```
                                      I 6

   CCCCCCCCCCCC     OOOOOOOOO     NNN         NNN  VVV         VVV
   CCCCCCCCCCCC     OOOOOOOOO     NNN         NNN  VVV         VVV
   CCCCCCCCCCCC     OOOOOOOOO     NNN         NNN  VVV         VVV
CCC              OOO         OOO  NNN         NNN  VVV         VVV
CCC              OOO         OOO  NNN         NNN  VVV         VVV
CCC              OOO         OOO  NNN         NNN  VVV         VVV
CCC              OOO         OOO  NNNNNN      NNN  VVV         VVV
CCC              OOO         OOO  NNNNNN      NNN  VVV         VVV
CCC              OOO         OOO  NNNNNN      NNN  VVV         VVV
CCC              OOO         OOO  NNN   NNN   NNN  VVV         VVV
CCC              OOO         OOO  NNN   NNN   NNN  VVV         VVV
CCC              OOO         OOO  NNN   NNN   NNN  VVV         VVV
CCC              OOO         OOO  NNN      NNNNNN  VVV         VVV
CCC              OOO         OOO  NNN      NNNNNN  VVV         VVV
CCC              OOO         OOO  NNN      NNNNNN  VVV         VVV
CCC              OOO         OOO  NNN         NNN     VVV   VVV
CCC              OOO         OOO  NNN         NNN     VVV   VVV
CCC              OOO         OOO  NNN         NNN     VVV   VVV
   CCCCCCCCCCCC     OOOOOOOOO     NNN         NNN        VVV
   CCCCCCCCCCCC     OOOOOOOOO     NNN         NNN        VVV
   CCCCCCCCCCCC     OOOOOOOOO     NNN         NNN        VVV
```

```
**FILE**ID**RECLREC                                    D 12



RRRRRRRR    EEEEEEEEEE    CCCCCCCC  LL          RRRRRRRR    EEEEEEEEEE    CCCCCCCC
RRRRRRRR    EEEEEEEEEE    CCCCCCCC  LL          RRRRRRRR    EEEEEEEEEE    CCCCCCCC
RR      RR  EE          CC          LL          RR      RR  EE          CC
RR      RR  EE          CC          LL          RR      RR  EE          CC
RR      RR  EE          CC          LL          RR      RR  EE          CC
RR      RR  EE          CC          LL          RR      RR  EE          CC
RRRRRRRR    EEEEEEEE    CC          LL          RRRRRRRR    EEEEEEEE    CC
RRRRRRRR    EEEEEEEE    CC          LL          RRRRRRRR    EEEEEEEE    CC
RR  RR      EE          CC          LL          RR  RR      EE          CC
RR  RR      EE          CC          LL          RR  RR      EE          CC
RR    RR    EE          CC          LL          RR    RR    EE          CC              ....
RR    RR    EE          CC          LL          RR    RR    EE          CC              ....
RR      RR  EEEEEEEEEE    CCCCCCCC  LLLLLLLLLL  RR      RR  EEEEEEEEEE    CCCCCCCC      ....
RR      RR  EEEEEEEEEE    CCCCCCCC  LLLLLLLLLL  RR      RR  EEEEEEEEEE    CCCCCCCC      ....

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
```

```
0001  0 %TITLE  'VAX-11 CONVERT/RECLAIM'
0002  0 MODULE  RECL$REC            ( IDENT='V04-000',
0003  0                             OPTLEVEL=3
0004  0                             ) =
0005  0
0006  1 BEGIN
0007  1
0008  1 !****************************************************************************
0009  1 !*                                                                          *
0010  1 !*   COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                *
0011  1 !*   DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                 *
0012  1 !*   ALL RIGHTS RESERVED.                                                   *
0013  1 !*                                                                          *
0014  1 !*   THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED  *
0015  1 !*   ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE  *
0016  1 !*   INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER  *
0017  1 !*   COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY  *
0018  1 !*   OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY  *
0019  1 !*   TRANSFERRED.                                                           *
0020  1 !*                                                                          *
0021  1 !*   THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE  *
0022  1 !*   AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT  *
0023  1 !*   CORPORATION.                                                           *
0024  1 !*                                                                          *
0025  1 !*   DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS  *
0026  1 !*   SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                *
0027  1 !*                                                                          *
0028  1 !*                                                                          *
0029  1 !****************************************************************************
```

```
1 !++
1 !
1 ! Facility:      VAX-11 CONVERT/RECLAIM
1 !
1 ! Environment:
1 !
1 !               VAX/VMS Operating system
1 !
1 ! Abstract:
1 !
1 !               This module contains routines to handle index records.
1 !
1 ! Contents:
1 !               BUCKET_EMPTY
1 !               SQUISH_PRIMARY_BUCKET
1 !               SQUISH_SIDR_BUCKET
1 !               GET_DOWN_POINTER
1 !               COMPARE_POINTER
1 !               SWING_POINTER
1 !               REMOVE_INDEX_RECORD
1 !               RECOMPRESS_RECORD
1 !
1 !--
1
1 !
1 ! Author:       Peter Lieberwirth       Creation Date:   2-Sep-1981
1 !
1 ! Modified by:
1 !
1 !       V03-009 TMK0001         Todd M. Katz            03-Feb-1983
1 !               Add support for Recovery Unit Journalling and RU ROLLBACK
1 !               Recovery of ISAM files.
1 !
1 !               The routine SQUISH_PRIMARY_BUCKET has been modified to squish
1 !               primary data records that are marked RU_DELETE and re-format
1 !               primary data records that have been marked RU_UPDATE.
1 !
1 !               The routine SQUISH_SIDR_BUCKET has been modified to squish
1 !               SIDR array elements that are marked RU_DELETE.
1 !
1 !               NOTE: The routine SQUISH_SIDR_BUCKET is algorithmically wrong.
1 !               It doesn't squish out anything! I plan on leaving it the way it
1 !               is until a massive re-write can be done.
1 !
1 !       V03-008 KBT0396         Keith B. Thompson       2-Nov-1982
1 !               Fix some bugs in squish_primary_bucket and squish_sidr_bucket
1 !
1 !       V03-007 KBT0389         Keith B. Thompson       28-Oct-1982
1 !               Add support for prologue 3 sidrs and do record level
1 !               space reclamition
1 !
1 !       V03-006 KBT0357         Keith B. Thompson       6-Oct-1982
1 !               Use new merged ctx definitions
1 !
1 !       V03-005 KBT0354         Keith B. Thompson       5-Oct-1982
1 !               Use new linkage definitions
1 !
```

```
1 !       V03-004 KBT0049         Keith Thompson          21-Apr-1982
1 !               Add routine to check for last index record in bucket
1 !
1 !       V03-003 KBT0046         Keith Thompson          12-Apr-1982
1 !               Fix compression bug and increase the key buffers to 257 bytes
1 !
1 !       V03-002 KBT0042         Keith Thompson          3-Apr-1982
1 !               Add routines to compare and swing index pointers
1 !
1 !       V03-001 KBT0026         Keith Thompson          29-Mar-1982
1 !               Do not reclaim data buckets with zero id
1 !
```

```
1
1 LIBRARY 'SYS$LIBRARY:LIB.L32';
1 LIBRARY 'SRC$:CONVERT';
1
1 EXTERNAL ROUTINE
1            CONV$$RMS_READ_ERROR       : NOVALUE;
1
1 FORWARD ROUTINE
1            SQUISH_PRIMARY_BUCKET      : RL$JSB_REG_9  NOVALUE,
1            SQUISH_SIDR_BUCKET         : RL$JSB_REG_9  NOVALUE,
1            RECOMPRESS_RECORD          : RL$JSB_REG_8  NOVALUE;
1
1 EXTERNAL
1            RECL$GL_BUCKET_COUNT,
1            RECL$GL_SEARCH_BUFFER,
1            CONV$AB_OUT_FAB            : $FAB_DECL,
1            CONV$AB_OUT_RAB            : $RAB_DECL;
1
1 OWN
1            INDEX,
1            VBN_OFFSET,
1            VBN_FREE_SPACE,
1            KEY_BUFFER_1     :        BLOCK [ 257, BYTE ],
1            KEY_BUFFER_2     :        BLOCK [ 257, BYTE ];
1
```

```
1 %SBTTL  'BUCKET_EMPTY'
1 GLOBAL ROUTINE RECL$$BUCKET_EMPTY : RL$JSB_REG_9 =
1 !++
1 !
1 ! Functional Description:
1 !
1 !       This routine determines if a bucket is empty.  It handles both
1 !       index level and data level buckets.
1 !
1 ! Calling Sequence:
1 !
1 !       RECL$$BUCKET_EMPTY();
1 !
1 ! Input Parameters:
1 !
1 !       None.
1 !
1 ! Implicit Inputs:
1 !
1 !       BUCKET          - address of buffer containing bucket
1 !
1 ! Output Parameters:
1 !
1 !       None.
1 !
1 ! Implicit Outputs:
1 !
1 !       RECL$GL_BUCKET_COUNT is incremented.
1 !
1 ! Routine Value:
1 !
1 !       TRUE if bucket is empty
1 !       FALSE if bucket is not empty or can't be reclaimed
1 !
1 ! Routines Called:
1 !
1 !       SQUISH_PRIMARY_BUCKET
1 !       SQUISH_SIDR_BUCKET
1 !
1 ! Side Effects:
1 !
1 !       None.
1 !
1 !--
1
2     BEGIN
2
2     DEFINE_CTX;
2     DEFINE_BUCKET;
2     DEFINE_KEY_DESC;
2
2     LITERAL
2         RECL$_DATA_LEVEL          = 0,
2         RECL$_BUCKET_EMPTY        = 1,
2         RECL$_BUCKET_NOT_EMPTY    = 0;
2
2     ! Determine if bucket is data level or index level
```

```
;  184        0181   2          !
;  185        0182   2          IF .BUCKET [ BKT$B_LEVEL ] EQLU RECL$_DATA_LEVEL
;  186        0183   2          THEN
;  187        0184   2
;  188        0185   2              ! Determine key
;  189        0186   2              !
;  190        0187   2              IF .KEY_DESC [ KEY$B_KEYREF ] EQL 0
;  191        0188   2              THEN
;  192        0189   2                  SQUISH_PRIMARY_BUCKET()
;  193        0190   2              ELSE
;  194        0191   2                  SQUISH_SIDR_BUCKET();
;  195        0192   2
;  196        0193   2          ! See if it's empty
;  197        0194   2          !
;  198        0195   2          ! NOTE: Never reclaim the last bucket in a level, due to the complexity of
;  199        0196   2          ! updating high key values in all the levels above.  This is not a serious
;  200        0197   2          ! restriction since most reclamation will be of aging buckets early in
;  201        0198   2          ! collating sequence.
;  202        0199   2          !
