```
RRRRRRRRRRR      MMM          MMM     SSSSSSSSSSS
RRRRRRRRRRRR     MMM          MMM     SSSSSSSSSSS
RRRRRRRRRRRR     MMM          MMM     SSSSSSSSSSS
RRR         RRR  MMMMMM    MMMMMM  SSS
RRR         RRR  MMMMMM    MMMMMM  SSS
RRR         RRR  MMMMMM    MMMMMM  SSS
RRR         RRR  MMM   MMM   MMM   SSS
RRR         RRR  MMM   MMM   MMM   SSS
RRR         RRR  MMM   MMM   MMM   SSS
RRRRRRRRRRR      MMM          MMM     SSSSSSSSS
RRRRRRRRRRRR     MMM          MMM     SSSSSSSSS
RRRRRRRRRRRR     MMM          MMM     SSSSSSSSS
RRR   RRR        MMM          MMM             SSS
RRR   RRR        MMM          MMM             SSS
RRR   RRR        MMM          MMM             SSS
RRR      RRR     MMM          MMM             SSS
RRR      RRR     MMM          MMM             SSS
RRR      RRR     MMM          MMM             SSS
RRR         RRR  MMM          MMM  SSSSSSSSSSS
RRR         RRR  MMM          MMM  SSSSSSSSSSS
RRR         RRR  MMM          MMM  SSSSSSSSSSS
```

```
RRRRRRRR    MM      MM    333333     SSSSSSSS    IIIIII    DDDDDDDD    XX      XX    SSSSSSSS  PPPPPPPP
RRRRRRRR    MM      MM    333333     SSSSSSSS    IIIIII    DDDDDDDD    XX      XX    SSSSSSSS  PPPPPPPP
RR      RR  MMMM  MMMM  33      33  SS            II      DD      DD  XX      XX  SS          PP      PP
RR      RR  MMMM  MMMM  33      33  SS            II      DD      DD  XX      XX  SS          PP      PP
RR      RR  MM  MM  MM        33    SS            II      DD      DD    XX  XX    SS          PP      PP
RR      RR  MM  MM  MM        33    SS            II      DD      DD    XX  XX    SS          PP      PP
RRRRRRRR    MM      MM      33      SSSSSS        II      DD      DD      XX      SSSSSS    PPPPPPPP
RRRRRRRR    MM      MM      33      SSSSSS        II      DD      DD      XX      SSSSSS    PPPPPPPP
RR   RR     MM      MM        33          SS      II      DD      DD    XX  XX          SS  PP
RR   RR     MM      MM        33          SS      II      DD      DD    XX  XX          SS  PP
RR    RR    MM      MM  33    33          SS      II      DD      DD  XX      XX      SS  PP
RR    RR    MM      MM  33    33          SS      II      DD      DD  XX      XX      SS  PP
RR      RR  MM      MM    333333    SSSSSSSS    IIIIII    DDDDDDDD    XX      XX  SSSSSSSS  PP
RR      RR  MM      MM    333333    SSSSSSSS    IIIIII    DDDDDDDD    XX      XX  SSSSSSSS  PP


LL              IIIIII      SSSSSSSS
LL              IIIIII      SSSSSSSS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II        SSSSSS
LL                II        SSSSSS
LL                II              SS
LL                II              SS
LL                II              SS
LL                II              SS
LLLLLLLLLL      IIIIII    SSSSSSSS
LLLLLLLLLL      IIIIII    SSSSSSSS
```

```
;     1          0001  0 MODULE RM3SIDXSP (LANGUAGE (BLISS32) ,
;     2          0002  0                    IDENT = 'V04-000'
;     3          0003  0                    ) =
;     4          0004  1 BEGIN
;     5          0005  1 !
;     6          0006  1 !**********************************************************************
;     7          0007  1 !*                                                                    *
;     8          0008  1 !*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                          *
;     9          0009  1 !*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.           *
;    10          0010  1 !*  ALL RIGHTS RESERVED.                                           *
;    11          0011  1 !*                                                                    *
;    12          0012  1 !*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED  *
;    13          0013  1 !*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE  *
;    14          0014  1 !*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER  *
;    15          0015  1 !*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY  *
;    16          0016  1 !*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY  *
;    17          0017  1 !*  TRANSFERRED.                                                       *
;    18          0018  1 !*                                                                    *
;    19          0019  1 !*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE  *
;    20          0020  1 !*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT  *
;    21          0021  1 !*  CORPORATION.                                                       *
;    22          0022  1 !*                                                                    *
;    23          0023  1 !*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS  *
;    24          0024  1 !*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.            *
;    25          0025  1 !*                                                                    *
;    26          0026  1 !*                                                                    *
;    27          0027  1 !**********************************************************************
;    28          0028  1
;    29          0029  1 !++
;    30          0030  1 !
;    31          0031  1 ! FACILITY:     RMS32 INDEX SEQUENTIAL FILE ORGANIZATION
;    32          0032  1 !
;    33          0033  1 ! ABSTRACT:
;    34          0034  1 !                Split SIDR data level or any index level 50-50
;    35          0035  1 !
;    36          0036  1 !
;    37          0037  1 ! ENVIRONMENT:
;    38          0038  1 !
;    39          0039  1 !                VAX/VMS OPERATING SYSTEM
;    40          0040  1 !
;    41          0041  1 !--
;    42          0042  1
;    43          0043  1 !
;    44          0044  1 ! AUTHOR:        Christian Saether          CREATION DATE:            1-AUG-78  10:18
;    45          0045  1 !
;    46          0046  1 !
;    47          0047  1 ! MODIFIED BY:
;    48          0048  1 !
;    49          0049  1 !       V03-015 MCN0004         Maria del C. Nasr       04-Apr-1983
;    50          0050  1 !               Change linkage of RM$COMPRESS_KEY.
;    51          0051  1 !
;    52          0052  1 !       V03-014 MCN0003         Maria del C. Nasr       22-Mar-1983
;    53          0053  1 !               More changes in the linkages
;    54          0054  1 !
;    55          0055  1 !       V03-013 MCN0002         Maria del C. Nasr       28-Feb-1983
;    56          0056  1 !               Reorganize linkages
;    57          0057  1 !
```

```
;    58          0058  1 !       V03-012 TMK0010         Todd M. Katz            05-Nov-1982
;    59          0059  1 !               Document SIDR bucket splits, much as index bucket splits were
;    60          0060  1 !               documented by TMK0005.
;    61          0061  1 !
;    62          0062  1 !       V03-011 TMK0009         Todd M. Katz            02-Oct-1982
;    63          0063  1 !               I made a mistake when I re-wrote the routine RM$EXT_HIGH_KEY.
;    64          0064  1 !               If the bucket whose key is to be extracted is a prologue 3 index
;    65          0065  1 !               bucket with non-compressed keys, then there is no need to scan
;    66          0066  1 !               the bucket to position to the high key of the bucket because all
;    67          0067  1 !               the records are of a fixed size. I made an error in making this
;    68          0068  1 !               check. What I ended up checking for was for a prologue 3 index
;    69          0069  1 !               bucket with compressed keys rather than a prologue 3 bucket with
;    70          0070  1 !               non-compressed keys.
;    71          0071  1 !
;    72          0072  1 !       V03-010 TMK0008         Todd M. Katz            14-Sep-1982
;    73          0073  1 !               Add support for prologue 3 SIDRs. This involves modifications
;    74          0074  1 !               to RM$SPLIT_EM, RM$COMP_SPL_PNT, and RM$MOVE_VBN, and the
;    75          0075  1 !               re-write of RM$EXT_HIGH_KEY.
;    76          0076  1 !
;    77          0077  1 !       V03-009 KBT0233         Keith B. Thompson       23-Aug-1982
;    78          0078  1 !               Reorganize psects
;    79          0079  1 !
;    80          0080  1 !       V03-008 TMK0007         Todd M. Katz            02-Jul-1982
;    81          0081  1 !               Implement the RMS cluster solution for next record positioning.
;    82          0082  1 !               This means that code must be added to reassign the ID's in
;    83          0083  1 !               the new SIDR bucket following a SIDR bucket split instead of
;    84          0084  1 !               calling the routine RM$ID_AND_NRP to do this (because that
;    85          0085  1 !               routine along with all the others in the module RM3NRP are
;    86          0086  1 !               disappearing.
;    87          0087  1 !
;    88          0088  1 !       V03-007 TMK0006         Todd M. Katz            16-Jun-1982
;    89          0089  1 !               Make a change to the routine COMP_SPL_PNT which affects how the
;    90          0090  1 !               split point of a prologue 3 index bucket with compressed keys
;    91          0091  1 !               is calculated. In such a bucket, the keys are separate from
;    92          0092  1 !               the VBNs, and only the keys contribute to the end-of-bucket
;    93          0093  1 !               calculation. However, both the keysize and the VBN size of the
;    94          0094  1 !               new index record(s) to be added where influencing where the
;    95          0095  1 !               split point should be instead of just the key size. This is
;    96          0096  1 !               wrong, and lead to a nonoptimum calculation of the index
;    97          0097  1 !               bucket split point. Change this such that only the key size of
;    98          0098  1 !               the new record(s) influences where the index bucket split point
;    99          0099  1 !               is computed to be.
;   100          0100  1 !
;   101          0101  1 !       V03-006 TMK0005         Todd M. Katz            15-Jun-1982
;   102          0102  1 !               Add documentation at the fron of the routine for index bucket
;   103          0103  1 !               update and split cases.
;   104          0104  1 !
;   105          0105  1 !       V03-005 TMK0004         Todd M. Katz            11-Jun-1982
;   106          0106  1 !               When the code was added for the missing index bucket split
;   107          0107  1 !               cases in TMK0002, it accidently broke the two-pass multi-bucket
;   108          0108  1 !               split cases where one of the two new index records goes in
;   109          0109  1 !               each of the index buckets resulting from the split. The reason
;   110          0110  1 !               for this is that I set the position of insertion of the new
;   111          0111  1 !               records in the new (right) index bucket, IRB$W_POS_INS, to be
;   112          0112  1 !               at the beginning of the bucket, before that parameter could
;   113          0113  1 !               be used to diffrentiate between the two two-pass multibucket
;   114          0114  1 !               index bucket split cases (both in one bucket vs. one in each).
```

```
;  115     0115  1 !                The end result was that no matter what two-pass multi-bucket
;  116     0116  1 !                split case had occurred, it was handled the same - as a both
;  117     0117  1 !                in the same bucket two-pass multi-bucket split case. In the
;  118     0118  1 !                case of the the other two-pass multi-bucket split case this
;  119     0119  1 !                would result in failure to insert a new key-VBN pair into the
;  120     0120  1 !                new (right) index bucket. The fix simply is to set
;  121     0121  1 !                IRB$W_POS_INS after the differentiation has been made as to the
;  122     0122  1 !                type of two-pass multi-bucket index bucket split that had
;  123     0123  1 !                occurred.
;  124     0124  1 !
;  125     0125  1 !        V03-004 TMK0003         Todd M. Katz            26-May-1982
;  126     0126  1 !                Performance Enhancement to the routine EXT_HIGH_KEY. This
;  127     0127  1 !                routine extracts the high key of the old (left) bucket for
;  128     0128  1 !                use in updating the index level above. This extraction must
;  129     0129  1 !                include the key's re-expansion if index key compression is
;  130     0130  1 !                enabled. The way this routine is currently written mandates
;  131     0131  1 !                two complete passes down the index bucket - one to find the
;  132     0132  1 !                high key and the other to uncompress it. By setting the field
;  133     0133  1 !                IRB$L_LST_NCMP, during the first scan, to the last
;  134     0134  1 !                noncompressed index key in the entire index bucket it is
;  135     0135  1 !                possible to cut down on the amount of the index bucket which
;  136     0136  1 !                is necessary to scan in order to re-expand the high key.
;  137     0137  1 !
;  138     0138  1 !        V03-003 TMK0002         Todd M. Katz            11-May-1982
;  139     0139  1 !                Add support for the two two-pass index bucket split cases
;  140     0140  1 !                which currently are not supported and return an alternate
;  141     0141  1 !                success status of index not updated. These two-pass index
;  142     0142  1 !                bucket split cases are: two-pass two-bucket with empty
;  143     0143  1 !                bucket split and two-pass multi-bucket with empty bucket
;  144     0144  1 !                split. In the case of the former split case, the routine
;  145     0145  1 !                RM$INS_REC will only be called once since there is only a
;  146     0146  1 !                need to swing the level-one index downpointer of the new
;  147     0147  1 !                index bucket's low-order index record from the leftmost data
;  148     0148  1 !                bucket to the rightmost new bucket. In the case of a two-pass
;  149     0149  1 !                multi-bucket with empty bucket split case, the routine
;  150     0150  1 !                RM$INS_REC is called twice. First, the new key in keybuffer 2
;  151     0151  1 !                is inserted into the old (left) index bucket along with the
;  152     0152  1 !                VBN that corresponds to it (in IRB$L_VBN_MID). The second time
;  153     0153  1 !                this routine is called the VBN downpointer of the new (right)
;  154     0154  1 !                index bucket's low-order index record is swung from the
;  155     0155  1 !                leftmost to the rightmost bucket.
;  156     0156  1 !
;  157     0157  1 !        V03-002 TMK0001         Todd M. Katz            21-Apr-1982
;  158     0158  1 !                The routine COMP_SPL_PNT is responsible for computing the
;  159     0159  1 !                index bucket split point. Under certain circumstances, it
;  160     0160  1 !                will sometimes have to find the record size of the index
;  161     0161  1 !                record(s) which are to be inserted. This record size comes in
;  162     0162  1 !                two parts - key and VBN, which must be added together. Because
;  163     0163  1 !                the macros used to define the VBN offsets were wrong, this
;  164     0164  1 !                mean't that when the total size of the new record(s) was
;  165     0165  1 !                computed, it was computed incorrectly. This could lead to the
;  166     0166  1 !                split point being incorrectly calculated, and to a RMS
;  167     0167  1 !                signalled bugcheck in at least one case. To fix this problem,
;  168     0168  1 !                I fixed the macro definitions, and but them in ONE place.
;  169     0169  1 !
;  170     0170  1 !        V03-001 MCN0001         Maria del C. Nasr       25-Mar-1982
;  171     0171  1 !                Use macro to get key buffer address.
```

```
;  172          0172  1 ;
;  173          0173  1 ;       V02-019 KPL0001         Peter Lieberwirth       2-Mar-1982
;  174          0174  1 ;               Define linkage omitted in v02-018
;  175          0175  1 ;
;  176          0176  1 ;       V02-018 TMK0004         Todd M. Katz    01-Mar-1982
;  177          0177  1 ;               Add support for rear-end truncation of keys in the index
;  178          0178  1 ;               of prologue 3 files with compressed keys. This involves changes
;  179          0179  1 ;               to RM$SPLIT_EM. When an index bucket split occurs, the new
;  180          0180  1 ;               (right) bucket is constructed without any thought as to where
;  181          0181  1 ;               the new index records will be inserted. If the prologue 3 file
;  182          0182  1 ;               contains compressed index keys, then the compression of the
;  183          0183  1 ;               first key in the new bucket must be redetermined. At the very
;  184          0184  1 ;               least, the key must be expanded so that it is zero front
;  185          0185  1 ;               compressed. As any change in front compression may affect
;  186          0186  1 ;               rear-end truncation, the entire compression of the first key
;  187          0187  1 ;               must be redetermined.
;  188          0188  1 ;
;  189          0189  1 ;       V02-017 TMK0003         Todd M. Katz    12-Feb-1982
;  190          0190  1 ;               It is possible for the two-pass two-bucket split case to be
;  191          0191  1 ;               confused with the two-pass multibucket split case where both
;  192          0192  1 ;               keys are to go in the new bucket. This will occur when each
;  193          0193  1 ;               index bucket can only hold two keys, an index bucket containing
;  194          0194  1 ;               two keys (the second of which is nonzero front compressed) must
;  195          0195  1 ;               split to make room for an additional key, and the insertion
;  196          0196  1 ;               point is computed to be at the beginning of the bucket. In this
;  197          0197  1 ;               case, both the insertion point and the split point are at the
;  198          0198  1 ;               beginning of the bucket. Since RMS doesn't always check for a
;  199          0199  1 ;               multibucket case when it wants to special case the special
;  200          0200  1 ;               multibucket split case described above because it wrongly
;  201          0201  1 ;               assumes that the insertion point can only be at the beginning
;  202          0202  1 ;               of a bucket in the speical two-pass multibucket split case,
;  203          0203  1 ;               the two-bucket case can be confused with the multibucket case.
;  204          0204  1 ;               Add checks for a multibucket split in all instances when it
;  205          0205  1 ;               isn't absolutly clear which of the two split cases is taking
;  206          0206  1 ;               place.
;  207          0207  1 ;
;  208          0208  1 ;       V02-016 TMK0002         Todd M. Katz    25-Jan-1982
;  209          0209  1 ;               Made multiple changes affecting bucket splitting of
;  210          0210  1 ;               index buckets. These changes affect all prologs, and both
;  211          0211  1 ;               compressed and noncompressed prologue 3 keys.
;  212          0212  1 ;
;  213          0213  1 ;                   1. When the new key is to be inserted as the very first record
;  214          0214  1 ;                      in the new (right) index bucket following an index bucket
;  215          0215  1 ;                      split, the offset to the point of insertion,
;  216          0216  1 ;                      IRAB[IRB$W_POS_INS], in the new bucket was not being
;  217          0217  1 ;                      correctly computed. This was because the amount of front
;  218          0218  1 ;                      compression of the new low order key is saved before it is
;  219          0219  1 ;                      uncompressed, and then added to the offset. This is correct
;  220          0220  1 ;                      except when the new key to be inserted will become the very
;  221          0221  1 ;                      first key in the new bucket, and thus preceed it. What RMS
;  222          0222  1 ;                      will now do is move the records into the new bucket
;  223          0223  1 ;                      regardless of where the new record will be inserted, extract
;  224          0224  1 ;                      the amount of front compression of the new low order key and
;  225          0225  1 ;                      uncompress it, zero the amount of front compression if the
;  226          0226  1 ;                      new key is to be inserted as the very first key in the new
;  227          0227  1 ;                      bucket, and then proceed to insert the new key.
;  228          0228  1 ;
```

```
;  229          0229  1 ;
;  230          0230  1 ;
;  231          0231  1 ;
;  232          0232  1 ;
;  233          0233  1 ;
;  234          0234  1 ;
;  235          0235  1 ;
;  236          0236  1 ;
;  237          0237  1 ;
;  238          0238  1 ;
;  239          0239  1 ;
;  240          0240  1 ;
;  241          0241  1 ;
;  242          0242  1 ;
;  243          0243  1 ;
;  244          0244  1 ;
;  245          0245  1 ;
;  246          0246  1 ;
;  247          0247  1 ;
;  248          0248  1 ;
;  249          0249  1 ;
;  250          0250  1 ;
;  251          0251  1 ;
;  252          0252  1 ;
;  253          0253  1 ;
;  254          0254  1 ;
;  255          0255  1 ;
;  256          0256  1 ;
;  257          0257  1 ;
;  258          0258  1 ;
;  259          0259  1 ;
;  260          0260  1 ;
;  261          0261  1 ;
;  262          0262  1 ;
;  263          0263  1 ;
;  264          0264  1 ;
;  265          0265  1 ;
;  266          0266  1 ;
;  267          0267  1 ;
;  268          0268  1 ;
;  269          0269  1 ;
;  270          0270  1 ;
;  271          0271  1 ;
;  272          0272  1 ;
;  273          0273  1 ;
;  274          0274  1 ;
;  275          0275  1 ;
;  276          0276  1 ;
;  277          0277  1 ;
;  278          0278  1 ;
;  279          0279  1 ;
;  280          0280  1 ;
;  281          0281  1 ;
;  282          0282  1 ;
;  283          0283  1 ;
;  284          0284  1 ;
;  285          0285  1 ;
```

2. The series of checks made before incrementing the pointer, IRAB[IRB$L_LST_NCOMP], to the last noncompressed key seen was not stringent enough. What must be checked is that the current position in the bucket, REC_ADDR, is less than the point of insertion, INS_PNT. What was being checked was whether the bucket scan had past the insertion point. Since REC_ADDR is incremented to the next key position and the pointer to the last noncompressed key seen adjusted before the check is made as to whether the current bucket position is also the point of insertion, it was possible for the last noncompressed key pointer to be incremented to the key which will follow the new key when the latter is eventually inserted. If the new key, and the key which follows it (and is pointed at by the last noncompressed key pointer) share some common front characters, the front compression of the new key will be incorrect, and the index will be corrupted.

3. There was no code handling the situation when a prologue 3 index bucket splits, and the point of insertion of the new key(s) is at the split point. In this instance, changes must be made to both the old, and the new index bucket. More specifically, if a multibucket split occurs, a new key is inserted in both the old (as the high key) and the new (as the low key) index buckets, and a VBN pointer in the new index bucket bucket must be swung from IRAB[IRB$L_VBN_LEFT] (its value before the split occurred) to IRAB[IRB$L_VBN_RIGHT]. If a nonmultibucket split occurs, a new key is inserted as the high order key in the old index bucket, and the same VBN pointer swing, described above, occurs. These changes are made by forcing prologue 3 files to go through the same top-level code sequence as prologue 1 and 2 files, affecting must of the changes in lower level routines.

4. In those cases when there is a key to go in the old index bucket, and the point of insertion is at the point at which the bucket was split, and empty buckets are involved, do not update the index, and instead return an alternate success status of index not updated. Previously, the multibucket split empty bucket case was handled exactly like this, and for the two-bucket split empty bucket case, a situation which should almost never happen, the existing code for nonprologue 3 files did not handle it correctly, and there was no code for prologue 3 files.

5. It is possible for both the point of insertion and the the split point to be at the very first record in an index bucket. This will occur when the key size is such that it is possible for two index records to be in the same bucket, and a multibucket split has occurred requiring updating of such a bucket. In this case, we want to put both new keys in the old bucket, but still swing the VBN pointer of what will become the low order key in the new index bucket. The key updating is accomplished by lower routines which will notice that not only is IRAB[IRB$V_BIG_SPLIT] and IRAB[IRB$V_SPL_IDX] set but that the point of insertion

```
;  286          0286  1 ;          is at the very first key position of the old index bucket.
