  !
:006 CDS0006 1318  7  !
:007 CDS0006 1319  7  !
:008 CDS0006 1320  7  !
:009 CDS0006 1321  6  !
:010 CDS0006 1322  6  !
:011 CDS0006 1323  4  !
:012 CDS0006 1324  4  !
267-4     1325  4  !
268       1326  4  !
269       1327  4  !
270       1328  4  !
271       1329  4  !
272       1330  4  !
273       1331  4  !
274       1332  4  !
275       1333  4  !
276       1334  4  !
277       1335  4  !
278       1336  4  !
279       1337  4  !
280       1338  4  !
281       1339  4  !
282       1340  4  !
283       1341  4  !
284       1342  4  !
285       1343  4  !
286       1344  4  !
287       1345  4  !
288       1346  4  !
289       1347  4  !
290       1348  4  !
291       1349  4  !
292       1350  4  !
293       1351  4  !
294       1352  4  !
295       1353  5  !
296       1354  5  !
297       1355  4  !
298       1356  3  !
299       1357  4  !
300       1358  4  !
301       1359  4  !
302       1360  4  !
303       1361  4  !
304       1362  4  !
305       1363  3  !
306       1364  3  !

```

```

BEGIN
  IF NOT .CLUSTER
  THEN
    KILL_BUFFERS (0,0)
  ELSE
    BEGIN
      ALLOCATION_LOCK ();
      (LB_DATASEQ [0])<0,16> = .(LB_DATASEQ [0])<0,16> + 1;
    END;
  END;
END;
! of either file 1 or 2.
! of WRITE_TURN

UNLOCK_XQP ();
KERNEL_CALL (REQUEUE_REQ, .PACKET, .LBN, .UNMAPPED);
RETURN 1;
END

! If the virtual bit is not set, this is an I/O error on a file sent here
! for bad block processing. If the error is a parity, format, or datacheck
! error, we set the bad block bit in the FCB of the file and enter the
! block in question into the volume's bad block log. Note that we do not
! do this on errors on the volume's reserved files, which are not subject
! to dynamic bad block processing.

