H 5

```
   SSSSSSSSSSSS     000000000     RRRRRRRRRRRR     TTTTTTTTTTTTTTT     333333333        222222222
   SSSSSSSSSSSS     000000000     RRRRRRRRRRRR     TTTTTTTTTTTTTTT     333333333        222222222
   SSSSSSSSSSSS     000000000     RRRRRRRRRRRR     TTTTTTTTTTTTTTT     333333333        222222222
SSS              000         000  RRR         RRR        TTT        333         333  222         222
SSS              000         000  RRR         RRR        TTT        333         333  222         222
SSS              000         000  RRR         RRR        TTT        333         333  222         222
SSS              000         000  RRR         RRR        TTT                    333              222
SSS              000         000  RRR         RRR        TTT                    333              222
SSS              000         000  RRR         RRR        TTT                    333              222
   SSSSSSSSS     000         000  RRRRRRRRRRRR           TTT                 333              222
   SSSSSSSSS     000         000  RRRRRRRRRRRR           TTT                 333              222
   SSSSSSSSS     000         000  RRRRRRRRRRRR           TTT                 333              222
            SSS  000         000  RRR   RRR              TTT                    333        222
            SSS  000         000  RRR   RRR              TTT                    333        222
            SSS  000         000  RRR   RRR              TTT                    333        222
            SSS  000         000  RRR      RRR           TTT        333         333     222
            SSS  000         000  RRR      RRR           TTT        333         333     222
            SSS  000         000  RRR      RRR           TTT        333         333     222
SSSSSSSSSSSS        000000000     RRR         RRR        TTT           333333333     222222222222222
SSSSSSSSSSSS        000000000     RRR         RRR        TTT           333333333     222222222222222
SSSSSSSSSSSS        000000000     RRR         RRR        TTT           333333333     222222222222222
```

_$2

Pse

SOR

SOR

SOR

SOR

_LI

FILEID**SORLIB

J 8

```
  SSSSSSSS    000000    RRRRRRRR  LL          IIIIII    BBBBBBBB
  SSSSSSSS    000000    RRRRRRRR  LL          IIIIII    BBBBBBBB
SS          00      00  RR      RR LL            II      BB      BB
SS          00      00  RR      RR LL            II      BB      BB
SS          00      00  RR      RR LL            II      BB      BB
SS          00      00  RR      RR LL            II      BB      BB
  SSSSSS    00      00  RRRRRRRR  LL            II      BBBBBBBB
  SSSSSS    00      00  RRRRRRRR  LL            II      BBBBBBBB
        SS  00      00  RR  RR    LL            II      BB      BB
        SS  00      00  RR  RR    LL            II      BB      BB
        SS  00      00  RR    RR  LL            II      BB      BB        ....
        SS  00      00  RR    RR  LL            II      BB      BB        ....
SSSSSSSS      000000    RR      RR LLLLLLLLLL  IIIIII    BBBBBBBB          ....
SSSSSSSS      000000    RR      RR LLLLLLLLLL  IIIIII    BBBBBBBB          ....

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
```

```
! File: SORLIB.REQ IDENT = 'V04-000'    ! File: SORLIB.REQ Edit: PDG3034
!
!****************************************************************************
!*                                                                          *
!*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
!*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
!*  ALL RIGHTS RESERVED.                                                    *
!*                                                                          *
!*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
!*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
!*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
!*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
!*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
!*  TRANSFERRED.                                                            *
!*                                                                          *
!*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
!*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
!*  CORPORATION.                                                            *
!*                                                                          *
!*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
!*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
!*                                                                          *
!*                                                                          *
!****************************************************************************
!


!++
!
! FACILITY:     VAX-11 SORT / MERGE
!
! ABSTRACT:
!
!       This is the common definition file for VAX-11 SORT / MERGE.
!       All definitions of interest to more than one module are in this file.
!       This file is used as a library source.
!
! ENVIRONMENT:  VAX/VMS user mode
!
! AUTHOR: P. Gilbert, CREATION DATE: 07-Dec-1981
!
! MODIFIED BY:
!
!       T03-015         Original
!       T03-016 Add section on pad characters, and correct the extension for
!               specification files (.SRT).  PDG 13-Dec-1982
!       T03-017 Add WF_NAMES, CFT indices of work file names.  PDG 26-Dec-1982
!       T03-018 Added DDB_CHAN.  PDG 28-Dec-1982
!       T03-019 Make work-file description blocks (WFBs) distinct from DDBs.
!               PDG 31-Dec-1982
!       T03-020 Add clean-up routines.  PDG 4-Jan-1983
!       T03-021 Add WFB_DEV.  PDG 6-Jan-1983
!       T03-022 Removed PT/ST_ADR; added BS_DECM, WRK_SIZ.  PDG 26-Jan-1983
!       T03-023 Change STAT_K_WRK_USE to STAT_K_WRK_ALQ.  Added WFB_USE field.
!               Added COM_MRG_STREAM fcr stable merges.  PDG 27-Jan-1983
!       T03-024 Remove section on pad characters.  Add COM_PAD.  PDG 8-Feb-1983
!       T03-025 Remove unreferenced fields.  Change linkage declarations so
!               register information is available to SOR$$KEY_SUB at run time.
!               Define the macro SOR $FATAL.  PDG 16-Mar-1983
!       T03-026 Give the SOR$RO_CODE_n PSECTs the EXE attr.  PDG 7-Ap -1983
!       T03-027 Information hiding of WFB structure.  PDG 12-Apr-1983
!       T03-028 Move definitions of fields specific to scratch-i/o to SORSCRIO
!               from this module.  PDG 18-Apr-1983
!       T03-029 Reduce COM_K_SCRATCH.  PDG 22-Apr-1983
!       T03-030 Correct size of COM_WF_NAMES.  PDG 17-May-1983
!       T03-031 Add COM_ARCHFLAG.  PDG 31-Jan-1984
!       T03-032 Add COLL_BLOCK stuff.  PDG 22-Feb-1984
!       T03-033 Change TUN_K_BUFSIZE to 5 blocks for VAXELN.
!               Add support for VAXELN.  Jeff East 3/13/84
!       T03-034 Change COM_RHB to COM_RHB_INP and COM_RHB_OUT.
!               This is to avoid problems with merge, where an incoming
!               record overwrites the VFC area for the outgoing record.
!               PDG 24-Jul-1984
!--

LIBRARY 'SYS$LIBRARY:STARLET';
LIBRARY 'SYS$LIBRARY:XPORT';
```

```
!                   X P O R T
!
!       The use of XPORT causes some problems, most notably with alignment,
!       and the default sign extension.  The following macros are used.
!
MACRO
    XBYTE =     $ALIGN(BYTE) %EXPAND $BITS(8) %,
    XWORD =     $ALIGN(WORD) %EXPAND $BITS(16) %,
    XLONG =     $ALIGN(FULLWORD) %EXPAND $BITS(32) %,
    XDESC =     $ALIGN(FULLWORD) $SUB_BLOCK(2) %,
    XADDR =     $ALIGN(FULLWORD) $ADDRESS %;
$SHOW(FIELDS)
```

```
!                         POSITION AND SIZE MACROS
!
!
MACRO
        ! Macros used for field references
        !
        A_=              0, 0,0 %,
        L_=              0,32,0 %,
        BASE_=         0,0, 0,0 %,


        ! Macros to construct a bit mask from a standard four-component field
        ! definition (offset, position, size, extension).  The result has set
        ! bits in those positions that belong to the field.  A list of field
        ! definitions can be specified.
        !
        ! Example:
        !
        !       MACRO
        !           A=0,2,4,0%,
        !           B=0,9,1,0%;
        !
        !       MASK_(A,B) is equal to %B'1000111100'
        !
        XMASK_[O,P,S,E]=
            (1 ^ ((P)+(S))) - (1 ^ (P)) %,


        MASK_[]=
            (0 OR XMASK_(%REMAINING)) %,


        ! Macros to align a specified value at the bit position specified by a
        ! standard four-component field definition (offset, position, size,
        ! extension).  A list of values and field definitions can be specified.
        !
        ! Example:
        !
        !       MACRO
        !           A=0,2,4,0%,
        !           B=0,9,1,0%;
        !
        !       ALIGN_(7,A,1,B) is equal to 7^2 OR 1^9
        !