;  203        0200   2          IF ( .BUCKET [ BKT$W_KEYFRESPC ] NEQU BKT$C_OVERHDSZ ) OR
;  204        0201   2                                                          .BUCKET [ BKT$V_LASTBKT ]
;  205        0202   2          THEN
;  206        0203   2              RETURN RECL$_BUCKET_NOT_EMPTY
;  207        0204   2          ELSE
;  208        0205   2              RETURN RECL$_BUCKET_EMPTY
;  209        0206   2
;  210        0207   1          END;


                                                             .TITLE  RECL$REC VAX-11 CONVERT/RECLAIM
                                                             .IDENT  \V04-000\

                                                             .PSECT  $OWN$,NOEXE,2

                                          00000 INDEX:  .BLKB   4
                                          00004 VBN_OFFSET:
                                                        .BLKB   4
                                          00008 VBN_FREE_SPACE:
                                                        .BLKB   4
                                          0000C KEY_BUFFER_1:
                                                        .BLKB   257
                                          0010D         .BLKB   3
                                          00110 KEY_BUFFER_2:
                                                        .BLKB   257

                                                             .EXTRN  CONV$$RMS_READ_ERROR
                                                             .EXTRN  RECL$GL_BUCKET_COUNT
                                                             .EXTRN  RECL$GL_SEARCH_BUFFER
                                                             .EXTRN  CONV$AB_OUT_FAB
                                                             .EXTRN  CONV$AB_OUT_RAB

                                                             .PSECT  $CODE$,NOWRT,2

                        0C    A9   95 00000 RECL$$BUCKET_EMPTY::
                                                             TSTB    12(BUCKET)                                   ; 0182
                              0D   12 00003                  BNEQ    2$                                           ;
                        15    AB   95 00005                  TSTB    21(KEY_DESC)                                 ; 0187
```

```
                                       05  12 00008          BNEQ    1$                                                  ;
                                    0000V  30 0000A          BSBW    SQUISH_PRIMARY_BUCKET                               ; 0189
                                       03  11 0000D          BRB     2$                                                  ;
                                    0000V  30 0000F  1$:     BSBW    SQUISH_SIDR_BUCKET                                  ; 0191
                  0E         04        A9  B1 00012  2$:     CMPW    4(BUCKET), #14                                      ; 0200
                                       04  12 00016          BNEQ    3$                                                  ;
                  03         0D        A9  E9 00018          BLBC    13(BUCKET), 4$                                      ; 0201
                                       50  D4 0001C  3$:     CLRL    R0                                                  ; 0205
                                       05  0001E             RSB                                                         ;
                  50         01        D0  0001F     4$:     MOVL    #1, R0                                              ;
                                       05  00022             RSB                                                         ; 0207

; Routine Size:  35 bytes,    Routine Base:  $CODE$ + 0000
```

```
1 %SBTTL  'SQUISH_PRIMARY_BUCKET'
1 ROUTINE SQUISH_PRIMARY_BUCKET : RL$JSB_REG_9 NOVALUE =
1 !++
1 !
1 ! Functional Description:
1 !
1 !       Suqishes the deleted records out of the primary data buckets
1 !
1 ! Calling Sequence:
1 !
1 !       SQUISH_PRIMARY_BUCKET()
1 !
1 ! Input Parameters:
1 !       None
1 !
1 ! Implicit Inputs:
1 !
1 !       BUCKET          - address of buffer containing bucket
1 !       KEY_DESC
1 !
1 ! Output Parameters:
1 !       None
1 !
1 ! Implicit Outputs:
1 !       None
1 !
1 ! Routine Value:
1 !       None
1 !
1 ! Routines Called:
1 !
1 !       None.
1 !
1 ! Side Effects:
1 !
1 !       None.
1 !
1 !--
1
2     BEGIN
2
2     DEFINE_BUCKET;
2     DEFINE_KEY_DESC;
2
2     LOCAL
2         LAST,
2         POINTER,
2         RECORD_CTRL     : REF BLOCK [ ,BYTE ];
2
2     ! Point to the first record in the bucket
2     !
2     POINTER = BKT$K_OVERHDSZ + .BUCKET;
2
2     LAST = .POINTER;
2
2     ! Count the bucket
2     !
```

```
2       RECL$GL_BUCKET_COUNT = .RECL$GL_BUCKET_COUNT + 1;
2
2       ! If this bucket has an id of zero then don't bother reclaiming it
2       !
2       IF .BUCKET [ BKT$W_NXTRECID ] EQLU 0
2       THEN
2           RETURN;
2
2       ! Loop untill we have looked at all of the records
2       !
3       WHILE .POINTER LSSU ( .BUCKET [ BKT$W_FREESPACE ] + .BUCKET )
2       DO
3           BEGIN
3
3           ! Point to the control bytes of the record
3           !
3           RECORD_CTRL = .POINTER;
3
3           ! If this record not deleted check to see if there were any deleted
3           ! records before it, if so squish them out
3           !
4           IF NOT  (.RECORD_CTRL [ IRC$V_DELETED ]
4                     OR
4                    .RECORD_CTRL [ IRC$V_RU_DELETE ])
3           THEN
4               BEGIN
4
4               LOCAL       SQUISH;
4
4               ! The current record is not deleted so squish out the
4               ! deleted ones if there where any
4               !
4               SQUISH = .POINTER - .LAST;
4
4               IF .SQUISH NEQ 0
4               THEN
5                   BEGIN
5
5                   LOCAL BYTES;
5
5                   ! Number of bytes left in the bucket
5                   !
5                   BYTES = ( .BUCKET + .BUCKET [ BKT$W_FREESPACE ] ) - .POINTER;
5
5                   ! Move the rest of the records
5                   !
5                   CH$MOVE( .BYTES,.POINTER,.LAST );
5
5                   ! Update the bucket pointer
5                   !
5                   BUCKET [ BKT$W_FREESPACE ] = .BUCKET [ BKT$W_FREESPACE ] -
5                                                                     .SQUISH;
5
5                   ! Update our pointers
5                   !
5                   POINTER = .POINTER - .SQUISH;
5                   RECORD_CTRL = .POINTER
```

```
5
4            END;
4
4        ! If the current non-deleted primary data record is marked RU_UPDATE
4        ! then re-format at this time.
4        !
4        IF .RECORD_CTRL [ IRC$V_RU_UPDATE ]
4        THEN
5            BEGIN
5
5            LOCAL
5                BYTES,
5                FAKE_SIZE    : WORD,
5                TRUE_SIZE    : WORD;
5
5            ! Turn of the RU_UPDATE bit and retrieve the record's true size
5            ! and the number of bytes in the bucket it currently occupies.
5            !
5            RECORD_CTRL [ IRC$V_RU_UPDATE ] = 0;
5            FAKE_SIZE = .RECORD_CTRL [ 9,0,16,0 ];
5            TRUE_SIZE = .(.RECORD_CTRL + .FAKE_SIZE + 9)<0,16>;
5
5            ! Place the true size of the primary data record in the size
5            ! field of the record overhead, shift the rest of the records
5            ! in the bucket to take up the available space, and update the
5            ! bucket's freespace offset pointer.
5            !
5            RECORD_CTRL [ 9,0,16,0 ] = .TRUE_SIZE;
5
5            BYTES = .BUCKET + .BUCKET [ BKT$W_FREESPACE ]
5                            - .RECORD_CTRL
5                            - .FAKE_SIZE;
5
5            IF .BYTES GTRU 0
5            THEN
5                CH$MOVE ( .BYTES,
5                          .RECORD_CTRL + .FAKE_SIZE,
5                          .RECORD_CTRL + .TRUE_SIZE );
5
5            BUCKET [ BKT$W_FREESPACE ] = .BUCKET [ BKT$W_FREESPACE ]
5                            - ( .FAKE_SIZE - .TRUE_SIZE );
4            END;
3        END;
3
3    ! Find the next record
3    !
3    ! Is this record a RRV record
3    !
3    IF .RECORD_CTRL [ IRC$V_RRV ]
3    THEN
3
3        ! If this record has no RRV pointer then set the size to the
3        ! smallest record there is
3        !
3        IF .RECORD_CTRL [ IRC$V_NOPTRSZ ]
3        THEN
3
```

```
                        ! The least case size of a record is 3 bytes (CTRL and ID)
                        !
                        POINTER = .POINTER + 3

                    ELSE

                        ! The size of the record with an RRV pointer is
                        ! CTRL, ID and Pointer Size (ID and VBN)
                        !
                        POINTER = .POINTER + 3 + .RECORD_CTRL [ IRC$V_PTRSZ ] + 4

                ELSE

                    ! It is not a RRV, so does it have a size field
                    !
                    IF .KEY_DESC [ KEY$V_REC_COMPR ] OR
                        .KEY_DESC [ KEY$V_KEY_COMPR ] OR
                        ( .CONV$AB_OUT_FAB [ FAB$B_RFM ] EQL FAB$C_VAR )
                    THEN

                        ! Add the size of the record from the size field and control
                        !
                        POINTER = .POINTER + .RECORD_CTRL [ 9,0,16,0 ] + 11

                    ELSE

                        ! Add the size of the record and control bytes
                        !
                        POINTER = .POINTER + .CONV$AB_OUT_FAB [ FAB$W_MRS ] + 9;

                ! If the last record was not deleted update the last record pointer
                !
                IF NOT  (.RECORD_CTRL [ IRC$V_DELETED ]
                        OR
                        .RECORD_CTRL [ IRC$V_RU_DELETE ])
                THEN
                    LAST = .POINTER

                END;

            ! Update the bucket pointer to catch the last record if it was deleted
            !
            ! We exit the loop under two cases, 1) the last n records were deleted
            ! in which case LAST points to the first deleted record or 2) the last
            ! record was not deleted in which case LAST will be pointing to the
            ! END of the last record, i.e. same as freespace.
            !