;  287          0287  1 ;          That just the VBN swing should be performed for this
;  288          0288  1 ;          multibucket split case will be signalled by clearing
;  289          0289  1 ;          IRAB[IRB$L_VBN_MID] if the point of insertion is at the
;  290          0290  1 ;          very first key position. This is contrary to what normally
;  291          0291  1 ;          would occur when a multibucket split has occurred and the
;  292          0292  1 ;          point of insertion is also the split point.
;  293          0293  1 ;
;  294          0294  1 ;      6.  Made two changes in the computation of the split point
;  295          0295  1 ;          of a prologue 3 index bucket with noncompressed keys. First,
;  296          0296  1 ;          if the point of insertion of the new index record(s) is the
;  297          0297  1 ;          same as that of the split point, the new index record (at
;  298          0298  1 ;          least one of them) will be put in the old bucket. Second,
;  299          0299  1 ;          if the new index record is to go in the old bucket and
;  300          0300  1 ;          there is insufficient room, an additional record is moved
;  301          0301  1 ;          out to make room only if the point of insertion and the
;  302          0302  1 ;          split point are not the same. If they are the same, this
;  303          0303  1 ;          inability to fit both records must be do to a multibucket
;  304          0304  1 ;          split since all records are the same size and at least one
;  305          0305  1 ;          has been moved out. In such a case one new index record
;  306          0306  1 ;          will go in the old bucket, and the second in the new index
;  307          0307  1 ;          bucket, and so no action need be taken.
;  308          0308  1 ;
;  309          0309  1 ;      7.  Made a change in the computation of the split point
;  310          0310  1 ;          of a prologue 2 index bucket, SIDR bucket or prologue 3
;  311          0311  1 ;          index bucket with compressed keys. Only if the new record
;  312          0312  1 ;          is to go in the old bucket, and there appears to be
;  313          0313  1 ;          insufficient room for it, and the split point and
;  314          0314  1 ;          insertion points are not the same do we make more room by
;  315          0315  1 ;          moving out additional records. This can only occur for
;  316          0316  1 ;          SIDR buckets. For index buckets there will always be
;  317          0317  1 ;          sufficent room unless a multibucket split has occurred
;  318          0318  1 ;          requiring the insertion of two new keys. But in such
;  319          0319  1 ;          instances the bucket will be split at the insertion point
;  320          0320  1 ;          and one key will end up in the old and the other in the
;  321          0321  1 ;          new index bucket.
;  322          0322  1 ;
;  323          0323  1 ;  V02-015 TMK0001          Todd M. Katz    09-Jan-1982
;  324          0324  1 ;          Made two changes affecting bucket splitting of
;  325          0325  1 ;          index buckets with compressed keys.
;  326          0326  1 ;
;  327          0327  1 ;          1.  In some cases when the key to be inserted was to
;  328          0328  1 ;              go in the new (right) bucket, the pointer to the
;  329          0329  1 ;              last noncompressed key, IRB$L_LST_NCOMP, pointed to
;  330          0330  1 ;              a key preceeding the insertion point in what will
;  331          0331  1 ;              become the old (left) bucket. This resulted in the
;  332          0332  1 ;              incorrect determination of the amount of front
;  333          0333  1 ;              compression of the new key, and corruption of the file.
;  334          0334  1 ;              Update this pointer to the first key in the new bucket
;  335          0335  1 ;              if it does not point to a key in the new bucket
;  336          0336  1 ;              preceeding the insertion point.
;  337          0337  1 ;
;  338          0338  1 ;          2.  The routine RM$RECORD_SIZE returns the length of the new
;  339          0339  1 ;              index record to be inserted in two parts. The first word
;  340          0340  1 ;              is the key size, and the second word of the longword
;  341          0341  1 ;              is the VBN size. However, the routine COMP_SPL_PNT
;  342          0342  1 ;              thought that only a single value was returned. Thus, when
```

```
;  343          0343  1 !                the split point of a index bucket with compressed keys
;  344          0344  1 !                was being determined, and the routine encountered in its
;  345          0345  1 !                scan of the the bucket the point of insertion and had to
;  346          0346  1 !                take into account the size of the new record, the resulting
;  347          0347  1 !                incorrect use of the record size returned prevented the
;  348          0348  1 !                proper determination of the split point. In fact the
;  349          0349  1 !                incorrect use of the record size returned resulted in the
;  350          0350  1 !                inability to correctly split an index bucket whenever the
;  351          0351  1 !                new key was to go in the old (left) side.
;  352          0352  1 !
;  353          0353  1 !        V02-014 PSK0006         P S Knibbe      14-Dec-1981
;  354          0354  1 !                Change RECORD_SIZE to take BKT_ADDR as an input
;  355          0355  1 !
;  356          0356  1 !        V02-013 PSK0005         P S Knibbe      13-Nov-1981
;  357          0357  1 !                Change COMP_SPL_PNT to keep track of last record
;  358          0358  1 !                with no front compression and the SPL_COUNT for
;  359          0359  1 !                compressed keys
;  360          0360  1 !
;  361          0361  1 !        V02-012 PSK0004         P S Knibbe      26-Oct-1981
;  362          0362  1 !                Add support for splitting compressed buckets
;  363          0363  1 !
;  364          0364  1 !        V02-011 PSK0003         P S Knibbe      25-Oct-1981
;  365          0365  1 !                Set the AP flags correctly when calling RM$RECORD_KEY
;  366          0366  1 !
;  367          0367  1 !        V02-010 PSK0002         P S Knibbe      07-Oct-1981
;  368          0368  1 !                Make sure the ptr_sz from one bucket is correctly
;  369          0369  1 !                moved into the new bucket when a split occurs.
;  370          0370  1 !
;  371          0371  1 !        V02-009 PSK0001         P S Knibbe      07-Aug-1981
;  372          0372  1 !                Add support for prologue three indexes.
;  373          0373  1 !
;  374          0374  1 !        V02-008 CDS0071         C D Saether     26-Feb-1981     20:00
;  375          0375  1 !                Return error on spl_idx, non-empty, big_split case.
;  376          0376  1 !
;  377          0377  1 !        V02-007 REFORMAT        R A SCHAEFER    23-Jul-1980     14:15
;  378          0378  1 !                Reformat the source.
;  379          0379  1 !
;  380          0380  1 !        V02-006 CDS0070         C D SAETHER     14-JAN-1980     14:35
;  381          0381  1 !                Fix splitting alternate data level when no dupes count
;  382          0382  1 !                field present in record.  Use rm$rec_ovhd routine.
;  383          0383  1 !
;  384          0384  1 ! REVISION HISTORY:
;  385          0385  1 !
;  386          0386  1 !   Wendy Koenig,       12-OCT-78  14:46
;  387          0387  1 !   X0002 - CHANGE THE NRP STUFF
;  388          0388  1 !
;  389          0389  1 !   Wendy Koenig,       24-OCT-78  14:03
;  390          0390  1 !   X0003 - MAKE CHANGES CAUSED BY SHARING CONVENTIONS
;  391          0391  1 !
;  392          0392  1 !   Wendy Koenig,       3-NOV-78  11:35
;  393          0393  1 !   X0004 - GET RID OF ROUTINE RM$ID_AND_NRP
;  394          0394  1 !
;  395          0395  1 !   Christian Saether,  26-JAN-79  9:45
;  396          0396  1 !   X0005 - don't split insert of index entries on empty bucket cases
;  397          0397  1 !
;  398          0398  1 !*****
```

```
;  400     0399  1 !
;  401     0400  1 !
;  402     0401  1 !                                Index Bucket Update Cases
;  403     0402  1 !
;  404     0403  1 !    There are four and only four index bucket update cases: two-bucket,
;  405     0404  1 ! two-bucket with empty bucket, multi-bucket, and multi-bucket with empty
;  406     0405  1 ! bucket. I will discuss these cases from the point of view of updating the
;  407     0406  1 ! index following a primary data bucket split.
;  408     0407  1 !
;  409     0408  1 !
;  410     0409  1 ! Two-bucket Index Bucket Update Case.
;  411     0410  1 !
;  412     0411  1 !    When a primary data bucket splits roughly in half, the high key of the
;  413     0412  1 ! bucket now becomes the high key of the new (right) bucket, and the old (left)
;  414     0413  1 ! bucket has a new key which is not yet represented in the index. If there is
;  415     0414  1 ! sufficient room in the level 1 index bucket for a new index record pointing
;  416     0415  1 ! to the old (left) bucket then a two-bucket index bucket update is performed.
;  417     0416  1 ! This index bucket update case has four steps to it.
;  418     0417  1 !
;  419     0418  1 !        1. Position to the index record pointing to the bucket before the split.
;  420     0419  1 !           This index record's key will be identical to the high key of the
;  421     0420  1 !           new (right) bucket, but its VBN downpointer will be to the old (left)
;  422     0421  1 !           bucket.
;  423     0422  1 !
;  424     0423  1 !        2. Make sufficient room for the new index record. This step is very
;  425     0424  1 !           prologue dependent.
;  426     0425  1 !
;  427     0426  1 !        3. Insert the new index record. The key of this new index record will
;  428     0427  1 !           be the new high key of the old (left) data bucket while the VBN
;  429     0428  1 !           downpointer will also be to old bucket.
;  430     0429  1 !
;  431     0430  1 !        4. Position to the record following this new index record. Of course,
;  432     0431  1 !           the key of this index record was the high key of the data bucket
;  433     0432  1 !           before the split, and it is now the high key of the new (right)
;  434     0433  1 !           bucket. However, its VBN downpointer will still be pointing to the
;  435     0434  1 !           old bucket that has just split and not to the new (right) bucket.
;  436     0435  1 !           Change this index record's VBN downpointer from pointing to the old
;  437     0436  1 !           (left) bucket, to pointing to the new (right) bucket.
;  438     0437  1 !
;  439     0438  1 !
;  440     0439  1 ! Two-bucket with Empty Bucket Index Bucket Update Case.
;  441     0440  1 !
;  442     0441  1 !    If after performing a data bucket split one new bucket has been added,
;  443     0442  1 ! and the left bucket is found to be empty, then index maintenance consists of
;  444     0443  1 ! performing a two-bucket with empty bucket index bucket update. There are a
;  445     0444  1 ! number of primary data bucket split cases that can result in this particular
;  446     0445  1 ! index maintenance case, but they all will be handled the same as far as the
;  447     0446  1 ! index update is concerned. Note that for this particular index bucket update
;  448     0447  1 ! case the new bucket is regarded as the "middle" bucket, and there is no
;  449     0448  1 ! "right" bucket. Three steps are involved in the index update.
;  450     0449  1 !
;  451     0450  1 !        1. Position to the index record pointing to the bucket before the split.
;  452     0451  1 !           This index record's key will be identical to the high key of the
;  453     0452  1 !           new (right) bucket.
;  454     0453  1 !
;  455     0454  1 !        2. If the VBN downpointer of this index record does NOT point to the
;  456     0455  1 !           data bucket that has split (the old or left data bucket) then do NOT
```

```
;  457     0456  1 !          data the index update. If this index update was performed, then RMS
;  458     0457  1 !          would be causing crossed downpointers to the primary data level below
;  459     0458  1 !          and would no longer be able to retrieve certain records randomly by
;  460     0459  1 !          their primary key value.
;  461     0460  1 !
;  462     0461  1 !       3. If the VBN downpointer of this index record does point to the data
;  463     0462  1 !          bucket that has split, then update its VBN downpointer so that it now
;  464     0463  1 !          points to the new (middle) bucket (whose high key value is the same
;  465     0464  1 !          as the key of the index record, and was the same as the high key of
;  466     0465  1 !          the old data bucket before it split, and the old (left) bucket became
;  467     0466  1 !          empty), instead of pointing to the old (left) bucket which is now
;  468     0467  1 !          empty.
;  469     0468  1 !
;  470     0469  1 ! Note that this in effect removes the old (left) bucket from the index tree
;  471     0470  1 ! (there is no index record with a VBN downpointer pointing to it), but this is
;  472     0471  1 ! acceptable because the bucket is empty. The empty bucket remains linked into
;  473     0472  1 ! the horizontal data bucket chain.
;  474     0473  1 !
;  475     0474  1 !
;  476     0475  1 ! Multi-bucket Index Bucket Update Case.
;  477     0476  1 !
;  478     0477  1 !       If a multi-bucket split has occurred at the primary data level then the
;  479     0478  1 ! old (left) bucket and the new middle bucket contain new high keys, and the
;  480     0479  1 ! new right bucket contains as its high key the high key of the splitting data
;  481     0480  1 ! bucket before the split took place. In such a situation RMS will attempt a
;  482     0481  1 ! multi-bucket index bucket update provided there is sufficient room in the
;  483     0482  1 ! level one index bucket to contain the two new index records (one for each of
;  484     0483  1 ! the new buckets) which must be added. This index bucket update case consists
;  485     0484  1 ! of five steps.
;  486     0485  1 !
;  487     0486  1 !       1. Position to the index record pointing to the bucket before the split.
;  488     0487  1 !          This index record's key will be identical to the high key of the
;  489     0488  1 !          new right bucket, but its VBN downpointer will be to the old (left)
;  490     0489  1 !          bucket.
;  491     0490  1 !
;  492     0491  1 !       2. Make sufficient room in the index bucket for the two new index
;  493     0492  1 !          record. This step is very prologue dependent.
;  494     0493  1 !
;  495     0494  1 !       3. Insert the first of the two new index records. The key of this new
;  496     0495  1 !          index record will be the new high key of the old (left) data bucket
;  497     0496  1 !          while the VBN downpointer will also be to old bucket.
;  498     0497  1 !
;  499     0498  1 !       4. Insert the second of the two new index records after the first. The
;  500     0499  1 !          key of this new index record will be the new high key of the new
;  501     0500  1 !          middle data bucket while the VBN downpointer will also be to the new
;  502     0501  1 !          middle bucket.
;  503     0502  1 !
;  504     0503  1 !       5. Position to the record following the new index records. Of course,
;  505     0504  1 !          the key of this index record was the high key of the data bucket
;  506     0505  1 !          before the split, and it is now the high key of the new right bucket.
;  507     0506  1 !          However, its VBN downpointer will still be pointing to the old bucket
;  508     0507  1 !          that has just split and not to the new right bucket. Change this
;  509     0508  1 !          index record's VBN downpointer from pointing to the old (left) bucket
;  510     0509  1 !          to pointing to the new right bucket.
;  511     0510  1 !
;  512     0511  1 !
;  513     0512  1 ! Multi-bucket with Empty Bucket Index Bucket Update Case.
```

```
;  514     0513  1 !
;  515     0514  1 !       If a multi-bucket split has occurred at the primary data level, and
;  516     0515  1 ! following the split the bucket splitting (the old or left bucket) is empty
;  517     0516  1 ! then RMS will attempt a multi-bucket with empty bucket index bucket update
;  518     0517  1 ! provided there is sufficient room in the level one index bucket for the new
;  519     0518  1 ! index record that must be added. There is only a need to add one new index
;  520     0519  1 ! record in this particular index bucket update case, because although there
;  521     0520  1 ! will be be two new primary data buckets (the new middle bucket containing a
;  522     0521  1 ! new high key and the new right bucket containing the high key of the old left
;  523     0522  1 ! bucket before the split took place) there is only one new high key. This
;  524     0523  1 ! index bucket update consists of five steps.
;  525     0524  1 !
;  526     0525  1 !       1. Position to the index record pointing to the bucket before the split.
;  527     0526  1 !          This index record's key will be identical to the high key of the
;  528     0527  1 !          new right bucket, but its VBN downpointer will be to the old (left)
;  529     0528  1 !          bucket.
;  530     0529  1 !
;  531     0530  1 !       2. Make sufficient room for the new index record. This step is very
;  532     0531  1 !          prologue dependent.
;  533     0532  1 !
;  534     0533  1 !       3. Insert the new index record. The key of this new index record will
;  535     0534  1 !          be the new high key of the new middle data bucket while the VBN
;  536     0535  1 !          downpointer will also be to this bucket.
;  537     0536  1 !
;  538     0537  1 !       4. Position to the record following this new index record. Of course,
;  539     0538  1 !          the key of this index record was the high key of the data bucket
;  540     0539  1 !          before the split, and it is now the high key of the new right bucket.
;  541     0540  1 !          If the VBN downpointer of this index record does point to the bucket
;  542     0541  1 !          that has just split (the old left bucket), then update the VBN
;  543     0542  1 !          downpointer so that it points to the new right bucket. Note that this
;  544     0543  1 !          in effect removes the old (left) bucket from the index tree (there is
;  545     0544  1 !          no index record with a VBN downpointer pointing to it), but this is
;  546     0545  1 !          acceptable because the bucket is empty. The empty bucket remains
;  547     0546  1 !          linked into the horizontal data bucket chain.
;  548     0547  1 !
;  549     0548  1 !       5. If the VBN downpointer of the index record does not point to the
;  550     0549  1 !          bucket splitting (the old left) bucket then do NOT update the VBN
;  551     0550  1 !          downpointer so that it points to the new right data bucket. If this
;  552     0551  1 !          index update was performed, then RMS would be causing crossed
;  553     0552  1 !          downpointers to the primary data level below, and would no longer be
;  554     0553  1 !          able to retrieve certain records randomly by their primary key value.
;  555     0554  1 !
;  556     0555  1 !
;  557     0556  1 !
;  558     0557  1 !                                Index Bucket Split Cases
;  559     0558  1 !
;  560     0559  1 !       There are four one-pass and five two-pass index bucket split cases. The
;  561     0560  1 ! four one-pass index bucket split cases are: two-bucket, two-bucket with empty
;  562     0561  1 ! bucket, multi-bucket, and multi-bucket with empty bucket. The five two-pass
;  563     0562  1 ! index bucket split cases are: two-bucket, two-bucket with empty bucket,
;  564     0563  1 ! multi-bucket case I, multi-bucket case II, and multibucket with empty bucket.
;  565     0564  1 ! I will discuss these cases from the point of view of updating the index
;  566     0565  1 ! following a primary data bucket split.
;  567     0566  1 !
;  568     0567  1 !
;  569     0568  1 ! One-pass Two-bucket Index Bucket Split Case.
;  570     0569  1 !
```

```
;  571          0570  1 !       Index maintenance will consist of a one-pass two-bucket index bucket
;  572          0571  1 ! split when RMS is unable to perform a two-bucket index bucket update because
;  573          0572  1 ! there is insufficient room in the index bucket for the new index record, and
;  574          0573  1 ! either the split point of the index bucket and the position of insertion of
;  575          0574  1 ! the new index record do not coincide, or if they do, the new index record is
;  576          0575  1 ! to go into the new (right) bucket. This particular split case consists of
;  577          0576  1 ! three steps.
;  578          0577  1 !
;  579          0578  1 !       1. Split the index bucket. This step is very prologue dependent.
;  580          0579  1 !
;  581          0580  1 !       2. Perform a two-bucket index bucket update on the index bucket (old or
;  582          0581  1 !          new) which after the index bucket split is to contain the new index
;  583          0582  1 !          record which is to be added.
;  584          0583  1 !
;  585          0584  1 !       3. As there has been a split of the level one index bucket, the level
;  586          0585  1 !          two index bucket containing the index record with the VBN downpointer
;  587          0586  1 !          pointing to this level one index bucket must be updated. If the index
;  588          0587  1 !          bucket splitting was the root bucket, a new root bucket is created,
;  589          0588  1 !          and then the index update is performed.
;  590          0589  1 !
;  591          0590  1 !
;  592          0591  1 ! One-pass Two-bucket with Empty Bucket Index Bucket Split Case.
;  593          0592  1 !
;  594          0593  1 !       Index maintenance will consist of a one-pass two-bucket with empty
;  595          0594  1 ! bucket index bucket split when RMS is unable to perform a two-bucket with
;  596          0595  1 ! empty bucket index bucket update because the pointer size required to hold
;  597          0596  1 ! the VBN of the new (right) bucket is greater than that required to hold the
;  598          0597  1 ! VBN of the old (left) empty bucket, there is insufficient room in the index
;  599          0598  1 ! bucket to accommidate the new size, and either the split point of the index
;  600          0599  1 ! bucket and the position of the index record whose VBN downpointer is to be
;  601          0600  1 ! updated do not coincide, or if they do, the index record will exist in the
;  602          0601  1 ! new (right) bucket following the index bucket split. This is extremely
;  603          0602  1 ! prologue dependent. It can be expected that this split case will occur much
;  604          0603  1 ! more frequently in the case of prologue 3 files, where the VBN downpointer
;  605          0604  1 ! size is fixed on an index bucket basis, then in the case of nonprologue 3
;  606          0605  1 ! files where the VBN downpointer size is fixed on an index record basis. Three
;  607          0606  1 ! steps constitute this split case.
;  608          0607  1 !
;  609          0608  1 !       1. Split the index bucket. This step is very prologue dependent.
;  610          0609  1 !
;  611          0610  1 !       2. Perform a two-bucket with empty bucket index bucket update on the
;  612          0611  1 !          index bucket (old or new) which after the index bucket split contains
;  613          0612  1 !          the index record whose VBN downpointer needs to be updated.
;  614          0613  1 !
;  615          0614  1 !       3. As there has been a split of the level one index bucket, the level
;  616          0615  1 !          two index bucket containing the index record with the VBN downpointer
;  617          0616  1 !          pointing to this level one index bucket must be updated. If the index
;  618          0617  1 !          bucket splitting was the root bucket, a new root bucket is created,
;  619          0618  1 !          and then the index update is performed.
;  620          0619  1 !
;  621          0620  1 !
;  622          0621  1 ! One-pass Multi-bucket Index Bucket Split Case.
;  623          0622  1 !
;  624          0623  1 !       Index maintenance will consist of a one-pass multi-bucket index bucket
;  625          0624  1 ! split when RMS is unable to perform a multi-bucket index bucket update
;  626          0625  1 ! because there is insufficient room in the index bucket for the two new index
;  627          0626  1 ! records, and either the split point of the index bucket and the position of
```

```
;  628     0627  1 ! insertion of the two new index records do not coincide, or if they do, the
;  629     0628  1 ! new index records are to go into the new (right) bucket. This particular
;  630     0629  1 ! split case consists of three steps.
;  631     0630  1 !
;  632     0631  1 !       1. Split the index bucket. This step is very prologue dependent.
;  633     0632  1 !
;  634     0633  1 !       2. Perform a multi-bucket index bucket update on the index bucket (old
;  635     0634  1 !          or new) which after the index bucket split is to contain the two new
;  636     0635  1 !          index records which are to be added.
;  637     0636  1 !