        XALIGN_[V,O,P,S,E]=
            ((V) ^ (P)) %,


        ALIGN_[]=
            (0 OR XALIGN_(%REMAINING)) %;
```

```
!                       G E N E R A L
!
!
LITERAL
        TRUE=           1,
        FALSE=          0;

MACRO
        ELIF=           ELSE IF %;

MACRO
        ! Macro to round a value to the next higher multiple of a number.
        !
        ! The first parameter is the number which is to be rounded.
        ! The second parameter is the multiple up to which we round.
        !       If omitted, the default for the second parameter is %UPVAL
        !       The second parameter should be a literal, and a power of 2.
        !
        ROUND_(A,B) =
            %IF %NULL(B)
            %THEN (((A) + %UPVAL-1) AND NOT (%UPVAL-1))
            %ELSE (((A)+  (B)   -1) AND NOT ((B)   -1))
            %FI %;


MACRO
        ! Macro to calculate floor(log2(constant))
        !
        LN2_(A)=
            (%NBITSU(A)-1) %;

MACRO
        ! Macro to signal an internal consistency check.
        !
        BUGCHECK(A)=
            BEGIN BUILTIN CHMU;
            CHMU(%REF(0));
            0
            END %;

MACRO
        ! Macro to establish a condition handler.
        !
        ESTABLISH_(X) =
            BEGIN BUILTIN FP;
            .FP = X;
            END %;

MACRO
        ! Macro to produce a list of names
        !
        PREFIX_(A)[B] = %NAME(A,B) %;
```

```
MACRO

        ! Macros to determine if the value of an expression is one of a set of
        ! specified small-integer values.  These macros can be used only if the
        ! following conditions are met:
        !
        !       The value to be tested is in the range 0 through 127.
        !
        !       The values to be tested for are all in the range 0 through 31.
        !
        ! Example:
        !
        !       IF ONEOF_(.X, BMSK_(1,3,5)) ...
        !
        ! The code generated is much more efficient than a series of comparisons
        ! (provided that the parameters of BMSK_ are all compile-time constant).

        XBMSK_[A]=
            %IF (A) GTRU 31 %THEN %WARN('ONEOF won''t work') %FI
            (1 ^ (31 - (A))) %,


        BMSK_[]=
            (0 OR XBMSK_(%REMAINING)) %,


        ONEOF_(A,B)=
            (((B) ^ (A)) LSS 0) %;

MACRO

        ! Macros to create initialized, read-only bit-vectors.
        ! The first parameter to BV_ is the largest element which will be
        ! accessed in the bit-vector.
        !
        ! For example:
        !
        ! OWN   PRIMES: BV_( 51, 2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,51 );
        ! IF .PRIMES[.I]
        ! THEN  %( I is Prime )%
        ! ELSE  %( I is Composite )%
        !

        BV_1_[A] = [A] = 1 %,

        BV_(M) = BITVECTOR[M+1]
            PSECT(SOR$RO_CODE) PRESET( BV_1_(%REMAINING) ) %;

MACRO
        ! Macros to distinguish whether the value of an expression is amoung
        ! one set of values, or another set of values, based on a single bit.
        ! An error diagnostic is issued if a single bit will not suffice.
        !
        DIST_(X,Y,Z) =
            BEGIN
            LITERAL
                M =(DIST1_(%REMOVE(Y)) XOR DIST1_(%REMOVE(Z))) AND NOT
```

```
; M 0250  0                     (DIST2_(%REMOVE(Y))  OR DIST2_(%REMOVE(Z))),
; M 0251  0                L = %NBITSU(M XOR (M-1))-1;
; M 0252  0           %IF M EQL 0 %THEN %ERROR('Oops') %FI
; M 0253  0           %IF (DIST1_(%REMOVE(Y)) AND 1^L) EQL 0
; M 0254  0           %THEN               ((X) AND 1^L) EQL 0
; M 0255  0           %ELSE               ((X) AND 1^L) NEQ 0
; M 0256  0           %FI
;   0257  0           END %,
;   0258  0       DIST1_(X) = X %,
;   0259  0       DIST2_(X)[] = (0 OR DIST3_(X,%REMAINING) + 0) %,
;   0260  0       DIST3_(X)[Y] = (X XOR Y) %;
;      0    !                      DEBUGGING CODE
;      0    !
;      0    !
;      0    !       This section defines macros to aid in writing debugging code.
;      0    !
;      0    !       The %VARIANT switch is used to conditionally include compiler debugging
;      0    !       code.  When %VARIANT is true, debugging code is included.  When it is
;      0    !       false, debugging code is omitted.  The macro DEB_CODE is provided to
;      0    !       bracket debugging code that is to be unconditionally executed.
;      0    !
;      0    !       In addition, the global variable "SOR$$D" in the COMENTRY module can be
;      0    !       used to obtain conditional execution of debugging code.  This variable
;      0    !       is initialized to zero, but may be altered during the initial DEBUG
;      0    !       dialogue, before the compiler is started:
;      0    !
;      0    !               DBG>D SOR$$D=%X'D6003FFF'       (for example)
;      0    !               DBG>D SOR$$D=1                  (for example)
;      0    !               DBG>G
;      0    !
;      0    !       The bits in the variable "SOR$$D" are allocated as follows:
;      0    !                0      %X'00000001'    Dump run information
;      0    !                1      %X'00000002'    Dump incremental statistics
;      0    !                2      %X'00000004'    Dump allocation information
;      0    !               ..          ...              ...
;      0    !               30      %X'40000000'    Unassigned
;      0    !               31      %X'80000000'    Unassigned
;      0    !
;      0    !       The macro DEB_SWITCH is provided to bracket conditionally executed
;      0    !       debugging code.
;      0    !
;      0    MACRO
;      0
;      0            ! Macro to bracket unconditional debugging code.  The parameter is an
;      0            ! expression that will be compiled if %VARIANT is true.
;      0            !
; M    0            DEB_CODE(A)=
; M    0                %IF %VARIANT
; M    0                %THEN
; M    0                    A
;      0                %FI %,
;      0
;      0
;      0            ! Macro to bracket conditional debugging code.  The first parameter is
;      0            ! a bit number in the variable SOR$$D, and the second parameter is an
;      0            ! expression that will be evaluated if that bit is set.  The entire
;      0            ! expansion is compiled only if %VARIANT is true.
;      0            !
; M    0            DEB_SWITCH(A,B)=
; M    0                %IF %VARIANT
; M    0                %THEN
; M    0                    BEGIN EXTERNAL SOR$$D;
; M    0                    IF .SOR$$D<A,1> THEN B;
; M    0                    END
;      0                %FI %,
;      0
;      0
;      0            ! Macro to test an assertion about compile-time constants.
;    0318  0                    !
; M  0319  0                    ASSERT (A)=
; M  0320  0                        %IF NOT (A)
; M  0321  0                        %THEN
; M  0322  0                            %ERROR('Assertion failed')
;    0323  0                        %FI %;
```

```
        !                       MAXIMUM VALUES
        !
        !
        LITERAL
                MAX_KEYS=       255,                ! Maximum number of sort keys allowed
                MAX_FILES=      10,                 ! Maximum number of input files.
                MIN_WORK_FILES= 1,                  ! Minimum number of work files
                DEF_WORK_FILES= 2,                  ! Default number of work files
                MAX_WORK_FILES= 10,                 ! Maximum number of work files
                MAX_MERGE_ORDER=10,                 ! Maximum merge order
                MAX_SPC_LINE=   132,                ! Maximum length of spec file line

                MAX_SEQ_RECLEN= 32767,              ! Maximum sequential file record length
                MAX_REL_RECLEN= 16384,              ! Maximum relative file record length
                MAX_IDX_RECLEN= 16384,              ! Maximum indexed file record length
                MAX_ISAMKEYLEN= 255,                ! Maximum index key data item length
                MAX_REFSIZE=    65535,              ! Maximum length of a referenceable data-item
                MAX_PSECTSIZE=  2147483647;         ! Maximum length of a PSECT
        LITERAL
                MIN_MBC=        7,                  ! Minimum MBC count
                MAX_MBC=        16,                 ! Maximum MBC count (for RP06)
                MIN_MBF=        0,                  ! Minimum MBF count
                MAX_MBF=        2;                  ! Maximum MBF count
        LITERAL
                DEF_FILE_ALLOC= 128*3,              ! Default file allocation
                DEF_TRM_ALLOC=  16;                 ! Default allocation for terminals
        LITERAL
                COM_K_BPERPAGE= 512,                ! Bytes per page
                COM_K_BPERBLOCK= 512;               ! Bytes per disk block
        LITERAL
                ! Define a literal for the amount of work space to allocate
                ! for specification text, and another for the amount of work space
                ! to allocate if we only need to process a collating sequence.
