


```

RRRRRRRR MM MM 333333 PPPPPPPP CCCCCCCC KK KK UU UU NN NN PPPPPPPP
RRRRRRRR MM MM 333333 PPPPPPPP CCCCCCCC KK KK UU UU NN NN PPPPPPPP
RR RR RR MMMM MMMM 33 33 PP PP CC CC KK KK UU UU NN NN PP PP
RR RR RR MMMM MMMM 33 33 PP PP CC CC KK KK UU UU NN NN PP PP
RR RR RR MM MM MM 33 33 PP PP CC CC KK KK UU UU NNNN NN PP PP
RR RR RR MM MM MM 33 33 PP PP CC CC KK KK UU UU NNNN NN PP PP
RRRRRRRR MM MM 33 PPPPPPPP CCCCCCCC KKKKKK UU UU NN NN PPPPPPPP
RRRRRRRR MM MM 33 PPPPPPPP CCCCCCCC KKKKKK UU UU NN NN PPPPPPPP
RR RR MM MM 33 PP CC CC KK KK UU UU NN NNNN PP
RR RR MM MM 33 PP CC CC KK KK UU UU NN NNNN PP
RR RR MM MM 33 PP CC CC KK KK UU UU NN NN PP
RR RR MM MM 333333 PP CCCCCCCC KK KK UUUUUUUUU NN NN PP
RR RR MM MM 333333 PP CCCCCCCC KK KK UUUUUUUUU NN NN PP

```

```

LL 111111 SSSSSSSS
LL 111111 SSSSSSSS
LL 11 SS
LL 11 SS
LL 11 SS
LL 11 SS
LL 11 SSSSSS
LL 11 SSSSSS
LL 11 SS
LL 11 SS
LL 11 SS
LLLLLLLLLL 111111 SSSSSSSS
LLLLLLLLLL 111111 SSSSSSSS

```

(2)	229	DECLARATIONS
(3)	245	RMSCOMPRESS_KEY - does front end and rear end compression on key
(4)	339	RMSCOMPRESS_REC - does random compression on data section
(5)	468	RMSEXPAND_KEY - expand a compressed key
(6)	690	RMSPACK_REC - pack record to compressed form
(7)	820	RMSRECOMP_KEY - recompresses a compressed key
(8)	960	RMSUNPACK_REC - unpack compressed record

```
0000 1          $BEGIN RM3PCKUNP,000,RMSRMS3,<>,<PIC,NOWRT,QUAD>
0000 2
0000 3
0000 4 :*****
0000 5 :*
0000 6 :*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 7 :*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 8 :*  ALL RIGHTS RESERVED.
0000 9 :*
0000 10 :*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 11 :*  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 12 :*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 13 :*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 14 :*  OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 15 :*  TRANSFERRED.
0000 16 :*
0000 17 :*  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 18 :*  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 19 :*  CORPORATION.
0000 20 :*
0000 21 :*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 22 :*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 23 :*
0000 24 :*
0000 25 :*****
0000 26 :
0000 27 :
0000 28 :++
0000 29 :
0000 30 : Facility:      RMS32 index sequential file organization
0000 31 :
0000 32 : Abstract:
0000 33 :   This module contains the routine that is called whenever a data
0000 34 :   record that is in compressed form must be expanded.
0000 35 :
0000 36 : Environment:
0000 37 :   VAX/VMS Operating System internal exec routines.
0000 38 :
0000 39 : Author:  Maria del C. Nasr          creation date: 10-Apr-1981
0000 40 :
0000 41 : Modified By:
0000 42 :
0000 43 :   V03-011 MCN0009      Maria del C. Nasr      04-Apr-1983
0000 44 :   Change input and output registers in routine RMSCOMPRESS_KEY,
0000 45 :   so that we can use general linkage for it.
0000 46 :
0000 47 :   V03-010 MCN0008      Maria del C. Nasr      15-Mar-1983
0000 48 :   Preserve registers 2 and 3 in entry to RMSPACK_REC and
0000 49 :   RMSUNPACK_REC so that the general linkage definition can
0000 50 :   be used. Push parameter in stack when calling RMSRECORD_KEY.
0000 51 :
0000 52 :   V03-009 TMK0007      Todd M. Katz          31-Dec-1982
0000 53 :   Fixed a bug in RMSEXPAND_KEYD, the routine responsible for
0000 54 :   expanding the key of the record which follows a record that has
0000 55 :   been completely deleted and has had its space reclaimed. If the
0000 56 :   number of characters the key must be expanded is greater than
0000 57 :   the number of characters in the key on which to base the
```

0000 58 : expansion, then the rear-end truncated character of the base
0000 59 : key must be extended to provide the difference. The
0000 60 : determination as to whether this expansion would be necessary
0000 61 : was being incorrectly done, and this fixes the problem.
0000 62 :
0000 63 : V03-008 TMK0006 Todd M. Katz 29-Sep-1982
0000 64 : I made a mistake in TMK0005. Because the primary data records
0000 65 : to be unpacked in RMSUNPACK_REC are in compressed form, AP
0000 66 : must be set to 1 and not 3 before calling RMSRECORD_KEY.
0000 67 :
0000 68 : V03-007 TMK0005 Todd M. Katz 12-Sep-1982
0000 69 : Add support for prologue 3 SIDRs. Several of the routines
0000 70 : required changes to the comments, and RMSRECOMPR_KEY required
0000 71 : changes to the code. RMSRECORD_KEY also required AP to be set
0000 72 : to 3 before calling RMSRECORD_KEY instead of to 1 since
0000 73 : prologue 3 data records are always in "compressed" form.
0000 74 :
0000 75 : Delete the routines RMSCONV_FROM_ASCII and RMSCONV_TO_ASCII.
0000 76 :
0000 77 : V03-006 KBT0226 Keith B. Thompson 23-Aug-1982
0000 78 : Reorganize psects
0000 79 :
0000 80 : V03-005 KBT0065 Keith B. Thompson 17-Jun-1982
0000 81 : Remove rms\$sig_chars routine
0000 82 :
0000 83 : V03-004 TMK0004 Todd M. Katz 24-May-1982
0000 84 : Modify the routine RMSUNPACK_REC, so that the value of AP is
0000 85 : used as input. If the primary key of the data record to be
0000 86 : unpacked already lies within a keybuffer, then on input AP
0000 87 : will contain the number (1-5) of the keybuffer in which it
0000 88 : maybe found. If, on the other hand, the primary key is to be
0000 89 : extracted from its place in front of the data record and
0000 90 : re-expanded, then AP on input has been set to 0. This is a
0000 91 : performance optimization to cut down on unnecessary bucket
0000 92 : passes to obtain the primary key in its expanded form when RMS
0000 93 : already has the key in that form.
0000 94 :
0000 95 : V03-003 TMK0003 Todd M. Katz 20-Apr-1982
0000 96 : The routine RMSRECOMPR_KEY currently has an undocumented
0000 97 : assumption that the new key is only leading compressed never
0000 98 : trailing compressed when it is called. This is always true
0000 99 : at the primary data level where the key is not in the bucket
0000 100 : but in an internal record buffer. This is never true at the
0000 101 : index level (regardless if this routine is called to do record
0000 102 : sizing or the actual recompression) where the key may or may
0000 103 : not be in the actual index bucket, and it is always fully
0000 104 : compressed - both front and rear. As a result, recompression
0000 105 : at the index level may terminate prematurely allowing for
0000 106 : corruption of the file at the index level at some later point.
0000 107 : To fix this problem, remove the need for this assumption when
0000 108 : the index level is involved. We will continue to assume that
0000 109 : the new key can not be smaller than the old key when the
0000 110 : primary data level is involved.
0000 111 :
0000 112 : V03-002 TMK0002 Todd M. Katz 15-Apr-1982
0000 113 : The routine RMSRECOMPR_KEY may be called either to
0000 114 : fully recompress a key including elimination of newly

0000 115 :
0000 116 :
0000 117 :
0000 118 :
0000 119 :
0000 120 :
0000 121 :
0000 122 :
0000 123 :
0000 124 :
0000 125 :
0000 126 :
0000 127 :
0000 128 :
0000 129 :
0000 130 :
0000 131 :
0000 132 :
0000 133 :
0000 134 :
0000 135 :
0000 136 :
0000 137 :
0000 138 :
0000 139 :
0000 140 :
0000 141 :
0000 142 :
0000 143 :
0000 144 :
0000 145 :
0000 146 :
0000 147 :
0000 148 :
0000 149 :
0000 150 :
0000 151 :
0000 152 :
0000 153 :
0000 154 :
0000 155 :
0000 156 :
0000 157 :
0000 158 :
0000 159 :
0000 160 :
0000 161 :
0000 162 :
0000 163 :
0000 164 :
0000 165 :
0000 166 :
0000 167 :
0000 168 :
0000 169 :
0000 170 :
0000 171 :

compressed characters and adjustment of the bucket's freespace offset, or for just a size determination (for example during computation of the amount of space needed to hold two index record keys in a prologue 3 file with index compression during a multi-bucket split index update). A size determination is signalled by R5 (BKT_ADDR) being cleared. In such a circumstance we never want to reference any field in the nonexistent bucket. Change this routine so this is indeed the case.