;  638     0637  1 !       3. As there has been a split of the level one index bucket, the level
;  639     0638  1 !          two index bucket containing the index record with the VBN downpointer
;  640     0639  1 !          pointing to this level one index bucket must be updated. If the index
;  641     0640  1 !          bucket splitting was the root bucket, a new root bucket is created,
;  642     0641  1 !          and then the index update is performed.
;  643     0642  1 !
;  644     0643  1 !
;  645     0644  1 ! One-pass Multi-bucket with Empty Bucket Index Bucket Split Case.
;  646     0645  1 !
;  647     0646  1 !       Index maintenance will consist of a one-pass multi-bucket with empty
;  648     0647  1 ! bucket index bucket split when RMS is unable to perform a multi-bucket with
;  649     0648  1 ! empty bucket index bucket update because there is insufficient room in the
;  650     0649  1 ! index bucket for the new index record, and either the split point of the
;  651     0650  1 ! index bucket and the position of insertion of the new index record do not
;  652     0651  1 ! coincide, or if they do, the new index record is to go into the new (right)
;  653     0652  1 ! bucket. This particular split case consists of three steps.
;  654     0653  1 !
;  655     0654  1 !       1. Split the index bucket. This step is very prologue dependent.
;  656     0655  1 !
;  657     0656  1 !       2. Perform a multi-bucket with empty bucket index bucket update on the
;  658     0657  1 !          index bucket (old or new) which after the index bucket split is to
;  659     0658  1 !          contain the new index record which is to be added.
;  660     0659  1 !
;  661     0660  1 !       3. As there has been a split of the level one index bucket, the level
;  662     0661  1 !          two index bucket containing the index record with the VBN downpointer
;  663     0662  1 !          pointing to this level one index bucket must be updated. If the index
;  664     0663  1 !          bucket splitting was the root bucket, a new root bucket is created,
;  665     0664  1 !          and then the index update is performed.
;  666     0665  1 !
;  667     0666  1 !
;  668     0667  1 ! Two-pass Two-bucket Index Bucket Split Case.
;  669     0668  1 !
;  670     0669  1 !       Index maintenance will consist of a two-pass two-bucket index bucket
;  671     0670  1 ! split when RMS is unable to perform a two-bucket index bucket update because
;  672     0671  1 ! there is insufficient room in the index bucket for the new index record, the
;  673     0672  1 ! split point of the index bucket and the position of insertion of the new
;  674     0673  1 ! index record coincide, and the new index record is to go into the old (left)
;  675     0674  1 ! index bucket. This particular split case consists of four steps.
;  676     0675  1 !
;  677     0676  1 !       1. Split the index bucket. This step is very prologue dependent.
;  678     0677  1 !
;  679     0678  1 !       2. Insert the new index record as the rightmost index record of the old
;  680     0679  1 !          (left) index bucket. The key of this index record will be the new
;  681     0680  1 !          high key of the old (left) primary data bucket, and the VBN
;  682     0681  1 !          downpointer of this new index record will also point to this very
;  683     0682  1 !          same bucket.
;  684     0683  1 !
```

```
;  685     0684  1 !       3. Update the VBN downpointer of the first index record of the new
;  686     0685  1 !          (right) index bucket so that it points to the new (right) primary
;  687     0686  1 !          data bucket instead of to the old (left) data bucket. The key of
;  688     0687  1 !          this index record is identical to the key of the primary data bucket
;  689     0688  1 !          which split (before the split took place), and of course, to the high
;  690     0689  1 !          key of the new (right) primary data bucket.
;  691     0690  1 !
;  692     0691  1 !       4. As there has been a split of the level one index bucket, the level
;  693     0692  1 !          two index bucket containing the index record with the VBN downpointer
;  694     0693  1 !          pointing to this level one index bucket must be updated. If the index
;  695     0694  1 !          bucket splitting was the root bucket, a new root bucket is created,
;  696     0695  1 !          and then the index update is performed.
;  697     0696  1 !
;  698     0697  1 !
;  699     0698  1 ! Two-pass Two-bucket with Empty Bucket Index Bucket Split Case.
;  700     0699  1 !
;  701     0700  1 !       Index maintenance will consist of a two-pass two-bucket with empty
;  702     0701  1 ! bucket index bucket split when RMS is unable to perform a two-bucket with
;  703     0702  1 ! empty bucket index bucket update because the pointer size required to hold
;  704     0703  1 ! the VBN of the new (right) bucket is greater than that required to hold the
;  705     0704  1 ! VBN of the old (left) empty bucket, the split point of the index bucket and
;  706     0705  1 ! the position of the index record whose VBN downpointer is to be updated
;  707     0706  1 ! coincide, and the index record whose VBN downpointer is to be updated is
;  708     0707  1 ! "marked" as to exist as the rightmost index record in the old (left) index
;  709     0708  1 ! bucket following the index bucket split. This is extremely prologue dependent
;  710     0709  1 ! It can be expected that this split case will occur much more frequently in
;  711     0710  1 ! the case of prologue 3 files, where the VBN downpointer size is fixed on an
;  712     0711  1 ! index bucket basis, then in the case of nonprologue 3 files where the VBN
;  713     0712  1 ! downpointer size is fixed on an index record basis. Four steps constitute
;  714     0713  1 ! this split case.
;  715     0714  1 !
;  716     0715  1 !       1. Split the index bucket. This step is very prologue dependent.
;  717     0716  1 !
;  718     0717  1 !       2. Position to the first index record position in the new index bucket.
;  719     0718  1 !          This index record will contain the high key of the splitting primary
;  720     0719  1 !          data bucket before it split, and thus, will be identical to the high
;  721     0720  1 !          key of the new (middle) primary data bucket. If the VBN downpointer
;  722     0721  1 !          of this index record does NOT point to the data bucket that has split
;  723     0722  1 !          (the old or left data bucket) then do NOT change its VBN downpointer.
;  724     0723  1 !          If this VBN downpointer was changed, then RMS would be causing
;  725     0724  1 !          crossed downpointers to the primary data level below, and would no
;  726     0725  1 !          longer be able to retrieve certain records randomly by their primary
;  727     0726  1 !          key value.
;  728     0727  1 !
;  729     0728  1 !       3. If the VBN downpointer of this index record does point to the data
;  730     0729  1 !          bucket that has split, then update its VBN downpointer so that it now
;  731     0730  1 !          points to the new (middle) bucket (whose high key value is the same
;  732     0731  1 !          as the key of the index record, and was the same as the high key of
;  733     0732  1 !          the old data bucket before it split, and the old (left) bucket became
;  734     0733  1 !          empty), instead of pointing to the old (left) bucket which is now
;  735     0734  1 !          empty.
;  736     0735  1 !
;  737     0736  1 !       4. As there has been a split of the level one index bucket, the level
;  738     0737  1 !          two index bucket containing the index record with the VBN downpointer
;  739     0738  1 !          pointing to this level one index bucket must be updated. If the index
;  740     0739  1 !          bucket splitting was the root bucket, a new root bucket is created,
;  741     0740  1 !          and then the index update is performed.
```

```
;  742     0741  1 !
;  743     0742  1 !
;  744     0743  1 ! Two-pass Multi-bucket Index Bucket Split Case I.
;  745     0744  1 !
;  746     0745  1 !       Index maintenance will consist of a two-pass multi-bucket index bucket
;  747     0746  1 ! split (case I) when RMS is unable to perform a multi-bucket index bucket
;  748     0747  1 ! update because there is insufficient room in the index bucket for the two new
;  749     0748  1 ! index records, the split point of the index bucket and the position of
;  750     0749  1 ! insertion of the two new index records coincide, the new index records are
;  751     0750  1 ! "marked" as to go into the old (left) index bucket following the index
;  752     0751  1 ! bucket split, and the point of insertion of the new index records and the
;  753     0752  1 ! split point do NOT coincide with the position of the very first index record
;  754     0753  1 ! in the old (left) index bucket. This particular split case consists of five
;  755     0754  1 ! steps.
;  756     0755  1 !
;  757     0756  1 !       1. Split the index bucket. This step is very prologue dependent.
;  758     0757  1 !
;  759     0758  1 !       2. Insert the first of the two new index records as the rightmost index
;  760     0759  1 !          record of the old (left) index bucket. The key of this index record
;  761     0760  1 !          will be the high key of the old (left) primary data bucket, and the
;  762     0761  1 !          VBN downpointer of this new index record will also point to this very
;  763     0762  1 !          same data bucket.
;  764     0763  1 !
;  765     0764  1 !       3. Position to the first index record position in the new (right) index
;  766     0765  1 !          bucket, make sufficient room for the second of the two new index
;  767     0766  1 !          records (this is very prologue dependent), and insert the new record.
;  768     0767  1 !          The key will be the new high key of the new middle primary data
;  769     0768  1 !          bucket, and the new index record's VBN downpointer will also point to
;  770     0769  1 !          this data bucket.
;  771     0770  1 !
;  772     0771  1 !       4. Update the VBN downpointer of the second index record of the new
;  773     0772  1 !          (right) index bucket so that it points to the new right primary data
;  774     0773  1 !          bucket instead of to the old (left) data bucket. The key of this
;  775     0774  1 !          index record is identical to the key of the primary data bucket which
;  776     0775  1 !          split before the split occurred, and of course, to the high key of
;  777     0776  1 !          the new right primary data bucket.
;  778     0777  1 !
;  779     0778  1 !       5. As there has been a split of the level one index bucket, the level
;  780     0779  1 !          two index bucket containing the index record with the VBN downpointer
;  781     0780  1 !          pointing to this level one index bucket must be updated. If the index
;  782     0781  1 !          bucket splitting was the root bucket, a new root bucket is created,
;  783     0782  1 !          and then the index update is performed.
;  784     0783  1 !
;  785     0784  1 !
;  786     0785  1 ! Two-pass Multi-bucket Index Bucket Split Case II.
;  787     0786  1 !
;  788     0787  1 !       Index maintenance will consist of a two-pass multi-bucket index bucket
;  789     0788  1 ! split (case II) when RMS is unable to perform a multi-bucket index bucket
;  790     0789  1 ! update because there is insufficient room in the index bucket for the two new
;  791     0790  1 ! index records, the split point of the index bucket and the position of
;  792     0791  1 ! insertion of the two new index records coincide, the new index records are
;  793     0792  1 ! "marked" as to go into the new (left) index bucket following the index
;  794     0793  1 ! bucket split, and the point of insertion of the new index records and the
;  795     0794  1 ! split point coincide with the position of the very first index record in the
;  796     0795  1 ! old (left) index bucket. This particular split case is a very special split
;  797     0796  1 ! case that can be required during the processing of only those files whose
;  798     0797  1 ! bucket size/key size ratio allows for only two (or at the least two if index
```

```
;  799     0798  1 ! compression is enabled) index records per index bucket, and when a
;  800     0799  1 ! multi-bucket split occurs such the the keys of both new index records are
;  801     0800  1 ! less than the key of the leftmost index record of the index bucket in which
;  802     0801  1 ! both are to go. Five steps are required to process this particular split
;  803     0802  1 ! case.
;  804     0803  1 !
;  805     0804  1 !       1. Split the index bucket. This step is very prologue dependent. Because
;  806     0805  1 !          the split point will be the same as the first index record position
;  807     0806  1 !          in the splitting index bucket and only two index records will be
;  808     0807  1 !          allowed in each index bucket this means that following the split, the
;  809     0808  1 !          new (right) index bucket will contain both the records that were
;  810     0809  1 !          formerly in the splitting bucket, and the old (left) index bucket
;  811     0810  1 !          will temporarily be empty until the index update has been completed.
;  812     0811  1 !
;  813     0812  1 !       2. Insert the first of the two new index records as the very first index
;  814     0813  1 !          record in the old (left) index bucket. The key of this new index
;  815     0814  1 !          record will be the new high key of the old (left) data bucket while
;  816     0815  1 !          the VBN downpointer will also be to old bucket.
;  817     0816  1 !
;  818     0817  1 !       3. Insert the second of the two new index records after the first. The
;  819     0818  1 !          key of this new index record will be the new high key of the new
;  820     0819  1 !          middle data bucket while the VBN downpointer will also be to the new
;  821     0820  1 !          middle bucket.
;  822     0821  1 !
;  823     0822  1 !       4. Update the VBN downpointer of the first index record of the new
;  824     0823  1 !          (right) index bucket so that it points to the new right primary data
;  825     0824  1 !          bucket instead of to the old (left) data bucket. The key of this
;  826     0825  1 !          index record is identical to the key of the primary data bucket which
;  827     0826  1 !          split (before the split took place), and of course, to the high key
;  828     0827  1 !          of the new right primary data bucket.
;  829     0828  1 !
;  830     0829  1 !       5. As there has been a split of the level one index bucket, the level
;  831     0830  1 !          two index bucket containing the index record with the VBN downpointer
;  832     0831  1 !          pointing to this level one index bucket must be updated. If the index
;  833     0832  1 !          bucket splitting was the root bucket, a new root bucket is created,
;  834     0833  1 !          and then the index update is performed.
;  835     0834  1 !
;  836     0835  1 !
;  837     0836  1 ! Two-pass Multi-bucket with Empty Bucket Index Bucket Split Case.
;  838     0837  1 !
;  839     0838  1 !       Index maintenance will consist of a two-pass multi-bucket with empty
;  840     0839  1 ! bucket index bucket split when RMS is unable to perform a multi-bucket with
;  841     0840  1 ! empty bucket index bucket update because there is insufficient room in the
;  842     0841  1 ! index bucket for the new index record, the split point of the index bucket
;  843     0842  1 ! and the position of insertion of the new index record coincide, and the new
;  844     0843  1 ! index record is marked as to go into the old (left) index bucket following
;  845     0844  1 ! the index bucket split. This particular split case consists of five steps.
;  846     0845  1 !
;  847     0846  1 !       1. Split the index bucket. This step is very prologue dependent.
;  848     0847  1 !
;  849     0848  1 !       2. Insert the new index record as the rightmost index record of the old
;  850     0849  1 !          (left) index bucket. The key of this index record will be the high
;  851     0850  1 !          key of the new middle primary data bucket, and the VBN downpointer of
;  852     0851  1 !          this new index record will also point to this very same bucket.
;  853     0852  1 !
;  854     0853  1 !       3. Position to the first index record position in the new index bucket.
;  855     0854  1 !          This index record will contain the high key of the splitting primary
```

```
;  856     0855  1 !          data bucket before it split, and thus, will be identical to the high
;  857     0856  1 !          key of the new right primary data bucket. If the VBN downpointer of
;  858     0857  1 !          this index record does NOT point to the data bucket that has split
;  859     0858  1 !          (the old or left data bucket) then do NOT change its VBN downpointer.
;  860     0859  1 !          If this VBN downpointer was changed, then RMS would be causing
;  861     0860  1 !          crossed downpointers to the primary data level below, and would no
;  862     0861  1 !          longer be able to retrieve certain records randomly by their primary
;  863     0862  1 !          key value.
;  864     0863  1 !
;  865     0864  1 !      4. If the VBN downpointer of this index record does point to the data
;  866     0865  1 !          bucket that has split, then update its VBN downpointer so that it now
;  867     0866  1 !          points to the new right bucket (whose high key value is the same as
;  868     0867  1 !          the key of the index record, and was the same as the high key of the
;  869     0868  1 !          old data bucket before it split, and the old (left) bucket became
;  870     0869  1 !          empty), instead of pointing to the old (left) bucket which is now
;  871     0870  1 !          empty.
;  872     0871  1 !
;  873     0872  1 !      5. As there has been a split of the level one index bucket, the level
;  874     0873  1 !          two index bucket containing the index record with the VBN downpointer
;  875     0874  1 !          pointing to this level one index bucket must be updated. If the index
;  876     0875  1 !          bucket splitting was the root bucket, a new root bucket is created,
;  877     0876  1 !          and then the index update is performed.
;  878     0877  1 !
;  879     0878  1 !
;  880     0879  1 !
;  881     0880  1 !              Index Maintenance Requirements for Index Bucket Splitting
;  882     0881  1 !
;  883     0882  1 !      Whenever an index bucket splits it is necessary to update the index level
;  884     0883  1 ! immediately above, and if it is the root bucket that is splitting, a new root
;  885     0884  1 ! bucket is created before this updating takes place. If there is sufficient
;  886     0885  1 ! room in the index bucket that is to be updated because of the index bucket
;  887     0886  1 ! split, then an index update takes place; however, if there is also
;  888     0887  1 ! insufficient room in that bucket then it too must split, and the index level
;  889     0888  1 ! above it must be updated as well (or created if there is no level above).
;  890     0889  1 !
;  891     0890  1 !      As there can be no empty index buckets, and no index bucket splits
;  892     0891  1 ! involving more than one new index bucket this eliminates all empty bucket and
;  893     0892  1 ! multi-bucket update and split cases from occurring as the result of an index
;  894     0893  1 ! bucket split. In fact, only one update case (Two-bucket Index Bucket Update)
;  895     0894  1 ! and two split cases (One-pass Two-bucket Index Bucket Split and Two-pass
;  896     0895  1 ! Two-bucket Index Bucket Split) can ever result from the splitting of an index
;  897     0896  1 ! bucket. These three cases are handled exactly the same as if the maintenance
;  898     0897  1 ! of the index was required because of the splitting of a primary data bucket.
;  899     0898  1 !
;  900     0899  1 !
;  901     0900  1 !              Index Maintenance Requirements for SIDR Bucket Splitting
;  902     0901  1 !
;  903     0902  1 !      A split involving a SIDR bucket can also never create an empty bucket or
;  904     0903  1 ! involve more than one new bucket; therefore, only the Two-bucket Index Bucket
;  905     0904  1 ! Update, the One-pass Two-bucket Index Bucket Split, and the Two-pass
;  906     0905  1 ! Two-bucket Index Bucket Split cases can occur as the result of the splitting
;  907     0906  1 ! of a SIDR bucket. As was the case with the splitting of primary data and
;  908     0907  1 ! index buckets, an index update is first attempted, and if there is
;  909     0908  1 ! insufficient room in the index bucket to be update a split is done.
;  910     0909  1 ! Furthermore, as was the case with the index updated resulting from the
;  911     0910  1 ! splitting of index buckets, the various split and update cases resulting from
;  912     0911  1 ! the splitting of SIDR buckets are handled exactly the same as if the
```

```
;  913          0912  1 ! maintenance was required because of the splitting of a primary data buckets.
;  914          0913  1 ! Where the maintenance requirements of SIDR bucket splits differs from that
;  915          0914  1 ! of index buckets is when the bucket splitting is a continuation bucket.
;  916          0915  1 ! Actually, as SIDR continuation buckets never split (new one are just created
;  917          0916  1 ! as they are needed), whenever a SIDR continuation bucket is created no index
;  918          0917  1 ! updating is required.
;  919          0918  1 !
;  920          0919  1 !
;  921          0920  1 !                       The Goals of SIDR Bucket Splitting.
;  922          0921  1 !
;  923          0922  1 !     RMS will always try and split index buckets roughly in half. This is also
;  924          0923  1 ! the same basic goal for the computation of the split point in SIDR buckets.
;  925          0924  1 ! Most of the time RMS would like to split SIDR buckets roughly in half. It is
;  926          0925  1 ! not always successful because of odd numbers of SIDRs, and the differing
;  927          0926  1 ! sizes of those SIDRs. The differing SIDR size maybe caused by SIDR key
;  928          0927  1 ! compression, or by differing numbers of duplicates associated with each SIDR
;  929          0928  1 ! that can cause some SIDRs to be very large, even take up an entire SIDR
;  930          0929  1 ! bucket, or to be quite small.
;  931          0930  1 !
;  932          0931  1 !    To compute the SIDR bucket split point, RMS computes a target address
;  933          0932  1 ! within the bucket, and proceeds to scan the bucket. The target address is
;  934          0933  1 ! computed by looking at the amount of SIDR bucket in use, dividing it in half,
;  935          0934  1 ! and adding it to the current address of the bucket in memory. The bucket scan
;  936          0935  1 ! terminates when a SIDR is encountered whose address equals or exceeds the
;  937          0936  1 ! target split address. That address becomes the split point of the bucket.
;  938          0937  1 ! While performing this scan, RMS takes into acount the point of insertion of
;  939          0938  1 ! the new SIDR. If it encounters this address during the scan without exceeding
;  940          0939  1 ! the target address, RMS assumes that the new record is to go into the old or
;  941          0940  1 ! left bucket. A state bit is set to indicate this, and the size of the new
;  942          0941  1 ! record is taken into account from this point on when determining whether or
;  943          0942  1 ! not the target address has been exceeded. The most interesting SIDR bucket
;  944          0943  1 ! split cases arise when the address of the calculated SIDR bucket split point,
;  945          0944  1 ! and the point of insertion of the new record end up being one and the same.
;  946          0945  1 !
;  947          0946  1 !    There is one exception to how the bucket split point is calculated, and it
;  948          0947  1 ! points out a second SIDR bucket split goal. If the point of insertion of the
;  949          0948  1 ! new record is at the end of the SIDR bucket, in certain cases RMS will split
;  950          0949  1 ! the SIDR bucket at the end of the bucket (the point of insertion of the new
;  951          0950  1 ! record), and the new record is put in the new (right) SIDR bucket as the sole
;  952          0951  1 ! record. The reason for this is as follows. If the new record is a duplicate,
;  953          0952  1 ! and the SIDR it would belong to occupies the entire bucket, then that bucket
;  954          0953  1 ! is already being maximally used, and RMS will gain nothing by splitting the
;  955          0954  1 ! bucket any other way except the waste of space in the old (left) bucket
;  956          0955  1 ! (since SIDR array elements are aways added in FIFO ordering). RMS may benefit
;  957          0956  1 ! more by splitting the bucket at its end, and creating a SIDR continuation
;  958          0957  1 ! bucket with the new record as the first array element of the sole SIDR in the
;  959          0958  1 ! new bucket. On the other hand, if the record is a brand new SIDR (not a
;  960          0959  1 ! duplicate), then placing the new record in a bucket all by itself will be a
;  961          0960  1 ! real space win if the records are being added to the file in ascending order
;  962          0961  1 ! by this secondary key of reference. This is because each SIDR bucket will
;  963          0962  1 ! have been completely packed, before packing of the next one is initiated. If
;  964          0963  1 ! records are being added randomly then nothing is really lost by splitting the
;  965          0964  1 ! bucket here.
;  966          0965  1 !
