```
RRRRRRRRRRRR     MMM         MMM     SSSSSSSSSSSS
RRRRRRRRRRRR     MMM         MMM     SSSSSSSSSSSS
RRRRRRRRRRRR     MMM         MMM     SSSSSSSSSSSS
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
RRRRRRRRRRRR     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  SSSSSSSSSSSS
RRR         RRR  MMM         MMM  SSSSSSSSSSSS
RRR         RRR  MMM         MMM  SSSSSSSSSSSS
```

FILEID**RM3SSIDR

```
RRRRRRRR    MM      MM    333333      SSSSSSSS    SSSSSSSS    IIIIII      DDDDDDDD    RRRRRRRR
RRRRRRRR    MM      MM    333333      SSSSSSSS    SSSSSSSS    IIIIII      DDDDDDDD    RRRRRRRR
RR      RR  MMMM  MMMM  33      33  SS          SS              II        DD      DD  RR      RR
RR      RR  MMMM  MMMM  33      33  SS          SS              II        DD      DD  RR      RR
RR      RR  MM  MM  MM          33  SS          SS              II        DD      DD  RR      RR
RR      RR  MM  MM  MM          33  SS          SS              II        DD      DD  RR      RR
RRRRRRRR    MM      MM        33      SSSSSS      SSSSSS        II        DD      DD  RRRRRRRR
RRRRRRRR    MM      MM        33      SSSSSS      SSSSSS        II        DD      DD  RRRRRRRR
RR  RR      MM      MM          33          SS          SS      II        DD      DD  RR  RR
RR  RR      MM      MM          33          SS          SS      II        DD      DD  RR  RR
RR    RR    MM      MM  33      33          SS          SS      II        DD      DD  RR    RR      ....
RR    RR    MM      MM  33      33          SS          SS      II        DD      DD  RR    RR      ....
RR      RR  MM      MM    333333    SSSSSSSS    SSSSSSSS      IIIIII      DDDDDDDD    RR      RR    ....
RR      RR  MM      MM    333333    SSSSSSSS    SSSSSSSS      IIIIII      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
```

RM
VO

```
0 MODULE RM3SSIDR (LANGUAGE (BLISS32) ,
0                 IDENT = 'V04-000'
0                 ) =
1 BEGIN
1 !
1 !****************************************************************************
1 !*                                                                          *
1 !*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
1 !*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
1 !*  ALL RIGHTS RESERVED.                                                    *
1 !*                                                                          *
1 !*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
1 !*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
1 !*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
1 !*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
1 !*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
1 !*  TRANSFERRED.                                                            *
1 !*                                                                          *
1 !*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
1 !*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
1 !*  CORPORATION.                                                            *
1 !*                                                                          *
1 !*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
1 !*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
1 !*                                                                          *
1 !*                                                                          *
1 !****************************************************************************
1
1 !++
1 !
1 ! FACILITY:     RMS32 INDEX SEQUENTIAL FILE ORGANIZATION
1 !
1 ! ABSTRACT:
1 !                Search thru a SIDR array and return the first
1 !                non-deleted record
1 !
1 !
1 ! ENVIRONMENT:
1 !
1 !               VAX/VMS OPERATING SYSTEM
1 !
1 !--
1
1 !
1 ! AUTHOR:        Todd M. Katz          RE-CREATION DATE:       23-Jun-1982
1 !
1 !
1 ! MODIFIED BY:
1 !
1 !       V03-022 RAS0164         Ron Schaefer            29-Jun-1983
1 !               Make references to RM$RU_RECLAIM be LONG_RELATIVE addressing.
1 !
1 !       V03-021 MCN0002         Maria del C. Nasr       24-Mar-1983
1 !               More linkages reorganization
1 !
1 !       V03-020 TMK0013         Todd M. Katz            11-Mar-1983
1 !               Make a change to RM$FOLLOW_PTR so that the primary data bucket
```

```
!                 containing the target record is exclusively accessed if it is
!                 possible that some reclamation maybe done (the file is write
!                 accessed and Recovery Unit Journallable).
!
!       V03-019 MCN0001         Maria del C. Nasr       28-Feb-1983
!               Reorganize linkages
!
!       V03-018 TMK0012         Todd M. Katz            18-Jan-1983
!               Add support for Recovery Unit Journalling and RU ROLLBACK
!               Recovery of ISAM files. Support involves modifications to
!               the routine RM$FOLLOW_PTR.
!
!               The purpose of the routines within this module is to find the
!               next non-deleted primary data record by means of an alternate
!               index. Towards this goal, RMS will search all SIDRs with key
!               values matching the key in keybuffer 2 according to the
!               characteristics of the search, until such a record is found. If
!               during its search RMS encounters records that are marked
!               RU_DELETE, RMS will try and delete them for good at this time
!               provided it has write access to the file and the Recovery Unit
!               in which they were deleted has completed successfully.
!
!               If RMS is able to delete a primary data record marked RU_DELETE,
!               then RMS proceeds to continue looking for a non-deleted primary
!               data record just as if it had encountered a deleted record in
!               the first place. Likewise, if RMS is unable to delete a record
!               that is marked RU_DELETE because it does not have write access
!               to the file, it will also continue its search. However, if RMS
!               is unable to delete a RU_DELETE marked record for good because
!               the Recovery Unit in which it was marked RU_DELETE has not
!               successfully terminated, then RMS returns this record as if it
!               was the next non-deleted primary data record, and lets a higher
!               higher level routine decide whether or not to wait for the
!               Recovery Unit in which the record was deleted to complete, or to
!               return an error to the user.
!
!               RMS will also re-format any records that are marked RU_UPDATE
!               and are in a special format provided the stream has write access
!               to the file, and the Recovery Unit in which the record was
!               updated has terminated.
!
!               In addition to this change, I have made a further enhancement to
!               the routine RM$SEARCH_SIDR. At the present time, when a SIDR
!               array element is encountered that should have been marked
!               deleted, it is marked deleted without recovering any space if
!               and only if the stream has write access to the file, and the
!               file is not being shared. I have changed this by removing the
!               restriction that to mark a SIDR element deleted the file can not
!               be shared. Furthermore, RMS will now not only mark such elements
!               as deleted, but reclaim as much space as possible as long as the
!               stream has write access to the file.
!
!       V03-017 TMK0011         Todd M. Katz            05-Jan-1983
!               Eliminate the routine RM$FND_SDR_ARRY. This global routine is
!               now being called in only one place, and has been folded directly
!               into the code.
!
```

```
!       V03-016 TMK0010         Todd M. Katz            04-Jan-1983
!               If during positioning to the next primary data record by means
!               of an alternate index, RMS encounters a SIDR element that is
!               not marked deleted although the primary data record it points
!               to is either deleted or the alternate key does not match the
!               key of the SIDR, then RMS may mark the element deleted provided
!               the file is not being shared and has been open for write access
!               before it continues its search for the next primary data record.
!               If either of these two conditions is violated, RMS can not mark
!               the element deleted, but just positions past it to continue the
!               search. What it should also be doing when it positions past the
!               element in this case is increment the count of the number of
!               preceeding elements in the current SIDR array. RMS was not
!               doing this and this represents a bug.
!
!       V03-015 TMK0009         Todd M. Katz            07-Dec-1982
!               Remove all the re-positioning code. Sigh. This code was
!               originally required to fix the 1.5 SIDR deadlock problem. I
!               have come up with a different way to fix this problem. I have
!               changed how buckets are locked during $DELETEs. A stream can now
!               never hold onto a primary data bucket while attempting to
!               access a SIDR bucket. Therefore, a stream is able to hold onto
!               a SIDR bucket while waiting for a primary data bucket without
!               the possibility of a deadlock existing. Thus, there is no
!               longer any need to perform any type of SIDR re-positioning, and
!               I have eliminated all the new code that used to do it.
!
!       V03-014 TMK0008         Todd M. Katz            12-Nov-1982
!               The routine RM$FND_SDR_ARRY requires as input, the size of the
!               secondary key it is to search for in IRB$B_KEYSZ. This IRAB
!               cell might not have the correct key size when this routine is
!               called from within RM$SIDR_REPOS. Yet, RM$SIDR_REPOS can not
!               merely overwrite the cell with the correct key size because
!               RMS might be performing a generic search at the time SIDR
!               re-positioning was required, and might later need the value
!               stored there. Therefore, RMS must save the value in IRB$B_KEYSZ
!               when the routine RM$SIDR_REPOS is entered, set it to the full
!               key size, and restore the original value when the routine is
!               exited. If this is not done, the possibility exists that
!               RM$FND_SDR_ARRY might position RMS to the wrong SIDR instead
!               of to the SIDR it was positioned to when the routine was first
!               entered.
!
!       V03-013 TMK0007         Todd M. Katz            09-Nov-1982
!               Make another change to the SIDR re-positioning code. IF RMS is
!               unable to position either to the primary data record, or back to
!               the SIDR during SIDR re-positioning, then before returning RMS
!               ascertains whether it was unsuccessful in positioning to the
!               current SIDR array, or whether it was unsuccessful at
!               positioning even to the SIDR itself. The former case represents
!               an error, the latter is reasonable because the SIDR could have
!               been deleted and the space it occupied reclaimed. During this
!               attempt to re-position to the current SIDR, performed by the
!               routine RM$FIND_SIDR, RMS will position either to the current
!               SIDR (which would represent an error), or to the next SIDR that
!               followed this SIDR. However, if RMS is unable to position to the
!               current SIDR and there is no next SIDR (the end-of-file is
```

```
1 !             encountered), then RM$FIND_SIDR returns an error of RNF.
1 !             Unfortunately, the routine RM$SIDR_REPOS can not return this
1 !             status to its caller (RM$SEARCH_SIDR) because that would
1 !             indicate that RMS was unable to find the primary data record
1 !             during re-positioning - a very grave error and definitly not
1 !             the case here. When RM$SIDR_REPOS returns an error of RNF
1 !             signalling this very bad condition, RMS converts it into a BUG
1 !             error, and returns it to the user. However, the error RMS does
1 !             want to return from RM$SEARCH_SIDR in the case when RMS finds
1 !             both the primary data record and the SIDR deleted during
1 !             re-positioning and is unable to position to the next SIDR
1 !             because of encountering the end-of-file is in fact RNF! In
1 !             order to arrange things so that this error will be returned
1 !             under these circumstances, I have changed RM$SIDR_REPOS so
1 !             that it will return an error of EOF when RM$FIND_SIDR returns
1 !             RNF and RM$SEARCH_SIDR so that it changes the EOF returned by
1 !             RM$SIDR_REPOS back to RNF.