V03-001 TMK0001 Todd M. Katz 24-Mar-1982
Change all references to IFB\$B_KBUFSZ to IFB\$W_KBUFSZ.

V02-013 TMK0001 Todd M. Katz 01-Mar-1982
Made several changes to the routines in this module:

1. Added support for rear-end truncation of keys in key compressed prolog 3 index buckets. This included changing the linkage of RM\$COMPRESS_KEY so that R3 serves as an output parameter (required for data level but not index key compression), changing RM\$COMPRESS_KEY so that R2 is also saved on input (and restored on output), and changing RM\$RECOMPR_KEY, so that index level keys (as well as data level keys) will be rear-end truncated as well as front-end compressed when this recompression routine is called.
2. Fixed two \$DELETE bugs which either prevented records in prolog 3 files from being deleted, or caused corruption of the file itself and problems in just closing it.
 - a. The first problem occurred because many of the branches are not unsigned when they should be. Thus, in for example RM\$EXPAND_KEYD, this would result in a failure to reexpand the duplicate of the key being deleted, and corruption of the file. To correct this, I changed all branches to be unsigned when necessary.
 - b. The second problem occurred during expansion of the key following a deleted record in RM\$EXPAND_KEY. At the time this routine is called, the record to be deleted has already been removed, and its key is residing in keybuffer 5. To expand the key that followed it (if expansion is necessary) we shift the contents of the bucket to make room for the number of characters the key is to be expanded by. Unfortunately, there was an error in the calculation of the number of characters to shift the bucket contents although the actual beginning and end points of the shift were correct, and the expansion was done correctly. The number of bytes to move included the number of characters to expand the trailing key by when it should not have. Thus, if the bucket was fairly full, and the number of characters to expand the trailing key by fairly large, garbage could be moved into the control information at the end of the bucket, and into whatever follows the bucket in the process's virtual memory space. When the number of characters to shift is correctly

0000 172 :
0000 173 :
0000 174 :
0000 175 :
0000 176 :
0000 177 :
0000 178 :
0000 179 :
0000 180 :
0000 181 :
0000 182 :
0000 183 :
0000 184 :
0000 185 :
0000 186 :
0000 187 :
0000 188 :
0000 189 :
0000 190 :
0000 191 :
0000 192 :
0000 193 :
0000 194 :
0000 195 :
0000 196 :
0000 197 :
0000 198 :
0000 199 :
0000 200 :
0000 201 :
0000 202 :
0000 203 :
0000 204 :
0000 205 :
0000 206 :
0000 207 :
0000 208 :
0000 209 :
0000 210 :
0000 211 :
0000 212 :
0000 213 :
0000 214 :
0000 215 :
0000 216 :
0000 217 :
0000 218 :
0000 219 :
0000 220 :
0000 221 :
0000 222 :
0000 223 :
0000 224 :
0000 225 :
0000 226 :
0000 227 :--

computed, not only is the deletion performed correctly and the data bucket not corrupted, but several access violations during \$CLOSE are no longer seen!

3. The last thing I did was to change some RAB references in RM\$PACK_REC, to the corresponding field in the IRAB so as to prevent having to reprobe the RAB at that point.

- V02-012 PSK002 Paulina S. Knibbe 02-Oct-1981
Fix problem in RM\$CONV_TO_ASC and RM\$CONV_FROM_ASC where the stack was being handled incorrectly.
- V02-011 PSK0001 Paulina S. Knibbe 24-Sep-1981
Fix problem in compress_rec that occurs when a record has less than four characters after primary key is extracted.
- V02-010 MCN0007 Maria del C. Nasr 24-Jul-1981
Modify RMSUNPACK_REC to do key type conversion, and use key buffer 5 as the work buffer, instead of 4. Include dummy type conversion routines.
- V02-009 MCN0006 Maria del C. Nasr 04-Aug-1981
Modify RM\$EXPAND_KEY to return the number of characters expanded. Also, eliminate calculation of significant characters in RM\$SIG_CHARS.
- V02-008 MCN0005 Maria del C. Nasr 16-Jul-1981
Modify RM\$PACK_REC and RM\$COMPRESS_KEY to use the routine RM\$FRNT_CMPR to determine the compression count of a key to be inserted without having to rebuild the previous key.
- V02-007 MCN0004 Maria del C. Nasr 14-Jul-1981
Fix problem with compression of sequences longer than 255 bytes.
- V02-006 PSK0002 Paulina S. Knibbe 10-Jul-1981
Modify recompr_key so it doesn't update the record size in index and sidr buckets.
- V02-005 MCN0003 Maria del C. Nasr 07-Jul-1981
Fix RM\$RECOMPR_KEY to deal with duplicate keys correctly.
- V02-004 MCN0002 Maria del C. Nasr 01-Jul-1981
Modify RM\$SIG_CHARS to calculate number of significant characters in three different modes.
- V02-003 MCN0001 Maria del C. Nasr 24-Jun-1981
Modify RM\$EXPAND_KEY to take care correctly of duplicate keys. Add entry_point RM\$EXPAND_KEYD to expand key after a record is deleted.
- V02-002 PSK0001 Paulina S. Knibbe
Modify RM\$RECOMPR to handle index buckets also

DECLARATIONS

```
0000 229      .SBTTL  DECLARATIONS
0000 230
0000 231      :
0000 232      : Include files:
0000 233      :
0000 234      :
0000 235      :
0000 236      : Macros:
0000 237      :
0000 238
0000 239      $BKDEF
0000 240      $RABDEF
0000 241      $IRBDEF
0000 242      $IFBDEF
0000 243      $IDXDEF
```



```

RMSCOMPRESS_KEY - does front end and rear end compression on key
0000 245 .SBTTL RMSCOMPRESS_KEY - does front end and rear end compression on key
0000 246
0000 247 :++
0000 248 :
0000 249 : FUNCTIONAL DESCRIPTION:
0000 250 :
0000 251 : This routine is called to do compression on primary data, SIDR, and
0000 252 : index level keys. It calls RMSFRNT_CMPR to determine the number of
0000 253 : characters that can be front end compressed in the key, and then does
0000 254 : rear end truncation of repeating chars.
0000 255 :
0000 256 : CALLING SEQUENCE:
0000 257 :
0000 258 : BSBW RMSCOMPRESS_KEY
0000 259 :
0000 260 : INPUT PARAMETERS:
0000 261 :
0000 262 : R0 : key to compress, including overhead
0000 263 :
0000 264 : IMPLICIT INPUTS:
0000 265 :
0000 266 : R7 : IDX_DFN - index descriptor for the primary key
0000 267 :
0000 268 : OUTPUT PARAMETERS:
0000 269 : NONE
0000 270 :
0000 271 : IMPLICIT OUTPUTS:
0000 272 : NONE
0000 273 :
0000 274 : ROUTINE VALUE:
0000 275 : Pointer past compressed key.
0000 276 :
0000 277 : SIDE EFFECTS:
0000 278 :
0000 279 : The compressed key in the buffer, with key length and compression
0000 280 : count updated.
0000 281 : Register R1 is clobbered.
0000 282 :
0000 283 : Working registers:
0000 284 :
0000 285 : R2 : key size
0000 286 : R3 : where key really starts
0000 287 : R8 : address of key to compress
0000 288 :
0000 289 : --
0000 290 :
0000 291 RMSCOMPRESS_KEY::
0000 292
60 013C 8F BB 0000 293 PUSHR #*M<R2,R3,R4,R5,R8> ; save registers
01 20 A7 90 0004 294 MOVB IDX$B_KEYSZ(R7),(R0) ; initialize length to total key size
01 01 A0 94 0008 295 CLRB 1(R0) ; init compr count to 0
52 52 60 9A 000B 296 MOVZBL (R0),R2 ; store key size for the future
53 02 A0 9E 000E 297 MOVAB 2(R0),R3 ; save start address of key
58 50 D0 0012 298 MOVL R0,R8 ; parameter for next call
51 53 D0 0015 299 MOVL R3,R1 ; assume first char will not match
FF E5 30 0018 300 BSBW RMSFRNT_CMPR ; determine compression count
50 D5 001B 301 TSTL R0 ; R0 equals zero when no front end