;  967          0966  1 !    The third goal is more a matter of common sense. RMS can not split the SIDR
;  968          0967  1 ! bucket such that when RMS goes to insert the new record it finds that there is
;  969          0968  1 ! insufficient room in the SIDR bucket in which the new record is to go. Because
```

```
;  970          0969  1 ! RMS will never split a SIDR bucket in the middle of a SIDR, and because SIDRs
;  971          0970  1 ! can be very large so as to occupy almost the entire bucket it is possible for
;  972          0971  1 ! this situation to arise. For example, consider the case of a bucket consisting
;  973          0972  1 ! of one very small SIDR followed by one very large SIDR, and the new record
;  974          0973  1 ! was to go between the two. During the bucket scan RMS positions to the very
;  975          0974  1 ! large record, and even taking into account the new record, RMS still will not
;  976          0975  1 ! have positioned past the target split point address. Thus, RMS would conclude
;  977          0976  1 ! that the SIDR it is currently positioned to (the very large one) should go
;  978          0977  1 ! with the others in the old bucket, and would position past it. Since this
;  979          0978  1 ! positions RMS past the target split point address, RMS would stop scanning.
;  980          0979  1 ! However, this also would position RMS to the end of the bucket, and this is
;  981          0980  1 ! not a suitable split point since it would leave the old SIDR bucket intact
;  982          0981  1 ! and RMS already knows that there was insufficient room in the bucket to
;  983          0982  1 ! accomidate the new record or it would not be attempting a split in the first
;  984          0983  1 ! place. What this third goal forces RMS to do in this situation is to back up
;  985          0984  1 ! one SIDR and split the SIDR bucket between the very small and the very large
;  986          0985  1 ! records with the new SIDR to be placed into the old (left) bucket. Even
;  987          0986  1 ! though the SIDR bucket will be split into disapportionate halves, splitting
;  988          0987  1 ! at this place guarentees that the new record will have sufficient room for
;  989          0988  1 ! its insertion.
;  990          0989  1 !
;  991          0990  1 !    The forth and final goal of SIDR bucket splitting is to make sure that if
;  992          0991  1 ! the new record is a duplicate (ie - there are other records in the file with
;  993          0992  1 ! this secondary key value), it should always go into the SIDR bucket containing
;  994          0993  1 ! the SIDR it is a duplicate of. In fact, it should always end up as the last
;  995          0994  1 ! array element in this SIDR after its addition is completed. This goal never
;  996          0995  1 ! affects where the SIDR bucket split point is calculated to be; but rather,
;  997          0996  1 ! once this split point has been calculated and it is seen to coincide with the
;  998          0997  1 ! point of insertion of the new record, in which bucket the new record is
;  999          0998  1 ! placed. Note that there is one exception to this rule which has already been
; 1000          0999  1 ! described. If the SIDR the new record is a duplicate of occupies the entire
; 1001          1000  1 ! SIDR bucket, then the new record is placed by itself in a SIDR continuation
; 1002          1001  1 ! bucket.
; 1003          1002  1 !
; 1004          1003  1 !    Thus, there are four goals which are taken into account during SIDR bucket
; 1005          1004  1 ! split point determination. Goal number 2 takes precedence in this
; 1006          1005  1 ! determination followed by goal number 1. Goal number 3 serves to modulate
; 1007          1006  1 ! goal number 1 such that a split point will always be calculated which will
; 1008          1007  1 ! allow sufficient room for the new record in whatever bucket it is to go.
; 1009          1008  1 ! Finally, goal number 4 can affect in which bucket the new record is placed
; 1010          1009  1 ! provided it is not superceeded by goal number 2.
; 1011          1010  1 !
; 1012          1011  1 !
; 1013          1012  1 !     The SIDR Bucket Split Point ~= The Point of Insertion of the New Record.
; 1014          1013  1 !
; 1015          1014  1 !    There are six bucket split cases which fall into this catagory. They are
; 1016          1015  1 ! all handled basically the same. After the split point has been determined, the
; 1017          1016  1 ! SIDR bucket split takes place and then the new record is inserted into the
; 1018          1017  1 ! appropriate bucket at the appropriate place.
; 1019          1018  1 !
; 1020          1019  1 !    Case Number 1: The point of insertion of the new SIDR is at the front of
; 1021          1020  1 !                   the SIDR bucket.
; 1022          1021  1 !
; 1023          1022  1 !    Case Number 2: The point of insertion of the new SIDR is to the left of the
; 1024          1023  1 !                   SIDR bucket split point but not at the front of the bucket.
; 1025          1024  1 !
; 1026          1025  1 !    Case Number 3: The point of insertion of the new duplicate is to the left
```

```
;  1027     1026  1 !                    of the SIDR bucket split point but not at the front of the
;  1028     1027  1 !                    SIDR bucket.
;  1029     1028  1 !
;  1030     1029  1 !    Case Number 4: The point of insertion of the new SIDR is to the right of
;  1031     1030  1 !                    the SIDR bucket split point but not at the end of the
;  1032     1031  1 !                    bucket.
;  1033     1032  1 !
;  1034     1033  1 !    Case Number 5: The point of insertion of the new duplicate is to the right
;  1035     1034  1 !                    of the SIDR bucket split point but not at the end of the
;  1036     1035  1 !                    SIDR bucket.
;  1037     1036  1 !
;  1038     1037  1 !    Case Number 6: The point of insertion of the new duplicate is at the end
;  1039     1038  1 !                    of the bucket, but the SIDR it belongs does not occupy the
;  1040     1039  1 !                    entire SIDR bucket.
;  1041     1040  1 !
;  1042     1041  1 ! Actually, case number 6 is handled somewhat differently from the other 5. The
;  1043     1042  1 ! split point is initially determined to be at the end of the bucket (Goal 2).
;  1044     1043  1 ! However, when it is determined that the SIDR the new duplicate belongs to does
;  1045     1044  1 ! not occupy the entire bucket, RMS decides to split the SIDR bucket not at the
;  1046     1045  1 ! end, but at the beginning address of the last SIDR in the bucket. Thus, the
;  1047     1046  1 ! SIDR bucket split is performed such that the new (right) SIDR bucket contains
;  1048     1047  1 ! one SIDR - the SIDR that was formerly the last record in the old bucket, and
;  1049     1048  1 ! the new duplicate is inserted as the last member of its array.
;  1050     1049  1 !
;  1051     1050  1 !
;  1052     1051  1 !     The SIDR Bucket Split Point = The Point of Insertion of the New Record.
;  1053     1052  1 !
;  1054     1053  1 !    There are seven bucket split cases which fall into this catagory. They are
;  1055     1054  1 ! all handled basically the same. After the split point has been determined, the
;  1056     1055  1 ! SIDR bucket split takes place and then the new record is inserted into the
;  1057     1056  1 ! appropriate bucket at the appropriate place.
;  1058     1057  1 !
;  1059     1058  1 !    Case Number 1: The point of insertion of the new SIDR is at the front of
;  1060     1059  1 !                    the SIDR bucket. In this case the SIDR currently at the
;  1061     1060  1 !                    front of the SIDR bucket must occupy the entire bucket.
;  1062     1061  1 !                    It will be moved into the new (right) bucket and the new
;  1063     1062  1 !                    SIDR will occupy the old (left) bucket by itself.
;  1064     1063  1 !
;  1065     1064  1 !    Case Number 2: The point of insertion of the new SIDR is to the left of
;  1066     1065  1 !                    or at the hypothetical 50-50 split point of the SIDR bucket
;  1067     1066  1 !                    but not at the front of the bucket. The new SIDR will be
;  1068     1067  1 !                    inserted as the last record in the old (left) SIDR bucket.
;  1069     1068  1 !
;  1070     1069  1 !    Case Number 3: The point of insertion of the new duplicate is to the left
;  1071     1070  1 !                    of or at the hypothetical 50-50 split point of the SIDR
;  1072     1071  1 !                    bucket but not at the front of the bucket. The new duplicate
;  1073     1072  1 !                    will be inserted as the last array element in what will
;  1074     1073  1 !                    become the last SIDR in the old (left) SIDR bucket.
;  1075     1074  1 !
;  1076     1075  1 !    Case Number 4: The point of insertion of the new SIDR is to the right
;  1077     1076  1 !                    of the hypothetical 50-50 split point of the SIDR bucket
;  1078     1077  1 !                    but not at the end of the bucket. The new SIDR will become
;  1079     1078  1 !                    the first record in the new (right) SIDR bucket.
;  1080     1079  1 !
;  1081     1080  1 !    Case Number 5: The point of insertion of the new duplicate is to the right
;  1082     1081  1 !                    of the hypothetical 50-50 split point of the SIDR bucket
;  1083     1082  1 !                    but not at the end of the bucket. The new duplicate will be
```

```
;  1084          1083  1 !                       inserted as the last array element in what will become the
;  1085          1084  1 !                       last SIDR in the old (left) SIDR bucket.
;  1086          1085  1 !
;  1087          1086  1 !     Case Number 6:    The point of insertion of the new SIDR is at the end of the
;  1088          1087  1 !                       SIDR bucket. The new SIDR will become the first and only
;  1089          1088  1 !                       record in the new (right) SIDR bucket.
;  1090          1089  1 !
;  1091          1090  1 !     Case Number 7:    The point of insertion of the new duplicate is at the end of
;  1092          1091  1 !                       the SIDR bucket. Because the SIDR occupies the entire old
;  1093          1092  1 !                       (left) SIDR bucket, the new (right) bucket is created as a
;  1094          1093  1 !                       SIDR continuation bucket. A new SIDR is created to exist
;  1095          1094  1 !                       within this new bucket. It is the sole SIDR within the new
;  1096          1095  1 !                       bucket, and the new duplicate is inserted as the sole array
;  1097          1096  1 !                       element within this SIDR.
;  1098          1097  1 !
;  1099          1098  1 !*****
;  1100          1099  1
;  1101          1100  1 LIBRARY 'RMSLIB:RMS';
;  1102          1101  1
;  1103          1102  1 REQUIRE 'RMSSRC:RMSIDXDEF';
;  1104          1167  1
;  1105          1168  1 ! Define default PSECTS for code.
;  1106          1169  1 !
;  1107          1170  1 PSECT
;  1108          1171  1     CODE = RM$RMS3(PSECT_ATTR),
;  1109          1172  1     PLIT = RM$RMS3(PSECT_ATTR);
;  1110          1173  1
;  1111          1174  1 ! Linkages
;  1112          1175  1 !
;  1113          1176  1 LINKAGE
;  1114          1177  1     L_JSB01,
;  1115          1178  1     L_PRESERVE1,
;  1116          1179  1     L_RABREG_567,
;  1117          1180  1     L_RABREG_67,
;  1118          1181  1     L_REC_OVHD,
;  1119          1182  1
;  1120          1183  1     ! Local Linkages
;  1121          1184  1     !
;  1122          1185  1     RL$COMP_SPL_PNT = JSB (REGISTER = 1)
;  1123          1186  1                       : GLOBAL (R_IDX_DFN, R_IRAB, R_REC_ADDR, R_BKT_ADDR,R_IFAB),
;  1124          1187  1     RL$LINKAGE = JSB ()
;  1125          1188  1                     : GLOBAL (R_IDX_DFN, R_IFAB, R_IRAB, R_REC_ADDR);
;  1126          1189  1
;  1127          1190  1
;  1128          1191  1 ! External Routines
;  1129          1192  1 !
;  1130          1193  1 EXTERNAL ROUTINE
;  1131          1194  1     RM$CNTRL_ADDR       : RL$RABREG_567,
;  1132          1195  1     RM$COMPRESS_KEY     : RL$JSB01,
;  1133          1196  1     RM$GETNEXT_REC      : RL$RABREG_67,
;  1134          1197  1     RM$INS_REC          : RL$RABREG_67,
;  1135          1198  1     RM$MOVE             : RL$PRESERVE1,
;  1136          1199  1     RM$RECORD_KEY       : RL$PRESERVE1,
;  1137          1200  1     RM$RECORD_SIZE      : RL$RABREG_567,
;  1138          1201  1     RM$REC_OVHD         : RL$REC_OVHD;
;  1139          1202  1
;  1140          1203  1 ! Key compression overhead macros.
```

```
;  1141          1204  1 !
;  1142          1205  1 MACRO
;  1143          1206  1     COMPR       = 0,8,8,0 %,
;  1144          1207  1     KEY_LEN     = 0,0,8,0 %;
```

```
;  1146          1208  1 %SBTTL 'RM$COMP_SPL_PNT'
;  1147          1209  1 ROUTINE RM$COMP_SPL_PNT (INS_PNT) : RL$COMP_SPL_PNT =
;  1148          1210  1
;  1149          1211  1 !+++
;  1150          1212  1 !
;  1151          1213  1 !  FUNCTIONAL DESCRIPTION:
;  1152          1214  1 !
;  1153          1215  1 !       This routine returns the address at which the bucket should
;  1154          1216  1 !       be split.
;  1155          1217  1 !
;  1156          1218  1 !  CALLING SEQUENCE:
;  1157          1219  1 !
;  1158          1220  1 !       RM$COMP_SPL_PNT()
;  1159          1221  1 !
;  1160          1222  1 !  INPUT PARAMETERS:
;  1161          1223  1 !
;  1162          1224  1 !       Address to insert the new record
;  1163          1225  1 !
;  1164          1226  1 !  IMPLICIT INPUT:
;  1165          1227  1 !
;  1166          1228  1 !       IDX_DFN
;  1167          1229  1 !       IRAB
;  1168          1230  1 !       BKT_ADDR
;  1169          1231  1 !
;  1170          1232  1 !  OUTPUT PARAMTERS:
;  1171          1233  1 !       NONE
;  1172          1234  1 !
;  1173          1235  1 !  IMPLICIT OUTPUT:
;  1174          1236  1 !
;  1175          1237  1 !       IRAB [IRB$L_SPL_CNT] - Number of first VBN which should be in new bucket
;  1176          1238  1 !       IRAB [IRB$V_REC_W_LO] - Set if new record should go in left bucket
;  1177          1239  1 !       IRAB [IRB$L_LST_NCMP] - Address of last record with a front compressoin
;  1178          1240  1 !                               count of zero (before insertion point)
;  1179          1241  1 !
;  1180          1242  1 !  ROUTINE VALUE:
;  1181          1243  1 !
;  1182          1244  1 !       SPL_PNT - Address of first record which should be in new bucket
;  1183          1245  1 !
;  1184          1246  1 !  SIDE EFFECTS:
;  1185          1247  1 !       NONE
;  1186          1248  1 !
;  1187          1249  1 !---
;  1188          1250  1
;  1189          1251  2     BEGIN
;  1190          1252  2
;  1191          1253  2     EXTERNAL REGISTER
;  1192          1254  2         R_BKT_ADDR_STR,
;  1193          1255  2         R_IDX_DFN_STR,
;  1194          1256  2         R_IFAB,
;  1195          1257  2         R_IRAB_STR,
;  1196          1258  2         R_REC_ADDR_STR;
;  1197          1259  2
;  1198          1260  2     LOCAL
;  1199          1261  2         EOB,
;  1200          1262  2         SPL_PNT;
;  1201          1263  2
;  1202          1264  2     MACRO
```

```
;  1203     1265  2          KEY_SZ = 0,0,16,0 %,
;  1204     1266  2          VBN_SZ = 0,16,16,0 %;
;  1205     1267  2
;  1206     1268  2      REC_ADDR = .BKT_ADDR + BKT$C_OVERHDSZ;
;  1207     1269  2      EOB = .BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE];
;  1208     1270  2
;  1209     1271  2      ! Prologue 3 non-compressed index record.
;  1210     1272  2      !
;  1211     1273  2      IF  .BKT_ADDR[BKT$B_LEVEL] GTRU 0
;  1212     1274  2           AND
;  1213     1275  2          .IDX_DFN [IDX$B_IDXBKTYP] EQLU IDX$C_NCMPIDX
;  1214     1276  2      THEN
;  1215     1277  2
;  1216     1278  3          BEGIN
;  1217     1279  3
;  1218     1280  3          GLOBAL REGISTER
;  1219     1281  3              R_RAB,
;  1220     1282  3              R_IMPURE;
;  1221     1283  3
;  1222     1284  3          LOCAL
;  1223     1285  3              VBN_ADDR,
;  1224     1286  3              TEMP: BLOCK[1];
;  1225     1287  3
;  1226     1288  3          TEMP = RM$RECORD_SIZE();
;  1227     1289  3
;  1228     1290  3          ! Compute the address of first record past the midpoint.
;  1229     1291  3          !
;  1230     1292  3          SPL_PNT = .REC_ADDR +
;  1231     1293  4                    (((.EOB - .REC_ADDR) / 2) / .IDX_DFN [IDX$B_KEYSZ])
;  1232     1294  3                                              * .IDX_DFN [IDX$B_KEYSZ];
;  1233     1295  3
;  1234     1296  3          ! Get the address of VBN split point.
;  1235     1297  3          !
;  1236     1298  4          BEGIN
;  1237     1299  4
;  1238     1300  4          LOCAL
;  1239     1301  4              SAVE;
;  1240     1302  4
;  1241     1303  4          IRAB [IRB$L_SPL_COUNT] = (.SPL_PNT - .REC_ADDR) / .IDX_DFN [IDX$B_KEYSZ];
;  1242     1304  4          SAVE = .IRAB [IRB$L_REC_COUNT];
;  1243     1305  4          IRAB [IRB$L_REC_COUNT] = .IRAB [IRB$L_SPL_COUNT];
;  1244     1306  4          VBN_ADDR = RM$CNTRL_ADDR();
;  1245     1307  4          IRAB [IRB$L_REC_COUNT] = .SAVE;
;  1246     1308  3          END;
;  1247     1309  3
;  1248     1310  3          ! The new record will go into the left bucket.
;  1249     1311  3          !
;  1250     1312  3          IF .SPL_PNT GEQA .INS_PNT
;  1251     1313  3          THEN
;  1252     1314  4              BEGIN
;  1253     1315  4
;  1254     1316  4              IRAB [IRB$V_REC_W_LO] = 1;
;  1255     1317  4
;  1256     1318  4              ! If there is insufficient room in the old bucket for the index
;  1257     1319  4              ! records which must be inserted, and we are not at the insertion
;  1258     1320  4              ! point then move out one more record to make room. If we are at
;  1259     1321  4              ! the insertion point then there is insufficient room because
```

```
;  1260     1322  4              ! we are attempting to insert two keys into an index bucket where
;  1261     1323  4              ! there is only room for one. In such a case one key will go in
;  1262     1324  4              ! the old bucket, and another will go in the new index bucket.
;  1263     1325  4              !
;  1264     1326  4              ! REMEMBER: keys are fixed in size!!!
;  1265     1327  4              !
;  1266     1328  5              IF (.VBN_ADDR - .SPL_PNT) LSSU (.TEMP [KEY_SZ] + .TEMP [VBN_SZ])
;  1267     1329  4                 AND
;  1268     1330  5                 (.INS_PNT NEQA .SPL_PNT)
;  1269     1331  4              THEN
;  1270     1332  4
;  1271     1333  4                  ! The records we are moving out don't leave enough room
;  1272     1334  4                  ! Move one more
;  1273     1335  4                  !
;  1274     1336  5                  BEGIN
;  1275     1337  5                  SPL_PNT = .SPL_PNT - .IDX_DFN [IDX$B_KEYSZ];
;  1276     1338  5                  IRAB [IRB$L_SPL_COUNT] = .IRAB [IRB$L_SPL_COUNT] - 1;
;  1277     1339  4                  END;
;  1278     1340  4
;  1279     1341  3              END;
;  1280     1342  3
;  1281     1343  3          END
;  1282     1344  2      ELSE
;  1283     1345  2
;  1284     1346  2          ! Prologue two index or SIDR bucket.
;  1285     1347  2          ! Prologue three compressed index or SIDR bucket.
;  1286     1348  2          !
;  1287     1349  3          BEGIN
;  1288     1350  3
;  1289     1351  3          LOCAL
;  1290     1352  3              SAVE_COUNT;
;  1291     1353  3
;  1292     1354  3          SAVE_COUNT = .IRAB [IRB$L_REC_COUNT];
;  1293     1355  3          IRAB [IRB$L_LST_NCMP] = .REC_ADDR;
;  1294     1356  3
;  1295     1357  3          IF .INS_PNT NEQ .EOB
;  1296     1358  3          THEN
;  1297     1359  4              BEGIN                                   ! of block scan
;  1298     1360  4
;  1299     1361  4              LOCAL
;  1300     1362  4                  BUCKET_LEVEL,
;  1301     1363  4                  LRA,
;  1302     1364  4                  NEW_REC_SZ;
;  1303     1365  4
;  1304     1366  4              ! Determine the level of the bucket once and only once.
;  1305     1367  4              !
;  1306     1368  4              IF (BUCKET_LEVEL = .BKT_ADDR[BKT$B_LEVEL]) EQLU 0
;  1307     1369  4              THEN
;  1308     1370  4                  BUCKET_LEVEL = -1;
;  1309     1371  4
;  1310     1372  4
;  1311     1373  4              ! Compute the hypothetical 50-50 split point based on the
;  1312     1374  4              ! current bucket freespace offset pointer.
;  1313     1375  4              !
;  1314     1376  4              NEW_REC_SZ = 0;
;  1315     1377  4              SPL_PNT = (.EOB - .REC_ADDR)/2 + .REC_ADDR;
;  1316     1378  4              IRAB [IRB$L_REC_COUNT] = 0;
```

```
;  1317     1379  4
;  1318     1380  4              ! Scan bucket until RMS gets to the record past the split point
;  1319     1381  4              ! calculated (if the new record is going to end up in the new
;  1320     1382  4              ! (right) bucket, or until the current position plus the size of
;  1321     1383  4              ! the new record to be inserted exceeds the split point (if the
;  1322     1384  4              ! new record is going to end up in the old (left) bucket.
;  1323     1385  4              !
;  1324     1386  4              DO
;  1325     1387  5                  BEGIN
;  1326     1388  5
;  1327     1389  5                  LRA = .REC_ADDR;
;  1328     1390  5
;  1329     1391  5                  ! if RMS's current position in the bucket scan matches the
;  1330     1392  5                  ! point of insertion, then the new record will go into
;  1331     1393  5                  ! the old (left) bucket. at this time the size of the new
;  1332     1394  5                  ! record is computed so its value maybe taken into account
;  1333     1395  5                  ! in the determination of the split point. if the insertion
;  1334     1396  5                  ! of the new record would not cross the split point, the
;  1335     1397  5                  ! bucket scan continues with the next record.