1 !
1 !     V03-012 TMK0006         Todd M. Katz            29-Oct-1982
1 !             If RMS is successful at finding the target primary data record
1 !             within RM$FOLLOW_PTR, then save the current index descriptor and
1 !             replace it with the index descriptor for the primary key before
1 !             determining record overhead and size of the primary data record.
1 !             After this determination is made, the descriptor is restored.
1 !
1 !     V03-011 TMK0005         Todd M. Katz            22-Sep-1982
1 !             If a SIDR re-positioning is required, and key compression is
1 !             enabled, then reset the variable containing the address of the
1 !             last record in the bucket encountered with a zero front
1 !             compressed key to be the first record in the bucket. This
1 !             resetting is only required when RMS must continue the search
1 !             for a non-deleted primary data record within the current SIDR,
1 !             and when the possibility exists that the search maybe continued
1 !             in a SIDR with a completly different key value necessitating
1 !             its expansion into keybuffer 2. This resetting is necessary
1 !             because during the re-positioning RMS had to release the SIDR
1 !             bucket, and then reclaim it, and while it didn't have it locked,
1 !             anything might have happened to its contents. The reason why
1 !             the variable is initialized with the address of the first SIDR
1 !             in the bucket is because the key of the first SIDR must be zero
1 !             front compressed, and it is the last known record with such a
1 !             property.
1 !
1 !     V03-010 KBT0329         Keith B. Thompson       22-Sep-1982
1 !             Change check for sharing to sfsb test
1 !
1 !     V03-009 TMK0004         Todd M. Katz            06-Sep-1982
1 !             The field IRB$B_SRCHFLAGS is now a word. Fix all references
1 !             to it.
1 !
1 !             The routine RM$SEARCH_SIDR maybe called to search SIDR arrays
1 !             for a non-deleted primary data record either when RMS is
1 !             positioning sequentially, or when RMS is positioning randomly
1 !             by an alternate key value. When I initially wrote this routine
1 !             I mistakenly wrote it for the sequential case only. Since in
1 !             this case RM$SEARCH_SIDR only has to search those SIDRs whose
1 !             keys exactly match the full size search key, this mean't that
```

```
!			positioning randomly by key was broken for all cases involving
!			generic search keys, greater-than searches, or greater-than or
!			equal searches when more than one SIDR had to be searched.
!			The fix for this set of problems was actually quite simple. If
!			the search characteristics are not setup such that RMS is
!			performing an exact match search, the routine RM$SEARCH_SIDR
!			saves the search key in keybuffer 5, and extracts the key of
!			the current SIDR into keybuffer 2. When this SIDR is exhausted
!			and a non-deleted primary data record has not been found, RMS
!			returns the search key saved in keybuffer 5 to keybuffer 2,
!			and determines whether the key of the next SIDR matches the
!			search key according to the characteristics of the search. If
!			so, this whole cycle repeats itself, otherwise, this routine
!			returns the appropriate error to its caller.
!
!			I also made a change to RM$SIDR_REPOS involving the case when
!			RMS is unable to position to a primary data record because it
!			has been deleted, and when it re-positions back to the SIDR
!			it finds that it too has been deleted. In such a case, because
!			RMS was unable to re-position to the SIDR, it did a
!			greater-than or equal search and had positioned exactly to the
!			SIDR that followed. This is exactly where RMS wants to continue
!			its search for a non-deleted primary data record provided the
!			key of this SIDR matches search key according to the search
!			characteristics. Therefore, all the routine RM$SIDR_REPOS has to
!			do is return a 0. The routine RM$SEARCH_SIDR, when it sees this
!			0, knows that the current SIDR has been exhaused (or in this
!			case deleted), and that it should go determine whether the
!			search can continue with the SIDR it now finds itself
!			positioned to.
!
!		V03-008	KBT0298		Keith B. Thompson	24-Aug-1982
!			Reorganize psects
!
!		V03-007	TMK0003		Todd M. Katz		10-Aug-1982
!			Change the linkage of RM$SIDR_REPOS. The address of the
!			beginning of the SIDR is now both in the input parameter
!			list and in the output parameter list. This is necessary
!			instead of just passing its address to this routine, an address
!			of a stack location, because this routine will allow RMS to
!			stall, and when RMS resumes after a stall, the stack addresses
!			are not necessairly the same as they were before the stall.
!
!		V03-006	TMK0002		Todd M. Katz		10-Aug-1982
!			When RMS positions to the first non-deleted primary data
!			record by alternate key, it first skips past all entries
!			within a SIDR that are marked deleted until either the
!			end of the SIDR is encountered, or a non-deleted entry is
!			found. Using this non-deleted entry, RMS attempts to position
!			to the primary data record. If RMS finds that the data record
!			is in fact deleted, it wants to mark the SIDR entry deleted,
!			and return to the loop that looks for a non-deleted SIDR entry.
!			There it starts its search for a non-deleted SIDR array element
!			with the current SIDR element which of course RMS has just
!			marked deleted. However, RMS can only mark such a SIDR entry
!			deleted if it has write access to the file and the file is not
!			being shared. Thus, the way this RMS currently works, if the
```

```
1 !                     file was being shared or had not been opened for write access
1 !                     and RMS has positioned to a non-deleted SIDR array element
1 !                     which pointed to a deleted primary data record, RMS would not
1 !                     mark the SIDR entry deleted, it would return to the loop which
1 !                     searches for the first non-deleted SIDR entry starting with the
1 !                     current entry, and it would position to the very same
1 !                     non-deleted entry (which points to a deleted primary data
1 !                     record). The result is an infinite loop! To fix this problem,
1 !                     RMS will now immediately position past the current element to
1 !                     the next one when it finds that the current non-deleted SIDR
1 !                     element points to a deleted primary data record, and it is
1 !                     unable to mark the SIDR element deleted at that point.
1 !
1 !             V03-005 TMK0001         Todd M. Katz            22-Jun-1982
1 !                     Revised entire module.
1 !
1 !*****
1
1 LIBRARY 'RMSLIB:RMS'
1
1 REQUIRE 'RMSSRC:RMSIDXDEF';
1
1 ! Define default PSECTS for code
1 !
1 PSECT
1     CODE = RM$RMS3(PSECT_ATTR),
1     PLIT = RM$RMS3(PSECT_ATTR);
1
1 ! Linkages
1 !
1 LINKAGE
1     L_PRESERVE1,
1     L_RABREG_567,
1     L_RABREG_67,
1     L_RABREG_7,
1     L_REC_OVHD,
1     L_SIDR_FIRST,
1     L_JSBOT,
1
1 ! Linkages for Local Routines
1 !
1     RL$FOLLOW_PTR = JSB ()
1                     : GLOBAL (COMMON_RABREG, R_REC_ADDR, R_IDX_DFN),
1     RL$POS_BY_COUNT = JSB (REGISTER = 1)
1                     : GLOBAL (COMMON_RABREG, R_REC_ADDR, R_IDX_DFN);
1
1 ! External Routines
1 !
1 EXTERNAL ROUTINE
1     RM$COMPARE_REC      : RL$RABREG_67,
1     RM$CSEARCH_TREE     : RL$RABREG_67,
1     RM$EXT_ARRY_RFA     : RL$RABREG_67,
1     RM$FIND_BY_RRV      : RL$RABREG_67,
1     RM$KEY_DESC         : RL$RABREG_7,
1     RM$GETNXT_ARRAY     : RL$RABREG_67,
1     RM$RECORD_KEY       : RL$PRESERVE1,
1     RM$REC_OVHD         : RL$REC_OVHD,
```

```
1      RM$RLSBKT           : RL$PRESERVE1,
1      RM$RU_RECLAIM       : RL$RABREG_67 ADDRESSING_MODE(LONG_RELATIVE),
1      RM$SIDR_END         : RL$RABREG_67,
1      RM$SIDR_FIRST       : RL$SIDR_FIRST,
1      RM$SQUISH_SIDR      : RL$RABREG_567,
1      RM$UNPACK_REC       : RL$JSB01;
1
1 ! Forward Routine
1 !
1 FORWARD ROUTINE
1          RM$POS_BY_COUNT : RL$POS_BY_COUNT;
```

```
%SBTTL 'RM$FOLLOW_PTR'
ROUTINE RM$FOLLOW_PTR (VBN, ID) : RL$FOLLOW_PTR =

!++
!
! FUNCTIONAL DESCRIPTION:
!
!       This routine recieves as input the RFA of an allegedly non-deleted
!       primary data record containing a secondary key of specific value. It
!       positions to that primary data record, and checks whether it is deleted
!       and if not, whether the secondary key whose specific value is stored
!       in keybuffer 2 has been deleted from the primary data record.
!
!       If RMS finds that the target primary data record is marked RU_DELETE
!       and the Recovery Unit in which the record was deleted is still active,
!       then RMS returns positioned to this record and lets a higher level
!       routine decide what to do. If the Recovery Unit in which the record was
!       deleted has successfully terminated, then RMS will return an error of
!       DEL after deleting this RU_DELETEd record (if it has write access to
!       the file), and releasing the primary data bucket.
!
!       If RMS encounters a record that is marked RU_UPDATE and is in a special
!       format then RMS will return positioned to this record after
!       reformatting it. The reformatting is done if RMS has write access to
!       the file, and the Recovery Unit in which it was updated has
!       successfully terminated.
!
! CALLING SEQUENCE:
!
!       BSBW RM$FOLLOW_PTR ()
!
! INPUT PARAMETERS:
!
!       VBN             - RFA VBN of the target primary data record
!       ID              - RFA ID of the target primary data record
!
! IMPLICIT INPUTS:
!
!       IDX_DFN                 - address of current index descriptor
!           IDX$B_KEYSZ         - size of secondary key
!           IDX$W_MINRECSZ      - minimum size of record to contain key
!
