```
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  PPPPPPPP    LL          UU      UU  DDDDDDDD    RRRRRRRR
RRRRRRRR    MM      MM    333333      SSSSSSSS  PPPPPPPP    LL          UU      UU  DDDDDDDD    RRRRRRRR
RR      RR  MMMM  MMMM  33      33  SS          PP      PP  LL          UU      UU  DD      DD  RR      RR
RR      RR  MMMM  MMMM  33      33  SS          PP      PP  LL          UU      UU  DD      DD  RR      RR
RR      RR  MM  MM  MM        33    SS          PP      PP  LL          UU      UU  DD      DD  RR      RR
RR      RR  MM  MM  MM        33    SS          PP      PP  LL          UU      UU  DD      DD  RR      RR
RRRRRRRR    MM      MM      33      SSSSSS      PPPPPPPP    LL          UU      UU  DD      DD  RRRRRRRR
RRRRRRRR    MM      MM      33      SSSSSS      PPPPPPPP    LL          UU      UU  DD      DD  RRRRRRRR
RR   RR     MM      MM        33          SS  PP          LL          UU      UU  DD      DD  RR   RR
RR   RR     MM      MM        33          SS  PP          LL          UU      UU  DD      DD  RR   RR
RR    RR    MM      MM  33    33          SS  PP          LL          UU      UU  DD      DD  RR    RR
RR    RR    MM      MM  33    33          SS  PP          LL          UU      UU  DD      DD  RR    RR
RR      RR  MM      MM    333333    SSSSSSSS  PP          LLLLLLLLLL  UUUUUUUUUU  DDDDDDDD    RR      RR
RR      RR  MM      MM    333333    SSSSSSSS  PP          LLLLLLLLLL  UUUUUUUUUU  DDDDDDDD    RR      RR

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 RM3SPLUDR (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 user data record buckets
;    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:         Wendy Koenig     CREATION DATE:                5-JUL-78  14:46
;    45          0045  1 !
;    46          0046  1 !
;    47          0047  1 ! MODIFIED BY:
;    48          0048  1 !
;    49          0049  1 !       V03-013 JWT0157         Jim Teague                23-Feb-1984
;    50          0050  1 !               When RMS attempted to calculate whether a series of
;    51          0051  1 !               duplicate records (including the new record) would
;    52          0052  1 !               fit within a single bucket, it neglected to account
;    53          0053  1 !               for the fact that the first record in the chain will
;    54          0054  1 !               undergo full expansion when it ends up as the first
;    55          0055  1 !               record in the new bucket.  If it is currently partially
;    56          0056  1 !               compressed based on the previous key, then that could
;    57          0057  1 !               (and sometimes DID) cause bucket overflow when the
```

```
;    58          0058  1 !                duplicate chain is moved merrily into the new bucket.
;    59          0059  1 !                Keep track of the compression count for the first
;    60          0060  1 !                record in the dup chain, and add it to the total
;    61          0061  1 !                size of the chain before comparing to bucket size.
;    62          0062  1 !
;    63          0063  1 !        V03-012 JWT0142         Jim Teague              16-Dec-1983
;    64          0064  1 !                Correct incorrect bucket VBN comparison.
;    65          0065  1 !
;    66          0066  1 !        V03-011 MCN0008         Maria del C. Nasr       22-Mar-1983
;    67          0067  1 !                More changes in the linkages
;    68          0068  1 !
;    69          0069  1 !        V03-010 MCN0007         Maria del C. Nasr       28-Feb-1983
;    70          0070  1 !                Reorganize linkages
;    71          0071  1 !
;    72          0072  1 !        V03-009 TMK0004         Todd M. Katz            10-Nov-1982
;    73          0073  1 !                At the present time, under certain circumstances, the number
;    74          0074  1 !                of RRVs which will be required to be created when a simple
;    75          0075  1 !                two-bucket split is done is being incorrectly calculated. This
;    76          0076  1 !                will happen only during $UPDATEs when the record being updated
;    77          0077  1 !                is to go into the old (left) bucket and prior to the split is
;    78          0078  1 !                in its original bucket. Even then it does not happen under all
;    79          0079  1 !                possible circumstances, but only when duplicate records are
;    80          0080  1 !                involved. It is possible that the number of RRVs calculated to
;    81          0081  1 !                be required will be several less than the actual number which
;    82          0082  1 !                will be needed. Under certain circumstances, the number of RRVs
;    83          0083  1 !                needed may actually be calculated as a negative number - an
;    84          0084  1 !                impossibility. Much depends upon the bucket composition. While
;    85          0085  1 !                this does not influence the actual creation of RRVs, what it
;    86          0086  1 !                does affect is where the bucket split point is calculated to be
;    87          0087  1 !                since RRVs to be created do take up space in the old (left)
;    88          0088  1 !                bucket. In fact, this problem came to my attention because of
;    89          0089  1 !                the occurrance of a bucket split which resulted in the right
;    90          0090  1 !                bucket, the new bucket, being empty, and the old (left) bucket
;    91          0091  1 !                containing all the records even though there was no room for
;    92          0092  1 !                them (or the bucket split would not have been required in the
;    93          0093  1 !                first place). This split was caused by the number of RRVs
;    94          0094  1 !                required being calculated as -1 instead of 0 such that, instead
;    95          0095  1 !                of having the RRV spacial requirements added to the left bucket
;    96          0096  1 !                size requirements, they were subtracted.
;    97          0097  1 !
;    98          0098  1 !                To fix this problem I have adjusted how the number of needed
;    99          0099  1 !                RRVs are to be calculated. To start, the number of needed RRVs
;   100          0100  1 !                is calculated to be the number of records (including the record
;   101          0101  1 !                being updated which is not currently in the bucket) whose
;   102          0102  1 !                original bucket is the bucket splitting. Then, as the split
;   103          0103  1 !                point of the bucket is adjusted from left to right, this number
;   104          0104  1 !                is decremented as records (which are in their orginal bucket)
;   105          0105  1 !                are designated to stay in the left or old bucket. This is where
;   106          0106  1 !                my change comes in. Previously, that the updated record was to
;   107          0107  1 !                stay in the old bucket was determined at several different
;   108          0108  1 !                points, and each time the count of the number of needed RRVs
;   109          0109  1 !                was decremented, as long as the other conditions were met.
;   110          0110  1 !                Unfortunately, this allowed for this determination to take place
;   111          0111  1 !                more than once, and for the RRV count to be decremented multiple
;   112          0112  1 !                times for the same record. My fix prevents this from occurring.
;   113          0113  1 !                While it is still determined in several places that the updated
;   114          0114  1 !                record is to go in the old bucket, I have made sure that those
```

```
;  115          0115  1 !                places are orthogonal to one another, so that the RRV count is
;  116          0116  1 !                not decremented more than once for the same record, the record
;  117          0117  1 !                whose update is causing the split.
;  118          0118  1 !
;  119          0119  1 !        V03-008 KBT0234         Keith B. Thompson       23-Aug-1982
;  120          0120  1 !                Reorganize psects
;  121          0121  1 !
;  122          0122  1 !        V03-007 TMK0003         Todd M. Katz            02-Jul-1982
;  123          0123  1 !                Implement RMS cluster solution for next record positioning.
;  124          0124  1 !                The next record positioning context is now kept in the IRAB,
;  125          0125  1 !                where it maybe retrieved from, instead of in the NRP list
;  126          0126  1 !                which has been eliminated. When refering to the RFA address of
;  127          0127  1 !                the new/changed primary data record use the subfields
;  128          0128  1 !                IRB$L_PUTUP_VBN and IRB$W_PUTUPD_ID.
;  129          0129  1 !
;  130          0130  1 !        V03-006 MCN0006         Maria del C. Nasr       29-Jun-1982
;  131          0131  1 !                Allow keys of different data types other than string
;  132          0132  1 !                in prologue 3 files.
;  133          0133  1 !                Change all CH$COMPARE calls to RM$COMPARE_KEY to compare
;  134          0134  1 !                keys taking into consideration the different data types.
;  135          0135  1 !
;  136          0136  1 !        V03-005 MCN0005         Maria del C. Nasr       11-Jun-1982
;  137          0137  1 !                Eliminate overhead at end of data bucket that was to be
;  138          0138  1 !                used for duplicate continuation bucket processing.
;  139          0139  1 !
;  140          0140  1 !        V03-004 TMK0002         Todd M. Katz            31-May-1982
;  141          0141  1 !                Performance enhancements. I have made four changes to the
;  142          0142  1 !                routine RM$SPLIT_UDR_3 which should cut done on the length of
;  143          0143  1 !                the bucket scans required at various times to re-expand keys.
;  144          0144  1 !                The enhancements involve setting the IRAB field IRB$L_LST_NCMP
;  145          0145  1 !                to the current record if key compression is enabled and the
;  146          0146  1 !                key of the current record is zero front compressed during
;  147          0147  1 !                various bucket scans required in the determination of the
;  148          0148  1 !                split point(s). The first of these scans is made to position
;  149          0149  1 !                to and extract the key of the last record in the bucket.
;  150          0150  1 !                The second and third scans are when the split code has decided
;  151          0151  1 !                that the best split point is one record previous to the current
;  152          0152  1 !                position, and must scan the bucket to obtain the record
;  153          0153  1 !                previous to that position so its key can be extracted and used
;  154          0154  1 !                during the index update. The forth scan is the right-to-left
;  155          0155  1 !                record-by-record scan made to decide whether a two-bucket
;  156          0156  1 !                split is possible, and if so, where is the best place to
;  157          0157  1 !                split. In all four cases, I have added code to set the
;  158          0158  1 !                last noncompressed record pointer before continuing the scan
;  159          0159  1 !                with the next record, if the current record was zero front
;  160          0160  1 !                compressed.
;  161          0161  1 !
;  162          0162  1 !        V03-003 TMK0001         Todd M. Katz            10-May-1982
;  163          0163  1 !                The algorithm for determining the split point of a prologue
;  164          0164  1 !                three data bucket with compressed keys first determines whether
;  165          0165  1 !                a two-bucket split can be done by scanning the old bucket from
;  166          0166  1 !                left-to-right record-by-record determining whether the lefthand
;  167          0167  1 !                sides and righthand sides of each possible split point will
;  168          0168  1 !                fit into a bucket. This size determination must take into
;  169          0169  1 !                account the position of insertion of the new (or updated)
;  170          0170  1 !                record, and the size determination of the righthand side must
;  171          0171  1 !                take into account the number of characters currently front
```

```
;  172          0172  1 !                compressed of what will become its low-order (and thus
;  173          0173  1 !                non-compressed key) record. What was missing, and what this
;  174          0174  1 !                change rectifies, is that what may become the low-order record
;  175          0175  1 !                of the righthand bucket is in fact the new (updated) record
;  176          0176  1 !                whose insertion is forcing this split to take place. In this
;  177          0177  1 !                case, the number of front compressed characters to be added to
;  178          0178  1 !                the righthand side total must come from the compressed key in
;  179          0179  1 !                keybuffer 5, if this is an $UPDATE, or from the compressed key
;  180          0180  1 !                in the record buffer whose address is stored in IRB$L_RECBUF,
;  181          0181  1 !                if this is a $PUT. This change will be included as a patch on
;  182          0182  1 !                the V3.1 update floppy.
;  183          0183  1 !
;  184          0184  1 !           V03-002 MCN0004          Maria del C. Nasr       31-Mar-1982
;  185          0185  1 !                   Do not count records that will not need rrv's when moved out
;  186          0186  1 !                   of the bucket.  Their id's cannot be recycled in plg 3 files.
;  187          0187  1 !
;  188          0188  1 !           V03-001 MCN0003          Maria del C. Nasr       25-Mar-1982
;  189          0189  1 !                   Use macro to calculate keybuffer address.
;  190          0190  1 !
;  191          0191  1 !           V02-016 DJD0001          Darrell Duffy           1-March-1982
;  192          0192  1 !                   Fix references to RBF for better probing
;  193          0193  1 !
;  194          0194  1 !           V02-015 MCN0002          Maria del C. Nasr       09-Jul-1981
;  195          0195  1 !                   Fix a problem with update of the first record in a duplicate
;  196          0196  1 !                   chain, in both old code, and new code.  Also fix problem in
;  197          0197  1 !                   new code with non-compressed keys.
;  198          0198  1 !
;  199          0199  1 !           V02-014 MCN0001          Maria del C. Nasr       02-Jun-1981
;  200          0200  1 !                   Add the routine to split prologue 3 data buckets.
;  201          0201  1 !
;  202          0202  1 !           V02-013 REFORMAT         Ron Schaefer            23-Jul-1980  14:10
;  203          0203  1 !                   Reformat the source
;  204          0204  1 !
;  205          0205  1 !           V02-012 CDS0000          Christian Saether,      01-Jan-1980  15:00
;  206          0206  1 !                   FIX PROBLEM WHEN SPLITTING BECAUSE OUT OF ID'S.
;  207          0207  1 !
;  208          0208  1 ! REVISION HISTORY:
;  209          0209  1 !
;  210          0210  1 !   Wendy Koenig,       18-SEP-78  16:53
;  211          0211  1 !   X0002 - FIX BUG IN BACKING UP PAST NEW RECORD
;  212          0212  1 !
;  213          0213  1 !   Wendy Koenig,       19-SEP-78  10:52
;  214          0214  1 !   X0003 - DO SPLIT AT POINT OF INSERT IF ASCENDING ORDER DETECTED
;  215          0215  1 !
;  216          0216  1 !   Wendy Koenig,       12-OCT-78  13:21
;  217          0217  1 !   X0004 - CHANGES FOR UPDATE
;  218          0218  1 !
;  219          0219  1 !   Wendy Koenig,       18-OCT-78  14:03
;  220          0220  1 !   X0005 - IF WE PASS BY POS_INSERT WHILE SKIPPING OVER DUPS, NOTE IT
;  221          0221  1 !
;  222          0222  1 !   Wendy Koenig,       18-OCT-78  14:37
;  223          0223  1 !   X0006 - FIX SOME PROBLEMS W/ 4-BKT SPLIT ($UPDATE ONLY)
;  224          0224  1 !
;  225          0225  1 !   Wendy Koenig,       24-OCT-78  14:03
;  226          0226  1 !   X0007 - MAKE CHANGES CAUSED BY SHARING CONVENTIONS
;  227          0227  1 !
;  228          0228  1 !   Wendy Koenig,       7-NOV-78  8:58
```

```
;  229     0229  1 !   X0008 - FIX EMPTY_BKT BUG, NOT BEING SET WHEN SHOULD BE
;  230     0230  1 !
;  231     0231  1 !   Wendy Koenig,       22-JAN-79  17:03
;  232     0232  1 !   X0009 - IF LOA TRIES TO FORCE US TO SPLIT ALL DUPS, SPLIT AT POS_INS
;  233     0233  1 !
;  234     0234  1 !   Wendy Koenig,       24-JAN-79  9:51
;  235     0235  1 !   X0010 - CONDITION HOLDS EVEN IF LOA NOT SET
;  236     0236  1 !
;  237     0237  1 !   Wendy Koenig,       29-JAN-79  15:58
;  238     0238  1 !   X0011 - FIX PROBLEM W/ DUPLICATE ENTRIES IN INDEX
;  239     0239  1 !
;  240     0240  1 !*****
;  241     0241  1
;  242     0242  1 LIBRARY 'RMSLIB:RMS';
;  243     0243  1
;  244     0244  1 REQUIRE 'RMSSRC:RMSIDXDEF';
;  245     0309  1
;  246     0310  1 ! define default psects for code
;  247     0311  1 !
;  248     0312  1 PSECT
;  249     0313  1     CODE = RM$RMS3(PSECT_ATTR),
;  250     0314  1     PLIT = RM$RMS3(PSECT_ATTR);
;  251     0315  1
;  252     0316  1 ! Linkages
;  253     0317  1 !
;  254     0318  1 LINKAGE
;  255     0319  1     L_COMPARE_KEY,
;  256     0320  1     L_PRESERVE1,
;  257     0321  1     L_RABREG_4567,
;  258     0322  1     L_RABREG_67,
;  259     0323  1     L_REC_OVHD,
;  260     0324  1
;  261     0325  1 ! Local linkages
;  262     0326  1 !
;  263     0327  1     RL$BUILD_KEY        = JSB () :
;  264     0328  1                         GLOBAL (R_IDX_DFN) PRESERVE(1,2,3,4,5),
;  265     0329  1     RL$MOVE_KEY         = JSB (REGISTER = 0, REGISTER = 6) :
;  266     0330  1                         GLOBAL (R_RAB, R_IRAB, R_IFAB, R_IDX_DFN, R_BKT_ADDR);
;  267     0331  1
;  268     0332  1 ! Forward Routine
;  269     0333  1 !
;  270     0334  1
;  271     0335  1 FORWARD ROUTINE
;  272     0336  1     RM$BUILD_KEY        : RL$BUILD_KEY NOVALUE,
;  273     0337  1     RM$MOVE_KEY         : RL$MOVE_KEY NOVALUE;
;  274     0338  1
;  275     0339  1 ! External Routines
;  276     0340  1 !
;  277     0341  1
;  278     0342  1 EXTERNAL ROUTINE
;  279     0343  1     RM$MOVE             : RL$PRESERVE1,
;  280     0344  1     RM$RECORD_VBN       : RL$PRESERVE1,
;  281     0345  1     RM$RECORD_KEY       : RL$PRESERVE1,
;  282     0346  1     RM$REC_OVHD         : RL$REC_OVHD,
;  283     0347  1     RM$VBN_SIZE         : RL$PRESERVE1,
;  284     0348  1     RM$COMPARE_KEY      : RL$COMPARE_KEY,
;  285     0349  1     RM$COMPARE_REC      : RL$RABREG_67,
```

```
;  286          0350  1     RM$GETNEXT_REC      : RL$RABREG_67;
```

```
;  288          0351  1 !++
;  289          0352  1 !       ALGORITHM FOR A TWO-BUCKET 50/50 SPLIT
;  290          0353  1 !
;  291          0354  1 !       GIVEN: that the record will not fit in the bucket.
;  292          0355  1 !       i.e., we must split the bucket in some form.
;  293          0356  1 !
;  294          0357  1 !       INPUTS: the bucket, the record size and the position
;  295          0358  1 !       to insert the record in the bucket
;  296          0359  1 !
;  297          0360  1 !
;  298          0361  1 !       GOALS: to make the split as efficient as possible:
;  299          0362  1 !         1)  to create the fewest number of new buckets possible
;  300          0363  1 !         2)  to use the space in the available buckets efficiently --
;  301          0364  1 !             i.e., the bucket with the most available space should contain
;  302          0365  1 !             the most data after the split.
;  303          0366  1 !
;  304          0367  1 ! ALGORITHM IN A NUTSHELL:
;  305          0368  1 !       1)  A two-bucket split will occur IF AND ONLY IF there is a point in
;  306          0369  1 !           the bucket at which all records to the left of the point and
;  307          0370  1 !           necessary rrv's fit in a single bucket and all records to the right
;  308          0371  1 !           of the point fit in a single bucket.  This point must be on a
;  309          0372  1 !           record boundary and must not be in the middle of a chain of
;  310          0373  1 !           duplicates.
;  311          0374  1 !       2)  Given that such a point exists, the most optimal point for a
;  312          0375  1 !           2-bucket split is the point at which the actual data records
;  313          0376  1 !           are divided evenly between the available space in the original
;  314          0377  1 !           bucket and the available space in the new (previously empty) bucket.
;  315          0378  1 !
;  316          0379  1 !       In theory, therefore, the idea is to find a point in the bucket such that
;  317          0380  1 !       the point is on a boundary between duplicate records and that
;  318          0381  1 !       1) records in the left hand side / space in the left hand bucket
;  319          0382  1 !                                =
;  320          0383  1 !       2) records in the right hand side / space in the right hand bucket.
;  321          0384  1 !       In practice, the idea is to mimimize the absolute difference between
;  322          0385  1 !       ratio 1) and ratio 2). Just to make it clearer, "records in the left
;  323          0386  1 !       hand side" means the total size of the data records left of this point
;  324          0387  1 !       (not including rrv's of any kind) and "space in the left hand bucket"
;  325          0388  1 !       means the bucketsize of the data bucket minus the total size of existing
;  326          0389  1 !       rrv's and the total size of rrv's which would have to be generated.
;  327          0390  1 !
;  328          0391  1 !
;  329          0392  1 ! IMPLEMENTATION:
;  330          0393  1 !       This algorithm needs two scans of the bucket.  The first scan is very
;  331          0394  1 !       quick and determines the total size of the existing rrv's.  It also
;  332          0395  1 !       counts the number of rrv's that would have to be generated in a worst
;  333          0396  1 !       case situation (i.e., all records would be moved out).  Thus, as the
;  334          0397  1 !       second scan proceeds, all information needed to calculate the above
;  335          0398  1 !       ratios EXACTLY is available.
;  336          0399  1 !
;  337          0400  1 !       In order for there to be a 2-bucket split, there must be a point
;  338          0401  1 !       in the bucket such that the right hand side fits in a single bucket.
;  339          0402  1 !       Scanning from the left (beginning) of the bucket, we can find the
;  340          0403  1 !       first point at which the right hand side will fit. Since as we
;  341          0404  1 !       continue scanning to the right we are decreasing the right hand side,
;  342          0405  1 !       the righthand side will continue to fit as we scan rightward.
;  343          0406  1 !
;  344          0407  1 !       If at this point, the left hand side will not fit, we can not possibly
```

```
;  345     0408  1 !       have a 2-bucket split, since continuing our scan would only make
;  346     0409  1 !       the left hand side larger ( or it may stay the same size). Once we
;  347     0410  1 !       have found a point at which we can do a 2-bucket split we can always
;  348     0411  1 !       return to it, if in our search for a more optimal split point we
;  349     0412  1 !       leave the range in which the left hand side will fit.
;  350     0413  1 !       This can occur if the records in the bucket are of miminal size,
;  351     0414  1 !       that is to say that the records are the same size as rrv's and
;  352     0415  1 !       therefore no additional space for data is gained by scanning to the
;  353     0416  1 !       right.
;  354     0417  1 !
;  355     0418  1 !       At this point ( the first point at which the right hand side will fit),
;  356     0419  1 !       ratio 1 is less than ratio 2. As we proceed to the right, ratio 1 will
;  357     0420  1 !       increase and ratio 2 will decrease. This is due to the fact that
;  358     0421  1 !       the size of the right hand size ( the numerator of ratio 2) decreases
;  359     0422  1 !       as we move rightward and the available space in the right bucket is a
;  360     0423  1 !       constant ( the denominator of ratio 2). In ratio 1, both the numerator
;  361     0424  1 !       and denominator are increasing, but the numerator is increasing at a
;  362     0425  1 !       faster rate. As soon as we reach a point where ratio 1 is greater
;  363     0426  1 !       than or equal to ratio 2, we can stop the scan.  Now we have a choice
;  364     0427  1 !       of split points available. We can use this point or the one immediately
;  365     0428  1 !       before it ( if such a point exists). The decision is made by
;  366     0429  1 !       minimizing the absolute difference between the ratios and we have an
;  367     0430  1 !       optimal split point.
;  368     0431  1 !
;  369     0432  1 !
;  370     0433  1 !       Things become complicated by the presence of duplicate records.
;  371     0434  1 !       When duplicate records occupy more than one bucket, the subsequent
;  372     0435  1 !       buckets are termed continuation buckets.  In prologue version 1 and
;  373     0436  1 !       2 files, there is a pointer from the index to the first bucket only,
;  374     0437  1 !       and the continuation buckets are found only from the horizontal links
;  375     0438  1 !       in the buckets.  At one point, it was thought that disaster would
;  376     0439  1 !       ensue if the continuation buckets ever had a record with a key value
;  377     0440  1 !       other than that of the duplicates.  Normally, this will not happen
;  378     0441  1 !       because the key value of the index pointer to the first bucket will
;  379     0442  1 !       be the same as that of the duplicate records in the chain and a record
;  380     0443  1 !       with a higher key value will follow the next index pointer down when
;  381     0444  1 !       positioning for insert.  This will place it in the next bucket beyond
;  382     0445  1 !       the chain of continuation buckets.  However, a bucket in which the
;  383     0446  1 !       record with the highest value has been deleted that subsequently
;  384     0447  1 !       recieves a series of duplicates creating a continuation chain will
;  385     0448  1 !       generate a situation where a record with a key value between that
;  386     0449  1 !       of the duplicate chain and the original high key value of the bucket
;  387     0450  1 !       will be inserted at the end of the duplicate chain.  A far more common
;  388     0451  1 !       situation is created by RMS-11 (at least thru v1.5) when loading a
;  389     0452  1 !       file in ascending primary key sequence will pack the buckets 100%
;  390     0453  1 !       (or the load factor) full, including records of non-dupe key values
;  391     0454  1 !       at the end of continuation buckets.
;  392     0455  1 !       At any rate, the fact that the situation exists notwithstanding,
;  393     0456  1 !       much of the code that follows is there to keep duplicates together
;  394     0457  1 !       when splitting, and to put only records with duplicate key values
;  395     0458  1 !       in continuation buckets.  It appears to be a good thing to do from
;  396     0459  1 !       an overall space efficiency standpoint over a period of time, but
;  397     0460  1 !       the code could probably be considerably simplified if it wasn't
;  398     0461  1 !       necessary.  With all that in mind, the split situation with all
;  399     0462  1 !       possible record 'partitions' within the bucket prior to splitting
;  400     0463  1 !       is as follows:
;  401     0464  1 !
```

```
;  402     0465  1 !       ---------------------------------------------
;  403     0466  1 !       ! low set ! low dupes !! high dupes ! high set !
;  404     0467  1 !       ---------------------- ----------------------
;  405     0468  1 !                              ^
;  406     0469  1 !                    point of insert (new record)
;  407     0470  1 !
;  408     0471  1 !       From the point of view of the split code, an update operation in
;  409     0472  1 !       which the record is growing and causes a split is identical (almost)
;  410     0473  1 !       to a new record being inserted.  The original record is removed from
;  411     0474  1 !       the bucket after determining that the updated record will cause a
;  412     0475  1 !       split and the updated record is more or less treated as a new record.
;  413     0476  1 !       One of the most important differences is that in an update situation,
;  414     0477  1 !       the 'new' record gets the id of the old record, rather than a new id.
;  415     0478  1 !       Another is that because duplicate records are always inserted at
;  416     0479  1 !       the end of a chain of duplicates, some split cases can only occur
;  417     0480  1 !       on an update operation.
;  418     0481  1 !       In fact, the situation postulated above can happen only in an update
;  419     0482  1 !       situation, and may cause 3 new buckets to be generated on the split
;  420     0483  1 !       operation.  This will occur when the updated record is in the
;  421     0484  1 !       middle of a group of duplicate records and grows to the extent that
;  422     0485  1 !       no other records will fit in the bucket with it anymore.
;  423     0486  1 !       Using 1 byte key values to make this easier to visualize, the bucket
;  424     0487  1 !       above prior to the update may look like this (the artificial
;  425     0488  1 !       partitioning of the bucket corresponds to the breakdown above):
;  426     0489  1 !
;  427     0490  1 !       ---------------------------------
;  428     0491  1 !       ! A B C ! D D D ! D ! D D ! E F G !
;  429     0492  1 !       ---------------- ---------------
;  430     0493  1 !                        /\
;  431     0494  1 !              this record gets updated
;  432     0495  1 !
;  433     0496  1 !       The record being updated changes size and grows such that it
;  434     0497  1 !       needs an entire bucket for itself.  To keep all the duplicates
;  435     0498  1 !       together, the situation after the split looks like this:
;  436     0499  1 !
;  437     0500  1 !       ---------------     -----     -------     ---------
;  438     0501  1 !       ! A B C D D D ! ->  ! D ! ->  ! D D ! ->  ! E F G !
;  439     0502  1 !       ---------------     -----     -------     ---------
;  440     0503  1 !       this is the         these two are
;  441     0504  1 !       original bucket     continuation buckets
;  442     0505  1 !
;  443     0506  1 !       The original bucket probably had an index pointer with the value 'G'
;  444     0507  1 !       pointing to it (or some previous bucket if there was a previous index
;  445     0508  1 !       update failure).  After the split, the key value for that pointer will
;  446     0509  1 !       be updated to have the key value 'D', and the key value that used to
;  447     0510  1 !       point to it (probably 'G'), will now point to the right hand bucket
;  448     0511  1 !       (with 'E', 'F', and 'G' in it).  The continuation buckets never have
;  449     0512  1 !       an index pointer to them.
;  450     0513  1 !
;  451     0514  1 !       All other split situations are a variation of this one, with
;  452     0515  1 !       one or more of the 'partitions' not present, dependent on the key
;  453     0516  1 !       value and position of insert within the bucket of the record being
;  454     0517  1 !       inserted or updated.  For example, if there are no duplicates, there
;  455     0518  1 !       are no 'low dupes' or 'high dupes'.  Or if the position of insert is
;  456     0519  1 !       at the end of the bucket, there is no 'high set'.
;  457     0520  1 !
;  458     0521  1 !       Now that I've started on it, may as well try to document some other
```

```
;  459          0522  1 !       interesting split situations.  Note that a '2 bucket split' means
;  460          0523  1 !       that there are 2 buckets after the split, i.e., 1 new bucket is
;  461          0524  1 !       added.  The situation described above is a 4 bucket split.
;  462          0525  1 !
;  463          0526  1 !       The most interesting split from an index updating point of view
;  464          0527  1 !       is the 3 bucket split where a record is being inserted in the middle
;  465          0528  1 !       of the bucket and doesn't fit in a bucket with either the low set
;  466          0529  1 !       or the high set.  Again with 1 byte key values to illustrate:
;  467          0530  1 !
;  468          0531  1 !                           G (this is supposed to represent an index
;  469          0532  1 !                           ¦ pointer to this bucket with key value 'g')
;  470          0533  1 !                           ¦
;  471          0534  1 !                           v
;  472          0535  1 !       ------------------
;  473          0536  1 !       ! A B C !! E F G !
;  474          0537  1 !       -------- --------
;  475          0538  1 !              /\
;  476          0539  1 !              new record with key value 'D' inserted, but is so large
;  477          0540  1 !              that it has to have bucket of its own.
;  478          0541  1 !
;  479          0542  1 !       After split (with new index pointers):
;  480          0543  1 !
;  481          0544  1 !              C          D             G
;  482          0545  1 !              ¦          ¦             ¦
;  483          0546  1 !              ¦          ¦             ¦
;  484          0547  1 !              v          v             v
;  485          0548  1 !
;  486          0549  1 !       T-A-B-C-T  ->  T-D-T  ->  T-E-F-G-T
;  487          0550  1 !       ---------      -----      ---------
;  488          0551  1 !
;  489          0552  1 !       The new pointer 'C' is the bucket pointer from the original index
;  490          0553  1 !       record 'G' with the new key value 'C'.  The 'D' pointer is an
;  491          0554  1 !       entirely new record (i.e., key value 'D' and bucket pointer).  The
;  492          0555  1 !       pointer 'G' is the key value from the original record 'G' with
;  493          0556  1 !       a new bucket pointer.  The bucket pointer for the 'D' bucket comes
;  494          0557  1 !       from irb$l_vbn_mid and the bucket pointer for the 'G' bucket comes
;  495          0558  1 !       from irb$l_vbn_right.  Remember that all of this stuff works correctly
;  496          0559  1 !       if the index update failed and we got to the bucket thats splitting
;  497          0560  1 !       by following the horizontal bucket links at the data level.
;  498          0561  1 !       For example, consider the following case where prior index corruption
;  499          0562  1 !       exists:
;  500          0563  1 !