;  1336     1398  5                  !
;  1337     1399  5                  IF .REC_ADDR EQLA .INS_PNT
;  1338     1400  5                  THEN
;  1339     1401  6                      BEGIN
;  1340     1402  6
;  1341     1403  6                      GLOBAL REGISTER
;  1342     1404  6                          R_RAB,
;  1343     1405  6                          R_IMPURE;
;  1344     1406  6
;  1345     1407  6                      IF .IRAB[IRB$V_REC_W_LO]
;  1346     1408  6                      THEN
;  1347     1409  6                          RM$GETNEXT_REC()
;  1348     1410  6                      ELSE
;  1349     1411  7                          BEGIN
;  1350     1412  7
;  1351     1413  7                          LOCAL
;  1352     1414  7                              TEMP: BLOCK[1];
;  1353     1415  7
;  1354     1416  7                          ! Obtain size of the new index record(s). This is a
;  1355     1417  7                          ! single quantity in a non-prologue 3 file, but two
;  1356     1418  7                          ! contigious words in a prologue 3 file. However, since
;  1357     1419  7                          ! the entire prologue 1 index record but only the key
;  1358     1420  7                          ! portion of the prologue 3 index record contributes
;  1359     1421  7                          ! to the end-of-bucket computation, just use the low
;  1360     1422  7                          ! order word (key size in the case of a prologue 3
;  1361     1423  7                          ! index record but entire index record size if
;  1362     1424  7                          ! nonprologue 3) as the new record size.
;  1363     1425  7                          !
;  1364     1426  7                          TEMP = RM$RECORD_SIZE();
;  1365     1427  7                          NEW_REC_SZ = .TEMP[KEY_SZ];
;  1366     1428  7                          IRAB[IRB$V_REC_W_LO] = 1;
;  1367     1429  7                          END
;  1368     1430  6                      END
;  1369     1431  5                  ELSE
;  1370     1432  6                      BEGIN
;  1371     1433  6
;  1372     1434  6                      LOCAL
;  1373     1435  6                          REC_SIZE,
```

```
;  1374     1436  6                    RECORD_OVHD;
;  1375     1437  6
;  1376     1438  6                ! Position to the next record in the bucket.
;  1377     1439  6                !
;  1378     1440  6                RECORD_OVHD = RM$REC_OVHD(.BUCKET_LEVEL; REC_SIZE);
;  1379     1441  6                REC_ADDR = .REC_ADDR + .RECORD_OVHD + .REC_SIZE;
;  1380     1442  6
;  1381     1443  6                IRAB[IRB$L_REC_COUNT] = .IRAB[IRB$L_REC_COUNT] + 1;
;  1382     1444  6
;  1383     1445  6                ! if index or key compression is enabled, this is an
;  1384     1446  6                ! appropriate bucket, the front compression of the current
;  1385     1447  6                ! record is zero, and RMS has not reached the insertion
;  1386     1448  6                ! point, then update the pointer to the last noncompressed
;  1387     1449  6                ! record to facilitate computation of the front compression
;  1388     1450  6                ! of the new record to be inserted.
;  1389     1451  6                !
;  1390     1452  8                IF ((.BKT_ADDR[BKT$B_LEVEL] EQLU 0
;  1391     1453  8                                    AND
;  1392     1454  8                                    .IDX_DFN[IDX$V_KEY_COMPR])
;  1393     1455  7                                OR
;  1394     1456  8                                (.BKT_ADDR[BKT$B_LEVEL] GTRU 0
;  1395     1457  8                                    AND
;  1396     1458  7                                    .IDX_DFN [IDX$V_IDX_COMPR]))
;  1397     1459  6                     AND
;  1398     1460  6                    .(.REC_ADDR + .RECORD_OVHD)<8,8> EQLU 0
;  1399     1461  6                     AND
;  1400     1462  6                    .REC_ADDR LSSA .INS_PNT
;  1401     1463  6                THEN
;  1402     1464  6                    IRAB [IRB$L_LST_NCMP] = .REC_ADDR;
;  1403     1465  6                END
;  1404     1466  6
;  1405     1467  5                END
;  1406     1468  4            UNTIL .REC_ADDR + .NEW_REC_SZ GTRA .SPL_PNT;
;  1407     1469  4
;  1408     1470  4                ! if the new record goes in the old bucket, make sure we are at
;  1409     1471  4                ! least moving out enough records to make room for it. SIDRs and
;  1410     1472  4                ! index records must be separately considered.
;  1411     1473  4                !
;  1412     1474  4                ! In the case of SIDRs, it is guaranteed that the new record will
;  1413     1475  4                ! be as small as possible, and if even one record is moved out of
;  1414     1476  4                ! the old bucket there will be room for it. However, if a very
;  1415     1477  4                ! small record is to be added, the possibility exists that by the
;  1416     1478  4                ! time the bucket scanning loop is exited, there will be
;  1417     1479  4                ! insufficient room for the new record. This is because while the
;  1418     1480  4                ! small record size at the insertion point did not cause the split
;  1419     1481  4                ! point to be exceeded, the very large record which follows did
;  1420     1482  4                ! and if the split was made following the large record there would
;  1421     1483  4                ! not be enough room in the old bucket for the new record. in this
;  1422     1484  4                ! instance we still want to put the new record in the old (left)
;  1423     1485  4                ! bucket, but we want to split at the point of inserting forcing
;  1424     1486  4                ! the following big record to go in the new (right) bucket.
;  1425     1487  4                !
;  1426     1488  4                ! in the case of index records, if we are not at the insertion
;  1427     1489  4                ! point then there must be room for the new records in the old
;  1428     1490  4                ! bucket, otherwise, we would have exited the above loop while
;  1429     1491  4                ! at the insertion point. if we are at the insertion point and
;  1430     1492  4                ! there isn't room to insert new index records this can only be
```

```
4                   ! because there has been a multibucket split requiring two new
4                   ! keys to be inserted. in such a case, one will go in the old
4                   ! bucket, and the other in the new index bucket, so nothing
4                   ! further need be done.
4                   !
4                   IF .IRAB[IRB$V_REC_W_LO]
4                       AND
4                       (.EOB - .REC_ADDR) LSSU .NEW_REC_SZ
4                       AND
5                       (.REC_ADDR NEQA .INS_PNT)
4                   THEN
5                       BEGIN
5                       REC_ADDR = .LRA;
5                       IRAB [IRB$L_REC_COUNT] = .IRAB [IRB$L_REC_COUNT] - 1;
4                       END;
4
4                   ! At this point, REC_ADDR is pointing to the place where RMS is
4                   ! going to split the bucket.
4                   !
4                   SPL_PNT = .REC_ADDR;
4                   END
4
4               ! If the point of insertion is at the end of the free-space of the
4               ! bucket, then this is also where RMS will split the bucket.
4               !
3               ELSE
3                   SPL_PNT = .INS_PNT;
3
3               IRAB [IRB$L_SPL_COUNT] = .IRAB [IRB$L_REC_COUNT];
3               IRAB [IRB$L_REC_COUNT] = .SAVE_COUNT;
3
2               END;
2
2           RETURN .SPL_PNT;
1           END;


                                                .TITLE  RM3SIDXSP
                                                .IDENT  \V04-000\

                                                .EXTRN  RM$CNTRL_ADDR, RM$COMPRESS_KEY
                                                .EXTRN  RM$GETNEXT_REC, RM$INS_REC
                                                .EXTRN  RM$MOVE, RM$RECORD_KEY
                                                .EXTRN  RM$RECORD_SIZE, RM$REC_OVHD

                                                .PSECT  RM$RMS3,NOWRT,  GBL,  PIC,2

                  091C   8F  BB 00000 RM$COMP_SPL_PNT:
                                                PUSHR   #^M<R2,R3,R4,R8,R11>                    ; 1209
          5E             0C  C2 00004          SUBL2   #12, SP                                 ;
                         51  DD 00007          PUSHL   R1                                      ;
          56       0E    A5  9E 00009          MOVAB   14(R5), REC_ADDR                        ; 1268
          53       04    A5  3C 0000D          MOVZWL  4(BKT_ADDR), EOB                        ; 1269
          53             55  C0 00011          ADDL2   BKT_ADDR, EOB                           ;
                   0C    A5  95 00014          TSTB    12(BKT_ADDR)                            ; 1273
                         78  13 00017          BEQL    2$                                      ;
          02       28    A7  91 00019          CMPB    40(IDX_DFN), #2                         ; 1275
```

```
                                                72  12 0001D            BNEQ    2$                                      ; 1288
                                              0000G 30 0001F            BSBW    RM$RECORD_SIZE                          ; 1293
                                         0C   AE   50  D0 00022            MOVL    R0, TEMP                                ; 1294
                                         50        53  C3 00026            SUBL3   REC_ADDR, EOB, R0                       ; 1303
                                                   50  C6 0002A            DIVL2   #2, R0                                  ; 1304
                                                   51  A7 9A 0002D            MOVZBL  32(IDX_DFN), R1                         ; 1305
                                                   50  C6 00031            DIVL2   R1, R0                                  ; 1306
                                                   52  A7 9A 00034            MOVZBL  32(IDX_DFN), R2                         ; 1307
                                                   50  C4 00038            MULL2   R2, R0                                  ; 1312
                                         52        50  C1 0003B            ADDL3   REC_ADDR, R0, SPL_PNT                   ; 1316
                                         50        52  C3 0003F            SUBL3   REC_ADDR, SPL_PNT, R0                   ; 1328
                                                   51  A7 9A 00043            MOVZBL  32(IDX_DFN), R1                         ;
                                   009C  C9        50  C7 00047            DIVL3   R1, R0, 156(IRAB)                       ;
                                                   54  0094 C9 D0 0004D        MOVL    148(IRAB), SAVE                         ;
                                              0094 C9  009C C9 D0 00052        MOVL    156(IRAB), 148(IRAB)                    ;
                                                 0000G 30 00059            BSBW    RM$CNTRL_ADDR                           ;
                                              0094 C9    54  D0 0005C        MOVL    SAVE, 148(IRAB)                         ;
                                                   6E    52  D1 00061        CMPL    SPL_PNT, INS_PNT                        ;
                                                         28  1F 00064        BLSSU   1$                                      ;
                                                   44 A9   08  88 00066        BISB2   #8, 68(IRAB)                            ;
                                         51        50      52  C3 0006A        SUBL3   SPL_PNT, VBN_ADDR, R1                   ;
                                                   50  0C  AE  3C 0006E        MOVZWL  TEMP, R0                                ;
                                                   54  0E  AE  3C 00072        MOVZWL  TEMP+2, R4                              ;
                                                   50      54  C0 00076        ADDL2   R4, R0                                  ;
                                                   50      51  D1 00079        CMPL    R1, R0                                  ;
                                                           10  1E 0007C        BGEQU   1$                                      ;
                                                   52      6E  D1 0007E        CMPL    INS_PNT, SPL_PNT                        ;
                                                           0B  13 00081        BEQL    1$                                      ;
                                                   50  20  A7  9A 00083        MOVZBL  32(IDX_DFN), R0                         ;
                                                   52      50  C2 00087        SUBL2   R0, SPL_PNT                             ;
                                                       009C C9  D7 0008A        DECL    156(IRAB)                               ;
                                                     00BA  31   0008E 1$:       BRW     14$                                     ;
                                              0C   AE  0094 C9  D0 00091 2$:    MOVL    148(IRAB), SAVE_COUNT                   ;
                                           0098 C9         56  D0 00097        MOVL    REC_ADDR, 152(IRAB)                     ;
                                                   53      6E  D1 0009C        CMPL    INS_PNT, EOB                            ;
                                                           03  12 0009F        BNEQ    3$                                      ;
                                                         0097  31   000A1      BRW     12$                                     ;
                                              04   AE      0C  A5 9A 000A4 3$:    MOVZBL  12(BKT_ADDR), BUCKET_LEVEL              ;
                                                           04  12 000A9        BNEQ    4$                                      ;
                                              04   AE          01  CE 000AB        MNEGL   #1, BUCKET_LEVEL                        ;
                                                               54  D4 000AF 4$:    CLRL    NEW_REC_SZ                              ;
                                         50                    56  C3 000B1        SUBL3   REC_ADDR, EOB, R0                       ;
                                                               50  02 C6 000B5        DIVL2   #2, R0                                  ;
                                         52                    50  56 C1 000B8        ADDL3   REC_ADDR, R0, SPL_PNT                   ;
                                                             0094  C9 D4 000BC        CLRL    148(IRAB)                               ;
                                              08   AE          56  D0 000C0 5$:    MOVL    REC_ADDR, LRA                           ;
                                                               6E  56 D1 000C4        CMPL    REC_ADDR, INS_PNT                       ;
                                                               16  12 000C7        BNEQ    7$                                      ;
                                         05   44   A9          03  E1 000C9        BBC     #3, 68(IRAB), 6$                        ;
                                                             0000G 30 000CE        BSBW    RM$GETNEXT_REC                          ;
                                                               40  11 000D1        BRB     10$                                     ;
                                                             0000G 30 000D3 6$:    BSBW    RM$RECORD_SIZE                          ;
                                                   54          50  3C 000D6        MOVZWL  TEMP, NEW_REC_SZ                        ;
                                              44   A9          08  88 000D9        BISB2   #8, 68(IRAB)                            ;
                                                               34  11 000DD        BRB     10$                                     ;
                                                   51      04  AE  D0 000DF 7$:    MOVL    BUCKET_LEVEL, R1                        ;
                                                             0000G 30 000E3        BSBW    RM$REC_OVHD                             ;
```

```
                58              56              50  C1 000E6          ADDL3   RECORD_OVHD, REC_ADDR, R8                ; 1441
                56              58              51  C1 000EA          ADDL3   REC_SIZE, R8, REC_ADDR
                                          0094  C9  D6 000EE          INCL    148(IRAB)                                ; 1443
                                            0C  A5  95 000F2          TSTB    12(BKT_ADDR)                             ; 1452
                                                07  12 000F5          BNEQ    8$
                07          1C  A7              06  E0 000F7          BBS     #6, 28(IDX_DFN), 9$                      ; 1454
                                                15  13 000FC          BEQL    10$                                      ; 1456
                10          1C  A7              03  E1 000FE  8$:     BBC     #3, 28(IDX_DFN), 10$                     ; 1458
                                             01 A046  95 00103  9$:     TSTB    1(RECORD_OVHD)[REC_ADDR]                 ; 1460
                                                0A  12 00107          BNEQ    10$
                            6E                  56  D1 00109          CMPL    REC_ADDR, INS_PNT                        ; 1462
                                                05  1E 0010C          BGEQU   10$
                        0098    C9              56  D0 0010E          MOVL    REC_ADDR, 152(IRAB)                      ; 1464
                50              56              54  C1 00113  10$:    ADDL3   NEW_REC_SZ, REC_ADDR, R0                 ; 1468
                                52              50  D1 00117          CMPL    R0, SPL_PNT
                                                A4  1B 0011A          BLEQU   5$
                15          44  A9              03  E1 0011C          BBC     #3, 68(IRAB), 11$                        ; 1498
                                53              56  C2 00121          SUBL2   REC_ADDR, R3                             ; 1500
                                54              53  D1 00124          CMPL    R3, NEW_REC_SZ
                                                0D  1E 00127          BGEQU   11$
                            6E                  56  D1 00129          CMPL    REC_ADDR, INS_PNT                        ; 1502
                                                08  13 0012C          BEQL    11$
                            56          08      AE  D0 0012E          MOVL    LRA, REC_ADDR                            ; 1505
                                          0094  C9  D7 00132          DECL    148(IRAB)                                ; 1506
                            52                  56  D0 00136  11$:    MOVL    REC_ADDR, SPL_PNT                        ; 1512
                                                03  11 00139          BRB     13$                                      ; 1357
                            52                  6E  D0 0013B  12$:    MOVL    INS_PNT, SPL_PNT                         ; 1519
                    009C    C9          0094    C9  D0 0013E  13$:    MOVL    148(IRAB), 156(IRAB)                     ; 1521
                    0094    C9            0C    AE  D0 00145          MOVL    SAVE_COUNT, 148(IRAB)                    ; 1522
                            50                  52  D0 0014B  14$:    MOVL    SPL_PNT, R0                              ; 1526
                            5E                  10  C0 0014E          ADDL2   #16, SP                                  ; 1527
                                          091C  8F  BA 00151          POPR    #^M<R2,R3,R4,R8,R11>
                                                    05 00155          RSB                                              ;

; Routine Size:  342 bytes,    Routine Base:  RM$RMS3 + 0000
```

```
;1467     1528  1 %SBTTL 'RM$EXT_HIGH_KEY'
;1468     1529  1 ROUTINE RM$EXT_HIGH_KEY : RL$LINKAGE NOVALUE =
;1469     1530  1
;1470     1531  1 !+++
;1471     1532  1 !
;1472     1533  1 !  FUNCTIONAL DESCRIPTION:
;1473     1534  1 !
;1474     1535  1 !       This routine extracts the high key from the bucket in IRB$L_CURBDB
;1475     1536  1 !       and places it in keybuffer 2.
;1476     1537  1 !
;1477     1538  1 !  CALLING SEQUENCE:
;1478     1539  1 !
;1479     1540  1 !       RM$EXT_HIGH_KEY()
;1480     1541  1 !
;1481     1542  1 !  INPUT PARAMETERS:
;1482     1543  1 !       NONE
;1483     1544  1 !
;1484     1545  1 !  IMPLICIT INPUT:
;1485     1546  1 !
;1486     1547  1 !       BKT_ADDR                - address of the bucket
;1487     1548  1 !           BKT$W_FREESPACE     - offset to first free byte in the bucket
;1488     1549  1 !           BKT$B_LEVEL         - level of the bucket
;1489     1550  1 !
;1490     1551  1 !       IDX_DFN                 - address of the index descriptor
;1491     1552  1 !           IDX$V_IDX_COMPR     - if set, index key compression is enabled
;1492     1553  1 !           IDX$V_KEY_COMPR     - if set, key compression is enabled
;1493     1554  1 !           IDX$B_KEYSZ         - size of the key
;1494     1555  1 !
;1495     1556  1 !       IFAB                    - address of te IFAB
;1496     1557  1 !           IFB$B_PLG_VER       - prologue version of the file
;1497     1558  1 !           IFB$W_KBUFSZ        - size of a contigious keybuffer
;1498     1559  1 !
;1499     1560  1 !       IRAB                    - address of the IRAB
;1500     1561  1 !           IRB$L_CURBDB        - address of BDB for current bucket
;1501     1562  1 !           IRB$L_KEYBUF        - address of contigious keybuffers
;1502     1563  1 !
;1503     1564  1 !  OUTPUT PARAMETERS:
;1504     1565  1 !       NONE
;1505     1566  1 !
;1506     1567  1 !  IMPLICIT OUTPUT:
;1507     1568  1 !
;1508     1569  1 !       IRB$L_LST_NCMP          - address of last record in bucket with a zero
;1509     1570  1 !                                 front compressed key
;1510     1571  1 !
;1511     1572  1 !       Keybuffer 2 has high key value of bucket.
;1512     1573  1 !
;1513     1574  1 !       REC_ADDR                - address of last record in the bucket
;1514     1575  1 !
;1515     1576  1 !  ROUTINE VALUE:
;1516     1577  1 !       NONE
;1517     1578  1 !
;1518     1579  1 !  SIDE EFFECTS:
;1519     1580  1 !
;1520     1581  1 !       AP is trashed.
;1521     1582  1 !
;1522     1583  1 !---
;1523     1584  2     BEGIN
```

```
;  1524          1585  2
;  1525          1586  2      BUILTIN
;  1526          1587  2          AP;
;  1527          1588  2
;  1528          1589  2      EXTERNAL REGISTER
;  1529          1590  2          R_IDX_DFN_STR,
;  1530          1591  2          R_IFAB_STR,
;  1531          1592  2          R_IRAB_STR;
;  1532          1593  2
;  1533          1594  2      GLOBAL REGISTER
;  1534          1595  2          R_BKT_ADDR_STR,
;  1535          1596  2          R_REC_ADDR_STR;
;  1536          1597  2
;  1537          1598  2      LOCAL
;  1538          1599  2          EOB,
;  1539          1600  2          RECORD_OVHD;
;  1540          1601  2
;  1541          1602  2      ! Setup several variables inorder to be able to position to the last record
;  1542          1603  2      ! with a key in the bucket.
;  1543          1604  2      !
;  1544          1605  2      BKT_ADDR = .BBLOCK[.IRAB[IRB$L_CURBDB], BDB$L_ADDR];
;  1545          1606  2      EOB = .BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE];
;  1546          1607  2      REC_ADDR = .BKT_ADDR + BKT$C_OVERHDSZ;
;  1547          1608  2
;  1548          1609  2      ! The bucket is NOT a prologue 3 index bucket with non-compressed key
;  1549          1610  2      ! index records. In every situation but this one, the size of the records
;  1550          1611  2      ! in the bucket are variable, so in order to find the last record in the
;  1551          1612  2      ! bucket, the entire bucket must be scanned record-by-record. If the keys
;  1552          1613  2      ! of the records in the bucket are compressed, during this scan the address
;  1553          1614  2      ! of the last record with a zero-front compressed key is saved to be used in
;  1554          1615  2      ! expanding the key of the last record in the bucket.
;  1555          1616  2      !
;  1556          1617  3      IF  NOT (.IFAB[IFB$B_PLG_VER] GEQU PLG$C_VER_3
;  1557          1618  3               AND
;  1558          1619  3               .BKT_ADDR[BKT$B_LEVEL] NEQU 0
;  1559          1620  3               AND
;  1560          1621  3              NOT .IDX_DFN[IDX$V_IDX_COMPR])
;  1561          1622  2      THEN
;  1562          1623  3          BEGIN
;  1563          1624  3
;  1564          1625  3          LOCAL
;  1565          1626  3              BUCKET_LEVEL;
;  1566          1627  3
;  1567          1628  3          ! Set BUCKET_LEVEL to the index level, if the bucket is an index bucket;
;  1568          1629  3          ! otherwise, set it to -1.
;  1569          1630  3          !
;  1570          1631  3          IF (BUCKET_LEVEL = .BKT_ADDR[BKT$B_LEVEL]) EQLU 0
;  1571          1632  3          THEN
;  1572          1633  3              BUCKET_LEVEL = -1;
;  1573          1634  3
;  1574          1635  3          ! Scan the bucket until the end of the bucket is encountered.
;  1575          1636  3          !