!       IFAB                    - address of IFAB
!           IFB$B_KBUFSZ        - size of each keybuffer
!           IFB$B_PLG_VER       - prologue version of file
!           IFB$V_RU            - if set, the file is Recovery Unit Journallable
!           IFB$V_WRTACC        - if set, file is open for write access
!
!       IRAB                    - address of IRAB
!           IRB$L_CURBDB        - address of SIDR bucket's BDB
!                                 (0 if re-positioning)
!           IRB$L_KEYBUF        - address of keybuffers
!           IRB$L_RECBUF        - address of internal record buffer
!
!       REC_ADDR                - address of SIDR array element
!                                 (invalid if re-positioning)
!
```

```
! OUTPUT PARAMETERS:
!       NONE
!
! IMPLICIT OUTPUTS:
!
!       IRAB
!           IRB$L_CURBDB        - address of primary data bucket's BDB (success)
!                                 or contents on input (failure)
!           IRB$L_NXTBDB        - contents on input (success)
!                                 or 0 (failure)
!           IRB$L_RECBUF        - unpacked primary data record
!                                 (success and prologue 3 file)
!           IRB$V_RU_DELETE     - if set, do not reclaim SIDR array space
!
!       REC_ADDR                - address of primary data bucket (success)
!                                 or contents on input (failure)
!
! ROUTINE VALUE:
!
!       SUC     - non-deleted primary data record successfully positioned to.
!       DEL     - primary data record deleted or specific alternate key deleted
!                 from primary data record.
!       RNF     - primary data record not found.
!       various I/O errors
!
! SIDE EFFECTS:
!
!       On success, REC_ADDR points to the primary data record (which will be
!            unpacked if the file's prologue version is 3), the primary data
!            bucket's BDB address is stored in IRB$L_CURBDB, and IRB$L_NXTBDB
!            contains whatever was in IRB$L_CURBDB on input.
!       On any and all failures, REC_ADDR points to whatever it had pointed to
!            on input, IRB$L_CURBDB contains whatever it contained on input,
!            and any buckets accessed have been released.
!       If the record is marked RU_DELETEd, it might have been deleted.
!       If the record is marked RU_UPDATE or RU_DELETEd and RMS is unable to
!            recover any space from it because the Recovery Unit in which the
!            record was modified has not completed, then RMS sets the state
!            bit IRB$V_RU_DELETE whenever it makes a decision to return an error
!            status of RMS$_DEL and the file has been opened for write access.
!            This will guarentee that the coresponding SIDR array element will
!            only be marked RU_DELETE, and no space will be reclaimed from it.
!       If the record is marked RU_UPDATEd, it might have been reformatted.
!
!--

    BEGIN

    BUILTIN
        AP;

    EXTERNAL REGISTER
        COMMON_RAB_STR,
        R_IDX_DFN_STR,
        R_REC_ADDR_STR;

    LOCAL
```

```
        SAVE_SDR_ADDR,
        STATUS;

    ! Save the current SIDR array's address, and the address of the SIDR
    ! bucket's BDB while performing the primary data record lookup.
    !
    SAVE_SDR_ADDR = .REC_ADDR;
    IRAB[IRB$L_NXTBDB] = .IRAB[IRB$L_CURBDB];

    ! If the file is write accessed and Recovery Unit Journallable, then make
    ! sure the primary data bucket containing the target record is exclusively
    ! accessed in case reclamation is required.
    !
    IF  .IFAB[IFB$V_WRTACC]
          AND
        .IFAB[IFB$V_RU]
    THEN
        IRAB[IRB$B_CACHEFLGS] = CSH$M_LOCK;

    ! Position to the primary data record pointing at by the SIDR array element
    ! RMS has positioned to.
    !
    STATUS = RM$FIND_BY_RRV (.VBN, .ID, 0);

    ! If RMS is successful at positioning to tthe target primary data record
    ! only to find that it has been deleted within a Recovery Unit, subject
    ! this record to further processing before deciding what to do with this
    ! record.
    !
    IF  .STATUS
          AND
        .REC_ADDR[IRC$V_RU_DELETE]
    THEN
        BEGIN

        LOCAL
            TEMP_STATUS;

        ! If the Recovery Unit in which the current record was deleted has
        ! not completed, or if the file has not been opened for write access
        ! then RMS will not be able to reclaim any space from the record. RMS
        ! may still want to consider this record as a non-deleted primary data
        ! record though.
        !
        IF  NOT (TEMP_STATUS = RM$RU_RECLAIM())
        THEN
            BEGIN

            ! If the current record was deleted within a currently active
            ! Recovery Unit and it was this stream that deleted it; or, if
            ! this stream is able to lock the record but does not have write
            ! access to the file, then RMS wants to treat this record as if it
            ! is actually deleted. To do so, RMS will return a status of
            ! RMS$_DEL so that the search will continue. However, RMS will also
            ! set the state bit IRB$V_RU_DELETE, if it has write access to the
            ! file, so that no space is reclaimed from the corresponding SIDR
            ! array element.
```

```
            !
            IF .TEMP_STATUS<0,16> NEQU RMSERR(RLK)
            THEN
                BEGIN
                STATUS = RMSERR(DEL);

                IF .IFAB[IFB$V_WRTACC]
                THEN
                    IRAB[IRB$V_RU_DELETE] = 1;
                END;
            END

        ! The Recovery Unit in which the current record was deleted has
        ! completed successfully and RMS was able to reclaim some space
        ! from the record. RMS changes the status to RMS$_DEL so that the
        ! search will be continued.
        !
        ELSE
            STATUS = RMSERR(DEL);

        ! If RMS has discovered that the Recovery Unit in which the target
        ! primary data record was deleted has terminated successfully and that
        ! the record is in fact deleted, then release the primary data bucket
        ! containing the record.
        !
        IF .STATUS<0,16> EQLU RMSERR(DEL)
        THEN
            BEGIN

            GLOBAL REGISTER
                R_BDB_STR;

            RELEASE (IRAB[IRB$L_CURBDB]);
            END;
        END;

    ! If RMS was successful at positioning to the target primary data record
    ! only to find that it had been updated but not deleted within a Recovery
    ! Unit, and if the file has been opened for write access, then RMS will
    ! try to re-format the record provided the Recovery Unit in which the record
    ! was modified has completed.
    !
    IF  .STATUS
         AND
        .IFAB[IFB$V_WRTACC]
         AND
        .REC_ADDR[IRC$V_RU_UPDATE]
         AND
        NOT .REC_ADDR[IRC$V_RU_DELETE]
    THEN
        RM$RU_RECLAIM();

    ! If RMS was successful at positioning to the primary data record pointed
    ! at by the SIDR array element, and the record is NOT marked deleted, then
    ! RMS must make sure that it is the right one - ie that the alternate key in
    ! this index matches the key of the SIDR which has been saved in
    ! keybuffer 2.
```

```
2        !
2        IF .STATUS
2        THEN
3            BEGIN
3
3            LOCAL
3                RECORD_OVHD,
3                RECORD_SIZE,
3                SAVE_UDR_ADDR : REF BBLOCK;
3
3            SAVE_UDR_ADDR = .REC_ADDR;
3
3            ! Determine the size of the primary data record's overhead, and the
3            ! size of the primary data record itself. This is necessary to
3            ! determine whether or not the alternate key has been deleted through
3            ! truncation of part of the record.
3            !
4            BEGIN
4
4            LOCAL
4                REC_SIZE,
4                SAVE_IDX_DFN : REF BBLOCK;
4
4            SAVE_IDX_DFN = .IDX_DFN;
4            RM$KEY_DESC (0);
4
4            RECORD_OVHD = RM$REC_OVHD(0; REC_SIZE);
4
4            ! If the target primary data record was updated within a Recovery
4            ! Unit then the true size of the primary data record maybe retrieved
4            ! from the last two bytes in the reserved space of the record.
4            !
4            IF .REC_ADDR[IRC$V_RU_UPDATE]
4            THEN
5                RECORD_SIZE = .(.REC_ADDR + .RECORD_OVHD
5                                          + .REC_SIZE
4                                          - IRC$C_DATSZFLD)<0,16>
4            ELSE
4                RECORD_SIZE = .REC_SIZE;
4
4            IDX_DFN = .SAVE_IDX_DFN;
3            END;
3
3            ! If this is a prologue 3 file then the record must be unpacked in
3            ! order to obta'n the full size of the record and to be able to compare
3            ! the SIDR key with the alternate key in the data record itself. RMS
3            ! also sets REC_ADDR to point to the unpacked record, so that the
3            ! alternate key may easily be extracted from the record for key
3            ! comparison.
3            !
3            ! NOTE that because RMS always unpacks prologue 3 primary data records
3            ! when positioning by a secondary key for key comparison, it is never
3            ! necessary to unpack the record a second time during a $GET/$FIND.
3            !
3            IF .IFAB[IFB$B_PLG_VER] GEQU PLG$C_VER_3
3            THEN
4                BEGIN
```

```
4
4              BUILTIN
4                  AP;
4
4              GLOBAL REGISTER
4                  R_BKT_ADDR;
4
4              REC_ADDR = .REC_ADDR + .RECORD_OVHD;
4
4              AP = 0;
4              RECORD_SIZE = RM$UNPACK_REC (.IRAB[IRB$L_RECBUF], .RECORD_SIZE);
4
4              REC_ADDR = .IRAB[IRB$L_RECBUF];
3              END;
3
3          ! If the primary data record is of sufficient size to contain the
3          ! alternate key (ie - the alternate key has not been deleted through
3          ! truncation of the record during the process of updating the record
3          ! by some other process), then RMS determines whether or not the
3          ! alternate key in the record still matches the SIDR it has positioned
3          ! to. If this is the case then RMS has found the next non-deleted
3          ! primary data record, and returns success.
3          !