```

```

RM$COMPRESS_KEY - does front end and rea
OE 13 001D 302 BEQL 10$ ; compression is to be done
    001F 303
    001F 304 ;
    001F 305 ; Set front end compression values
    001F 306 ;
    001F 307 ;
01 A8 50 90 001F 308 MOVB R0,1(R8) ; set compr count
    68 50 82 0023 309 SUBB2 R0,(R8) ; decrement length by compr count
    52 50 C2 0026 310 SUBL2 R0,R2 ; determine number of chars left
51 53 50 C1 0029 311 ADDL3 R0,R3,R1 ; point to last char missed
    002D 312
    002D 313 ;
    C02D 314 ; Do rear end truncation
    002D 315 ;
    002D 316 ;
    52 D7 002D 317 10$: DECL R2 ; if one or no characters left, then
    OE 15 002F 318 BLEQ 30$ ; rear end trun cannot be done, branch
50 51 52 C1 0031 319 ADDL3 R2,R1,R0 ; find last byte in key
    70 60 91 0035 320 20$: CMPB (R0),-(R0) ; compare consecutive bytes
    05 12 0038 321 BNEQ 30$ ; if not equal, exit
    68 97 003A 322 DECB (R8) ; decrement key length
    F6 52 F5 003C 323 SOBGTR R2,20$ ; if more chars left, branch
    003F 324
    003F 325 ;
    003F 326 ; At this moment
    003F 327 ; (R8) - contains compressed key length
    003F 328 ; R1 - points to first character of compressed key
    003F 329 ; R3 - to where key should be moved
    003F 330 ;
    003F 331 ;
63 50 68 9A 003F 332 30$: MOVZBL (R8),R0 ; copy key length
    61 50 28 0042 333 MOVCS R0,(R1),(R3) ; shift chars up
    50 53 D0 0046 334 MOVL R3,R0 ; routine value
    0049 335
    013C 8F BA 0049 336 POPR #^M<R2,R3,R4,R5,R8>
    05 004D 337 RSB
  
```

RM\$COMPRESS_REC - does random compressio

```

004E 339 .SBTTL RM$COMPRESS_REC - does random compression on data section
004E 340 :++
004E 341 :
004E 342 : FUNCTIONAL DESCRIPTION:
004E 343 :
004E 344 : This routine is called to do compression on the data section of the
004E 345 : record. It searches for consecutive sequences of 5 or more
004E 346 : repeating characters, and compresses them. For each sequence that is
004E 347 : not compressed, it allocates a word to count the number of characters
004E 348 : in the data segment, and a byte to indicate the number of characters
004E 349 : compressed from the end.
004E 350 :
004E 351 : CALLING SEQUENCE:
004E 352 :
004E 353 : BSBW RM$COMPRESS_REC
004E 354 :
004E 355 : INPUT PARAMETERS:
004E 356 :
004E 357 : R10 : Pointer to next field count
004E 358 : R11 : Pointer to truncation count (end of record)
004E 359 :
004E 360 : IMPLICIT INPUTS:
004E 361 : NONE
004E 362 :
004E 363 : OUTPUT PARAMETERS:
004E 364 : NONE
004E 365 :
004E 366 : IMPLICIT OUTPUTS:
004E 367 :
004E 368 : R3 points to one byte past end of record (byte after last
004E 369 : truncation count)
004E 370 :
004E 371 : ROUTINE VALUE:
004E 372 : NONE
004E 373 :
004E 374 : SIDE EFFECTS:
004E 375 :
004E 376 : The data section is compressed
004E 377 : Registers R1,R2,R3 are clobbered
004E 378 :
004E 379 : WORKING REGISTERS:
004E 380 :
004E 381 : R3 : starting point of destination buffer
004E 382 : R4 : starting point of non-compressed field
004E 383 : R5 : starting point of possible compressed field
004E 384 : R8 : index register thru search
004E 385 : R9 : count of characters compressed
004E 386 :
004E 387 : --
004E 388 :
004E 389 RM$COMPRESS_REC:
004E 390 :
58 0F 30 8F BB 004E 391 PUSHR #^M<R4,R5,R8,R9,R10,R11>; save registers
58 5A 02 C1 0052 392 ADDL3 #2,R10,R8 ; get pointer to start of data
58 53 58 D0 0056 393 MOVL R8,R3 ; save destination buffer start addr
58 54 58 D0 0059 394 10$: MOVL R8,R4 ; reset start point
58 58 58 D1 005C 395 20$: CMPL R11,R8 ; are we all done?

```

RMSCOMPRESS_REC - does random compressio

```

03 1A 005F 396 BGTRU 25$ ; if no, branch
0070 31 0061 397 BRW 60$ ; else, exit
55 58 D0 0064 398 25$: MOVL R8,R5 ; save start point of possible match
88 88 B1 0067 399 CMPW (R8)+,(R8)+ ; compare consecutive words
FE A8 FF A8 91 006A 400 BNEQU 20$ ; if no match, try next ones
E9 12 0071 401 CMPB -1(R8),-2(R8) ; compare characters in the word
58 5B D1 0073 402 BNEQU 20$ ; if no match, try next ones
5C 1B 0076 403 CMPL R11,R8 ; if we have positioned past end
59 03 D0 0078 404 BLEQU 60$ ; then we don't have enough to compress
55 54 D1 007B 405 MOVL #3,R9 ; a match of 4 found
OC 13 007E 406 CMPL R4,R5 ; should we go back?
0080 407 BEQLU 40$ ; no, do not
0080 408
0080 409 ;
0080 410 ; Move backwards to search for any characters that might have been missed
FF A5 65 91 0080 411 30$:
06 12 0084 412 CMPB (R5),-1(R5) ; compare bytes
55 D7 0086 413 BNEQU 40$ ; no match
F4 59 06 F2 0088 414 DECL R5 ; set new match point
008C 415 AOBLSS #6,R9,30$ ; indicate another match found, and if
008C 416 ; more left, try next
008C 417 ;
008C 418 ;
008C 419 ; Look for the first character that does not match
008C 420 ;
008C 421 ;
68 52 5B 58 C3 008C 422 40$:
52 52 FF A8 3B 0090 423 SUBL3 R8,R11,R2 ; find characters left in record
58 51 D0 0095 424 SKPC -1(R8),R2,(R8) ; find first char that does not match
FFBB 59 52 03 3D 0098 425 SUBL2 R0,R2 ; find how many matched
COA1 426 MOVL R1,R8 ; set new starting addresses
00A1 427 ACBW #3,R2,R9,20$ ; increment number matched, and if less
00A1 428 ; than 5, no good
00A1 429 ;
00A1 430 ; Make sure count is not bigger than 255 bytes
00A1 431 ;
00A1 432 ;
00000FF 8F 59 D1 00A1 433 CMPL R9,#255
59 00000FF 8F 11 15 00A8 434 BLEQ 50$ ; ok, if less
59 00000FF 8F 58 59 C2 00AA 435 SUBL2 #255,R9 ; find how many extra
59 00000FF 8F C2 00B1 436 SUBL2 R9,R8 ; back out that many
00BB 437 MOVL #255,R9 ; force 255 bytes
00BB 438 ;
00BB 439 ;
00BB 440 ; A sequence long enough has been found
00BB 441 ;
00BB 442 ;
51 55 54 C3 00BB 443 50$:
63 6A 51 B0 00BF 444 INCL R1 ; find length of non-compressed
83 59 90 00C1 445 MOVW R1,(R10) ; section
5A 53 D0 00C4 446 MOVW R1,(R4),(R3) ; store next field count
53 02 C0 00C8 447 MOVW R9,(R3)+ ; move to destination buffer
FF85 31 00CB 448 MOVW R9,(R3)+ ; store truncation count
00D1 449 MOVL R3,R10 ; reset new next field addr
00D4 450 ADDL2 #2,R3 ; new next field area
00D4 451 BRW 10$
00D4 452 ;

```

RM\$COMPRESS_REC - does random compressio

```

      00D4 453 ; Move the last field if not compressed
      00D4 454 ;
      00D4 455 ;
51 58 54 D1 00D4 456 60$: CMPL R4,R8 ; was last field compressed?
      11 13 00D7 457 BEQLU 70$ ; exit, if yes
63 58 54 C3 00D9 458 SUBL3 R4,R11,R1 ; find length of section
6A 51 B0 00DD 459 MOVW R1,(R10) ; store next field count
63 51 28 00E0 460 MOV C3 R1,(R4),(R3) ; move to destination buffer
      63 94 00E4 461 CLRB (R3) ; truncation count is zero
      53 D6 00E6 462 INCL R3 ; point to end of record
      03 11 00E8 463 BRB 80$
53 02 C2 00EA 464 70$: SUBL2 #2,R3 ; point to end of record
0F30 8F BA 00ED 465 80$: POPR #^M<R4,R5,R8,R9,R10,R11>; restore registers
      05 00F1 466 RSB ; return to caller
```