;  1576          1637  3          WHILE 1 DO
;  1577          1638  4          BEGIN
;  1578          1639  4
;  1579          1640  4              LOCAL
;  1580          1641  4                  REC_SIZE;
```

```
;  1581          1642  4
;  1582          1643  4              RECORD_OVHD = RM$REC_OVHD(.BUCKET_LEVEL; REC_SIZE);
;  1583          1644  4
;  1584          1645  4              ! If the keys in this bucket are compressed, and the current
;  1585          1646  4              ! record's key is zero front compressed, then save the address
;  1586          1647  4              ! of this record.
;  1587          1648  4              !
;  1588          1649  4              IF  .IFAB[IFB$B_PLG_VER] GEQU PLG$C_VER_3
;  1589          1650  4                  AND
;  1590          1651  5                  NOT (.BKT_ADDR[BKT$B_LEVEL] EQLU 0
;  1591          1652  5                          AND
;  1592          1653  5                          NOT .IDX_DFN[IDX$V_KEY_COMPR])
;  1593          1654  4                   AND
;  1594          1655  4                  .(.REC_ADDR + .RECORD_OVHD)<8,8> EQLU 0
;  1595          1656  4              THEN
;  1596          1657  4                  IRAB[IRB$L_LST_NCMP] = .REC_ADDR;
;  1597          1658  4
;  1598          1659  4              ! If this is the last record in the bucket, exit the loop.
;  1599          1660  4              !
;  1600          1661  4              IF (.REC_ADDR + .RECORD_OVHD + .REC_SIZE) GEQU .EOB
;  1601          1662  4              THEN
;  1602          1663  4                  EXITLOOP
;  1603          1664  4
;  1604          1665  4              ! Else position to the next record and continue the scan.
;  1605          1666  4              !
;  1606          1667  4              ELSE
;  1607          1668  4                  REC_ADDR = .REC_ADDR + .RECORD_OVHD + .REC_SIZE;
;  1608          1669  3              END;
;  1609          1670  3          END
;  1610          1671  3
;  1611          1672  3      ! The bucket is a prologue 3 index bucket containing non-compressed key
;  1612          1673  3      ! index records. As the records are of fixed size, RMS is able to quickly
;  1613          1674  3      ! position to the last record in the index bucket.
;  1614          1675  3      !
;  1615          1676  2      ELSE
;  1616          1677  2          REC_ADDR = .EOB - .IDX_DFN[IDX$B_KEYSZ];
;  1617          1678  2
;  1618          1679  2      ! Having positioned to the last record in the bucket, RMS can extract its
;  1619          1680  2      ! key into keybuffer 2, expanding it as is necessary.
;  1620          1681  2      !
;  1621          1682  3      BEGIN
;  1622          1683  3
;  1623          1684  3      GLOBAL REGISTER
;  1624          1685  3          R_BDB,
;  1625          1686  3          R_RAB,
;  1626          1687  3          R_IMPURE;
;  1627          1688  3
;  1628          1689  3      IF .IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3
;  1629          1690  3      THEN
;  1630          1691  3          RM$MOVE (.IDX_DFN[IDX$B_KEYSZ],
;  1631          1692  3                   .REC_ADDR + .RECORD_OVHD,
;  1632          1693  3                   KEYBUF_ADDR(2))
;  1633          1694  3      ELSE
;  1634          1695  4          BEGIN
;  1635          1696  4          AP = 0;
;  1636          1697  4          RM$RECORD_KEY (KEYBUF_ADDR(2));
;  1637          1698  3          END;
```

```
;  1638          1699  2      END;
;  1639          1700  2
;  1640          1701  2      RETURN;
;  1641          1702  1      END;


                        0970   8F   BB 00000 RM$EXT_HIGH_KEY:
                                                PUSHR   #^M<R4,R5,R6,R8,R11>                    ;  1529
                    50    20   A9   D0 00004          MOVL    32(IRAB), R0                       ;  1605
                    55    18   A0   D0 00008          MOVL    24(R0), BKT_ADDR                   ;
                    5B    04   A5   3C 0000C          MOVZWL  4(BKT_ADDR), EOB                   ;  1606
                    5B         55   C0 00010          ADDL2   BKT_ADDR, EOB                      ;
                    56    0E   A5   9E 00013          MOVAB   14(R5), REC_ADDR                   ;  1607
                    03  00B7   CA   91 00017          CMPB    183(IFAB), #3                      ;  1617
                              0A   1F 0001C          BLSSU   1$                                 ;
                        0C   A5   95 0001E          TSTB    12(BKT_ADDR)                       ;  1619
                              05   13 00021          BEQL    1$                                 ;
                 3C     1C   A7        03   E1 00023          BBC     #3, 28(IDX_DFN), 5$        ;  1621
                    58    0C   A5   9A 00028 1$:      MOVZBL  12(BKT_ADDR), BUCKET_LEVEL         ;  1631
                              03   12 0002C          BNEQ    2$                                 ;
                    58         01   CE 0002E          MNEGL   #1, BUCKET_LEVEL                   ;  1633
                    51         58   D0 00031 2$:      MOVL    BUCKET_LEVEL, R1                   ;  1643
                          0000G  30 00034          BSBW    RM$REC_OVHD                        ;
                    03  00B7   CA   91 00037          CMPB    183(IFAB), #3                      ;  1649
                              15   1F 0003C          BLSSU   4$                                 ;
                        0C   A5   95 0003E          TSTB    12(BKT_ADDR)                       ;  1651
                              05   12 00041          BNEQ    3$                                 ;
                 0B     1C   A7        06   E1 00043          BBC     #6, 28(IDX_DFN), 4$        ;  1653
                         01 A046   95 00048 3$:      TSTB    1(RECORD_OVHD)[REC_ADDR]           ;  1655
                              05   12 0004C          BNEQ    4$                                 ;
                0098    C9        56   D0 0004E          MOVL    REC_ADDR, 152(IRAB)            ;  1657
                54      56        50   C1 00053 4$:      ADDL3   RECORD_OVHD, REC_ADDR, R4      ;  1661
                          54        51   C0 00057          ADDL2   REC_SIZE, R4                   ;
                          5B        54   D1 0005A          CMPL    R4, EOB                        ;
                                    0D   1E 0005D          BGEQU   6$                             ;
                          56        54   D0 0005F          MOVL    R4, REC_ADDR                   ;  1668
                                    CD   11 00062          BRB     2$                             ;  1637
                          56    20   A7   9A 00064 5$:      MOVZBL  32(IDX_DFN), REC_ADDR          ;  1677
                56        5B        56   C3 00068          SUBL3   REC_ADDR, EOB, REC_ADDR        ;
                          03  00B7   CA   91 0006C 6$:      CMPB    183(IFAB), #3                  ;  1689
                                    18   1E 00071          BGEQU   7$                             ;
                          51  00B4   CA   3C 00073          MOVZWL  180(IFAB), R1                  ;  1693
                              60 B941   9F 00078          PUSHAB  @96(IRAB)[R1]                  ;
                                    6046   9F 0007C          PUSHAB  (RECORD_OVHD)[REC_ADDR]        ;  1692
                          7E    20   A7   9A 0007F          MOVZBL  32(IDX_DFN), -(SP)             ;  1691
                                0000G  30 00083          BSBW    RM$MOVE                        ;
                          5E        0C   C0 00086          ADDL2   #12, SP                        ;
                                    11   11 00089          BRB     8$                             ;
                                    5C   D4 0008B 7$:      CLRL    AP                             ;  1696
                          50  00B4   CA   3C 0008D          MOVZWL  180(IFAB), R0                  ;  1697
                              60 B940   9F 00092          PUSHAB  @96(IRAB)[R0]                  ;
                                0000G  30 00096          BSBW    RM$RECORD_KEY                  ;
                          5E        04   C0 00099          ADDL2   #4, SP                         ;
                                    0970   8F   BA 0009C 8$:      POPR    #^M<R4,R5,R6,R8,R11>   ;  1702
```

```
                                05 000A0        RSB                             ;

; Routine Size:  161 bytes,    Routine Base:  RM$RMS3 + 0156
```

```
;  1643          1703  1 %SBTTL 'RM$MOVE_VBNS'
;  1644          1704  1 ROUTINE RM$MOVE_VBNS (NEW_BKT): RL$LINKAGE NOVALUE =
;  1645          1705  1
;  1646          1706  1 !+++
;  1647          1707  1 !
;  1648          1708  1 !  FUNCTIONAL DESCRIPTION:
;  1649          1709  1 !
;  1650          1710  1 !       This routine moves the VBNs starting with the one corresponding
;  1651          1711  1 !       to REC_COUNT into the new bucket.
;  1652          1712  1 !
;  1653          1713  1 !  CALLING SEQUENCE:
;  1654          1714  1 !
;  1655          1715  1 !       RM$MOVE_VBNS()
;  1656          1716  1 !
;  1657          1717  1 !  INPUT PARAMETERS:
;  1658          1718  1 !
;  1659          1719  1 !       NEW_BKT - Address of new bucket
;  1660          1720  1 !
;  1661          1721  1 !  IMPLICIT INPUT:
;  1662          1722  1 !
;  1663          1723  1 !       IDX_DFN
;  1664          1724  1 !       IRAB
;  1665          1725  1 !       IFAB
;  1666          1726  1 !       REC_ADDR
;  1667          1727  1 !
;  1668          1728  1 !  OUTPUT PARAMETERS:
;  1669          1729  1 !       NONE
;  1670          1730  1 !
;  1671          1731  1 !  IMPLICIT OUTPUT:
;  1672          1732  1 !       NONE
;  1673          1733  1 !
;  1674          1734  1 !  ROUTINE VALUE:
;  1675          1735  1 !       NONE
;  1676          1736  1 !
;  1677          1737  1 !  SIDE EFFECTS:
;  1678          1738  1 !       NONE
;  1679          1739  1 !
;  1680          1740  1 !---
;  1681          1741  1
;  1682          1742  2     BEGIN
;  1683          1743  2
;  1684          1744  2     EXTERNAL REGISTER
;  1685          1745  2         R_IDX_DFN_STR,
;  1686          1746  2         R_IRAB_STR,
;  1687          1747  2         R_IFAB,
;  1688          1748  2         R_REC_ADDR;
;  1689          1749  2
;  1690          1750  2     GLOBAL REGISTER
;  1691          1751  2         R_BKT_ADDR_STR;
;  1692          1752  2
;  1693          1753  2     MAP
;  1694          1754  2         NEW_BKT : REF BBLOCK;
;  1695          1755  2
;  1696          1756  2     LOCAL
;  1697          1757  2         OLD_FREE  : REF BBLOCK, ! Address of old buckets VBN free pointer
;  1698          1758  2         NEW_FREE  : REF BBLOCk, ! Address of new buckets VBN free pointer
;  1699          1759  2         BKTSZ,                  ! Number of bytes in these buckets
```

```
2          NEW_END,                    ! Address of new end of old bucket's VBN chain
2          OLD_END,                    ! Address of old end of old bucket's VBN chain
2          SIZE,                       ! Number of bytes to move
2          DEST_ADDR;                  ! Starting address of VBN chain in new bucket
2
2      ! First compute the number of bytes to move.
2      !
2      BKT_ADDR = .BBLOCK [.IRAB [IRB$L_CURBDB], BDB$L_ADDR];
2      BKTSZ = .IDX_DFN [IDX$B_IDXBKTSZ];
2
2      BKTSZ = .BKTSZ * 512;
2      OLD_FREE = .BKT_ADDR + .BKTSZ - BKT$C_ENDOVHD;
2      NEW_FREE = .NEW_BKT + .BKTSZ - BKT$C_ENDOVHD;
2
2      ! Now get the start and end of the VBN chain to move.
2      !
3      BEGIN
3
3      GLOBAL REGISTER
3          R_RAB,
3          R_IMPURE;
3
3      LOCAL
3          SAVE;
3
3      SAVE = .IRAB [IRB$L_REC_COUNT];
3      IRAB [IRB$L_REC_COUNT] = .IRAB [IRB$L_SPL_COUNT];
3      NEW_END = RM$CNTRL_ADDR() + .BKT_ADDR [BKT$V_PTR_SZ] + 2;
3      IRAB [IRB$L_REC_COUNT] = .SAVE;
2      END;
2
2      OLD_END = .BKT_ADDR + .OLD_FREE [0,0,16,0];
2      SIZE = .NEW_END - .OLD_END;
2      OLD_FREE [0,0,16,0] = .OLD_FREE [0,0,16,0] + .SIZE - 1;
2
2      ! Finally get the destination address.
2      !
2      DEST_ADDR = .NEW_FREE - .SIZE;
2
2      CH$MOVE (.SIZE, .OLD_END, .DEST_ADDR);
2
2      NEW_FREE [0,0,16,0] = .DEST_ADDR - .NEW_BKT;
2
2      ! Move the PTR_SZ from the old bucket to the new bucket.
2      !
2      NEW_BKT [BKT$V_PTR_SZ] = .BKT_ADDR [BKT$V_PTR_SZ];
1      END;
```

```
                 093C   8F  BB 00000 RM$MOVE_VBNS:
                                        PUSHR   #^M<R2,R3,R4,R5,R8,R11>                  ; 1704
        50        20   A9  D0 00004       MOVL    32(IRAB), R0                             ; 1767
        55        18   A0  D0 00008       MOVL    24(R0), BKT_ADDR                         ;
        50        16   A7  9A 0000C       MOVZBL  22(IDX_DFN), BKTSZ                       ; 1768
```

```
                50          50          09  78 00010        ASHL    #9, BKTSZ, BKTSZ                        ; 1770
                            53      FC A045  9E 00014        MOVAB   -4(BKTSZ)[BKT_ADDR], OLD_FREE           ; 1771
                            51      1C   AE  D0 00019        MOVL    NEW_BKT, R1                             ; 1772
                                    FC A041  9F 0001D        PUSHAB  -4(BKTSZ)[R1]                           ;
                            52    0094   C9  D0 00021        MOVL    148(IRAB), SAVE                         ; 1785
                      0094  C9    009C   C9  D0 00026        MOVL    156(IRAB), 148(IRAB)                    ; 1786
                                  0000G  30 0002D            BSBW    RM$CNTRL_ADDR                           ; 1787
     5B       0D   A5      02          03  EF 00030        EXTZV   #3, #2, T3(BKT_ADDR), R11               ;
                            51      02 AB40  9E 00036        MOVAB   2(R11)[R0], NEW_END                     ;
                      0094  C9          52  D0 0003B        MOVL    SAVE, 148(IRAB)                         ; 1788
                            50          63  3C 00040        MOVZWL  (OLD_FREE), R0                          ; 1791
                52          55          50  C1 00043        ADDL3   R0, BKT_ADDR, OLD_END                   ;
                            51          52  C2 00047        SUBL2   OLD_END, SIZE                           ; 1792
                            54      FF A140  9E 0004A        MOVAB   -1(SIZE)[R0], R4                        ; 1793
                            63          54  B0 0004F        MOVW    R4, (OLD_FREE)                          ;
                58          6E          51  C3 00052        SUBL3   SIZE, NEW_FREE, DEST_ADDR               ; 1797
                68          62          51  28 00056        MOVC3   SIZE, (OLD_END), (DEST_ADDR)            ; 1799
                9E          58      20   AE  A3 0005A        SUBW3   NEW_BKT, DEST_ADDR, @NEW_FREE           ; 1801
                50      1C  AE          0D  C1 0005F        ADDL3   #13, NEW_BKT, R0                        ; 1805
     60         02          03          5B  F0 00064        INSV    R11, #3, #2, (R0)                       ;
                                    093C   8F  BA 00069        POPR    #^M<R2,R3,R4,R5,R8,R11>                 ; 1806
                                            05 0006D        RSB                                             ;

; Routine Size:  110 bytes,    Routine Base:  RM$RMS3 + 01F7
```

```
;  1748          1807  1 %SBTTL 'RM$SPLIT_EM'
;  1749          1808  1 GLOBAL ROUTINE RM$SPLIT_EM : RL$RABREG_67 =
;  1750          1809  1
;  1751          1810  1 !++
;  1752          1811  1 !
;  1753          1812  1 ! FUNCTIONAL DESCRIPTION:
;  1754          1813  1 !
;  1755          1814  1 ! Split the bucket in CURBDB into NXTBDB for SIDR data level and
;  1756          1815  1 ! any index level as close to 50-50 as possible.  Records are moved
;  1757          1816  1 ! into the new bucket, and the new record is inserted where appropriate.
;  1758          1817  1 ! Keybuffer 2 contains the high key value of the left hand bucket after
;  1759          1818  1 ! splitting.  No buckets are actually written out in this routine, only
;  1760          1819  1 ! the records are moved.
;  1761          1820  1 !
;  1762          1821  1 ! CALLING SEQUENCE:
;  1763          1822  1 !
;  1764          1823  1 !       RM$SPLIT_EM()
;  1765          1824  1 !
;  1766          1825  1 ! INPUT PARAMETERS:
;  1767          1826  1 !       NONE
;  1768          1827  1 !
;  1769          1828  1 ! IMPLICIT INPUTS:
;  1770          1829  1 !
;  1771          1830  1 !       IDX_DFN - pointer to index descriptor for this key of reference
;  1772          1831  1 !           IDX$B_KEYSZ - size of key
;  1773          1832  1 !
;  1774          1833  1 !       REC_ADDR - point in bucket where new record is to be inserted
;  1775          1834  1 !
;  1776          1835  1 !       IRAB - pointer to internal rab structure
;  1777          1836  1 !           IRB$V_REC_W_LO - 0 on input
;  1778          1837  1 !           IRB$V_DUPS_SEEN - set if positioned after existing record to add
;  1779          1838  1 !                             sidr array entry
;  1780          1839  1 !           IRB$L_LST_REC - address of beginning of record previous to one
;  1781          1840  1 !                           rec_addr is positioned to (used in conjunction with
;  1782          1841  1 !                           irb$v_dups_seen)
;  1783          1842  1 !           IRB$B_STOPLEVEL - level at which insert is being done (non-zero
;  1784          1843  1 !                           if this is primary key only)
;  1785          1844  1 !           IRB$L_CURBDB - bdb describing bucket to split (left hand bucket)
;  1786          1845  1 !           IRB$L_NXTBDB - bdb describing bucket to split into (right hand)
;  1787          1846  1 !
;  1788          1847  1 !       IFAB - pointer to internal fab structure
;  1789          1848  1 !           IFB$W_KBUFSZ - size of keybuffers
;  1790          1849  1 !
;  1791          1850  1 !       Routines called by this routine will reference additional fields in
;  1792          1851  1 !       the above structures.
;  1793          1852  1 !
;  1794          1853  1 ! OUTPUT PARAMETERS:
;  1795          1854  1 !       NONE
;  1796          1855  1 !
;  1797          1856  1 ! IMPLICIT OUTPUTS:
;  1798          1857  1 !
;  1799          1858  1 !       keybuffer 2 (irb$l_keybuff + ifb$w_kbufsz) contains high
;  1800          1859  1 !       value of left hand bucket.
;  1801          1860  1 !
;  1802          1861  1 ! ROUTINE VALUE:
;  1803          1862  1 !       NONE
;  1804          1863  1 !
```

```
;  1805          1864  1 ! SIDE EFFECTS:
;  1806          1865  1 !
;  1807          1866  1 !       All records in the original bucket described by IRB$L_CURBDB
;  1808          1867  1 !       and the new record are split between that bucket and the new
;  1809          1868  1 !       (empty) bucket described by IRB$L_NXTBDB.  Only when one record
;  1810          1869  1 !       occupies the entire bucket (curbdb) will a continuation record
;  1811          1870  1 !       be created, else the entire record will be moved to the new bucket.
;  1812          1871  1 !       ID's are reassigned in the new SIDR bucket as well.
;  1813          1872  1 !
;  1814          1873  1 !
;  1815          1874  1 !--
;  1816          1875  1
;  1817          1876  2     BEGIN
;  1818          1877  2
;  1819          1878  2     EXTERNAL REGISTER
;  1820          1879  2         COMMON_RAB_STR,
;  1821          1880  2         R_REC_ADDR_STR,
;  1822          1881  2         R_IDX_DFN_STR;
;  1823          1882  2
;  1824          1883  2     LOCAL
;  1825          1884  2         EOB;
;  1826          1885  2
;  1827          1886  3     BEGIN                           ! block to limit scope of following locals
;  1828          1887  3
;  1829          1888  3     LOCAL
;  1830          1889  3         INS_PNT;
;  1831          1890  3
;  1832          1891  3     GLOBAL REGISTER
;  1833          1892  3         R_BKT_ADDR_STR;
;  1834          1893  3
;  1835          1894  3     BKT_ADDR = .BBLOCK[.IRAB[IRB$L_CURBDB], BDB$L_ADDR];
;  1836          1895  3     EOB = .BKT_ADDR + .BKT_ADDR [BKT$W_FREESPACE];
;  1837          1896  3
;  1838          1897  3     ! First compute the split point.
;  1839          1898  3     !
;  1840          1899  3     INS_PNT = .REC_ADDR;
;  1841          1900  3
;  1842          1901  3     REC_ADDR = RM$COMP_SPL_PNT(.INS_PNT);
;  1843          1902  3
;  1844          1903  3     ! If the split point ended up exactly where we wanted to insert the record
;  1845          1904  3     ! anyway we must investigate a little further.
;  1846          1905  3     !
;  1847          1906  3     IF .REC_ADDR EQLA .INS_PNT
;  1848          1907  3     THEN
;  1849          1908  3         IF NOT .IRAB[IRB$V_REC_W_LO]
;  1850          1909  3         THEN
;  1851          1910  3             IF  .BKT_ADDR[BKT$B_LEVEL] EQL 0
;  1852          1911  3                 AND
;  1853          1912  3                 .IRAB[IRB$V_DUPS_SEEN]
;  1854          1913  3             THEN
;  1855          1914  3
;  1856          1915  3                 ! This is a continuation record being added to the end of
;  1857          1916  3                 ! an existing record so keep it with the low set.
;  1858          1917  3                 !
;  1859          1918  3                 IF .REC_ADDR NEQA .EOB
;  1860          1919  3                 THEN
;  1861          1920  3                     IRAB[IRB$V_REC_W_LO] = 1
```

```
;  1862     1921  3                ! RMS is trying to insert a continuation record at the end
;  1863     1922  3                ! of the bucket.  Unless the existing record occupies the
;  1864     1923  3                ! entire bucket RMS will move the entire record into the new
;  1865     1924  3                ! bucket and add the continuation record there.