            BUCKET [ BKT$W_FREESPACE ] = .LAST - .BUCKET;

            RETURN

            END;
```

```
                              05FC    8F  BB 00000 SQUISH_PRIMARY_BUCKET:
                                                            PUSHR   #^M<R2,R3,R4,R5,R6,R7,R8,R10>                 ; 0209
                     5E               04  C2 00004          SUBL2   #4, SP
                     57         0E    A9  9E 00007          MOVAB   14(R9), POINTER                               ; 0259
                                      57  DD 0000B          PUSHL   POINTER                                       ; 0261
                              0000G   CF  D6 0000D          INCL    RECL$GL_BUCKET_COUNT                          ; 0265
                                06    A9  B5 00011          TSTW    6(BUCKET)                                     ; 0269
                                      03  12 00014          BNEQ    1$
                                    00C8  31 00016          BRW     14$
                 04  AE         04    A9  9E 00019 1$:      MOVAB   4(BUCKET), 4(SP)                              ; 0275
                     51         04    BE  3C 0001E 2$:      MOVZWL  @4(SP), R1
          50         51               59  C1 00022          ADDL3   BUCKET, R1, R0
                     50               57  D1 00026          CMPL    POINTER, R0
                                      03  1F 00029          BLSSU   3$
                                    00AE  31 0002B          BRW     13$
                     56               57  D0 0002E 3$:      MOVL    POINTER, RECORD_CTRL                          ; 0281
          5A         66               02  E0 00031          BBS     #2, (RECORD_CTRL), 6$                         ; 0286
          56         66               05  E0 00035          BBS     #5, (RECORD_CTRL), 6$                         ; 0288
          58         57               6E  C3 00039          SUBL3   LAST, POINTER, SQUISH                         ; 0297
                                      16  13 0003D          BEQL    4$                                            ; 0299
          50         59               51  C1 0003F          ADDL3   R1, BUCKET, R0                                ; 0307
                     50               57  C2 00043          SUBL2   POINTER, BYTES
     00   BE         67               50  28 00046          MOVC3   BYTES, (POINTER), @LAST                       ; 0311
                 04  BE               58  A2 0004B          SUBW2   SQUISH, @4(SP)                                ; 0316
                     57               58  C2 0004F          SUBL2   SQUISH, POINTER                               ; 0320
                     56               57  D0 00052          MOVL    POINTER, RECORD_CTRL                          ; 0321
          36         66               06  E1 00055 4$:      BBC     #6, (RECORD_CTRL), 6$                         ; 0328
                     66         40    8F  8A 00059          BICB2   #64, (RECORD_CTRL)                            ; 0340
                     51         09    A6  B0 0005D          MOVW    9(RECORD_CTRL), FAKE_SIZE                     ; 0341
                     5A               51  3C 00061          MOVZWL  FAKE_SIZE, R10                                ; 0342
          51         56               5A  C1 00064          ADDL3   R10, RECORD_CTRL, R1
                     50         09    A1  B0 00068          MOVW    9(R1), TRUE_SIZE
                     58               50  3C 0006C          MOVZWL  TRUE_SIZE, R8                                 ; 0349
                 09  A6               58  B0 0006F          MOVW    R8, 9(RECORD_CTRL)
                     50         04    BE  3C 00073          MOVZWL  @4(SP), R0                                    ; 0351
                     50               59  C0 00077          ADDL2   BUCKET, R0
                     50               56  C2 0007A          SUBL2   RECORD_CTRL, R0                               ; 0352
                     50               5A  C2 0007D          SUBL2   R10, BYTES                                    ; 0353
                                      05  13 00080          BEQL    5$                                            ; 0355
        6846         61               50  28 00082          MOVC3   BYTES, (R1), (R8)[RECORD_CTRL]                ; 0359
          50         58               5A  C3 00087 5$:      SUBL3   R10, R8, R0                                   ; 0362
                 04  BE               50  A0 0008B          ADDW2   R0, @4(SP)
          15         66               03  E1 0008F 6$:      BBC     #3, (RECORD_CTRL), 8$                         ; 0370
          05         66               04  E1 00093          BBC     #4, (RECORD_CTRL), 7$                         ; 0376
                     57               03  C0 00097          ADDL2   #3, POINTER                                   ; 0381
                                      32  11 0009A          BRB     11$
 50       66         02               00  EF 0009C 7$:      EXTZV   #0, #2, (RECORD_CTRL), R0                     ; 0388
                     57       07 A047     9E 000A1          MOVAB   7(R0)[POINTER], POINTER
                                      26  11 000A6          BRB     11$                                           ; 0376
                                10    AB  95 000A8 8$:      TSTB    16(KEY_DESC)                                  ; 0394
                                      0C  19 000AB          BLSS    9$
          07     10  AB               06  E0 000AD          BBS     #6, 16(KEY_DESC), 9$                          ; 0395
                     02       0000G   CF  91 000B2          CMPB    CONV$AB_OUT_FAB+31, #2                        ; 0396
                                      0B  12 000B7          BNEQ    10$
                     50         09    A6  3C 000B9 9$:      MOVZWL  9(RECORD_CTRL), R0                            ; 0401
                     57       0B A047     9E 000BD          MOVAB   11(R0)[POINTER], POINTER
                                      0A  11 000C2          BRB     11$
```

```
                        50    0000G  CF  3C 000C4 10$:    MOVZWL  CONV$AB_OUT_FAB+54, R0                            ; 0407
                        57     09 A047   9E 000C9          MOVAB   9(R0)[POINTER], POINTER                           ;
           07           66           02  E0 000CE 11$:    BBS     #2, (RECORD_CTRL), 12$                            ; 0411
           03           66           05  E0 000D2          BBS     #5, (RECORD_CTRL), 12$                            ; 0413
                        6E           57  D0 000D6          MOVL    POINTER, LAST                                     ; 0415
                                   FF42  31 000D9 12$:    BRW     2$                                                ; 0411
      04   BE           6E           59  A3 000DC 13$:    SUBW3   BUCKET, LAST, @4(SP)                              ; 0426
                        5E           08  C0 000E1 14$:    ADDL2   #8, SP                                            ; 0430
                             05FC    8F  BA 000E4          POPR    #^M<R2,R3,R4,R5,R6,R7,R8,R10>                     ;
                                     05     000E8          RSB                                                       ;

; Routine Size:  233 bytes,    Routine Base:  $CODE$ + 0023
```

```
%SBTTL  'SQUISH_SIDR_BUCKET'
ROUTINE SQUISH_SIDR_BUCKET : RL$JSB_REG_9 NOVALUE =
!++
!
! Functional Description:
!
!       Suqishes the deleted records out of the sidr data buckets
!
! Calling Sequence:
!
!       SQUISH_SIDR_BUCKET()
!
! Input Parameters:
!       None
!
! Implicit Inputs:
!
!       BUCKET          - address of buffer containing bucket
!       KEY_DESC
!
! Output Parameters:
!       None
!
! Implicit Outputs:
!       None
!
! Routine Value:
!       None
!
! Routines Called:
!       None
!
! Side Effects:
!       None
!               NOTE: The routine SQUISH_SIDR_BUCKET is algorithmically wrong.
!               It doesn't squish out anything! I plan on leaving it the way it
!               is until a massive re-write can be done.
!
!--

    BEGIN

    DEFINE_BUCKET;
    DEFINE_KEY_DESC;

    LOCAL
        LAST,
        POINTER         : REF BLOCK [ ,BYTE ],
        SIDR            : REF BLOCK [ ,BYTE ];

    ! Point to the first record in the bucket
    !
    SIDR = BKT$K_OVERHDSZ + .BUCKET;

    ! Count the bucket
    !
    RECL$GL_BUCKET_COUNT = .RECL$GL_BUCKET_COUNT + 1;
```

```
2
2        ! Loop untill we have looked at all of the records
2        !
3        WHILE .SIDR LSSU ( .BUCKET [ BKT$W_FREESPACE ] + .BUCKET )
2        DO
3            BEGIN
3
3            ! Point to the first array element
3            !
3            IF .KEY_DESC [ KEY$V_KEY_COMPR ]
3            THEN
3                POINTER = .SIDR + .SIDR [ 2,0,8,0 ] + 4
3            ELSE
3                POINTER = .SIDR + .KEY_DESC [ KEY$B_KEYSZ ] + 2;
3
3            LAST = .POINTER;
3
3            ! Loop untill we have looked at all of the array elements
3            !
4            WHILE .POINTER LSSU ( .SIDR + .SIDR [ 0,0,16,0 ] + 2 )
3            DO
3
3                ! If this array elemented is deleted skip to the next one
3                !
3                IF   .POINTER [ IRC$V_DELETED ]
3                    OR
3                    .POINTER [ IRC$V_RU_DELETE ]
3                THEN
3
3                    ! Is there a pointer
3                    !
3                    IF .POINTER [ IRC$V_NOPTRSZ ]
3                    THEN
3                        POINTER = .POINTER + 1
3                    ELSE
3                        POINTER = .POINTER + 1 + .POINTER [ IRC$V_PTRSZ ] + 4
3
3                ELSE
4                    BEGIN
4
4                    LOCAL   SQUISH;
4
4                    ! The current sidr is not deleted so squish out the
4                    ! deleted ones if there where any
4                    !
4                    SQUISH = .POINTER - .LAST;
4
4                    IF .SQUISH NEQ 0
4                    THEN
5                        BEGIN
5
5                        LOCAL BYTES;
5
5                        ! Number of bytes left in the bucket
5                        !
5                        BYTES = ( .BUCKET + .BUCKET [ BKT$W_FREESPACE ] ) - .POINTER;
5
```

```
                        ! Move the rest of the records
                        !
                        CH$MOVE( .BYTES,.POINTER,.LAST );

                        ! Update the bucket pointer
                        !
                        BUCKET [ BKT$W_FREESPACE ] = .BUCKET [ BKT$W_FREESPACE ] -
                                                                        .SQUISH;


                        ! Update the sidr record size
                        !
                        SIDR [ 0,0,16,0 ] = .SIDR [ 0,0,16,0 ] - .SQUISH;

                        ! Update out pointers
                        !
                        POINTER = .POINTER - .SQUISH;

                        END;

                    ! Find the next sidr element
                    !
                    POINTER = .POINTER + 1 + .POINTER [ IRC$V_PTRSZ ] + 4;

                    LAST = .POINTER

                    END;

                ! Is the sidr array completely deleted
                !
                IF .POINTER EQL ( .SIDR + .SIDR [ 0,0,16,0 ] )
                THEN
                    BEGIN

                    ! Squish out the entire record (leaving SIDR pointing to the
                    ! next sidr record)
                    !
                    CH$MOVE( .SIDR [ 0,0,16,0 ],.POINTER,.SIDR );

                    ! Update the bucket pointer
                    !
                    BUCKET [ BKT$W_FREESPACE ] = .BUCKET [ BKT$W_FREESPACE ] -
                                                        .SIDR [ 0,0,16,0 ]

                    END
                ELSE

                    ! If we don't squish the record find the next one
                    !