ELSE
  BEGIN
    USER_STATUS[0] = .PACKET[IRPSL_IOST1];      ! get status to return to user
    USER_STATUS[1] = .PACKET[IRPSL_IOST2];
  IF
    NOT .BBLOCK [CURRENT_UCB[UCBSL_DEVCHAR], DEV$V_SWL]
  AND (
    .(PACKET[IRPSL_IOST1])<0,16> EQL SSS_PARITY
  OR .(PACKET[IRPSL_IOST1])<0,16> EQL SSS_DATACHECK
  OR .(PACKET[IRPSL_IOST1])<0,16> EQL SSS_FORMAT
  )
  AND (
    .FCB[FCBSW_FID_NUM] GTRU .CURRENT_VCB[VCBSB_RESFILES]
  OR (.CURRENT_VCB[VCBSV_EXTFID]
    AND .FCB[FCBSB_FID_NMX] NEQ 0)
  )
  THEN
    BEGIN
      KERNEL_CALL (MARKBAD_FCB, .FCB);
      MODE = ENTER_READERR;      ! assume read
      IF .PACKET[IRPSV_FCODE] EQL IOS_WRITEPBLK
      THEN MODE = ENTER_WRITERR;
      SCAN_BADLOG (FCB[FCBSW_FID], .VBN, .LBN, .MODE, 0);
    END;
  RETURN 0;

```

: 307
: 308
: 309
1365 2 END;
1366 2
1367 1 END;

! end of routine READ_WRITEVB

						.TITLE	RWVB		
						.IDENT	\V04-001\		
						.EXTRN	CLUSGL_CLUB, KILL_BUFFERS		
						.EXTRN	UNLOCK_XQP, SERIAL_FILE		
						.EXTRN	ALLOCATION_LOCK		
						.EXTRN	MAP_VBN, REQUEUE_REQ		
						.EXTRN	SCAN_BADLOG		
						.PSECT	\$CODE\$,NOWRT,2		
						.ENTRY	READ_WRITEVB, Save R2,R3,R4,R5,R6,R7,R8,R9	1080	
						SUBL2	#12, SP		
						MOVAB	-128(BASE), R8	1127	
						MOVAB	168(BASE), R6		
						MOVL	-112(BASE), PACKET	1150	
						MOVL	24(PACKET), WINDOW	1151	
						MOVZWL	50(PACKET), R0	1152	
						MOVAB	511(R0), R0		
52						DIVL3	#512, R0, BLOCK_COUNT		
						MOVL	72(PACKET), VBN	1153	
						BNEQ	1\$	1155	
						CHMU	#20		
						RET			
						MOVL	24(WINDOW), FCB	1157	
						PUSHAB	36(FCB)	1162	
						CALLS	#1, SERIAL_FILE		
						MOVL	R0, 24(BASE)		
						PUSHR	#*M<R2,SP>	1168	
						PUSHL	WINDOW		
						PUSHL	VBN		
						CALLS	#4, MAP_VBN		
						MOVL	R0, LBN		
						CMPL	LBN, #-1	1169	
						BNEQ	2\$		
						CHMU	#2160		
						RET			
						BBS	#4, 42(PACKET), 3\$	1171	
						BRW	20\$		
						SUBL3	UNMAPPED, BLOCK_COUNT, R0	1174	
						MOVAB	-1(R0)[LBN], LAST_LBN		
						MOVL	-108(BASE), R0	1175	
						CMPL	LBN, 176(R0)		
						BGEQU	4\$		
						CMPL	LAST_LBN, 176(R0)	1176	
						BLSSU	5\$		
						CHMU	#220	1177	
						RET			
						BBC	#4, 21(WINDOW), 8\$	1190	
						CLRL	CLUSTER	1196	
						MOVL	-108(BASE), R1	1198	
						BLBC	60(R1), 6\$		

		00000000G	00	D5	00095	TSTL	CLUSGL_CLUB	1199	
			03	13	0009B	BEQL	6\$		
		50	01	D0	0009D	MOVL	#1, CLUSTER	1201	
			29	A3	95 000A0	6\$: TSTB	41(FCB)	1203	
			06	12	000A3	BNEQ	7\$		
		02	24	A3	B1 000A5	CMPW	36(FCB), #2	1204	
			29	1B	000A9	BLEQU	10\$		
		09	50	E9	000AB	7\$: BLBC	CLUSTER, 9\$	1207	
		50	18	AA	D0 000AE	MOVL	24(BASE), R0	1215	
			6640	D6	000B2	INCL	(R6)[R0]		
			7C	11	000B5	8\$: BRB	15\$		
		50	18	AA	D0 000B7	9\$: MOVL	24(BASE), R0	1222	
			0080	CA40	DD 000BB	PUSHL	128(BASE)[R0]		
			01	DD	000C0	PUSHL	#1		
	0000G	CF	02	FB	000C2	CALLS	#2, KILL_BUFFERS		
		50	18	AA	D0 000C7	MOVL	24(BASE), R0	1223	
			0080	CA40	DD 000CB	PUSHL	128(BASE)[R0]		
			03	DD	000D0	PUSHL	#3		