RM\$EXPAND_KEY - expand a compressed key

```
00F2 468 .SBTTL RM$EXPAND_KEY - expand a compressed key
00F2 469
00F2 470 :++
00F2 471 :
00F2 472 : FUNCTIONAL DESCRIPTION:
00F2 473 :
00F2 474 : RM$EXPAND_KEY
00F2 475 :
00F2 476 : This routine is called to expand the key of the first record
00F2 477 : in the new bucket when a split occurs. It gets as input the
00F2 478 : address of the key to base expansion on, and the address of the
00F2 479 : new bucket with record to expand.
00F2 480 :
00F2 481 : RM$EXPAND_KEYD
00F2 482 :
00F2 483 : This routine is called to expand the key of the record which
00F2 484 : follows a deleted record. It gets as input the address of the
00F2 485 : key to base expansion on (deleted key), and the address of the
00F2 486 : key to expand.
00F2 487 :
00F2 488 : CALLING SEQUENCE:
00F2 489 :
00F2 490 : BSBW RM$EXPAND_KEY
00F2 491 : BSBW RM$EXPAND_KEYD
00F2 492 :
00F2 493 : INPUT PARAMETERS:
00F2 494 :
00F2 495 : RM$EXPAND_KEY
00F2 496 : R0 : address of key to base compression on
00F2 497 : R1 : address of new bucket which contains record with key to expand
00F2 498 :
00F2 499 : RM$EXPAND_KEYD
00F2 500 : R0 : address of key to base compression on with compression overhead
00F2 501 : R1 : address of key to expand with compression overhead
00F2 502 :
00F2 503 : IMPLICIT INPUTS:
00F2 504 :
00F2 505 : R5 : bucket address of bucket which contains record deleted, and
00F2 506 : record with key to be expanded.
00F2 507 : R7 : index descriptor
00F2 508 :
00F2 509 : OUTPUT PARAMETERS:
00F2 510 : NONE
00F2 511 :
00F2 512 : IMPLICIT OUTPUTS:
00F2 513 :
00F2 514 : The given key is expanded. Record size and bucket freespace are
00F2 515 : updated.
00F2 516 :
00F2 517 : ROUTINE VALUE:
00F2 518 :
00F2 519 : Number of characters expanded
00F2 520 :
00F2 521 : SIDE EFFECTS:
00F2 522 : NONE
00F2 523 :
00F2 524 : NOTES:
```

RM\$EXPAND_KEY - expand a compressed key

00F2 525 :
00F2 526 :
00F2 527 :
00F2 528 :
00F2 529 :
00F2 530 :
00F2 531 :
00F2 532 :
00F2 533 :
00F2 534 :
00F2 535 :
00F2 536 :
00F2 537 :
00F2 538 :
00F2 539 :
00F2 540 :
00F2 541 :
00F2 542 :
00F2 543 :
00F2 544 :
00F2 545 :
00F2 546 :
00F2 547 :
00F2 548 :
00F2 549 :
00F2 550 :
00F2 551 :
00F2 552 :
00F2 553 :
00F2 554 :
00F2 555 :
00F2 556 :
00F2 557 :
00F2 558 :
00F2 559 :
00F2 560 :
00F2 561 :
00F2 562 :
00F2 563 :
00F2 564 :
00F2 565 :
00F2 566 :
00F2 567 :
00F2 568 :
00F2 569 :
00F2 570 :
00F2 571 :
00F2 572 :
00F6 573 :
00F9 574 :
00FC 575 :
0100 576 :
0102 577 :
0105 578 :
0109 579 :
010C 580 :
010E 581 :

RM\$EXPAND_KEY

This routine determines the number of characters to be added in front of a compressed key, and the characters to add. It does this in the following way:

- If the compression count of the key to expand is zero, do nothing, since the key is already expanded.
- If the length of the key is greater than zero, then use the compression count as the number of characters to expand.
- If the length of the key is zero, then it is a duplicate, and do simple rear end truncation.
- If the length of the key is one, we can run into the situation in which one of the characters that was front end compressed, can be rear end truncated again, because of the order in which compression is executed. If that is the case, then the number of characters to expand will be less than the front end compression count.

RM\$EXPAND_KEYD

This routine determines the number of characters to be added in front of a compressed key, and the characters to add. It does this in the following way:

- It only expands the key if the compression count of the key to be deleted is less than the compression count of the key to be expanded.
- The number of characters to be expanded is equal to the difference between the two compression counts.

Once the number of characters to expand by is determined, all the data in the bucket from the key on is shifted down that number of bytes, and the characters are added to the front of the key from the input key. The key length and compression count are updated, and the record size, and bucket free space counts are incremented.

WORKING REGISTERS:

- R6 - number of characters to expand
- R8 - address of key to based compression on
- R9 - points to compression overhead of key to expand

RM\$EXPAND_KEY::

```

037C 8F BB
>8 50 D0
55 51 D0
56 OE A5 9E
51 D4
FEFB' 30
59 56 50 C1
01 A9 95
05 12
56 D4

```

```

PUSHR #*M<R2,R3,R4,R5,R6,R8,R9>; save registers
MOVL R0,R8 ; save address for future reference
MOVL R1,R5 ; set bucket address
MOVAB BK1$C_OVERHDSZ(R5),R6 ; set R6 to first record in the bucket
CLRL R1 ; indicate primary data level
BSBW RMSREC OVHD ; find record overhead
ADDL3 R0,R6,R9 ; R9 points to compression overhead
TSTB 1(R9) ; if compression count = 0, exit
BNEQ 10$
CLRL R6 ; no characters expanded

```

\$\$.
\$\$R
\$\$R
\$\$R
\$\$R
BKT
BKT
BKT
BKT
CC
END
EQ
EXT
EXI
GT
IDX
IDX
IDX
IDX
IDX
IDX
IFB
IRB
IRB
IRB
IRB
IRB
MOVI
OK
RMS
RMS
RMS
RMS
RMS
RMS
RMS
RMS
RMS
RMS

PSE

RMS
SAB

RM\$EXPAND_KEY - expand a compressed key

```

56 00AC 31 0110 582
01 01 A9 9A 0113 583 10$: BRW EXIT EXP
01 01 A9 94 0117 584 : R6 is the number of bytes we might
01 01 69 91 011A 585 : need to expand by
01 01 6C 13 011D 586 : compression count is zero
01 01 77 1A 011F 587 : is key length one?
: BEQL EQ_1 : branch if equal
: BGTRU GT_1 : branch if greater
: 0121 589
: 0121 590
: 0121 591 : The key length is equal to zero. Therefore, we know we have a duplicate
: 0121 592 : key. Just eliminate the characters that can be rear end truncated, and
: 0121 593 : move the rest.
: 0121 594
: 0121 595
53 56 D7 0121 596 : DECL R6 : decrement the # of chars to move
01 56 0C 15 0123 597 : BLEQ 30$ : if key was only one byte long, leave
01 56 58 C1 0125 598 : ADDL3 R8,R6,R3 : point to last char in expanded key
01 73 63 91 0129 599 20$: CMPB (R3),-(R3) : compare consecutive bytes
01 03 12 012C 600 : BNEQ 30$ : exit, when no match
01 F8 56 F5 012E 601 : SOBGTR R6,20$ : continue if any left
01 56 D6 0131 602 30$: INCL R6 : there is always one more
01 63 11 0133 603 : BRB GT_1
: 0135 604
: 0135 605 RM$EXPAND_KEYD::
: 0135 606
01 037C 8F BB 0135 607 : PUSHR #*M<R2,R3,R4,R5,R6,R8,R9> :save registers
01 58 50 7D 0139 608 : MOVQ R0,R8 : save input for future reference
01 01 A9 01 56 D4 013C 609 : CLRL R6 : no chars expanded
01 01 A9 01 01 A8 91 013E 610 : CMPB 1(R8),1(R9) : compare compression counts
01 01 A9 01 01 7A 1E 0143 611 : BGEQU EXIT EXP : continue only if deleted < next
01 01 A9 01 01 A8 83 0145 612 : SUBB3 1(R8),1(R9),R6 : set R6 to number of bytes to expand
01 01 A9 01 01 A8 90 014B 613 : MOVB 1(R8),1(R9) : compr count is that of deleted rec
01 69 95 0150 614 : TSTB (R9) : see if key len is zero
01 08 12 0152 615 : BNEQ 10$
: 0154 616
: 0154 617
: 0154 618 : The key length is equal to zero. Therefore, we know we have a duplicate
: 0154 619 : key. The compression overhead and key should be equal to the what the
: 0154 620 : key being deleted contains.
: 0154 621
: 0154 622
56 68 9A 0154 623 : MOVZBL (R8),R6 : number of characters that will be
01 58 02 C0 0157 624 : ADDL2 #2,R8 : skip compr ovhd, not needed
01 3C 11 015A 625 : BRB GT_1
: 015C 626
: 015C 627
: 015C 628
: 015C 629 : If the number of characters the key must be expanded is greater than the
: 015C 630 : number of characters in the key on which to base the expansion, then the
: 015C 631 : rear-end truncated character must be extended to provide the difference.
: 015C 632
: 015C 633
68 56 91 015C 634 10$: CMPB R6,(R8) : compare the number of chars to exp
: 015F 635 : BLEQU OK : with the key size
: 0161 636
: 0161 637
: 0161 638