;  501          0564  1 !              G (index update failed when right hand bucket split off
;  502          0565  1 !              ¦ during a previous insert operation)
;  503          0566  1 !              ¦
;  504          0567  1 !              v
;  505          0568  1 !       ---------      ------------
;  506          0569  1 !       ! A B C !  ->  ! D !! F G !
;  507          0570  1 !       ---------      ---- ------
;  508          0571  1 !                         /\
;  509          0572  1 !                        new record 'E' will be inserted here and cause split
;  510          0573  1 !
;  511          0574  1 !       After split:
;  512          0575  1 !
;  513          0576  1 !              E                       G
;  514          0577  1 !              ¦                       ¦
;  515          0578  1 !              ¦                       ¦
```

```
!                  v                         v
!       ---------       -------       -------
!       ! A B C !  ->   ! D E !  ->   ! F G !
!       ---------       -------       -------
!
!       The reason for the index updating behavior becomes more obvious.
!       The key value of the original down pointer 'G' has been changed to
!       the new value 'E', but retaining the original bucket pointer.  Note
!       that we split the bucket with 'D' in it, yet there is no bucket pointer
!       to it now (there wasn't before).  The key value 'G' of the original
!       bucket pointer 'G' has been used with a new bucket pointer for the
!       new bucket created by the split (this is irb$l_vbn_right).
!       Sometimes there will be a bucket split and no records will be
!       in the left hand bucket after the split.  This may happen if the
!       record being inserted belongs at the beginning of the bucket, but
!       there are enough rrv's present so that it doesn't physically fit.
!       In that case, all of the existing records will have to be moved out
!       also.  This may also occur if there are no id values left in the
!       bucket (typically caused by deleted rrv's).  In this case, we would
!       like to swing the index pointer away from the 'empty' bucket to
!       keep random access times from deteriorating.  As of prologue versions
!       1 and 2, however, it will remain in the horizontal link of data
!       buckets.  However,  we can only change the down pointer if it already
!       points to that bucket or we can potentially create crossed down
!       pointers.  The situation is illustrated below:
!
!                 C              I (index update failed on a previous split)
!                 !              !
!                 !              !
!                 v              v
!       ---------       ---------       --------
!       ! A B C !  ->   ! D E F !  ->   !! H I !
!       ---------       ---------       /\------
!                                        record with key value 'G' will be
!                                         inserted here
!
!       Also presume that the bucket 'G' is being inserted in has so many
!       rrv's in it that it won't fit into the existing bucket, even though
!       it will fit into a bucket without any rrv's in it.
!
!       After split:
!
!                 C              I
!                 !              !
!                 !              !
!                 v              v
!       ---------       ---------       ---------------       ---------
!       ! A B C !  ->   ! D E F !  ->   ! rrv's only !  ->   ! G H I !
!       ---------       ---------       ---------------       ---------
!
!       Note that the index pointer 'I' was not moved to point to the new
!       bucket.  If it had been, the bucket containing 'D E F' would have
!       been 'lost' by random access from the index.  This condition is
!       detected by setting irb$l_vbn_left to the vbn of the rrv only
!       bucket.  During the index update procedure, the pointer will be
!       moved to point to the new bucket only if the existing down pointer
!       points to the bucket that was split, i.e., irb$l_vbn_left (this
```

```
;  573          0636  1 !       is normally the case as index corruption is not normal).  Note that
;  574          0637  1 !       an empty left hand bucket may also be present in a 3 bucket split
;  575          0638  1 !       situation.
;  576          0639  1 !
;  577          0640  1 !       Following is a list of the specific split cases handled in the
;  578          0641  1 !       code.  They are basically variations of the above cases.
;  579          0642  1 !
;  580          0643  1 ! these are all the cases of 3 and 4 bucket splits that i can think of
;  581          0644  1 ! any or all of these cases can have the empty left-hand bucket
;  582          0645  1 !       -- this would occur if the first split point is at the beginning
;  583          0646  1 !       -- of the bucket and all data records got moved out
;  584          0647  1 !
;  585          0648  1 ! low dups exist -- no high dups
;  586          0649  1 !       low dups fit w/ rec
;  587          0650  1 !          3 bkt split  low, low dups w/ rec, hi set -- rec goes w/ lo
;  588          0651  1 !           ( SPLIT TYPE 1 )
;  589          0652  1 !
;  590          0653  1 !       low dups don't fit w/ rec
;  591          0654  1 !           3 bkt split w/ rec in its own continuation bucket
;  592          0655  1 !           ( SPLIT TYPE 2, W/ DUPS SEEN )
;  593          0656  1 !
;  594          0657  1 ! hi dups exist -- no low dups
;  595          0658  1 !       hi dups fit w/ rec
;  596          0659  1 !           3 bkt split low, hi dups w/ rec, hi set
;  597          0660  1 !           ( SPLIT TYPE 1 )
;  598          0661  1 !
;  599          0662  1 !       hi dups don't fit w/ rec
;  600          0663  1 !           if no more hi, 3 bkt split low, rec, hi = hi dups is a cont. bkt
;  601          0664  1 !           ( SPLIT TYPE 2 )
;  602          0665  1 !
;  603          0666  1 !           if there is more hi, 4 bkt split low, rec, hi dups, hi
;  604          0667  1 !           ( SPLIT TYPE 2B )
;  605          0668  1 !
;  606          0669  1 ! no dups at all
;  607          0670  1 !       record goes in its own bucket, 3 bkt split
;  608          0671  1 !       (SPLIT TYPE 3 )
;  609          0672  1 !
;  610          0673  1 ! low dups and hi dups
;  611          0674  1 !       all dups fit together
;  612          0675  1 !           3 bkt split w/ dups in middle bkt
;  613          0676  1 !           ( SPLIT TYPE 1 )
;  614          0677  1 !
;  615          0678  1 !       no dups fit w/ record
;  616          0679  1 !           if no more hi, 3 bkt split low, rec = cont. bkt, hi = hi dups = cont. bkt
;  617          0680  1 !
;  618          0681  1 !
;  619          0682  1 !           ( SPLIT TYPE 2B )
;  620          0683  1 !
;  621          0684  1 !           if there is more hi, 4 bkt split low, rec = cont. bkt, hi dups, hi
;  622          0685  1 !           ( SPLIT TYPE 2B )
;  623          0686  1 !
;  624          0687  1 !       hi dups fit w/ record
;  625          0688  1 !           if no more hi, 2 bkt split low, rec w/ hi dups = cont. bkt
;  626          0689  1 !           ( this is a 2 bkt split case that the previous alg. wouldn't handle)
;  627          0690  1 !           ( SPLIT TYPE 4B )
;  628          0691  1 !
;  629          0692  1 !           if there is more hi, 3 bkt split low, rec w/ hi dups = cont. bkt, hi
```

```
;  630          0693  1 !            ( SPLIT TYPE 4 )
;  631          0694  1 !
;  632          0695  1 !        low dups fit w/ record
;  633          0696  1 !            if lo and hi, 4 bkt split low, low dups w/ rec, hi dups, hi
;  634          0697  1 !            ( SPLIT TYPE 5 )
;  635          0698  1 !
;  636          0699  1 !            if no lo and no hi, 3 bkt split (rrv's), low dups w/ rec, hi dups = cont.
;  637          0700  1 !
;  638          0701  1 ! bkt
;  639          0702  1 !            ( SPLIT TYPE 5, w/ empty original bkt and no high )
;  640          0703  1 !
;  641          0704  1 !            if lo but no hi, 3 bkt split lo, low dups w/ rec, hi dups = cont. bkt
;  642          0705  1 !            ( SPLIT TYPE 5, w/ no high )
;  643          0706  1 !
;  644          0707  1 !            if hi but no low, 4 bkt split (rrv's), low dups w/ rec, hi dups, hi
;  645          0708  1 !            ( SPLIT TYPE 5, w/ empty bkt  )
;  646          0709  1 !--
```

```
;  648          0710  1 %SBTTL 'RM$MOVE_KEY'
;  649          0711  1 ROUTINE RM$MOVE_KEY (ADDRESS, CUR_REC_ADDR) : RL$MOVE_KEY NOVALUE =
;  650          0712  1
;  651          0713  1 !++
;  652          0714  1 !
;  653          0715  1 ! FUNCTIONAL DESCRIPTION:
;  654          0716  1 !
;  655          0717  1 !       Routine to move the key from wherever it is desired into
;  656          0718  1 !       key buffer 2.
;  657          0719  1 !
;  658          0720  1 ! CALLING SEQUENCE:
;  659          0721  1 !       bsbw rm$move_key (address,cur_rec_addr)
;  660          0722  1 !
;  661          0723  1 ! INPUT PARAMETERS:
;  662          0724  1 !       address from which to get the key from
;  663          0725  1 !       the current value of rec_addr
;  664          0726  1 !
;  665          0727  1 ! IMPLICIT INPUTS:
;  666          0728  1 !       BKT_ADDR,
;  667          0729  1 !       RAB - user's buffer address
;  668          0730  1 !       IRAB -- pos_ins, rec_w_lo, keybuf
;  669          0731  1 !       IFAB -- kbufsz, prologue version
;  670          0732  1 !       IDX_DFN -- for call to record_key, and compression flags
;  671          0733  1 !
;  672          0734  1 ! OUTPUT PARAMETERS:
;  673          0735  1 !       none
;  674          0736  1 !
;  675          0737  1 ! IMPLICIT OUTPUTS:
;  676          0738  1 !       key is moved into key buffer 2
;  677          0739  1 !
;  678          0740  1 ! ROUTINE VALUE:
;  679          0741  1 !       none
;  680          0742  1 !
;  681          0743  1 ! SIDE EFFECTS:
;  682          0744  1 !       key is moved into key buffer 2
;  683          0745  1 !       AP is clobbered
;  684          0746  1 !
;  685          0747  1 !--
;  686          0748  1
;  687          0749  2     BEGIN
;  688          0750  2
;  689          0751  2     BUILTIN
;  690          0752  2         AP;
;  691          0753  2
;  692          0754  2     GLOBAL REGISTER
;  693          0755  2         R_BDB,
;  694          0756  2         R_IMPURE,
;  695          0757  2         R_REC_ADDR_STR;
;  696          0758  2
;  697          0759  2     EXTERNAL REGISTER
;  698          0760  2         R_IFAB_STR,
;  699          0761  2         R_RAB,
;  700          0762  2         R_IRAB_STR,
;  701          0763  2         R_IDX_DFN_STR,
;  702          0764  2         R_BKT_ADDR_STR;
;  703          0765  2
;  704          0766  2     IF  .CUR_REC_ADDR - .BKT_ADDR EQLU .IRAB[IRB$W_POS_INS]
```

```
;  705     0767  2          AND
;  706     0768  2          .IRAB[IRB$V_REC_W_LO]
;  707     0769  2      THEN
;  708     0770  2          BEGIN
;  709     0771  3          AP = 3;                    ! no overhead, not compressed
;  710     0772  3          REC_ADDR = .IRAB[IRB$L_RBF];
;  711     0773  3          END
;  712     0774  2      ELSE
;  713     0775  3          BEGIN
;  714     0776  3          AP = 0;
;  715     0777  3          REC_ADDR = .ADDRESS;
;  716     0778  3
;  717     0779  3          ! In prologue 3 version files, if the key is compressed, it must be
;  718     0780  3          ! rebuilt.  Make sure that the last non-compressed pointer, is before
;  719     0781  3          ! the record we are looking at.
;  720     0782  3          !
;  721     0783  3
;  722     0784  3          IF .IDX_DFN[IDX$V_KEY_COMPR]
;  723     0785  3          THEN
;  724     0786  4              BEGIN
;  725     0787  4
;  726     0788  4              IF .IRAB[IRB$L_LST_NCMP] GTRU .ADDRESS
;  727     0789  4              THEN
;  728     0790  5                  BEGIN
;  729     0791  5
;  730     0792  5                  IF .(.ADDRESS + RM$REC_OVHD() + 1)<0,8> EQLU 0
;  731     0793  5                  THEN
;  732     0794  5                      IRAB[IRB$L_LST_NCMP] = .ADDRESS
;  733     0795  5                  ELSE
;  734     0796  5                      IRAB[IRB$L_LST_NCMP] = .BKT_ADDR + BKT$C_OVERHDSZ;
;  735     0797  4                  END;
;  736     0798  3              END;
;  737     0799  2          END;
;  738     0800  2
;  739     0801  2      ! We are storing in key buffer 2 the possible key to be inserted at the
;  740     0802  2      ! index level.
;  741     0803  2      !
;  742     0804  2
;  743     0805  2      RM$RECORD_KEY ( KEYBUF_ADDR(2) );
;  744     0806  2
;  745     0807  2      RETURN;
;  746     0808  2
;  747     0809  1      END;

                                                .TITLE  RM3SPLUDR
                                                .IDENT  \V04-000\

                                                .EXTRN  RM$MOVE, RM$RECORD_VBN
                                                .EXTRN  RM$RECORD_KEY, RM$REC_OVHD
                                                .EXTRN  RM$VBN_SIZE, RM$COMPARE_KEY
                                                .EXTRN  RM$COMPARE_REC, RM$GETNEXT_REC

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

                                0850    8F  BB 00000 RM$MOVE_KEY:
                                                PUSHR   #^M<R4,R6,R11>                  ; 0711
```

```
                         5B          50 D0 00004        MOVL    R0, R11                       ;
                         56          55 C2 00007        SUBL2   BKT_ADDR, R6                  ; 0766
     56      48   A9     10          00 ED 0000A        CMPZV   #0, #16, 72(IRAB), R6         ;
                                     0E 12 00010        BNEQ    1$                            ;
                  09     44  A9      03 E1 00012        BBC     #3, 68(IRAB), 1$              ; 0768
                         5C          03 D0 00017        MOVL    #3, AP                        ; 0771
                         56     58   A9 D0 0001A        MOVL    88(IRAB), REC_ADDR            ; 0772
                                     26 11 0001E        BRB     3$                            ; 0766
                                     5C D4 00020 1$:    CLRL    AP                            ; 0776
                         56          5B D0 00022        MOVL    ADDRESS, REC_ADDR             ; 0777
                  1C     1C  A7      06 E1 00025        BBC     #6, 28(IDX_DFN), 3$           ; 0784
                         54   0098   C9 9E 0002A        MOVAB   152(IRAB), R4                 ; 0788
                         5B          64 D1 0002F        CMPL    (R4), ADDRESS                 ;
                                     12 1B 00032        BLEQU   3$                            ;
                                  0000G 30 00034        BSBW    RM$REC_OVHD                   ; 0792
                              01 A04B 95 00037          TSTB    1(R0)[ADDRESS]                ;
                                     05 12 0003B        BNEQ    2$                            ;
                         64          5B D0 0003D        MOVL    ADDRESS, (R4)                 ; 0794
                                     04 11 00040        BRB     3$                            ;
                         64     0E   A5 9E 00042 2$:    MOVAB   14(R5), (R4)                  ; 0796
                         50   00B4   CA 3C 00046 3$:    MOVZWL  180(IFAB), R0                 ; 0805
                              60 B940 9F 0004B          PUSHAB  @96(IRAB)[R0]                 ;
                                  0000G 30 0004F        BSBW    RM$RECORD_KEY                 ;
                         5E          04 C0 00052        ADDL2   #4, SP                        ;
                              0850   8F BA 00055        POPR    #^M<R4,R6,R11>                ; 0809
                                        05 00059        RSB                                   ;

; Routine Size:  90 bytes,    Routine Base:  RM$RMS3 + 0000

;  748          0810  1
```

```
;  750          0811  1 %SBTTL 'RM$BUILD_KEY'
;  751          0812  1 ROUTINE RM$BUILD_KEY (ADDRESS, KEYBUF) : RL$BUILD_KEY NOVALUE =
;  752          0813  1
;  753          0814  1 !++
;  754          0815  1 !
;  755          0816  1 ! FUNCTIONAL DESCRIPTION:
;  756          0817  1 !
;  757          0818  1 !       This routine builds a compressed key from the record into the
;  758          0819  1 !       given buffer, knowing that the front characters are valid from
;  759          0820  1 !       the previous expansion.
;  760          0821  1 !
;  761          0822  1 ! CALLING SEQUENCE:
;  762          0823  1 !       bsbw rm$build_key (address,keybuf)
;  763          0824  1 !
;  764          0825  1 ! INPUT PARAMETERS:
;  765          0826  1 !       - address in bucket which points to key compression overhead
;  766          0827  1 !       - key output buffer
;  767          0828  1 !
;  768          0829  1 ! IMPLICIT INPUTS:
;  769          0830  1 !       IDX_DFN - index definition for key size
;  770          0831  1 !
;  771          0832  1 ! OUTPUT PARAMETERS:
;  772          0833  1 !       none
;  773          0834  1 !
;  774          0835  1 ! IMPLICIT OUTPUTS:
;  775          0836  1 !       key is moved into appropiate key buffer
;  776          0837  1 !
;  777          0838  1 ! ROUTINE VALUE:
;  778          0839  1 !       none
;  779          0840  1 !
;  780          0841  1 ! SIDE EFFECTS:
;  781          0842  1 !       key is moved into appropiate key buffer
;  782          0843  1 !
;  783          0844  1 !--
;  784          0845  1
;  785          0846  2     BEGIN
;  786          0847  2
;  787          0848  2     EXTERNAL REGISTER
;  788          0849  2         R_IDX_DFN_STR;
;  789          0850  2
;  790          0851  2     LOCAL
;  791          0852  2         TRUN_CHAR,
;  792          0853  2         LENGTH;
;  793          0854  2
;  794          0855  2     BIND
;  795          0856  2         REC_KEY = ADDRESS : REF BBLOCK;
;  796          0857  2
;  797          0858  2     MACRO
;  798          0859  2         KEY_LEN = 0,0,8,0 %,
;  799          0860  2         CMP_CNT = 1,0,8,0 %;
;  800          0861  2
;  801          0862  2     KEYBUF = .KEYBUF + .REC_KEY[CMP_CNT];       ! skip characters already moved
;  802          0863  2     TRUN_CHAR = .REC_KEY + .REC_KEY[KEY_LEN] + 1;
;  803          0864  2     LENGTH = .IDX_DFN[IDX$B_KEYSZ] - .REC_KEY[CMP_CNT];
;  804          0865  2     CH$COPY ( .REC_KEY[KEY_LEN], .REC_KEY + 2,
;  805          0866  2               ..TRUN_CHAR,
;  806          0867  2               .LENGTH, .KEYBUF );
```

```
;  807          0868  2
;  808          0869  2     RETURN;
;  809          0870  2
;  810          0871  1     END;



                                                3E  BB 00000 RM$BUILD_KEY:
                                                             PUSHR   #^M<R1,R2,R3,R4,R5>                                 ; 0812
                                    51      18  AE  D0 00002         MOVL    REC_KEY, R1                                 ; 0862
                                    50      01  A1  9A 00006         MOVZBL  1(R1), R0                                   ;
                                  1C    AE      50  C0 0000A         ADDL2   R0, KEYBUF                                  ;
                                    50          61  9A 0000E         MOVZBL  (R1), R0                                    ; 0863
                                    53      01 A140 9E 00011         MOVAB   1(R1)[R0], TRUN_CHAR                        ;
                                    52      20  A7  9A 00016         MOVZBL  32(IDX_DFN), LENGTH                         ; 0864
                                    50      01  A1  9A 0001A         MOVZBL  1(R1), R0                                   ;
                                    52          50  C2 0001E         SUBL2   R0, LENGTH                                  ;
                                    50          61  9A 00021         MOVZBL  (R1), R0                                    ; 0865
          52                63      02  A1          50  2C 00024         MOVC5   R0, 2(R1), (TRUN_CHAR), LENGTH, @KEYBUF     ; 0867
                                                1C  BE      0002A     ;
                                                3E  BA 0002C         POPR    #^M<R1,R2,R3,R4,R5>                         ; 0871
                                                    05 0002E         RSB                                                 ;

; Routine Size:  47 bytes,    Routine Base:  RM$RMS3 + 005A

;  811          0872  1
```

```
;  813     0873  1 %SBTTL 'RM$SPLIT_UDR'
;  814     0874  1 GLOBAL ROUTINE RM$SPLIT_UDR : RL$RABREG_4567 NOVALUE =
;  815     0875  1
;  816     0876  1 !++
;  817     0877  1 !
;  818     0878  1 ! FUNCTIONAL DESCRIPTION:
;  819     0879  1 !
;  820     0880  1 !
;  821     0881  1 ! CALLING SEQUENCE:
;  822     0882  1 !       BSBW RM$SPLIT_UDR()
;  823     0883  1 !
;  824     0884  1 ! INPUT PARAMETERS:
;  825     0885  1 !       none
;  826     0886  1 !
;  827     0887  1 ! IMPLICIT INPUTS:
;  828     0888  1 !       BDB pointer, BUFFER pointer, REC_ADDR = point of insert, IDX_DFN
;  829     0889  1 !       in IRAB -- curbdb, associated w/ bdb and bkt_addr
;  830     0890  1 !                  pos_ins corresponding to rec_addr
;  831     0891  1 !       in RAB -- rsz of record
;  832     0892  1 !       in IFAB -- rfm
;  833     0893  1 !       BKT$B_NXTRECID = 0 in original bucket signals that this is
;  834     0894  1 !       a split due to a lack of id's in the bucket
;  835     0895  1 !
;  836     0896  1 ! OUTPUT PARAMETERS:
;  837     0897  1 !       none
;  838     0898  1 !
;  839     0899  1 ! IMPLICIT OUTPUTS:
;  840     0900  1 !       in IRAB --
;  841     0901  1 !       if 2 bkt split --
;  842     0902  1 !               IRB$W_SPLIT, offset to split point
;  843     0903  1 !               IRB$V_REC_W_LO -- set if split point is pos_insert and
;  844     0904  1 !                       record goes w/ lo set
;  845     0905  1 !               new high key for original bucket in keybuffer 2
;  846     0906  1 !               number of new buckets = 1
;  847     0907  1 !               if original bucket was all rrv's, set IRB$V_EMPTY_BKT flag
;  848     0908  1 !               if new bucket is a continuation bkt., set IRB$V_CONT_BKT flag
;  849     0909  1 !       if 3 bkt split --
;  850     0910  1 !               same as above w/ these changes:
;  851     0911  1 !               IRB$W_SPLIT_1, offset to second split point
;  852     0912  1 !               number of new buckets = 2
;  853     0913  1 !               if right bucket is a continuation bkt, set IRB$V_CONT_R flag
;  854     0914  1 !       if 4 bkt split --
;  855     0915  1 !               same as above w/ these changes:
;  856     0916  1 !               IRB$W_SPLIT_2, offset to third split point
;  857     0917  1 !               number of new buckets = 3
;  858     0918  1 !
;  859     0919  1 ! ROUTINE VALUE:
;  860     0920  1 !       rmssuc
;  861     0921  1 !
;  862     0922  1 ! SIDE EFFECTS:
;  863     0923  1 !       AP is clobbered
;  864     0924  1 !
;  865     0925  1 !--
;  866     0926  1
;  867     0927  2     BEGIN
;  868     0928  2
;  869     0929  2     EXTERNAL REGISTER
```

```
;  870          0930  2          COMMON_RAB_STR,
;  871          0931  2          R_REC_ADDR_STR,
;  872          0932  2          R_IDX_DFN_STR,
;  873          0933  2          COMMON_IO_STR;
;  874          0934  2
;  875          0935  2      LOCAL
;  876          0936  2          SAVE_REC_W_LO,
;  877          0937  2          NUM_RRVS,
;  878          0938  2          POS_INSERT,
;  879          0939  2          EOB,
;  880          0940  2          RRV,
;  881          0941  2          RHS,
;  882          0942  2          LHS,
;  883          0943  2          LAST     : REF BBLOCK,
;  884          0944  2          LAST_DIFF,
;  885          0945  2          BKTSIZE,
;  886          0946  2          REC_SIZE,
;  887          0947  2          DIFFERENCE;
;  888          0948  2
;  889          0949  2      MACRO
;  890          0950  2          NEED_RRV = NUM_RRVS<0,16> %,
;  891          0951  2          NOT_NEED_RRV = NUM_RRVS<16,16> %;
;  892          0952  2
;  893          0953  2      LABEL
;  894          0954  2          DO_IT,
;  895          0955  2          HALF,
;  896          0956  2          NEXT;
;  897          0957  2
;  898          0958  2  DO_IT :
;  899          0959  2
;  900          0960  3      BEGIN
;  901          0961  3
;  902          0962  3      ! define a block so that we can have some common checks before returning
;  903          0963  3      ! successfully
;  904          0964  3      !
;  905          0965  3  HALF :
;  906          0966  3
;  907          0967  4      BEGIN
;  908          0968  4
;  909          0969  4      !+
;  910          0970  4      ! define a block so that we can simulate a go-to (naughty, naughty)
;  911          0971  4      ! if we have decided that we are positioning at the end of the bucket
;  912          0972  4      ! & we're in somewhat of an ascending order, where the last record
;  913          0973  4      ! inserted is a duplicate of the new record, skip over the 50-50 code
;  914          0974  4      ! and go to the code to take duplicates into account
;  915          0975  4      !
;  916          0976  4      ! scan 1 -- calculate
;  917          0977  4      ! size of existing rrv's and total number of rrv's needed to move the whole
;  918          0978  4      ! bucket out ( worst case) as a side effect, adjust eob ptr to pt to the
;  919          0979  4      ! rrv's instead of freespace assume not empty bucket until showed otherwise
;  920          0980  4      !-
;  921          0981  4
;  922          0982  4      IRAB[IRB$V_EMPTY_BKT] = 0;
;  923          0983  4
;  924          0984  4      ! new rec is tried 1st w/ hi set, then w/ lo set
;  925          0985  4      !
;  926          0986  4      IRAB[IRB$V_REC_W_LO] = 0;
```

```
;  927     0987  4     IRAB[IRB$V_NEW_BKTS] = 1;    ! assume 2-bkt split until showed otherwise
;  928     0988  4     NUM_RRVS = 0;                ! this zeroes NEED_RRV and NOT_NEED_RRV
;  929     0989  4     POS_INSERT = .REC_ADDR;
;  930     0990  4     REC_ADDR = .BKT_ADDR + BKT$C_OVERHDSZ;
;  931     0991  4     EOB = .BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE];
;  932     0992  4     LAST = 0;
;  933     0993  4
;  934     0994  4     DO
;  935     0995  5         BEGIN
;  936     0996  5
;  937     0997  5         BUILTIN
;  938     0998  5             AP;
;  939     0999  5
;  940     1000  5         IF .REC_ADDR[IRC$V_RRV]
;  941     1001  5         THEN
;  942     1002  5             EXITLOOP;
;  943     1003  5
;  944     1004  5         AP = 3;
;  945     1005  5
;  946     1006  5         IF .BDB[BDB$L_VBN] EQLU RM$RECORD_VBN()
;  947     1007  5         THEN
;  948     1008  5             NEED_RRV = .NEED_RRV + 1
;  949     1009  5
;  950     1010  5             ! the records not requiring rrv's are counted also because in the
;  951     1011  5             ! case where we're splitting due to lack of id's, the lhs side will
;  952     1012  5             ! fit with the new record if any of the record being moved to the
;  953     1013  5             ! new bucket doesn't require an rrv.  this will be checked when we
;  954     1014  5             ! check to see if the lhs will fit after the first point that the
;  955     1015  5             ! rhs fits.
;  956     1016  5             !
;  957     1017  5         ELSE
;  958     1018  5             NOT_NEED_RRV = .NOT_NEED_RRV + 1;
;  959     1019  5
;  960     1020  5         LAST = .REC_ADDR;
;  961     1021  5         RM$GETNEXT_REC()
;  962     1022  5         END
;  963     1023  4     UNTIL .REC_ADDR GEQU .EOB;
;  964     1024  4
;  965     1025  4     ! set split_2 and split_1 to be eob, so if there's less than 3 new buckets
;  966     1026  4     ! bkt_spl can use the value w/o having to recalculate it also set up the
;  967     1027  4     ! bucket size and the record size
;  968     1028  4     !
;  969     1029  4     IRAB[IRB$W_SPLIT_1] = IRAB[IRB$W_SPLIT_2] = .REC_ADDR - .BKT_ADDR;
;  970     1030  4     BKTSIZE = .IDX_DFN[IDX$B_DATBKTSZ]*512 - BKT$C_OVERHDSZ - 1;
;  971     1031  4
;  972     1032  4     REC_SIZE = .RAB[RAB$W_RSZ] + IRC$C_FIXOVHDSZ;
;  973     1033  4
;  974     1034  4     IF .IFAB[IFB$B_RFMORG] NEQ FAB$C_FIX
;  975     1035  4     THEN
;  976     1036  4         REC_SIZE = .REC_SIZE + 2;
;  977     1037  4
;  978     1038  4     ! if this is an update, may have to count in an rrv for the existing record
;  979     1039  4     !
;  980     1040  4     IF .IRAB[IRB$V_UPDATE]
;  981     1041  4     THEN
;  982     1042  5         BEGIN
;  983     1043  5
```

```
;  984          1044  5          IF .BDB[BDB$L_VBN] EQLU .IRAB[IRB$L_PUTUP_VBN]
;  985          1045  5          THEN
;  986          1046  5              NEED_RRV = .NEED_RRV + 1
;  987          1047  5
;  988          1048  4          END;
;  989          1049  4
;  990          1050  4      RRV = .EOB - .REC_ADDR;                     ! size of existing rrv's
;  991          1051  4      EOB = .REC_ADDR;                            ! adjust eob
;  992          1052  4
;  993          1053  4      ! special case it, if the bucket was all rrv's
;  994          1054  4      !
;  995          1055  4
;  996          1056  4      IF .REC_ADDR EQLU .BKT_ADDR + BKT$C_OVERHDSZ
;  997          1057  4      THEN
;  998          1058  5          BEGIN
;  999          1059  5
; 1000          1060  5          ! bkt is all rrv's yet the record wouldn't fit so we need to
; 1001          1061  5          ! allocate another bkt ( 2 bkt split) yet special case it so as not
; 1002          1062  5          ! to make another idx entry only to update the existing one by
; 1003          1063  5          ! setting empty bucket flag
; 1004          1064  5          !
; 1005          1065  5          IRAB[IRB$W_SPLIT] = .REC_ADDR - .BKT_ADDR;
; 1006          1066  5          LEAVE DO_IT
; 1007          1067  5
; 1008          1068  4          END;                        ! { of special case an all-rrv bucket }
; 1009          1069  4
; 1010          1070  4      !+
; 1011          1071  4      ! special case -- if we can detect a possible ascending order to these
; 1012          1072  4      ! records it probably will be better to do a straight point of insert split
; 1013          1073  4      ! this would put the new record in a bucket all by itself.
; 1014          1074  4      ! do this kind of split if and only if all the following conditions are met:
; 1015          1075  4      !    1) the record is being inserted at the end of bucket
; 1016          1076  4      !    2) the last record physically in the bkt is the last record to have
; 1017          1077  4      !       been inserted
; 1018          1078  4      !    3) the last record and the new record do not have duplicate key values
; 1019          1079  4      !
; 1020          1080  4      ! note that if they are duplicates, we can still make an optimization by
; 1021          1081  4      ! skipping the 50-50 split code
; 1022          1082  4      !