                    SIDR = .SIDR + .SIDR [ 0,0,16,0 ] + 2

                END;

            RETURN

            END;
```

```
                                 05FC   8F  BB 00000 SQUISH_SIDR_BUCKET:
                                                              PUSHR   #^M<R2,R3,R4,R5,R6,R7,R8,R10>                    ; 0432
                           5E           04  C2 00004          SUBL2   #4, SP                                           ;
                           56      0E   A9  9E 00007          MOVAB   14(R9), SIDR                                     ; 0483
                               0000G    CF  D6 0000B          INCL    RECL$GL_BUCKET_COUNT                             ; 0487
                           5A      04   A9  9E 0000F          MOVAB   4(BUCKET), R10                                   ; 0491
                           50           6A  3C 00013 1$:      MOVZWL  (R10), R0                                        ;
                           50           59  C0 00016          ADDL2   BUCKET, R0                                       ;
                           50           56  D1 00019          CMPL    SIDR, R0                                         ;
                                        03  1F 0001C          BLSSU   2$                                               ;
                                      0089  31 0001E          BRW     13$                                              ;
             0B       10   AB           06  E1 00021 2$:      BBC     #6, 16(KEY_DESC), 3$                             ; 0497
                           50      02   A6  9A 00026          MOVZBL  2(SIDR), R0                                      ; 0499
                           57      04 A046  9E 0002A          MOVAB   4(R0)[SIDR], POINTER                             ;
                                        09  11 0002F          BRB     4$                                               ;
                           50      14   AB  9A 00031 3$:      MOVZBL  20(KEY_DESC), R0                                 ; 0501
                           57      02 A046  9E 00035          MOVAB   2(R0)[SIDR], POINTER                             ;
                           6E           57  D0 0003A 4$:      MOVL    POINTER, LAST                                    ; 0503
                           50           66  3C 0003D 5$:      MOVZWL  (SIDR), R0                                       ; 0507
                           50      02 A046  9E 00040          MOVAB   2(R0)[SIDR], R0                                  ;
                           50           57  D1 00045          CMPL    POINTER, R0                                      ;
                                        45  1E 00048          BGEQU   10$                                              ;
             04            67           02  E0 0004A          BBS     #2, (POINTER), 6$                                ; 0512
             14            67           05  E1 0004E          BBC     #5, (POINTER), 8$                                ; 0514
             04            67           04  E1 00052 6$:      BBC     #4, (POINTER), 7$                                ; 0519
                                        57  D6 00056          INCL    POINTER                                          ; 0521
                                        E3  11 00058          BRB     5$                                               ;
  50         67            02           00  EF 0005A 7$:      EXTZV   #0, #2, (POINTER), R0                            ; 0523
                           57      05 A047  9E 0005F          MOVAB   5(R0)[POINTER], POINTER                          ;
                                        D7  11 00064          BRB     5$                                               ; 0519
             58            57           6E  C3 00066 8$:      SUBL3   LAST, POINTER, SQUISH                            ; 0533
                                        17  13 0006A          BEQL    9$                                               ; 0535
                           50           6A  3C 0006C          MOVZWL  (R10), R0                                        ; 0543
                           50           59  C0 0006F          ADDL2   BUCKET, R0                                       ;
                           50           57  C2 00072          SUBL2   POINTER, BYTES                                   ;
        00   BE            67           50  28 00075          MOVC3   BYTES, (POINTER), @LAST                          ; 0547
                           6A           58  A2 0007A          SUBW2   SQUISH, (R10)                                    ; 0552
                           66           58  A2 0007D          SUBW2   SQUISH, (SIDR)                                   ; 0557
                           57           58  C2 00080          SUBL2   SQUISH, POINTER                                  ; 0561
  50         67            02           00  EF 00083 9$:      EXTZV   #0, #2, (POINTER), R0                            ; 0567
                           57      05 A047  9E 00088          MOVAB   5(R0)[POINTER], POINTER                          ;
                                        AB  11 0008D          BRB     4$                                               ; 0569
                           50           66  3C 0008F 10$:     MOVZWL  (SIDR), R0                                       ; 0575
                           50           56  C0 00092          ADDL2   SIDR, R0                                         ;
                           50           57  D1 00095          CMPL    POINTER, R0                                      ;
                                        09  12 00098          BNEQ    11$                                              ;
             66            67           66  28 0009A          MOVC3   (SIDR), (POINTER), (SIDR)                        ; 0582
                           6A           66  A2 0009E          SUBW2   (SIDR), (R10)                                    ; 0587
                                        04  11 000A1          BRB     12$                                              ; 0586
                           56      02   A0  9E 000A3 11$:     MOVAB   2(R0), SIDR                                      ; 0594
                                      FF69  31 000A7 12$:     BRW     1$                                               ; 0575
                           5E           04  C0 000AA 13$:     ADDL2   #4, SP                                           ; 0600
                                 05FC   8F  BA 000AD          POPR    #^M<R2,R3,R4,R5,R6,R7,R8,R10>                    ;
```

```
                                           05 000B1          RSB                                              ;

; Routine Size:  178 bytes,    Routine Base:  $CODE$ + 010C
```

```
1 %SBTTL  'GET_DOWN_POINTER'
1 GLOBAL ROUTINE RECL$$GET_DOWN_POINTER ( VBN ) : RL$JSB_REG_8 =
1 !++
1 !
1 ! Functional Description:
1 !
1 !       This routine searches the current buffer for the specified
1 !       down pointer.
1 !
1 ! Calling Sequence:
1 !
1 !       GET_DOWN_POINTER( VBN );
1 !
1 ! Input Parameters:
1 !
1 !       VBN     - VBN of bucket on level below being deleted
1 !
1 ! Implicit Inputs:
1 !
1 !       BUCKET                  - address of buffer containing bucket
1 !       KEY_DESC
1 !
1 ! Output Parameters:
1 !
1 !       None.
1 !
1 ! Implicit Outputs:
1 !
1 !   If success:
1 !
1 !       INDEX           - number of the index record to remove (0=first)
1 !       KEY_POINTER     - points to key part to delete
1 !       KEY_BUFFER_1    - contains the expanded key bucket previous
1 !                         to one being deleted
1 !       KEY_BUFFER_2    - contains the expanded key of one being deleted
1 !
1 !   If failure the contents of the above registers are undefined.
1 !
1 ! Routine Value:
1 !
1 !       TRUE if down pointer found, else FALSE
1 !
1 ! Routines Called:
1 !
1 !       None.
1 !
1 ! Side Effects:
1 !
1 !       None.
1 !
1 !--
1
2     BEGIN
2
2     DEFINE_CTX;
2     DEFINE_BUCKET;
2     DEFINE_KEY_DESC;
```

```
DEFINE_KEY_POINTER;

LOCAL
    VBN_OFFSET,
    VBN_FREE_SPACE;

! Initialize the index which counts which record in is the down pointer.
!
INDEX = 0;

! Initialize offset in bucket to word containing VBN free space pointer
! so we can get the actual offset to the VBN free space.
!
VBN_OFFSET = .CTX [ CTX$W_BUCKET_SIZE ] - 2 - 2;

! Get actual offset of VBN free space.
!
VBN_FREE_SPACE = .BUCKET [ .VBN_OFFSET, 0, 16, 0 ];

! Now point to first VBN down pointer.
!
VBN_OFFSET = .VBN_OFFSET - ( .BUCKET [ BKT$V_PTR_SZ ] + 2);

! Scan the VBNs to see if the down pointer is in this bucket.
!
UNTIL .VBN_OFFSET LEQA .VBN_FREE_SPACE
DO

    ! Compare the VBN value pointed to by VBN_OFFSET.
    !
    IF .BUCKET [ .VBN_OFFSET,0,((.BUCKET[ BKT$V_PTR_SZ ] + 2) * 8), 0 ] EQLU
                                                                     .VBN
    THEN

        ! We found the down pointer, so point KEY_POINTER to the key part
        ! of the index record.
        !
        IF .KEY_DESC[ KEY$V_IDX_COMPR ]
        THEN
            BEGIN

            ! The key is compressed, so each key part is variable length.
            ! INDEX is currently an index to the right record, so
            ! skip over that many records.
            !
            KEY_POINTER = .BUCKET + BKT$K_OVERHDSZ;

            INCR I FROM 0 TO .INDEX - 1
            DO
                BEGIN

                ! Move the key into the buffer while expanding
                ! the rear end truncation
                !
                !               key_pointer
                !                   !
                !                   -----------
```

```
4        !               !l!c!      !
4        !               -----------
4        !                    \     \ fill \
4        !               --------------------
4        ! key_buffer_1  !l!c!  :     :     !
4        !               --------------------
4        !                    ^
4        !                    !
4        !                  filled in when c=0 ( always the first key
4        !                                        in the bucket )
4        !
4        ! CH$COPY( src_len, src, fill, dst_len, dst )
4        !
4       CH$COPY( .KEY_POINTER [ KEYR$B_LENGTH ],
4                .KEY_POINTER + 2,
5                .( .KEY_POINTER + 1 +
4                           .KEY_POINTER [ KEYR$B_LENGTH ] ),
4                .KEY_DESC [ KEY$B_KEYSZ ] -
4                           .KEY_POINTER [ KEYR$B_FRONT_COUNT ],
4                KEY_BUFFER_1 + 2 +
4                           .KEY_POINTER [ KEYR$B_FRONT_COUNT ] );
4
4        ! Skip to the next key.
4        !
4       KEY_POINTER = .KEY_POINTER + 2 +
4                     .KEY_POINTER [ KEYR$B_LENGTH ]
4
4       END;
3
3    ! Fill in key_buffer_2 with the expanded CURRENT key
3    ! first by stuffing the front compresed characters from
3    ! the previous key in key_buffer_1 then copy the rest
3    ! from the bucket extending it if rear truncation is present
3    !
3    !         key_pointer
3    !               !
3    !               -----------
3    !               !l!c!     !
3    !               -----------
3    !                    \     \
3    !               --------    !
3    ! key_buffer_1  !l!c!  :    !
3    !               --------    !
3    !               !      !    ! fill \
3    !               --------------------
3    ! key_buffer_2  !l!c!  :     :     !
3    !               --------------------
3    !                    ^
3    !                    !
3    !                 filled in when from key_buffer_1
3    !                 or from .key_pointer when c=0
3    !
3    !
3    ! Fill in the front if there were front compression
3    !
3    CH$MOVE( .KEY_POINTER [ KEYR$B_FRONT_COUNT ],
3             KEY_BUFFER_1 + 2,
;  778          0772  3                     KEY_BUFFER_2 + 2);
;  779          0773  3
;  780          0774  3             ! Copy the rest of the key and expand the rear if neccessary
;  781          0775  3             !
;  782          0776  3             CH$COPY( .KEY_POINTER [ KEYR$B_LENGTH ],
;  783          0777  3                     .KEY_POINTER + 2,
;  784          0778  3                     .( .KEY_POINTER + 1 + .KEY_POINTER [ KEYR$B_LENGTH ] ),
;  785          0779  3                     .KEY_DESC [ KEY$B_KEYSZ ] =
;  786          0780  3                                     .KEY_POINTER [ KEYR$B_FRONT_COUNT ],
;  787          0781  3                     KEY_BUFFER_2 + 2 +
;  788          0782  3                                     .KEY_POINTER [ KEYR$B_FRONT_COUNT ] );
;  789          0783  3
;  790          0784  3             RETURN RECL$_SUCCESS
;  791          0785  3
;  792          0786  3             END
;  793          0787  3
;  794          0788  2         ELSE
;  795          0789  2
;  796          0790  3             BEGIN
;  797          0791  3
;  798          0792  3             ! The key is not compressed, so the key part is fixed length
;  799          0793  3             ! and easy to find.
;  800          0794  3             !
;  801          0795  4             KEY_POINTER = ( .INDEX * .KEY_DESC[ KEY$B_KEYSZ ] )
;  802          0796  3                             + .BUCKET + BKT$K_OVERHDSZ;
;  803          0797  3
;  804          0798  3             RETURN RECL$_SUCCESS
;  805          0799  3
;  806          0800  3             END
;  807          0801  3
;  808          0802  2     ELSE
;  809          0803  3         BEGIN
;  810          0804  3
;  811          0805  3         ! This was not the down pointer, so get the next down pointer
;  812          0806  3         !
;  813          0807  3         VBN_OFFSET = .VBN_OFFSET - ( .BUCKET[ BKT$V_PTR_SZ ] + 2 );
;  814          0808  3         INDEX = .INDEX + 1;
;  815          0809  3
;  816          0810  2         END;
;  817          0811  2
;  818          0812  2 ! If we fell through the UNTIL - DO loop it means we didn't find a down
;  819          0813  2 ! pointer.
;  820          0814  2 !