```


RM\$EXPAND_KEY - expand a compressed key

```

0161 639 ; Key is too short. Must expand rear end truncated characters to make life
0161 640 ; easier.
0161 641 ;
0161 642 ;
50 68 9A 0161 643 MOVZBL (R8),R0 ; make R0 point to trun char
50 58 C0 0164 644 ADDL2 R8,R0
50 50 D6 0167 645 INCL R0
51 20 A7 51 D4 0169 646 CLRL R1 ; R1 will contain number of chars needed
51 51 01 A8 83 016B 647 SUBB3 (R8),IDX$B_KEYSZ(R7),R1
68 51 80 0170 648 SUBB2 1(R8),R1
68 55 DD 0174 649 ADDB2 R1,(R8) ; update key length
01 A0 51 60 6E 00 2C 0177 650 PUSHL R5 ; must save R5
55 8ED0 0179 651 MOVCS #0,(SP),(R0),R1,1(R0) ; extend the trun char
0180 652 POPL R5 ; restore R5
0183 653 ;
58 02 C0 0183 654 OK: ADDL2 #2,R8 ; skip compr ovhd, not needed
01 69 91 0186 655 CMPB (R9),#1 ; is key length one?
01 0D 1A 0189 656 BGTRU GT_1 ; branch if greater
0188 657 ;
0188 658 ;
0188 659 ; If the key length is one, then there is the possibility that some of the
0188 660 ; characters that were front end compressed are equal to the rear end truncated
0188 661 ; character, and therefore, they should not be added to the length of the key.
0188 662 ; To determine this, we position to the last character front end compressed in
0188 663 ; the expanded key, and start moving back, comparing each character with the
0188 664 ; last character of the compressed key, until a non-match is found or the first
0188 665 ; is reached. For each match, the number of bytes to expand by is decremented.
0188 666 ;
0188 667 ;
52 56 58 C1 0188 668 EQ_1: ADDL3 R8,R6,R2 ; point past last char that was front
018F 669 ; end compressed
02 A9 72 91 018F 670 10$: CMPB -(R2),2(R9) ; compare characters
03 12 0193 671 BNEQ GT_1 ; if not equal, exit loop
F7 56 F5 0195 672 SOBGTR R6,10$ ; decrement number to expand, and continue
0198 673 ;
FE 69 56 80 0198 674 GT_1: ADDB2 R6,(R9) ; increment length by number to expand
51 A9 56 A0 019B 675 ADDW2 R6,-2(R9) ; increment record size
51 02 A9 9E 019F 676 MOVAB 2(R9),R1 ; point to first character in key
53 04 A5 3C 01A3 677 MOVZWL BKT$W_FREESPACE(R5),R3 ; after extracting current freespace
04 A5 56 A0 01A7 678 ADDW2 R6,BKT$W_FREESPACE(R5) ; offset, adjust it to its new value
50 53 51 C3 01AB 679 ADDL2 R5,R3 ; find end of bucket address
53 51 56 C1 01AE 680 SUBL3 R1,R3,R0 ; number of bytes to shift down
63 61 50 28 01B2 681 ADDL3 R6,R1,R3 ; to where they should be moved
02 A9 68 56 28 01B6 682 MOVCS R0,(R1),(R3) ; shift them
01BF 683 MOVCS R6,(R8),2(R9) ; add the characters expanded
01BF 684 ;
01BF 685 EXIT_EXP: MOVL R6,R0 ; return number of chars expanded
037C 8F BA 01C2 686 POPR #*M<R2,R3,R4,R5,R6,R8,R9>; restore registers
05 01C6 688 RSB

```

RMSPACK_REC - pack record to compressed

```

01C7 690      .SBTTL  RMSPACK_REC - pack record to compressed form
01C7 691
01C7 692      :++
01C7 693      :
01C7 694      : FUNCTIONAL DESCRIPTION:
01C7 695      :
01C7 696      :     This routine is called when a data record must be transformed from
01C7 697      :     expanded form to compressed format.  The routine is divided in two
01C7 698      :     sections.  The first part moves the primary key to the beginning of
01C7 699      :     the output buffer, and compresses it if necessary.  The second section,
01C7 700      :     moves the data section doing compression when possible.
01C7 701
01C7 702      : CALLING SEQUENCE:
01C7 703
01C7 704      :     BSBW  RMSPACK_REC
01C7 705
01C7 706      : INPUT PARAMETERS:
01C7 707      :     NONE
01C7 708
01C7 709      : IMPLICIT INPUTS:
01C7 710
01C7 711      :     R5 : BKT_ADDR to determine if insert is at beginning of bucket
01C7 712      :     R6 : REC_ADDR - address where record is to be inserted in the bucket
01C7 713      :     R7 : IDX_DFN - index descriptor for the primary key
01C7 714      :     R8 : RAB address
01C7 715      :     R9 : IRAB address for L_KEYBUF, L_RECBUF, L_LST_NCMP, L_RBF, W_RSZ
01C7 716      :     R10: IFAB address for W_KBUFSZ
01C7 717
01C7 718      : OUTPUT PARAMETERS:
01C7 719
01C7 720      :     R0 : Actual output record size
01C7 721
01C7 722      : IMPLICIT OUTPUTS:
01C7 723      :     NONE
01C7 724
01C7 725      : ROUTINE VALUE:
01C7 726      :
01C7 727      :     Record size
01C7 728
01C7 729      : SIDE EFFECTS:
01C7 730
01C7 731      :     The packed record is in the output buffer.
01C7 732      :     Register R1 is destroyed.
01C7 733
01C7 734      : WORKING REGISTERS:
01C7 735
01C7 736      :     (SP) : stores REC_ADDR
01C7 737
01C7 738      :--
01C7 739
01C7 740  RMSPACK_REC::
01C7 741
01C7 742      PUSH  #*M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,AP>
01CB 743      PUSH  R6          ; save REC_ADDR for the time being
01CD 744      MOVL  IRB$L_RECBUF(R9),R0 ; set up output buffer pointer
01D1 745      BBC   #IDX$V_KEY COMPR,-   ; if key not compressed, branch
01D3 746      IDXB_  IDX$B_FLAGS(R7),10$

```

```

1FFC 8F  BB
      56  DD
50   68 A9  DO
      06  E1
03  1C A7

```

```

RMSPACK_REC - pack record to compressed
50 02 C0 01D6 747 ADDL2 #2,R0 ; skip key compression overhead
    01D9 748
    01D9 749 :
    01D9 750 : Prepare to extract the primary key from the user's buffer, and place it in
    01D9 751 : the record buffer
    01D9 752 :
    01D9 753 :
56 58 A9 D0 01D9 754 10$: MOVL IRB$RBF(R9),R6 ; change REC_ADDR to point to user's buf
   5C 03 D0 01DD 755 MOVL #3,AP ; indicate no rec overhead and expanded
     50 DD 01E0 756 PUSHL R0 ; output buffer pointer
   FE1B' 30 01E2 757 BSBW RMSRECORD_KEY ; move primary key out
   SE 04 C0 01E5 758 ADDL2 #4,SP
   53 50 D0 01E8 759 MOVL R0,R3 ; beginning of data section
     56 8ED0 01EB 760 POPL R6 ; restore REC_ADDR to pos of insert
     06 E1 01EE 761 BBC #IDX$V_KEY_COMPR,- ; if key not compressed, branch
   OA 1C A7 01F0 762 IDXB_FLAGS(R7),20$
    01F3 763
    01F3 764 :
    01F3 765 : Call routine to do compression on the primary key
    01F3 766 : R0 - address of key to be compressed, including overhead
    01F3 767 :
    01F3 768 :
50 68 A9 D0 01F3 769 MOVL IRB$RBUF(R9),R0 ; key to be compressed
   FE06 30 01F7 770 BSBW RMSCOMPRESS_KEY
   53 50 D0 01FA 771 MOVL R0,R3 ; store value returned
    01FD 772
    01FD 773 :
    01FD 774 : At this moment R3 is pointing to where data section should begin.
    01FD 775 :
    01FD 776 :
51 58 A9 DD 01FD 777 20$: PUSHL R3 ; save start of data section
52 56 A9 D0 01FF 778 MOVL IRB$RBF(R9),R1 ; set up pointer to input record
5B 51 52 3C 0203 779 MOVZWL IRB$RSZ(R9),R2 ; move size to a long word
   07 C1 0207 780 ADDL3 R2,R1,R1 ; determine end of input record
   03 1C A7 E1 020B 781 BBC #IDX$V_REC_COMPR,- ; branch if record not compressed
     53 02 C0 0210 782 IDXB_FLAGS(R7),40$
   58 51 D1 0213 783 ADDL2 #2,R3 ; skip next field count
     24 1E 0216 784 40$: CML R1,R1 ; are we finished?
50 58 A9 D0 0218 785 BGEQU 70$ ; branch if yes
   54 51 D0 021C 786 MOVL IRB$RBF(R9),R0 ; input buffer address
   FDDE' 30 021F 787 MOVL R1,R4 ; current byte in input buffer
   05 50 E9 0222 788 BSBW RMSCHECK_SEGMENT ; find out if byte belongs to prim key
   51 52 C0 0225 789 BLBC R0,50$ ; branch, if does not
     E9 11 0228 790 ADDL2 R2,R1 ; skip key segment
    022A 791 BR3 40$ ; try again
    022A 792
    022A 793 :
    022A 794 : Found a data segment, make sure it is not past the input buffer end
    022A 795 :
    022A 796 :
   5B 52 D1 022A 797 50$: CML R2,R1 ; is data segment too long
     03 1F 022D 798 BLSSU 60$ ; branch, if it is not
5A 52 5B D0 022F 799 MOVL R1,R2 ; otherwise, force the exact end of rec
63 52 51 C3 0232 800 60$: SUBL3 R1,R2,R10 ; find length of data segment to move
   61 5A 28 0236 801 MOVCL3 R10,(R1),(R3) ; move it
     D7 11 023A 802 BRB 40$ ; next one
   SA 8ED0 023C 803 70$: POPL R10 ; save start of data section in R10