                !
                WRK_K_ALLOC=    128 * COM_K_BPERPAGE,   ! Allocation for work area
                WRK_K_COLLATE=  6 * 256;                ! Alloc to process collating sequence
```

```
!                       INTERFACE VALUES
!
!
LITERAL
        ! Datatype values for use in the key definition buffer (KEY_BUFFER).
        !
        ! These are also used to define the global literals SOR$GK_xxx_KEY.
        !
        ! These are used only for compatability purposes.
        !
        KEY_K_CHAR=     1,              ! Character data
        KEY_K_BIN=      2,              ! Signed binary data
        KEY_K_ZONE=     3,              ! Zoned decimal
        KEY_K_PACK=     4,              ! Packed decimal
        KEY_K_USB=      5,              ! Unsigned binary
        KEY_K_DLO=      6,              ! Decimal leading overpunch
        KEY_K_DLS=      7,              ! Decimal leading separate
        KEY_K_DTO=      8,              ! Decimal trailing overpunch
        KEY_K_DTS=      9,              ! Decimal trailing separate
        KEY_K_FLT=      10,             ! Floating
        KEY_K_FLTD=     11,             ! D_floating
        KEY_K_FLTG=     12,             ! G_floating
        KEY_K_FLTH=     13,             ! H_floating
        KEY_K_MAX=      13;             ! Maximum


LITERAL
        ! Values for sort types, passed to SOR$INIT_SORT.
        !
        ! These are also used to define the global literals SOR$GK_xxx.
        !
        TYP_K_RECORD=   1,              ! Record sort
        TYP_K_TAG=      2,              ! Tag sort
        TYP_K_INDEX=    3,              ! Index sort
        TYP_K_ADDRESS=  4,              ! Address sort
        TYP_K_MAX=      4;              ! Maximum sort type

MACRO
        ! Options flags, passed to SOR$INIT_SORT and SOR$INIT_MERGE.
        !
        ! These are used to define the global literals SOR$V_xxx and SOR$M_xxx.
        !
        OPT_STABLE=     0, 0, 1, 0 %,   ! Stable sort
        OPT_EBCDIC=     0, 1, 1, 0 %,   ! EBCDIC collating sequence
        OPT_MULTI=      0, 2, 1, 0 %,   ! MULTINATIONAL collating sequence
        OPT_NOSIGNAL=   0, 3, 1, 0 %,   ! Don't signal errors
        OPT_SEQ_CHECK=  0, 4, 1, 0 %,   ! Sequence check on merge input
!       unused=         0, 5, 1, 0 %,
        OPT_NODUPS=     0, 6, 1, 0 %,   ! Delete records with duplicate keys
        OPT_FIXED=      0, 7, 1, 0 %,   ! Records are fixed length (NYUsed)
!       OPT_LOCATE=     0, 8, 1, 0 %,   ! Use locate mode with RETURN_REC
        OPT_LOAD_FILL=  0, 9, 1, 0 %;   ! Use LOAD_FILL on output file

LITERAL
        ! Values to index the sort statistics
        !
        ! These are also used to define the global literals SOR$GK_STAT_xxx.
```

```
;    0417  0           !
; P  0418  0           $EQULST(STAT_K_, GBL, 0, 1,
; P  0419  0           (IDENT,),         ! Address of ASCIC string for version number
; P  0420  0           (REC_INP,),       ! Records Input
; P  0421  0           (REC_SOR,),       ! Records Sorted
; P  0422  0           (REC_OUT,),       ! Records Output
; P  0423  0           (LRL_INP,),       ! LRL for Input
; P  0424  0           (LRL_INT,),       ! LRL of internal length record
; P  0425  0           (LRL_OUT,),       ! LRL for Output
; P  0426  0           (NODES,),         ! Nodes in sort tree
; P  0427  0           (INI_RUNS,),      ! Initial dispersion runs
; P  0428  0           (MRG_ORDER,),     ! Maximum merge order
; P  0429  0           (MRG_PASSES,),    ! Number of merge passes
; P  0430  0      !    (WSEXTENT,),      ! Working-set extent
; P  0431  0      !    (MEM_USE,),       ! Memory usage
; P  0432  0           (WRK_ALQ,),       ! Work file usage
; P  0433  0      !    (DIRIO,),         ! Direct I/Os
; P  0434  0      !    (BUFIO,),         ! Buffered I/Os
; P  0435  0      !    (PAGEFLTS,),      ! Page faults
; P  0436  0      !    (CPU_TIME,),      ! CPU time
; P  0437  0      !    (ELA_TIME,),      ! Elapsed time
; P  0438  0           (MBC_INP,),       ! MBC for Input
; P  0439  0           (MBC_OUT,),       ! MBC for Output
; P  0440  0           (MBF_INP,),       ! MBF for Input
; P  0441  0           (MBF_OUT,),       ! MBF for Output
;    0442  0           (MAX_STAT,));     ! Last stat value
;    0443  0
;    0444  0      ! Define a single key description in the key description buffer
;    0445  0      !
;    0446  0      $UNIT_FIELD
;    0447  0              KBF_FIELDS =
;    0448  0              SET
; L  0449  0              KBF_TYPE=       [XWORD],        ! Data type of key
; %PRINT:                                 [0,0,16,0]   (+%X'0')
; L  0450  0              KBF_ORDER=      [XWORD],        ! True iff descending order
; %PRINT:                                 [2,0,16,0]   (+%X'2')
; L  0451  0              KBF_POSITION=   [XWORD],        ! Offset to key within record (1..LRL)
; %PRINT:                                 [4,0,16,0]   (+%X'4')
; L  0452  0              KBF_LENGTH=     [XWORD]         ! Length of key
; %PRINT:                                 [6,0,16,0]   (+%X'6')
;    0453  0              TES;
;    0454  0      LITERAL
;    0455  0              KBF_K_SIZE =    $FIELD_SET_UNITS;       ! Size in bytes
;    0456  0      MACRO
;    0457  0              KBF_BLOCK =     %EXPAND $UNIT_BLOCK(KBF_K_SIZE) FIELD(KBF_FIELDS) %;
;    0458  0
;    0459  0      ! Define the key description buffer
;    0460  0      !
;    0461  0      MACRO
;    0462  0              KEY_NUMBER =    0, 0, 16, 0 %,                  ! Number of keys
;    0463  0              KEY_KBF(N) =    2 + KBF_K_SIZE * (N), 0, 0, 0 %;
;    0464  0      STRUCTURE
;    0465  0              KEY_BLOCK[O,P,S,E;BS=MAX_KEYS] =
;    0466  0                  [2 + KBF_K_SIZE*BS] (KEY_BLOCK + O) <P,S,E>;
;    0467  0
;    0468  0
;    0469  0      ! Define the structure of a COLL_BLOCK, which is passed to SOR$SPEC_FILE
;     0470  0      !
;     0471  0      MACRO
;     0472  0              COLL_W_LENGTH = 0, 0, 16, 0 %,                  ! Length of this block
;     0473  0              COLL_B_PAD =    3, 0,  8, 0 %,
;     0474  0              COLL_A_PTAB =   4, 0, 32, 0 %;
```

```
:   0475  0      !                       COMMON INFORMATION
:   0476  0      !
:   0477  0      !       Information that must be available between calls to sort/merge is
:   0478  0      !       stored in a dynamically allocated data structure.  The address of this
:   0479  0      !       data structure is stored in a context parameter that is passed to the
:   0480  0      !       sort/merge routines.  If the context parameter is missing, the global
:   0481  0      !       variable SOR$$CONTEXT is assumed to contain this pointer.
:   0482  0      !
:   0483  0      COMPILETIME
:   0484  0          U__ = 0;
:   0485  0      MACRO
: M 0486  0          U_= %ASSIGN(U__,U__+1)                ! Macro to generate unique names
:   0487  0              %NAME('U_',%NUMBER(U__)) %;
:   0488  0
:   0489  0      LITERAL
:   0490  0              COM_K_TREE=     13,              ! Number of longwords for TREE_INSERT
:   0491  0              COM_K_SCRATCH=  10,              ! Number of longwords for SCRATCH_IO
:   0492  0              COM_K_CDD=      2;               ! Number of longwords for CDD stuff
:   0493  0
:   0494  0      $FIELD  CTX_FIELDS =
:   0495  0              SET
:   0496  0              !