; 1023          1083  4      ! note that last cannot be zero, since if it were we
; 1024          1084  4      ! would have an all rrv bkt
; 1025          1085  4      !-
; 1026          1086  4
; 1027          1087  4      IF .POS_INSERT EQLU .REC_ADDR
; 1028          1088  4          AND
; 1029          1089  5          (((.LAST[IRC$B_ID] + 1)  AND  %X'FF') EQLU .BKT_ADDR[BKT$B_NXTRECID])
; 1030          1090  4      THEN
; 1031          1091  5          BEGIN
; 1032          1092  5          REC_ADDR = .LAST;
; 1033          1093  5
; 1034          1094  5          IF RM$COMPARE_REC(KEYBUF_ADDR(3), .IDX_DFN[IDX$B_KEYSZ], 0)
; 1035          1095  5          THEN
; 1036          1096  6              BEGIN
; 1037          1097  6
; 1038          1098  6              ! since we have detected a possible ascending order in the input
; 1039          1099  6              ! let's try to optimize a little and split at the point of insert
; 1040          1100  6              ! send the record by itself into the new bucket have to set up the
```

```
;  1041          1101  6              ! key value and the split point and that's it
;  1042          1102  6              !
;  1043          1103  6              RM$MOVE_KEY(.REC_ADDR, .REC_ADDR);
;  1044          1104  6              IRAB[IRB$W_SPLIT] = .IRAB[IRB$W_POS_INS];
;  1045          1105  6              LEAVE DO_IT;
;  1046          1106  6
;  1047          1107  6              END
;  1048          1108  5          ELSE
;  1049          1109  5              LEAVE HALF;
;  1050          1110  5
;  1051          1111  5          ! { end of trying to special case insertion of records in ascending
;  1052          1112  5          ; order }
;  1053          1113  5          !
;  1054          1114  4          END;
;  1055          1115  4
;  1056          1116  4      REC_ADDR = .BKT_ADDR + BKT$C_OVERHDSZ;
;  1057          1117  4      LAST_DIFF = %X'7FFFFFFF';
;  1058          1118  4      LAST = 0;
;  1059          1119  4      SAVE_REC_W_LO = 0;
;  1060          1120  4
;  1061          1121  4      ! start from the beginning of the bucket and scan rightward. first find the
;  1062          1122  4      ! 1st place the rhs will fit in 1 bkt then, as long as the lhs will fit in
;  1063          1123  4      ! a bkt, try to find an optimal point if there is no point where the rhs
;  1064          1124  4      ! and lhs will both fit. we can't do a 2-bkt split and this case will fall
;  1065          1125  4      ! out
;  1066          1126  4      !
;  1067          1127  4
;  1068          1128  4      WHILE 1
;  1069          1129  4      DO
;  1070          1130  5          BEGIN
;  1071          1131  5          RHS = .EOB - .REC_ADDR;
;  1072          1132  5
;  1073          1133  5          IF  .REC_ADDR LEQU .POS_INSERT
;  1074          1134  5              AND
;  1075          1135  5              NOT .IRAB[IRB$V_REC_W_LO]
;  1076          1136  5          THEN
;  1077          1137  5              RHS = .RHS + .REC_SIZE;
;  1078          1138  5
;  1079          1139  5          ! the right hand side fits if there is enough room and there are id's
;  1080          1140  5          ! available. id's are always available in the new bucket in the update
;  1081          1141  5          ! situation, or if we're leaving at least 1 record behind in the old
;  1082          1142  5          ! bucket. note that nxtrecid is always zeroed if this is a split due to
;  1083          1143  5          ! lack of id's.
;  1084          1144  5          !
;  1085          1145  5
;  1086          1146  5          IF  .RHS LSSU .BKTSIZE
;  1087          1147  5              AND
;  1088          1148  6              (.BKT_ADDR[BKT$B_NXTRECID] NEQ 0
;  1089          1149  6              OR
;  1090          1150  6              .IRAB[IRB$V_UPDATE]
;  1091          1151  6              OR
;  1092          1152  7              .REC_ADDR NEQA (.BKT_ADDR + BKT$C_OVERHDSZ)
;  1093          1153  6              OR
;  1094          1154  6              .IRAB[IRB$V_REC_W_LO])
;  1095          1155  5          THEN
;  1096          1156  6              BEGIN
;  1097          1157  6              LHS = .REC_ADDR - (.BKT_ADDR + BKT$C_OVERHDSZ);
```

```
;  1098          1158  6
;  1099          1159  6          IF  .REC_ADDR GEQU .POS_INSERT
;  1100          1160  6              AND
;  1101          1161  6              .IRAB[IRB$V_REC_W_LO]
;  1102          1162  6          THEN
;  1103          1163  6              LHS = .LHS + .REC_SIZE;
;  1104          1164  6
;  1105          1165  6          ! will lhs fit ? lhs doesn't fit if there is no space in the
;  1106          1166  6          ! bucket, or if there won't be any id's available in the bucket,
;  1107          1167  6          ! if not & if there is no previous point at which it fit, goto 3-bkt
;  1108          1168  6          ! split code if there is a previous place where we could have had a
;  1109          1169  6          ! 2-bkt split, use it
;  1110          1170  6          !
;  1111          1171  6
;  1112          1172  6          IF .LHS + .RRV + (7*.NEED_RRV) GTRU .BKTSIZE
;  1113          1173  6
;  1114          1174  6              ! id's will be available in the original bucket if we aren't
;  1115          1175  6              ! out of id's to begin with, this is an update, any record
;  1116          1176  6              ! being moved out doesn't need an rrv, or the new record is
;  1117          1177  6              ! going in the new bucket
;  1118          1178  6              !
;  1119          1179  6              OR
;  1120          1180  7              (.BKT_ADDR[BKT$B_NXTRECID] EQL 0
;  1121          1181  7              AND
;  1122          1182  7              NOT .IRAB[IRB$V_UPDATE]
;  1123          1183  7              AND
;  1124          1184  7              .NOT_NEED_RRV EQL 0
;  1125          1185  7              AND
;  1126          1186  7              .IRAB[IRB$V_REC_W_LO])
;  1127          1187  6          THEN
;  1128          1188  7              BEGIN
;  1129          1189  7
;  1130          1190  7              IF .LAST EQL 0
;  1131          1191  7              THEN
;  1132          1192  7                  EXITLOOP;
;  1133          1193  7
;  1134          1194  7              REC_ADDR = .LAST;
;  1135          1195  7
;  1136          1196  7              IF NOT .SAVE_REC_W_LO
;  1137          1197  7              THEN
;  1138          1198  7                  IRAB[IRB$V_REC_W_LO] = 0;
;  1139          1199  7
;  1140          1200  7              ! 2 bkt split is possible rec_addr points to the most
;  1141          1201  7              ! optimal place since we had to back up, reset last to point
;  1142          1202  7              ! to the record immediately before the split point
;  1143          1203  7              !
;  1144          1204  8              BEGIN
;  1145          1205  8
;  1146          1206  8              LOCAL
;  1147          1207  8                  TMP;
;  1148          1208  8
;  1149          1209  8              TMP = .REC_ADDR;
;  1150          1210  8              REC_ADDR = .BKT_ADDR + BKT$C_OVERHDSZ;
;  1151          1211  8              LAST = .REC_ADDR;
;  1152          1212  8
;  1153          1213  8              WHILE .REC_ADDR NEQU .TMP
;  1154          1214  8              DO
```

```
;  1155     1215  9                BEGIN
;  1156     1216  9                LAST = .REC_ADDR;
;  1157     1217  9                RM$GETNEXT_REC();
;  1158     1218  8                END;
;  1159     1219  8
;  1160     1220  7            END;
;  1161     1221  7            RM$MOVE_KEY(.LAST, .REC_ADDR);
;  1162     1222  7            IRAB[IRB$W_SPLIT] = .REC_ADDR - .BKT_ADDR;
;  1163     1223  7
;  1164     1224  7            ! treat another exception case of the new record going off into
;  1165     1225  7            ! a cont. bkt all by itself
;  1166     1226  7            !
;  1167     1227  7
;  1168     1228  7            IF .IRAB[IRB$W_SPLIT] EQLU .IRAB[IRB$W_POS_INS]
;  1169     1229  7            THEN
;  1170     1230  7
;  1171     1231  7                IF .IRAB[IRB$W_SPLIT] EQLU .IRAB[IRB$W_SPLIT_1]
;  1172     1232  7                THEN
;  1173     1233  7
;  1174     1234  7                    IF NOT .IRAB[IRB$V_REC_W_LO]
;  1175     1235  7                    THEN
;  1176     1236  8                        BEGIN
;  1177     1237  8
;  1178     1238  8                        BUILTIN
;  1179     1239  8                            AP;
;  1180     1240  8
;  1181     1241  8                        AP = 3;
;  1182     1242  8
;  1183     1243  8                        IF NOT RM$COMPARE_KEY(KEYBUF_ADDR(2),
;  1184     1244  8                                              KEYBUF_ADDR(3),
;  1185     1245  8                                              .IDX_DFN[IDX$B_KEYSZ])
;  1186     1246  8                        THEN
;  1187     1247  8                            IRAB[IRB$V_CONT_BKT] = 1;
;  1188     1248  8
;  1189     1249  7                        END;
;  1190     1250  7
;  1191     1251  7                LEAVE DO_IT
;  1192     1252  7
;  1193     1253  6                END;                    ! { end of lhs doesn't fit anymore }
;  1194     1254  6
;  1195     1255  6            ! lhs fits also, calculate the magic ratio
;  1196     1256  6            !
;  1197     1257  8            DIFFERENCE = (.LHS*.BKTSIZE) - (.RHS*(.BKTSIZE - (7*.NEED_RRV) -
;  1198     1258  6            .RRV));
;  1199     1259  6
;  1200     1260  6            IF .DIFFERENCE GEQ 0
;  1201     1261  6            THEN
;  1202     1262  7                BEGIN
;  1203     1263  7
;  1204     1264  7                ! found the 1st point at which the magic ratio is positive
;  1205     1265  7                ! was the last point more optimal, if so use it
;  1206     1266  7                !
;  1207     1267  7
;  1208     1268  7                IF ABS(.DIFFERENCE) GTRU ABS(.LAST_DIFF)
;  1209     1269  7                THEN
;  1210     1270  8                    BEGIN
;  1211     1271  8
```

```
;  1212          1272  8                    IF .REC_ADDR EQLU .LAST
;  1213          1273  8                    THEN
;  1214          1274  8                        IRAB[IRB$V_REC_W_LO] = 0
;  1215          1275  8                    ELSE
;  1216          1276  9                        (REC_ADDR = .LAST;
;  1217          1277  9
;  1218          1278  9                        IF .REC_ADDR LSSU .POS_INSERT
;  1219          1279  9                        THEN
;  1220          1280  8                            IRAB[IRB$V_REC_W_LO] = 0);
;  1221          1281  8
;  1222          1282  8                    LAST = 0;
;  1223          1283  7                    END;
;  1224          1284  7
;  1225          1285  7                ! 2-bkt split is possible rec_addr points to the most
;  1226          1286  7                ! optimal place
;  1227          1287  7                !
;  1228          1288  7
;  1229          1289  7                IF .LAST EQL 0
;  1230          1290  7                THEN            ! just backed up rec_addr, need to recalc last
;  1231          1291  8                    BEGIN
;  1232          1292  8
;  1233          1293  8                    LOCAL
;  1234          1294  8                        TMP;
;  1235          1295  8
;  1236          1296  8                    TMP = .REC_ADDR;
;  1237          1297  8                    REC_ADDR = .BKT_ADDR + BKT$C_OVERHDSZ;
;  1238          1298  8                    LAST = .REC_ADDR;
;  1239          1299  8
;  1240          1300  8                    WHILE .REC_ADDR NEQU .TMP
;  1241          1301  8                    DO
;  1242          1302  9                        BEGIN
;  1243          1303  9                        LAST = .REC_ADDR;
;  1244          1304  9                        RM$GETNEXT_REC();
;  1245          1305  8                        END;
;  1246          1306  8
;  1247          1307  7                    END;
;  1248          1308  7
;  1249          1309  7                RM$MOVE_KEY(.LAST, .REC_ADDR);
;  1250          1310  7                IRAB[IRB$W_SPLIT] = .REC_ADDR - .BKT_ADDR;
;  1251          1311  7
;  1252          1312  7                ! treat another exception case of the new record going off into
;  1253          1313  7                ! a cont. bkt all by itself
;  1254          1314  7                !
;  1255          1315  7
;  1256          1316  7                IF .IRAB[IRB$W_SPLIT] EQLU .IRAB[IRB$W_POS_INS]
;  1257          1317  7                THEN
;  1258          1318  7
;  1259          1319  7                    IF .IRAB[IRB$W_SPLIT] EQLU .IRAB[IRB$W_SPLIT_1]
;  1260          1320  7                    THEN
;  1261          1321  7
;  1262          1322  7                        IF NOT .IRAB[IRB$V_REC_W_LO]
;  1263          1323  7                        THEN
;  1264          1324  8                            BEGIN
;  1265          1325  8
;  1266          1326  8                            BUILTIN
;  1267          1327  8                                AP;
;  1268          1328  8
```

```
;  1269          1329  8                          AP = 3;
;  1270          1330  8
;  1271          1331  8                          IF NOT RM$COMPARE_KEY(KEYBUF_ADDR(2),
;  1272          1332  8                                                KEYBUF_ADDR(3),
;  1273          1333  8                                               .IDX_DFN[IDX$B_KEYSZ])
;  1274          1334  8                          THEN
;  1275          1335  8                              IRAB[IRB$V_CONT_BKT] = 1;
;  1276          1336  8
;  1277          1337  7                          END;
;  1278          1338  7
;  1279          1339  7                      LEAVE DO_IT
;  1280          1340  7
;  1281          1341  6                      END;
;  1282          1342  6
;  1283          1343  6                  ! the magic ratio isn't positive yet, so save all the context and
;  1284          1344  6                  ! move on to the next record
;  1285          1345  6                  !
;  1286          1346  6                  LAST_DIFF = .DIFFERENCE;
;  1287          1347  6                  LAST = .REC_ADDR;
;  1288          1348  6
;  1289          1349  6                  IF .IRAB[IRB$V_REC_W_LO]
;  1290          1350  6                  THEN
;  1291          1351  6                      SAVE_REC_W_LO = 1;
;  1292          1352  6
;  1293          1353  5                  END;                ! { end of rhs fits, is this a good point? }
;  1294          1354  5
;  1295          1355  5              ! go on to the next record
;  1296          1356  5              !
;  1297          1357  5 NEXT :
;  1298          1358  6              BEGIN
;  1299          1359  6
;  1300          1360  6              IF  .REC_ADDR EQLU .POS_INSERT
;  1301          1361  6                  AND
;  1302          1362  6                  NOT .IRAB[IRB$V_REC_W_LO]
;  1303          1363  6              THEN
;  1304          1364  7                  BEGIN
;  1305          1365  7
;  1306          1366  7                  ! if this is an update and we pass the record, check to see if it
;  1307          1367  7                  ! needed an rrv
;  1308          1368  7                  !
;  1309          1369  7                  IF .IRAB[IRB$V_UPDATE]
;  1310          1370  7                  THEN
;  1311          1371  8                      BEGIN
;  1312          1372  8
;  1313          1373  8                      IF .BDB[BDB$L_VBN] EQLU .IRAB[IRB$L_PUTUP_VBN]
;  1314          1374  8                      THEN
;  1315          1375  8                          NEED_RRV = .NEED_RRV - 1;
;  1316          1376  8
;  1317          1377  7                      END;
;  1318          1378  7
;  1319          1379  7                  IRAB[IRB$V_REC_W_LO] = 1;
;  1320          1380  7                  RM$MOVE_KEY(.REC_ADDR, .REC_ADDR);
;  1321          1381  7
;  1322          1382  7                  IF  .REC_ADDR EQLU .EOB
;  1323          1383  7                  THEN
;  1324          1384  7                      LEAVE NEXT
;  1325          1385  7
```

```
;  1326     1386  7          ELSE IF RM$COMPARE_REC(KEYBUF_ADDR(2), .IDX_DFN[IDX$B_KEYSZ], 0)
;  1327     1387  7          THEN
;  1328     1388  7              LEAVE NEXT;
;  1329     1389  7
;  1330     1390  6          END;        ! { end of at position for insert for the 1st time }
;  1331     1391  6
;  1332     1392  6      ! fool move key a little by always clearing rec_w_lo to always get the
;  1333     1393  6      ! key associated w/ the record at pos_ins.  (I think it is the key of
;  1334     1394  6      ! the record we are pointing to, not the one at pos_ins...)
;  1335     1395  6      !
;  1336     1396  7      BEGIN
;  1337     1397  7
;  1338     1398  7      LOCAL
;  1339     1399  7          TMP : BYTE;
;  1340     1400  7
;  1341     1401  7      TMP = .IRAB[IRB$B_SPL_BITS];
;  1342     1402  7      IRAB[IRB$V_REC_W_[0] = 0;
;  1343     1403  7      RM$MOVE_KEY(.REC_ADDR, .REC_ADDR);
;  1344     1404  7      IRAB[IRB$B_SPL_BITS] = .TMP
;  1345     1405  6      END;
;  1346     1406  6
;  1347     1407  6      DO
;  1348     1408  7          BEGIN
;  1349     1409  7
;  1350     1410  7          BUILTIN
;  1351     1411  7              AP;
;  1352     1412  7
;  1353     1413  7          IF .REC_ADDR EQLU .EOB
;  1354     1414  7          THEN
;  1355     1415  7              EXITLOOP;
;  1356     1416  7
;  1357     1417  7          AP = 3;
;  1358     1418  7
;  1359     1419  7          IF .BDB[BDB$L_VBN] EQLU RM$RECORD_VBN()
;  1360     1420  7          THEN
;  1361     1421  7              NEED_RRV = .NEED_RRV - 1
;  1362     1422  7          ELSE
;  1363     1423  7              NOT_NEED_RRV = .NOT_NEED_RRV - 1;
;  1364     1424  7
;  1365     1425  7          RM$GETNEXT_REC();
;  1366     1426  7
;  1367     1427  7          IF .REC_ADDR EQLU .EOB
;  1368     1428  7          THEN
;  1369     1429  7              EXITLOOP;
;  1370     1430  7
;  1371     1431  7          END
;  1372     1432  7
;  1373     1433  7      ! compare_rec returns 0 if a match
;  1374     1434  7      !
;  1375     1435  6      UNTIL RM$COMPARE_REC(KEYBUF_ADDR(2), .IDX_DFN[IDX$B_KEYSZ], 0);
;  1376     1436  6
;  1377     1437  6      ! if the key compares brought us up to the pos of insert, see if the
;  1378     1438  6      ! key of the new record matches. if it does, have to include it w/ the
;  1379     1439  6      ! lhs
;  1380     1440  6      !
;  1381     1441  6
;  1382     1442  6      IF .REC_ADDR EQLU .POS_INSERT
```

```
;  1383          1443  6          THEN
;  1384          1444  7              BEGIN
;  1385          1445  7
;  1386          1446  7              BUILTIN
;  1387          1447  7                  AP;
;  1388          1448  7
;  1389          1449  7              AP = 3;
;  1390          1450  7
;  1391          1451  7              IF NOT RM$COMPARE_KEY(KEYBUF_ADDR(2),
;  1392          1452  7                                    KEYBUF_ADDR(3),
;  1393          1453  7                                   .IDX_DFN[IDX$B_KEYSZ])
;  1394          1454  7              THEN
;  1395          1455  8                  BEGIN
;  1396          1456  8                  IRAB[IRB$V_REC_W_LO] = 1;
;  1397          1457  8
;  1398          1458  8                  IF  .IRAB[IRB$V_UPDATE]
;  1399          1459  8                      AND
;  1400          1460  8                      .BDB[BDB$L_VBN] EQLU .IRAB[IRB$L_PUTUP_VBN]
;  1401          1461  8                  THEN
;  1402          1462  8                      NEED_RRV = .NEED_RRV - 1;
;  1403          1463  7                  END;
;  1404          1464  7
;  1405          1465  6              END;
;  1406          1466  6
;  1407          1467  6          IF .REC_ADDR GTRU .POS_INSERT
;  1408          1468  6          THEN
;  1409          1469  7              BEGIN
;  1410          1470  7              IRAB[IRB$V_REC_W_LO] = 1;
;  1411          1471  7
;  1412          1472  7              IF  .IRAB[IRB$V_UPDATE]
;  1413          1473  7                  AND
;  1414          1474  7                  .BDB[BDB$L_VBN] EQLU .IRAB[IRB$L_PUTUP_VBN]
;  1415          1475  7              THEN
;  1416          1476  7                  NEED_RRV = .NEED_RRV - 1;
;  1417          1477  6              END;
;  1418          1478  6
;  1419          1479  5          END;                                    ! {end of next }
;  1420          1480  4          END;          ! { end of scanning to find optimal split point }
;  1421          1481  4
;  1422          1482  3      END;                                        ! { end of half }
;  1423          1483  3
;  1424          1484  3      ! define a new block here so local storage can be redefined
;  1425          1485  3      !
;  1426          1486  4      BEGIN
;  1427          1487  4
;  1428          1488  4      MACRO
;  1429          1489  4          BEG_CHAIN = LHS %,
;  1430          1490  4          END_CHAIN = RHS %,
;  1431          1491  4          NUM_DUPS = NUM_RRVS %,
;  1432          1492  4          DUPS = RRV %;
;  1433          1493  4
;  1434          1494  4      BUILTIN
;  1435          1495  4          AP;
;  1436          1496  4
;  1437          1497  4      ! must be a 3 or 4 bucket split or we detected ascending order and the new
;  1438          1498  4      ! record was a dupe. we'll optimize here to the extent of trying to keep a
;  1439          1499  4      ! dup chain around the new record together and in the middle bucket
```

```
;  1440          1500  4      ! note that in all the cases that follow the new record is going into the
;  1441          1501  4      ! middle bucket. therefore, the "lhs" will always fit, since it can only
;  1442          1502  4      ! get smaller ( or stay the same size, in the degenerate case). also note
;  1443          1503  4      ! that in any of these case, the left hand bucket may be empty of data
;  1444          1504  4      ! records (have only rrv's in it) if the first split point is at the
;  1445          1505  4      ! beginning and all data records get moved out
;  1446          1506  4      !
;  1447          1507  4      IRAB[IRB$V_NEW_BKTS] = 2;   ! assume 3-bkt split until shown otherwise
;  1448          1508  4      IRAB[IRB$V_REC_W_LO] = 0;
;  1449          1509  4
;  1450          1510  4      ! initialize key buffer 2 with the contents of key buffer 3 (the value
;  1451          1511  4      ! of the primary key of the record being inserted).  This is necessary
;  1452          1512  4      ! when the new record is at the beginning of the bucket and is going into
;  1453          1513  4      ! a bucket all by itself and there were already 255 records in their
;  1454          1514  4      ! original bucket and they all need rrv's therefore they all move into the
;  1455          1515  4      ! next bucket. At any rate, that seems to be the only case where key buffer
;  1456          1516  4      ! 2 is not correct coming into here and will be set correctly before
;  1457          1517  4      ! leaving.
;  1458          1518  4      !
;  1459          1519  4      RM$MOVE(.IDX_DFN[IDX$B_KEYSZ], KEYBUF_ADDR(3), KEYBUF_ADDR(2));
;  1460          1520  4
;  1461          1521  4      ! find beginning and end of this possible dups chain equal to the key value
;  1462          1522  4      ! of the record being inserted.
;  1463          1523  4      !
;  1464          1524  4      REC_ADDR = .BKT_ADDR + BKT$C_OVERHDSZ;
;  1465          1525  5      BEGIN
;  1466          1526  5
;  1467          1527  5      LOCAL
;  1468          1528  5          STATUS;
;  1469          1529  5
;  1470          1530  5      WHILE STATUS = RM$COMPARE_REC(KEYBUF_ADDR(3), .IRAB[IRB$B_KEYSZ], 0)
;  1471          1531  5      DO
;  1472          1532  6          BEGIN
;  1473          1533  6
;  1474          1534  6          IF .REC_ADDR LSSU .POS_INSERT
;  1475          1535  6          THEN
;  1476          1536  7              BEGIN
;  1477          1537  7              AP = 0;
;  1478          1538  7              RM$RECORD_KEY(KEYBUF_ADDR(2));
;  1479          1539  6              END;
;  1480          1540  6
;  1481          1541  6          IF  .REC_ADDR EQLU .EOB
;  1482          1542  6              OR
;  1483          1543  6              .STATUS LSS 0
;  1484          1544  6          THEN
;  1485          1545  7              BEGIN
;  1486          1546  7
;  1487          1547  7              ! !!!! SPLIT TYPE 3 !!!! no duplicates found for simplicity, do a
;  1488          1548  7              ! 3-bkt split at the point of insert w/ new record in its own
;  1489          1549  7              ! bucket
;  1490          1550  7              !
;  1491          1551  7              IRAB[IRB$W_SPLIT] = IRAB[IRB$W_SPLIT_1] = .IRAB[IRB$W_POS_INS];
;  1492          1552  7              LEAVE DO_IT
;  1493          1553  7
;  1494          1554  7              ! { end of didn't find a duplicate, put record in its own bucket }
;  1495          1555  7              !
;  1496          1556  6              END;
```

```
;  1497          1557  6
;  1498          1558  6          RM$GETNEXT_REC();
;  1499          1559  5          END;                        ! { end of while no duplicate has been found }
;  1500          1560  5
;  1501          1561  4      END;                    ! { end of block defining status for while loop }
;  1502          1562  4
;  1503          1563  4      ! found the beginning of the dups chain, now find the end
;  1504          1564  4      !
;  1505          1565  4      NUM_DUPS = 0;
;  1506          1566  4      BEG_CHAIN = .REC_ADDR;
;  1507          1567  4
;  1508          1568  4      DO
;  1509          1569  5          BEGIN
;  1510          1570  5          NUM_DUPS = .NUM_DUPS + 1;
;  1511          1571  5          RM$GETNEXT_REC();
;  1512          1572  5
;  1513          1573  5          IF .REC_ADDR EQLU .EOB
;  1514          1574  5          THEN
;  1515          1575  5              EXITLOOP;
;  1516          1576  5
;  1517          1577  5          END
;  1518          1578  4      UNTIL RM$COMPARE_REC(KEYBUF_ADDR(3), .IRAB[IRB$B_KEYSZ], 0);
;  1519          1579  4                                      ! compare_rec returns 0 if keys match
;  1520          1580  4
;  1521          1581  4      END_CHAIN = .REC_ADDR;
;  1522          1582  4
;  1523          1583  4      ! found the beginning and the end of the chain calculate its size if we got
;  1524          1584  4      ! here via an update, we never called rm$srch_by_key to set dups_seen
;  1525          1585  4      ! for us.  so let us do that now if necessary
;  1526          1586  4      !
;  1527          1587  4
;  1528          1588  4      IF .POS_INSERT GTRU .BEG_CHAIN
;  1529          1589  4      THEN
;  1530          1590  4          IRAB[IRB$V_DUPS_SEEN] = 1;
;  1531          1591  4
;  1532          1592  4      DUPS = .END_CHAIN - .BEG_CHAIN;
;  1533          1593  4      DUPS = .DUPS + .REC_SIZE;
;  1534          1594  4
;  1535          1595  4      IF .DUPS LSSU .BKTSIZE
;  1536          1596  4
;  1537          1597  4          ! if there are 255 dupes on a put, there won't be enough id's in the
;  1538          1598  4          ! new bucket even if there is enough space for them.
;  1539          1599  4          !
;  1540          1600  4          AND
;  1541          1601  5              (.IRAB[IRB$V_UPDATE]
;  1542          1602  5              OR
;  1543          1603  5              .NUM_DUPS<0, 8> LEQU 254)
;  1544          1604  4      THEN
;  1545          1605  5          BEGIN
;  1546          1606  5
;  1547          1607  5          !+
;  1548          1608  5          ! !!!! SPLIT TYPE 1 !!!!!
;  1549          1609  5          ! duplicates found and fortunately, they all fit
;  1550          1610  5          ! in one bucket so 3-bkt split w/ all of the dups in the middle bucket
;  1551          1611  5          ! because of the optimization used for dups being inserted "in order"
;  1552          1612  5          ! this can sitll be a 2-bkt split if the new record is being inserted
;  1553          1613  5          ! at the end of the bucket
```

```
;  1554     1614  5          ! 22-jan-79  if loa forced us to think that a
;  1555     1615  5          ! bkt w/ all dups had to be split ( only on put) be smart and just put
;  1556     1616  5          ! new record by itself a better solution would be not to split at all,
;  1557     1617  5          ! but at this date it's rather inconceivable
;  1558     1618  5          ! 23-jan-79 it's not only loa
;  1559     1619  5          ! that can fool us, the bkt might have had a lot of rrv's
;  1560     1620  5          !-
;  1561     1621  5
;  1562     1622  5          IRAB[IRB$W_SPLIT] = .BEG_CHAIN - .BKT_ADDR;
;  1563     1623  5          IRAB[IRB$W_SPLIT_1] = .END_CHAIN - .BKT_ADDR;
;  1564     1624  5
;  1565     1625  5          IF .END_CHAIN EQLU .EOB
;  1566     1626  5          THEN
;  1567     1627  6              BEGIN
;  1568     1628  6              IRAB[IRB$V_NEW_BKTS] = 1;
;  1569     1629  6
;  1570     1630  7              IF .BEG_CHAIN EQLU (.BKT_ADDR + BKT$C_OVERHDSZ)
;  1571     1631  6              THEN
;  1572     1632  7                  BEGIN
;  1573     1633  7                  IRAB[IRB$W_SPLIT_1] = .IRAB[IRB$W_SPLIT_2];
;  1574     1634  7                  IRAB[IRB$W_SPLIT] = .IRAB[IRB$W_POS_INS];
;  1575     1635  7                  IRAB[IRB$V_CONT_BKT] = 1;
;  1576     1636  7                  END
;  1577     1637  7
;  1578     1638  6              END
;  1579     1639  5          ELSE
;  1580     1640  6              BEGIN
;  1581     1641  6
;  1582     1642  6              IF .IRAB[IRB$W_SPLIT] EQLU BKT$C_OVERHDSZ<0, 16>
;  1583     1643  6              THEN
;  1584     1644  6                  IRAB[IRB$V_EMPTY_BKT] = 1;
;  1585     1645  6
;  1586     1646  6              ! Only force the record into the low bucket if it is not the
;  1587     1647  6              ! first one in the duplicate chain.
;  1588     1648  6              !
;  1589     1649  6
;  1590     1650  6              IF .END_CHAIN GEQU .POS_INSERT
;  1591     1651  6                AND .IRAB[IRB$W_SPLIT] NEQU .IRAB[IRB$W_POS_INS]
;  1592     1652  6              THEN
;  1593     1653  6                  IRAB[IRB$V_REC_W_LO] = 1;
;  1594     1654  5              END;
;  1595     1655  5
;  1596     1656  5          LEAVE DO_IT
;  1597     1657  5
;  1598     1658  4          END;     ! { end of duplicates found and they fit in one bucket }
;  1599     1659  4
;  1600     1660  4      ! if we had 255 dupes above we dropped thru to here and this next test
;  1601     1661  4      ! will fail because it can only happen on an update so the all dupes case
;  1602     1662  4      ! will fall thru to split type 2, which will put the new record by itself.
;  1603     1663  4      ! consider oddball update case in which there are dups before and after
;  1604     1664  4      ! position of insert. ( note that if this case doesn't apply, the duplicates
;  1605     1665  4      ! were only before or after -- and didn't fit w/ record -- so new record
;  1606     1666  4      ! will end up by itself. for code flow purposes, leave that till later).
;  1607     1667  4      !
;  1608     1668  4
;  1609     1669  4      IF  .IRAB[IRB$V_DUPS_SEEN]
;  1610     1670  4          AND
```

```
;  1611     1671  4          .END_CHAIN GTRU .POS_INSERT
;  1612     1672  4      THEN
;  1613     1673  5          BEGIN
;  1614     1674  5
;  1615     1675  5          IF .DUPS - (.POS_INSERT - .BEG_CHAIN) LSSU .BKTSIZE
;  1616     1676  5          THEN
;  1617     1677  5
;  1618     1678  5              ! if high dups will fit w/ record, put them in a bucket together
;  1619     1679  5              !