```

```
RM$PACK_REC - pack record to compressed
06 1C A7 07 E1 023F 804 BBC #IDX$V_REC COMPR,- ; branch, if record should not be compr
          0241 805          IDX$B_FLAGS(R7),80$
          0244 806
          0244 807 :
          0244 808 : Set input parameters to RM$COMPRESS_REC
          0244 809 : R10 - start of data section
          0244 810 : R11 - end of data section
          0244 811 :
          0244 812
          5B 53 D0 0244 813 MOVL R3,R11
          FE04 30 0247 814 BSBW RM$COMPRESS_REC ; compress data section
          024A 815
50 53 68 A9 C3 024A 816 80$: SUBL3 IRB$L_RECBUF(R9),R3,R0 ; compute total record size
          1FFC 8F BA 024F 817 POPR #^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,AP>
          05 0253 818 RSB
```

.....

RMSRECOMPR_KEY - recompresses a compress

```
0254 820 .SBTTL RMSRECOMPR_KEY - recompresses a compressed key
0254 821
0254 822 :++
0254 823 :
0254 824 : FUNCTIONAL DESCRIPTION:
0254 825 :
0254 826 : This routine is called to do recompression on the primary key in
0254 827 : primary data buckets or the key in SIDR or index buckets.
0254 828 : When a new record is added, the record that follows it might need
0254 829 : modification in the primary key compression. This procedure is
0254 830 : done by the this routine.
0254 831 :
0254 832 : In addition, this routine may be called to assist in the sizing of a
0254 833 : key, for example, to assist in the sizing of the second key of a
0254 834 : multi-bucket split index update. In such a case, we never want to
0254 835 : actually remove any newly compressed characters or adjust the
0254 836 : bucket's freespace pointer (BKT_ADDR will be 0).
0254 837 :
0254 838 : A basic assumption that has been made is that the new key can never be
0254 839 : smaller than the old key when a primary data level recompression is
0254 840 : involved. This restriction does not hold for the index or SIDR level
0254 841 : (or for size determination which is at the index or SIDR level), and
0254 842 : the new key may indeed be smaller than the old key.
0254 843 :
0254 844 : CALLING SEQUENCE:
0254 845 :
0254 846 : BSBW RMSRECOMPR_KEY
0254 847 :
0254 848 : INPUT PARAMETERS:
0254 849 :
0254 850 : R0 : new record address pointing to compression overhead
0254 851 : R1 : next record address pointing to compression overhead
0254 852 :
0254 853 : IMPLICIT INPUTS:
0254 854 :
0254 855 : R5 : BKT_ADDR - BKT$W_FREESPACE
0254 856 : R7 : IDX_DFN - IDX$B_REYSZ
0254 857 :
0254 858 : OUTPUT PARAMETERS:
0254 859 : NONE
0254 860 :
0254 861 : IMPLICIT OUTPUTS:
0254 862 :
0254 863 : Record size is updated.
0254 864 : If this is not a size determination, then the newly
0254 865 : compressed characters of the next record will be removed
0254 866 : and the bucket freespace will updated.
0254 867 :
0254 868 : ROUTINE VALUE:
0254 869 : NONE
0254 870 :
0254 871 : SIDE EFFECTS:
0254 872 : NONE
0254 873 :
0254 874 : --
0254 875 :
0254 876 RMSRECOMPR_KEY::
```

RMSRECOMPR_KEY - recompresses a compress

```

033C 8F  9B  0254  877
           0254  878           PUSHR  #^M<R2,R3,R4,R5,R8,R9> ; save registers
           0258  879
           0258  880
           0258  881           ; First check for any inconsistencies
           0258  882           ;
           0258  883           ;
52  51  D0  0258  884           MOVL   R1,R2           ; set pointer to key length
           81  95  025B  885           TSTB  (R1)+          ; if the length of the next key is 0,
           76  13  025D  886           BEQL  EXIT           ; then we are doing an update
53  51  D0  025F  887           MOVL   R1,R3           ; set pointer to compression count
           80  95  0262  888           TSTB  (R0)+          ; if length of new rec is 0, then it is
           6F  13  0264  889           BEQL  EXIT           ; one of dup chain, do nothing
81  80  91  0266  890           CMPB  (R0)+,(R1)+    ; compare compression counts
           6A  12  0269  891           BNEQU EXIT          ; if new < next, then something wrong
           026B  892           ; if new > next, then no compr possible
           026B  893
           026B  894
           026B  895           ;
           026B  896           ; Get pointer to truncated character in each key
           026B  897           ;
           026B  898           ;
58  62  9A  026B  899           MOVZBL (R2),R8        ; R8 - points to trun char of next key
58  53  C0  026E  900           ADDL2  R3,R8
59  FE  A0  9A  0271  901           MOVZBL -2(R0),R9     ; R9 - points to trun char of new key
59  50  C0  0275  902           ADDL2  R0,R9
           59  D7  0278  903           DECL  R9
           027A  904
           027A  905           ;
           027A  906           ; Start comparing chars to see if compression can be done
           027A  907           ;
           027A  908           ;
61  60  91  027A  909 10$:  CMPB  (R0),(R1)      ; compare chars
           2C  12  027D  910           BNEQ  30$           ; if no match, branch
           63  96  027F  911           INCB  (R3)          ; increment compression count
58  51  D1  0281  912           CMPL  R1,R8         ; at truncated char?
           04  13  0284  913           BEQL  20$           ; branch if yes
           62  97  0286  914           DECB  (R2)          ; decrement key length
           51  D6  0288  915           INCL  R1            ; get next char
           028A  916           ;
59  50  D1  028A  917 20$:  CMPL  R0,R9         ; at truncated char of new?
           18  12  028D  918           BNEQU 25$           ; No, go on
           028F  919           ;
           55  D5  028F  920           TSTL  R5            ; if this is just a size determination
           E7  13  0291  921           BEQL  10$           ; then continue the comparison
           0293  922           ;
54  0C  A5  89  0293  923           BISB3 BKT$B_LEVEL(R5) - ; if this is an index bucket or a SIDR
           01  A5  0296  924           BNEQU BKT$B_INDEXNO(R5),R4 ; bucket then also continue the
           DF  12  0299  925           BNEQU 10$           ; comparison
           029B  926           ;
20  A7  63  71  029B  927           CMPB  (R3),IDX$B_KEYSZ(R7) ; are all chars compressed?
           D9  1F  029F  928           BLSSU 10$           ; continue if not
           02A1  929           ;
           62  94  02A1  930           CLRB  (R2)          ; set length to zero, we have a dup
           51  D6  02A3  931           INCL  R1            ; skip last char
           04  11  02A5  932           BRB   30$           ; all done...
           02A7  933