:   0497  0              ! Routines
:   0498  0              !
: L 0499  0              COM_COMPARE=    [XADDR],          ! Address of user comparison routine
: %PRINT:                                  [0,0,32,0]   (+%X'0')
: L 0500  0              COM_EQUAL=      [XADDR],          ! Address of equal-key routine
: %PRINT:                                  [1,0,32,0]   (+%X'4')
: L 0501  0              COM_INPUT=      [XADDR],          ! Address of input conversion routine
: %PRINT:                                  [2,0,32,0]   (+%X'8')
: L 0502  0              COM_OUTPUT=     [XADDR],          ! Address of ouput routine
: %PRINT:                                  [3,0,32,0]   (+%X'C')
: L 0503  0              COM_LENADR=     [XADDR],          ! Address of length, address routine
: %PRINT:                                  [4,0,32,0]   (+%X'10')
: L 0504  0              COM_NEWRUN=     [XADDR],          ! Address of new run routine
: %PRINT:                                  [5,0,32,0]   (+%X'14')
: L 0505  0              COM_ROUTINES=   [XDESC],          ! A dymanic string descriptor
: %PRINT:                                  [6,0,0,0]   (+%X'18')
:   0506  0              !
:   0507  0              ! Storage for TREE_INSERT
:   0508  0              !
: L 0509  0              COM_TREE_INSERT=[$SUB_BLOCK(COM_K_TREE)], ! Storage for TREE_INSERT
: %PRINT:                                  [8,0,0,0]   (+%X'20')
:   0510  0              !
:   0511  0              ! Global sort information
:   0512  0              !
: L 0513  0              COM_CTXADR=     [XLONG],          ! Address of users context longword
: %PRINT:                                  [21,0,32,0]   (+%X'54')
: L 0514  0              COM_SORT_TYPE=  [XBYTE],          ! Type of sort (TYP_K_RECORD,...)
: %PRINT:                                  [22,0,8,0]   (+%X'58')
: L 0515  0              COM_NUM_FILES=  [XBYTE],          ! Number of input files
: %PRINT:                                  [22,8,8,0]   (+%X'59')
: L 0516  0              COM_WRK_FILES=  [XBYTE],          ! Number of work files to use
: %PRINT:                                  [22,16,8,0]   (+%X'5A')
: L 0517  0              COM_STABLE=     [$BIT],           ! Stable sort requested
: %PRINT:                                  [22,24,1,0]   (+%X'5B')
: L 0518  0              COM_SEQ_CHECK=  [$BIT],           ! Sequence check
```

```
; %PRINT:                                [22,25,1,0]    (+%X'5B')
; L 0519  0           COM_SIGNAL=      [$BIT],          ! Sort/merge should signal errors
; %PRINT:                                [22,26,1,0]    (+%X'5B')
; L 0520  0           COM_NOCHKPNT=    [$BIT],          ! Checkpointing should not be done
; %PRINT:                                [22,27,1,0]    (+%X'5B')
; L 0521  0           COM_LOAD_FILL=   [$BIT],          ! Use load-fill on indexed files
; %PRINT:                                [22,28,1,0]    (+%X'5B')
; L 0522  0           COM_NODUPS=      [$BIT],          ! Delete records with duplicate keys
; %PRINT:                                [22,29,1,0]    (+%X'5B')
; L 0523  0           U_=              [$BIT],          ! Use locate mode with RETURN_REC
; %PRINT:                                [22,30,1,0]    (+%X'5B')
;   0524  0           !
;   0525  0           ! Control flow flags
;   0526  0           !
; L 0527  0           COM_FLO_SORT=    [$BIT],          ! May call Sort-Merge
; %PRINT:                                [22,31,1,0]    (+%X'5B')
; L 0528  0           COM_FLO_NOINIT= [$BIT],           ! May not call Pass-Files, Init-Sort or Init-Merge
; %PRINT:                                [23,0,1,0]    (+%X'5C')
; L 0529  0           COM_FLO_RELEASE=[$BIT],           ! May call Release-Rec
; %PRINT:                                [23,1,1,0]    (+%X'5C')
; L 0530  0           COM_FLO_RETURN= [$BIT],           ! May call Return-Rec or End-Sort
; %PRINT:                                [23,2,1,0]    (+%X'5C')
; L 0531  0           COM_FLO_DOMERGE=[$BIT],           ! May call Do-Merge
; %PRINT:                                [23,3,1,0]    (+%X'5C')
; L 0532  0           COM_FLO_ABORT=  [$BIT],           ! May only call End-Sort
; %PRINT:                                [23,4,1,0]    (+%X'5C')
;   0533  0           !
;   0534  0           ! Flags to amend for V3 compatability hacks
;   0535  0           !
; L 0536  0           COM_HACK_2ARGS= [$BIT],           ! Pass only 2 args to callback routines
; %PRINT:                                [23,5,1,0]    (+%X'5C')
; L 0537  0           COM_HACK_STRIP= [$BIT],           ! Strip the keys
; %PRINT:                                [23,6,1,0]    (+%X'5C')
;   0538  0           !
;   0539  0           ! Merge-specific fields
;   0540  0           !
;   0541  0           ! Note that COM_MRG_ORDER is non-zero iff this is a merge
;   0542  0           !
; L 0543  0           COM_MERGE=       [$BIT],          ! Indicates a merge (not a sort)
; %PRINT:                                [23,7,1,0]    (+%X'5C')
; L 0544  0           COM_MRG_ORDER=   [XBYTE],         ! Order of the merge
; %PRINT:                                [23,8,8,0]    (+%X'5D')
;   0545  0           !
;   0546  0           ! Spec text processing stuff
;   0547  0           !
; L 0548  0           COM_SPEC_TKS=    [XWORD],         ! Size of keys portion of internal node
; %PRINT:                                [23,16,16,0]    (+%X'5E')
;   0549  0           !
;   0550  0           ! Merge-specific fields
;   0551  0           !
; L 0552  0           COM_MRG_INPUT=   [XADDR],         ! User-written merge input routine
; %PRINT:                                [24,0,32,0]    (+%X'60')
; L 0553  0           COM_MRG_STREAM= [XLONG],          ! Stream number for stable merges
; %PRINT:                                [25,0,32,0]    (+%X'64')
;   0554  0           !
;   0555  0           ! Collating sequence stuff
;   0556  0           !
; L 0557  0              COM_COLLATE=    [XADDR],           ! Addr of collating sequence routine
; %PRINT:                                  [26,0,32,0]   (+%X'68')
; L 0558  0              COM_ST_SIZ=     [XLONG],           ! Size (write-only)
; %PRINT:                                  [27,0,32,0]   (+%X'6C')
;   0559  0              !
;   0560  0              ! Key information
;   0561  0              !
; L 0562  0              U_=             [XADDR],           ! Address of key descriptions
; %PRINT:                                  [28,0,32,0]   (+%X'70')
; L 0563  0              COM_SPEC_FILE=  [XADDR],           ! Addr of structures from spec file
; %PRINT:                                  [29,0,32,0]   (+%X'74')
; L 0564  0              COM_TKS=        [XBYTE],           ! Total key size (as specified by user)
; %PRINT:                                  [30,0,8,0]    (+%X'78')
;   0565  0              !
;   0566  0              ! Override flags - ignore the specification text for these options
;   0567  0              !
; L 0568  0              COM_OVR_PROC=   [$BIT],            ! Process specified
; %PRINT:                                  [30,8,1,0]    (+%X'79')
; L 0569  0              COM_OVR_KEY=    [$BIT],            ! Key(s) specified
; %PRINT:                                  [30,9,1,0]    (+%X'79')
;   0570  0      !no way COM_OVR_CHKSEQ= [$BIT],            ! Check sequence specified
;   0571  0      !no way COM_OVR_STABLE= [$BIT],            ! Stable specified
; L 0572  0              COM_OVR_COLSEQ= [$BIT],            ! Collating sequence specified
; %PRINT:                                  [30,10,1,0]   (+%X'79')
; L 0573  0              COM_BS_DECM=    [$BIT],            ! Base sequence was DEC_MULTINATIONAL
; %PRINT:                                  [30,11,1,0]   (+%X'79')
; L 0574  0              U_=             [$BITS(4)],
; %PRINT:                                  [30,12,4,0]   (+%X'79')
;   0575  0              !