;  1620     1680  6              BEGIN
;  1621     1681  6
;  1622     1682  6              !+
;  1623     1683  6              ! !!!! SPLIT TYPE 4 !!!!
;  1624     1684  6              ! 3 bkt split where middle bkt is a continuation bkt containing
;  1625     1685  6              ! new record and dups following it
;  1626     1686  6              !
;  1627     1687  6              ! !!!! AND SPLIT TYPE 4B !!!!! however, if the hi set consists
;  1628     1688  6              ! solely of duplicates, we can still have a 2-bkt split case that
;  1629     1689  6              ! would not have been picked up by the previous algorithm ( since
;  1630     1690  6              ! it won't divide dups).
;  1631     1691  6              !-
;  1632     1692  6
;  1633     1693  6              IRAB[IRB$V_CONT_BKT] = 1;
;  1634     1694  6              IRAB[IRB$W_SPLIT] = .IRAB[IRB$W_POS_INS];
;  1635     1695  6
;  1636     1696  6              IF .END_CHAIN EQLU .EOB
;  1637     1697  6              THEN
;  1638     1698  6                  IRAB[IRB$V_NEW_BKTS] = 1
;  1639     1699  6              ELSE
;  1640     1700  6                  IRAB[IRB$W_SPLIT_1] = .END_CHAIN - .BKT_ADDR;
;  1641     1701  6
;  1642     1702  6              REC_ADDR = .BEG_CHAIN;
;  1643     1703  6              AP = 0;
;  1644     1704  6              RM$RECORD_KEY(KEYBUF_ADDR(2));
;  1645     1705  6              LEAVE DO_IT
;  1646     1706  6
;  1647     1707  5              END;
;  1648     1708  5
;  1649     1709  5          ! try to fit new record w/ before-dups in middle bucket
;  1650     1710  5          !
;  1651     1711  5
;  1652     1712  5          IF .DUPS - ( END_CHAIN - .POS_INSERT) LSSU .BKTSIZE
;  1653     1713  5          THEN
;  1654     1714  6              BEGIN
;  1655     1715  6
;  1656     1716  6              !+
;  1657     1717  6              ! !!!! SPLIT TYPE 5 !!!!!
;  1658     1718  6              ! 3 or 4 bkt split ( depending on status of
;  1659     1719  6              ! high set) where left-middle bkt is new record w/ before-dups
;  1660     1720  6              ! and right-middle bkt, if it is needed, is a continuation bkt
;  1661     1721  6              ! w/ the after-dups. it is needed iff the dups aren't the whole hi
;  1662     1722  6              ! set it still is a continuation bkt.
;  1663     1723  6              !
;  1664     1724  6              ! ***** NOTE FROM NOV-7-78
;  1665     1725  6              ! This case doesn't take into account the fact that the
;  1666     1726  6              ! whole bucket may be dups.  In the case of all dups, we could
;  1667     1727  6              ! end up generating an empty bucket when we don't have to (if
```

```
;  1668     1728  6              ! no RRV's) or a relatively useless bucket (some RRV's). In any
;  1669     1729  6              ! event we could end up generating an extra bucket when we
;  1670     1730  6              ! don't have to
;  1671     1731  6              !-
;  1672     1732  6
;  1673     1733  6              IRAB[IRB$W_SPLIT] = .BEG_CHAIN - .BKT_ADDR;
;  1674     1734  6              IRAB[IRB$W_SPLIT_1] = .IRAB[IRB$W_POS_INS];
;  1675     1735  6
;  1676     1736  6              IF .IRAB[IRB$W_SPLIT] EQLU BKT$C_OVERHDSZ<0, 16>
;  1677     1737  6              THEN
;  1678     1738  6                  IRAB[IRB$V_EMPTY_BKT] = 1;
;  1679     1739  6
;  1680     1740  6              IRAB[IRB$V_REC_W_LO] = 1;
;  1681     1741  6
;  1682     1742  6              IF .END_CHAIN LSSU .EOB
;  1683     1743  6              THEN
;  1684     1744  7                  BEGIN
;  1685     1745  7                  IRAB[IRB$V_NEW_BKTS] = 3;
;  1686     1746  7                  IRAB[IRB$W_SPLIT_2] = .END_CHAIN - .BKT_ADDR;
;  1687     1747  7                  END
;  1688     1748  6              ELSE
;  1689     1749  6                  IRAB[IRB$V_CONT_R] = 1;
;  1690     1750  6
;  1691     1751  6              LEAVE DO_IT
;  1692     1752  6
;  1693     1753  5              END;
;  1694     1754  5
;  1695     1755  5          ! { end of oddball update case w/ dups on both sides of new record }
;  1696     1756  5          !
;  1697     1757  4          END;
;  1698     1758  4
;  1699     1759  4          !+
;  1700     1760  4          ! !!!! SPLIT TYPE 2 !!!!!
;  1701     1761  4          ! the new record must go all by itself therefore,
;  1702     1762  4          ! this is a 3-bkt split if there are no after-dups or no hi set and a 4-bkt
;  1703     1763  4          ! split if both of those exist even more exceptional, this can still be a
;  1704     1764  4          ! 2-bkt split if there is no hi set at all ---- i.e., eob = end of the dups
;  1705     1765  4          ! chain
;  1706     1766  4          !-
;  1707     1767  4
;  1708     1768  4          IRAB[IRB$W_SPLIT] = IRAB[IRB$W_SPLIT_1] = .IRAB[IRB$W_POS_INS];
;  1709     1769  4
;  1710     1770  4          IF  .IRAB[IRB$V_DUPS_SEEN]
;  1711     1771  4          THEN
;  1712     1772  5              BEGIN
;  1713     1773  5              IRAB[IRB$V_CONT_BKT] = 1;
;  1714     1774  5              REC_ADDR = .BEG_CHAIN;
;  1715     1775  5              AP = 0;
;  1716     1776  5              RM$RECORD_KEY(KEYBUF_ADDR(2));
;  1717     1777  4              END;
;  1718     1778  4
;  1719     1779  4          IF .POS_INSERT EQLU .EOB
;  1720     1780  4          THEN
;  1721     1781  4              IRAB[IRB$V_NEW_BKTS] = 1
;  1722     1782  4          ELSE
;  1723     1783  4
;  1724     1784  4              IF .POS_INSERT LSSU .END_CHAIN
```

```
;  1725          1785  4          THEN
;  1726          1786  5              BEGIN
;  1727          1787  5
;  1728          1788  5              IF .END_CHAIN LSSU .EOB
;  1729          1789  5              THEN
;  1730          1790  5                  IRAB[IRB$V_NEW_BKTS] = 3
;  1731          1791  5              ELSE
;  1732          1792  5                  IRAB[IRB$V_CONT_R] = 1;
;  1733          1793  5
;  1734          1794  5              IRAB[IRB$W_SPLIT_2] = .END_CHAIN - .BKT_ADDR;
;  1735          1795  4              END;
;  1736          1796  4
;  1737          1797  3          END;                          ! { end of block defining local symbols }
;  1738          1798  3
;  1739          1799  2          END;                                              ! { end of do_it }
;  1740          1800  2
;  1741          1801  2          ! if the first split point is at the beginning of the data, this means that
;  1742          1802  2          ! all data records will be moved out and only rrv's will be left in the
;  1743          1803  2          ! original bucket ..... therefore, we can mark this bucket as empty
;  1744          1804  2          !
;  1745          1805  2
;  1746          1806  2          IF  .IRAB[IRB$W_SPLIT] EQLU BKT$C_OVERHDSZ<0, 16>
;  1747          1807  2              AND
;  1748          1808  2              NOT .IRAB[IRB$V_REC_W_LO]
;  1749          1809  2          THEN
;  1750          1810  2              IRAB[IRB$V_EMPTY_BKT] = 1;
;  1751          1811  2
;  1752          1812  2          RETURN;
;  1753          1813  2
;  1754          1814  1          END;                                              ! { end of routine }


                                        0C  BB 00000 RM$SPLIT_UDR::
                                                     PUSHR   #^M<R2,R3>                          ; 0874
                                    5E              20  C2 00002         SUBL2   #32, SP                     ;
                                 44     A9      48  8F  8A 00005         BICB2   #72, 68(IRAB)               ; 0986
     44   A9                 02         01          01  F0 0000A         INSV    #1, #1, #2, 68(IRAB)        ; 0987
                                                      7E  D4 00010         CLRL    NUM_RRVS                    ; 0988
                                                      56  DD 00012         PUSHL   REC_ADDR                    ; 0989
                                    52              0E  A5  9E 00014         MOVAB   14(R5), R2                  ; 0990
                                    56                  52  D0 00018         MOVL    R2, REC_ADDR            ;
                                    50              04  A5  3C 0001B         MOVZWL  4(BKT_ADDR), R0             ; 0991
                                                    6045  9F 0001F         PUSHAB  (R0)[BKT_ADDR]              ;
                                                      7E  D4 00022         CLRL    LAST                        ; 0992
                                 20                 66  03  E0 00024 1$:     BBS     #3, (REC_ADDR), 4$          ; 1000
                                    5C              03  D0 00028         MOVL    #3, AP                      ; 1004
                                                  0000G 30 0002B         BSBW    RM$RECORD_VBN               ; 1006
                                    50              1C  A4  D1 0002E         CMPL    28(BDB), R0             ;
                                                      05  12 00032         BNEQ    2$                      ;
                                                    0C  AE  B6 00034         INCW    NUM_RRVS                    ; 1008
                                                      03  11 00037         BRB     3$                      ;
                                                    0E  AE  B6 00039 2$:     INCW    NUM_RRVS+2                  ; 1018
                                    6E                  56  D0 0003C 3$:     MOVL    REC_ADDR, LAST          ; 1020
                                                  0000G 30 0003F         BSBW    RM$GETNEXT_REC              ; 1021
```

```
                                04   AE          56   D1 00042          CMPL    REC_ADDR, EOB                                   ; 1023
                                             DC   1F 00046          BLSSU   1$
                          50                 56   C3 00048 4$:       SUBL3   BKT_ADDR, REC_ADDR, R0                          ; 1029
                                4E   A9       50   B0 0004C          MOVW    R0, 78(IRAB)
                                4C   A9       50   B0 00050          MOVW    R0, 76(IRAB)
                                             51   17 A7 9A 00054      MOVZBL  23(IDX_DFN), R1                                 ; 1030
                          51                 51   09 78 00058          ASHL    #9, R1, R1
                                1C   AE    F1   A1 9E 0005C          MOVAB   -15(R1), BKTSIZE
                                2C   AE    22   A8 3C 00061          MOVZWL  34(RAB), REC_SIZE                               ; 1032
                                2C   AE       07   C0 00066          ADDL2   #7, REC_SIZE
                                             01   50 AA 91 0006A      CMPB    80(IFAB), #1                                    ; 1034
                                                04   13 0006E          BEQL    5$
                                2C   AE       02   C0 00070          ADDL2   #2, REC_SIZE                                    ; 1036
                          0A                 06   A9 03   E1 00074 5$:       BBC     #3, 6(IRAB), 6$                         ; 1040
                                78   A9    1C   A4 D1 00079          CMPL    28(BDB), 120(IRAB)                              ; 1044
                                             03   12 0007E          BNEQ    6$
                                0C   AE       B6   00080          INCW    NUM_RRVS                                        ; 1046
                    20   AE       04   AE       56   C3 00083 6$:       SUBL3   REC_ADDR, EOB, RRV                          ; 1050
                                04   AE       56   D0 00089          MOVL    REC_ADDR, EOB                                   ; 1051
                                52            56   D1 0008D          CMPL    REC_ADDR, R2                                    ; 1056
                                             03   12 00090          BNEQ    7$
                                          02FD   31 00092          BRW     47$
                                56       08   AE   D1 00095 7$:       CMPL    POS_INSERT, REC_ADDR                            ; 1087
                                             3B   12 00099          BNEQ    9$
                          51                 6E   01   C1 0009B          ADDL3   #1, LAST, R1                                ; 1089
                                50            61   9A 0009F          MOVZBL  (R1), R0
                                             50   D6 000A2          INCL    R0
                                06   A5       50   91 000A4          CMPB    R0, 6(BKT_ADDR)
                                             2C   12 000A8          BNEQ    9$
                                56            6E   D0 000AA          MOVL    LAST, REC_ADDR                                  ; 1092
                                             7E   D4 000AD          CLRL    -(SP)                                           ; 1094
                                7E        20   A7   9A 000AF          MOVZBL  32(IDX_DFN), -(SP)
                                50      00B4   CA   3C 000B3          MOVZWL  180(IFAB), R0
                                      60 B940   3F   000B8          PUSHAW  @96(IRAB)[R0]
                                      0000G   30   000BC          BSBW    RM$COMPARE_REC
                                5E            0C   C0 000BF          ADDL2   #12, SP
                                03            50   E8 000C2          BLBS    R0, 8$
                                          025B   31 000C5          BRW     43$
                                50            56   D0 000C8 8$:       MOVL    REC_ADDR, R0                                    ; 1103
                                          FEA9   30 000CB          BSBW    RM$MOVE_KEY
                                4A   A9       48   A9 B0 000CE          MOVW    72(IRAB), 74(IRAB)                              ; 1104
                                          044B   31 000D3          BRW     72$                                             ; 1105
                                56            0E   A5 9E 000D6 9$:       MOVAB   14(R5), REC_ADDR                            ; 1116
                                24   AE 7FFFFFFF   8F   D0 000DA          MOVL    #2147483647, LAST_DIFF                          ; 1117
                                             6E   D4 000E2          CLRL    LAST                                            ; 1118
                                          28   AE   D4 000E4          CLRL    SAVE_REC_W_LO                                   ; 1119
                    10   AE       04   AE          56   C3 000E7 10$:       SUBL3   REC_ADDR, EOB, RHS                          ; 1131
                                08   AE          56   D1 000ED          CMPL    REC_ADDR, POS_INSERT                            ; 1133
                                             0A   1A 000F1          BGTRU   11$
                          05        44   A9       03   E0 000F3          BBS     #3, 68(IRAB), 11$                           ; 1135
                                10   AE    2C   AE   C0 000F8          ADDL2   REC_SIZE, RHS                                   ; 1137
                                1C   AE    10   AE   D1 000FD 11$:       CMPL    RHS, BKTSIZE                                    ; 1146
                                             03   1F 00102          BLSSU   13$
                                          0139   31 00104 12$:       BRW     32$
                                          06   A5   95 00107 13$:       TSTB    6(BKT_ADDR)                                     ; 1148
                                             13   12 0010A          BNEQ    14$
                          0E        06   A9          03   E0 0010C          BBS     #3, 6(IRAB), 14$                            ; 1150
```

```
                50      0E  A5  9E 00111          MOVAB   14(R5), R0                          ; 1152
                50          56  D1 00115          CMPL    REC_ADDR, R0                        ;
                            05  12 00118          BNEQ    14$                                 ;
          E5    44  A9      03  E1 0011A          BBC     #3, 68(IRAB), 12$                   ; 1154
          50        56      55  C3 0011F  14$:    SUBL3   BKT_ADDR, REC_ADDR, R0              ; 1157
                    52  F2  A0  9E 00123          MOVAB   -14(R0), LHS                        ;
                08  AE          56  D1 00127          CMPL    REC_ADDR, POS_INSERT                ; 1159
                            09  1F 0012B          BLSSU   15$                                 ;
          04    44  A9      03  E1 0012D          BBC     #3, 68(IRAB), 15$                   ; 1161
                    52  2C  AE  C0 00132          ADDL2   REC_SIZE, LHS                       ; 1163
          50        52  20  AE  C1 00136  15$:    ADDL3   RRV, LHS, R0                        ; 1172
                    51  0C  AE  3C 0013B          MOVZWL  NUM_RRVS, R1                        ;
    14  AE          51      07  C5 0013F          MULL3   #7, R1, 20(SP)                      ;
                    50  14  AE  C0 00144          ADDL2   20(SP), R0                          ;
                1C  AE      50  D1 00148          CMPL    R0, BKTSIZE                         ;
                            14  1A 0014C          BGTRU   16$                                 ;
                        06  A5  95 0014E          TSTB    6(BKT_ADDR)                         ; 1180
                            38  12 00151          BNEQ    20$                                 ;
          33    06  A9      03  E0 00153          BBS     #3, 6(IRAB), 20$                    ; 1182
                        0E  AE  B5 00158          TSTW    NUM_RRVS+2                          ; 1184
                            2E  12 0015B          BNEQ    20$                                 ;
          29    44  A9      03  E1 0015D          BBC     #3, 68(IRAB), 20$                   ; 1186
                            6E  D5 00162  16$:    TSTL    LAST                                ; 1190
                            03  12 00164          BNEQ    17$                                 ;
                          01BA  31 00166          BRW     43$                                 ;
                    56      6E  D0 00169  17$:    MOVL    LAST, REC_ADDR                      ; 1194
                    04  28  AE  E8 0016C          BLBS    SAVE_REC_W_LO, 18$                  ; 1196
                    44  A9  08  8A 00170          BICB2   #8, 68(IRAB)                        ; 1198
                    53      56  D0 00174  18$:    MOVL    REC_ADDR, TMP                       ; 1209
                    56  0E  A5  9E 00177          MOVAB   14(R5), REC_ADDR                    ; 1210
                    6E      56  D0 0017B          MOVL    REC_ADDR, LAST                      ; 1211
                    53      56  D1 0017E  19$:    CMPL    REC_ADDR, TMP                       ; 1213
                            6C  13 00181          BEQL    28$                                 ;
                6E          56  D0 00183          MOVL    REC_ADDR, LAST                      ; 1216
                         0000G  30 00186          BSBW    RM$GETNEXT_REC                      ; 1217
                            F3  11 00189          BRB     19$                                 ; 1213
          50        52  1C  AE  C5 0018B  20$:    MULL3   BKTSIZE, LHS, R0                    ; 1257
          51    1C  AE  14  AE  C3 00190          SUBL3   20(SP), BKTSIZE, R1                 ;
          51    20  AE      51  C3 00196          SUBL3   R1, RRV, R1                         ; 1258
                    51  10  AE  C4 0019B          MULL2   RHS, R1                             ; 1257
    18  AE          50      51  C1 0019F          ADDL3   R1, R0, DIFFERENCE                  ;
                            03  18 001A4          BGEQ    21$                                 ; 1260
                          0086  31 001A6          BRW     31$                                 ;
                51      18  AE  D0 001A9  21$:    MOVL    DIFFERENCE, R1                      ; 1268
                            03  18 001AD          BGEQ    22$                                 ;
                51          51  CE 001AF          MNEGL   R1, R1                              ;
                50      24  AE  D0 001B2  22$:    MOVL    LAST_DIFF, R0                       ;
                            03  18 001B6          BGEQ    23$                                 ;
                50          50  CE 001B8          MNEGL   R0, R0                              ;
                50          51  D1 001BB  23$:    CMPL    R1, R0                              ;
                            14  1B 001BE          BLEQU   26$                                 ;
                6E          56  D1 001C0          CMPL    REC_ADDR, LAST                      ; 1272
                            09  13 001C3          BEQL    24$                                 ;
                56          6E  D0 001C5          MOVL    LAST, REC_ADDR                      ; 1276
          08    AE          56  D1 001C8          CMPL    REC_ADDR, POS_INSERT                ; 1278
                            04  1E 001CC          BGEQU   25$                                 ;
          44    A9          08  8A 001CE  24$:    BICB2   #8, 68(IRAB)                        ; 1280
```

```
                                                6E  D4 001D2 25$:    CLRL    LAST                                    ; 1282
                                                6E  D5 001D4 26$:    TSTL    LAST                                    ; 1289
                                                17  12 001D6          BNEQ    28$                                     ;
                                    53          56  D0 001D8          MOVL    REC_ADDR, TMP                           ; 1296
                                    56      0E  A5  9E 001DB          MOVAB   14(R5), REC_ADDR                        ; 1297
                                    6E          56  D0 001DF          MOVL    REC_ADDR, LAST                          ; 1298
                                    53          56  D1 001E2 27$:    CMPL    REC_ADDR, TMP                           ; 1300
                                                08  13 001E5          BEQL    28$                                     ;
                                    6E          56  D0 001E7          MOVL    REC_ADDR, LAST                          ; 1303
                                            0000G   30 001EA          BSBW    RM$GETNEXT_REC                          ; 1304
                                                F3  11 001ED          BRB     27$                                     ; 1300
                                    50          6E  D0 001EF 28$:     MOVL    LAST, R0                                ; 1309
                                            FD82    30 001F2          BSBW    RM$MOVE_KEY                             ;
                                4A  A9          56  55 A3 001F5       SUBW3   BKT_ADDR, REC_ADDR, 74(IRAB)            ; 1310
                                48  A9      4A  A9  B1 001FA          CMPW    74(IRAB), 72(IRAB)                      ; 1316
                                                0C  12 001FF          BNEQ    29$                                     ;
                                4C  A9      4A  A9  B1 00201          CMPW    74(IRAB), 76(IRAB)                      ; 1319
                                                05  12 00206          BNEQ    29$                                     ;
                            03  44  A9          03  E1 00208          BBC     #3, 68(IRAB), 30$                       ; 1322
                                            0311    31 0020D 29$:     BRW     72$                                     ;
                                    5C          05  D0 00210 30$:     MOVL    #3, AP                                  ; 1329
                                    50      00B4    CA 3C 00213       MOVZWL  180(IFAB), R0                           ; 1332
                                    53      60  B940    3E 00218      MOVAW   @96(IRAB)[R0], R3                       ;
                            51              50  60  A9  C1 0021D      ADDL3   96(IRAB), R0, R1                        ; 1331
                                    50      20  A7  9A 00222          MOVZBL  32(IDX_DFN), R0                         ;
                                            0000G   30 00226          BSBW    RM$COMPARE_KEY                          ;
                                    E1          50  E8 00229          BLBS    R0, 29$                                 ;
                                            01EF    31 0022C          BRW     55$                                     ; 1335
                                24  AE      18  AE  D0 0022F 31$:     MOVL    DIFFERENCE, LAST_DIFF                   ; 1346
                                    6E          56  D0 00234          MOVL    REC_ADDR, LAST                          ; 1347
                            04      44  A9          03  E1 00237      BBC     #3, 68(IRAB), 32$                       ; 1349
                                28  AE              01  D0 0023C      MOVL    #1, SAVE_REC_W_LO                       ; 1351
                                08  AE              56  D1 00240 32$:     CMPL    REC_ADDR, POS_INSERT                ; 1360
                                                    3F  12 00244      BNEQ    36$                                     ;
                            3A      44  A9          03  E0 00246      BBS     #3, 68(IRAB), 36$                       ; 1362
                            0A      06  A9          03  E1 0024B      BBC     #3, 6(IRAB), 33$                        ; 1369
                                78  A9      1C  A4  D1 00250          CMPL    28(BDB), 120(IRAB)                      ; 1373
                                                03  12 00255          BNEQ    33$                                     ;
                                            0C  AE  B7 00257          DECW    NUM_RRVS                                ; 1375
                                44  A9              08  88 0025A 33$:     BISB2   #8, 68(IRAB)                        ; 1379
                                    50              56  D0 0025E      MOVL    REC_ADDR, R0                            ; 1380
                                            FD13    30 00261          BSBW    RM$MOVE_KEY                             ;
                                04  AE              56  D1 00264      CMPL    REC_ADDR, EOB                           ; 1382
                                                    03  12 00268      BNEQ    35$                                     ;
                                            FE7A    31 0026A 34$:     BRW     10$                                     ;
                                                    7E  D4 0026D 35$:     CLRL    -(SP)                               ; 1386
                                    7E      20  A7  9A 0026F          MOVZBL  32(IDX_DFN), -(SP)                      ;
                                    50      00B4    CA 3C 00273       MOVZWL  180(IFAB), R0                           ;
                                            60  B940    9F 00278      PUSHAB  @96(IRAB)[R0]                           ;
                                            0000G   30 0027C          BSBW    RM$COMPARE_REC                          ;
                                    5E          0C  C0 0027F          ADDL2   #12, SP                                 ;
                                    E5          50  E8 00282          BLBS    R0, 34$                                 ;
                                    53      44  A9  90 00285 36$:     MOVB    68(IRAB), TMP                           ; 1401
                                44  A9              08  8A 00289      BICB2   #8, 68(IRAB)                            ; 1402
                                    50              56  D0 0028D      MOVL    REC_ADDR, R0                            ; 1403
                                            FCE4    30 00290          BSBW    RM$MOVE_KEY                             ;
                                44  A9              53  90 00293      MOVB    TMP, 68(IRAB)                           ; 1404
```

```
                                04   AE          56   D1 00297 37$:    CMPL    REC_ADDR, EOB                   ; 1413
                                             35   13 0029B          BEQL    40$
                                5C           03   D0 0029D          MOVL    #3, AP                          ; 1417
                                         0000G   30 002A0          BSBW    RM$RECORD_VBN                   ; 1419
                                50       1C      A4   D1 002A3          CMPL    28(BDB), R0
                                             05   12 002A7          BNEQ    38$
                                         0C      AE   B7 002A9          DECW    NUM_RRVS                        ; 1421
                                             03   11 002AC          BRB     39$
                                         0E      AE   B7 002AE 38$:     DECW    NUM_RRVS+2                      ; 1423
                                         0000G   30 002B1 39$:     BSBW    RM$GETNEXT_REC                  ; 1425
                                04   AE          56   D1 002B4          CMPL    REC_ADDR, EOB                   ; 1427
                                             18   13 002B8          BEQL    40$
                                7E           7E   D4 002BA          CLRL    -(SP)                           ; 1435
                                7E       20      A7   9A 002BC          MOVZBL  32(IDX_DFN), -(SP)
                                50       00B4    CA   3C 002C0          MOVZWL  180(IFAB), R0
                                         60  B940  9F   9F 002C5          PUSHAB  @96(IRAB)[R0]
                                         0000G   30 002C9          BSBW    RM$COMPARE_REC
                                5E           0C   C0 002CC          ADDL2   #12, SP
                                C5           50   E9 002CF          BLBC    R0, 37$
                                08   AE          56   D1 002D2 40$:     CMPL    REC_ADDR, POS_INSERT            ; 1442
                                             2F   12 002D6          BNEQ    41$
                                5C           03   D0 002D8          MOVL    #3, AP                          ; 1449
                                50       00B4    CA   3C 002DB          MOVZWL  180(IFAB), R0                   ; 1452
                                         60  B940  3E   3E 002E0          MOVAW   @96(IRAB)[R0], R3
                           51            50   60  A9   C1 002E5          ADDL3   96(IRAB), R0, R1                ; 1451
                                         50   20  A7   9A 002EA          MOVZBL  32(IDX_DFN), R0
                                         0000G   30 002EE          BSBW    RM$COMPARE_KEY
                                13           50   E8 002F1          BLBS    R0, 41$
                                44   A9          08   88 002F4          BISB2   #8, 68(IRAB)                    ; 1456
                           0A   06   A9          03   E1 002F8          BBC     #3, 6(IRAB), 41$                ; 1458
                                78   A9      1C      A4   D1 002FD          CMPL    28(BDB), 120(IRAB)              ; 1460
                                             03   12 00302          BNEQ    41$
                                         0C      AE   B7 00304          DECW    NUM_RRVS                        ; 1462
                                08   AE          56   D1 00307 41$:     CMPL    REC_ADDR, POS_INSERT            ; 1467
                                             13   1B 0030B          BLEQU   42$
                                44   A9          08   88 0030D          BISB2   #8, 68(IRAB)                    ; 1470
                           0A   06   A9          03   E1 00311          BBC     #3, 6(IRAB), 42$                ; 1472
                                78   A9      1C      A4   D1 00316          CMPL    28(BDB), 120(IRAB)              ; 1474
                                             03   12 0031B          BNEQ    42$
                                         0C      AE   B7 0031D          DECW    NUM_RRVS                        ; 1476
                                         FDC4    31   31 00320 42$:     BRW     10$                             ; 1128
44   A9                    02                01   02   F0 00323 43$:     INSV    #2, #1, #2, 68(IRAB)            ; 1507
                                44   A9          08   8A 00329          BICB2   #8, 68(IRAB)                    ; 1508
                                50       00B4    CA   3C 0032D          MOVZWL  180(IFAB), R0                   ; 1519
                                         60  B940  9F   9F 00332          PUSHAB  @96(IRAB)[R0]
                                         60  B940  3F   3F 00336          PUSHAW  @96(IRAB)[R0]
                                7E       20      A7   9A 0033A          MOVZBL  32(IDX_DFN), -(SP)
                                         0000G   30 0033E          BSBW    RM$MOVE
                                5E           0C   C0 00341          ADDL2   #12, SP
                                28   AE      0E      A5   9E 00344          MOVAB   14(R5), 40(SP)                  ; 1524
                                56       28      AE   D0 00349          MOVL    40(SP), REC_ADDR
                                             7E   D4 0034D 44$:     CLRL    -(SP)                           ; 1530
                                7E       00A6    C9   9A 0034F          MOVZBL  166(IRAB), -(SP)
                                50       00B4    CA   3C 00354          MOVZWL  180(IFAB), R0
                                         60  B940  3F   3F 00359          PUSHAW  @96(IRAB)[R0]
                                         0000G   30 0035D          BSBW    RM$COMPARE_REC
                                5E           0C   C0 00360          ADDL2   #12, SP
```

```
                         53         50  D0 00363       MOVL    R0, STATUS                       ;
                         35         53  E9 00366       BLBC    STATUS, 50$                      ;
                  08     AE         56  D1 00369       CMPL    REC_ADDR, POS_INSERT             ; 1534
                                    11  1E 0036D       BGEQU   45$                              ;
                                    5C  D4 0036F       CLRL    AP                               ; 1537
                         50  00B4   CA  3C 00371       MOVZWL  180(IFAB), R0                    ; 1538
                               60 B940  9F 00376       PUSHAB  @96(IRAB)[R0]                    ;