```



```

RMSUNPACK_REC - unpack compressed record
02DA 960      .SBTTL RMSUNPACK_REC - unpack compressed record
02DA 961
02DA 962      :++
02DA 963      :
02DA 964      : FUNCTIONAL DESCRIPTION:
02DA 965      :
02DA 966      :     This routine is called when a data record must be transformed from
02DA 967      :     compressed format to "normal" format. The routine is divided in two
02DA 968      :     sections. The first part moves the primary key segments to their
02DA 969      :     corresponding place in the output buffer. The second section examines
02DA 970      :     the data fields and moves them to the output buffer, expanding them at
02DA 971      :     the same time.
02DA 972      :
02DA 973      : CALLING SEQUENCE:
02DA 974      :
02DA 975      :     BSBW  RMSUNPACK_REC
02DA 976      :
02DA 977      : INPUT PARAMETERS:
02DA 978      :
02DA 979      :     R0 : output buffer address
02DA 980      :     R1 : size of input record
02DA 981      :
02DA 982      : IMPLICIT INPUTS:
02DA 983      :
02DA 984      :     R6 : REC_ADDR - pointer to record in the data bucket to be unpacked
02DA 985      :           past record size word
02DA 986      :     R9 : IRAB address for key buffer address
02DA 987      :     R10: IFAB address for key buffer size
02DA 988      :     AP : number (1-5) of keybuffer in which primary key maybe found
02DA 989      :           (in expanded form without compression overhead)
02DA 990      :           0 if key must be extracted and then re-expanded
02DA 991      :
02DA 992      : OUTPUT PARAMETERS:
02DA 993      :
02DA 994      :     R0 : Actual output record size
02DA 995      :
02DA 996      : IMPLICIT OUTPUTS:
02DA 997      :     NONE
02DA 998      :
02DA 999      : ROUTINE VALUE:
02DA 1000     :
02DA 1001     :     Record size
02DA 1002     :
02DA 1003     : SIDE EFFECTS:
02DA 1004     :
02DA 1005     :     The unpacked record is in the output buffer.
02DA 1006     :     Registers R1 are clobbered
02DA 1007     :
02DA 1008     : WORKING REGISTERS:
02DA 1009     :
02DA 1010     :     R1      : will always be the index pointer to the input buffer
02DA 1011     :     R3      : index pointer to the output buffer
02DA 1012     :     8(SP)   : size of input record
02DA 1013     :     4(SP)   : output buffer address
02DA 1014     :     (SP)    : number of bytes transferred to output record
02DA 1015     :
02DA 1016     :     First section (move primary key):

```


RMSUNPACK_REC - unpack compressed record

```

02DA 1017 . R8 : index descriptor pointer
02DA 1018 . R9 : number of segments in key
02DA 1019 . R10: key length
02DA 1020 . R11: segment length
02DA 1021 .
02DA 1022 . Second section (move data fields):
02DA 1023 . R2 : Parameter returned by RMS$CHECK_SEGMENT
02DA 1024 . R8 : Truncation count for the field in the compressed record
02DA 1025 . R9 : Points to byte past the input record
02DA 1026 . R10: Length of field in compressed record
02DA 1027 . R11: Length of data field to be moved
02DA 1028 .
02DA 1029 .--
02DA 1030
02DA 1031 RMSUNPACK_REC::
02DA 1032
02DA 1033 .
02DA 1034 . Move and expand key section - this is done by first moving the key into key
02DA 1035 . buffer 5, performing all necessary reconstruction, and expansion, and then,
02DA 1036 . calling the type conversion routine to convert each segment to its ASCII
02DA 1037 . equivalent, and move it to its place in the record.
02DA 1038 .
02DA 1039 .
1FBC 8F BB 02DA 1040 PUSHR #^M<R2,R3,R4,R5,R7,R8,R9,R10,R11,AP>; save registers
7E 50 7D 02DE 1041 MOVQ R0,-(SP) ; save size of input record and
; output buffer address
; no bytes transferred so far
00 DD 02E1 1042 PUSHL #0
7E D4 02E3 1044 CLRL -(SP) ; get key descriptor for primary key
FD18' 30 02E3 1045 BSBW RMS$KEY_DESC
SE 04 C0 02E5 1046 ADDL2 #4,SP
5C D5 02EB 1048 TSTL AP ; if primary key must be extracted
18 13 02ED 1049 BEQL 10$ ; and re-expanded then go do it
51 D4 02EF 1051 CLRL R1 ; if the primary key has already been
5C D7 02F1 1052 DECL AP ; extracted and re-expanded into a
08 13 02F3 1053 BEQL 5$ ; keybuffer, then setup r1 with the
51 00B4 CA 3C 02F5 1054 MOVZWL IFBSW_KBUFSZ(R10),R1 ; address of that keybuffer (using
51 51 5C C4 02FA 1055 MULL2 AP,R1 ; AP which contains the number of the
5A 60 A9 C0 02FD 1056 5$: ADDL2 IRBSL_KEYBUF(R9),R1 ; keybuffer), and load into r10 the
5A 20 A7 9A 0301 1057 MOVZBL IDX$B_KEYSZ(R7),R10 ; size of the primary key
1F 11 0305 1058 BRB 15$ ; before continuing
0307 1059
50 00B4 CA 3C 0307 1060 10$: MOVZWL IFBSW_KBUFSZ(R10),R0 ; determine destination buffer
50 50 04 C4 030C 1061 MULL2 #4,R0 ; parameter for next call (kbuf5)
50 60 A9 C0 030F 1062 ADDL2 IRBSL_KEYBUF(R9),R0
5C 01 D0 0313 1063 MOVL #1,AP ; indicate no record overhead/compr
50 DD 0316 1064 PUSHL R0 ; output buffer pointer
FCE5' 30 0318 1065 BSBW RMS$RECORD_KEY
SE 04 C0 031B 1066 ADDL2 #4,SP
031E 1067
5A 20 A7 9A 031E 1068 MOVZBL IDX$B_KEYSZ(R7),R10 ; get key size
51 50 5A C3 0322 1069 SUBL3 R10,R0,R1 ; find pointer to beginning of buffer
0326 1070
59 1E A7 9A 0326 1071 15$: MOVZBL IDX$B_SEGMENTS(R7),R9 ; determine number of segments
0326 1072
0326 1073

```

```

RMSUNPACK_REC - unpack compressed record
58 2C A7 3E 032A 1074      MOVAV  IDX$W_POSITION(R7),R8 ; setup pointer to pos/size/type array
      032E 1075
53 53 88 3C 032E 1076 20$:  MOVZWL  (R8)+,R3 ; get segment position
      04 AE C0 0331 1077      ADDL2  4(SP),R3 ; add output buffer start address
      5B 88 9A 0335 1078      MOVZBL  (R8)+,R11 ; get segment length
      58 D6 0338 1079      INCL   R8 ; position past segment type
63 61 5B 28 033A 1080      MOVCL3  R11,(R1),(R3) ; move the segment into output buffer
      ED 59 F5 033E 1081      SOBGTR  R9,20$ ; if more segments, continue
      6E 5A C0 0341 1082      ADDL2  R10,(SP) ; set num of bytes transferred so far
      0344 1083
      0344 1084 ;
      0344 1085 ; Move data section
      0344 1086 ;
      0344 1087 ; Registers are initialized with the following values:
      0344 1088 ; R1 - beginning of data section
      0344 1089 ; R3 - beginning of output buffer
      0344 1090 ; R9 - end of input record
      0344 1091 ;
      51 56 D0 0344 1092      MOVL   R6,R1 ; beginning of input buffer to R1
      06 E1 0347 1093      BBC    #IDX$V_KEY_COMPR,- ;
      0A 1C A7 0349 1094      #IDX$B_FLAGS(R7),30$ ; check if key compressed
      5A 81 9A 034C 1095      MOVZBL (R1)+,R10 ; if yes, get length from record
      51 51 D6 034F 1096      INCL  R1 ; skip compression count
      51 5A C0 0351 1097      ADDL2  R10,R1 ; add to rec addr
      03 11 0354 1098      BRB   40$
      51 5A C0 0356 1099 30$:  ADDL2  R10,R1 ; add fixed key size if not compressed
59 53 04 AE D0 0359 1100 40$:  MOVL  4(SP),R3 ; move output record addr to work reg
      56 08 AE C1 035D 1101      ADDL3  8(SP),R6,R9 ; add size to rec addr giving end of rec
      07 E0 0362 1102      BBS   #IDX$V_REC_COMPR,- ;
      06 1C A7 0364 1103      #IDX$B_FLAGS(R7),50$ ; check if record compressed
      5A 59 51 C3 0367 1104      SUBL3  R1,R9,R10 ; if not, compute size of data section
      03 11 0368 1105      BRB   60$
      5A 81 3C 036D 1106 50$:  MOVZWL (R1)+,R10 ; store next field count
      58 D4 0370 1107 60$:  CLRL  R8 ; truncation count should equal 0
      0372 1108
      0372 1109 ;
      0372 1110 ; Loop thru record identifying data fields and moving them to output buffer
      0372 1111 ;
      0372 1112 ;
      59 51 D1 0372 1113 MOVFLD: CMPL  R1,R9 ; at end of record?
      03 1F 0375 1114      BLSSU  10$ ; continue if not
      0083 31 0377 1115      BRW   END ; return, otherwise
      5A D5 037A 1116 10$:  TSTL  R10 ; is next field count 0?
      07 12 037C 1117      BNEQ  15$ ; continue if not
      58 D5 037E 1118      TSTL  R8 ; is truncation count 0?
      03 12 0380 1119      BNEQ  15$ ; continue if not
      0078 31 0382 1120      BRW   END ; input record is empty now
      0385 1121
      0385 1122 ;
      0385 1123 ; Set up parameters for RMS$CHECK_SEGMENT call
      0385 1124 ; R0 - start address of output buffer
      0385 1125 ; R4 - current byte in the output buffer
      0385 1126 ;
      0385 1127 ;
      50 04 AE D0 0385 1128 15$:  MOVL  4(SP),R0
      54 53 D0 0389 1129      MOVL  R3,R4
      FC71' 30 038C 1130      BSBW  RMS$CHECK_SEGMENT ; find out if data belongs to a key seg