4          IF (.RECORD_SIZE GEQU .IDX_DFN[IDX$W_MINRECSZ])
3              AND
4            (NOT RM$COMPARE_REC (KEYBUF_ADDR(2), .IDX_DFN[IDX$B_ EYSZ], -1))
3          THEN
4              BEGIN
4              REC_ADDR = .SAVE_UDR_ADDR;
4              RETURN RMSSUC();
4              END
4
4          ! If the secondary key has been deleted totally from the record through
4          ! truncation of the record during an $UPDATE by another process, or
4          ! if during the $UPDATE the secondary key changed, and the secondary
4          ! key in the primary data record no longer matches the key of the SIDR
4          ! RMS is positioned to, then this is not the next non-deleted primary
4          ! data record. In this case, RMS releases the primary data bucket, and
4          ! sets up to return an error status of record deleted, so that the
4          ! search for the next non-deleted primary data record can continue.
4          !
3          ELSE
4              BEGIN
4
4              GLOBAL REGISTER
4                  R_BDB_STR;
4
4              ! If the target primary data record has been modified within a
4              ! Recovery Unit, and the stream has write access to the file so
4              ! that some attempt will be made to reclaim the space occupied by
4              ! SIDR array element, then set up so that when the SIDR array
4              ! element pointing to this primary data record is deleted, it is
4              ! only marked RU_DELETE, and no space is reclaimed.
4              !
4              IF  .IFAB[IFB$V_WRTACC]
4                    AND
5                  (.SAVE_UDR_ADDR[IRC$V_RU_DELETE]
```

```
5                          OR
5                          .SAVE_UDR_ADDR[IRC$V_RU_UPDATE])
4              THEN
4                  IRAB[IRB$V_RU_DELETE] = 1;
4
4              RELEASE(IRAB[IRB$L_CURBDB]);
4              STATUS = RMSERR(DEL);
3              END;
3
2          END;
2
2      ! If RMS was unable to position to the target primary data record, or if
2      ! after positioning to the record it found that the record itself, or the
2      ! secondary key it expected to find there had been deleted, then RMS
2      ! restores the address of the SIDR bucket's BDB to IRB$L_CURBDB, and the
2      ! address of the SIDR array element it currently has positioned to
2      ! to REC_ADDR, and it returns the appropriate status.
2      !
2      IRAB[IRB$L_CURBDB] = .IRAB[IRB$L_NXTBDB];
2      IRAB[IRB$L_NXTBDB] = 0;
2      REC_ADDR = .SAVE_SDR_ADDR;
2
2      RETURN .STATUS;
2
1      END;                                        ! { of routine }


                                                    .TITLE  RM3SSIDR
                                                    .IDENT  \V04-000\

                                                    .EXTRN  RM$COMPARE_REC, RM$CSEARCH_TREE
                                                    .EXTRN  RM$EXT_ARRY_RFA
                                                    .EXTRN  RM$FIND_BY_RRV, RM$KEY_DESC
                                                    .EXTRN  RM$GETNXT_ARRAY
                                                    .EXTRN  RM$RECORD_KEY, RM$REC_OVHD
                                                    .EXTRN  RM$RLSBKT, RM$RU_RECLAIM
                                                    .EXTRN  RM$SIDR_END, RM$SIDR_FIRST
                                                    .EXTRN  RM$SQUISH_SIDR, RM$UNPACK_REC

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

                                 3C  BB 00000 RM$FOLLOW_PTR:
                                              PUSHR   #^M<R2,R3,R4,R5>              ; 0419
                                 56  DD 00002 PUSHL   REC_ADDR                      ; 0538
             3C     A9     20    A9  D0 00004 MOVL    32(IRAB), 60(IRAB)            ; 0539
                    0A     06    AA  E9 00009 BLBC    6(IFAB), 1$                   ; 0545
    04     00A0     CA           01  E1 0000D BBC     #1, 160(IFAB), 1$             ; 0547
             40     A9           01  90 00013 MOVB    #1, 64(IRAB)                  ; 0549
                                 7E  D4 00017 1$:  CLRL    -(SP)                    ; 0554
                           20    AE  DD 00019 PUSHL   ID                            ;
                           20    AE  DD 0001C PUSHL   VBN                           ;
                              0000G  30 0001F BSBW    RM$FIND_BY_RRV                ;
                    5E           0C  C0 00022 ADDL2   #12, SP                       ;
                    53           50  D0 00025 MOVL    R0, STATUS                    ;
                    53           53  E9 00028 BLBC    STATUS, 5$                    ; 0561
    3A              66           05  E1 0002B BBC     #5, (REC_ADDR), 4$            ; 0563
                  00000000G      EF  16 0002F JSB     RM$RU_RECLAIM                 ; 0576
```

```
                      16         50 E8 00035         BLBS    TEMP_STATUS, 2$                          ;
           82AA       8F         50 B1 00038         CMPW    TEMP_STATUS, #33450                      ; 0590
                      14         13 0003D            BEQL    3$                                       ;
                      53   8262  8F 3C 0003F         MOVZWL  #33378, STATUS                           ; 0593
                      0B     06  AA E9 00044         BLBC    6(IFAB), 3$                              ; 0595
             07       A9         20 88 00048         BISB2   #32, 7(IRAB)                             ; 0597
                                 05 11 0004C         BRB     3$                                       ; 0576
                      53   8262  8F 3C 0004E 2$:     MOVZWL  #33378, STATUS                           ; 0607
           8262       8F         53 B1 00053 3$:     CMPW    STATUS, #33378                           ; 0614
                                 0F 12 00058         BNEQ    4$                                       ;
                      54     20  A9 D0 0005A         MOVL    32(IRAB), BDB                            ; 0621
                             20  A9 D4 0005E         CLRL    32(IRAB)                                 ;
                                 7E D4 00061         CLRL    -(SP)                                    ;
                              0000G 30 00063         BSBW    RM$RLSBKT                                ;
                      5E         04 C0 00066         ADDL2   #4, SP                                   ;
                      12         53 E9 00069 4$:     BLBC    STATUS, 5$                               ; 0631
                      0E     06  AA E9 0006C         BLBC    6(IFAB), 5$                              ; 0633
        0A            66         06 E1 00070         BBC     #6, (REC_ADDR), 5$                       ; 0635
        06            66         05 E0 00074         BBS     #5, (REC_ADDR), 5$                       ; 0637
                     00000000G   EF 16 00078         JSB     RM$RU_RECLAIM                            ; 0639
                      03         53 E8 0007E 5$:     BLBS    STATUS, 6$                               ; 0647
                               0094 31 00081         BRW     13$                                      ;
                      52         56 D0 00084 6$:     MOVL    REC_ADDR, SAVE_UDR_ADDR                  ; 0656
                      55         57 D0 00087         MOVL    IDX_DFN, SAVE_IDX_DFN                    ; 0669
                                 7E D4 0008A         CLRL    -(SP)                                    ; 0670
                              0000G 30 0008C         BSBW    RM$KEY_DESC                              ;
                      5E         04 C0 0008F         ADDL2   #4, SP                                   ;
                                 51 D4 00092         CLRL    R1                                       ; 0672
                              0000G 30 00094         BSBW    RM$REC_OVHD                              ;
        0D            66         06 E1 00097         BBC     #6, (REC_ADDR), 7$                       ; 0678
        54            56         50 C1 0009B         ADDL3   RECORD_OVHD, REC_ADDR, R4                ; 0680
                           FE A144 9F 0009F          PUSHAB  -2(REC_SIZE)[R4]                         ;
                      54         9E 3C 000A3         MOVZWL  @(SP)+, RECORD_SIZE                      ;
                                 03 11 000A6         BRB     8$                                       ;
                      54         51 D0 000A8 7$:     MOVL    REC_SIZE, RECORD_SIZE                    ; 0684
                      57         55 D0 000AB 8$:     MOVL    SAVE_IDX_DFN, IDX_DFN                    ; 0686
                      03   00B7  CA 91 000AE         CMPB    183(IFAB), #3                            ; 0700
                                 16 1F 000B3         BLSSU   9$                                       ;
                      56         50 C0 000B5         ADDL2   RECORD_OVHD, REC_ADDR                    ; 0710
                                 5C D4 000B8         CLRL    AP                                       ; 0712
                      51         54 D0 000BA         MOVL    RECORD_SIZE, R1                          ; 0713
                      50     68  A9 D0 000BD         MOVL    104(IRAB), R0                            ;
                              0000G 30 000C1         BSBW    RM$UNPACK_REC                            ;
                      54         50 D0 000C4         MOVL    R0, RECORD_SIZE                          ;
                      56     68  A9 D0 000C7         MOVL    104(IRAB), REC_ADDR                      ; 0715
  54      22   A7     10         00 ED 000CB 9$:     CMPZV   #0, #16, 34(IDX_DFN), RECORD_SIZE        ; 0726
                                 21 1A 000D1         BGTRU   10$                                      ;
                      7E         01 CE 000D3         MNEGL   #1, -(SP)                                ; 0728
                      7E     20  A7 9A 000D6         MOVZBL  32(IDX_DFN), -(SP)                       ;
                      50   00B4  CA 3C 000DA         MOVZWL  180(IFAB), R0                            ;
                           60 B940 9F 000DF          PUSHAB  @96(IRAB)[R0]                            ;
                              0000G 30 000E3         BSBW    RM$COMPARE_REC                           ;
                      5E         0C C0 000E6         ADDL2   #12, SP                                  ;
                      08         50 E8 000E9         BLBS    R0, 10$                                  ;
                      56         52 D0 000EC         MOVL    SAVE_UDR_ADDR, REC_ADDR                  ; 0731
                      50         01 D0 000EF         MOVL    #1, R0                                   ; 0732
                                 32 11 000F2         BRB     14$                                      ;
```

```
                   0C      06   AA  E9 000F4 10$:   BLBC    6(IFAB), 12$                            ; 0757
          04       62           05  E0 000F8        BBS     #5, (SAVE_UDR_ADDR), 11$                ; 0759
          04       62           06  E1 000FC        BBC     #6, (SAVE_UDR_ADDR), 12$                ; 0761
              07   A9           20  88 00100 11$:   BISB2   #32, 7(IRAB)                            ; 0763
                   54      20   A9  D0 00104 12$:   MOVL    32(IRAB), BDB                           ; 0765
                           20   A9  D4 00108        CLRL    32(IRAB)                                ;
                                7E  D4 0010B        CLRL    -(SP)                                   ;
                             0000G  30 0010D        BSBW    RM$RLSBKT                               ;
                   5E           04  C0 00110        ADDL2   #4, SP                                  ;
                   53    8262   8F  3C 00113        MOVZWL  #33378, STATUS                          ; 0766
              20   A9      3C   A9  D0 00118 13$:   MOVL    60(IRAB), 32(IRAB)                      ; 0778
                           3C   A9  D4 0011D        CLRL    60(IRAB)                                ; 0779
                   56           6E  D0 00120        MOVL    SAVE_SDR_ADDR, REC_ADDR                 ; 0780
                   50           53  D0 00123        MOVL    STATUS, R0                              ; 0782
                   5E           04  C0 00126 14$:   ADDL2   #4, SP                                  ; 0784
                                3C  BA 00129        POPR    #^M<R2,R3,R4,R5>                        ;
                                    05 0012B        RSB                                             ;

; Routine Size:  300 bytes,    Routine Base:  RM$RMS3 + 0000
```

```
%SBTTL 'RM$POS_BY_COUNT'
ROUTINE RM$POS_BY_COUNT (SKIP_NUMBER) : RL$POS_BY_COUNT =
!++
!