                                 0000G  30 0037A       BSBW    RM$RECORD_KEY                    ;
                         5E         04  C0 0037D       ADDL2   #4, SP                           ;
                  04     AE         56  D1 00380 45$:  CMPL    REC_ADDR, EOB                    ; 1541
                                    04  13 00384       BEQL    46$                              ;
                                    53  D5 00386       TSTL    STATUS                           ; 1543
                                    0F  18 00388       BGEQ    49$                              ;
                         50    48   A9  3C 0038A 46$:  MOVZWL  72(IRAB), R0                     ; 1551
                  4C     A9         50  B0 0038E       MOVW    R0, 76(IRAB)                     ;
                  4A     A9         50  B0 00392 47$:  MOVW    R0, 74(IRAB)                     ;
                                  0188  31 00396 48$:  BRW     72$                              ; 1552
                                 0000G  30 00399 49$:  BSBW    RM$GETNEXT_REC                   ; 1558
                                    AF  11 0039C       BRB     44$                              ; 1530
                               0C   AE  D4 0039E 50$:  CLRL    NUM_RRVS                         ; 1565
                         52         56  D0 003A1       MOVL    REC_ADDR, LHS                    ; 1566
                               0C   AE  D6 003A4 51$:  INCL    NUM_RRVS                         ; 1570
                                 0000G  30 003A7       BSBW    RM$GETNEXT_REC                   ; 1571
                  04     AE         56  D1 003AA       CMPL    REC_ADDR, EOB                    ; 1573
                                    19  13 003AE       BEQL    52$                              ;
                                    7E  D4 003B0       CLRL    -(SP)                            ; 1578
                         7E  00A6   C9  9A 003B2       MOVZBL  166(IRAB), -(SP)                 ;
                         50  00B4   CA  3C 003B7       MOVZWL  180(IFAB), R0                    ;
                               60 B940  3F 003BC       PUSHAW  @96(IRAB)[R0]                    ;
                                 0000G  30 003C0       BSBW    RM$COMPARE_REC                   ;
                         5E         0C  C0 003C3       ADDL2   #12, SP                          ;
                         DB         50  E9 003C6       BLBC    R0, 51$                          ;
                  10     AE         56  D0 003C9 52$:  MOVL    REC_ADDR, RHS                    ; 1581
                         52    08   AE  D1 003CD       CMPL    POS_INSERT, LHS                  ; 1588
                                    05  1B 003D1       BLEQU   53$                              ;
                  44     A9    80   8F  88 003D3       BISB2   #128, 68(IRAB)                   ; 1590
           20     AE  10 AE         52  C3 003D8 53$:  SUBL3   LHS, RHS, RRV                    ; 1592
                  20     AE    2C   AE  C0 003DE       ADDL2   REC_SIZE, RRV                    ; 1593
                  1C     AE    20   AE  D1 003E3       CMPL    RRV, BKTSIZE                     ; 1595
                                    59  1E 003E8       BGEQU   58$                              ;
           07     06     A9         03  E0 003EA       BBS     #3, 6(IRAB), 54$                 ; 1601
                  FE     8F    0C   AE  91 003EF       CMPB    NUM_RRVS, #254                   ; 1603
                                    4D  1A 003F4       BGTRU   58$                              ;
    4A     A9            52         55  A3 003F6 54$:  SUBW3   BKT_ADDR, LHS, 74(IRAB)          ; 1622
    4C     A9     10     AE         55  A3 003FB       SUBW3   BKT_ADDR, RHS, 76(IRAB)          ; 1623
                  04     AE    10   AE  D1 00401       CMPL    RHS, EOB                         ; 1625
                                    1C  12 00406       BNEQ    56$                              ;
44  A9     02            01             F0 00408       INSV    #1, #1, #2, 68(IRAB)             ; 1628
                  28     AE         52  D1 0040E       CMPL    LHS, 40(SP)                      ; 1630
                                    82  12 00412       BNEQ    48$                              ;
                  4C     A9    4E   A9  B0 00414       MOVW    78(IRAB), 76(IRAB)               ; 1633
                  4A     A9    48   A9  B0 00419       MOVW    72(IRAB), 74(IRAB)               ; 1634
                  44     A9         10  88 0041E 55$:  BISB2   #16, 68(IRAB)                    ; 1635
                                    6F  11 00422       BRB     62$                              ; 1630
                         0E    4A   A9  B1 00424 56$:  CMPW    74(IRAB), #14                    ; 1642
                                    05  12 00428       BNEQ    57$                              ;
                  44     A9    40   8F  88 0042A       BISB2   #64, 68(IRAB)                    ; 1644
```

```
                                08    AE          10    AE    D1 0042F 57$:     CMPL    RHS, POS_INSERT                          ; 1650
                                                        5D    1F 00434          BLSSU   62$                                      ;
                                48    A9          4A    A9    B1 00436          CMPW    74(IRAB), 72(IRAB)                       ; 1651
                                                        56    13 0043B          BEQL    62$                                      ;
                                44    A9                08    88 0043D          BISB2   #8, 68(IRAB)                             ; 1653
                                                        50    11 00441          BRB     62$                                      ; 1656
                                                  44    A9    95 00443 58$:     TSTB    68(IRAB)                                 ; 1669
                                                        03    19 00446          BLSS    59$                                      ;
                                                  0080  31    00448             BRW     65$                                      ;
                                08    AE          10    AE    D1 0044B 59$:     CMPL    RHS, POS_INSERT                          ; 1671
                                                        79    1B 00450          BLEQU   65$                                      ;
                          50                    52    08    AE    C3 00452          SUBL3   POS_INSERT, LHS, R0              ; 1675
                                                50    20    AE    C0 00457          ADDL2   RRV, R0                                  ;
                                1C    AE              50    D1 0045B             CMPL    R0, BKTSIZE                              ;
                                                        34    1E 0045F          BGEQU   63$                                      ;
                                44    A9              10    88 00461          BISB2   #16, 68(IRAB)                            ; 1693
                                4A    A9        48    A9    B0 00465          MOVW    72(IRAB), 74(IRAB)                       ; 1694
                                04    AE        10    AE    D1 0046A          CMPL    RHS, EOB                                 ; 1696
                                                        08    12 0046F          BNEQ    60$                                      ;
    44    A9                02                  01          01    F0 00471    INSV    #1, #1, #2, 68(IRAB)                     ; 1698
                                                        06    11 00477          BRB     61$                                      ;
                    4C    A9        10    AE          55    A3 00479 60$:     SUBW3   BKT_ADDR, RHS, 76(IRAB)                  ; 1700
                                56                  52    D0 0047F 61$:     MOVL    LHS, REC_ADDR                            ; 1702
                                                    5C    D4 00482          CLRL    AP                                       ; 1703
                                      50      00B4  CA    3C 00484          MOVZWL  180(IFAB), R0                            ; 1704
                                            60  B940  9F    00489             PUSHAB  @96(IRAB)[R0]                            ;
                                          0000G 30    0048D                 BSBW    RM$RECORD_KEY                            ;
                                      5E            04    C0 00490          ADDL2   #4, SP                                   ;
                                                    6C    11 00493 62$:     BRB     67$                                      ; 1705
                          50        08    AE          10    AE    C3 00495 63$:     SUBL3   RHS, POS_INSERT, R0              ; 1712
                                                50    20    AE    C0 0049B          ADDL2   RRV, R0                                  ;
                                1C    AE              50    D1 0049F             CMPL    R0, BKTSIZE                              ;
                                                        26    1E 004A3          BGEQU   65$                                      ;
    4A    A9                      52                  55    A3 004A5          SUBW3   BKT_ADDR, LHS, 74(IRAB)                  ; 1733
                    4C    A9        48    A9    B0 004AA          MOVW    72(IRAB), 76(IRAB)                       ; 1734
                                0E        4A    A9    B1 004AF          CMPW    74(IRAB), #14                            ; 1736
                                                        05    12 004B3          BNEQ    64$                                      ;
                                44    A9        40    8F    88 004B5          BISB2   #64, 68(IRAB)                            ; 1738
                                44    A9              08    88 004BA 64$:     BISB2   #8, 68(IRAB)                             ; 1740
                                04    AE        10    AE    D1 004BE          CMPL    RHS, EOB                                 ; 1742
                                                        4C    1F 004C3          BLSSU   69$                                      ;
                                44    A9              20    88 004C5          BISB2   #32, 68(IRAB)                            ; 1749
                                                        56    11 004C9          BRB     72$                                      ; 1751
                                      50        48    A9    3C 004CB 65$:     MOVZWL  72(IRAB), R0                             ; 1768
                                4C    A9              50    B0 004CF          MOVW    R0, 76(IRAB)                             ;
                                4A    A9              50    B0 004D3          MOVW    R0, 74(IRAB)                             ;
                                                  44    A9    95 004D7          TSTB    68(IRAB)                                 ; 1770
                                                        18    18 004DA          BGEQ    66$                                      ;
                                44    A9              10    88 004DC          BISB2   #16, 68(IRAB)                            ; 1773
                                                50    52    D0 004E0          MOVL    LHS, REC_ADDR                            ; 1774
                                                    5C    D4 004E3          CLRL    AP                                       ; 1775
                                      50      00B4  CA    3C 004E5          MOVZWL  180(IFAB), R0                            ; 1776
                                            60  B940  9F    004EA             PUSHAB  @96(IRAB)[R0]                            ;
                                          0000G 30    004EE                 BSBW    RM$RECORD_KEY                            ;
                                      5E            04    C0 004F1          ADDL2   #4, SP                                   ;
                                04    AE        08    AE    D1 004F4 66$:     CMPL    POS_INSERT, EOB                          ; 1779
                                                        08    12 004F9          BNEQ    68$                                      ;
```

```
  44   A9          02          01          01  F0 004FB        INSV    #1, #1, #2, 68(IRAB)              ; 1781
                                           1E  11 00501 67$:   BRB     72$
                         10    AE    08    AE  D1 00503 68$:   CMPL    POS_INSERT, RHS                   ; 1784
                                           17  1E 00508        BGEQU   72$
                         04    AE    10    AE  D1 0050A        CMPL    RHS, EOB                          ; 1788
                                           06  1E 0050F        BGEQU   70$
                         44    A9          06  88 00511 69$:   BISB2   #6, 68(IRAB)                      ; 1790
                                           04  11 00515        BRB     71$
                         44    A9          20  88 00517 70$:   BISB2   #32, 68(IRAB)                     ; 1792
             4E   A9     10    AE          55  A3 0051B 71$:   SUBW3   BKT_ADDR, RHS, 78(IRAB)           ; 1794
                               0E    4A    A9  B1 00521 72$:   CMPW    74(IRAB), #14                     ; 1806
                                           0A  12 00525        BNEQ    73$
                   05    44    A9          03  E0 00527        BBS     #3, 68(IRAB), 73$                 ; 1808
                         44    A9    40    8F  88 0052C        BISB2   #64, 68(IRAB)                     ; 1810
                               5E          30  C0 00531 73$:   ADDL2   #48, SP                           ; 1814
                                           0C  BA 00534        POPR    #^M<R2,R3>
                                               05 00536        RSB

; Routine Size:  1335 bytes,    Routine Base:  RM$RMS3 + 0089

; 1755          1815  1
```

```
;  1757     1816  1 %SBTTL 'RM$SPLIT_UDR_3'
;  1758     1817  1 GLOBAL ROUTINE RM$SPLIT_UDR_3(RECSZ) : RL$RABREG_4567 NOVALUE =
;  1759     1818  1
;  1760     1819  1 !++
;  1761     1820  1 !
;  1762     1821  1 ! FUNCTIONAL DESCRIPTION:
;  1763     1822  1 !       This routine calculates bucket splits for prologue 3 version files.
;  1764     1823  1 !
;  1765     1824  1 ! CALLING SEQUENCE:
;  1766     1825  1 !       BSBW RM$SPLIT_UDR_3(RECSZ)
;  1767     1826  1 !
;  1768     1827  1 ! INPUT PARAMETERS:
;  1769     1828  1 !       RECSZ - packed record size including overhead
;  1770     1829  1 !
;  1771     1830  1 ! IMPLICIT INPUTS:
;  1772     1831  1 !       BDB pointer
;  1773     1832  1 !       BUFFER pointer
;  1774     1833  1 !       REC_ADDR -- point of insert
;  1775     1834  1 !       RAB -- to be passed to RM$MOVE_KEY
;  1776     1835  1 !       IDX_DFN
;  1777     1836  1 !       in IRAB -- CURBDB, associated with bdb and bkt_addr
;  1778     1837  1 !                  POS_INS corresponding to REC_ADDR
;  1779     1838  1 !                  key buffer address
;  1780     1839  1 !       in IFAB -- key buffer size
;  1781     1840  1 !       BKT$B_NXTRECID = 0 in original bucket signals that this is
;  1782     1841  1 !           a split due to a lack of id's in the bucket
;  1783     1842  1 !
;  1784     1843  1 ! OUTPUT PARAMETERS:
;  1785     1844  1 !       none
;  1786     1845  1 !
;  1787     1846  1 ! IMPLICIT OUTPUTS:
;  1788     1847  1 !       in IRAB --
;  1789     1848  1 !       if 2 bkt split --
;  1790     1849  1 !               IRB$W_SPLIT, offset to split point
;  1791     1850  1 !               IRB$V_REC_W_LO -- set if split point is pos_insert and
;  1792     1851  1 !                   record goes with lo set
;  1793     1852  1 !               key buffer 2 - new high key for original bucket, i.e. key to be
;  1794     1853  1 !                   inserted at the index level
;  1795     1854  1 !               key buffer 4 - old high key
;  1796     1855  1 !               number of new buckets = 1
;  1797     1856  1 !               if original bucket was all rrv's, set IRB$V_EMPTY_BKT flag
;  1798     1857  1 !               if new bucket is a continuation bkt., set IRB$V_CONT_BKT flag
;  1799     1858  1 !       if 3 bkt split --
;  1800     1859  1 !               same as above with these changes:
;  1801     1860  1 !               key buffer 3 - implicitly it contains second key to be inserted
;  1802     1861  1 !                   at the index level
;  1803     1862  1 !               IRB$W_SPLIT_1, offset to second split point
;  1804     1863  1 !               number of new buckets = 2
;  1805     1864  1 !               if right bucket is a continuation bkt, set IRB$V_CONT_R flag
;  1806     1865  1 !       if 4 bkt split --
;  1807     1866  1 !               same as above with these changes:
;  1808     1867  1 !               IRB$W_SPLIT_2, offset to third split point
;  1809     1868  1 !               number of new buckets = 3
;  1810     1869  1 !
;  1811     1870  1 ! ROUTINE VALUE:
;  1812     1871  1 !       rmssuc
;  1813     1872  1 !
```

```
;  1814          1873  1 ! SIDE EFFECTS:
;  1815          1874  1 !       AP is clobbered
;  1816          1875  1 !
;  1817          1876  1 !--
;  1818          1877  1
;  1819          1878  2     BEGIN
;  1820          1879  2
;  1821          1880  2     EXTERNAL REGISTER
;  1822          1881  2         COMMON_RAB_STR,
;  1823          1882  2         R_REC_ADDR_STR,
;  1824          1883  2         R_IDX_DFN_STR,
;  1825          1884  2         COMMON_IO_STR;
;  1826          1885  2
;  1827          1886  2     LOCAL
;  1828          1887  2         SAVE_REC_W_LO,
;  1829          1888  2         NEED_RRV,
;  1830          1889  2         POS_INSERT,
;  1831          1890  2         EOB,
;  1832          1891  2         RRV,
;  1833          1892  2         RHS,
;  1834          1893  2         LHS,
;  1835          1894  2         LAST    : REF BBLOCK,
;  1836          1895  2         LAST_DIFF,
;  1837          1896  2         BKTSIZE,
;  1838          1897  2         DIFFERENCE;
;  1839          1898  2
;  1840          1899  2     LITERAL
;  1841          1900  2         RRV_SIZE = 9;
;  1842          1901  2
;  1843          1902  2     LABEL
;  1844          1903  2         DO_IT,
;  1845          1904  2         HALF,
;  1846          1905  2         NEXT;
;  1847          1906  2
;  1848          1907  2 DO_IT :
;  1849          1908  2
;  1850          1909  3     BEGIN
;  1851          1910  3
;  1852          1911  3     ! define a block so that we can have some common checks before returning
;  1853          1912  3     ! successfully
;  1854          1913  3     !
;  1855          1914  3 HALF :
;  1856          1915  3
;  1857          1916  4     BEGIN
;  1858          1917  4
;  1859          1918  4     !+
;  1860          1919  4     ! Define a block so that we can simulate a go-to (naughty, naughty),
;  1861          1920  4     ! if we have decided that we are positioning at the end of the bucket
;  1862          1921  4     ! & we're in somewhat of an ascending order, where the last record
;  1863          1922  4     ! inserted is a duplicate of the new record, skip over the 50-50 code
;  1864          1923  4     ! and go to the code to take duplicates into account.
;  1865          1924  4     !
;  1866          1925  4     ! scan 1 -- Calculate size of existing rrv's and total number of rrv's
;  1867          1926  4     ! needed to move the whole bucket out ( worst case).  As a side effect,
;  1868          1927  4     ! adjust eob pointer to point to the rrv's instead of freespace.  Assume
;  1869          1928  4     ! not empty bucket until showed otherwise.
;  1870          1929  4     !-
```

```
;  1871          1930  4
;  1872          1931  4      IRAB[IRB$V_EMPTY_BKT] = 0;
;  1873          1932  4
;  1874          1933  4      ! new rec is tried 1st with hi set, then with lo set
;  1875          1934  4      !
;  1876          1935  4      IRAB[IRB$V_REC_W_LO] = 0;
;  1877          1936  4      IRAB[IRB$V_NEW_BKTS] = 1;    ! assume 2-bkt split until showed otherwise
;  1878          1937  4      NEED_RRV = 0;
;  1879          1938  4      POS_INSERT = .REC_ADDR;
;  1880          1939  4      REC_ADDR = .BKT_ADDR + BKT$C_OVERHDSZ;
;  1881          1940  4      EOB = .BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE];
;  1882          1941  4      LAST = 0;
;  1883          1942  4
;  1884          1943  4      DO
;  1885          1944  5          BEGIN
;  1886          1945  5
;  1887          1946  5          BUILTIN
;  1888          1947  5              AP;
;  1889          1948  5
;  1890          1949  5          IF .REC_ADDR[IRC$V_RRV]
;  1891          1950  5          THEN
;  1892          1951  5              EXITLOOP;
;  1893          1952  5
;  1894          1953  5          AP = 3;
;  1895          1954  5
;  1896          1955  5          IF .BDB[BDB$L_VBN] EQLU RM$RECORD_VBN()
;  1897          1956  5          THEN
;  1898          1957  5              NEED_RRV = .NEED_RRV + 1;
;  1899          1958  5
;  1900          1959  5          LAST = .REC_ADDR;
;  1901          1960  5
;  1902          1961  5          ! If the front compression of the current record is zero, save its
;  1903          1962  5          ! address as the last noncompressed key. This may prevent a bucket
;  1904          1963  5          ! scan when it comes time to extract and re-expand the key of the
;  1905          1964  5          ! last record in the bucket.
;  1906          1965  5          !
;  1907          1966  5          IF .IDX_DFN[IDX$V_KEY_COMPR]
;  1908          1967  5          THEN
;  1909          1968  6              BEGIN
;  1910          1969  6
;  1911          1970  6              IF .(.REC_ADDR + RM$REC_OVHD() + 1)<0,8> EQLU 0
;  1912          1971  6              THEN
;  1913          1972  6                  IRAB[IRB$L_LST_NCMP] = .REC_ADDR;
;  1914          1973  5              END;
;  1915          1974  5
;  1916          1975  5          RM$GETNEXT_REC()
;  1917          1976  5          END
;  1918          1977  4      UNTIL .REC_ADDR GEQU .EOB;                    ! end of first scan
;  1919          1978  4
;  1920          1979  4      ! Now that we have the address of the last record in the bucket, store
;  1921          1980  4      ! the key of that record in key buffer 4, to be used by index updating.
;  1922          1981  4      !
;  1923          1982  4      IF .LAST NEQU 0
;  1924          1983  4      THEN
;  1925          1984  5          BEGIN
;  1926          1985  5
;  1927          1986  5          LOCAL
```

```
;  1928     1987  5              TMP_ADDR;
;  1929     1988  5
;  1930     1989  5          BUILTIN
;  1931     1990  5              AP;
;  1932     1991  5
;  1933     1992  5          TMP_ADDR = .REC_ADDR;
;  1934     1993  5          REC_ADDR = .LAST;
;  1935     1994  5          AP = 0;                 ! overhead and compressed form
;  1936     1995  5          RM$RECORD_KEY(KEYBUF_ADDR(4));
;  1937     1996  5          REC_ADDR = .TMP_ADDR;
;  1938     1997  4          END;
;  1939     1998  4
;  1940     1999  4      ! Set SPLIT_2 and SPLIT_1 to be EOB, so if there are less than 3 new
;  1941     2000  4      ! buckets BKT_SPL can use the value without having to recalculate it.
;  1942     2001  4      !
;  1943     2002  4      IRAB[IRB$W_SPLIT_1] = IRAB[IRB$W_SPLIT_2] = .REC_ADDR - .BKT_ADDR;
;  1944     2003  4
;  1945     2004  4      ! Set up the bucket size
;  1946     2005  4      !
;  1947     2006  4      BKTSIZE = .IDX_DFN[IDX$B_DATBKTSZ]*512 - BKT$C_OVERHDSZ - BKT$C_DATBKTOVH;
;  1948     2007  4
;  1949     2008  4      ! If this is an update, may have to count in an rrv for the existing record
;  1950     2009  4      !
;  1951     2010  4
;  1952     2011  4      IF .IRAB[IRB$V_UPDATE]
;  1953     2012  4      THEN
;  1954     2013  5          BEGIN
;  1955     2014  5
;  1956     2015  5          IF .BDB[BDB$L_VBN] EQLU .IRAB[IRB$L_PUTUP_VBN]
;  1957     2016  5          THEN
;  1958     2017  5              NEED_RRV = .NEED_RRV + 1;
;  1959     2018  4          END;
;  1960     2019  4
;  1961     2020  4      RRV = .EOB - .REC_ADDR;                     ! size of existing rrv's
;  1962     2021  4      EOB = .REC_ADDR;                            ! adjust eob
;  1963     2022  4
;  1964     2023  4      ! special case it, if the bucket was all rrv's
;  1965     2024  4      !
;  1966     2025  4
;  1967     2026  4      IF .REC_ADDR EQLU .BKT_ADDR + BKT$C_OVERHDSZ
;  1968     2027  4      THEN
;  1969     2028  5          BEGIN
;  1970     2029  5
;  1971     2030  5          ! Bkt is all rrv's yet the record wouldn't fit so we need to
;  1972     2031  5          ! allocate another bkt ( 2 bkt split). Yet special case it so as not
;  1973     2032  5          ! to make another idx entry, only to update the existing one by
;  1974     2033  5          ! setting empty bucket flag.
;  1975     2034  5          !
;  1976     2035  5          IRAB[IRB$W_SPLIT] = .REC_ADDR - .BKT_ADDR;
;  1977     2036  5          LEAVE DO_IT
;  1978     2037  5
;  1979     2038  4          END;                        ! end { of special case an all-rrv bucket }
;  1980     2039  4
;  1981     2040  4      !+
;  1982     2041  4      ! * BLOCK 1 *
;  1983     2042  4      ! Special Case -- If we can detect a possible ascending order to these
;  1984     2043  4      ! records it probably will be better to do a straight point of insert split
```

```
;1985   2044  4      ! which would put the new record in a bucket all by itself.
;1986   2045  4      ! Do this kind of split if and only if all the following conditions are met:
;1987   2046  4      !    1) the record is being inserted at the end of bucket
;1988   2047  4      !    2) the last record physically in the bkt is the last record to have
;1989   2048  4      !       been inserted
;1990   2049  4      !    3) the last record and the new record do not have duplicate key values
;1991   2050  4      !
;1992   2051  4      ! Note that if they are duplicates, we can still make an optimization by
;1993   2052  4      ! skipping the 50-50 split code.
;1994   2053  4      !
;1995   2054  4      ! Note that LAST cannot be zero, since if it were we would have an all
;1996   2055  4      ! rrv bkt.
;1997   2056  4      !-
;1998   2057  4
;1999   2058  4      IF .POS_INSERT EQLU .REC_ADDR
;2000   2059  4          AND
;2001   2060  5          (((.LAST[IRC$W_ID] + 1) AND %X'FFFF') EQLU .BKT_ADDR[BKT$W_NXTRECID])
;2002   2061  4      THEN
;2003   2062  5          BEGIN
;2004   2063  5
;2005   2064  5          REC_ADDR = .LAST;
;2006   2065  5
;2007   2066  5          ! Check for duplicates:
;2008   2067  5          ! If the key is compressed, and the new key has a length of zero, then
;2009   2068  5          !   we know it is a duplicate of the previous one.
;2010   2069  5          ! If the key is not compressed, then compare the new key (key buffer 3)
;2011   2070  5          !   with the previous key.
;2012   2071  5          !
;2013   2072  5
;2014   2073  5          IF .IDX_DFN[IDX$V_KEY_COMPR]
;2015   2074  5          THEN
;2016   2075  6              BEGIN
;2017   2076  6
;2018   2077  6              IF .(.IRAB[IRB$L_RECBUF])<0,8> NEQU 0
;2019   2078  6              THEN
;2020   2079  7                  BEGIN
;2021   2080  7
;2022   2081  7                  ! Since we have detected a possible ascending order in the
;2023   2082  7                  ! input, let's try to optimize a little and split at the point
;2024   2083  7                  ! of insert.  Send the record by itself into the new bucket
;2025   2084  7                  ! and store the new high key of the old bucket in keybuf2,
;2026   2085  7                  ! the high key of the new bucket in keybuf4, and split point.
;2027   2086  7                  !
;2028   2087  7                  RM$MOVE(.IDX_DFN[IDX$B_KEYSZ],
;2029   2088  7                          KEYBUF_ADDR(4),
;2030   2089  7                          KEYBUF_ADDR(2));
;2031   2090  7                  RM$MOVE(.IDX_DFN[IDX$B_KEYSZ],
;2032   2091  7                          KEYBUF_ADDR(3),
;2033   2092  7                          KEYBUF_ADDR(4));
;2034   2093  7                  IRAB[IRB$W_SPLIT] = .IRAB[IRB$W_POS_INS];
;2035   2094  7                  LEAVE DO_IT;
;2036   2095  7                  END
;2037   2096  6              ELSE
;2038   2097  6                  LEAVE HALF
;2039   2098  6              END
;2040   2099  5          ELSE
;2041   2100  6              BEGIN
```

```
;  2042          2101  6
;  2043          2102  6              LOCAL
;  2044          2103  6                  REC_OVHD;
;  2045          2104  6
;  2046          2105  6              BUILTIN
;  2047          2106  6                  AP;
;  2048          2107  6
;  2049          2108  6              REC_OVHD = RM$REC_OVHD(0);
;  2050          2109  6              AP = 3;                 ! Contiguous compare of keys
;  2051          2110  6
;  2052          2111  6              IF RM$COMPARE_KEY ( .REC_ADDR + .REC_OVHD,
;  2053          2112  6                                  KEYBUF_ADDR(3),
;  2054          2113  6                                  .IDX_DFN[IDX$B_KEYSZ] )
;  2055          2114  6              THEN
;  2056          2115  7                  BEGIN
;  2057          2116  7                  RM$MOVE(.IDX_DFN[IDX$B_KEYSZ],
;  2058          2117  7                          KEYBUF_ADDR(4),
;  2059          2118  7                          KEYBUF_ADDR(2));
;  2060          2119  7                  RM$MOVE(.IDX_DFN[IDX$B_KEYSZ],
;  2061          2120  7                          KEYBUF_ADDR(3),
;  2062          2121  7                          KEYBUF_ADDR(4));
;  2063          2122  7                  IRAB[IRB$W_SPLIT] = .IRAB[IRB$W_POS_INS];
;  2064          2123  7                  LEAVE DO_IT;
;  2065          2124  7                  END
;  2066          2125  6              ELSE
;  2067          2126  6                  LEAVE HALF
;  2068          2127  6              END
;  2069          2128  6
;  2070          2129  6          ! * end of BLOCK 1 *
;  2071          2130  6          !
;  2072          2131  4          END;
;  2073          2132  4
;  2074          2133  4      REC_ADDR = .BKT_ADDR + BKT$C_OVERHDSZ;
;  2075          2134  4      IRAB[IRB$L_LST_NCMP] = .REC_ADDR;
;  2076          2135  4      LAST_DIFF = %X'7FFFFFFF';
;  2077          2136  4      LAST = 0;
;  2078          2137  4      SAVE_REC_W_LO = 0;
;  2079          2138  4
;  2080          2139  4      ! * BLOCK 2 *
;  2081          2140  4      ! Start from the beginning of the bucket and scan rightward. First find the
;  2082          2141  4      ! 1st place the rhs will fit in 1 bkt then, as long as the lhs will fit in
;  2083          2142  4      ! a bkt, try to find an optimal point. If there is no point where the rhs
;  2084          2143  4      ! and lhs will both fit, we can't do a 2-bkt split and this case will fall
;  2085          2144  4      ! out.
;  2086          2145  4      !
;  2087          2146  4
;  2088          2147  4      WHILE 1
;  2089          2148  4      DO
;  2090          2149  5          BEGIN
;  2091          2150  5          RHS = .EOB - .REC_ADDR;
;  2092          2151  5
;  2093          2152  5          IF  .REC_ADDR LEQU .POS_INSERT
;  2094          2153  5              AND
;  2095          2154  5              NOT .IRAB[IRB$V_REC_W_LO]
;  2096          2155  5          THEN
;  2097          2156  5              RHS = .RHS + .RECSZ;
;  2098          2157  5
```

```
;  2099     2158  5          ! If the primary key is compressed, then the righthand side total must
;  2100     2159  5          ! include the count of characters currently front compressed off the
;  2101     2160  5          ! key of the record which will be first in the right bucket.
;  2102     2161  5          !
;  2103     2162  5          IF .IDX_DFN[IDX$V_KEY_COMPR]
;  2104     2163  5          THEN
;  2105     2164  5              ! If the point of insertion of the new (updated) record is the same
;  2106     2165  5              ! as that of the current split point, and the new (updated) record
;  2107     2166  5              ! is to go in the new (right) bucket, the the number of front
;  2108     2167  5              ! compressed characters to be added to the righthand total comes
;  2109     2168  5              ! from the currently compressed key of the new (updated) record.
;  2110     2169  5              ! This key will be found in keybuffer 5, if the current operation
;  2111     2170  5              ! is an $UPDATE, or in a record buffer, if the current operation is
;  2112     2171  5              ! a $PUT.
;  2113     2172  5              !
;  2114     2173  6              IF  (.REC_ADDR EQLA .POS_INSERT)
;  2115     2174  5                   AND
;  2116     2175  5                  NOT .IRAB[IRB$V_REC_W_LO]
;  2117     2176  5              THEN
;  2118     2177  5                  IF .IRAB[IRB$V_UPDATE]
;  2119     2178  5                  THEN
;  2120     2179  5                      RHS = .RHS + .(KEYBUF_ADDR(5) + 1)<0,8>
;  2121     2180  5                  ELSE
;  2122     2181  5                      RHS = .RHS + .(.IRAB[IRB$L_RECBUF] + 1)<0,8>
;  2123     2182  5
;  2124     2183  5              ! If the current split point is not at the point of insertion of
;  2125     2184  5              ! the new (updated) record, or if it is but the new (updated)
;  2126     2185  5              ! record is to go in the old (left) bucket, then the first record
;  2127     2186  5              ! in the new (right) bucket will be the current record, and the
;  2128     2187  5              ! number of characters currently front compressed off its key is
;  2129     2188  5              ! added to the righthand side total.
;  2130     2189  5              !
;  2131     2190  5              ELSE
;  2132     2191  5                  IF .REC_ADDR LSSA .EOB
;  2133     2192  5                  THEN
;  2134     2193  5                      RHS = .RHS + .(.REC_ADDR + RM$REC_OVHD(0) + 1)<0,8>;
;  2135     2194  5
;  2136     2195  5          ! * BLOCK 3 *
;  2137     2196  5          ! The right hand side fits if there is enough room and there are id's
;  2138     2197  5          ! available. Id's are always available in the new bucket in the update
;  2139     2198  5          ! situation, or if we're leaving at least 1 record behind in the old
;  2140     2199  5          ! bucket. note that nxtrecid is always zeroed if this is a split due to
;  2141     2200  5          ! lack of id's.
;  2142     2201  5          !