;  821          0815  2 RETURN RECL$_FAILURE
;  822          0816  2
;  823          0817  1 END;




                            00FC    8F  BB 00000 RECL$$GET_DOWN_POINTER::
                                                         PUSHR   #^M<R2,R3,R4,R5,R6,R7>                                 ; 0602
                  5E                0C  C2 00004         SUBL2   #12, SP
                        0000'       CF  D4 00007         CLRL    INDEX                                                  ; 0666
                  56          58    AA  3C 0000B         MOVZWL  88(CTX), VBN_OFFSET                                    ; 0671
                  56                03  C2 0000F         SUBL2   #3, VBN_OFFSET                                         ;
```

```
                                                                     N 13
RECL$REC         VAX-11 CONVERT/RECLAIM                              15-Sep-1984 23:59:42    VAX-11 Bliss-32 V4.0-742                 Page  23
V04-000          GET_DOWN_POINTER                                    14-Sep-1984 12:14:05    DISK$VMSMASTER:[CONV.SRC]RECLREC.B32;1    (7)

                                              7649  9F 00012       PUSHAB  -(VBN_OFFSET)[BUCKET]                              ; 0675
                             08   AE            9E  3C 00015       MOVZWL  @(SP)+, VBN_FREE_SPACE                             ;
     04   AE      0D   A9         02            03  EF 00019       EXTZV   #3, #2, 13(BUCKET), 4(SP)                          ; 0679
                       50         56      04    AE  C3 00020       SUBL3   4(SP), VBN_OFFSET, R0                              ;
                                  56      FE    A0  9E 00025       MOVAB   -2(R0), VBN_OFFSET                                 ;
                             08   AE            56  D1 00029 1$:   CMPL    VBN_OFFSET, VBN_FREE_SPACE                         ; 0683
                                                03  1A 0002D       BGTRU   2$                                                 ;
                                              0090  31 0002F       BRW     8$                                                 ;
                       50    04   AE            03  78 00032 2$:   ASHL    #3, 4(SP), R0                                      ; 0688
                                  50            10  C0 00037       ADDL2   #16, R0                                            ;
          51         6649         50            00  EF 0003A       EXTZV   #0, R0, (VBN_OFFSET)[BUCKET], R1                   ;
                             28   AE            51  D1 00040       CMPL    R1, VBN                                            ; 0689
                                                6C  12 00044       BNEQ    7$                                                 ;
                       54    10   AB            03  E1 00046       BBC     #3, 16(KEY_DESC), 5$                               ; 0695
                                  58      0E    A9  9E 0004B       MOVAB   14(R9), KEY_POINTER                                ; 0703
                                  6E            01  CE 0004F       MNEGL   #1, I                                              ; 0705
                                                21  11 00052       BRB     4$                                                 ;
                                  52            68  9A 00054 3$:   MOVZBL  (KEY_POINTER), R2                                  ; 0728
                       57         58            52  C1 00057       ADDL3   R2, KEY_POINTER, R7                                ; 0730
                                  50      01    A8  9A 0005B       MOVZBL  1(KEY_POINTER), R0                                 ; 0733
                                  51      14    AB  9A 0005F       MOVZBL  20(KEY_DESC), R1                                   ;
                                  51            50  C2 00063       SUBL2   R0, R1                                             ;
          51      01   A7    02   A8            52  2C 00066       MOVC5   R2, 2(KEY_POINTER), 1(R7), R1, -                   ; 0734
                                         0000'CF40     0006D               KEY_BUFFER_1+2[R0]                                 ;
                                  58      02    A7  9E 00071       MOVAB   2(R7), KEY_POINTER                                 ; 0739
                       D9         6E   0000'    CF  F2 00075 4$:   AOBLSS  INDEX, I, 3$                                       ;
                                  57      01    A8  9A 0007B       MOVZBL  1(KEY_POINTER), R7                                 ; 0770
             0000'    CF   0000'  CF            57  28 0007F       MOVC3   R7, KEY_BUFFER_1+2, KEY_BUFFER_2+2                 ; 0772
                                  50            68  9A 00087       MOVZBL  (KEY_POINTER), R0                                  ; 0776
                                  51      14    AB  9A 0008A       MOVZBL  20(KEY_DESC), R1                                   ; 0780
                                  51            57  C2 0008E       SUBL2   R7, R1                                             ;
          51      01 A048    02   A8            50  2C 00091       MOVC5   R0, 2(KEY_POINTER), 1(R0)[KEY_POINTER], -          ; 0781
                                         0000'CF47     00099               R1, KEY_BUFFER_2+2[R7]                             ;
                                                0E  11 0009D       BRB     6$                                                 ; 0790
                                  50      14    AB  9A 0009F 5$:   MOVZBL  20(KEY_DESC), R0                                   ; 0795
                                  50   0000'    CF  C4 000A3       MULL2   INDEX, R0                                          ;
                                  58      0E  A940  9E 000A8       MOVAB   14(BUCKET)[R0], KEY_POINTER                        ; 0796
                                  50            01  D0 000AD 6$:   MOVL    #1, R0                                             ; 0798
                                                12  11 000B0       BRB     9$                                                 ; 0790
                       50         56      04    AE  C3 000B2 7$:   SUBL3   4(SP), VBN_OFFSET, R0                              ; 0807
                                  56      FE    A0  9E 000B7       MOVAB   -2(R0), VBN_OFFSET                                 ;
                                       0000'    CF  D6 000BB       INCL    INDEX                                              ; 0808
                                              FF67  31 000BF       BRW     1$                                                 ; 0688
                                                50  D4 000C2 8$:   CLRL    R0                                                 ; 0815
                                  5E            0C  C0 000C4 9$:   ADDL2   #12, SP                                            ; 0817
                                       00FC     8F  BA 000C7       POPR    #^M<R2,R3,R4,R5,R6,R7>                             ;
                                                    05 000CB       RSB                                                        ;

; Routine Size:  204 bytes,    Routine Base:  $CODE$ + 01BE
```

```
1 %SBTTL  'CHECK_LAST'
1 GLOBAL ROUTINE RECL$$CHECK_LAST : RL$JSB_REG_8 =
1 !++
1 !
1 ! Functional Description:
1 !
1 !       This routine checks to see if the current index record
1 !       indexed by INDEX is the last record in the bucket and if it
1 !       is the only record
1 !
1 ! Calling Sequence:
1 !
1 !       CHECK_LAST();
1 !
1 ! Input Parameters:
1 !       none
1 !
1 ! Implicit Inputs:
1 !
1 !       BUCKET          - address of buffer containing bucket
1 !       INDEX           - current index record (set by get_down_pointer
1 !
1 ! Output Parameters:
1 !
1 !       None.
1 !
1 ! Implicit Outputs:
1 !       none
1 !
1 ! Routine Value:
1 !
1 !       RECL$_SUCCESS   - index record IS the last in bucket and there more
1 !                         then one record in the bucket
1 !       RECL$_FAILURE   - index record IS NOT the last in bucket or is the
1 !                         only one in the bucket
1 !
1 ! Routines Called:
1 !
1 !       None.
1 !
1 ! Side Effects:
1 !
1 !       None.
1 !
1 !--
1
2     BEGIN
2
2     DEFINE_CTX;
2     DEFINE_BUCKET;
2     DEFINE_KEY_DESC;
2     DEFINE_KEY_POINTER;
2
2     LOCAL
2         VBN_OFFSET,
2         LAST_VBN_OFFSET;
2
```

```
;  882          0875  2       ! We can always reclaim the first record (even if its the last because the
;  883          0876  2       ! whole bucket will then be recalimed)
;  884          0877  2       !
;  885          0878  2       IF .INDEX EQL 0
;  886          0879  2       THEN
;  887          0880  2           RETURN RECL$_FAILURE;
;  888          0881  2
;  889          0882  2       ! Initialize offset in bucket to word containing VBN free space pointer
;  890          0883  2       ! so we can get the actual offset to the VBN free space.
;  891          0884  2       !
;  892          0885  2       VBN_OFFSET = .CTX [ CTX$W_BUCKET_SIZE ] - 2 - 2;
;  893          0886  2
;  894          0887  2       ! Get actual offset of the last VBN (free_space pointer + 1)
;  895          0888  2       !
;  896          0889  2       LAST_VBN_OFFSET = .BUCKET [ .VBN_OFFSET, 0, 16, 0 ] + 1;
;  897          0890  2
;  898          0891  2       ! Now point to the current VBN down pointer found by get_down_pointer
;  899          0892  2       !
;  900          0893  2       VBN_OFFSET = .VBN_OFFSET -
;  901          0894  2                   ( ( .BUCKET [ BKT$V_PTR_SZ ] + 2 ) * ( .INDEX + 1 ) );
;  902          0895  2
;  903          0896  2       ! If they are equal then this is the last record in the bucket
;  904          0897  2       !
;  905          0898  2       IF .VBN_OFFSET EQLU .LAST_VBN_OFFSET
;  906          0899  2       THEN
;  907          0900  2           RETURN RECL$_SUCCESS
;  908          0901  2       ELSE
;  909          0902  2           RETURN RECL$_FAILURE
;  910          0903  2
;  911          0904  1       END;



                                          0C  BB 00000 RECL$$CHECK_LAST::
                                                               PUSHR   #^M<R2,R3>                                  ; 0819
                             50      0000'  CF  D0 00002       MOVL    INDEX, R0                                   ; 0878
                                            2A  13 00007       BEQL    1$                                          ;
                             53        58   AA  3C 00009       MOVZWL  88(CTX), VBN_OFFSET                         ; 0885
                             53             03  C2 0000D       SUBL2   #3, VBN_OFFSET                              ;
                                          7349  9F 00010       PUSHAB  -(VBN_OFFSET)[BUCKET]                       ; 0889
                             52             9E  3C 00013       MOVZWL  @(SP)+, LAST_VBN_OFFSET                     ;
                                            52  D6 00016       INCL    LAST_VBN_OFFSET                             ;
         51      0D    A9    02             03  EF 00018       EXTZV   #3, #2, 13(BUCKET), R1                      ; 0894
                             51             02  C0 0001E       ADDL2   #2, R1                                      ;
                                            50  D6 00021       INCL    R0                                          ;
                             50             51  C4 00023       MULL2   R1, R0                                      ;
                             53             50  C2 00026       SUBL2   R0, VBN_OFFSET                              ;
                             52             53  D1 00029       CMPL    VBN_OFFSET, LAST_VBN_OFFSET                 ; 0898
                                            05  12 0002C       BNEQ    1$                                          ;
                             50             01  D0 0002E       MOVL    #1, R0                                      ; 0902
                                            02  11 00031       BRB     2$                                          ;
                                            50  D4 00033 1$:   CLRL    R0                                          ; 0904
                                            0C  BA 00035 2$:   POPR    #^M<R2,R3>                                  ;
                                            05     00037       RSB                                                 ;
```

```
; Routine Size:  56 bytes,    Routine Base:  $CODE$ + 028A
```

```
1 %SBTTL  'COMPARE_POINTER'
1 GLOBAL ROUTINE RECL$$COMPARE_POINTER ( VBN ) : RL$JSB_REG_8 =
1 !++
1 !
1 ! Functional Description:
1 !
1 !       This routine compares the next index record pointer in the current
1 !       buffer for the specified down pointer if necessary is reads in the next
1 !       bucket in the index chain to get the next index record.
1 !
1 ! Calling Sequence:
1 !
1 !       COMPARE_POINTER( VBN );
1 !
1 ! Input Parameters:
1 !
1 !       VBN     - VBN to compare
1 !
1 ! Implicit Inputs:
1 !