;   0576  0              ! Counts
;   0577  0              !
; L 0578  0              COM_RUNS=       [XWORD],           ! Current number of runs
; %PRINT:                                  [30,16,16,0]  (+%X'7A')
; L 0579  0              COM_INP_RECNUM= [XLONG],           ! Input record number (stable & stats)
; %PRINT:                                  [31,0,32,0]   (+%X'7C')
;   0580  0              !
;   0581  0              ! Collating sequence information
;   0582  0              !
; L 0583  0              COM_TIE_BREAK=  [$BIT],            ! Indicates tie-breaking
; %PRINT:                                  [32,0,1,0]    (+%X'80')
;   0584  0              !
;   0585  0              ! Record format information
;   0586  0              !
; L 0587  0              COM_VAR=        [$BIT],            ! Flag indicating variable length input
; %PRINT:                                  [32,1,1,0]    (+%X'80')
; L 0588  0              U_=             [$BITS(6)],
; %PRINT:                                  [32,2,6,0]    (+%X'80')
; L 0589  0              COM_MINVFC=     [XBYTE],           ! Length of VFC area in internal node
; %PRINT:                                  [32,8,8,0]    (+%X'81')
; L 0590  0              COM_MAXVFC=     [XBYTE],           ! Length of COM_RHB buffer
; %PRINT:                                  [32,16,8,0]   (+%X'82')
; L 0591  0              COM_FORMATS=    [XBYTE],           ! Number of different record formats
; %PRINT:                                  [32,24,8,0]   (+%X'83')
; L 0592  0              COM_LRL=        [XWORD],           ! Longest input record length
; %PRINT:                                  [33,0,16,0]   (+%X'84')
; L 0593  0              COM_SRL=        [XWORD],           ! Shortest record length
; %PRINT:                                  [33,16,16,0]  (+%X'86')
; L 0594  0           COM_LRL_INT=    [XWORD],         ! Length of internal format record
; %PRINT:                               [34,0,16,0]    (+%X'88')
; L 0595  0           COM_LRL_OUT=    [XWORD],         ! Longest output record length
; %PRINT:                               [34,16,16,0]   (+%X'8A')
; L 0596  0           COM_RHB_INP=    [XADDR],         ! Address of VFC area (input side)
; %PRINT:                               [35,0,32,0]    (+%X'8C')
; L 0597  0           COM_RHB_OUT=    [XADDR],         ! Address of VFC area (output side)
; %PRINT:                               [36,0,32,0]    (+%X'90')
;   0598  0           !
;   0599  0           ! File information
;   0600  0           !
; L 0601  0           COM_PASS_FILES= [XADDR],         ! Output file characteristics
; %PRINT:                               [37,0,32,0]    (+%X'94')
; L 0602  0           COM_OUT_DDB=    [XADDR],         ! Address of output file DDB
; %PRINT:                               [38,0,32,0]    (+%X'98')
; L 0603  0           COM_INP_DDB=    [XADDR],         ! Address of input file DDBs
; %PRINT:                               [39,0,32,0]    (+%X'9C')
; L 0604  0           COM_INP_CURR=   [XADDR],         ! Address of current input file DDB
; %PRINT:                               [40,0,32,0]    (+%X'A0')
; L 0605  0           COM_INP_ARRAY=  [XADDR],         ! Array of input DDB pointers
; %PRINT:                               [41,0,32,0]    (+%X'A4')
; L 0606  0           COM_FILE_ALLOC= [XLONG],         ! File allocation specified by user
; %PRINT:                               [42,0,32,0]    (+%X'A8')
; L 0607  0           COM_SPC_DDB=    [XADDR],         ! Address of spec file DDB
; %PRINT:                               [43,0,32,0]    (+%X'AC')
;   0608  0           !
;   0609  0           ! Statistics information (used only for statistics)
;   0610  0           !
; L 0611  0           COM_STAT_NODES= [XLONG],         ! Number of nodes in sort tree
; %PRINT:                               [44,0,32,0]    (+%X'B0')
; L 0612  0           COM_STAT_RUNS=  [XWORD],         ! Number of runs from dispersion
; %PRINT:                               [45,0,16,0]    (+%X'B4')
; L 0613  0           COM_STAT_PASSES=[XWORD],         ! Number of merge passes
; %PRINT:                               [45,16,16,0]   (+%X'B6')
; L 0614  0           COM_STAT_MERGE= [XBYTE],         ! Order of the merge
; %PRINT:                               [46,0,8,0]   (+%X'B8')
; L 0615  0           U_=             [$BITS(24)],
; %PRINT:                               [46,8,24,0]    (+%X'B9')
;   0616  0     !     COM_STAT_WS=    [XLONG],         ! Maximum WS used
;   0617  0     !     COM_STAT_VM=    [XLONG],         ! Maximum VM used
; L 0618  0           COM_OMI_RECNUM= [XLONG],         ! Number of omitted records (for stats)
; %PRINT:                               [47,0,32,0]    (+%X'BC')
; L 0619  0           COM_OUT_RECNUM= [XLONG],         ! Output record number (for stats)
; %PRINT:                               [48,0,32,0]    (+%X'C0')
;   0620  0           !
;   0621  0           ! Storage for TREE_INSERT
;   0622  0           !
; L 0623  0           COM_TREE_LEN=   [XLONG],         ! Length of storage for tree
; %PRINT:                               [49,0,32,0]    (+%X'C4')
; L 0624  0           COM_TREE_ADR=   [XLONG],         ! Address of storage for tree
; %PRINT:                               [50,0,32,0]    (+%X'C8')
;   0625  0           !
;   0626  0           ! Scratch I/O information
;   0627  0           !
; L 0628  0           COM_SCRATCH_IO= [$SUB_BLOCK(COM_K_SCRATCH)], ! Storage for SCRATCH_IO
; %PRINT:                               [51,0,0,0]    (+%X'CC')
;   0629  0           !
;        0              ! Locking information
;        0              !
;        0     !        COM_LOCKED=     [XADDR],         ! List of locked code sections
;        0              !
;        0              ! Specification file stuff
;        0              !
; L      0              COM_SPC_TXT=    [XDESC],         ! Dymanic string for spec file text
; %PRINT:                                 [61,0,0,0]   (+%X'F4')
;        0              !
;        0              ! Specification file stufr
;        0              !
; L      0              COM_RDT_SIZ=    [XBYTE],
; %PRINT:                                 [63,0,8,0]   (+%X'FC')
; L      0              COM_KFT_SIZ=    [XBYTE],
; %PRINT:                                 [63,8,8,0]   (+%X'FD')
; L      0              COM_CFT_SIZ=    [XBYTE],
; %PRINT:                                 [63,16,8,0]   (+%X'FE')
; L      0              COM_FDT_SIZ=    [XBYTE],
; %PRINT:                                 [63,24,8,0]   (+%X'FF')
; L      0              COM_TDT_SIZ=    [XBYTE],
; %PRINT:                                 [64,0,8,0]   (+%X'100')
; L      0              COM_PAD=        [XBYTE],         ! Pad character
; %PRINT:                                 [64,8,8,0]   (+%X'101')
; L      0              U_=             [$BITS(16)],
; %PRINT:                                 [64,16,16,0]   (+%X'102')
; L      0              COM_RDT_ADR=    [XADDR],         ! Record definition table
; %PRINT:                                 [65,0,32,0]  (+%X'104')
; L      0              COM_KFT_ADR=    [XADDR],         ! Key/data field table
; %PRINT:                                 [66,0,32,0]  (+%X'108')
; L      0              COM_CFT_ADR=    [XADDR],         ! Constant field table
; %PRINT:                                 [67,0,32,0]  (+%X'10C')
; L      0              COM_FDT_ADR=    [XADDR],         ! Field definition table
; %PRINT:                                 [68,0,32,0]  (+%X'110')
; L      0              COM_TDT_ADR=    [XADDR],         ! Test definition table
; %PRINT:                                 [69,0,32,0]  (+%X'114')
; L      0              COM_CONST_AREA= [XADDR],         ! Constant area (address)
; %PRINT:                                 [70,0,32,0]  (+%X'118')
; L      0              COM_PTAB=       [XADDR],         ! Pointer to 256-byte table
; %PRINT:                                 [71,0,32,0]  (+%X'11C')
; L      0              U_=             [XADDR],
; %PRINT:                                 [72,0,32,0]  (+%X'120')
; L      0              COM_WRK_SIZ=    [XLONG],         ! Length of work area
; %PRINT:                                 [73,0,32,0]  (+%X'124')
; L      0              COM_WRK_ADR=    [XADDR],         ! Address of work area
; %PRINT:                                 [74,0,32,0]  (+%X'128')
; L      0              COM_WRK_END=    [XADDR],         ! Address past end of work area
; %PRINT:                                 [75,0,32,0]  (+%X'12C')
;        0              !