;  2143     2202  5
;  2144     2203  5          IF  .RHS LSSU .BKTSIZE
;  2145     2204  5              AND
;  2146     2205  6              (.BKT_ADDR[BKT$W_NXTRECID] NEQ 0
;  2147     2206  6              OR
;  2148     2207  6              .IRAB[IRB$V_UPDATE]
;  2149     2208  6              OR
;  2150     2209  7              .REC_ADDR NEQA (.BKT_ADDR + BKT$C_OVERHDSZ)
;  2151     2210  6              OR
;  2152     2211  6              .IRAB[IRB$V_REC_W_LO])
;  2153     2212  5          THEN
;  2154     2213  6              BEGIN
;  2155     2214  6              LHS = .REC_ADDR - (.BKT_ADDR + BKT$C_OVERHDSZ);
```

```
;  2156     2215  6              IF  .REC_ADDR GEQU .POS_INSERT
;  2157     2216  6                  AND
;  2158     2217  6                  .IRAB[IRB$V_REC_W_LO]
;  2159     2218  6              THEN
;  2160     2219  6                  LHS = .LHS + .RECSZ;
;  2161     2220  6
;  2162     2221  6              ! * BLOCK 4 *
;  2163     2222  6              ! will lhs fit ? lhs doesn't fit if there is no space in the
;  2164     2223  6              ! bucket, or if there won't be any id's available in the bucket,
;  2165     2224  6              ! if not & if there is no previous point at which it fit, goto 3-bkt
;  2166     2225  6              ! split code if there is a previous place where we could have had a
;  2167     2226  6              ! 2-bkt split, use it
;  2168     2227  6              !
;  2169     2228  6
;  2170     2229  6              IF .LHS + .RRV + (RRV_SIZE * .NEED_RRV) GTRU .BKTSIZE
;  2171     2230  6
;  2172     2231  6                  ! Id's will be available in the original bucket if we aren't
;  2173     2232  6                  ! out of id's to begin with, if this is an update,
;  2174     2233  6                  ! or if the new record is going in the new bucket
;  2175     2234  6                  !
;  2176     2235  6                  OR
;  2177     2236  6                  (.BKT_ADDR[BKT$W_NXTRECID] EQL 0
;  2178     2237  7                  AND
;  2179     2238  7                  NOT .IRAB[IRB$V_UPDATE]
;  2180     2239  7                  AND
;  2181     2240  7                  .IRAB[IRB$V_REC_W_LO])
;  2182     2241  7              THEN
;  2183     2242  6                  BEGIN
;  2184     2243  7
;  2185     2244  7                  IF .LAST EQL 0
;  2186     2245  7                  THEN
;  2187     2246  7                      EXITLOOP;
;  2188     2247  7
;  2189     2248  7                  REC_ADDR = .LAST;
;  2190     2249  7
;  2191     2250  7                  IF NOT .SAVE_REC_W_LO
;  2192     2251  7                  THEN
;  2193     2252  7                      IRAB[IRB$V_REC_W_LO] = 0;
;  2194     2253  7
;  2195     2254  7                  ! 2 bkt split is possible rec_addr points to the most
;  2196     2255  7                  ! optimal place since we had to back up, reset last to point
;  2197     2256  7                  ! to the record immediately before the split point
;  2198     2257  7                  !
;  2199     2258  7                  BEGIN
;  2200     2259  8
;  2201     2260  8                  LOCAL
;  2202     2261  8                      TMP;
;  2203     2262  8
;  2204     2263  8                  TMP = .REC_ADDR;
;  2205     2264  8                  REC_ADDR = .BKT_ADDR + BKT$C_OVERHDSZ;
;  2206     2265  8                  LAST = .REC_ADDR;
;  2207     2266  8
;  2208     2267  8                  WHILE .REC_ADDR NEQU .TMP
;  2209     2268  8                  DO
;  2210     2269  8                      BEGIN
;  2211     2270  9                      LAST = .REC_ADDR;
;  2212     2271  9
```

```
;  2213          2272  9                    ! If the front compression of the current record is zero,
;  2214          2273  9                    ! save its address as the last noncompressed key. This may
;  2215          2274  9                    ! prevent a bucket scan when it comes time to extract and
;  2216          2275  9                    ! re-expand the key of the last record in the bucket
;  2217          2276  9                    ! immediately before the split point.
;  2218          2277  9                    !
;  2219          2278  9                    IF .IDX_DFN[IDX$V_KEY_COMPR]
;  2220          2279  9                    THEN
;  2221          2280  9                        BEGIN
;  2222          2281 10
;  2223          2282 10                        IF .(.REC_ADDR + RM$REC_OVHD() + 1)<0,8> EQLU 0
;  2224          2283 10                        THEN
;  2225          2284 10                            IRAB[IRB$L_LST_NCMP] = .REC_ADDR;
;  2226          2285 10                        END;
;  2227          2286  9
;  2228          2287  9                    RM$GETNEXT_REC();
;  2229          2288  9                    END;
;  2230          2289  8
;  2231          2290  8
;  2232          2291  7                END;
;  2233          2292  7                RM$MOVE_KEY(.LAST, .REC_ADDR);
;  2234          2293  7                IRAB[IRB$W_SPLIT] = .REC_ADDR - .BKT_ADDR;
;  2235          2294  7
;  2236          2295  7                ! treat another exception case of the new record going off into
;  2237          2296  7                ! a cont. bkt all by itself
;  2238          2297  7                !
;  2239          2298  7
;  2240          2299  7                IF .IRAB[IRB$W_SPLIT] EQLU .IRAB[IRB$W_POS_INS]
;  2241          2300  7                THEN
;  2242          2301  7
;  2243          2302  7                    IF .IRAB[IRB$W_SPLIT] EQLU .IRAB[IRB$W_SPLIT_1]
;  2244          2303  7                    THEN
;  2245          2304  7
;  2246          2305  7                        IF NOT .IRAB[IRB$V_REC_W_LO]
;  2247          2306  7                        THEN
;  2248          2307  8                            BEGIN
;  2249          2308  8
;  2250          2309  8                            BUILTIN
;  2251          2310  8                                AP;
;  2252          2311  8
;  2253          2312  8                            AP = 3;
;  2254          2313  8
;  2255          2314  8                            ! If the new last key in the bucket equals the key
;  2256          2315  8                            ! to be inserted in the new bucket, then we have a
;  2257          2316  8                            ! continuation bucket.
;  2258          2317  8                            !
;  2259          2318  8                            IF NOT RM$COMPARE_KEY ( KEYBUF_ADDR(2),
;  2260          2319  8                                                    KEYBUF_ADDR(3),
;  2261          2320  8                                                    .IDX_DFN[IDX$B_KEYSZ] )
;  2262          2321  8                            THEN
;  2263          2322  8                                IRAB[IRB$V_CONT_BKT] = 1;
;  2264          2323  8
;  2265          2324  7                            END;
;  2266          2325  7
;  2267          2326  7            LEAVE DO_IT
;  2268          2327  7
;  2269          2328  6            END;                    ! end of * BLOCK 4 * (LHS does not fit)
```

```
;  2270          2329  6          ! lhs fits also, calculate the magic ratio
;  2271          2330  6          !
;  2272          2331  6          DIFFERENCE = (.LHS * .BKTSIZE) -
;  2273          2332  6              (.RHS * (.BKTSIZE - (RRV_SIZE * .NEED_RRV) - .RRV));
;  2274          2333  6
;  2275          2334  6          ! * BLOCK 5 *
;  2276          2335  6          !
;  2277          2336  6
;  2278          2337  6          IF .DIFFERENCE GEQ 0
;  2279          2338  6          THEN
;  2280          2339  6              BEGIN
;  2281          2340  7
;  2282          2341  7              ! found the 1st point at which the magic ratio is positive
;  2283          2342  7              ! was the last point more optimal, if so use it
;  2284          2343  7              !
;  2285          2344  7
;  2286          2345  7              IF ABS(.DIFFERENCE) GTRU ABS(.LAST_DIFF)
;  2287          2346  7              THEN
;  2288          2347  7                  BEGIN
;  2289          2348  8
;  2290          2349  8                  IF .REC_ADDR EQLU .LAST
;  2291          2350  8                  THEN
;  2292          2351  8                      IRAB[IRB$V_REC_W_LO] = 0
;  2293          2352  8                  ELSE
;  2294          2353  8                      BEGIN
;  2295          2354  9                      REC_ADDR = .LAST;
;  2296          2355  9
;  2297          2356  9                      IF .REC_ADDR LSSU .POS_INSERT
;  2298          2357  9                      THEN
;  2299          2358  9                          IRAB[IRB$V_REC_W_LO] = 0;
;  2300          2359  9                      END;
;  2301          2360  8
;  2302          2361  8                  LAST = 0;
;  2303          2362  8                  END;
;  2304          2363  7
;  2305          2364  7              ! 2-bkt split is possible rec_addr points to the most
;  2306          2365  7              ! optimal place
;  2307          2366  7              !
;  2308          2367  7
;  2309          2368  7              IF .LAST EQL 0
;  2310          2369  7              THEN            ! just backed up rec_addr, need to recalc last
;  2311          2370  7                  BEGIN
;  2312          2371  8
;  2313          2372  8                  LOCAL
;  2314          2373  8                      TMP;
;  2315          2374  8
;  2316          2375  8                  TMP = .REC_ADDR;
;  2317          2376  8                  REC_ADDR = .BKT_ADDR + BKT$C_OVERHDSZ;
;  2318          2377  8                  LAST = .REC_ADDR;
;  2319          2378  8
;  2320          2379  8                  WHILE .REC_ADDR NEQU .TMP
;  2321          2380  8                  DO
;  2322          2381  8                      BEGIN
;  2323          2382  9                      LAST = .REC_ADDR;
;  2324          2383  9
;  2325          2384  9                      ! If the front compression of the current record is
;  2326          2385  9
```

```
;  2327     2386  9                    ! zero, save its address as the last noncompressed key.
;  2328     2387  9                    ! This may prevent a bucket scan when it comes time to
;  2329     2388  9                    ! extract and re-expand the key of the last record in
;  2330     2389  9                    ! the bucket immediately before the split point.
;  2331     2390  9                    !
;  2332     2391  9                    IF .IDX_DFN[IDX$V_KEY_COMPR]
;  2333     2392  9                    THEN
;  2334     2393 10                        BEGIN
;  2335     2394 10
;  2336     2395 10                        IF .(.REC_ADDR + RM$REC_OVHD() + 1)<0,8> EQLU 0
;  2337     2396 10                        THEN
;  2338     2397 10                            IRAB[IRB$L_LST_NCMP] = .REC_ADDR;
;  2339     2398  9                        END;
;  2340     2399  9
;  2341     2400  9                    RM$GETNEXT_REC();
;  2342     2401  8                    END;
;  2343     2402  8
;  2344     2403  7                END;
;  2345     2404  7
;  2346     2405  7            RM$MOVE_KEY(.LAST, .REC_ADDR);
;  2347     2406  7            IRAB[IRB$W_SPLIT] = .REC_ADDR - .BKT_ADDR;
;  2348     2407  7
;  2349     2408  7            ! treat another exception case of the new record going off into
;  2350     2409  7            ! a cont. bkt all by itself
;  2351     2410  7            !
;  2352     2411  7
;  2353     2412  7            IF .IRAB[IRB$W_SPLIT] EQLU .IRAB[IRB$W_POS_INS]
;  2354     2413  7            THEN
;  2355     2414  7
;  2356     2415  7                IF .IRAB[IRB$W_SPLIT] EQLU .IRAB[IRB$W_SPLIT_1]
;  2357     2416  7                THEN
;  2358     2417  7
;  2359     2418  7                    IF NOT .IRAB[IRB$V_REC_W_LO]
;  2360     2419  7                    THEN
;  2361     2420  8                        BEGIN
;  2362     2421  8
;  2363     2422  8                        BUILTIN
;  2364     2423  8                            AP;
;  2365     2424  8
;  2366     2425  8                        AP = 3;
;  2367     2426  8
;  2368     2427  8                        IF NOT RM$COMPARE_KEY ( KEYBUF_ADDR(2),
;  2369     2428  8                                                KEYBUF_ADDR(3),
;  2370     2429  8                                                .IDX_DFN[IDX$B_KEYSZ] )
;  2371     2430  8                        THEN
;  2372     2431  8                            IRAB[IRB$V_CONT_BKT] = 1;
;  2373     2432  8
;  2374     2433  7                        END;
;  2375     2434  7
;  2376     2435  7                LEAVE DO_IT
;  2377     2436  7
;  2378     2437  6                END;            ! end of * BLOCK 5 *
;  2379     2438  6
;  2380     2439  6            ! the magic ratio isn't positive yet, so save all the context and
;  2381     2440  6            ! move on to the next record
;  2382     2441  6            !
;  2383     2442  6            LAST_DIFF = .DIFFERENCE;
```

```
;  2384     2443  6              LAST = .REC_ADDR;
;  2385     2444  6
;  2386     2445  6              IF .IRAB[IRB$V_REC_W_LO]
;  2387     2446  6              THEN
;  2388     2447  6                  SAVE_REC_W_LO = 1;
;  2389     2448  6
;  2390     2449  5              END;                    ! end of * BLOCK 3 *
;  2391     2450  5
;  2392     2451  5          ! Go get the next record, but special case when we are at the position
;  2393     2452  5          ! of insert.
;  2394     2453  5          !
;  2395     2454  5 NEXT :
;  2396     2455  6          BEGIN
;  2397     2456  6
;  2398     2457  6          IF  .REC_ADDR EQLU .POS_INSERT
;  2399     2458  6              AND
;  2400     2459  6              NOT .IRAB[IRB$V_REC_W_LO]
;  2401     2460  6          THEN
;  2402     2461  7              BEGIN
;  2403     2462  7
;  2404     2463  7              ! If this is an update, check to see if it needed an rrv, since
;  2405     2464  7              ! the record will go in the left bucket.
;  2406     2465  7              !
;  2407     2466  7              IF .IRAB[IRB$V_UPDATE]
;  2408     2467  7              THEN
;  2409     2468  8                  BEGIN
;  2410     2469  8
;  2411     2470  8                  IF .BDB[BDB$L_VBN] EQLU .IRAB[IRB$L_PUTUP_VBN]
;  2412     2471  8                  THEN
;  2413     2472  8                      NEED_RRV = .NEED_RRV - 1;
;  2414     2473  8
;  2415     2474  7                  END;
;  2416     2475  7
;  2417     2476  7              ! Force record to low bucket, and put in key buffer 2 the key
;  2418     2477  7              ! of the record we are inserting (currently in keybuffer 3).
;  2419     2478  7              !
;  2420     2479  7              IRAB[IRB$V_REC_W_LO] = 1;
;  2421     2480  7              RM$MOVE(.IDX_DFN[IDX$B_KEYSZ], KEYBUF_ADDR(3), KEYBUF_ADDR(2));
;  2422     2481  7
;  2423     2482  7              ! If we are inserting at the end of the bucket, or if the record
;  2424     2483  7              ! at position of insert has a different key from that to be inserted,
;  2425     2484  7              ! leave NEXT so that no other record goes to the left bucket (so far).
;  2426     2485  7              ! If the key is a duplicate, then keep them together in the left
;  2427     2486  7              ! bucket.
;  2428     2487  7
;  2429     2488  7              IF  .REC_ADDR EQLU .EOB
;  2430     2489  7              THEN
;  2431     2490  7                  LEAVE NEXT
;  2432     2491  7              ELSE
;  2433     2492  8                  BEGIN
;  2434     2493  8
;  2435     2494  8                  BUILTIN
;  2436     2495  8                      AP;
;  2437     2496  8
;  2438     2497  8                  LOCAL
;  2439     2498  8                      CURR_KEY,
;  2440     2499  8                      REC_OVHD;
```

```
;  2441     2500  8                    REC_OVHD = RM$REC_OVHD(0);
;  2442     2501  8
;  2443     2502  8                    ! When the key is compressed, we must build it first in key
;  2444     2503  8                    ! buffer 5, and then compare.  This build is easy because we
;  2445     2504  8                    ! can take the front chars from the key to be inserted.
;  2446     2505  8                    !
;  2447     2506  8
;  2448     2507  8                    IF .IDX_DFN[IDX$V_KEY_COMPR]
;  2449     2508  8                    THEN
;  2450     2509  9                        BEGIN
;  2451     2510  9                        CURR_KEY = KEYBUF_ADDR(5);
;  2452     2511  9                        RM$MOVE ( .(.REC_ADDR + .REC_OVHD + 1)<0,8>,
;  2453     2512  9                                  KEYBUF_ADDR(2),
;  2454     2513  9                                  .CURR_KEY );
;  2455     2514  9                        RM$BUILD_KEY ( .REC_ADDR + .REC_OVHD, .CURR_KEY );
;  2456     2515  9                        END
;  2457     2516  8                    ELSE
;  2458     2517  8                        CURR_KEY = .REC_ADDR + .REC_OVHD;
;  2459     2518  8                    AP = 3;          ! Contiguous compare of keys
;  2460     2519  8
;  2461     2520  8                    IF RM$COMPARE_KEY ( .CURR_KEY,
;  2462     2521  8                                        KEYBUF_ADDR(2),
;  2463     2522  8                                        .IDX_DFN[IDX$B_KEYSZ] )
;  2464     2523  8                    THEN
;  2465     2524  8                        LEAVE NEXT;
;  2466     2525  8                    END;
;  2467     2526  7
;  2468     2527  7
;  2469     2528  6                END;         ! end of { at position for insert for the 1st time }
;  2470     2529  6
;  2471     2530  6                ! Now RMS will scan the bucket starting from the current record
;  2472     2531  6                ! position and keeping duplicates together, since RMS does not want to
;  2473     2532  6                ! split the bucket in the middle of a duplicate chain. Before scanning
;  2474     2533  6                ! RMS obtains the size of the current record, saves its address in
;  2475     2534  6                ! IRB$L_LST_NCMP, if its key in zero front compressed, and saves the
;  2476     2535  6                ! key of the current record in keybuffer 2
;  2477     2536  6                !
;  2478     2537  7                BEGIN
;  2479     2538  7
;  2480     2539  7                LOCAL
;  2481     2540  7                    REC_OVHD,
;  2482     2541  7                    S_REC_SIZE,
;  2483     2542  7                    NOT_DUP;
;  2484     2543  7
;  2485     2544  7                NOT_DUP = 0;             ! assume duplicates
;  2486     2545  7
;  2487     2546  7                ! Determine the size of the current record.
;  2488     2547  7                !
;  2489     2548  7                REC_OVHD = RM$REC_OVHD(0; S_REC_SIZE);
;  2490     2549  7
;  2491     2550  7                ! Save the address of the current record if its key is zero front
;  2492     2551  7                ! compressed.
;  2493     2552  7                !
;  2494     2553  7                IF .IDX_DFN[IDX$V_KEY_COMPR]
;  2495     2554  7                    AND
;  2496     2555  7                    .(.REC_ADDR + .REC_OVHD + 1)<0,8> EQLU 0
;  2497     2556  7                THEN
```

```
;  2498     2557  7              IRAB[IRB$L_LST_NCMP] = .REC_ADDR;
;  2499     2558  7
;  2500     2559  7          ! Move the key of the current record into keybuffer 2. Fool RM$MOVE_KEY
;  2501     2560  7          ! a little by always clearing REC_W_LO so that we get in key buffer 2
;  2502     2561  7          ! the key associated with the record we are pointing to.
;  2503     2562  7          !
;  2504     2563  8          BEGIN
;  2505     2564  8
;  2506     2565  8          LOCAL
;  2507     2566  8              TMP : BYTE;
;  2508     2567  8
;  2509     2568  8          TMP = .IRAB[IRB$B_SPL_BITS];
;  2510     2569  8          IRAB[IRB$V_REC_W_LO] = 0;
;  2511     2570  8          RM$MOVE_KEY(.REC_ADDR, .REC_ADDR);
;  2512     2571  8          IRAB[IRB$B_SPL_BITS] = .TMP
;  2513     2572  7          END;
;  2514     2573  7
;  2515     2574  7          ! Position to the next record which does not contain a key duplicate to
;  2516     2575  7          ! that of the current record (whose key has been saved in keybuffer 2).
;  2517     2576  7          !
;  2518     2577  7          DO
;  2519     2578  8              BEGIN
;  2520     2579  8
;  2521     2580  8              BUILTIN
;  2522     2581  8                  AP;
;  2523     2582  8
;  2524     2583  8              IF .REC_ADDR EQLU .EOB
;  2525     2584  8              THEN
;  2526     2585  8                  EXITLOOP;
;  2527     2586  8
;  2528     2587  8              AP = 3;
;  2529     2588  8
;  2530     2589  8              IF .BDB[BDB$L_VBN] EQLU RM$RECORD_VBN()
;  2531     2590  8              THEN
;  2532     2591  8                  NEED_RRV = .NEED_RRV - 1;
;  2533     2592  8
;  2534     2593  8              REC_ADDR = .REC_ADDR + .REC_OVHD + .S_REC_SIZE;      ! get next rec
;  2535     2594  8
;  2536     2595  8              IF .REC_ADDR EQLU .EOB
;  2537     2596  8              THEN
;  2538     2597  8                  EXITLOOP;
;  2539     2598  8
;  2540     2599  8              REC_OVHD = RM$REC_OVHD(0; S_REC_SIZE);
;  2541     2600  8
;  2542     2601  8              IF .IDX_DFN[IDX$V_KEY_COMPR]
;  2543     2602  8              THEN
;  2544     2603  9                  BEGIN
;  2545     2604  9
;  2546     2605  9                  IF .(.REC_ADDR + .REC_OVHD)<0,8> NEQU 0
;  2547     2606  9                  THEN
;  2548     2607  9                      NOT_DUP = 1;
;  2549     2608  9                  END
;  2550     2609  8              ELSE
;  2551     2610  9                  BEGIN
;  2552     2611  9                  AP = 3;          ! Contiguous compare of keys
;  2553     2612  9
;  2554     2613  9                  IF RM$COMPARE_KEY ( .REC_ADDR + .REC_OVHD,
```

```
;  2555     2614  9                                    KEYBUF_ADDR(2),
;  2556     2615  9                                    .IDX_DFN[IDX$B_KEYSZ] )
;  2557     2616  9                THEN
;  2558     2617  9                    NOT_DUP = 1;
;  2559     2618  8                END;
;  2560     2619  8
;  2561     2620  8                ! If RMS is currently positioned to the point of insertion of the
;  2562     2621  8                ! updated record, and if the key of the next record matches the
;  2563     2622  8                ! key of the previous record, then the updated record must go
;  2564     2623  8                ! into the old (left) bucket.
;  2565     2624  8                !
;  2566     2625  8                IF  .REC_ADDR EQLU .POS_INSERT
;  2567     2626  8                    AND
;  2568     2627  8                NOT .NOT_DUP
;  2569     2628  8                    AND
;  2570     2629  8                .IRAB[IRB$V_UPDATE]
;  2571     2630  8                THEN
;  2572     2631  9                    BEGIN
;  2573     2632  9
;  2574     2633  9                    IRAB[IRB$V_REC_W_LO] = 1;
;  2575     2634  9
;  2576     2635  9                    IF  .BDB[BDB$L_VBN] EQLU .IRAB[IRB$L_PUTUP_VBN]
;  2577     2636  9                    THEN
;  2578     2637  9                        NEED_RRV = .NEED_RRV - 1;
;  2579     2638  8                    END;
;  2580     2639  8
;  2581     2640  8                END
;  2582     2641  8
;  2583     2642  8            ! Loop until a non-duplicate record is found
;  2584     2643  8            !
;  2585     2644  7            UNTIL .NOT_DUP;
;  2586     2645  7
;  2587     2646  6            END;                                ! end of block defining NOT_DUP
;  2588     2647  6
;  2589     2648  6            ! If the key compares brought us up to the pos of insert, see if the
;  2590     2649  6            ! key of the new record matches the key of the record before the
;  2591     2650  6            ! position of insert. If it does, have to include the new record with
;  2592     2651  6            ! the lhs.
;  2593     2652  6            !
;  2594     2653  6
;  2595     2654  6            IF .REC_ADDR EQLU .POS_INSERT
;  2596     2655  6            THEN
;  2597     2656  7                BEGIN
;  2598     2657  7
;  2599     2658  7                BUILTIN
;  2600     2659  7                    AP;
;  2601     2660  7
;  2602     2661  7                AP = 3;
;  2603     2662  7
;  2604     2663  7                IF NOT RM$COMPARE_KEY( KEYBUF_ADDR(2),
;  2605     2664  7                                       KEYBUF_ADDR(3),
;  2606     2665  7                                       .IDX_DFN[IDX$B_KEYSZ] )
;  2607     2666  7                THEN
;  2608     2667  8                    BEGIN
;  2609     2668  8                    IRAB[IRB$V_REC_W_LO] = 1;
;  2610     2669  8
;  2611     2670  8                    IF  .IRAB[IRB$V_UPDATE]
```

```
;  2612          2671  8                    AND
;  2613          2672  8                    .BDB[BDB$L_VBN] EQLU .IRAB[IRB$L_PUTUP_VBN]
;  2614          2673  8                THEN
;  2615          2674  8                    NEED_RRV = .NEED_RRV - 1;
;  2616          2675  7                END;
;  2617          2676  7
;  2618          2677  6            END;
;  2619          2678  6
;  2620          2679  5        END;                ! end of NEXT
;  2621          2680  4        END;                ! end of * BLOCK 2 *
;  2622          2681  4
;  2623          2682  3    END;                    ! end of HALF
;  2624          2683  3
;  2625          2684  3    ! define a new block here so local storage can be redefined
;  2626          2685  3    !
;  2627          2686  4    BEGIN
;  2628          2687  4
;  2629          2688  4    MACRO
;  2630          2689  4        BEG_CHAIN = LHS %,
;  2631          2690  4        END_CHAIN = RHS %,
;  2632          2691  4        DUPS = RRV %;
;  2633          2692  4
;  2634          2693  4    LOCAL
;  2635          2694  4        FIRST_KEY_EXPANSION;
;  2636          2695  4
;  2637          2696  4    BUILTIN
;  2638          2697  4        AP;
;  2639          2698  4
;  2640          2699  4    ! If we end up with a duplicate chain here, we need to account for the
;  2641          2700  4    ! the fact that the first record which would end up in a new bucket
;  2642          2701  4    ! will have it's first key expanded fully.  Initialize the expansion
;  2643          2702  4    ! amount to 0.
;  2644          2703  4    !
;  2645          2704  4    FIRST_KEY_EXPANSION = 0;
;  2646          2705  4
;  2647          2706  4    ! Must be a 3 or 4 bucket split or we detected ascending order and the new
;  2648          2707  4    ! record was a dupe.  We'll optimize here to the extent of trying to keep a
;  2649          2708  4    ! dup chain around the new record together and in the middle bucket.
;  2650          2709  4    ! Note that in all the cases that follow the new record is going into the
;  2651          2710  4    ! middle bucket.  Therefore, the "lhs" will always fit, since it can only
;  2652          2711  4    ! get smaller (or stay the same size, in the degenerate case).  Also note
;  2653          2712  4    ! that in any of these cases, the left hand bucket may be empty of data
;  2654          2713  4    ! records (have only rrv's in it) if the first split point is at the
;  2655          2714  4    ! beginning and all data records get moved
;  2656          2715  4    !
;  2657          2716  4    IRAB[IRB$V_NEW_BKTS] = 2;    ! assume 3-bkt split until shown otherwise
;  2658          2717  4    IRAB[IRB$V_REC_W_LO] = 0;
;  2659          2718  4
;  2660          2719  4    ! Initialize key buffer 2 with the contents of key buffer 3 (the value
;  2661          2720  4    ! of the primary key of the record being inserted).  This is necessary
;  2662          2721  4    ! when the new record is at the beginning of the bucket and is going into
;  2663          2722  4    ! a bucket all by itself so that all the records in the bucket need rrv's
;  2664          2723  4    ! since they all move into the next bucket.
;  2665          2724  4    ! At any rate, that seems to be the only case where key buffer 2 is not
;  2666          2725  4    ! correct coming into here and will be set correctly before leaving.
;  2667          2726  4    !
;  2668          2727  4    RM$MOVE(.IDX_DFN[IDX$B_KEYSZ], KEYBUF_ADDR(3), KEYBUF_ADDR(2));
```

```
;2669    2728  4
;2670    2729  4      ! Find beginning and end of this possible dups chain equal to the key value
;2671    2730  4      ! of the record being inserted.
;2672    2731  4      !
;2673    2732  4      REC_ADDR = .BKT_ADDR + BKT$C_OVERHDSZ;
;2674    2733  5      BEGIN
;2675    2734  5
;2676    2735  5      LOCAL
;2677    2736  5          STATUS,
;2678    2737  5          REC_OVHD,
;2679    2738  5          S_REC_SIZE,
;2680    2739  5          CURR_KEY;
;2681    2740  5
;2682    2741  5      WHILE 1
;2683    2742  5      DO
;2684    2743  6          BEGIN
;2685    2744  6
;2686    2745  6          REC_OVHD = RM$REC_OVHD(0; S_REC_SIZE);
;2687    2746  6
;2688    2747  6          ! If the key is compressed, it must be rebuilt into keybuffer 5 first
;2689    2748  6          !
;2690    2749  6
;2691    2750  6          IF .IDX_DFN[IDX$V_KEY_COMPR]
;2692    2751  6          THEN
;2693    2752  7              BEGIN
;2694    2753  7              CURR_KEY = KEYBUF_ADDR(5);
;2695    2754  7              RM$BUILD_KEY ( .REC_ADDR + .REC_OVHD, .CURR_KEY );
;2696    2755  7              END
;2697    2756  6          ELSE
;2698    2757  6              ! Otherwise, we are already pointing to the beginning of the key
;2699    2758  6              !
;2700    2759  6              CURR_KEY = .REC_ADDR + .REC_OVHD;
;2701    2760  6          AP = 3;                     ! Contiguous compare of keys
;2702    2761  6          STATUS = RM$COMPARE_KEY ( .CURR_KEY,
;2703    2762  6                                    KEYBUF_ADDR(3),
;2704    2763  6                                    .IDX_DFN[IDX$B_KEYSZ] );
;2705    2764  6
;2706    2765  6          IF NOT .STATUS              ! If key matched, found beginning of chain
;2707    2766  6          THEN
;2708    2767  6              EXITLOOP;
;2709    2768  6
;2710    2769  6          IF .REC_ADDR LSSU .POS_INSERT
;2711    2770  6          THEN
;2712    2771  6              RM$MOVE( .IDX_DFN[IDX$B_KEYSZ], .CURR_KEY, KEYBUF_ADDR(2) );
;2713    2772  6
;2714    2773  6          IF  .REC_ADDR EQLU .EOB
;2715    2774  6              OR
;2716    2775  6              .STATUS LSS 0
;2717    2776  6          THEN
;2718    2777  7              BEGIN
;2719    2778  7
;2720    2779  7              ! !!!! SPLIT TYPE 3 !!!!
;2721    2780  7              ! No duplicates found.  For simplicity, do a 3-bkt split at the
;2722    2781  7              ! point of insert with the new record in its own bucket.
;2723    2782  7              !
;2724    2783  7              IRAB[IRB$W_SPLIT] = IRAB[IRB$W_SPLIT_1] = .IRAB[IRB$W_POS_INS];
;2725    2784  7              LEAVE DO_IT
```

```
;  2726     2785  7
;  2727     2786  7              ! { end of didn't find a duplicate, put record in its own bucket }
;  2728     2787  7              !
;  2729     2788  6              END;
;  2730     2789  6
;  2731     2790  6          REC_ADDR = .REC_ADDR + .REC_OVHD + .S_REC_SIZE;
;  2732     2791  5          END;                        ! { end of while no duplicate has been found }
;  2733     2792  5
;  2734     2793  4      END;                    ! { end of block defining status for while loop }
;  2735     2794  4
;  2736     2795  4      ! Found the beginning of the dups chain, now find the end.
;  2737     2796  4      !