1 !       BUCKET              - address of buffer containing bucket
1 !       INDEX               - current index record (set by get_down_pointer
1 !
1 ! Output Parameters:
1 !
1 !       None.
1 !
1 ! Implicit Outputs:
1 !       none
1 !
1 ! Routine Value:
1 !
1 !       RECL$_SUCCESS   - next index record DOES point to the vbn
1 !       RECL$_FAILURE   - next index record DOES NOT point to the vbn
1 !
1 ! Routines Called:
1 !
1 !       None.
1 !
1 ! Side Effects:
1 !
1 !       None.
1 !
1 !--
1
2    BEGIN
2
2    DEFINE_CTX;
2    DEFINE_BUCKET;
2    DEFINE_KEY_DESC;
2    DEFINE_KEY_POINTER;
2
2    LOCAL
2        VBN_OFFSET,
2        LAST_VBN_OFFSET,
2        SEARCH_BUCKET   : REF BLOCK [ ,BYTE ];
2
```

```

! Initialize offset in bucket to word containing VBN free space pointer
! so we can get the actual offset to the VBN free space.
!
VBN_OFFSET = .CTX [ CTX$W_BUCKET_SIZE ] - 2 - 2;

! Get actual offset of the last VBN (free_space pointer + 1)
!
LAST_VBN_OFFSET = .BUCKET [ .VBN_OFFSET, 0, 16, 0 ] + 1;

! Now point to the current VBN down pointer found by get_down_pointer
!
VBN_OFFSET = .VBN_OFFSET -
            ( ( .BUCKET [ BKT$V_PTR_SZ ] + 2 ) * ( .INDEX + 1 ) );

! If this is not the end of the pointers then check the next vbn here
! else read in the next index bucket and search there
!
IF .VBN_OFFSET NEQU .LAST_VBN_OFFSET
THEN
    BEGIN

    ! Search in the current bucket
    !
    SEARCH_BUCKET = .BUCKET;

    ! Point to the next vbn
    !
    VBN_OFFSET = .VBN_OFFSET - ( .BUCKET [ BKT$V_PTR_SZ ] + 2 )

    END
ELSE

    ! Get the next bucket (if this is the last in the chain return failure)
    !
    IF .BUCKET [ BKT$V_LASTBKT ]
    THEN
        RETURN RECL$_FAILURE
    ELSE
        BEGIN

        ! Search in the search buffer
        !
        SEARCH_BUCKET = .RECL$GL_SEARCH_BUFFER;

        ! Read in the next index bucket
        !
        CONV$AB_OUT_RAB [ RAB$L_UBF ] = .SEARCH_BUCKET;
        CONV$AB_OUT_RAB [ RAB$W_USZ ] = .CTX [ CTX$W_BUCKET_SIZE ];
        CONV$AB_OUT_RAB [ RAB$L_BKT ] = .BUCKET [ BKT$L_NXTBKT ];

        $READ( RAB=CONV$AB_OUT_RAB,ERR=CONV$$RMS_READ_ERROR );

        ! Point to the first vbn there
        !
        VBN_OFFSET = .CTX [ CTX$W_BUCKET_SIZE ] - 2 - 2 -
                                    ( .SEARCH_BUCKET [ BKT$V_PTR_SZ ] + 2 )
```

```
; 1027      1019  4
; 1028      1020  2             END;
; 1029      1021  2
; 1030      1022  2       ! Compare the vbns
; 1031      1023  2       !
; 1032      1024  2       IF .VBN EQLU
; 1033      1025  2          .SEARCH_BUCKET [ .VBN_OFFSET,0,((.SEARCH_BUCKET[ BKT$V_PTR_SZ ]+2)*8),0 ]
; 1034      1026  2       THEN
; 1035      1027  2          RETURN RECL$_SUCCESS
; 1036      1028  2       ELSE
; 1037      1029  2          RETURN RECL$_FAILURE
; 1038      1030  2
; 1039      1031  1       END;


                                                                .EXTRN  SYS$READ

                                          1C  BB 00000 RECL$$COMPARE_POINTER::
                                                               PUSHR   #^M<R2,R3,R4>                               ; 0906
                         54        58  AA  3C 00002            MOVZWL  88(CTX), R4                                 ; 0966
                         52            74  DE 00006            MOVAL   -(R4), VBN_OFFSET                           ;
                                     6249  9F 00009            PUSHAB  (VBN_OFFSET)[BUCKET]                        ; 0970
                         53            9E  3C 0000C            MOVZWL  @(SP)+, LAST_VBN_OFFSET                     ;
                                       53  D6 0000F            INCL    LAST_VBN_OFFSET                             ;
51      0D    A9         02            03  EF 00011            EXTZV   #3, #2, 13(BUCKET), R1                      ; 0975
                         51            02  C0 00017            ADDL2   #2, R1                                      ;
              50  0000'  CF            01  C1 0001A            ADDL3   #1, INDEX, R0                               ;
                         50            51  C4 00020            MULL2   R1, R0                                      ;
                         52            50  C2 00023            SUBL2   R0, VBN_OFFSET                              ;
                         53            52  D1 00026            CMPL    VBN_OFFSET, LAST_VBN_OFFSET                 ; 0980
                                       08  13 00029            BEQL    1$                                          ;
                         53            59  D0 0002B            MOVL    BUCKET, SEARCH_BUCKET                       ; 0986
                         52            51  C2 0002E            SUBL2   R1, VBN_OFFSET                              ; 0990
                                       37  11 00031            BRB     2$                                          ;
                         50        0D  A9  E8 00033 1$:        BLBS    13(BUCKET), 3$                              ; 0997
                         53     0000G  CF  D0 00037            MOVL    RECL$GL_SEARCH_BUFFER, SEARCH_BUCKET        ; 1005
                  0000G  CF            53  D0 0003C            MOVL    SEARCH_BUCKET, CONV$AB_OUT_RAB+36           ; 1009
                  0000G  CF        58  AA  B0 00041            MOVW    88(CTX), CONV$AB_OUT_RAB+32                 ; 1010
                  0000G  CF        08  A9  D0 00047            MOVL    8(BUCKET), CONV$AB_OUT_RAB+56               ; 1011
                                0000G  CF  9F 0004D            PUSHAB  CONV$$RMS_READ_ERROR                        ; 1013
                                0000G  CF  9F 00051            PUSHAB  CONV$AB_OUT_RAB                             ;
              00000000G  00            02  FB 00055            CALLS   #2, SYS$READ                                ;
50      0D    A3         02            03  EF 0005C            EXTZV   #3, #2, 13(SEARCH_BUCKET), R0               ; 1018
              50         54            50  C3 00062            SUBL3   R0, R4, R0                                  ;
                         52        FE  A0  9E 00066            MOVAB   -2(R0), VBN_OFFSET                          ; 1017
50      0D    A3         02            03  EF 0006A 2$:        EXTZV   #3, #2, 13(SEARCH_BUCKET), R0               ; 1025
                         50            08  C4 00070            MULL2   #8, R0                                      ;
                         50            10  C0 00073            ADDL2   #16, R0                                     ;
51          6243         50            00  EF 00076            EXTZV   #0, R0, (VBN_OFFSET)[SEARCH_BUCKET], R1     ;
                         51        10  AE  D1 0007C            CMPL    VBN, R1                                     ;
                                       05  12 00080            BNEQ    3$                                          ;
                         50            01  D0 00082            MOVL    #1, R0                                      ; 1029
                                       02  11 00085            BRB     4$                                          ;
                                       50  D4 00087 3$:        CLRL    R0                                          ; 1031
                                       1C  BA 00089 4$:        POPR    #^M<R2,R3,R4>                               ;
                                           05 0008B            RSB                                                 ;
```

```
; Routine Size:  140 bytes,    Routine Base:  $CODE$ + 02C2
```

```
1 %SBTTL  'SWING_POINTER'
1 GLOBAL ROUTINE RECL$$SWING_POINTER ( VBN ) : RL$JSB_REG_8 NOVALUE =
1 !++
1 !
1 ! Functional Description:
1 !
1 !       This routine will stuff the VBN into the curretn index record
1 !
1 ! Calling Sequence:
1 !
1 !       SWING_POINTER( VBN );
1 !
1 ! Input Parameters:
1 !
1 !       VBN      - VBN to stuff
1 !
1 ! Implicit Inputs:
1 !
1 !       BUCKET             - address of buffer containing bucket
1 !       INDEX              - index record to stuff
1 !
1 ! Output Parameters:
1 !
1 !       None.
1 !
1 ! Implicit Outputs:
1 !       none
1 !
1 ! Routine Value:
1 !       none
1 !
1 ! Routines Called:
1 !
1 !       None.
1 !
1 ! Side Effects:
1 !
1 !       None.
1 !
1 !--
1
2    BEGIN
2
2    DEFINE_CTX;
2    DEFINE_BUCKET;
2    DEFINE_KEY_DESC;
2    DEFINE_KEY_POINTER;
2
2    LOCAL
2       VBN_OFFSET;
2
2    ! Point to current VBN down pointer
2    ! Which is: Bucket size - 2 bytes for check and spare - 2 bytes for
2    !           vbn freespace pointer - index into the array
2    !
2    VBN_OFFSET = .CTX [ CTX$W_BUCKET_SIZE ] - 2 - 2 -
2                         ( .BUCKET [ BKT$V_PTR_SZ ] + 2 ) * ( .INDEX + 1 );
```

```
; 1098          1089  2
; 1099          1090  2       ! Stuff the vbn
; 1100          1091  2       !
; 1101          1092  2       BUCKET [ .VBN_OFFSET,0,( ( .BUCKET [ BKT$V_PTR_SZ ] + 2 ) * 8 ),0 ] = .VBN;
; 1102          1093  2
; 1103          1094  2       RETURN
; 1104          1095  2
; 1105          1096  1       END;



                                                  0C  BB 00000 RECL$$SWING_POINTER::
                                                                     PUSHR    #^M<R2,R3>                                  ; 1033
     52        0D    A9           02              03  EF 00002       EXTZV    #3, #2, 13(BUCKET), R2                      ; 1088
                                  51        02    A2  9E 00008       MOVAB    2(R2), R1                                   ;
                     50    0000'  CF              01  C1 0000C       ADDL3    #1, INDEX, R0                               ;
                                  50              51  C4 00012       MULL2    R1, R0                                      ;
                                  53        58    AA  3C 00015       MOVZWL   88(CTX), R3                                 ;
                     50           53              50  C3 00019       SUBL3    R0, R3, R0                                  ;
                                  50              03  C2 0001D       SUBL2    #3, VBN_OFFSET                              ; 1087
                     51           52              03  78 00020       ASHL     #3, R2, R1                                  ; 1092
                                  51              10  C0 00024       ADDL2    #16, R1                                     ;
   7049              51           00        0C    AE  F0 00027       INSV     VBN, #0, R1, -(VBN_OFFSET)[BUCKET]          ;
                                                  0C  BA 0002E       POPR     #^M<R2,R3>                                  ; 1096
                                                      05 00030       RSB                                                  ;

; Routine Size:  49 bytes,    Routine Base:  $CODE$ + 034E
```

```
1 %SBTTL  'REMOVE_INDEX_RECORD'
1 GLOBAL ROUTINE RECL$$REMOVE_INDEX_RECORD : RL$JSB_REG_8 NOVALUE =
1 !++
1 !