! FUNCTIONAL DESCRIPTION:
!
!       This routine's responsibility is to position to a particular SIDR
!       array element. It does this by skipping SKIP_NUMBER array elements.
!
! CALLING SEQUENCE:
!
!       BSBW RM$POS_BY_COUNT()
!
! INPUT PARAMETERS:
!
!       SKIP_NUMBER     - number of array elements to skip
!
! IMPLICIT INPUTS:
!
!       REC_ADDR                - address of SIDR
!
! OUTPUT PARAMETERS:
!       NONE
!
! IMPLICIT OUTPUTS:
!
!       REC_ADDR                - address of desired array element
!
! ROUTINE VALUE:
!
!       SUC     - positioned to desired array element.
!       BUG     - the number of array elements preceeding the desired
!                 element exceeds the number of elements in the entire array.
!
! SIDE EFFECTS:
!
!       on success, REC_ADDR points to the desired array element.
!--

    BEGIN

    EXTERNAL REGISTER
        COMMON_RAB_STR,
        R_IDX_DFN_STR,
        R_REC_ADDR_STR;

    LOCAL
        END_OF_SIDR;

    ! Find the end of the current SIDR.
    !
    END_OF_SIDR = RM$SIDR_END();

    ! Position to the SIDR's first array element
    !
    REC_ADDR = RM$SIDR_FIRST(0);

```

```
;  780        0842  2      ! Skip the desired number of array elements, or until the end of the SIDR
;  781        0843  2      !  s encountered - which ever comes first.
;  782        0844  2      !
;  783        0845  2      DECRU LOOP_INDEX FROM .SKIP_NUMBER TO 1 BY 1
;  784        0846  2      DO
;  785        0847  2
;  786        0848  2          ! If the end of the SIDR is encountered before the desired array
;  787        0849  2          ! element is encountered then this represents an invalid internal
;  788        0850  2          ! condition, and report it as such.
;  789        0851  2          !
;  790        0852  2          IF .REC_ADDR GEQA .END_OF_SIDR
;  791        0853  2          THEN
;  792        0854  3              RETURN RMSERR(BUG)
;  793        0855  3
;  794        0856  3          ! Position past the current array element to the next array element.
;  795        0857  3          !
;  796        0858  2          ELSE
;  797        0859  2              RM$GETNXT_ARRAY();
;  798        0860  2
;  799        0861  2      ! Having positioned to the desired SIDR arrary element, return success.
;  800        0862  2      !
;  801        0863  2      RETURN RMSSUC();
;  802        0864  1      END;                                    ! { of routine }


                                 1C  BB 00000 RM$POS_BY_COUNT:
                                                     POSHR   #^M<R2,R3,R4>                       ; 0786
                        53       51  D0 00002        MOVL    R1, R3                              ;
                              0000G  30 00005        BSBW    RM$SIDR_END                         ; 0836
                        54       50  D0 00008        MOVL    R0, END_OF_SIDR                     ;
                                 7E  D4 0000B        CLRL    -(SP)                               ; 0840
                              0000G  30 0000D        BSBW    RM$SIDR_FIRST                       ;
                        5E       04  C0 00010        ADDL2   #4, SP                              ;
                        56       50  D0 00013        MOVL    R0, REC_ADDR                        ;
                        52       53  D0 00016        MOVL    SKIP_NUMBER, LOOP_INDEX             ; 0852
                        11       11  00019           BRB     3$                                  ;
                        54       56  D1 0001B 1$:    CMPL    REC_ADDR, END_OF_SIDR               ;
                        07       1F  0001E           BLSSU   2$                                  ;
                        50  8434 8F  3C 00020        MOVZWL  #33844, R0                          ; 0854
                                 0A  11 00025        BRB     4$                                  ;
                              0000G  30 00027 2$:    BSBW    RM$GETNXT_ARRAY                     ; 0859
                                 52  D7 0002A        DECL    LOOP_INDEX                          ; 0852
                                 ED  12 0002C 3$:    BNEQ    1$                                  ;
                        50       01  D0 0002E        MOVL    #1, R0                              ; 0863
                                 1C  BA 00031 4$:    POPR    #^M<R2,R3,R4>                       ; 0864
                                     05 00033        RSB                                         ;

; Routine Size:  52 bytes,    Routine Base:  RM$RMS3 + 012C
```

```
1 %SBTTL 'RMS$SEARCH_SIDR'
1 GLOBAL ROUTINE RMS$SEARCH_SIDR : RL$RABREG_67 =
1
1 !++
1 !
1 ! FUNCTIONAL DESCRIPTION:
1 !
1 !       The purpose of this routine is to search for a non-deleted primary
1 !       data record by means of an alternate key of reference. This routine
1 !       starts its search with an array element in the current SIDR array. If
1 !       RMS is positioning sequentially, then the array element RMS starts
1 !       its search with will always be one more that the value in
1 !       IRB$W_SAVE_POS. Thus, RMS may start its search with any of the elements
1 !       in the current SIDR array depending upon the value in this variable.
1 !       However, if RMS is positioning randomly by key value, then RMS will
1 !       always start its search for a non-deleted primary data record with the
1 !       very first element in the current SIDR.
1 !
1 !       When the current SIDR array is exhausted, RMS will attempt to continue
1 !       its search within the very next SIDR array that follows. RMS may
1 !       continue its search provided the key of this next SIDR matches the
1 !       characteristics of the search. These search characteristics include the
1 !       search key itself (in keybuffer 2), the size of the search key (in
1 !       IRB$B_KEYSZ) which may not be the same as the size of the alternate key
1 !       itself, and whether the search is for an equal match, a greater-than
1 !       or equal match (IRB$V_SRCHGE is set), or a greater-than match
1 !       (IRB$V_SRCHGT is set). If RMS is positioning randomly then any possible
1 !       combination of these search characteristics is possible, but when RMS is
1 !       positioning sequentially, RMS will only search those SIDRs whose key
1 !       exactly matches the full size search key in keybuffer 2.
1 !
1 !       If during its search for a non-deleted primary data record RMS
1 !       encounters a non-deleted SIDR array element pointing to a deleted
1 !       primary data record, then RMS will mark the SIDR array element deleted
1 !       and reclaim as much space from the SIDR as it can according to the
1 !       normal rules of SIDR deletion. However, such activity will only take
1 !       place if the stream has write access to the file.
1 !
1 !       This routine concludes its search when either an error occurs, it is
1 !       successful at positioning to a non-deleted primary data record, or it
1 !       exhausts all SIDRs matching the search characteristics without
1 !       positioning to such a primary data record.
1 !
1 ! CALLING SEQUENCE:
1 !
1 !       BSBW RMS$SEARCH_SIDR()
1 !
1 ! INPUT PARAMETERS:
1 !       NONE
1 !
1 ! IMPLICIT INPUTS:
1 !
1 !       IDX_DFN                 - address of index descriptor describing key
1 !           IDX$V_KEY_COMPR     - if set, SIDR key compression is enabled
1 !           IDX$B_KEYSZ         - size of key
1 !
1 !       IFAB                    - address of IFAB
```

```
1 !                 IFB$L_SFSB_PTR         - pointer to f    ared file sync. block
1 !                 IFB$W_KBUFSZ           - size of a key.  fer
1 !                 IFB$V_WRTACC           - if set, file is write accessed
1 !
1 !             IRAB                       - address of IRAB
1 !                 IRB$L_CURBDB           - address of SIDR bucket's BDB
1 !                 IRB$L_KEYBUF           - pointer to the contigious keybuffers
1 !                 IRB$B_KEYSZ            - size of search key
1 !                 IRB$L_LST_NCMP         - address of last zero front compressed key
1 !                 IRB$V_PRM              - if set, permanence should be set in SIDR BDB
1 !                 IRB$W_SAVE_POS         - number of elements preceeding starting point
1 !                 IRB$V_SRCHGE           - if set, search for GE match
1 !                 IRB$V_SRCHGT           - if set, search for GT match
1 !
1 !             REC_ADDR                   - address of SIDR within bucket
1 !
1 ! OUTPUT PARAMETERS:
1 !             NONE
1 !
1 ! IMPLICIT OUTPUTS:
1 !
1 !             IRAB                       - address of IRAB
1 !                 IRB$L_CURBDB           - address of primary data bucket's BDB
1 !                 IRB$W_FIRST_ID         - RFA ID of new current SIDR's first element
1 !                 IRB$L_FIRST_VBN        - RFA VBN of new current SIDR's first element
1 !                 IRB$L_LST_NCMP         - address of last zero front compressed key
1 !                 IRB$L_RFA_VBN          - VBN of new current SIDR's bucket
1 !                 IRB$W_SAVE_POS         - number of elements before new current element
1 !
1 !             REC_ADDR        - address of primary data record
1 !
1 ! ROUTINE VALUE:
1 !
1 !             BUG     - invalid internal condition
1 !             SUC     - next non-deleted primary data record found
1 !             RNF     - all SIDRs of this key value exhausted
1 !             various I/O errors
1 !
1 ! SIDE EFFECTS:
1 !
1 !             AP is trashed.