;  2738     2797  4      BEG_CHAIN = .REC_ADDR;
;  2739     2798  4
;  2740     2799  5      BEGIN
;  2741     2800  5
;  2742     2801  5      LOCAL
;  2743     2802  5          NOT_DUP,
;  2744     2803  5          REC_OVHD,
;  2745     2804  5          S_REC_SIZE;
;  2746     2805  5
;  2747     2806  5      NOT_DUP = 0;                                    ! assume more duplicates
;  2748     2807  5      REC_OVHD = RM$REC_OVHD(0; S_REC_SIZE);
;  2749     2808  5
;  2750     2809  5      ! Ok, keep track of how much the first key would expand if placed
;  2751     2810  5      !  at the beginning of a new bucket.
;  2752     2811  5      !
;  2753     2812  5      IF .IDX_DFN[IDX$V_KEY_COMPR]
;  2754     2813  5      THEN
;  2755     2814  5          FIRST_KEY_EXPANSION = .(.REC_ADDR + .REC_OVHD + 1)<0,8>;
;  2756     2815  5
;  2757     2816  5      DO
;  2758     2817  6          BEGIN
;  2759     2818  6          !
;  2760     2819  6          REC_ADDR = .REC_ADDR + .REC_OVHD + .S_REC_SIZE;
;  2761     2820  6          IF .REC_ADDR EQLU .EOB
;  2762     2821  6          THEN
;  2763     2822  6              EXITLOOP;
;  2764     2823  6          !
;  2765     2824  6          REC_OVHD = RM$REC_OVHD(0; S_REC_SIZE);
;  2766     2825  6          !
;  2767     2826  6          IF .IDX_DFN[IDX$V_KEY_COMPR]
;  2768     2827  6          THEN
;  2769     2828  7              BEGIN
;  2770     2829  7              !
;  2771     2830  7              IF .(.REC_ADDR + .REC_OVHD)<0,8> NEQU 0
;  2772     2831  7              THEN
;  2773     2832  7                  NOT_DUP = 1
;  2774     2833  7              END
;  2775     2834  6          ELSE
;  2776     2835  7              BEGIN
;  2777     2836  7              !
;  2778     2837  7              AP = 3;                         ! Contiguous compare of keys
;  2779     2838  7              !
;  2780     2839  7              IF RM$COMPARE_KEY ( .REC_ADDR + .REC_OVHD,
;  2781     2840  7                                  KEYBUF_ADDR(3),
;  2782     2841  7                                  .IDX_DFN[IDX$B_KEYSZ] )
```

```
;  2783     2842  7              THEN
;  2784     2843  7                  NOT_DUP = 1
;  2785     2844  6              END;
;  2786     2845  6
;  2787     2846  6          END
;  2788     2847  6
;  2789     2848  5      UNTIL .NOT_DUP
;  2790     2849  4      END;                            ! end of found end of dups chain
;  2791     2850  4
;  2792     2851  4      END_CHAIN = .REC_ADDR;
;  2793     2852  4
;  2794     2853  4      ! Found the beginning and the end of the chain.  Calculate its size.
;  2795     2854  4      ! If we got here via an update, we never called RM$SRCH_BY_KEY to set
;  2796     2855  4      ! DUPS_SEEN for us, so let us do that now if necessary.  Also be sure
;  2797     2856  4      ! to factor in the amount of key expansion that the first key would
;  2798     2857  4      ! undergo if placed first in a new bucket.  If the keys aren't
;  2799     2858  4      ! compressed, don't sweat it -- FIRST_KEY_EXPANSION was initialized
;  2800     2859  4      ! to zero, and only changed if key compression is in effect.
;  2801     2860  4
;  2802     2861  4      IF .POS_INSERT GTRU .BEG_CHAIN
;  2803     2862  4      THEN
;  2804     2863  4          IRAB[IRB$V_DUPS_SEEN] = 1;
;  2805     2864  4
;  2806     2865  4      DUPS = .END_CHAIN - .BEG_CHAIN;
;  2807     2866  4      DUPS = .DUPS + .RECSZ + .FIRST_KEY_EXPANSION;
;  2808     2867  4
;  2809     2868  4      IF .DUPS LSSU .BKTSIZE
;  2810     2869  4      THEN
;  2811     2870  5          BEGIN
;  2812     2871  5
;  2813     2872  5          !+
;  2814     2873  5          ! !!!! SPLIT TYPE 1 !!!!!
;  2815     2874  5          ! Duplicates found and fortunately, they all fit in one bucket,
;  2816     2875  5          ! so do a 3-bkt split with all of the dups in the middle bucket.
;  2817     2876  5          ! Because of the optimization used for dups being inserted "in order"
;  2818     2877  5          ! this can sitll be a 2-bkt split if the new record is being inserted
;  2819     2878  5          ! at the end of the bucket .
;  2820     2879  5          !
;  2821     2880  5          ! 22-jan-79  If LOA forced us to think that a bkt with all dups had to
;  2822     2881  5          ! be split ( only on put) be smart and just put new record by itself.
;  2823     2882  5          ! A better solution would be not to split at all, but at this date
;  2824     2883  5          ! it's rather inconceivable.
;  2825     2884  5          !
;  2826     2885  5          ! 23-jan-79  It's not only LOA that can fool us, the bkt might have
;  2827     2886  5          ! had a lot of rrv's.
;  2828     2887  5          !-
;  2829     2888  5
;  2830     2889  5          IRAB[IRB$W_SPLIT] = .BEG_CHAIN - .BKT_ADDR;
;  2831     2890  5          IRAB[IRB$W_SPLIT_1] = .END_CHAIN - .BKT_ADDR;
;  2832     2891  5
;  2833     2892  5          IF .END_CHAIN EQLU .EOB
;  2834     2893  5          THEN
;  2835     2894  6              BEGIN
;  2836     2895  6              IRAB[IRB$V_NEW_BKTS] = 1;
;  2837     2896  6
;  2838     2897  7              IF .BEG_CHAIN EQLU (.BKT_ADDR + BKT$C_OVERHDSZ)
;  2839     2898  6              THEN
```

```
;  2840     2899  7                BEGIN
;  2841     2900  7                IRAB[IRB$W_SPLIT_1] = .IRAB[IRB$W_SPLIT_2];
;  2842     2901  7                IRAB[IRB$W_SPLIT] = .IRAB[IRB$W_POS_INS];
;  2843     2902  7                IRAB[IRB$V_CONT_BKT] = 1;
;  2844     2903  7                END
;  2845     2904  7
;  2846     2905  6            END
;  2847     2906  5        ELSE
;  2848     2907  6            BEGIN
;  2849     2908  6
;  2850     2909  6            IF .IRAB[IRB$W_SPLIT] EQLU BKT$C_OVERHDSZ<0, 16>
;  2851     2910  6            THEN
;  2852     2911  6                IRAB[IRB$V_EMPTY_BKT] = 1;
;  2853     2912  6
;  2854     2913  6            ! Only force record into the low bucket if it is not the first
;  2855     2914  6            ! one in a duplicate chain.
;  2856     2915  6            !
;  2857     2916  6
;  2858     2917  6            IF .END_CHAIN GEQU .POS_INSERT
;  2859     2918  6              AND .IRAB[IRB$W_SPLIT] NEQU .IRAB[IRB$W_POS_INS]
;  2860     2919  6            THEN
;  2861     2920  6                IRAB[IRB$V_REC_W_LO] = 1;
;  2862     2921  5            END;
;  2863     2922  5
;  2864     2923  5        LEAVE DO_IT
;  2865     2924  5
;  2866     2925  4        END;    ! { end of duplicates found and they fit in one bucket }
;  2867     2926  4
;  2868     2927  4    ! This next test can only happen on an update so the all dupes case
;  2869     2928  4    ! will fall thru to split type 2, which will put the new record by itself.
;  2870     2929  4    ! Consider oddball update case in which there are dups before and after
;  2871     2930  4    ! position of insert. ( note that if this case doesn't apply, the duplicates
;  2872     2931  4    ! were only before or after -- and didn't fit with record -- so new record
;  2873     2932  4    ! will end up by itself.  For code flow purposes, leave that till later).
;  2874     2933  4    !
;  2875     2934  4
;  2876     2935  4    IF  .IRAB[IRB$V_DUPS_SEEN]
;  2877     2936  4        AND
;  2878     2937  4        .END_CHAIN GTRU .POS_INSERT
;  2879     2938  4    THEN
;  2880     2939  5        BEGIN
;  2881     2940  5
;  2882     2941  5        IF .DUPS - (.POS_INSERT - .BEG_CHAIN) LSSU .BKTSIZE
;  2883     2942  5        THEN
;  2884     2943  5
;  2885     2944  5            ! if high dups will fit with record, put them in a bucket together
;  2886     2945  5            !
;  2887     2946  6            BEGIN
;  2888     2947  6
;  2889     2948  6            !+
;  2890     2949  6            ! !!!! SPLIT TYPE 4 !!!!
;  2891     2950  6            ! 3 bkt split where middle bkt is a continuation bkt containing
;  2892     2951  6            ! new record and dups following it
;  2893     2952  6            !
;  2894     2953  6            ! !!!! AND SPLIT TYPE 4B !!!!! however, if the hi set consists
;  2895     2954  6            ! solely of duplicates, we can still have a 2-bkt split case that
;  2896     2955  6            ! would not have been picked up by the previous algorithm ( since
```

```
;  2897     2956  6              ! it won't divide dups).
;  2898     2957  6              !-
;  2899     2958  6
;  2900     2959  6              IRAB[IRB$V_CONT_BKT] = 1;
;  2901     2960  6              IRAB[IRB$W_SPLIT] = .IRAB[IRB$W_POS_INS];
;  2902     2961  6
;  2903     2962  6              IF .END_CHAIN EQLU .EOB
;  2904     2963  6              THEN
;  2905     2964  6                  IRAB[IRB$V_NEW_BKTS] = 1
;  2906     2965  6              ELSE
;  2907     2966  6                  IRAB[IRB$W_SPLIT_1] = .END_CHAIN - .BKT_ADDR;
;  2908     2967  6
;  2909     2968  6              REC_ADDR = .BEG_CHAIN;
;  2910     2969  6              RM$MOVE ( .IDX_DFN[IDX$B_KEYSZ], KEYBUF_ADDR(3), KEYBUF_ADDR(2) );
;  2911     2970  6              LEAVE DO_IT
;  2912     2971  6
;  2913     2972  5              END;
;  2914     2973  5
;  2915     2974  5          ! try to fit new record with before-dups in middle bucket
;  2916     2975  5          !
;  2917     2976  5
;  2918     2977  5          IF .DUPS - (.END_CHAIN - .POS_INSERT) LSSU .BKTSIZE
;  2919     2978  5          THEN
;  2920     2979  6              BEGIN
;  2921     2980  6
;  2922     2981  6              !+
;  2923     2982  6              ! !!!! SPLIT TYPE 5 !!!!!
;  2924     2983  6              ! 3 or 4 bkt split ( depending on status of
;  2925     2984  6              ! high set) where left-middle bkt is new record with before-dups
;  2926     2985  6              ! and right-middle bkt, if it is needed, is a continuation bkt
;  2927     2986  6              ! with the after-dups. it is needed if the dups aren't the whole hi
;  2928     2987  6              ! set it still is a continuation bkt.
;  2929     2988  6              !
;  2930     2989  6              ! ***** NOTE FROM NOV-7-78
;  2931     2990  6              ! This case doesn't take into account the fact that the
;  2932     2991  6              ! whole bucket may be dups.  In the case of all dups, we could
;  2933     2992  6              ! end up generating an empty bucket when we don't have to (if
;  2934     2993  6              ! no RRV's) or a relatively useless bucket (some RRV's). In any
;  2935     2994  6              ! event we could end up generating an extra bucket when we
;  2936     2995  6              ! don't have to
;  2937     2996  6              !-
;  2938     2997  6
;  2939     2998  6              IRAB[IRB$W_SPLIT] = .BEG_CHAIN - .BKT_ADDR;
;  2940     2999  6              IRAB[IRB$W_SPLIT_1] = .IRAB[IRB$W_POS_INS];
;  2941     3000  6
;  2942     3001  6              IF .IRAB[IRB$W_SPLIT] EQLU BKT$C_OVERHDSZ<0, 16>
;  2943     3002  6              THEN
;  2944     3003  6                  IRAB[IRB$V_EMPTY_BKT] = 1;
;  2945     3004  6
;  2946     3005  6              IRAB[IRB$V_REC_W_LO] = 1;
;  2947     3006  6
;  2948     3007  6              IF .END_CHAIN LSSU .EOB
;  2949     3008  6              THEN
;  2950     3009  7                  BEGIN
;  2951     3010  7                  IRAB[IRB$V_NEW_BKTS] = 3;
;  2952     3011  7                  IRAB[IRB$W_SPLIT_2] = .END_CHAIN - .BKT_ADDR;
;  2953     3012  7                  END
```

```
;  2954     3013  6              ELSE
;  2955     3014  6                  IRAB[IRB$V_CONT_R] = 1;
;  2956     3015  6
;  2957     3016  6              LEAVE DO_IT
;  2958     3017  6
;  2959     3018  5              END;
;  2960     3019  5
;  2961     3020  5          ! { end of oddball update case with dups on both sides of new record }
;  2962     3021  5          !
;  2963     3022  4          END;
;  2964     3023  4
;  2965     3024  4      !+
;  2966     3025  4      ! !!!! SPLIT TYPE 2 !!!!!
;  2967     3026  4      ! the new record must go all by itself therefore,
;  2968     3027  4      ! this is a 3-bkt split if there are no after-dups or no hi set and a 4-bkt
;  2969     3028  4      ! split if both of those exist even more exceptional, this can still be a
;  2970     3029  4      ! 2-bkt split if there is no hi set at all ---- i.e., eob = end of the dups
;  2971     3030  4      ! chain
;  2972     3031  4      !-
;  2973     3032  4
;  2974     3033  4      IRAB[IRB$W_SPLIT] = IRAB[IRB$W_SPLIT_1] = .IRAB[IRB$W_POS_INS];
;  2975     3034  4
;  2976     3035  4      IF  .IRAB[IRB$V_DUPS_SEEN]
;  2977     3036  4      THEN
;  2978     3037  5          BEGIN
;  2979     3038  5          IRAB[IRB$V_CONT_BKT] = 1;
;  2980     3039  5          REC_ADDR = .BEG_CHAIN;
;  2981     3040  5          RMSMOVE ( .IDX_DFN[IDX$B_KEYSZ], KEYBUF_ADDR(3), KEYBUF_ADDR(2) );
;  2982     3041  4          END;
;  2983     3042  4
;  2984     3043  4      IF .POS_INSERT EQLU .EOB
;  2985     3044  4      THEN
;  2986     3045  4          IRAB[IRB$V_NEW_BKTS] = 1
;  2987     3046  4      ELSE
;  2988     3047  4
;  2989     3048  4          IF .POS_INSERT LSSU .END_CHAIN
;  2990     3049  4          THEN
;  2991     3050  5              BEGIN
;  2992     3051  5
;  2993     3052  5              IF .END_CHAIN LSSU .EOB
;  2994     3053  5              THEN
;  2995     3054  5                  IRAB[IRB$V_NEW_BKTS] = 3
;  2996     3055  5              ELSE
;  2997     3056  5                  IRAB[IRB$V_CONT_R] = 1;
;  2998     3057  5
;  2999     3058  5              IRAB[IRB$W_SPLIT_2] = .END_CHAIN - .BKT_ADDR;
;  3000     3059  4              END;
;  3001     3060  4
;  3002     3061  3      END;                            ! { end of block defining local symbols }
;  3003     3062  3
;  3004     3063  2      END;                                            ! { end of do_it }
;  3005     3064  2
;  3006     3065  2      ! if the first split point is at the beginning of the data, this means that
;  3007     3066  2      ! all data records will be moved out and only rrv's will be left in the
;  3008     3067  2      ! original bucket ..... therefore, we can mark this bucket as empty
;  3009     3068  2      !
;  3010     3069  2
```

```
;  3011          3070  2     IF  .IRAB[IRB$W_SPLIT] EQLU BKT$C_OVERHDSZ<0, 16>
;  3012          3071  2         AND
;  3013          3072  2         NOT .IRAB[IRB$V_REC_W_LO]
;  3014          3073  2     THEN
;  3015          3074  2         IRAB[IRB$V_EMPTY_BKT] = 1;
;  3016          3075  2
;  3017          3076  2     RETURN;
;  3018          3077  2
;  3019          3078  1     END;                                    ! { end of routine }



                                    0C  BB 00000 RM$SPLIT_UDR_3::
                                                 PUSHR   #^M<R2,R3>                          ; 1817
                                 5E                  24  C2 00002         SUBL2   #36, SP
                              44 A9          48      8F  8A 00005         BICB2   #72, 68(IRAB)          ; 1935
     44 A9             02                   01      01  F0 0000A         INSV    #1, #1, #2, 68(IRAB)   ; 1936
                                                  7E  D4 00010         CLRL    NEED_RRV               ; 1937
                                                  56  DD 00012         PUSHL   REC_ADDR               ; 1938
                              28 AE          0E      A5  9E 00014         MOVAB   14(R5), 40(SP)         ; 1939
                                 56          28      AE  D0 00019         MOVL    40(SP), REC_ADDR
                                 50          04      A5  3C 0001D         MOVZWL  4(BKT_ADDR), R0        ; 1940
                                           6045      9F  00021         PUSHAB  (R0)[BKT_ADDR]
                                                  7E  D4 00024         CLRL    LAST                   ; 1941
                          2E                     66  03  E0 00026 1$:     BBS     #3, (REC_ADDR), 4$     ; 1949
                                                 5C  03  D0 0002A         MOVL    #3, AP                 ; 1953
                                              0000G  30  0002D         BSBW    RM$RECORD_VBN          ; 1955
                                                 50  1C  A4  D1 00030         CMPL    28(BDB), R0
                                                     03  12 00034         BNEQ    2$
                                                 0C  AE  D6 00036         INCL    NEED_RRV               ; 1957
                                                 6E      56  D0 00039 2$:     MOVL    REC_ADDR, LAST         ; 1959
                          0E         1C  A7              06  E1 0003C         BBC     #6, 28(IDX_DFN), 3$    ; 1966
                                              0000G  30  00041         BSBW    RM$REC_OVHD            ; 1970
                                                 01 A046  95 00044         TSTB    1(R0)[REC_ADDR]
                                                     05  12 00048         BNEQ    3$
                                       0098      C9      56  D0 0004A         MOVL    REC_ADDR, 152(IRAB)    ; 1972
                                              0000G  30  0004F 3$:     BSBW    RM$GETNEXT_REC         ; 1975
                                         04      AE      56  D1 00052         CMPL    REC_ADDR, EOB          ; 1977
                                                     CE  1F 00056         BLSSU   1$
                                                     6E  D5 00058 4$:     TSTL    LAST                   ; 1982
                                                     1D  13 0005A         BEQL    5$
                                                 51      56  D0 0005C         MOVL    REC_ADDR, TMP_ADDR     ; 1992
                                                 56      6E  D0 0005F         MOVL    LAST, REC_ADDR         ; 1993
                                                         5C  D4 00062         CLRL    AP                     ; 1994
                                                 50  00B4   CA  3C 00064         MOVZWL  180(IFAB), R0          ; 1995
                                                 50      03  C4 00069         MULL2   #3, R0
                                                     60 B940  9F 0006C         PUSHAB  @96(IRAB)[R0]
                                                      0000G  30  00070         BSBW    RM$RECORD_KEY
                                                 5E      04  C0 00073         ADDL2   #4, SP
                                                 56      51  D0 00076         MOVL    TMP_ADDR, REC_ADDR     ; 1996
                          50                     56      55  C3 00079 5$:     SUBL3   BKT_ADDR, REC_ADDR, R0 ; 2002
                                         4E      A9      50  B0 0007D         MOVW    R0, 78(IRAB)
                                         4C      A9      50  B0 00081         MOVW    R0, 76(IRAB)
                                                 51  17  A7  9A 00085         MOVZBL  23(IDX_DFN), R1        ; 2006
                          51                     51      09  78 00089         ASHL    #9, R1, R1
```

```
                                1C   AE        F0   A1   9E 0008D          MOVAB   -16(R1), BKTSIZE                                ;
                           0A   06   A9           03   E1 00092          BBC     #3, 6(IRAB), 6$                                 ; 2011
                                78   A9        1C   A4   D1 00097          CMPL    28(BDB), 120(IRAB)                              ; 2015
                                               03   12 0009C            BNEQ    6$                                              ;
                                               0C   AE   D6 0009E          INCL    NEED_RRV                                        ; 2017
                           18   AE   04   AE           56   C3 000A1 6$:       SUBL3   REC_ADDR, EOB, RRV                      ; 2020
                                04   AE              56   D0 000A7          MOVL    REC_ADDR, EOB                                   ; 2021
                                30   AE              56   D1 000AB          CMPL    REC_ADDR, 48(SP)                                ; 2026
                                               03   12 000AF            BNEQ    7$                                              ;
                                            049E   31 000B1            BRW     68$                                             ;
                                     56         08   AE   D1 000B4 7$:       CMPL    POS_INSERT, REC_ADDR                            ; 2058
                                               0D   12 000B8            BNEQ    8$                                              ;
                                51              6E   01   C1 000BA          ADDL3   #1, LAST, R1                                    ; 2060
                                     50              61   3C 000BE          MOVZWL  (R1), R0                                        ;
                                               50   D6 000C1            INCL    R0                                              ;
                                06   A5              50   B1 000C3          CMPW    R0, 6(BKT_ADDR)                                 ;
                                               03   13 000C7 8$:       BEQL    9$                                              ;
                                            0080   31 000C9            BRW     14$                                             ;
                                     56              6E   D0 000CC 9$:       MOVL    LAST, REC_ADDR                                  ; 2064
                                1F   1C   A7           06   E1 000CF          BBC     #6, 28(IDX_DFN), 12$                            ; 2073
                                           68   B9   95 000D4            TSTB    @104(IRAB)                                      ; 2077
                                               03   12 000D7            BNEQ    11$                                             ;
                                            03C3   31 000D9 10$:       BRW     62$                                             ;
                                     50      00B4   CA   3C 000DC 11$:       MOVZWL  180(IFAB), R0                                   ; 2089
                                           60 B940   9F 000E1            PUSHAB  @96(IRAB)[R0]                                   ;
                                     50      00B4   CA   3C 000E5          MOVZWL  180(IFAB), R0                                   ; 2088
                                     50         03   C4 000EA            MULL2   #3, R0                                          ;
                                           60 B940   9F 000ED            PUSHAB  @96(IRAB)[R0]                                   ;
                                               2B   11 000F1            BRB     13$                                             ; 2087
                                               51   D4 000F3 12$:       CLRL    R1                                              ; 2108
                                            0000G   30 000F5            BSBW    RM$REC_OVHD                                     ;
                                5C             03   D0 000F8            MOVL    #3, AP                                          ; 2109
                                52      00B4   CA   3C 000FB            MOVZWL  180(IFAB), R2                                   ; 2112
                                53        60 B942   3E 00100            MOVAW   @96(IRAB)[R2], R3                               ;
                           51   56           50   C1 00105            ADDL3   REC_OVHD, REC_ADDR, R1                          ; 2111
                                50        20   A7   9A 00109            MOVZBL  32(IDX_DFN), R0                                 ;
                                            0000G   30 0010D            BSBW    RM$COMPARE_KEY                                  ;
                                C6             50   E9 00110            BLBC    R0, 10$                                         ;
                                           60 B942   9F 00113            PUSHAB  @96(IRAB)[R2]                                   ; 2118
                                52             03   C4 00117            MULL2   #3, R2                                          ; 2117
                                           60 B942   9F 0011A            PUSHAB  @96(IRAB)[R2]                                   ;
                                7E        20   A7   9A 0011E 13$:       MOVZBL  32(IDX_DFN), -(SP)                              ; 2116
                                            0000G   30 00122            BSBW    RM$MOVE                                         ;
                                5E             08   C0 00125            ADDL2   #8, SP                                          ;
                                51      00B4   CA   3C 00128            MOVZWL  180(IFAB), R1                                   ; 2121
                           50   51             03   C5 0012D            MULL3   #3, R1, R0                                      ;
                                6E        60 B940   9E 00131            MOVAB   @96(IRAB)[R0], (SP)                             ;
                                           60 B941   3F 00136            PUSHAW  @96(IRAB)[R1]                                   ; 2120
                                7E        20   A7   9A 0013A            MOVZBL  32(IDX_DFN), -(SP)                              ; 2119
                                            0000G   30 0013E            BSBW    RM$MOVE                                         ;
                                5E             0C   C0 00141            ADDL2   #12, SP                                         ;
                             4A   A9        48   A9   B0 00144            MOVW    72(IRAB), 74(IRAB)                              ; 2122
                                            05EF   31 00149            BRW     96$                                             ; 2123
                                56        30   AE   D0 0014C 14$:       MOVL    48(SP), REC_ADDR                                ; 2133
                          0098   C9           56   D0 00150            MOVL    REC_ADDR, 152(IRAB)                             ; 2134
                            28   AE 7FFFFFFF   8F   D0 00155            MOVL    #2147483647, LAST_DIFF                          ; 2135
                                               6E   D4 0015D            CLRL    LAST                                            ; 2136
```

```
                                 2C   AE  D4 0015F         CLRL    SAVE_REC_W_LO                     ; 2137
       10   AE      04   AE           56  C3 00162  15$:   SUBL3   REC_ADDR, EOB, RHS                ; 2150
                    08   AE           56  D1 00168         CMPL    REC_ADDR, POS_INSERT              ; 2152
                                      0A  1A 0016C         BGTRU   16$                               ;
            05      44   A9           03  E0 0016E         BBS     #3, 68(IRAB), 16$                 ; 2154
                    10   AE      40   AE  C0 00173         ADDL2   RECSZ, RHS                        ; 2156
            3E      1C   A7           06  E1 00178  16$:   BBC     #6, 28(IDX_DFN), 20$              ; 2162
                    08   AE           56  D1 0017D         CMPL    REC_ADDR, POS_INSERT              ; 2173
                                      24  12 00181         BNEQ    19$                               ;
            1F      44   A9           03  E0 00183         BBS     #3, 68(IRAB), 19$                 ; 2175
            0C      06   A9           03  E1 00188         BBC     #3, 6(IRAB), 17$                  ; 2177
                         50    00B4   CA  3C 0018D         MOVZWL  180(IFAB), R0                     ; 2179
                         50   60 B940     DE 00192         MOVAL   @96(IRAB)[R0], R0                 ;
                                      04  11 00197         BRB     18$                               ;
                         50      68   A9  D0 00199  17$:   MOVL    104(IRAB), R0                     ; 2181
                         51      01   A0  9A 0019D  18$:   MOVZBL  1(R0), R1                         ;
                    10   AE           51  C0 001A1         ADDL2   R1, RHS                           ;
                                      14  11 001A5         BRB     20$                               ; 2177
                    04   AE           56  D1 001A7  19$:   CMPL    REC_ADDR, EOB                     ; 2191
                                      0E  1E 001AB         BGEQU   20$                               ;
                                      51  D4 001AD         CLRL    R1                                ; 2193
                                   0000G  30 001AF         BSBW    RM$REC_OVHD                       ;
                         53   01 A046     9A 001B2         MOVZBL  1(R0)[REC_ADDR], R3               ;
                    10   AE           53  C0 001B7         ADDL2   R3, RHS                           ;
                    1C   AE      10   AE  D1 001BB  20$:   CMPL    RHS, BKTSIZE                      ; 2203
                                      03  1F 001C0         BLSSU   22$                               ;
                                    0157  31 001C2  21$:   BRW     44$                               ;
                                 06   A5  B5 001C5  22$:   TSTW    6(BKT_ADDR)                       ; 2205
                                      13  12 001C8         BNEQ    23$                               ;
            0E      06   A9           03  E0 001CA         BBS     #3, 6(IRAB), 23$                  ; 2207
                         50      0E   A5  9E 001CF         MOVAB   14(R5), R0                        ; 2209
                         50           56  D1 001D3         CMPL    REC_ADDR, R0                      ;
                                      05  12 001D6         BNEQ    23$                               ;
            E5      44   A9           03  E1 001D8         BBC     #3, 68(IRAB), 21$                 ; 2211
            50           56           55  C3 001DD  23$:   SUBL3   BKT_ADDR, REC_ADDR, R0            ; 2214
                         52      F2   A0  9E 001E1         MOVAB   -14(R0), LHS                      ;
                    08   AE           56  D1 001E5         CMPL    REC_ADDR, POS_INSERT              ; 2216
                                      09  1F 001E9         BLSSU   24$                               ;
            04      44   A9           03  E1 001EB         BBC     #3, 68(IRAB), 24$                 ; 2218
                         52      40   AE  C0 001F0         ADDL2   RECSZ, LHS                        ; 2220
            50           52      18   AE  C1 001F4  24$:   ADDL3   RRV, LHS, R0                      ; 2230
       30   AE      0C   AE           09  C5 001F9         MULL3   #9, NEED_RRV, 48(SP)              ;
                         50      30   AE  C0 001FF         ADDL2   48(SP), R0                        ;
                    1C   AE           50  D1 00203         CMPL    R0, BKTSIZE                       ;
                                      0F  1A 00207         BGTRU   25$                               ;
                                 06   A5  B5 00209         TSTW    6(BKT_ADDR)                       ; 2237
                                      46  12 0020C         BNEQ    30$                               ;
            41      06   A9           03  E0 0020E         BBS     #3, 6(IRAB), 30$                  ; 2239
            3C      44   A9           03  E1 00213         BBC     #3, 68(IRAB), 30$                 ; 2241
                                      6E  D5 00218  25$:   TSTL    LAST                              ; 2245
                                      03  12 0021A         BNEQ    26$                               ;
                                    0280  31 0021C         BRW     62$                               ;
                         56           6E  D0 0021F  26$:   MOVL    LAST, REC_ADDR                    ; 2249
                         04      2C   AE  E8 00222         BLBS    SAVE_REC_W_LO, 27$                ; 2251
                    44   A9           08  8A 00226         BICB2   #8, 68(IRAB)                      ; 2253
                         53           56  D0 0022A  27$:   MOVL    REC_ADDR, TMP                     ; 2264
                         56      0E   A5  9E 0022D         MOVAB   14(R5), REC_ADDR                  ; 2265
```

```
                                 6E            56  D0 00231          MOVL    REC_ADDR, LAST                          ; 2266
                                 53            56  D1 00234 28$:     CMPL    REC_ADDR, TMP                           ; 2268
                                               75  13 00237          BEQL    38$                                     ;
                                 6E            56  D0 00239          MOVL    REC_ADDR, LAST                          ; 2271
                          0E     1C     A7      06  E1 0023C          BBC     #6, 28(IDX_DFN), 29$                    ; 2279
                                        0000G  30  00241          BSBW    RM$REC_OVHD                             ; 2283
                                      01 A046  95  00244          TSTB    1(R0)[REC_ADDR]                         ;
                                               05  12 00248          BNEQ    29$                                     ;
                                0098   C9      56  D0 0024A          MOVL    REC_ADDR, 152(IRAB)                     ; 2285
                                      0000G  30  0024F 29$:     BSBW    RM$GETNEXT_REC                          ; 2288
                                               E0  11 00252          BRB     28$                                     ; 2268
                          51           52  1C   AE  C5 00254 30$:     MULL3   BKTSIZE, LHS, R1                        ; 2332
                          50     1C    AE  30   AE  C3 00259          SUBL3   48(SP), BKTSIZE, R0                     ; 2333
                          50     18    AE  50   C3 0025F          SUBL3   R0, RRV, R0                             ;
                                           50  10   AE  C4 00264          MULL2   RHS, R0                         ;
                    24    AE             51  50   C1 00268          ADDL3   R0, R1, DIFFERENCE                      ;
                                               03  18 0026D          BGEQ    31$                                     ; 2338
                                        0099  31  0026F          BRW     43$                                     ;
                                 51     24    AE  D0 00272 31$:     MOVL    DIFFERENCE, R1                          ; 2346
                                               03  18 00276          BGEQ    32$                                     ;
                                 51            51  CE 00278          MNEGL   R1, R1                                  ;
                                 50     28    AE  D0 0027B 32$:     MOVL    LAST_DIFF, R0                           ;
                                               03  18 0027F          BGEQ    33$                                     ;
                                 50            50  CE 00281          MNEGL   R0, R0                                  ;
                                 50            51  D1 00284 33$:     CMPL    R1, R0                                  ;
                                               14  1B 00287          BLEQU   36$                                     ;
                                 6E            56  D1 00289          CMPL    REC_ADDR, LAST                          ; 2350
                                               09  13 0028C          BEQL    34$                                     ;
                                 56            6E  D0 0028E          MOVL    LAST, REC_ADDR                          ; 2355
                          08     AE            56  D1 00291          CMPL    REC_ADDR, POS_INSERT                    ; 2357