			66	11	000D2	BRB	17\$		
		50	50	D2	000D4	10\$: MCOML	CLUSTER, R0	1240	
		01	24	A3	B1 000D7	CMPW	36(FCB), #1	1228	
			58	12	000DB	BNEQ	16\$		
		05	50	E9	000DD	BLBC	R0, 11\$	1240	
		7E	02	7D	000E0	MOVQ	#2, -(SP)	1253	
			55	11	000E3	BRB	17\$		
		50	98	AA	D0 000E5	11\$: MOVL	-104(BASE), R0	1257	
		51	3C	A0	3C 000E9	MOVZWL	60(R0), R1		
		50	38	A0	9A 000ED	MOVZBL	56(R0), R0	1258	
		58	6041	DE	000F1	MOVAL	(R0)[R1], HDRBASE		
		53	01	CE	000F5	MNEGL	#1, I	1263	
			35	11	000F8	BRB	14\$		
	04	AE	0000G	CF	00	FB 000FA	12\$: CALLS	#0, UNLOCK_XQP	1269
			59	53	C1 000FF	ADDL3	I, VBN, FID	1271	
			58	04	AE D1 00104	CMP	FID, HDRBASE	1273	
				0A	1A 00108	BGTRU	13\$		
			0000G	CF	00	FB 0010A	CALLS	#0, ALLOCATION_LOCK	1280
				02	A6 B6 0010F	INCW	2(R6)	1281	
				1B	11 00112	BRB	14\$	1273	
			04	AE	58	C2 00114	13\$: SUBL2	HDRBASE, FID	1298
			09	AE	06	AE 90 00118	MOVB	FID+2, FID+5	1299
			08	AE	A0	AA 90 0011D	MOVB	-96(BASE), FID+4	1300
				04	AE 9F 00122	PUSHAB	FID	1301	
			0000G	CF	01	FB 00125	CALLS	#1, SERIAL FILE	
				0094	CA40	D6 0012A	INCL	148(BASE)[[CKINDX]]	1302
				53	52	F2 0012F	14\$: AOBLS	R2, I, 12\$	1262
					13	11 00133	15\$: BRB	19\$	1228
				09	50	E9 00135	16\$: BLBC	R0, 18\$	1314
					7E	7C 00138	CLRQ	-(SP)	1316
			0000G	CF	02	FB 0013A	17\$: CALLS	#2, KILL_BUFFERS	
					07	11 0013F	BRB	19\$	
			0000G	CF	00	FB 00141	18\$: CALLS	#0, ALLOCATION_LOCK	1319
					66	B6 00146	INCW	(R6)	1320
			0000G	CF	00	FB 00148	19\$: CALLS	#0, UNLOCK_XQP	1326
					6E	DD 0014D	PUSHL	UNMAPPED	1327
				0090	8F	BB 0014F	PUSHR	#*M<R4, R7>	
			0000G	CF	03	FB 00153	CALLS	#3, REQJEUE_REQ	
				50	01	D0 00158	MOVL	#1, R0	1340

		68	38	A4	04	0015B	20\$:	RET			
		50	94	AA	7D	0015C		MOVQ	56(PACKET), (R8)	:	1341
	55	A0		01	D0	00160		MOVL	-108(BASE), R0	:	1345
		8F	38	A4	E0	00164		BBS	#1, 59(R0), 24\$:	
				10	B1	00169		CMPW	56(PACKET), #500	:	1347
		8F	38	A4	13	0016F		BEQL	21\$:	
				08	B1	00171		CMPW	56(PACKET), #92	:	1348
		8F	38	A4	13	00177		BEQL	21\$:	
				08	B1	00179		CMPW	56(PACKET), #188	:	1349
		50	98	AA	3D	0017F	21\$:	BNEQ	24\$:	
		51	4F	A0	D0	00181		MOVL	-104(BASE), R0	:	1352
		A3		51	9A	00185		MOVZBL	79(R0), R1	:	
	2A			0A	B1	00189		CMPW	R1, 36(FCB)	:	
		A0		05	1F	0018D		BLSSU	22\$:	
			29	A3	E1	0018F		BBC	#5, 11(R0), 24\$:	1353
				25	95	00194		TSTB	41(FCB)	:	1354
				53	13	00197		BEQL	24\$:	
		CF		01	DD	00199	22\$:	PUSHL	FCB	:	1358
		50		01	FB	0019B		CALLS	#1, MARKBAD_FCB	:	
OB	20	06		00	D0	001A0		MOVL	#1, MODE	:	1359
				00	ED	001A3		CMPZV	#0, #6, 32(PACKET), #11	:	1360
		50		03	12	001A9		BNEQ	23\$:	
				02	D0	001AB		MOVL	#2, MODE	:	1361
				7E	D4	001AE	23\$:	CLRL	-(SP)	:	1362
				50	DD	001B0		PUSHL	MODE	:	
				57	DD	001B2		PUSHL	LBN	:	
				59	DD	001B4		PUSHL	VBN	:	
			24	A3	9F	001B6		PUSHAB	36(FCB)	:	
		CF		05	FB	001B9		CALLS	#5, SCAN_BADLOG	:	
				50	D4	001BE	24\$:	CLRL	R0	:	1364
				04	001C0			RET		:	1367

; Routine Size: 449 bytes, Routine Base: \$CODE\$ + 0000