1 !             Keybuffer 5 will contain the search key, if this routine was called
1 !                     with a generic search key (search key size less than key size),
1 !                     or to find the first SIDR with a key greater-than or
1 !                     greater-than-or-equal to the search key in keybuffer 2.
1 !             On success, IRB$L_CURBDB contains the primary data bucket's BDB
1 !                     address, the SIDR bucket will have been released, REC_ADDR
1 !                     points to the primary data bucket, and the IRAB contains
1 !                     sufficient information necessary to uniquely identify and
1 !                     re-position to the SIDR array element pointing to the
1 !                     non-deleted primary data record. In addition, keybuffer 2 will
1 !                     contain the key of the SIDR.
1 !             On all errors except for a RNF found error, the SIDR bucket will have
1 !                     been released and possibly marked dirty/permanent.
1 !             SIDR array elements and in fact whole SIDRs might have been deleted
1 !                     if the stream positioning has write access to the file.
1 !
```

```
1 !--
1
2        BEGIN
2
2        BUILTIN
2            AP,
2            TESTBITSC;
2
2        EXTERNAL REGISTER
2            COMMON_RAB_STR,
2            R_IDX_DFN_STR,
2            R_REC_ADDR_STR;
2
2        LABEL
2            NEXT;
2
2        LOCAL
2            BEG_OF_SIDR,
2            END_OF_SIDR,
2            FLAGS              : BLOCK [1],
2            STATUS;
2
2        MACRO
2            FIRST_TIME         = 0,0,1,0 %,
2            NEW_KEY            = 0,1,1,0 %;
2
2        FLAGS[FIRST_TIME] = 1;
2
2        ! Save the address of the SIDR, and the address of the first byte past the
2        ! last SIDR array element (ie the end of the SIDR).
2        !
2        BEG_OF_SIDR = .REC_ADDR;
2        END_OF_SIDR = RM$SIDR_END();
2
2        ! RMS must perform an initial positioning before it begins its search for
2        ! the next non-deleted primary data record. This positioning depends upon
2        ! whether RMS is positioning randomly by key value, or is positioning
2        ! sequentially. In the latter case, where RMS initially positions to will
2        ! depend upon whether RMS had been able to position to the current SIDR
2        ! before calling this routine. Thus, this routine is faced with four
2        ! different ways in which it could perform this initial positioning
2        ! depending upon the mode of access, and if the access mode is sequential,
2        ! whether or not RMS had positioned to the current SIDR before this
2        ! routine was called.
2        !
2        ! 1. If RMS is positioning randomly by key value then RMS has positioned to
2        !    the first SIDR whose key matches the search key according to the
2        !    search characteristics, and RMS wants to begin its search for a
2        !    non-deleted primary data record with the very first element in the
2        !    SIDR array.
2        !
2        ! 2. This is the first positioning attempt after a $REWIND or $CONNECT and
2        !    as it is sequential there is no current SIDR. In this case RMS has
2        !    positioned to the very first SIDR it could find, and begins its search
2        !    for the next non-deleted primary data record with the very first
2        !    element in the SIDR array.
2        !
```

```
        ! 3. RMS has successfully positioned to the current SIDR. In this case RMS
        !    begins its search for the next non-deleted primary data record with
        !    the first element that follows the current SIDR element. Note that
        !    there may not be a following element in which case RMS exhausts the
        !    current SIDR very quickly.
        !
        ! 4. RMS was unable to position to the current SIDR before calling this
        !    routine. This could only be because the entire current SIDR was
        !    deleted. In this case RMS has positioned to the first SIDR with a
        !    key value greater than or equal to the key of the current SIDR, and
        !    it begins its search for the next non-deleted primary data record with
        !    the first array element in this SIDR.
        !
        RM$POS_BY_COUNT (.IRAB[IRB$W_SAVE_POS]);

        ! If RMS is positioning randomly by key value and either the size of the
        ! search key is less than the true key size or the search is for the first
        ! record with an alternate key either greater-than-or-equal or greater-than
        ! that of the search key, then save the search key presently located in
        ! keybuffer 2 in keybuffer 5. It is necessary to save the search key because
        ! if the SIDR positioned to is exhausted without locating a non-deleted
        ! primary data record, it will be necessary to continue the search with the
        ! next SIDR matching the search characteristics, and the search key will be
        ! required in order to continue this search. In addition, it will be
        ! necessary each time RMS positions to a new SIDR to extract its key into
        ! keybuffer 2, since the key of the SIDR RMS has currently positioned to
        ! might not be the same as the keys of SIDRs RMS has previously positioned
        ! to during the course of its search for a non-deleted primary data record.
        !
        IF   .IRAB[IRB$V_SRCHGE]
              OR
             .IRAB[IRB$V_SRCHGT]
              OR
             .IRAB[IRB$B_KEYSZ] LSSU .IDX_DFN[IDX$B_KEYSZ]
        THEN
            BEGIN
            FLAGS[NEW_KEY] = 1;
            CH$MOVE (.IRAB[IRB$B_KEYSZ], KEYBUF_ADDR(2), KEYBUF_ADDR(5));
            END

        ! Regardless of how many SIDRs RMS will position to in its quest for a
        ! non-deleted primary data record, their key value will always be the
        ! same as the key in keybuffer 2 (once keybuffer 2 has a key if it doesn't
        ! already), and there will not be a need to extract the key out of each
        ! SIDR positioned to.
        !
        ELSE
            FLAGS[NEW_KEY] = 0;

        ! Continue until either the next non-deleleted primary data record is
        ! encountered, the end-of-file is reached, or some error is encountered.
        !
    NEXT :
        BEGIN

        WHILE 1
        DO
```

```
        BEGIN

        ! If RMS is currently positioned to the very first element of a
        ! SIDR array, and either this is first such array being scanned
        ! during this current positioning or the search characteristics
        ! indicate that the key of each SIDR positioned to must be
        ! extracted, then extract the key of the SIDR into keybuffer 2.
        !
        IF  (TESTBITSC(FLAGS[FIRST_TIME])
                OR
               .FLAGS[NEW_KEY])
             AND
            (.IRAB[IRB$W_SAVE_POS] EQLU 0)

        THEN
            BEGIN

            LOCAL
                TMP_REC_ADDR;

            TMP_REC_ADDR = .REC_ADDR;
            REC_ADDR = .BEG_OF_SIDR;

            REC_ADDR = .REC_ADDR + RM$REC_OVHD(-1);

            IF .IDX_DFN[IDX$V_KEY_COMPR]
            THEN
                BEGIN

                GLOBAL REGISTER
                    R_BDB;

                AP = 1;
                RM$RECORD_KEY (KEYBUF_ADDR(2));
                END
            ELSE
                CH$MOVE (.IDX_DFN[IDX$B_KEYSZ], .REC_ADDR, KEYBUF_ADDR(2));

            REC_ADDR = .TMP_REC_ADDR;
            END;

        ! Continue searching for the next non-deleted primary data record
        ! until either it is found, an error occurs, the end of the SIDR
        ! is encountered, or no SIDR with a key matching the search key
        ! according to the search characteristics remains to be scanned for an
        ! array element pointing to a non-deleted primary data record.
        !
        WHILE .REC_ADDR LSSA .END_OF_SIDR
        DO
            BEGIN

            ! Starting with the current SIDR array element, search for the
            ! first element in the SIDR array that is not marked deleted
            ! until either one is found or the end of the SIDR is encountered.
            !
            WHILE ((.REC_ADDR LSSA .END_OF_SIDR)
                    AND
```

```
                .REC_ADDR[IRC$V_DELETED])
        DO
            BEGIN

            RM$GETNXT_ARRAY();
            IRAB[IRB$W_SAVE_POS] = .IRAB[IRB$W_SAVE_POS] + 1;
            END;

        ! If RMS has not runoff the end of the current SIDR array it is
        ! searching, but has successfully positioned to an element in the
        ! SIDR array that is not marked deleted, then attempt to access the
        ! primary data record that it points to, and determine whether or
        ! not it is marked deleted.
        !
        IF .REC_ADDR LSSA .END_OF_SIDR
        THEN
            BEGIN

            LOCAL
                RFA_ID,
                RFA_VBN;

            ! Attempt to access the primary data record while holding on to
            ! the SIDR bucket lock; waiting for the lock on the primary data
            ! bucket to be released if someone else has it while holding
            ! onto the SIDR bucket lock if necessary.
            !
            RM$EXT_ARRY_RFA (RFA_VBN, RFA_ID);

            STATUS = RM$FOLLOW_PTR (.RFA_VBN, .RFA_ID);

            ! If RMS has successfully found the next non-deleted primary
            ! data record, or some serious error has been encountered,
            ! then terminate the search for the next non-deleted primary
            ! data record. If the error encountered was just that the
            ! primary data record accessed had been deleted, then RMS will
            ! be able to continue the search for the next non-deleted
            ! primary data record with the next element in the SIDR array
            ! currently being searched which is not marked deleted.
            !
            IF .STATUS<0,16> NEQU RMSERR(DEL)
            THEN
                LEAVE NEXT;

            ! If the primary data record positioned to is deleted, but the
            ! SIDR array element is not marked as such, then alleviate this
            ! discrepency provided the user has accessed the file for
            ! writing, before continuing on with the search for the next
            ! non-deleted primary data record. If the current SIDR array
            ! element is marked RU_DELETE, and RMS is to RU_DELETE it, then
            ! in this case there is no need to do anything.
            !
            IF .IFAB[IFB$V_WRTACC]
                AND
               NOT  (.REC_ADDR[IRC$V_RU_DELETE]
                    AND
                    .IRAB[IRB$V_RU_DELETE])
```

```
        THEN
            BEGIN

            GLOBAL REGISTER
                COMMON_IO_STR;

            LOCAL
                SAVE_REC_ADDR;

            ! Retrieve the address of the SIDR bucket and mark it dirty.
            !
            BDB = .IRAB[IRB$L_CURBDB];
            BDB[BDB$V_DRT] = T;
            BKT_ADDR = .BDB[BDB$L_ADDR];

            ! Save the address of the current SIDR array element before
            ! attempting to delete any space.
            !