;  1866     1925  3                !
;  1867     1926  3                ELSE
;  1868     1927  3                    BEGIN
;  1869     1928  4
;  1870     1929  4                    ! RMS is going to back up the split point to the
;  1871     1930  4                    ! beginning of this record so that it does not create a
;  1872     1931  4                    ! continuation bucket, but force the whole thing
;  1873     1932  4                    ! into the new bucket
;  1874     1933  4                    !
;  1875     1934  4                    IF .IRAB[IRB$L_LST_REC] NEQA .BKT_ADDR + BKT$C_OVERHDSZ
;  1876     1935  4                    THEN
;  1877     1936  4                        REC_ADDR = .IRAB[IRB$L_LST_REC]
;  1878     1937  4
;  1879     1938  4                    ! The existing record that RMS is positioned at the end
;  1880     1939  4                    ! of occupies the entire bucket, therefore RMS is forced
;  1881     1940  4                    ! to make the new record anew in the new bucket, causing a
;  1882     1941  4                    ! continuation bucket. This is the only time RMS will
;  1883     1942  4                    ! ever create a continuation bucket at the SIDR data level
;  1884     1943  4                    ! and it is done with great reluctance. Clearing DUPS_SEEN
;  1885     1944  4                    ! will cause the record size to be correct when it is
;  1886     1945  4                    ! recalculated later.
;  1887     1946  4                    !
;  1888     1947  4                    ELSE
;  1889     1948  4                        BEGIN
;  1890     1949  5                        IRAB[IRB$L_LST_REC] = .INS_PNT;
;  1891     1950  5                        IRAB[IRB$V_DUPS_SEEN] = 0;
;  1892     1951  5                        IRAB[IRB$V_CONT_BKT] = 1;
;  1893     1952  5                        END;
;  1894     1953  4
;  1895     1954  4                    END;
;  1896     1955  3
;  1897     1956  3            ! Now set up things as offsets to simplify life later on.
;  1898     1957  3            !
;  1899     1958  3            IRAB[IRB$W_POS_INS] = .INS_PNT - .BKT_ADDR;
;  1900     1959  3            IRAB[IRB$W_SPLIT] = .REC_ADDR - .BKT_ADDR;
;  1901     1960  3
;  1902     1961  3            ! Adjust the freespace pointer for the old (left) bucket to reflect
;  1903     1962  3            ! its new size.
;  1904     1963  3            !
;  1905     1964  3            BKT_ADDR[BKT$W_FREESPACE] = .IRAB[IRB$W_SPLIT];
;  1906     1965  3            IRAB[IRB$L_LST_REC] = .IRAB[IRB$L_LST_REC] - .BKT_ADDR;
;  1907     1966  3            END;                                    ! of block defining INS_PNT, BKT_ADDR
;  1908     1967  2
;  1909     1968  2            ! RMS moves everything beyond the split point into the new bucket
;  1910     1969  2            ! without caring where the new record or records are going to go.
;  1911     1970  2            !
;  1912     1971  2            BEGIN
;  1913     1972  3
;  1914     1973  3            LOCAL
;  1915     1974  3                COMPRESSION,
;  1916     1975  3                NEW_BKT : REF BBLOCK;
;  1917     1976  3
;  1918     1977  3
```

```
;  1919          1978  3     COMPRESSION = 0;
;  1920          1979  3
;  1921          1980  3     NEW_BKT = .BBLOCK[.IRAB[IRB$L_NXTBDB], BDB$L_ADDR];
;  1922          1981  3
;  1923          1982  3     ! If there is anything at all to move into the new bucket, RMS knows
;  1924          1983  3     ! to move it without any regard as to where the new record or records will
;  1925          1984  3     ! be inserted.
;  1926          1985  3     !
;  1927          1986  3     IF (NEW_BKT[BKT$W_FREESPACE] = .EOB - .REC_ADDR) NEQU 0
;  1928          1987  3     THEN
;  1929          1988  4         BEGIN
;  1930          1989  4
;  1931          1990  4         LOCAL
;  1932          1991  4             REC_FROM    : REF BBLOCK,
;  1933          1992  4             REC_TO      : REF BBLOCK,
;  1934          1993  4             RECORD_OVHD,
;  1935          1994  4             LENGTH;
;  1936          1995  4
;  1937          1996  4         REC_FROM   = .REC_ADDR;
;  1938          1997  4         REC_TO = .NEW_BKT + BKT$C_OVERHDSZ;
;  1939          1998  4         LENGTH = .NEW_BKT[BKT$W_FREESPACE];
;  1940          1999  4
;  1941          2000  4         ! Determine the number of bytes of record overhead for the current
;  1942          2001  4         ! record. This record will become the first record in the new bucket.
;  1943          2002  4         !
;  1944          2003  5         BEGIN
;  1945          2004  5
;  1946          2005  5         LOCAL
;  1947          2006  5             REC_SIZE;
;  1948          2007  5
;  1949          2008  5         IF (REC_SIZE = .NEW_BKT[BKT$B_LEVEL]) EQLU 0
;  1950          2009  5         THEN
;  1951          2010  5             REC_SIZE = -1;
;  1952          2011  5
;  1953          2012  5         RECORD_OVHD = RM$REC_OVHD(.REC_SIZE);
;  1954          2013  4         END;
;  1955          2014  4
;  1956          2015  4         ! If this is a prologue 3 bucket with compressed keys, and if the key
;  1957          2016  4         ! of the record which is to be the first record in the new bucket is
;  1958          2017  4         ! not zero front compressed, RMS constructs the contents of the new
;  1959          2018  4         ! bucket in two steps. First, RMS moves in the new low order key front
;  1960          2019  4         ! expanding and rear-end recompressing as required. Finally, the
;  1961          2020  4         ! remainder of the contents of the new index/SIDR bucket is moved into
;  1962          2021  4         ! the new bucket.
;  1963          2022  4         !
;  1964          2023  6         IF ((.NEW_BKT[BKT$B_LEVEL] EQLU 0
;  1965          2024  6                         AND
;  1966          2025  6                        .IDX_DFN[IDX$V_KEY_COMPR])
;  1967          2026  5                  OR
;  1968          2027  6                 (.NEW_BKT[BKT$B_LEVEL] NEQU 0
;  1969          2028  6                         AND
;  1970          2029  5                        .IDX_DFN [IDX$V_IDX_COMPR]))
;  1971          2030  4             AND
;  1972          2031  5            (.(.REC_ADDR + .RECORD_OVHD)<8,8> GTRU 0)
;  1973          2032  4         THEN
;  1974          2033  5             BEGIN
;  1975          2034  5
```

```
;  1976     2035  5          BUILTIN
;  1977     2036  5              AP;
;  1978     2037  5
;  1979     2038  5          LOCAL
;  1980     2039  5              SAVE_LST_NCMP,
;  1981     2040  5              SAVE_REC_ADDR;
;  1982     2041  5
;  1983     2042  5          GLOBAL REGISTER
;  1984     2043  5              R_BDB,
;  1985     2044  5              R_BKT_ADDR;
;  1986     2045  5
;  1987     2046  5          ! If the record which is to be the first record in the new bucket
;  1988     2047  5          ! has any overhead associated with it, move the overhead into the
;  1989     2048  5          ! new bucket, and then position past it to the key of the record.
;  1990     2049  5          ! Also, position in the new bucket to the first byte past the
;  1991     2050  5          ! record overhead just moved into it.
;  1992     2051  5          !
;  1993     2052  5          IF .RECORD_OVHD GTRU 0
;  1994     2053  5          THEN
;  1995     2054  6              BEGIN
;  1996     2055  6              REC_TO = RM$MOVE (.RECORD_OVHD, .REC_ADDR, .REC_TO);
;  1997     2056  6              REC_ADDR = .REC_ADDR + .RECORD_OVHD;
;  1998     2057  5              END;
;  1999     2058  5
;  2000     2059  5          ! Move the key which is to be the low-order key in the new bucket
;  2001     2060  5          ! from its current position in the old (left) bucket to its
;  2002     2061  5          ! position as the first key in the new bucket leaving room
;  2003     2062  5          ! for the record overhead bytes (in the case of SIDRs) and the key
;  2004     2063  5          ! compression overhead bytes.
;  2005     2064  5          !
;  2006     2065  5          AP = 1;
;  2007     2066  5          RM$RECORD_KEY (.REC_TO + 2);
;  2008     2067  5
;  2009     2068  5          ! Determine the compression of the low-order key of the new bucket.
;  2010     2069  5          ! This key must be and will be zero-front compressed, but it will
;  2011     2070  5          ! be rear-end truncated if possible.
;  2012     2071  5          !
;  2013     2072  5          SAVE_REC_ADDR = .REC_ADDR;
;  2014     2073  5          REC_ADDR = .REC_TO - .RECORD_OVHD;
;  2015     2074  5
;  2016     2075  5          SAVE_LST_NCMP = .IRAB[IRB$L_LST_NCMP];
;  2017     2076  5          IRAB[IRB$L_LST_NCMP] = .REC_ADDR;
;  2018     2077  5
;  2019     2078  5          BKT_ADDR = .NEW_BKT;
;  2020     2079  5
;  2021     2080  5          RM$COMPRESS_KEY (.REC_TO);
;  2022     2081  5
;  2023     2082  5          IRAB[IRB$L_LST_NCMP] = .SAVE_LST_NCMP;
;  2024     2083  5          REC_ADDR = .SAVE_REC_ADDR;
;  2025     2084  5
;  2026     2085  5          ! Adjust the freespace offset of the new bucket to reflect the
;  2027     2086  5          ! difference between the key size of the low-order key in its
;  2028     2087  5          ! new position (as low-order key of the new bucket), and the key
;  2029     2088  5          ! size of the low-order key in its old position (as a key somewhere
;  2030     2089  5          ! in the old bucket).
;  2031     2090  5          !
;  2032     2091  5          COMPRESSION = .REC_ADDR[KEY_LEN] - .REC_TO[KEY_LEN];
```

```
;  2033          2092  5          NEW_BKT[BKT$W_FREESPACE] = .NEW_BKT[BKT$W_FREESPACE] - .COMPRESSION;
;  2034          2093  5
;  2035          2094  5          ! If a SIDR bucket split occurred, adjust the record size field
;  2036          2095  5          ! of the SIDR that becomes the first SIDR in the new bucket, (the
;  2037          2096  5          ! SIDR whose key was just front expanded), to reflect the difference
;  2038          2097  5          ! between the key size of the first SIDR in the new bucket, and the
;  2039          2098  5          ! key size of this SIDR in its old position in the old bucket
;  2040          2099  5          ! (before the split took place).
;  2041          2100  5          !
;  2042          2101  5          IF .NEW_BKT[BKT$B_LEVEL] EQLU 0
;  2043          2102  5          THEN
;  2044          2103  5              (.REC_TO - IRC$C_DATSZFLD)<0,16> =
;  2045          2104  5                      .(.REC_TO - IRC$C_DATSZFLD)<0,16> - .COMPRESSION;
;  2046          2105  5
;  2047          2106  5          ! If the new record is not to be inserted as the very first
;  2048          2107  5          ! record in the new bucket, then the difference between the low
;  2049          2108  5          ! order key in its old position in the old bucket and its new
;  2050          2109  5          ! position as low order key of the new bucket must be saved (and
;  2051          2110  5          ! already has as the value in COMPRESSION). It will be used in
;  2052          2111  5          ! recomputing the point of insertion of the new record to be
;  2053          2112  5          ! inserted if it is to go in the new bucket. If the new record is
;  2054          2113  5          ! to be inserted as the very first record in the new bucket then
;  2055          2114  5          ! this difference is not taken into account in the recomputation
;  2056          2115  5          ! of the point of insertion, and so COMPRESSION is reset to 0.
;  2057          2116  5          !
;  2058          2117  5          IF  (.IRAB[IRB$W_POS_INS] EQLU .IRAB[IRB$W_SPLIT])
;  2059          2118  6          THEN
;  2060          2119  5              COMPRESSION = 0;
;  2061          2120  5
;  2062          2121  5              ! Having completed part 1 of the construction of the new bucket
;  2063          2122  5              ! (movement and recompression of the new bucket's low order key)
;  2064          2123  5              ! set up to remove the remainder of the old bucket which is to be
;  2065          2124  5              ! moved into the new bucket. This will include the SIDR array of the
;  2066          2125  5              ! the new bucket's first record, if the split being processed is a
;  2067          2126  5              ! SIDR bucket split.
;  2068          2127  5              !
;  2069          2128  5              LENGTH = .LENGTH - .RECORD_OVHD -.REC_ADDR[KEY_LEN] - 2;
;  2070          2129  5              REC_FROM = .REC_ADDR + .REC_ADDR[KEY_LEN] + 2;
;  2071          2130  5              REC_TO = .REC_TO + .REC_TO[KEY_LEN] + 2;
;  2072          2131  5              END;
;  2073          2132  4
;  2074          2133  4          ! Move those records (and record parts if this is a SIDR bucket split)
;  2075          2134  4          ! remaining to be moved from the old to the new bucket. Note that this
;  2076          2135  4          ! may include the first record of the new bucket (in those cases not
;  2077          2136  4          ! requiring recomputation of the compression of the new low order key),
;  2078          2137  4          ! and will include the VBNs for non-prologue 3 index and SIDR buckets,
;  2079          2138  4          ! and prologue 3 SIDR buckets.
;  2080          2139  4          !
;  2081          2140  4          IF .LENGTH GTR 0
;  2082          2141  4          THEN
;  2083          2142  4              CH$MOVE(.LENGTH,.REC_FROM,.REC_TO);
;  2084          2143  4          END;
;  2085          2144  3
;  2086          2145  3      NEW_BKT[BKT$W_FREESPACE] = .NEW_BKT[BKT$W_FREESPACE] + BKT$C_OVERHDSZ;
;  2087          2146  3
;  2088          2147  3      ! The VBN chain will have to be separately moved if this is a prolgoue 3
;  2089          2148  3
```

```
;  2090     2149  3      ! index bucket.
;  2091     2150  3      !
;  2092     2151  3      IF  .IFAB [IFB$B_PLG_VER] GEQU PLG$C_VER_3
;  2093     2152  3           AND
;  2094     2153  3          .NEW_BKT[BKT$B_LEVEL] GTRU 0
;  2095     2154  3      THEN
;  2096     2155  3          RM$MOVE_VBNS(.NEW_BKT);
;  2097     2156  3
;  2098     2157  3      ! If this is a level 0 bucket (SIDR data level) RMS must now reassign the
;  2099     2158  3      ! ID's of the records we moved out. This reassigning is only required for
;  2100     2159  3      ! prologue 1 and 2 SIDR buckets, because prologue 3 SIDRs do not contain
;  2101     2160  3      ! an ID.
;  2102     2161  3      !
;  2103     2162  4      IF  (.NEW_BKT[BKT$B_LEVEL] EQL 0)
;  2104     2163  3           AND
;  2105     2164  4          (.IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3)
;  2106     2165  3      THEN
;  2107     2166  4          BEGIN
;  2108     2167  4
;  2109     2168  4          LOCAL
;  2110     2169  4              EOB,
;  2111     2170  4              ID;
;  2112     2171  4
;  2113     2172  4          ID = 1;
;  2114     2173  4          REC_ADDR = .NEW_BKT + BKT$C_OVERHDSZ;
;  2115     2174  4          EOB = .NEW_BKT + .NEW_BKT[BKT$W_FREESPACE];
;  2116     2175  4
;  2117     2176  4          ! Reassign IDs to all the records in the prologue 2 SIDR bucket.
;  2118     2177  4          !
;  2119     2178  4          WHILE .REC_ADDR LSSA .EOB
;  2120     2179  4          DO
;  2121     2180  5              BEGIN
;  2122     2181  5
;  2123     2182  5              ! Reassign the ID in the new bucket.
;  2124     2183  5              !
;  2125     2184  5              REC_ADDR[IRC$B_ID] = .ID;
;  2126     2185  5              ID = .ID + 1;
;  2127     2186  5              RM$GETNEXT_REC();
;  2128     2187  4              END;
;  2129     2188  4
;  2130     2189  4          ! Update the next ID field in bucket header.
;  2131     2190  4          !
;  2132     2191  4          NEW_BKT[BKT$B_NXTRECID] = .ID
;  2133     2192  3          END;
;  2134     2193  3
;  2135     2194  3      ! Now set up NEW_BKT and REC_ADDR to point to the buffer and position of
;  2136     2195  3      ! insert for the new record.
;  2137     2196  3      !
;  2138     2197  3      IF .IRAB[IRB$V_REC_W_LO]
;  2139     2198  3      THEN
;  2140     2199  3
;  2141     2200  3          ! Since IRAB[IRB$V_REC_W_LO] is set, the new record will be inserted
;  2142     2201  3          ! into the old bucket.
;  2143     2202  3          !
;  2144     2203  4          BEGIN
;  2145     2204  4          NEW_BKT = .BBLOCK[.IRAB[IRB$L_CURBDB], BDB$L_ADDR];
;  2146     2205  4
```

```
;  2147          2206  4          ! If this is an index bucket split, and the point of insertion is
;  2148          2207  4          ! exactly the same as the split point, then two calls to RM$INS_REC
;  2149          2208  4          ! are required. The first pass will be to update the contents of the
;  2150          2209  4          ! (left) index bucket, as are required by the particular split case.
;  2151          2210  4          ! The second call will be to update the contents of the new (right)
;  2152          2211  4          ! index bucket as is required by the particular split case.
;  2153          2212  4          !
;  2154          2213  5          IF  (.NEW_BKT[BKT$B_LEVEL] NEQ 0)
;  2155          2214  4              AND
;  2156          2215  5              (.IRAB[IRB$W_POS_INS] + .NEW_BKT EQLA .REC_ADDR)
;  2157          2216  4          THEN
;  2158          2217  5              BEGIN
;  2159          2218  5              GLOBAL REGISTER
;  2160          2219  5                  R_BKT_ADDR;
;  2161          2220  5
;  2162          2221  5              IRAB[IRB$V_SPL_IDX] = 1;
;  2163          2222  5
;  2164          2223  5              ! First, the updating of the old (left) index bucket is done. This
;  2165          2224  5              ! will involve the insertion of a new key as the high key in the
;  2166          2225  5              ! bucket with its downpointer (pointing to the leftmost data
;  2167          2226  5              ! bucket) left unchanged. However, if a two-pass two-bucket with
;  2168          2227  5              ! empty bucket split had taken place, then as there is no need to
;  2169          2228  5              ! update the old (left) index bucket, nothing is done.
;  2170          2229  5              !
;  2171          2230  5              ! There is one special case which is entirely handled by the
;  2172          2231  5              ! lower level routines. If only two keys fit in an index
;  2173          2232  5              ! bucket, a multibucket split occurs, and both keys proceed
;  2174          2233  5              ! the lower order key, then both new keys must be put in
;  2175          2234  5              ! the old bucket instead of putting one in the old and the
;  2176          2235  5              ! other in the new index bucket.
;  2177          2236  5              !
;  2178          2237  6              IF NOT (.IRAB[IRB$V_EMPTY_BKT]
;  2179          2238  6                      AND
;  2180          2239  6                     NOT .IRAB[IRB$V_BIG_SPLIT])
;  2181          2240  5              THEN
;  2182          2241  6                  BEGIN
;  2183          2242  6                  BKT_ADDR = .NEW_BKT;
;  2184          2243  6                  IF NOT RM$INS_REC(.NEW_BKT, RM$RECORD_SIZE())
;  2185          2244  6                  THEN
;  2186          2245  6                      BUG_CHECK;
;  2187          2246  5                  END;
;  2188          2247  5
;  2189          2248  5              ! Finally, update the new (right) index bucket. This is done
;  2190          2249  5              ! by clearing IRAB[IRB$L_VBN_LEFT] and IRAB[IRB$V_SPL_IDX]
;  2191          2250  5              ! as a signal that only a VBN should be swung, and in the
;  2192          2251  5              ! case of a multi-bucket non-empty bucket split, only a single
;  2193          2252  5              ! key and VBN pair should be inserted into the new bucket.
;  2194          2253  5              !
;  2195          2254  5              ! If this is the two-pass two-bucket empty bucket split case then
;  2196          2255  5              ! this will be RMS's first and only call to RM$INS_REC. In such a
;  2197          2256  5              ! case we do not clear IRB$V_VBN_LEFT since we still have to
;  2198          2257  5              ! compare its contents to the downpointer of the first record
;  2199          2258  5              ! in the new (right) index bucket.
;  2200          2259  5              !
;  2201          2260  6              IF NOT (.IRAB[IRB$V_EMPTY_BKT]
;  2202          2261  6                      AND
;  2203          2262  6                     NOT .IRAB[IRB$V_BIG_SPLIT])
```

```
;  2204     2263  5              THEN
;  2205     2264  5                  IRAB[IRB$L_VBN_LEFT] = 0;
;  2206     2265  5
;  2207     2266  5              IRAB[IRB$V_SPL_IDX] = 0;
;  2208     2267  5              IRAB[IRB$L_REC_COUNT] = 0;
;  2209     2268  5              NEW_BKT = .BBLOCK[.IRAB[IRB$L_NXTBDB], BDB$L_ADDR];
;  2210     2269  5
;  2211     2270  5              ! If this is a multi-bucket split then we may have to insert a
;  2212     2271  5              ! new index record into the new (right) index bucket. This will
;  2213     2272  5              ! require adjustment of several IRAB cells for the second call
;  2214     2273  5              ! to RM$INS_REC.
;  2215     2274  5              !
;  2216     2275  5              IF .IRAB[IRB$V_BIG_SPLIT]
;  2217     2276  5              THEN
;  2218     2277  5
;  2219     2278  5                  ! If this is the special multi-bucket split case which requires
;  2220     2279  5                  ! both new keys to go in the old (left) bucket, or if this is
;  2221     2280  5                  ! the two-pass multi-bucket with empty bucket split case, then
;  2222     2281  5                  ! we just clear IRAB[IRB$L_VBN_MID] as a signal that just the
;  2223     2282  5                  ! VBN pointer need be swung.
;  2224     2283  5                  !
;  2225     2284  7                  IF ((.IRAB[IRB$W_POS_INS] EQLA BKT$C_OVERHDSZ)
;  2226     2285  6                      OR
;  2227     2286  6                      .IRAB[IRB$V_EMPTY_BKT])
;  2228     2287  5                  THEN
;  2229     2288  5                      IRAB[IRB$L_VBN_MID] = 0
;  2230     2289  5
;  2231     2290  5                  ! If there is a new index record to be inserted into the new
;  2232     2291  5                  ! index bucket during the second call to RM$INS_REC, then
;  2233     2292  5                  ! adjust the pointer to the last noncompressed key to be the
;  2234     2293  5                  ! first record in the new (right) index bucket.