;        0              ! Other stuff
;        0              !
; L      0              COM_WORST=      [XLONG],         ! Worst error we've ever seen
; %PRINT:                                 [76,0,32,0]  (+%X'130')
;        0              COM_WF_NAMES=   ! Counted list of indices into CFT of work file names
; L      0                              [$BYTES(1+MAX_WORK_FILES)],
; %PRINT:                                 [77,0,0,0]   (+%X'134')
;        0              $ALIGN(FULLWORD)
; L      0              COM_CDD=        [$SUB_BLOCK(COM_K_CDD)],        ! Storage for CDD stuff
; %PRINT:                               [80,0,0,0]    (+%X'140')
;    0666  0            !
;    0667  0            ! Additional storage for checkpoint stuff
;    0668  0            !
; L 0669  0             COM_COUNTDOWN=  [XLONG],
; %PRINT:                                 [82,0,32,0]   (+%X'148')
;    0670  0            !
;    0671  0            ! Architectural flags (indicates which instructions are implemented)
;    0672  0            !
; L 0673  0             COM_ARCHFLAG=   [XLONG]
; %PRINT:                                 [83,0,32,0]   (+%X'14C')
;    0674  0            TES;
;    0675  0        LITERAL
;    0676  0            CTX_K_SIZE=     $FIELD_SET_SIZE;        ! Size in longwords
;    0677  0        MACRO
;    0678  0            CTX_BLOCK=      BLOCK[CTX_K_SIZE] FIELD(CTX_FIELDS) %,
;    0679  0            CTX_BLOCK_(S)=  BLOCK[CTX_K_SIZE] FIELD(CTX_FIELDS,S) %;
;    0680  0
; L 0681  0        %MESSAGE('CTX_K_SIZE = ', %NUMBER(CTX_K_SIZE))
;    0682  0
;    0683  0        UNDECLARE %QUOTE U_, U__;
```

```
!                       RECORD FORMATS
!
!       This section describes the various record formats that are used
!       throughout Sort/Merge.
!
!       INPUT RECORD FORMAT:
!
!               VAR (a word) is present only for variable length records
!               VFC is present only for VFC files
!               DATA is always present
!
!       INTERNAL RECORD FORMAT:
!
!           FORM KEY VAR VFC DATA STAB          ! Record sort
!           FORM KEY RFA FILE STAB              ! Tag, address, index
!
!               VAR (a word) is present only for variable length records
!               VFC is present only for VFC files
!               KEY is present for keys or converted keys
!               FORM (a byte) is present only for multiple record formats
!               FILE (a byte) is present only for multi-file non-record sorts
!               STAB (a longword) is present only for stable sorts
!               RFA (RAB$S_RFA bytes) is present for non-record sorts
!
!       OUTPUT RECORD FORMAT:
!
!           VAR VFC DATA                        ! Record, tag sort
!           RFA FILE                            ! Address sort
!           RFA FILE OKEY STAB                  ! Index sort
!
!               VAR (a word) is present only for variable length records
!               VFC is present only for VFC files
!               FILE (a byte) is present only for multi-file non-record sorts
!               OKEY is the unconverted keys
!               STAB (a longword) is present only for stable index sorts


! Assertions can be made on the following literals to determine the relative
! ordering of fields within a record.
!
LITERAL
        COM_ORD_RFA     = 0,    ! RFA field
        COM_ORD_FILE    = 1,    ! File number field
        COM_ORD_FORM    = 2,    ! Format field
        COM_ORD_OKEY    = 3,    ! Original keys (for index sorts)
        COM_ORD_STAB    = 4,    ! Stable longword field
        COM_ORD_KEY     = 5,    ! Key or converted key field
        COM_ORD_VAR     = 6,    ! Length field
        COM_ORD_VFC     = 7,    ! VFC field
        COM_ORD_DATA    = 8,    ! Data field
        COM_ORD_MAX     = 9;    ! Largest order value
```

```
;   0735  0      !                      DEVICE DESCRIPTION BLOCK
;   0736  0      !
;   0737  0      !       The DDB contains information for reading/writing a file.  It does not
;   0738  0      !       contain all RMS structures, since the FAB, NAM, and other blocks may
;   0739  0      !       be discarded, thus decreasing the amount of virtual memory required.
;   0740  0      !
;   0741  0      $UNIT_FIELD
;   0742  0              DDB_FIELDS =
;   0743  0              SET
; L 0744  0              DDB_NEXT=       [XADDR],                ! Pointer to next DDB
; %PRINT:                                  [0,0,32,0]   (+%X'0')
; L 0745  0              DDB_NAME=       [$SUB_BLOCK(2)],        ! File name length/address
; %PRINT:                                  [4,0,0,0]   (+%X'4')
; L 0746  0              DDB_IFI=        [XLONG],                ! Internal file identifier
; %PRINT:                                  [12,0,32,0]   (+%X'C')
; L 0747  0              DDB_FOP=        [XLONG],                ! File options
; %PRINT:                                  [16,0,32,0]   (+%X'10')
; L 0748  0              DDB_RAB_RAB=    [$BYTES(RAB$C_BLN)],    ! Record Access Block
; %PRINT:                                  [20,0,0,0]   (+%X'14')
; L 0749  0              DDB_FIL=        [XBYTE]                 ! Input File number (0 on up)
; %PRINT:                                  [88,0,8,0]   (+%X'58')
;   0750  0              TES;
;   0751  0      LITERAL
;   0752  0              DDB_RAB=        %FIELDEXPAND(DDB_RAB_RAB,0);
;   0753  0      UNDECLARE
;   0754  0              DDB_RAB_RAB;
;   0755  0      LITERAL
;   0756  0              DDB_K_SIZE=     $FIELD_SET_UNITS;       ! Size in bytes
;   0757  0      MACRO
;   0758  0              DDB_BLOCK=      %EXPAND $UNIT_BLOCK(DDB_K_SIZE) FIELD(DDB_FIELDS) %;
;   0759  0
; L 0760  0      %MESSAGE('DDB_K_SIZE = ', %NUMBER(DDB_K_SIZE))
;   0761  0
;   0762  0
;   0763  0      UNDECLARE
;   0764  0              %QUOTE $DESCRIPTOR;
```

SO

```
!                   L I N K A G E S
!
!
!       Several internal routines use JSB linkages to improve performance.
!       Common linkages are defined here.  Linkages to external routines
!       are defined as LNK_routine_name.
!
LITERAL
        COM_REG_SRC1 =  9,
        COM_REG_SRC2 =  10,
        COM_REG_CTX =   11;
MACRO
        %PRESERVE(X)    = %NAME(X,'_PR') %,
        %NOPRESERVE(X)  = %NAME(X,'_NP') %,
        %NOTUSED(X)     = %NAME(X,'_NU') %,
        XREGMASK_[P]    = 1^P %,
        REGMASK_[]      = 0 OR XREGMASK_(%REMAINING) %;
KEYWORDMACRO
        JSB_DEFN_(NAM,PM,GL,PR,NP,NU) =
        LITERAL
            %PRESERVE(NAM)      = REGMASK_(%REMOVE(PR)) + 0,
            %NOPRESERVE(NAM)    = REGMASK_(%REMOVE(NP)) + 0,
            %NOTUSED(NAM)       = REGMASK_(%REMOVE(NU)) + 0;
        LINKAGE NAM = JSB(%REMOVE(PM)):
            %IF NOT %NULL(GL) %THEN GLOBAL(%REMOVE(GL)) %FI
            %IF NOT %NULL(PR) %THEN PRESERVE(%REMOVE(PR)) %FI
            %IF NOT %NULL(NP) %THEN NOPRESERVE(%REMOVE(NP)) %FI
            %IF NOT %NULL(NU) %THEN NOTUSED(%REMOVE(NU)) %FI
        %;

JSB_DEFN_(
        NAM = JSB_INPUT,                ! For COM_INPUT
        PM = <REGISTER=COM_REG_SRC1,REGISTER=COM_REG_SRC2>,
        PR = <COM_REG_SRC2>,
        NP = <0,1,2,3,4,5,6,COM_REG_SRC1>,      ! R6 holds the variable length
        NU = <7,8>,
        GL = <CTX=COM_REG_CTX> );

JSB_DEFN_(
        NAM = JSB_NEWRUN,               ! For COM_NEWRUN
        NU = <4,5,6,7,8,10>,
        NP = <0,1>,
        PR = <2,3,9>,
        GL = <CTX=COM_REG_CTX> );

JSB_DEFN_(
        NAM = JSB_COMPARE,              ! For COM_COMPARE
        PM = <REGISTER=COM_REG_SRC1,REGISTER=COM_REG_SRC2>,
        PR = <COM_REG_SRC1,COM_REG_SRC2>,
        NP = <0,1,2,3,4,5>,
        NU = <6,7,8>,                           ! Really???