```

RMSUNPACK_REC - unpack compressed record

```

05 50 E9 038F 1131          BLBC      R0,R20$          ; branch, if it is a data field
53 52 C0 0392 1132          ADDL2    R2,R3           ; otherwise, skip key segment
   DB 11 0395 1133          BRB      MOVFLD         ; try next field
   0397 1134
   0397 1135
   0397 1136          ; The data we are pointing at with R3 is a non-key field. Move it to the
   0397 1137          ; output buffer.
   0397 1138
   0397 1139
5B 52 53 C3 0397 1140 20$:  SUBL3    R3,R2,R11          ; find length of segment to move
   5A 5B D1 039B 1141  CONT:  CMPL    R11,R10         ; compare data len with next fld count
   OE 1E 039E 1142          BGEQU   30$            ; br, if not enough data in input field
63 61 5B 28 03A0 1143          MOVCS   R11,(R1),(R3)   ; move data length
   5A 5B C2 03A4 1144          SUBL2   R11,R10         ; find characters left in next field
   6E 5B C0 03A7 1145          ADPL2   R11,(SP)        ; increment number of bytes transferred
   5B D4 03AA 1146          CLRL   R11             ; all of data moved
   C4 11 03AC 1147          BRB     MOVFLD         ; find next segment
   5A D5 03AE 1148 30$:  TSTL    R10             ; only move nxt field if not equal to 0
   14 13 03B0 1149          BEQL   40$            ;
63 61 5A 28 03B2 1150          MOVCS   R10,(R1),(R3)   ; move next field amount of data
   5B 5A C2 03B6 1151          SUBL2   R10,R11        ; find how many left in data length
   6E 5A C0 03B9 1152          ADDL2   R10,(SP)        ; increment number of bytes transferred
   5A D4 03BC 1153          CLRL   R10             ; set next field count to zero
   07 E1 03BE 1154          BBC     #IDX$V_REC_COMP,-
03 1C A7 03C0 1155          ; check if record compressed
   58 61 9A 03C3 1156          MOVZBL  (R1),R8         ; store truncation count, if true
   58 5B D1 03C6 1157 40$:  CMPL    R11,R8         ; compare data length with trun count
   11 1E 03C9 1158          BGEQU   50$            ; branch if using trun count (smaller)
63 5B FF A1 61 00 2C 03CB 1159          MOVCS   #0,(R1),-1(R1),R11,(R3) ; fill output buffer with trun char
   58 5B C2 03D2 1160          SUBL2   R11,R8         ; truncation count is reduced
   6E 5B C0 03D5 1161          ADDL2   R11,(SP)        ; increment number of bytes tranferred
   5B D4 03D8 1162          CLRL   R11             ; data length should equal zero
   96 11 03DA 1163          BRB     MOVFLD         ; try next field
63 58 FF A1 61 00 2C 03DC 1164 50$:  MOVCS   #0,(R1),-1(R1),R8,(R3) ; fill output buffer with trun char
   5B 58 C2 03E3 1165          SUBL2   R8,R11         ; reduce data length
   6E 58 C0 03E6 1166          ADDL2   R8,(SP)        ; increment number of bytes transferred
   58 D4 03E9 1167          CLRL   R8             ; set truncation count to zero
   51 D6 03EB 1168          INCL   R1             ; get to next field addr
   59 51 D1 03ED 1169 60$:  CMPL    R1,R9         ; are we all done?
   OB 1E 03F0 1170          BGEQU   END           ; branch if yes
   5A 81 3C 03F2 1171          MOVZWL  (R1)+,R10      ; store next field count, and point
   03F5 1172          ; to data area
   5A D5 03F5 1173          TSTL   R10             ; is next field count zero?
   A2 12 03F7 1174          BNEQ   CONT          ; branch if not, more to move
   58 D5 03F9 1175          TSTL   R8             ; is truncation count zero?
   C9 12 03FB 1176          BNEQ   40$           ; branch if not, more to move
   03FD 1177
   50 BED0 03FD 1178 END:  POPL    R0             ; return size of output record
   5E 08 C0 0400 1179          ADDL2   #8,SP         ; pop output buffer addr and
   0403 1180          ; input record size
1FBC 8F BA 0403 1181          POPR    #*M<R2,R3,R4,R5,R7,R8,R9,R10,R11,AP>; restore registers
   05 0407 1182          RSB
   0408 1183
   0408 1184          .END

```

```

$$PSECT EP           = 00000000
$$RMSTEST           = 0000001A
$$RMS_PBUGCHK      = 00000010
$$RMS_TBUGCHK      = 00000008
$$RMS_UMODE        = 00000004
BKTSB_INDEXNO      = 00000001
BKTSB_LEVEL        = 0000000C
BKTSB_OVERHDSZ     = 0000000E
BKTSB_FREESPACE    = 00000004
CG,                = 0000039B R      01
END                = 000003FD R R    01
EQ 1               = 0000018B R R    01
EXIT               = 000002D5 R R    01
EXIT_EXP          = 000001BF R      01
GT 1               = 00000198 R      01
IDXSB_FLAGS       = 0000001C
IDXSB_KEYSZ       = 00000020
IDXSB_SEGMENTS    = 0000001E
IDXSV_KEY_COMPR   = 00000006
IDXSV_REC_COMPR   = 00000007
IDXSW_POSITION    = 0000002C
IFBSW_KBUFSZ      = 000000B4
IRBSL_KEYBUF      = 00000060
IRBSL_RBF         = 00000058
IRBSL_RECBUF      = 00000068
IRBSW_RSZ         = 00000056
MOVFLD            = 00000372 R      01
OK                = 00000183 R      01
RMSCHECK_SEGMENT ***** X    01
RMSCOMPRESS_KEY   = 00000000 RG    01
RMSCOMPRESS_REC   = 0000004E R      01
RMSEXPAND_KEY     = 000000F2 RG    01
RMSEXPAND_KEYD    = 00000135 RG    01
RMSFRNT_CMPR      ***** X    01
RMSKEY_DESC       ***** X    01
RMSPACK_REC       = 000001C7 RG    01
RMSRECOMPRESS_KEY = 00000254 RG    01
RMSRECORD_KEY     ***** X    01
RMSREC_OVRD       ***** X    01
RMSUNPACK_REC     = 000002DA RG    01
    
```

-----+
Psect synopsis !
-----+

PSECT name	Allocation	PSECT No.	Attributes
. ABS	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
RMSRMS3	00000408 (1032.)	01 (1.)	PIC USR CON REL GBL NOSHR EXE RD NOWRT NOVEC QUAD
\$ABSS	00000000 (0.)	02 (2.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE

! Performance indicators !

Phase	Page faults	CPU Time	Elapsed Time
Initialization	30	00:00:00.07	00:00:00.86
Command processing	111	00:00:00.82	00:00:05.09
Pass 1	266	00:00:07.21	00:00:18.73
Symbol table sort	0	00:00:00.84	00:00:01.46
Pass 2	213	00:00:02.92	00:00:09.71
Symbol table output	6	00:00:00.06	00:00:00.06
Psect synopsis output	2	00:00:00.02	00:00:00.05
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	630	00:00:11.96	00:00:35.97

The working set limit was 1500 pages.
45006 bytes (88 pages) of virtual memory were used to buffer the intermediate code.
There were 40 pages of symbol table space allocated to hold 594 non-local and 44 local symbols.
1184 source lines were read in Pass 1, producing 15 object records in Pass 2.
14 pages of virtual memory were used to define 13 macros.

! Macro library statistics !

Macro library name	Macros defined
_\$255\$DUA28:[RMS.OBJ]RMS.MLB;1	6
-\$255\$DUA28:[SYS.OBJ]LIB.MLB;1	0
-\$255\$DUA28:[SYSLIB]STARLET.MLB;2	3
TOTALS (all libraries)	9

658 GETS were required to define 9 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS\$:RM3PCKUNP/OBJ=OBJ\$:RM3PCKUNP MSRCS\$:RM3PCKUNP/UPDATE=(ENHS:RM3PCKUNP)+EXECMLS/LIB+LIBS:RMS/LIB