1 ! Functional Description:
1 !
1 !       This routine actually squishes out the index record from the index
1 !       bucket.
1 !
1 ! Calling Sequence:
1 !
1 !       REMOVE_INDEX_RECORD();
1 !
1 ! Input Parameters:
1 !
1 !       None.
1 !
1 ! Implicit Inputs:
1 !
1 !       INDEX           - number of the index record to remove
1 !       KEY_POINTER     - points to key part of index record to remove
1 !       KEY_BUFFER_1    - contains fully expanded previous key
1 !       KEY_BUFFER_2    - contains fully expanded current key
1 !       BUCKET          - points to buffer containing bucket
1 !
1 ! Output Parameters:
1 !
1 !       None.
1 !
1 ! Implicit Outputs:
1 !
1 !       Index bucket has more freespace, since a record was squished out.
1 !
1 ! Routine Value:
1 !
1 !       None.
1 !
1 ! Routines Called:
1 !
1 !       RECOMPRESS_RECORD
1 !
1 ! Side Effects:
1 !
1 !       None.
1 !
1 !--
1
2     BEGIN
2
2     DEFINE_CTX;
2     DEFINE_BUCKET;
2     DEFINE_KEY_DESC;
2     DEFINE_KEY_POINTER;
2
2     !++
2     !
2     ! Squish out the VBN part of the index record
```

```
2       !
2       !--
2
3       BEGIN
3
3       LOCAL
3           OFFSET,             ! Offset to the vbn_freespace pointer
3           VBN_SIZE,           ! Size of vbn in bytes
3           BITS,               ! Size of vbn in bits
3           FREESPACE,          ! Pointer offset to the top of the vbns
3           VBN,                ! Pointer offset to the vbn to remove
3           SOURCE,             ! Pointer offset to the Source
3           DEST;               ! Pointer offset to the Destination
3
3       ! Find the offset to the vbn freespace pointer
3       !
3       OFFSET = .CTX [ CTX$W_BUCKET_SIZE ] - 4;
3
3       ! Get the size of the vbns in bytes
3       !
3       VBN_SIZE = .BUCKET [ BKT$V_PTR_SZ ] + 2;
3
3       ! Now get it in bits
3       !
3       BITS = .VBN_SIZE * 8;
3
3       ! Find the top the vbns
3       !
3       FREESPACE = .BUCKET [ .OFFSET,0,16,0 ];
3
3       ! Find the vbn we want to remove
3       !
3       VBN = .OFFSET - ( .VBN_SIZE * ( .INDEX + 1 ) );
3
3       ! Set up the destindtion
3       !
3       DEST = .VBN;
3
3       ! And the source
3       !
3       SOURCE = .DEST - .VBN_SIZE;
3
3       ! Do each vbn
3       !
3       WHILE .SOURCE GEQU .FREESPACE
3       DO
4           BEGIN
4
4           ! Copy the vbn to the new location
4           !
4           BUCKET [ .DEST,0,.BITS,0 ] = .BUCKET [ .SOURCE,0,.BITS,0 ];
4
4           ! Update the pointers
4           !
4           DEST = .DEST - .VBN_SIZE;
4           SOURCE = .SOURCE - .VBN_SIZE
4
```

```
          END;

        ! Update the freespace pointer in the bucket
        !
        BUCKET [ .OFFSET,0,16,0 ] = .FREESPACE + .VBN_SIZE;

        ! If freespace pointer points to the bottom of the bucket it is
        ! empty so don't bother to fool with the data part (but do set
        ! the keyfreespace pointer)
        !
        IF .BUCKET [ .OFFSET,0,16,0 ] EQLU .OFFSET
        THEN
            BEGIN
            BUCKET[ BKT$W_KEYFRESPC ] = BKT$C_OVERHDSZ;
            RETURN
            END

        END;

        !++
        !
        ! Squeeze out the KEY part of the index record
        !
        !--

        BEGIN

        LOCAL
            DELETE_SIZE;

        ! Calculate from address and size for squish differently if index is
        ! compressed or not.  Also do KEYFRESPC depending on index compression.
        !
        IF .KEY_DESC[ KEY$V_IDX_COMPR ]
        THEN
            BEGIN

            LOCAL
                NEXT         : REF BLOCK [ ,BYTE ];   ! Pointer to the next key
                                                      ! to replace the deleted one

            ! The size of the deleted space is size of the old record MINUS
            ! the DIFFERENCE between the size of next record before compression
            ! and the size of it after compression.
            ! First save the size of old record.
            !
            DELETE_SIZE = .KEY_POINTER [ KEYR$B_LENGTH ] + 2;

            ! Next thing to do is recompress the next record after the current
            ! one we start by coping it into key_buffer_2 (where the to-be-deleted
            ! key is)
            !
            NEXT = .KEY_POINTER + .KEY_POINTER [ KEYR$B_LENGTH ] + 2;

            ! If there IS a next key then copy it and compress it
            !
            IF .NEXT LSSU ( .BUCKET + .BUCKET [ BKT$W_KEYFRESPC ] )
```

```
    THEN
        BEGIN

        LOCAL OLD_SIZE;

        ! Save the old size of the next record
        !
        OLD_SIZE = .NEXT [ KEYR$B_LENGTH ];

        ! Copy the next key while expanding the rear
        !
        CH$COPY( .NEXT [ KEYR$B_LENGTH ],
                 .NEXT + 2,
                 .( .NEXT + 1 + .NEXT [ KEYR$B_LENGTH ] ),
                 .KEY_DESC [ KEY$B_KEYSZ ] - .NEXT [ KEYR$B_FRONT_COUNT ],
                 KEY_BUFFER_2 + 2 + .NEXT [ KEYR$B_FRONT_COUNT ] );

        ! Recompress the new key in key_buffer_2
        !
        RECOMPRESS_RECORD();

        ! Key_buffer_2 now contains a compressed key (w/control info)
        ! so move it into the bucket
        !
        CH$MOVE( .KEY_BUFFER_2 [ KEYR$B_LENGTH ] + 2,
                 KEY_BUFFER_2,
                 .KEY_POINTER );

        ! Now we can figure the ammount of space deleted
        !
        DELETE_SIZE = .DELETE_SIZE -
                        ( .KEY_POINTER [ KEYR$B_LENGTH ] - .OLD_SIZE );

        ! We must now move the rest of the keys in the bucket
        !
        CH$MOVE( ( .BUCKET + .BUCKET [ BKT$W_KEYFRESPC ] ) -
                                     ( .NEXT + .NEXT [ KEYR$B_LENGTH ] + 2 ),
                 .NEXT + .NEXT [ KEYR$B_LENGTH ] + 2,
                 .KEY_POINTER + .KEY_POINTER [ KEYR$B_LENGTH ] + 2 )

        END;
    END
ELSE
    BEGIN

    ! Set the delete size
    !
    DELETE_SIZE = .KEY_DESC [ KEY$B_KEYSZ ];

    ! Move the rest of the keys
    !
    CH$MOVE( ( .BUCKET + .BUCKET [ BKT$W_KEYFRESPC ] ) -
                            ( .KEY_POINTER + .KEY_DESC [ KEY$B_KEYSZ ] ),
             .KEY_POINTER + .KEY_DESC [ KEY$B_KEYSZ ],
             .KEY_POINTER )
```

```
                                                              B 15
RECL$REC        VAX-11 CONVERT/RECLAIM                        15-Sep-1984 23:59:42   VAX-11 Bliss-32 V4.0-742                Page  37
V04-000         REMOVE_INDEX_RECORD                           14-Sep-1984 12:14:05   DISK$VMSMASTER:[CONV.SRC]RECLREC.B32;1   (11)

; 1335          1325 3           END;
; 1336          1326 3
; 1337          1327 3        ! Update KEYFRESPC since we squished out a key
; 1338          1328 3        !