                                               04  1E 00295          BGEQU   35$                                     ;
                          44     A9            08  8A 00297 34$:     BICB2   #8, 68(IRAB)                            ; 2359
                                               6E  D4 0029B 35$:     CLRL    LAST                                    ; 2362
                                               6E  D5 0029D 36$:     TSTL    LAST                                    ; 2369
                                               2A  12 0029F          BNEQ    40$                                     ;
                                 53            56  D0 002A1          MOVL    REC_ADDR, TMP                           ; 2376
                                 56     0E     A5  9E 002A4          MOVAB   14(R5), REC_ADDR                        ; 2377
                                 6E            56  D0 002A8          MOVL    REC_ADDR, LAST                          ; 2378
                                 53            56  D1 002AB 37$:     CMPL    REC_ADDR, TMP                           ; 2380
                                               1B  13 002AE 38$:     BEQL    40$                                     ;
                                 6E            56  D0 002B0          MOVL    REC_ADDR, LAST                          ; 2383
                          0E     1C     A7      06  E1 002B3          BBC     #6, 28(IDX_DFN), 39$                    ; 2391
                                        0000G  30  002B8          BSBW    RM$REC_OVHD                             ; 2395
                                      01 A046  95  002BB          TSTB    1(R0)[REC_ADDR]                         ;
                                               05  12 002BF          BNEQ    39$                                     ;
                                0098   C9      56  D0 002C1          MOVL    REC_ADDR, 152(IRAB)                     ; 2397
                                      0000G  30  002C6 39$:     BSBW    RM$GETNEXT_REC                          ; 2400
                                               E0  11 002C9          BRB     37$                                     ; 2380
                                 50            6E  D0 002CB 40$:     MOVL    LAST, R0                                ; 2405
                                      F76F     30  002CE          BSBW    RM$MOVE_KEY                             ;
                    4A    A9             56  55   A3 002D1          SUBW3   BKT_ADDR, REC_ADDR, 74(IRAB)            ; 2406
                          48     A9     4A  A9   B1 002D6          CMPW    74(IRAB), 72(IRAB)                      ; 2412
                                               0C  12 002DB          BNEQ    41$                                     ;
                          4C     A9     4A  A9   B1 002DD          CMPW    74(IRAB), 76(IRAB)                      ; 2415
                                               05  12 002E2          BNEQ    41$                                     ;
                          03     44     A9      03  E1 002E4          BBC     #3, 68(IRAB), 42$                       ; 2418
                                        044F  31  002E9 41$:     BRW     96$                                     ;
```

```
                 5C             03  D0 002EC 42$:   MOVL    #3, AP                                  ; 2425
                 50       00B4  CA  3C 002EF        MOVZWL  180(IFAB), R0                           ; 2428
                 53    60 B940  3E  002F4           MOVAW   @96(IRAB)[R0], R3
      51         50         60  A9  C1 002F9        ADDL3   96(IRAB), R0, R1                        ; 2427
                 50         20  A7  9A 002FE        MOVZBL  32(IDX_DFN), R0
                         0000G  30  00302           BSBW    RM$COMPARE_KEY
                 E1             50  E8 00305        BLBS    R0, 41$
                          0321  31  00308           BRW     78$                                     ; 2431
           28    AE         24  AE  D0 0030B 43$:   MOVL    DIFFERENCE, LAST_DIFF                   ; 2442
                 6E             56  D0 00310        MOVL    REC_ADDR, LAST                          ; 2443
      04   44    A9             03  E1 00313        BBC     #3, 68(IRAB), 44$                       ; 2445
           2C    AE             01  D0 00318        MOVL    #1, SAVE_REC_W_LO                       ; 2447
           08    AE             56  D1 0031C 44$:   CMPL    REC_ADDR, POS_INSERT                    ; 2457
                                03  13 00320        BEQL    46$
                          008D  31  00322 45$:      BRW     52$
      F8   44    A9             03  E0 00325 46$:   BBS     #3, 68(IRAB), 45$                       ; 2459
      0A   06    A9             03  E1 0032A        BBC     #3, 6(IRAB), 47$                        ; 2466
           78    A9         1C  A4  D1 0032F        CMPL    28(BDB), 120(IRAB)                      ; 2470
                                03  12 00334        BNEQ    47$
                 0C             AE  D7 00336        DECL    NEED_RRV                                ; 2472
           44    A9             08  88 00339 47$:   BISB2   #8, 68(IRAB)                            ; 2479
                 50       00B4  CA  3C 0033D        MOVZWL  180(IFAB), R0                           ; 2480
                       60 B940  9F  00342           PUSHAB  @96(IRAB)[R0]
                       60 B940  3F  00346           PUSHAW  @96(IRAB)[R0]
                 7E         20  A7  9A 0034A        MOVZBL  32(IDX_DFN), -(SP)
                         0000G  30  0034E           BSBW    RM$MOVE
                 5E             0C  C0 00351        ADDL2   #12, SP
           04    AE             56  D1 00354        CMPL    REC_ADDR, EOB                           ; 2488
                                03  12 00358        BNEQ    49$
                          FE05  31  0035A 48$:      BRW     15$
                                51  D4 0035D 49$:   CLRL    R1                                      ; 2501
                         0000G  30  0035F           BSBW    RM$REC_OVHD
           30    AE             50  D0 00362        MOVL    R0, REC_OVHD
      53   56         30        AE  C1 00366        ADDL3   REC_OVHD, REC_ADDR, R3                  ; 2512
      29   1C    A7             06  E1 0036B        BBC     #6, 28(IDX_DFN), 50$                    ; 2508
                 50       00B4  CA  3C 00370        MOVZWL  180(IFAB), R0                           ; 2511
                 51    60 B940  DE  00375           MOVAL   @96(IRAB)[R0], CURR_KEY
                                51  DD 0037A        PUSHL   CURR_KEY                                ; 2514
                       60 B940  9F  0037C           PUSHAB  @96(IRAB)[R0]                           ; 2513
                 7E         01  A3  9A 00380        MOVZBL  1(R3), -(SP)                            ; 2512
                         0000G  30  00384           BSBW    RM$MOVE
                 5E             08  C0 00387        ADDL2   #8, SP
                 6E             51  D0 0038A        MOVL    CURR_KEY, (SP)                          ; 2515
                       34 BE46  9F  0038D           PUSHAB  @REC_OVHD[REC_ADDR]
                          F706  30  00391           BSBW    RM$BUILD_KEY
                 5E             08  C0 00394        ADDL2   #8, SP
                                03  11 00397        BRB     51$                                     ; 2508
                 51             53  D0 00399 50$:   MOVL    R3, CURR_KEY                            ; 2518
                 5C             03  D0 0039C 51$:   MOVL    #3, AP                                  ; 2519
                 53       00B4  CA  3C 0039F        MOVZWL  180(IFAB), R3                           ; 2522
                 53         60  A9  C0 003A4        ADDL2   96(IRAB), R3
                 50         20  A7  9A 003A8        MOVZBL  32(IDX_DFN), R0                         ; 2521
                         0000G  30  003AC           BSBW    RM$COMPARE_KEY
                 A8             50  E8 003AF        BLBS    R0, 48$
                            20  AE  D4 003B2 52$:   CLRL    NOT_DUP                                 ; 2544
                                51  D4 003B5        CLRL    R1                                      ; 2548
                         0000G  30  003B7           BSBW    RM$REC_OVHD                             ;
```

```
                    14   AE          50   D0 003BA          MOVL    R0, REC_OVHD                                    ;
                    30   AE          51   D0 003BE          MOVL    R1, 48(SP)                                      ;
                OF  1C   A7          06   E1 003C2          BBC     #6, 28(IDX_DFN), 53$                            ; 2553
                50  14   AE          01   C1 003C7          ADDL3   #1, REC_OVHD, R0                                ; 2555
                                   6046   95 003CC          TSTB    (R0)[REC_ADDR]                                  ;
                                     05   12 003CF          BNEQ    53$                                             ;
                  0098   C9          56   D0 003D1          MOVL    REC_ADDR, 152(IRAB)                             ; 2557
                        53      44   A9   90 003D6 53$:     MOVB    68(IRAB), TMP                                   ; 2568
                    44   A9          08   8A 003DA          BICB2   #8, 68(IRAB)                                    ; 2569
                        50          56   D0 003DE          MOVL    REC_ADDR, R0                                    ; 2570
                                   F65C   30 003E1          BSBW    RM$MOVE_KEY                                     ;
                    44   A9          53   90 003E4          MOVB    TMP, 68(IRAB)                                   ; 2571
                    04   AE          56   D1 003E8 54$:     CMPL    REC_ADDR, EOB                                   ; 2583
                                     79   13 003EC          BEQL    60$                                             ;
                        5C          03   D0 003EE          MOVL    #3, AP                                          ; 2587
                                  0000G   30 003F1          BSBW    RM$RECORD_VBN                                   ; 2589
                        50      1C   A4   D1 003F4          CMPL    28(BDB), R0                                     ;
                                     03   12 003F8          BNEQ    55$                                             ;
                                     0C   AE   D7 003FA     DECL    NEED_RRV                                        ; 2591
                50          56   14   AE   C1 003FD 55$:     ADDL3   REC_OVHD, REC_ADDR, R0                          ; 2593
                56          50   30   AE   C1 00402          ADDL3   S_REC_SIZE, R0, REC_ADDR
                    04   AE          56   D1 00407          CMPL    REC_ADDR, EOB                                   ; 2595
                                     5A   13 0040B          BEQL    60$                                             ;
                                     51   D4 0040D          CLRL    R1                                              ; 2599
                                  0000G   30 0040F          BSBW    RM$REC_OVHD                                     ;
                    14   AE          50   D0 00412          MOVL    R0, REC_OVHD                                    ;
                    30   AE          51   D0 00416          MOVL    R1, 48(SP)                                      ;
                08  1C   A7          06   E1 0041A          BBC     #6, 28(IDX_DFN), 56$                            ; 2601
                                14  BE46   95 0041F          TSTB    @REC_OVHD[REC_ADDR]                             ; 2605
                                     21   13 00423          BEQL    58$                                             ;
                                     1B   11 00425          BRB     57$                                             ; 2607
                        5C          03   D0 00427 56$:     MOVL    #3, AP                                          ; 2611
                        53      00B4   CA   3C 0042A          MOVZWL  180(IFAB), R3                                   ; 2614
                        53      60   A9   C0 0042F          ADDL2   96(IRAB), R3
                51      56      14   AE   C1 00433          ADDL3   REC_OVHD, REC_ADDR, R1                          ; 2613
                        50      20   A7   9A 00438          MOVZBL  32(IDX_DFN), R0
                              0000G   30 0043C          BSBW    RM$COMPARE_KEY                                  ;
                        04          50   E9 0043F          BLBC    R0, 58$                                         ;
                    20   AE          01   D0 00442 57$:     MOVL    #1, NOT_DUP                                     ; 2617
                    08   AE          56   D1 00446 58$:     CMPL    REC_ADDR, POS_INSERT                            ; 2625
                                     17   12 0044A          BNEQ    59$                                             ;
                        17      20   AE   E8 0044C          BLBS    NOT_DUP, 60$                                    ; 2627
                0E      06   A9          03   E1 00450          BBC     #3, 6(IRAB), 59$                            ; 2629
                        44   A9          08   88 00455          BISB2   #8, 68(IRAB)                                ; 2633
                        78   A9      1C   A4   D1 00459          CMPL    28(BDB), 120(IRAB)                          ; 2635
                                     03   12 0045E          BNEQ    59$                                             ;
                                     0C   AE   D7 00460          DECL    NEED_RRV                                    ; 2637
                        81      20   AE   E9 00463 59$:     BLBC    NOT_DUP, 54$                                    ; 2644
                    08   AE          56   D1 00467 60$:     CMPL    REC_ADDR, POS_INSERT                            ; 2654
                                     2F   12 0046B          BNEQ    61$                                             ;
                        5C          03   D0 0046D          MOVL    #3, AP                                          ; 2661
                        50      00B4   CA   3C 00470          MOVZWL  180(IFAB), R0                                   ; 2664
                        53      60 B940   3E 00475          MOVAW   @96(IRAB)[R0], R3
                51      50      60   A9   C1 0047A          ADDL3   96(IRAB), R0, R1                                ; 2663
                        50      20   A7   9A 0047F          MOVZBL  32(IDX_DFN), R0
                              0000G   30 00483          BSBW    RM$COMPARE_KEY                                  ;
                        13          50   E8 00486          BLBS    R0, 61$                                         ;
```

```
                   44    A9           08    88 00489          BISB2   #8, 68(IRAB)                          ; 2668
              0A   06    A9           03    E1 0048D          BBC     #3, 6(IRAB), 61$                      ; 2670
                   78    A9     1C    A4    D1 00492          CMPL    28(BDB), 120(IRAB)                    ; 2672
                                      03    12 00497          BNEQ    61$                                   ;
                         0C           AE    D7 00499          DECL    NEED_RRV                              ; 2674
                                    FCC3    31 0049C  61$:    BRW     15$                                   ; 2147
                         24           AE    D4 0049F  62$:    CLRL    FIRST_KEY_EXPANSION                   ; 2704
      44    A9    02     01           02    F0 004A2          INSV    #2, #1, #2, 68(IRAB)                  ; 2716
                   44    A9           08    8A 004A8          BICB2   #8, 68(IRAB)                          ; 2717
                         50    00B4   CA    3C 004AC          MOVZWL  180(IFAB), R0                         ; 2727
                               60 B940      9F 004B1          PUSHAB  @96(IRAB)[R0]                         ;
                               60 B940      3F 004B5          PUSHAW  @96(IRAB)[R0]                         ;
                         7E     20    A7    9A 004B9          MOVZBL  32(IDX_DFN), -(SP)                    ;
                                   0000G    30 004BD          BSBW    RM$MOVE                               ;
                         5E           0C    C0 004C0          ADDL2   #12, SP                               ;
                   20    AE     0E    A5    9E 004C3          MOVAB   14(R5), 32(SP)                        ; 2732
                         56     20    AE    D0 004C8          MOVL    32(SP), REC_ADDR                      ;
                                      51    D4 004CC  63$:    CLRL    R1                                    ; 2745
                                   0000G    30 004CE          BSBW    RM$REC_OVHD                           ;
                   28    AE           50    D0 004D1          MOVL    R0, REC_OVHD                          ;
                   30    AE           51    D0 004D5          MOVL    R1, 48(SP)                            ;
              51         56     28    AE    C1 004D9          ADDL3   REC_OVHD, REC_ADDR, R1                ; 2754
              18   1C    A7           06    E1 004DE          BBC     #6, 28(IDX_DFN), 64$                  ; 2750
                         50    00B4   CA    3C 004E3          MOVZWL  180(IFAB), R0                         ; 2753
                   14    AE    60 B940      DE 004E8          MOVAL   @96(IRAB)[R0], CURR_KEY               ;
                                14    AE    DD 004EE          PUSHL   CURR_KEY                              ; 2754
                                      51    DD 004F1          PUSHL   R1                                    ;
                                    F5A4    30 004F3          BSBW    RM$BUILD_KEY                          ;
                         5E           08    C0 004F6          ADDL2   #8, SP                                ;
                                      04    11 004F9          BRB     65$                                   ; 2750
                   14    AE           51    D0 004FB  64$:    MOVL    R1, CURR_KEY                          ; 2759
                         5C           03    D0 004FF  65$:    MOVL    #3, AP                                ; 2760
                   0C    AE    00B4   CA    3C 00502          MOVZWL  180(IFAB), 12(SP)                     ; 2762
                         50     0C    AE    D0 00508          MOVL    12(SP), R0                            ;
                         53    60 B940      3E 0050C          MOVAW   @96(IRAB)[R0], R3                     ;
                         50     20    A7    9A 00511          MOVZBL  32(IDX_DFN), R0                       ; 2761
                         51     14    AE    D0 00515          MOVL    CURR_KEY, R1                          ;
                                   0000G    30 00519          BSBW    RM$COMPARE_KEY                        ;
                   2C    AE           50    D0 0051C          MOVL    R0, STATUS                            ;
                         42     2C    AE    E9 00520          BLBC    STATUS, 70$                           ; 2765
                   08    AE           56    D1 00524          CMPL    REC_ADDR, POS_INSERT                  ; 2769
                                      15    1E 00528          BGEQU   66$                                   ;
                         50     0C    AE    D0 0052A          MOVL    12(SP), R0                            ; 2771
                               60 B940      9F 0052E          PUSHAB  @96(IRAB)[R0]                         ;
                                18    AE    DD 00532          PUSHL   CURR_KEY                              ;
                         7E     20    A7    9A 00535          MOVZBL  32(IDX_DFN), -(SP)                    ;
                                   0000G    30 00539          BSBW    RM$MOVE                               ;
                         5E           0C    C0 0053C          ADDL2   #12, SP                               ;
                   04    AE           56    D1 0053F  66$:    CMPL    REC_ADDR, EOB                         ; 2773
                                      05    13 00543          BEQL    67$                                   ;
                                2C    AE    D5 00545          TSTL    STATUS                                ; 2775
                                      0F    18 00548          BGEQ    69$                                   ;
                         50     48    A9    3C 0054A  67$:    MOVZWL  72(IRAB), R0                          ; 2783
                   4C    A9           50    B0 0054E          MOVW    R0, 76(IRAB)                          ;
                   4A    A9           50    B0 00552  68$:    MOVW    R0, 74(IRAB)                          ;
                                    01E2    31 00556          BRW     96$                                   ; 2784
              50         56     28    AE    C1 00559  69$:    ADDL3   REC_OVHD, REC_ADDR, R0                ; 2790
```

```
                56          50      30   AE   C1 0055E          ADDL3   S_REC_SIZE, R0, REC_ADDR                        ;
                                  FF66   31 00563              BRW     63$                                             ; 2741
                            52          56   D0 00566 70$:     MOVL    REC_ADDR, LHS                                   ; 2797
                                    2C   AE   D4 00569          CLRL    NOT_DUP                                         ; 2806
                                         51   D4 0056C          CLRL    R1                                              ; 2807
                                    0000G 30   0056E          BSBW    RM$REC_OVHD                                     ;
                        14   AE          50   D0 00571          MOVL    R0, REC_OVHD                                    ;
                        30   AE          51   D0 00575          MOVL    R1, 48(SP)                                      ;
                    0A  1C   A7          06   E1 00579          BBC     #6, 28(IDX_DFN), 71$                            ; 2812
                    50  14   AE          01   C1 0057E          ADDL3   #1, REC_OVHD, R0                                ; 2814
                        24   AE      6046   9A 00583          MOVZBL  (R0)[REC_ADDR], FIRST_KEY_EXPANSION             ;
                        28   AE   14 BE46   9E 00588 71$:     MOVAB   @REC_OVHD[REC_ADDR], 40(SP)                     ; 2819
                    56  28   AE   30   AE   C1 0058E 72$:     ADDL3   S_REC_SIZE, 40(SP), REC_ADDR                    ;
                        04   AE          56   D1 00594          CMPL    REC_ADDR, EOB                                   ; 2820
                                         42   13 00598          BEQL    76$                                             ;
                                         51   D4 0059A          CLRL    R1                                              ; 2824
                                    0000G 30   0059C          BSBW    RM$REC_OVHD                                     ;
                        14   AE          50   D0 0059F          MOVL    R0, REC_OVHD                                    ;
                        30   AE          51   D0 005A3          MOVL    R1, 48(SP)                                      ;
                        28   AE   14 BE46   9E 005A7          MOVAB   @REC_OVHD[REC_ADDR], 40(SP)                     ; 2830
                    07  1C   A7          06   E1 005AD          BBC     #6, 28(IDX_DFN), 73$                            ; 2826
                                    28   BE   95 005B2          TSTB    @40(SP)                                         ; 2830
                                         21   13 005B5          BEQL    75$                                             ;
                                         1B   11 005B7          BRB     74$                                             ; 2832
                            5C          03   D0 005B9 73$:     MOVL    #3, AP                                          ; 2837
                            50    00B4   CA   3C 005BC          MOVZWL  180(IFAB), R0                                   ; 2840
                            53      60 B940   3E 005C1          MOVAW   @96(IRAB)[R0], R3                               ;
                            50      20   A7   9A 005C6          MOVZBL  32(IDX_DFN), R0                                 ; 2839
                            51      28   AE   D0 005CA          MOVL    40(SP), R1                                      ;
                                    0000G 30   005CE          BSBW    RM$COMPARE_KEY                                  ;
                            04          50   E9 005D1          BLBC    R0, 75$                                         ;
                        2C  AE          01   D0 005D4 74$:     MOVL    #1, NOT_DUP                                     ; 2843
                            B2      2C   AE   E9 005D8 75$:     BLBC    NOT_DUP, 72$                                    ; 2848
                        10  AE          56   D0 005DC 76$:     MOVL    REC_ADDR, RHS                                   ; 2851
                            52      08   AE   D1 005E0          CMPL    POS_INSERT, LHS                                 ; 2861
                                         05   1B 005E4          BLEQU   77$                                             ;
                        44  A9      80   8F   88 005E6          BISB2   #128, 68(IRAB)                                  ; 2863
                18  AE      10   AE          52   C3 005EB 77$:     SUBL3   LHS, RHS, RRV                               ; 2865
                    50      18   AE   40   AE   C1 005F1          ADDL3   RECSZ, RRV, R0                              ; 2866
                            18   AE   24 BE40   9E 005F7          MOVAB   @FIRST_KEY_EXPANSION[R0], RRV                   ;
                            1C   AE   18   AE   D1 005FD          CMPL    RRV, BRTSIZE                                ; 2868
                                         4D   1E 00602          BGEQU   82$                                             ;
                4A  A9                   52   55   A3 00604          SUBW3   BKT_ADDR, LHS, 74(IRAB)                     ; 2889
                4C  A9          10   AE          55   A3 00609          SUBW3   BKT_ADDR, RHS, 76(IRAB)                     ; 2890
                            04   AE   10   AE   D1 0060F          CMPL    RHS, EOB                                    ; 2892
                                         1C   12 00614          BNEQ    79$                                             ;
44  A9                  02               01   F0 00616          INSV    #1, #1, #2, 68(IRAB)                            ; 2895
                        20   AE          52   D1 0061C          CMPL    LHS, 32(SP)                                     ; 2897
                                         2D   12 00620          BNEQ    81$                                             ;
                        4C   A9     4E   A9   B0 00622          MOVW    78(IRAB), 76(IRAB)                              ; 2900
                        4A   A9     48   A9   B0 00627          MOVW    72(IRAB), 74(IRAB)                              ; 2901
                        44   A9          10   88 0062C 78$:     BISB2   #16, 68(IRAB)                                   ; 2902
                                         75   11 00630          BRB     86$                                             ; 2897
                                0E  4A   A9   B1 00632 79$:     CMPW    74(IRAB), #14                                   ; 2909
                                         05   12 00636          BNEQ    80$                                             ;
                        44   A9     40   8F   88 00638          BISB2   #64, 68(IRAB)                                   ; 2911
                        08   AE     10   AE   D1 0063D 80$:     CMPL    RHS, POS_INSERT                                 ; 2917
```

```
                                63  1F 00642       BLSSU   86$
            48  A9       4A     A9  B1 00644       CMPW    74(IRAB), 72(IRAB)                    ; 2918
                                5C  13 00649       BEQL    86$
            44  A9              08  88 0064B       BISB2   #8, 68(IRAB)                          ; 2920
                                56  11 0064F 81$:  BRB     86$                                   ; 2923
                         44     A9  95 00651 82$:  TSTB    68(IRAB)                              ; 2935
                                03  19 00654       BLSS    83$
                              0086  31 00656       BRW     89$
            08  AE       10     AE  D1 00659 83$:  CMPL    RHS, POS_INSERT                       ; 2937
                                7F  1B 0065E       BLEQU   89$
        50      52       08     AE  C3 00660       SUBL3   POS_INSERT, LHS, R0                   ; 2941
                50       18     AE  C0 00665       ADDL2   RRV, R0
            1C  AE              50  D1 00669       CMPL    R0, BKTSIZE
                                3A  1E 0066D       BGEQU   87$
            44  A9              10  88 0066F       BISB2   #16, 68(IRAB)                         ; 2959
            4A  A9       48     A9  B0 00673       MOVW    72(IRAB), 74(IRAB)                    ; 2960
            04  AE       10     AE  D1 00678       CMPL    RHS, EOB                              ; 2962
                                08  12 0067D       BNEQ    84$
 44  A9     02  01              01  F0 0067F       INSV    #1, #1, #2, 68(IRAB)                  ; 2964
                                06  11 00685       BRB     85$
    4C  A9  10  AE              55  A3 00687 84$:  SUBW3   BKT_ADDR, RHS, 76(IRAB)               ; 2966
                56              52  D0 0068D 85$:  MOVL    LHS, REC_ADDR                         ; 2968
                50     00B4     CA  3C 00690       MOVZWL  180(IFAB), R0                         ; 2969
                       60 B940  9F 00695           PUSHAB  @96(IRAB)[R0]
                       60 B940  3F 00699           PUSHAW  @96(IRAB)[R0]
                7E       20     A7  9A 0069D       MOVZBL  32(IDX_DFN), -(SP)
                            0000G  30 006A1        BSBW    RM$MOVE
                5E              0C  C0 006A4       ADDL2   #12, SP
                                72  11 006A7 86$:  BRB     91$                                   ; 2970
        50  08  AE       10     AE  C3 006A9 87$:  SUBL3   RHS, POS_INSERT, R0                   ; 2977
                50       18     AE  C0 006AF       ADDL2   RRV, R0
            1C  AE              50  D1 006B3       CMPL    R0, BKTSIZE
                                26  1E 006B7       BGEQU   89$
    4A  A9      52              55  A3 006B9       SUBW3   BKT_ADDR, LHS, 74(IRAB)               ; 2998
            4C  A9       48     A9  B0 006BE       MOVW    72(IRAB), 76(IRAB)                    ; 2999
                0E       4A     A9  B1 006C3       CMPW    74(IRAB), #14                         ; 3001
                                05  12 006C7       BNEQ    88$
            44  A9       40     8F  88 006C9       BISB2   #64, 68(IRAB)                         ; 3003
            44  A9              08  88 006CE 88$:  BISB2   #8, 68(IRAB)                          ; 3005
            04  AE       10     AE  D1 006D2       CMPL    RHS, EOB                              ; 3007
                                52  1F 006D7       BLSSU   93$
            44  A9              20  88 006D9       BISB2   #32, 68(IRAB)                         ; 3014
                                5C  11 006DD       BRB     96$                                   ; 3016
                50       48     A9  3C 006DF 89$:  MOVZWL  72(IRAB), R0                          ; 3033
            4C  A9              50  B0 006E3       MOVW    R0, 76(IRAB)
            4A  A9              50  B0 006E7       MOVW    R0, 74(IRAB)
                         44     A9  95 006EB       TSTB    68(IRAB)                              ; 3035
                                1E  18 006EE       BGEQ    90$
            44  A9              10  88 006F0       BISB2   #16, 68(IRAB)                         ; 3038
                56              52  D0 006F4       MOVL    LHS, REC_ADDR                         ; 3039
                50     00B4     CA  3C 006F7       MOVZWL  180(IFAB), R0                         ; 3040
                       60 B940  9F 006FC           PUSHAB  @96(IRAB)[R0]
                       60 B940  3F 00700           PUSHAW  @96(IRAB)[R0]
                7E       20     A7  9A 00704       MOVZBL  32(IDX_DFN), -(SP)
                            0000G  30 00708        BSBW    RM$MOVE
                5E              0C  C0 0070B       ADDL2   #12, SP
            04  AE       08     AE  D1 0070E 90$:  CMPL    POS_INSERT, EOB                       ; 3043
```

```
                                   08  12 00713          BNEQ    92$                                ;
    44   A9        02        01    01  F0 00715          INSV    #1, #1, #2, 68(IRAB)               ; 3045
                                   1E  11 0071B 91$:     BRB     96$                                ;
                     10   AE    08 AE  D1 0071D 92$:     CMPL    POS_INSERT, RHS                    ; 3048
                                   17  1E 00722          BGEQU   96$                                ;
                     04   AE    10 AE  D1 00724          CMPL    RHS, EOB                           ; 3052
                                   06  1E 00729          BGEQU   94$                                ;
                     44   A9       06  88 0072B 93$:     BISB2   #6, 68(IRAB)                       ; 3054
                                   04  11 0072F          BRB     95$                                ;
                     44   A9       20  88 00731 94$:     BISB2   #32, 68(IRAB)                      ; 3056
          4E   A9    10   AE       55  A3 00735 95$:     SUBW3   BKT_ADDR, RHS, 78(IRAB)            ; 3058
                          0E    4A A9  B1 0073B 96$:     CMPW    74(IRAB), #14                      ; 3070
                                   0A  12 0073F          BNEQ    97$                                ;
               05    44   A9       03  E0 00741          BBS     #3, 68(IRAB), 97$                  ; 3072
                     44   A9    40 8F  88 00746          BISB2   #64, 68(IRAB)                      ; 3074
                          5E       34  C0 0074B 97$:     ADDL2   #52, SP                            ; 3078
                                   0C  BA 0074E          POPR    #^M<R2,R3>                         ;
                                       05 00750          RSB                                        ;

; Routine Size:  1873 bytes,    Routine Base:  RM$RMS3 + 05C0

; 3020          3079  1
; 3021          3080  1 END
; 3022          3081  1
; 3023          3082  0 ELUDOM
```

```
;                                   PSECT SUMMARY
;
;       Name                       Bytes                     Attributes
;
;   RM$RMS3                           3345  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       63         2        154        00:00.4


;                                   COMMAND QUALIFIERS
;
;       BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS$:RM3SPLUDR/OBJ=OBJ$:RM3SPLUDR MSRC$:RM3SPLUDR/UPDATE=(ENH$:RM3SPLUDR)
```

```
; Size:          3345 code + 0 data bytes
; Run Time:        01:25.9
; Elapsed Time:    02:43.7
; Lines/CPU Min:    2152
; Lexemes/CPU-Min: 13733
; Memory Used:  634 pages
; Compilation Complete
```

RM3PROBE
LIS

RM3SIOXSP
LIS

RM3PUTERR
LIS

RM3PUTUPD
LIS

RM3SPLUDR
LIS

RM3RRV
LIS

RM3ROOT
LIS

RM3PUT
LIS

RM3SSIDR
LIS

RM3UPSIDX
LIS

RM3SRCHKY
LIS

RM3UPDDEL
LIS

RM3UPDATE
LIS