            SAVE_REC_ADDR = .REC_ADDR;
            REC_ADDR[IRC$V_RU_DELETE] = 0;

            ! If RMS is able to only delete the space occupied by the
            ! current SIDR array element's RRV pointer, then RMS may
            ! continue its search for a non-deleted primary data record
            ! within the current SIDR array after re-determining the
            ! address of the end of the current SIDR array.
            !
            IF RM$SQUISH_SIDR (1, .BEG_OF_SIDR)
            THEN
                IF .REC_ADDR EQLU .SAVE_REC_ADDR
                THEN
                    BEGIN
                    REC_ADDR    = .BEG_OF_SIDR;
                    END_OF_SIDR = RM$SIDR_END();
                    REC_ADDR    = .SAVE_REC_ADDR;
                    END

                ! If RMS is able to delete the entire current SIDR
                ! because all the elements contained within it have
                ! been deleted, then leave the loop responsible for
                ! searching the current SIDR so that conditions might
                ! be set up to attempt to search the next SIDR array
                ! in the alternate index.
                !
                ELSE
                    EXITLOOP

            ! If RMS is able to only mark the current SIDR array as
            ! deleted without recovering any space then make sure the
            ! state bit IRB$V_RU_DELETE is clear (if it was set) and
            ! continue the search for a non-deleted primary data record
            ! with the next SIDR array element.
            !
            ELSE
                BEGIN
                IRAB[IRB$V_RU_DELETE] = 0;
                RM$GETNXT_ARRAY();
```

```
8                       IRAB[IRB$W_SAVE_POS] = .IRAB[IRB$W_SAVE_POS] + 1;
7                       END;
7                   END
7
7               ! If the file has not been opened for write access or there are
7               ! really no changes that need to be made then the SIDR element
7               ! can not be marked de eted, and RMS must immediately position
7               ! to the next SIDR array element. This is because having
7               ! unexpectantly encountered a deleted primary data record RMS
7               ! will want to continue its search for the next non-deleted
7               ! SIDR element. Since RMS always starts such a search with the
7               ! current element, and since RMS was unabie to mark the current
7               ! SIDR array element deleted, RMS would end up positioning to
7               ! the current element as if it was the next element.
7               ! Immediately positioning to the next element prevents this.
7               !
6               ELSE
7                   BEGIN
7                   RM$GETNXT_ARRAY();
7                   IRAB[IRB$W_SAVE_POS] = .IRAB[IRB$W_SAVE_POS] + 1;
7                   IRAB[IRB$V_RU_DELETE] = 0;
6                   END;
5               END;
5
4           END;
4
4       ! Having exhausted this current SIDR array in the search for the next
4       ! non-deleted primary data record, RMS positions to the next SIDR
4       ! in order to continue the search. NOTE that if the search
4       ! characteristics require the original search key, it is restored to
4       ! keybuffer 2 from keybuffer 5 before initiating the search. This
4       ! will always be the case when the key of the next SIDR RMS positions
4       ! to maybe different from the key of the current SIDR.
4       !
4       IF .FLAGS[NEW_KEY]
4       THEN
4           CH$MOVE (.IRAB[IRB$B_KEYSZ], KEYBUF_ADDR(5), KEYBUF_ADDR(2));
4
4       IRAB[IRB$V_NORLS_RNF] = 1;
4       RETURN_ON_ERROR (RM$CSEARCH_TREE(), IRAB[IRB$V_PRM] = 0);
4
4       ! Reset all the fields necessary to search a SIDR array starting
4       ! from its first element.
4       !
4       BEG_OF_SIDR = .REC_ADDR;
4       END_OF_SIDR = RM$SIDR_END();
4
4       REC_ADDR = RM$SIDR_FIRST(0);
4       IRAB[IRB$W_SAVE_POS] = 0;
3       END;
3
3
2   END;                                ! {of NEXT}
2
2   ! If some serious error was encountered in RMS's search for the next
2   ! non-deleted primary data record such that the search could no longer
2   ! be continued, then RMS releases any bucket locks the process has
```

```
	! outstanding (primary or SIDR but not both), and returns the error.
	!
	! There is one error case which is handled differently. A record not found
	! error on a search for a primary data record is a serious error. To signal
	! to a caller of this routine that in this particular case this routine
	! should never be recalled, RMS returns an invalid internal condition or
	! bug error after storing the RNF status in the RAB's RAB$L_STV field.
	!
	IF NOT .STATUS
	THEN
	    BEGIN

	    GLOBAL REGISTER
		COMMON_IO_STR;

	    IF .IRAB[IRB$L_CURBDB] NEQU 0
	    THEN
		RELEASE (IRAB[IRB$L_CURBDB]);

	    IF .STATUS<0,16> EQLU RMSERR(RNF)
	    THEN
		BEGIN
		RAB[RAB$L_STV] = .STATUS;
		STATUS = RMSERR(BUG);
		END;

	    IRAB[IRB$V_PRM] = 0;
	    RETURN .STATUS;
	    END

	! If RMS has been successful in positioning to the next non-deleted
	! primary data record, then save all information necessary to position
	! to this SIDR array element as if it was the current element (because it
	! soon will be), release the SIDR bucket, and return success.
	!
	ELSE
	    BEGIN

	    GLOBAL REGISTER
		COMMON_IO_STR;

	    LOCAL
		SAVE_REC_ADDR;

	    BDB = .IRAB[IRB$L_NXTBDB];
	    IRAB[IRB$L_NXTBDB] = 0;

	    IRAB[IRB$L_RFA_VBN] = .BDB[BDB$L_VBN];

	    SAVE_REC_ADDR = .REC_ADDR;
	    REC_ADDR = .BEG_OF_SIDR;
	    RM$SIDR_FIRST (1; IRAB[IRB$L_FIRST_VBN], IRAB[IRB$W_FIRST_ID]);
	    REC_ADDR = .SAVE_REC_ADDR;


	    IF .IRAB[IRB$V_PRM]
	    THEN
```

```
; 1317     1378 3              BDB[BDB$V_PRM] = 1;
; 1318     1379 3
; 1319     1380 3           RM$RLSBKT(0);
; 1320     1381 3           IRAB[IRB$V_PRM] = 0;
; 1321     1382 3
; 1322     1383 3           RETURN RMSSUC();
; 1323     1384 2           END;
; 1324     1385 2
; 1325     1386 1       END;                                  . {of the routine RM$SEARCH_SIDR}
```

```
                               3C  BB 00000 RM$SEARCH_SIDR::
                                              POSHR    #^M<R2,R3,R4,R5>                          ; 0866
                   5E          1C  C2 00002   SUBL2    #28, SP                                   ;
         08        AE          01  88 00005   BISB2    #1, FLAGS                                 ; 1005
         04        AE          56  D0 00009   MOVL     REC_ADDR, BEG_OF_SIDR                     ; 1010
                            0000G  30 0000D   BSBW     RM$SIDR_END                               ; 1011
                   6E          50  D0 00010   MOVL     R0, END_OF_SIDR                           ;
                   51      76  A9  3C 00013   MOVZWL   118(IRAB), R1                             ; 1049
                               B3  10 00017   BSBB     RM$POS_BY_COUNT                           ;
    0D    42       A9          04  E0 00019   BBS      #4, 66(IRAB), 1$                          ; 1065
    08    42       A9          01  E0 0001E   BBS      #1, 66(IRAB), 1$                          ; 1067
          20       A7    00A6  C9  91 00023   CMPB     166(IRAB), 32(IDX_DFN)                    ; 1069
                               1A  1E 00029   BGEQU    2$                                        ;
          08       AE          02  88 0002B 1$: BISB2  #2, FLAGS                                 ; 1072
                   51    00A6  C9  9A 0002F   MOVZBL   166(IRAB), R1                             ; 1073
                   50    00B4  CA  3C 00034   MOVZWL   180(IFAB), R0                             ;
                           60 B940 DF 00039   PUSHAL   @96(IRAB)[R0]                             ;
    9E       60 B940           51  28 0003D   MOVC3    R1, @96(IRAB)[R0], @(SP)+                 ;
                               04  11 00043   BRB      3$                                        ; 1065
          08       AE          02  8A 00045 2$: BICB2  #2, FLAGS                                 ; 1083
    05    08       AE          00  E4 00049 3$: BBSC   #0, FLAGS, 4$                             ; 1101
    44    08       AE          01  E1 0004E   BBC      #1, FLAGS, 7$                             ; 1103
                           76  A9  B5 000[illegible] 4$: TSTW 118(IRAB)                          ; 1105
                               3F  12 00056   BNEQ     7$                                        ;
          0C       AE          56  D0 00058   MOVL     REC_ADDR, TMP_REC_ADDR                    ; 1113
                   56      04  AE  D0 0005C   MOVL     BEG_OF_SIDR, REC_ADDR                     ; 1114
                   51          01  CE 00060   MNEGL    #1, R1                                    ; 1116
                            0000G  30 00063   BSBW     RM$REC_OVHD                               ;
                   56          50  C0 00066   ADDL2    R0, REC_ADDR                              ;
    14    1C       A7          06  E1 00069   BBC      #6, 28(IDX_DFN), 5$                       ; 1118
                   5C          01  D0 0006E   MOVL     #1, AP                                    ; 1125
                   50    00B4  CA  3C 00071   MOVZWL   180(IFAB), R0                             ; 1126
                           60 B940 9F 00076   PUSHAB   @96(IRAB)[R0]                             ;
                            0000G  30 0007A   BSBW     RM$RECORD_KEY                             ;
                   5E          04  C0 0007D   ADDL2    #4, SP                                    ;
                               11  11 00080   BRB      6$                                        ; 1118
                   51      20  A7  9A 00082 5$: MOVZBL 32(IDX_DFN), R1                           ; 1129
                   50    00B4  CA  3C 00086   MOVZWL   180(IFAB), R0                             ;
                   50      60  A9  C0 0008B   ADDL2    96(IRAB), R0                              ;
    60    66                   51  28 0008F   MOVC3    R1, (REC_ADDR), (R0)                      ;
                   56      0C  AE  D0 00093 6$: MOVL   TMP_REC_ADDR, REC_ADDR                    ; 1131
                   6E          56  D1 00097 7$: CMPL   REC_ADDR, END_OF_SIDR                     ; 1140
                               03  1F 0009A   BLSSU    8$                                        ;
```

```
                                0094  31 0009C        BRW     15$                                  :
                  6E              56  D1 0009F 8$:    CMPL    REC_ADDR, END_OF_SIDR                : 1148
                                  F3  1E 000A2        BGEQU   7$                                   :
        08        66              02  E1 000A4        BBC     #2, (REC_ADDR), 9$                   : 1150
                               0000G  30 000A8        BSBW    RM$GETNXT_ARRAY                      : 1154
                           76     A9  B6 000AB        INCW    118(IRAB)                            : 1155
                                  EF  11 000AE        BRB     8$                                   : 1148
                                  E5  1E 000B0 9$:    BGEQU   7$                                   : 1164