        GL = <CTX=COM_REG_CTX> );

JSB_DEFN_(
        NAM = JSB_OUTPUT,               ! For COM_OUTPUT
        PM = <REGISTER=COM_REG_SRC2>,
        PR = <COM_REG_SRC2>,
```

Th
ME

```
                NU = <7,8,9>,
                NP = <0,1,2,3,4,5,6>,               ! R6 needed???
                GL = <CTX=COM_REG_CTX> );

        JSB_DEFN (
                NAM = JSB_EQUAL,                ! For COM_EQUAL
                PM = <REGISTER=COM_REG_SRC1,REGISTER=COM_REG_SRC2>,
                PR = <COM_REG_SRC1,COM_REG_SRC2>,
                NP = <0,1>,
                NU = <2,3,4,5,6,7,8>,
                GL = <CTX=COM_REG_CTX> );

        JSB_DEFN (
                NAM = JSB_LENADR,               ! For COM_LENADR
                PM = <REGISTER=COM_REG_SRC2;REGISTER=0,REGISTER=1>,
                PR = <COM_REG_SRC2>,
                NP = <0,1>,
                NU = <2,3,4,5,6,7,8,9>,
                GL = <CTX=COM_REG_CTX> );

        JSB_DEFN (
                NAM = JSB_INSERT,               ! For SOR$$TREE_INSERT
                PM = <STANDARD>,                ! Can we use registers???
                PR = <7,8>,
                NP = <0,1,2,3,4,5,6,COM_REG_SRC1,COM_REG_SRC2>,
                GL = <CTX=COM_REG_CTX> );

        JSB_DEFN (
                NAM = JSB_READINS,              ! For READ_INSERT
                PM = <REGISTER=6,REGISTER=8>,
                PR = <7,8>,
                NP = <0,1,2,3,4,5,6,9,10>,
                GL = <CTX=COM_REG_CTX> );

        JSB_DEFN (
                NAM = JSB_EXTRACT,              ! For SOR$$TREE_EXTRACT
                PM = <STANDARD>,                ! Can we use registers???
                PR = <7,8>,
                NP = <0,1,2,3,4,5,6,COM_REG_SRC1,COM_REG_SRC2>,
                GL = <CTX=COM_REG_CTX> );

        LINKAGE
                CAL_ACCESS =    CALL (  STANDARD;               ! For SOR$$RFA_ACCESS
                                        REGISTER=0,
                                        REGISTER=1):
                                        GLOBAL(CTX=COM_REG_CTX);
        LINKAGE
                CAL_CTXREG =    CALL:   GLOBAL(CTX=COM_REG_CTX);
```

```
	!			T U N I N G   P A R A M E T E R S
	!
	!
	!	These values are used to tune the sort.
	!
	LITERAL
	    TUN_K_NONTREE =	192,	! Number of pages to not use for the tree
	    TUN_K_FALLBACK =	64,	! Minimum pages for tree for a large sort
	    TUN_K_CALC_FI =	TRUE,	! True to calculate FI in sort tree
	    TUN_K_CALC_FE =	TRUE,	! True to calculate FE in sort tree
	    TUN_K_OUT_PREALL =	TRUE,	! True to preallocate output file
	    TUN_K_WRK_PREALL =	FALSE,	! True to preallocate work files
	    TUN_K_ALIGN_NODE =	2,	! Log2 of alignment for nodes (longword align)
	    TUN_K_ALIGN_TREE =	9,	! Log2 of alignment for sort tree (page align)
	    TUN_K_MRGCOST =	0,	! Cost of merge
	    TUN_K_PURGWS =	FALSE,	! True to purge working set before INIT_TREE
	!   TUN_K_LCK_CTX =	TPUE,	! True to lock context area in WS
	!   TUN_K_LCK_TREE =	3,	! Pages of tree to lock in WS
	!   TUN_K_LCK_CODE =	TRUE,	! True to lock code in WS
	    TUN_K_BINMOVE =	32,	! Max number of bytes to move with binary moves
	    TUN_K_MAX_MERGE =	20;	! Maximum merge order for internal merges
	MACRO
	    TUN_K_BUFSIZE =
		%IF NOT HOSTILE_ELAN
		%THEN		50 * COM_K_BPERPAGE	! Bytes in a buffer
		%ELSE		 5 * COM_K_BPERPAGE	! Bytes in a buffer
		%FI %;
	LITERAL
	    FUN_K_CHECKPOINT =	FALSE;	! True to generate code for checkpointing
	ASSERT_(TUN_K_MAX_MERGE GEQ MAX_MERGE_ORDER)

	%IF NOT FUN_K_CHECKPOINT
	%THEN
	    UNDECLARE %QUOTE COM_NOCHKPNT, %QUOTE COM_COUNTDOWN;
	%FI
```

```
;   0905  0        !                        E R R O R   N U M B E R S
;   0906  0        !
;   0907  0        !
;   0908  0        !       Each message issued has an associated literal value.  The name of the
;   0909  0        !       value is of the form "SOR$_xxx", where "xxx" is the message identifier.
;   0910  0        !
;   0911  0        !       Other shared messages are defined in the SORCOMMAN module.
;   0912  0        !
;   0913  0        REQUIRE 'SRC$:SORMSG';
; L 1090  0        %IF NOT %DECLARED(SORT$_FACILITY)
; U 1091  0        %THEN
; U 1092  0            LITERAL
; U 1093  0                SORT$_FACILITY = SOR$_FACILITY;
; U 1094  0            UNDECLARE
; U 1095  0                SOR$_FACILITY;
;   1096  0            %FI
;   1097  0        MACRO
; M 1098  0            DEFSHR [MSG,SEV] =
; M 1099  0                %NAME('SOR$_SHR_',MSG) =
; M 1100  0                        %NAME('SHR$_',MSG) +
;   1101  0                        %NAME('STS$K_',SEV) + SORT$_FACILITY ^ 16 %;
;   1102  0        LITERAL
; P 1103  0            DEFSHR_(
; P 1104  0                BADLOGIC, SEVERE,       ! Internal logic error detected
; P 1105  0                CLOSEDEL, ERROR,        ! Error closing !AS
; P 1106  0                CLOSEIN,  ERROR,        ! Error closing !AS as input
; P 1107  0                CLOSEOUT, ERROR,        ! Error closing !AS as output
; P 1108  0                INSVIRMEM,SEVERE,       ! Insufficient virtual memory
; P 1109  0                OPENIN,   SEVERE,       ! Error opening !AS as input
; P 1110  0                OPENOUT,  SEVERE,       ! Error opening !AS as output
; P 1111  0                READERR,  ERROR,        ! Error reading !AS
; P 1112  0                SYSERROR, SEVERE,       ! System service error
; P 1113  0                TEXT,     WARNING,      ! !AS
;   1114  0                WRITEERR, ERROR);       ! Error writing !AS
;   1115  0
;   1116  0        ! The following macro is used to diagnose an unrecoverable error, instead of
;   1117  0        ! calling SOR$$ERROR directly.
;   1118  0        !
;   1119  0        MACRO
; M 1120  0            SOR$$FATAL(X) = (RETURN SOR$$ERROR(
; M 1121  0                (X) AND NOT STS$M_SEVERITY OR STS$K_SEVERE
;   1122  0                %IF %LENGTH GTR 1 %THEN , %REMAINING %FI)) %;
```

```
!                       T E X T U A L   I N F O R M A T I O N
!