;  2235     2294  5                  !
;  2236     2295  5                  ELSE
;  2237     2296  5                      IF .IDX_DFN[IDX$V_IDX_COMPR]
;  2238     2297  5                      THEN
;  2239     2298  5                          IRAB[IRB$L_LST_NCMP] = .NEW_BKT + BKT$C_OVERHDSZ;
;  2240     2299  5
;  2241     2300  5                  ! Make sure that the position of insertion of the new index records
;  2242     2301  5                  ! (or position of replacement of the new VBNs if the VBN is just
;  2243     2302  5                  ! to be swung) is at the first record of the new (right) index
;  2244     2303  5                  ! bucket.
;  2245     2304  5                  !
;  2246     2305  5                  IRAB[IRB$W_POS_INS] = BKT$C_OVERHDSZ;
;  2247     2306  4                  END;                                ! Of BKT_ADDR definition
;  2248     2307  4
;  2249     2308  4              END
;  2250     2309  3          ELSE
;  2251     2310  3
;  2252     2311  3              ! If the record goes into the new bucket then adjust the offsets so
;  2253     2312  3              ! that things come out right later.
;  2254     2313  3              !
;  2255     2314  4              BEGIN
;  2256     2315  4
;  2257     2316  4              ! The only time it is not necessary to adjust the variable containing
;  2258     2317  4              ! the address of the preceeding record by the number of bytes the first
;  2259     2318  4              ! record in the new bucket was front compressed is when the new record
;  2260     2319  4              ! is a duplicate SIDR (ie the SIDR itself is the preceeding record),
```

```
;  2261     2320  4          ! SIDR key compresion is enabled, and the SIDR occupies the first
;  2262     2321  4          ! record position in the new bucket.
;  2263     2322  4          !
;  2264     2323  5          IF NOT (.IRAB[IRB$V_DUPS_SEEN]
;  2265     2324  5                      AND
;  2266     2325  5                     .IDX_DFN[IDX$V_KEY_COMPR]
;  2267     2326  5                      AND
;  2268     2327  5                     .IRAB[IRB$L_LST_REC] EQLA .IRAB[IRB$W_SPLIT])
;  2269     2328  4          THEN
;  2270     2329  4              IRAB[IRB$L_LST_REC] = .IRAB[IRB$L_LST_REC] - .COMPRESSION;
;  2271     2330  4
;  2272     2331  4          IRAB[IRB$W_POS_INS] = .IRAB[IRB$W_POS_INS] - .IRAB[IRB$W_SPLIT]
;  2273     2332  4                                                     + BKT$C_OVERHDSZ
;  2274     2333  4                                                     - .COMPRESSION;
;  2275     2334  4
;  2276     2335  4          IRAB[IRB$L_LST_REC] = .IRAB[IRB$L_LST_REC] - .IRAB[IRB$W_SPLIT]
;  2277     2336  4                                                     + BKT$C_OVERHDSZ;
;  2278     2337  4
;  2279     2338  4          IF .IFAB [IFB$B_PLG_VER] GEQU PLG$C_VER_3
;  2280     2339  4          THEN
;  2281     2340  5              BEGIN
;  2282     2341  5              IRAB [IRB$L_REC_COUNT] = .IRAB [IRB$L_REC_COUNT]
;  2283     2342  5                                     - .IRAB [IRB$L_SPL_COUNT];
;  2284     2343  5
;  2285     2344  5              ! If the record to be inserted is to go in the new bucket,
;  2286     2345  5              ! and the last noncompressed key remains in the old bucket,
;  2287     2346  5              ! or would follow the new record to be inserted, change this
;  2288     2347  5              ! pointer so that it points to the first record in the new bucket.
;  2289     2348  5              !
;  2290     2349  7              IF ((.NEW_BKT[BKT$B_LEVEL] EQLU 0
;  2291     2350  7                                         AND
;  2292     2351  7                                        .IDX_DFN[IDX$V_KEY_COMPR])
;  2293     2352  6                                  OR
;  2294     2353  7                                 (.NEW_BKT[BKT$B_LEVEL] NEQU 0
;  2295     2354  7                                         AND
;  2296     2355  6                                        .IDX_DFN[IDX$V_IDX_COMPR]))
;  2297     2356  5                 AND
;  2298     2357  6                NOT (.IRAB[IRB$L_LST_NCMP] GEQU .NEW_BKT
;  2299     2358  6                                  AND
;  2300     2359  6                                 .IRAB[IRB$L_LST_NCMP] LSSU .NEW_BKT + .IRAB[IRB$W_POS_INS])
;  2301     2360  5              THEN
;  2302     2361  5                  IRAB[IRB$L_LST_NCMP] = .NEW_BKT + BKT$C_OVERHDSZ;
;  2303     2362  4              END;
;  2304     2363  4
;  2305     2364  3              END;
;  2306     2365  3
;  2307     2366  3          IRAB[IRB$L_LST_REC] = .IRAB[IRB$L_LST_REC] + .NEW_BKT;
;  2308     2367  3          REC_ADDR = .IRAB[IRB$W_POS_INS] + .NEW_BKT;
;  2309     2368  3
;  2310     2369  3          ! Call spread and insert appropriate record routine. The only possible error
;  2311     2370  3          ! from this routine is a scarcity of ID's in the case of prologue 1 or 2
;  2312     2371  3          ! and that is a bug here.
;  2313     2372  3          !
;  2314     2373  4          BEGIN
;  2315     2374  4
;  2316     2375  4          GLOBAL REGISTER
;  2317     2376  4              R_BKT_ADDR;
```

```
;  2318          2377  4
;  2319          2378  4     BKT_ADDR = .NEW_BKT;
;  2320          2379  4
;  2321          2380  4     IF NOT RM$INS_REC(.NEW_BKT, RM$RECORD_SIZE())
;  2322          2381  4     THEN
;  2323          2382  4         BUG_CHECK;
;  2324          2383  4
;  2325          2384  3     END; ! Of bkt_addr definition
;  2326          2385  3
;  2327          2386  2     END;
;  2328          2387  2
;  2329          2388  2     ! Position to last record in left bucket and put key into keybuffer 2
;  2330          2389  2     ! for index update.
;  2331          2390  2     !
;  2332          2391  2     RM$EXT_HIGH_KEY();
;  2333          2392  3     RMSSUC (SUC)
;  2334          2393  1     END;


                                                .EXTRN  RM$BUG3

                                3C  BB 00000 RM$SPLIT_EM::
                                             PUSHR   #^M<R2,R3,R4,R5>                ; 1808
                             5E      18  C2 00002        SUBL2   #24, SP
                             50  20  A9  D0 00005        MOVL    32(IRAB), R0            ; 1894
                             55  18  A0  D0 00009        MOVL    24(R0), BKT_ADDR
                             52  04  A5  3C 0000D        MOVZWL  4(BKT_ADDR), EOB        ; 1895
                             52      55  C0 00011        ADDL2   BKT_ADDR, EOB
                             53      56  D0 00014        MOVL    REC_ADDR, INS_PNT       ; 1899
                             51      53  D0 00017        MOVL    INS_PNT, R1             ; 1901
                                 FD7E    30 0001A        BSBW    RM$COMP_SPL_PNT
                             56          50  D0 0001D    MOVL    R0, REC_ADDR
                             53          56  D1 00020    CMPL    REC_ADDR, INS_PNT       ; 1906
                                         36  12 00023    BNEQ    3$
                             50      44  A9  9E 00025    MOVAB   68(IRAB), R0            ; 1908
                         2E              60  03 E0 00029    BBS     #3, (R0), 3$
                                     0C  A5  95 0002D    TSTB    12(BKT_ADDR)            ; 1910
                                         29  12 00030    BNEQ    3$
                                         60  95 00032    TSTB    (R0)                    ; 1912
                                         25  18 00034    BGEQ    3$
                             52          56  D1 00036    CMPL    REC_ADDR, EOB           ; 1918
                                         05  13 00039    BEQL    1$
                             60          08  88 0003B    BISB2   #8, (R0)                ; 1920
                                         1B  11 0003E    BRB     3$
                             51      0E  A5  9E 00040 1$: MOVAB   14(R5), R1              ; 1935
                             51      4C  A9  D1 00044    CMPL    76(IRAB), R1
                                         06  13 00048    BEQL    2$
                             56      4C  A9  D0 0004A    MOVL    76(IRAB), REC_ADDR      ; 1937
                                         0B  11 0004E    BRB     3$
                         4C  A9          53  D0 00050 2$: MOVL    INS_PNT, 76(IRAB)       ; 1950
                             60      80  8F  8A 00054    BICB2   #128, (R0)              ; 1951
                             60          10  88 00058    BISB2   #16, (R0)               ; 1952
                 48  A9              53      55  A3 0005B 3$: SUBW3   BKT_ADDR, INS_PNT, 72(IRAB)    ; 1959
                 4A  A9              56      55  A3 00060    SUBW3   BKT_ADDR, REC_ADDR, 74(IRAB)    ; 1960
                             04  A5      4A  A9  B0 00065    MOVW    74(IRAB), 4(BKT_ADDR)           ; 1965
                         4C  A9          55  C2 0006A        SUBL2   BKT_ADDR, 76(IRAB)              ; 1966
```

```
                                 0C   AE  D4 0006E          CLRL    COMPRESSION                          ; 1978
                        50       3C   A9  D0 00071          MOVL    60(IRAB), R0                         ; 1980
                        6E       18   A0  D0 00075          MOVL    24(R0), NEW_BKT                      ;
          50            6E           04  C1 00079          ADDL3   #4, NEW_BKT, R0                      ; 1986
                  04    AE           60  9E 0007D          MOVAB   (R0), 4(SP)                          ;
                        52           56  C2 00081          SUBL2   REC_ADDR, R2                         ;
                  04    BE           52  B0 00084          MOVW    R2, @4(SP)                           ;
                                     52  D5 00088          TSTL    R2                                   ;
                                     03  12 0008A          BNEQ    4$                                   ;
                                   00DB  31 0008C          BRW     13$                                  ;
                  14    AE           56  D0 0008F 4$:      MOVL    REC_ADDR, REC_FROM                   ; 1996
          53            6E           0E  C1 00093          ADDL3   #14, NEW_BKT, REC_TO                 ; 1997
                        52       04   BE  3C 00097          MOVZWL  @4(SP), LENGTH                       ; 1998
          50            6E           0C  C1 0009B          ADDL3   #12, NEW_BKT, R0                     ; 2008
                        51           60  9A 0009F          MOVZBL  (R0), REC_SIZE                       ;
                                     03  12 000A2          BNEQ    5$                                   ;
                        51           01  CE 000A4          MNEGL   #1, REC_SIZE                         ; 2010
                                  0000G  30 000A7 5$:      BSBW    RM$REC_OVHD                          ; 2012
                  08    AE           50  D0 000AA          MOVL    R0, RECORD_OVHD                      ;
          50            6E           0C  C1 000AE          ADDL3   #12, NEW_BKT, R0                     ; 2023
                                     60  95 000B2          TSTB    (R0)                                 ;
                                     05  12 000B4          BNEQ    6$                                   ;
          10      1C    A7           06  E0 000B6          BBS     #6, 28(IDX_DFN), 8$                  ; 2025
          50            6E           0C  C1 000BB 6$:      ADDL3   #12, NEW_BKT, R0                     ; 2027
                                     60  95 000BF          TSTB    (R0)                                 ;
                                     05  13 000C1          BEQL    7$                                   ;
          03      1C    A7           03  E0 000C3          BBS     #3, 28(IDX_DFN), 8$                  ; 2029
                                   0096  31 000C8 7$:      BRW     12$                                  ;
                                 08 BE46  9F 000CB 8$:     PUSHAB  @RECORD_OVHD[REC_ADDR]               ; 2031
                FF00    8F           9E  B3 000CF          BITW    @(SP)+, #65280                       ;
                                     F2  13 000D4          BEQL    7$                                   ;
                                 08   AE  D5 000D6          TSTL    RECORD_OVHD                          ; 2052
                                     14  13 000D9          BEQL    9$                                   ;
                                     53  DD 000DB          PUSHL   REC_TO                               ; 2055
                                     56  DD 000DD          PUSHL   REC_ADDR                             ;
                                 10   AE  DD 000DF          PUSHL   RECORD_OVHD                          ;
                                  0000G  30 000E2          BSBW    RM$MOVE                              ;
                        5E           0C  C0 000E5          ADDL2   #12, SP                              ;
                        53           50  D0 000E8          MOVL    R0, REC_TO                           ;
                        56       08   AE  C0 000EB          ADDL2   RECORD_OVHD, REC_ADDR                ; 2056
                        5C           01  D0 000EF 9$:      MOVL    #1, AP                               ; 2065
                                 02   A3  9F 000F2          PUSHAB  2(REC_TO)                            ; 2066
                                  0000G  30 000F5          BSBW    RM$RECORD_KEY                        ;
                        5E           04  C0 000F8          ADDL2   #4, SP                               ;
                  10    AE           56  D0 000FB          MOVL    REC_ADDR, SAVE_REC_ADDR              ; 2072
          56            53       08   AE  C3 000FF          SUBL3   RECORD_OVHD, REC_TO, REC_ADDR        ; 2073
                        54     0098   C9  D0 00104          MOVL    152(IRAB), SAVE_LST_NCMP             ; 2075
                0098    C9           56  D0 00109          MOVL    REC_ADDR, 152(IRAB)                  ; 2076
                        55           6E  D0 0010E          MOVL    NEW_BKT, BKT_ADDR                    ; 2078
                        50           53  D0 00111          MOVL    REC_TO, R0                           ; 2080
                                  0000G  30 00114          BSBW    RM$COMPRESS_KEY                      ;
                0098    C9           54  D0 00117          MOVL    SAVE_LST_NCMP, 152(IRAB)             ; 2082
                        56       10   AE  D0 0011C          MOVL    SAVE_REC_ADDR, REC_ADDR              ; 2083
                        50           66  9A 00120          MOVZBL  (REC_ADDR), R0                       ; 2091
                        51           63  9A 00123          MOVZBL  (REC_TO), R1                         ;
      0C    AE        50           51  C3 00126          SUBL3   R1, R0, COMPRESSION                  ;
                  04    BE       0C   AE  A2 0012B          SUBW2   COMPRESSION, @4(SP)                  ; 2093
```

```
        51          6E          0C  C1 00130        ADDL3   #12, NEW_BKT, R1                    ; 2102
                                61  95 00134        TSTB    (R1)                                ;
                                05  12 00136        BNEQ    10$                                 ;
                FE  A3      0C  AE  A2 00138        SUBW2   COMPRESSION, -2(REC_TO)             ; 2105
                4A  A9      48  A9  B1 0013D 10$:   CMPW    72(IRAB), 74(IRAB)                  ; 2118
                                03  12 00142        BNEQ    11$                                 ;
                            0C  AE  D4 00144        CLRL    COMPRESSION                         ; 2120
        51          52      08  AE  C3 00147 11$:   SUBL3   RECORD_OVHD, LENGTH, R1             ; 2129
                    51          50  C2 0014C        SUBL2   R0, R1                              ;
                    52      FE  A1  9E 0014F        MOVAB   -2(R1), LENGTH                      ;
                14  AE      02 A046 9E 00153        MOVAB   2(R0)[REC_ADDR], REC_FROM           ; 2130
                    50          63  9A 00159        MOVZBL  (REC_TO), R0                        ; 2131
                    53      02 A043 9E 0015C        MOVAB   2(R0)[REC_TO], REC_TO               ;
                                52  D5 00161 12$:   TSTL    LENGTH                              ; 2141
                                05  15 00163        BLEQ    13$                                 ;
        63      14  BE          52  28 00165        MOVC3   LENGTH, @REC_FROM, (REC_TO)         ; 2143
                04  BE          0E  A0 0016A 13$:   ADDW2   #14, @4(SP)                         ; 2146
                    03    00B7  CA  91 0016E        CMPB    183(IFAB), #3                       ; 2151
                                10  1F 00173        BLSSU   14$                                 ;
        50          6E          0C  C1 00175        ADDL3   #12, NEW_BKT, R0                    ; 2153
                                60  95 00179        TSTB    (R0)                                ;
                                08  13 0017B        BEQL    14$                                 ;
                                6E  DD 0017D        PUSHL   NEW_BKT                             ; 2155
                              FE10  30 0017F        BSBW    RM$MOVE_VBNS                        ;
                    5E          04  C0 00182        ADDL2   #4, SP                              ;
        50          6E          0C  C1 00185 14$:   ADDL3   #12, NEW_BKT, R0                    ; 2162
                                60  95 00189        TSTB    (R0)                                ;
                                2C  12 0018B        BNEQ    17$                                 ;
                    03    00B7  CA  91 0018D        CMPB    183(IFAB), #3                       ; 2164
                                25  1E 00192        BGEQU   17$                                 ;
                    52          01  D0 00194        MOVL    #1, ID                              ; 2172
        56          6E          0E  C1 00197        ADDL3   #14, NEW_BKT, REC_ADDR              ; 2173
                    53      04  BE  3C 0019B        MOVZWL  @4(SP), EOB                         ; 2174
                    53          6E  C0 0019F        ADDL2   NEW_BKT, EOB                        ;
                    53          56  D1 001A2 15$:   CMPL    REC_ADDR, EOB                       ; 2178
                                0B  1E 001A5        BGEQU   16$                                 ;
                01  A6          52  90 001A7        MOVB    ID, 1(REC_ADDR)                     ; 2184
                                52  D6 001AB        INCL    ID                                  ; 2185
                              0000G 30 001AD        BSBW    RM$GETNEXT_REC                      ; 2186
                                F0  11 001B0        BRB     15$                                 ; 2178
        50          6E          06  C1 001B2 16$:   ADDL3   #6, NEW_BKT, R0                     ; 2191
                    60          52  90 001B6        MOVB    ID, (R0)                            ;
                    51      44  A9  9E 001B9 17$:   MOVAB   68(IRAB), R1                       ; 2197
        03          61          03  E0 001BD        BBS     #3, (R1), 18$                       ;
                              0082  31 001C1        BRW     27$                                 ;
                    50      20  A9  D0 001C4 18$:   MOVL    32(IRAB), R0                        ; 2204
                    6E      18  A0  D0 001C8        MOVL    24(R0), NEW_BKT                     ;
        50          6E          0C  C1 001CC        ADDL3   #12, NEW_BKT, R0                    ; 2213
                                60  95 001D0        TSTB    (R0)                                ;
                                70  13 001D2        BEQL    26$                                 ;
                    50      48  A9  3C 001D4        MOVZWL  72(IRAB), R0                        ; 2215
                    50          6E  C0 001D8        ADDL2   NEW_BKT, R0                         ;
                    56          50  D1 001DB        CMPL    R0, REC_ADDR                        ;
                                64  12 001DE        BNEQ    26$                                 ;
                    61          01  88 001E0        BISB2   #1, (R1)                            ; 2221
        04          61          06  E1 001E3        BBC     #6, (R1), 19$                       ; 2237
        17          61          02  E1 001E7        BBC     #2, (R1), 20$                       ; 2239
```

```
                                                
```

```
                0098   C9          6E          0E  C1 002BC 32$:    ADDL3   #14, NEW_BKT, 152(IRAB)                  ; 2361
                                4C  A9          6E  C0 002C2 33$:    ADDL2   NEW_BKT, 76(IRAB)                        ; 2366
                                    56      48  A9  3C 002C6        MOVZWL  72(IRAB), REC_ADDR                       ; 2367
                                    56          6E  C0 002CA        ADDL2   NEW_BKT, REC_ADDR                        ;
                                    55          6E  D0 002CD        MOVL    NEW_BKT, BKT_ADDR                        ; 2378
                                              0000G 30 002D0        BSBW    RM$RECORD_SIZE                           ; 2380
                                                50  DD 002D3        PUSHL   R0                                       ;
                                            04  AE  DD 002D5        PUSHL   NEW_BKT                                  ;
                                              0000G 30 002D8        BSBW    RM$INS_REC                               ;
                                    5E          08  C0 002DB        ADDL2   #8, SP                                   ;
                                    03          50  E8 002DE        BLBS    R0, 34$                                  ;
                                              0000G 30 002E1        BSBW    RM$BUG3                                  ; 2381
                                              FC0A  30 002E4 34$:   BSBW    RM$EXT_HIGH_KEY                          ; 2391
                                    50          01  D0 002E7        MOVL    #1, R0                                   ; 2393
                                    5E          18  C0 002EA        ADDL2   #24, SP                                  ;
                                                3C  BA 002ED        POPR    #^M<R2,R3,R4,R5>                         ;
                                                    05 002EF        RSB                                              ;

; Routine Size:  752 bytes,    Routine Base:  RM$RMS3 + 0265


; 2335          2394  1
; 2336          2395  1 END
; 2337          2396  1
; 2338          2397  0 ELUDOM




;                                   PSECT SUMMARY
;
;       Name                    Bytes                       Attributes
;
;  RM$RMS3                         1365  NOVEC,NOWRT,  RD ,  EXE,NOSHR,  GBL,  REL,  CON,  PIC,ALIGN(2)


;                               Library Statistics
;
;                                          -------- Symbols --------      Pages       Processing
;       File                               Total    Loaded   Percent      Mapped      Time
;
;  _$255$DUA28:[RMS.OBJ]RMS.L32;1           3109        64         2       154         00:00.4




;                                   COMMAND QUALIFIERS

;       BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS$:RM3SIDXSP/OBJ=OBJ$:RM3SIDXSP MSRC$:RM3SIDXSP/UPDATE=(ENH$:RM3SIDXSP)
```

```
; Size:          1365 code + 0 data bytes
; Run Time:        00:38.2
; Elapsed Time:    01:16.7
; Lines/CPU Min:    3760
; Lexemes/CPU-Min: 13639
; Memory Used:  268 pages
; Compilation Complete
```

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

RM3PROBE
LIS

RM3SIOXSP
LIS

RM3PUTERR
LIS

RM3PUTUPD
LIS

RM3SPLUDR
LIS

RM3RRV
LIS

RM3ROOT
LIS

RM3PUT
LIS