                           14     AE  9F 000B2        PUSHAB  RFA_ID                               : 1177
                           1C     AE  9F 000B5        PUSHAB  RFA_VBN                              :
                               0000G  30 000B8        BSBW    RM$EXT_ARRY_RFA                      :
                  5E              08  C0 000BB        ADDL2   #8, SP                               :
                           14     AE  DD 000BE        PUSHL   RFA_ID                               : 1179
                           1C     AE  DD 000C1        PUSHL   RFA_VBN                              :
                                FDD9  30 000C4        BSBW    RM$FOLLOW_PTR                        :
                  5E              08  C0 000C7        ADDL2   #8, SP                               :
             10   AE              50  D0 000CA        MOVL    R0, STATUS                           :
           8262   8F       10     AE  B1 000CE        CMPW    STATUS, #33378                       : 1190
                                  03  13 000D4        BEQL    10$                                  :
                                009F  31 000D6        BRW     18$                                  :
                  49       06     AA  E9 000D9 10$:   BLBC    6(IFAB), 13$                         : 1202
        05        66              05  E1 000DD        BBC     #5, (REC_ADDR), 11$                  : 1204
        40        07       A9     05  E0 000E1        BBS     #5, 7(IRAB), 13$                     : 1206
                  54       20     A9  D0 000E6 11$:   MOVL    32(IRAB), BDB                        : 1218
             0A   A4              02  88 000EA        BISB2   #2, 10(BDB)                          : 1219
                  55       18     A4  D0 000EE        MOVL    24(BDB), BKT_ADDR                    : 1220
                  52              56  D0 000F2        MOVL    REC_ADDR, SAVE_REC_ADDR              : 1225
                  66              20  8A 000F5        BICB2   #32, (REC_ADDR)                      : 1226
                           04     AE  DD 000F8        PUSHL   BEG_OF_SIDR                          : 1234
                                  01  DD 000FB        PUSHL   #1                                   :
                               0000G  30 000FD        BSBW    RM$SQUISH_SIDR                       :
                  5E              08  C0 00100        ADDL2   #8, SP                               :
                  14              50  E9 00103        BLBC    R0, 12$                              :
                  52              56  D1 00106        CMPL    REC_ADDR, SAVE_REC_ADDR              : 1236
                                  28  12 00109        BNEQ    15$                                  :
                  56       04     AE  D0 0010B        MOVL    BEG_OF_SIDR, REC_ADDR                : 1239
                               0000G  30 0010F        BSBW    RM$SIDR_END                          : 1240
                  6E              50  D0 00112        MOVL    R0, END_OF_SIDR                      :
                  56              52  D0 00115        MOVL    SAVE_REC_ADDR, REC_ADDR              : 1241
                                  16  11 00118        BRB     14$                                  : 1236
             07   A9              20  8A 0011A 12$:   BICB2   #32, 7(IRAB)                         : 1262
                               0000G  30 0011E        BSBW    RM$GETNXT_ARRAY                      : 1263
                           76     A9  B6 00121        INCW    118(IRAB)                            : 1264
                                  0A  11 00124        BRB     14$                                  : 1202
                               0000G  30 00126 13$:   BSBW    RM$GETNXT_ARRAY                      : 1282
                           76     A9  B6 00129        INCW    118(IRAB)                            : 1283
             07   A9              20  8A 0012C        BICB2   #32, 7(IRAB)                         : 1284
                                FF64  31 00130 14$:   BRW     7$                                   : 1164
        14   08   AE              01  E1 00133 15$:   BBC     #1, FLAGS, 16$                       : 1298
                  51     00A6     C9  9A 00138        MOVZBL  166(IRAB), R1                        : 1300
                  50     00B4     CA  3C 0013D        MOVZWL  180(IFAB), R0                        :
                         60 @940  DF 00142            PUSHAL  @96(IRAB)[R0]                        :
   60 B940        9E              51  28 00146        MOVC3   R1, @(SP)+, @96(IRAB)[R0]            :
             42   A9              20  88 0014C 16$:   BISB2   #32, 66(IRAB)                        : 1302
                               0000G  30 00150        BSBW    RM$CSEARCH_TREE                      : 1303
                  07              50  E8 00153        BLBS    STATUS, 17$                          :
             42   A9       80     8F  8A 00156        BICB2   #128, 66(IRAB)                       :
```

```
                                      4F  11 0015B         BRB     21$                                     ;
           04    AE                   56  D0 0015D 17$:    MOVL    REC_ADDR, BEG_OF_SIDR                   ; 1308
                                   0000G  30 00161         BSBW    RM$SIDR_END                             ; 1309
                 6E                   50  D0 00164         MOVL    R0, END_OF_SIDR                         ;
                                      7E  D4 00167         CLRL    -(SP)                                   ; 1311
                                   0000G  30 00169         BSBW    RM$SIDR_FIRST                           ;
                 5E                   04  C0 0016C         ADDL2   #4, SP                                  ;
                 56                   50  D0 0016F         MOVL    R0, REC_ADDR                            ;
                            76        A9  B4 00172         CLRW    118(IRAB)                               ; 1312
                                    FED1  31 00175         BRW     3$                                      ; 1091
                 32         10        AE  E8 00178 18$:    BLBS    STATUS, 22$                             ; 1329
                            20        A9  D5 0017C         TSTL    32(IRAB)                                ; 1336
                                      0F  13 0017F         BEQL    19$                                     ;
                 54         20        A9  D0 00181         MOVL    32(IRAB), BDB                           ; 1338
                            20        A9  D4 00185         CLRL    32(IRAB)                                ;
                                      7E  D4 00188         CLRL    -(SP)                                   ;
                                   0000G  30 0018A         BSBW    RM$RLSBKT                               ;
                 5E                   04  C0 0018D         ADDL2   #4, SP                                  ;
         82B2    8F         10        AE  B1 00190 19$:    CMPW    STATUS, #33458                          ; 1340
                                      0B  12 00196         BNEQ    20$                                     ;
           0C    A8         10        AE  D0 00198         MOVL    STATUS, 12(RAB)                         ; 1343
           10    AE       8434        8F  3C 0019D         MOVZWL  #33844, STATUS                          ; 1344
           42    A9         80        8F  8A 001A3 20$:    BICB2   #128, 66(IRAB)                          ; 1347
           50               10        AE  D0 001A8         MOVL    STATUS, R0                              ; 1357
                                      40  11 001AC 21$:    BRB     24$                                     ;
                 54         3C        A9  D0 001AE 22$:    MOVL    60(IRAB), BDB                           ; 1365
                            3C        A9  D4 001B2         CLRL    60(IRAB)                                ; 1366
           70    A9         1C        A4  D0 001B5         MOVL    28(BDB), 112(IRAB)                      ; 1368
                 53                   56  D0 001BA         MOVL    REC_ADDR, SAVE_REC_ADDR                 ; 1370
                 56         04        AE  D0 001BD         MOVL    BEG_OF_SIDR, REC_ADDR                   ; 1371
                                      01  DD 001C1         PUSHL   #1                                      ; 1372
                                   0000G  30 001C3         BSBW    RM$SIDR_FIRST                           ;
                 5E                   04  C0 001C6         ADDL2   #4, SP                                  ;
           7C    A9                   51  D0 001C9         MOVL    R1, 124(IRAB)                           ;
         0082    C9                   52  B0 001CD         MOVW    R2, 130(IRAB)                           ;
                 56                   53  D0 001D2         MOVL    SAVE_REC_ADDR, REC_ADDR                 ; 1373
                            42        A9  95 001D5         TSTB    66(IRAB)                                ; 1376
                                      04  18 001D8         BGEQ    23$                                     ;
           0A    A4                   08  88 001DA         BISB2   #8, 10(BDB)                             ; 1378
                                      7E  D4 001DE 23$:    CLRL    -(SP)                                   ; 1380
                                   0000G  30 001E0         BSBW    RM$RLSBKT                               ;
                 5E                   04  C0 001E3         ADDL2   #4, SP                                  ;
           42    A9         80        8F  8A 001E6         BICB2   #128, 66(IRAB)                          ; 1381
                 50                   01  D0 001EB         MOVL    #1, R0                                  ; 1383
                 5E                   1C  C0 001EE 24$:    ADDL2   #28, SP                                 ; 1386
                                      3C  BA 001F1         POPR    #^M<R2,R3,R4,R5>                        ;
                                      05  001F3            RSB                                             ;

; Routine Size:  500 bytes,    Routine Base:  RM$RMS3 + 0160


; 1326          1387  1
; 1327          1388  1 END
; 1328          1389  1
; 1329          1390  0 ELUDOM
```

```
;                              PSECT SUMMARY
;
;      Name                    Bytes                         Attributes
;
;   RM$RMS3                        852  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        70         2        154          00:00.4



;                               COMMAND QUALIFIERS

;      BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS$:RM3SSIDR/OBJ=OBJ$:RM3SSIDR MSRC$:RM3SSIDR/UPDATE-(ENH$:RM3SSIDR)

; Size:          852 code + 0 data bytes
; Run Time:         00:23.7
; Elapsed Time:     00:48.1
; Lines/CPU Min:     3513
; Lexemes/CPU-Min: 14567
; Memory Used:  195 pages
; Compilation Complete
```

0328 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

RM3SSIDR
LIS

RM3UPSIDX
LIS

RM3SRCHKY
LIS

RM3UPDDEL
LIS

RM3UPDATE
LIS