```

1368 1 GLOBAL ROUTINE MARKBAD_FCB (FCB) =
1369 1
1370 1 !++
1371 1
1372 1 FUNCTIONAL DESCRIPTION:
1373 1
1374 1     This routine set the bad block bit in the indicated FCB.
1375 1
1376 1
1377 1 CALLING SEQUENCE:
1378 1     MARKBAD_FCB (ARG1)
1379 1
1380 1 INPUT PARAMETERS:
1381 1     ARG1: address of FCB
1382 1
1383 1 IMPLICIT INPUTS:
1384 1     NONE
1385 1
1386 1 OUTPUT PARAMETERS:
1387 1     NONE
1388 1
1389 1 IMPLICIT OUTPUTS:
1390 1     NONE
1391 1
1392 1 ROUTINE VALUE:
1393 1     1
1394 1
1395 1 SIDE EFFECTS:
1396 1     bad bit set in FCB
1397 1
1398 1 !--
1399 1
1400 2 BEGIN
1401 2
1402 2 MAP
1403 2     FCB           : REF BBLOCK;   ! FCB argument
1404 2
1405 2
1406 2 FCB[FCB$V_BADBLK] = 1;
1407 2
1408 2 RETURN 1;
1409 2
1410 1 END;

```

! end of routine MARKBAD_FCB

				0000 0000	.ENTRY MARKBAD_FCB, Save nothing	: 1368
	22	50	04	AC D0 00002	MOVL FCB, R0	: 1406
		A0		04 88 00006	BISB2 #4, 34(R0)	:
		50		01 D0 0000A	MOVL #1, R0	: 1408
				04 0000D	RET	: 1410

; Routine Size: 14 bytes, Routine Base: \$CODE\$ + 01C1

RWVB
V04-001

J 6
8-Jan-1985 18:31:43
2-Oct-1984 12:43:38

VAX-11 Bliss-32 V4.0-742 Page 12
DISK\$VMMASTER:[F11X.BUGSRC]RWVB.B32;1 (3)

: 354 1411 1
: 355 1412 1 END
: 356 1413 0 ELUDOM

PSECT SUMMARY

Name Bytes Attributes
\$CODE\$ 463 NOVEC,NOWRT, RD , EXE,NOSHR, LCL, REL, CON,NOPI,ALIGN(2)

Library Statistics

File	Total	Symbols Loaded	Percent	Page Mapped	Processing Time
_\$255\$DUA18:[SYSLIB]LIB.L32;1	18619	45	0	1000	00:02.0

COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LISS:RWVB/OBJ=OBJ\$:RWVB MSRC\$:RWVB/UPDATE=(BUG\$:RWVB)

: Size: 463 code + 0 data bytes
: Run Time: 00:24.7
: Elapsed Time: 00:46.7
: Lines/CPU Min: 3439
: Lexemes/CPU-Min: 38738
: Memory Used: 290 pages
: Compilation Complete

0444 AH-EF71A-SE
VAX/VMS V4.1 SRC LST MCRF UPD

Grid of source code listings for VAX/VMS V4.1 MCRF UPD. The grid contains numerous small windows, each displaying a different module's source code. Key modules identified include:

- LTDRIVER MAP (top right)
- LTDRIVER LIS (middle right)
- TRUNC LIS (middle right)
- RWUB LIS (middle left)
- SMALOC LIS (middle left)
- SNDER LIS (middle left)
- LAT (bottom right)

The source code within these windows is dense and follows a standard VAX/VMS listing format, including headers, comments, and assembly-like instructions.