; 1339          1329 3        BUCKET [ BKT$W_KEYFRESPC ] = .BUCKET [ BKT$W_KEYFRESPC ] - .DELETE_SIZE
; 1340          1330 3
; 1341          1331 2        END;
; 1342          1332 2
; 1343          1333 2        RETURN
; 1344          1334 2
; 1345          1335 1        END;



                                    00FC     8F   BB 00000 RECL$$REMOVE_INDEX_RECORD::
                                                                    PUSHR    #^M<R2,R3,R4,R5,R6,R7>                          ; 1098
                              5E          04   C2 00004             SUBL2    #4, SP
                              51     58   AA   3C 00007             MOVZWL   88(CTX), OFFSET                                 ; 1170
                              51          03   C2 0000B             SUBL2    #3, OFFSET
   52     0D   A9             02          03   EF 0000E             EXTZV    #3, #2, 13(BUCKET), VBN_SIZE                    ; 1174
                              52          02   C0 00014             ADDL2    #2, VBN_SIZE
               56             52          03   78 00017             ASHL     #3, VBN_SIZE, BITS                              ; 1178
                                        7149   9F 0001B             PUSHAB   -(OFFSET)[BUCKET]                               ; 1182
                              55          9E   3C 0001E             MOVZWL   @(SP)+, FREESPACE
               50    0000'    CF          01   C1 00021             ADDL3    #1, INDEX, R0                                   ; 1186
                              50          52   C4 00027             MULL2    VBN_SIZE, R0
               50             51          50   C3 0002A             SUBL3    R0, OFFSET, VBN
                              53          50   D0 0002E             MOVL     VBN, DEST                                       ; 1190
               54             53          52   C3 00031             SUBL3    VBN_SIZE, DEST, SOURCE                          ; 1194
                              55          54   D1 00035  1$:        CMPL     SOURCE, FREESPACE                               ; 1198
                                          14   1F 00038             BLSSU    2$
   50        6449             56          00   EF 0003A             EXTZV    #0, BITS, (SOURCE)[BUCKET], R0                  ; 1204
 6349          56             00          50   F0 00040             INSV     R0, #0, BITS, (DEST)[BUCKET]
                              53          52   C2 00046             SUBL2    VBN_SIZE, DEST                                  ; 1208
                              54          52   C2 00049             SUBL2    VBN_SIZE, SOURCE                                ; 1209
                                          E7   11 0004C             BRB      1$
                                        6149   9F 0004E  2$:        PUSHAB   (OFFSET)[BUCKET]                                ; 1215
               9E             55          52   A1 00051             ADDW3    VBN_SIZE, FREESPACE, @(SP)+
   51        6149             10          00   ED 00055             CMPZV    #0, #16, (OFFSET)[BUCKET], OFFSET               ; 1221
                                          07   12 0005B             BNEQ     3$
                      04      A9          0E   B0 0005D             MOVW     #14, 4(BUCKET)                                  ; 1224
                                        0091   31 00061             BRW      6$                                              ; 1223
               6E     10      AB          03   E1 00064  3$:        BBC      #3, 16(KEY_DESC), 4$                            ; 1244
                              50          68   9A 00069             MOVZBL   (KEY_POINTER), R0                               ; 1257
                              57     02   A0   9E 0006C             MOVAB    2(R0), DELETE_SIZE
                              51     02 A048   9E 00070             MOVAB    2(R0)[KEY_POINTER], NEXT                        ; 1263
                              50     04   A9   3C 00075             MOVZWL   4(BUCKET), R0                                   ; 1267
                              50          59   C0 00079             ADDL2    BUCKET, R0
                              50          51   D1 0007C             CMPL     NEXT, R0
                                          70   1E 0007F             BGEQU    5$
                              53          61   9A 00081             MOVZBL   (NEXT), R3                                      ; 1275
                              56          53   D0 00084             MOVL     R3, OLD_SIZE
               6E             51          53   C1 00087             ADDL3    R3, NEXT, (SP)                                  ; 1281
                              50     01   A1   9A 0008B             MOVZBL   1(NEXT), R0                                     ; 1282
                              52     14   AB   9A 0008F             MOVZBL   20(KEY_DESC), R2                                ;
```

```
                                52          50  C2 00093            SUBL2   R0, R2                                          ;
                         7E     6E          01  C1 00096            ADDL3   #1, (SP), -(SP)                                 ; 1283
      52                 9E     02   A1     53  2C 0009A            MOVC5   R3, 2(NEXT), @(SP)+, R2, KEY_BUFFER_2+2[R0]     ;
                                      0000'CF40    000A0                                                                    ;
                                        0000V  30 000A4             BSBW    RECOMPRESS_RECORD                               ; 1287
                         50           0000'  CF  9A 000A7           MOVZBL  KEY_BUFFER_2, R0                                ; 1292
                                50          02  C0 000AC            ADDL2   #2, R0                                          ;
                         68    0000'    CF  50  28 000AF            MOVC3   R0, KEY_BUFFER_2, (KEY_POINTER)                 ; 1294
                                51          68  9A 000B5            MOVZBL  (KEY_POINTER), R1                               ; 1299
                                56          51  C2 000B8            SUBL2   R1, R6                                          ;
                                57          56  C0 000BB            ADDL2   R6, DELETE_SIZE                                 ;
                                50     04   A9  3C 000BE            MOVZWL  4(BUCKET), R0                                   ; 1303
                                50          59  C0 000C2            ADDL2   BUCKET, R0                                      ;
                                50          6E  C2 000C5            SUBL2   (SP), R0                                        ; 1304
                                50          02  C2 000C8            SUBL2   #2, R0                                          ; 1303
                                52          6E  D0 000CB            MOVL    (SP), R2                                        ; 1305
                    02 A148     02   A2     50  28 000CE            MOVC3   R0, 2(R2), 2(R1)[KEY_POINTER]                   ; 1306
                                            1A  11 000D5            BRB     5$                                              ; 1244
                                52     14   AB  9A 000D7 4$:        MOVZBL  20(KEY_DESC), R2                                ; 1316
                                57          52  D0 000DB            MOVL    R2, DELETE_SIZE                                 ;
                                50     04   A9  3C 000DE            MOVZWL  4(BUCKET), R0                                   ; 1320
                         51     59          50  C1 000E2            ADDL3   R0, BUCKET, R1                                  ;
                         50     58          52  C1 000E6            ADDL3   R2, KEY_POINTER, R0                             ; 1321
                                51          50  C2 000EA            SUBL2   R0, R1                                          ;
                         68     60          51  28 000ED            MOVC3   R1, (R0), (KEY_POINTER)                         ; 1323
                         04     A9          57  A2 000F1 5$:        SUBW2   DELETE_SIZE, 4(BUCKET)                          ; 1329
                                5E          04  C0 000F5 6$:        ADDL2   #4, SP                                          ; 1335
                                      00FC  8F  BA 000F8            POPR    #^M<R2,R3,R4,R5,R6,R7>                          ;
                                                05 000FC            RSB                                                     ;

; Routine Size:  253 bytes,    Routine Base:  $CODE$ + 037F


; 1346         1336  1
```

```
1 %SBTTL  'RECOMPRESS_RECORD'
1 ROUTINE RECOMPRESS_RECORD : RL$JSB_REG_8 NOVALUE =
1 !++
1 !
1 ! Functional Description:
1 !
1 !       This routine will recompress the index record in key_buffer_2
1 !
1 ! Calling Sequence:
1 !
1 !       RECOMPRESS_RECORD()
1 !
1 ! Input Parameters:
1 !
1 !       None.
1 !
1 ! Implicit Inputs:
1 !
1 !       KEY_BUFFER_1     - contains expanded key to base re-compression upon
1 !       KEY_BUFFER_2     - contains expanded key to re-compress
1 !
1 ! Output Parameters:
1 !
1 !       None.
1 !
1 ! Implicit Outputs:
1 !
1 !       None.
1 !
1 ! Routine Value:
1 !
1 !       None.
1 !
1 ! Routines Called:
1 !
1 !       None.
1 !
1 ! Side Effects:
1 !
1 !       Index record in key_buffer_2 is compressed.
1 !
1 !--
1
2     BEGIN
2
2     DEFINE_CTX;
2     DEFINE_BUCKET;
2     DEFINE_KEY_DESC;
2     DEFINE_KEY_POINTER;
2
2     BIND
2         KEY_1 = KEY_BUFFER_1 + 2 : VECTOR [ ,BYTE ],     ! Key part of the record
2         KEY_2 = KEY_BUFFER_2 + 2 : VECTOR [ ,BYTE ];
2
2     LOCAL
2         LENGTH;
2
```

```
	! Assume no compression
	!
	KEY_BUFFER_2 [ KEYR$B_FRONT_COUNT ] = 0;

	LENGTH = .KEY_DESC [ KEY$B_KEYSZ ];

	! If this is NOT the first key in the bucket do front compression
	!
	IF .INDEX NEQU 0
	THEN

	    ! Find the first position where the two keys differ
	    !
	    INCR I FROM 0 TO ( .LENGTH - 1 ) BY 1
	    DO

		! If the characters are not equal we found the end
		!
		IF ( .KEY_1 [ .I ] NEQU .KEY_2 [ .I ] )
		THEN
		    BEGIN

		    ! I is now the number of compressed characters
		    !
		    KEY_BUFFER_2 [ KEYR$B_FRONT_COUNT ] = .I;

		    ! Shorten the length
		    !
		    LENGTH = .LENGTH - .I;

		    ! If there was some compression move the key a little
		    !
		    IF .I NEQU 0
		    THEN
			CH$MOVE( .LENGTH, KEY_2 + .I ,KEY_2 );

		    EXITLOOP

		    END;

	! Do rear end truncation
	!
	WHILE .LENGTH GTRU 1
	DO

	    ! If the trailing characters are the same cut it short
	    !
	    IF .KEY_2 [ .LENGTH - 1 ] EQLU .KEY_2 [ .LENGTH - 2 ]
	    THEN
		LENGTH = .LENGTH - 1
	    ELSE
		EXITLOOP;

	! Set the length field
	!
	KEY_BUFFER_2 [ KEYR$B_LENGTH ] = .LENGTH;
```

```
; 1462          1451  2      RETURN
; 1463          1452  2
; 1464          1453  1      END;



                                                                   KEY_1=                  KEY_BUFFER_1+2
                                                                   KEY_2=                  KEY_BUFFER_2+2

                                               00FC    8F    BB 00000 RECOMPRESS_RECORD:
                                                                              PUSHR    #^M<R2,R3,R4,R5,R6,R7>                        ; 1338
                                  5E                 04    C2 00004           SUBL2    #4, SP
                                               0000'   CF    94 00007           CLRB     KEY_BUFFER_2+1                                ; 1396
                                  56              14   AB    9A 0000B           MOVZBL   20(KEY_DESC), LENGTH                          ; 1398
                                               0000'   CF    D5 0000F           TSTL     INDEX                                         ; 1402
                                                     2E    13 00013           BEQL     3$                                            ;
                                  6E                 56    D0 00015           MOVL     LENGTH, (SP)                                  ; 1407
                                  57                 01    CE 00018           MNEGL    #1, I                                         ;
                                                     22    11 0001B           BRB      2$                                            ;
                        0000'CF47              0000'CF47    91 0001D 1$:       CMPB     KEY_1[I], KEY_2[I]                            ; 1412
                                                     17    13 00026           BEQL     2$                                            ;
                        0000'   CF                   57    90 00028           MOVB     I, KEY_BUFFER_2+1                             ; 1418
                                  56                 57    C2 0002D           SUBL2    I, LENGTH                                     ; 1422
                                                     57    D5 00030           TSTL     I                                             ; 1426
                                                     0F    13 00032           BEQL     3$                                            ;
             0000'    CF        0000'CF47            56    28 00034           MOVC3    LENGTH, KEY_2[I], KEY_2                       ; 1428
                                                     04    11 0003D           BRB      3$                                            ; 1414
                      DA          57                 6E    F2 0003F 2$:       AOBLSS   (SP), I, 1$                                   ; 1412
                                  01                 56    D1 00043 3$:       CMPL     LENGTH, #1                                    ; 1436
                                                     0F    1B 00046           BLEQU    4$                                            ;
                        0000'CF46              0000'CF46    91 00048           CMPB     KEY_2-1[LENGTH], KEY_2-2[LENGTH]              ; 1441
                                                     04    12 00051           BNEQ     4$                                            ;
                                                     56    D7 00053           DECL     LENGTH                                        ; 1443
                                                     EC    11 00055           BRB      3$                                            ;
                        0000'   CF                   56    90 00057 4$:       MOVB     LENGTH, KEY_BUFFER_2                          ; 1449
                                  5E                 04    C0 0005C           ADDL2    #4, SP                                        ; 1453
                                               00FC    8F    BA 0005F           POPR     #^M<R2,R3,R4,R5,R6,R7>                        ;
                                                           05 00063           RSB                                                    ;

; Routine Size:  100 bytes,    Routine Base:  $CODE$ + 047C


; 1465          1454  1
; 1466          1455  0 END ELUDOM




;                                   PSECT SUMMARY
;
;        Name                     Bytes                         Attributes
;
;   $OWN$                           529  NOVEC,  WRT,  RD ,NOEXE,NOSHR,  LCL,  REL,  CON,NOPIC,ALIGN(2)
;   $CODE$                         1248  NOVEC,NOWRT,  RD ,  EXE,NOSHR,  LCL,  REL,  CON,NOPIC,ALIGN(2)
;                               Library Statistics
;
;                                                    -------- Symbols --------       Pages        Processing
;       File                                         Total    Loaded   Percent       Mapped       Time
;
;  _$255$DUA28:[SYSLIB]LIB.L32;1                     18619        34         0       1000           00:01.9
;  _$255$DUA28:[CONV.SRC]CONVERT.L32;1                 165        11         6         17           00:00.2




;                                      COMMAND QUALIFIERS

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

; Size:          1248 code + 529 data bytes
; Run Time:         00:30.1
; Elapsed Time:     01:45.3
; Lines/CPU Min:     2903
; Lexemes/CPU-Min: 15252
; Memory Used:  148 pages
; Compilation Complete
```

0066 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

RECLDCL
LIS

RECLREC
LIS

RECLCTRL
LIS

RECLRMSIO
LIS

CONVMSG
LIS

CONVMAIN
LIS

CONVSORT
LIS

CONVVEC
LIS