!
!       User-visible text is defined here.  This text may be translated or
!       changed, subject to the restrictions described below.
!
! Default file extension
!
MACRO
        STR_DEF_EXT =           '.DAT' %;


! Default specification file, and default specification file extension
!
MACRO
        STR_DEF_SPECFILE =      'SYS$INPUT' %,
        STR_SPC_EXT =           '.SRT' %;

! These macroes define the external and internal representations of options for
! command line qualifiers.  The first parameter in each pair may be translated;
! the second, however, is used to define internal name for this option, and may
! not be translated.
!
MACRO
        STR_OPT_OUTFMT =                ! outfile/FORMAT=(...)
            'FIXED',            'FIXE',
            'VARIABLE',         'VARI',
            'CONTROLLED',       'CONT',
            'SIZE',             'SIZE',
            'BLOCK_SIZE',       'BLOC' %,

        STR_OPT_INPFMT =                ! inpfile/FORMAT=(...)
            'FILE_SIZE',        'FILE',
            'RECORD_SIZE',      'RECO' %,

        STR_OPT_PROCESS =               ! /PROCESS=...
            'RECORD',           'RECO',
            'TAG',              'TAG',
            'ADDRESS',          'ADDR',
            'INDEX',            'INDE' %,

        STR_OPT_KEY =                   ! /KEY=...
            'ASCENDING',        'ASCE',
            'BINARY',           'BINA',
            'CHARACTER',        'CHAR',
            'DECIMAL',          'DECI',
            'DESCENDING',       'DESC',
            'UNSIGNED',         'UNSI',
            'F_FLOATING',       'F_FL',
            'D_FLOATING',       'D_FL',
            'G_FLOATING',       'G_FL',
            'H_FLOATING',       'H_FL',
            'LEADING_SIGN',     'LEAD',
            'NUMBER',           'NUMB',         ! NUMBER:nn
            'OVERPUNCHED_SIGN', 'OVER',
            'POSITION',         'POSI',         ! POSITION:nn
            'PACKED_DECIMAL',   'PACK',
                'SI',                   'SI',           ! SIZE:nn
                'SIGNED',               'SIGN',
                'SIZE',                 'SIZE',         ! SI:nn
                'SEPARATE_SIGN',        'SEPA',
                'TRAILING_SIGN',        'TRAI',
                'ZONED',                'ZONE' %,

            STR_OPT_COLL =
                'ASCII',                'ASCI',
                'EBCDIC',               'EBCD',
                'DEC_MULTINATIONAL','DEC_' %;


    ! String passed to CLI$GET_VALUE to get the command line.
    !
    !MACRO
    !       STR_CLI_LINE =  '$LINE' %;


    ! FAO string used to output statistics via SYS$PUTMSG.
    !
    ! The following text interacts closely with the code in PRINT_STATS.
    ! The text can, however, be changed (translated) independent of the code, if
    ! the control string still uses the same FAO parameters, and text expands to
    ! no more than 1024 characters (a restriction of the way that the text is
    ! output), and lines are separated by carraige-return/line-feed pairs.
    !
    ! Note that the use of tab character in the text is avoided, since
    ! some terminals may not have tab stops at multiples of eight.
    !
    MACRO
        STR_STATS = %EXPAND %STRING(
            '!/!18* VAX-11 SORT/MERGE !AC Statistics',
            '!/',
            '!/Records read:!12UL',          '!10* Longest record length:!7UL',
            '!/Records sorted:!10UL',        '!10* Input multiblock count:!6UL',
            '!/Records output:!10UL',        '!10* Output multiblock count:!5UL',
            '!/Working set extent:!6UL',     '!10* Input multibuffer count:!5UL',
            '!/Virtual memory:!10UL',        '!10* Output multibuffer count:!4UL',
            '!/Direct I/O:!14UL',            '!10* Number of initial runs:!6UL',
            '!/Buffered I/O:!12UL',          '!10* Maximum merge order:!9UL',
            '!/Page faults:!13UL',           '!10* Number of merge passes:!6UL',
                  '!+!+',
            '!/Sort tree size:!10UL',        '!10* Work file size used:!9UL',
                  '!-!-!-!-',
            '!/Elapsed time: !14%T',         '!7* Elapsed CPU:!6* !14%T',
            '') %;


    ! Logical names to use for work file assignments.
    !       The nth logical name actually used is:
    !       %STRING(STR_LOG_WORKFILE, (n-1)th character of STR_LOG_WORKNUM)
    !
    MACRO
            STR_LOG_WORKFILE =      'SORTWORK' %,
            STR_LOG_WORKNUM =       '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ' %;
```

```
;       1237  0         ! Default file name string to use for the work files.
;       1238  0         !
;       1239  0         MACRO
;       1240  0                 STR_DEF_WORKFILE =      'SYS$SCRATCH:SORTWORK.TMP' %;
```

```
        !                        C L E A N - U P   R O U T I N E S
        !
        !
        !       Clean-up routines are called by SOR$$END_SORT.  To facilitate
        !       information-hiding, the following mechanism is used.  It allows
        !       each sub-system to declare a clean-up routine to clean up its data
        !       structures (so that SOR$END_SORT need not know the format of the
        !       data structures, or even the name of the clean-up routine).
        !
        !       A clean-up routine is declared by:
        !               FORWARD ROUTINE CLEAN_UP;
        !               SOR$$END_ROUTINE_(CLEAN_UP);
        !               ROUTINE CLEAN_UP: CAL_CTXREG NOVALUE = ...
        !
        MACRO   SOR$$END_PSECT_(X) =    %NAME(%EXACTSTRING(30,'_','SOR$RO_CODE'),X) %;
        MACRO   SOR$$END_ROUTINE_(X) =
        PSECT   NODEFAULT=      %EXPAND SOR$$END_PSECT_(2)(PIC,SHARE,NOWRITE,EXECUTE);
        OWN     %NAME('_',X):   PSECT(%EXPAND SOR$$END_PSECT_(2))
                                INITIAL(X-%NAME('_',X)) %;
```

```
SO
Sy

AD
CR
CT
DS
K_
K_
LE
OP
PR
SH
SO
SY
SY

PS
--
.
$A
SO

Ph
--
In
Co
Pa
Sy
Pa
Sy
Ps
Cr
As

Th
75
Th
13
8

Ma
--
_$

16

Th
```

```
;  0      !                          E X E C - M O D E   V A R I A N T
;  0      !
;  0      !       A variant of Sort/Merge is made available to the RDMS group for
;  0      !       use in EXEC mode.  This is gotten by compiling the following
;  0      !       modules with the /VARIANT=1 command qualifier.  Note that the /VARIANT
;  0      !       qualifier will have no effect when compiling the require files.
;  0      !       External references from these modules are named SOR$fac$name.
;  0      !       For example, the following code would be in SORINTERF.
;  0      !
;  0      !       %IF HOSTILE
;  0      !       %THEN
;  0      !           MACRO
;  0      !               LIB$GET_VM = SOR$LIB$GET_VM %,
;  0      !               LIB$FREE_VM = SOR$LIB$FREE_VM %;
;  0      !       %FI
;  0      !
;  0      !       Another variant of Sort/Merge is made available for JRD on ELAN.
;  0      !       This variant is gotten by compiling with /VARIANT=3.
;  0      !       The major distinction between this and the previous is that
;  0      !       the address of the context longword passed to Sort/Merge is passed
;  0      !       to several of the SOR$fac$name system services.
;  0      !
;  0      !       The following modules are needed for these variants:
;  0      !           COM.REQ, SORLIB.REQ, OPCODES.REQ, SORMSG.MSG, SORINTERF.B32,
;  0      !           SORKEYSUB.B32, SORSORT.B32, SORSCRIO.B32, SORFILNAM.B32
;  0      !
;  0      MACRO   HOSTILE = %VARIANT %;
;  0      MACRO   HOSTILE_ELAN = (%VARIANT AND %VARIANT^-1) %;
```

**F

```
;   1288  0     ! End of SORLIB.REQ


;                               Library Statistics
;
;                                                  -------- Symbols --------       Pages       Processing
;       File                                        Total    Loaded   Percent      Mapped      Time
;
;  _$255$DUA28:[SYSLIB]STARLET.L32;1                 9776        20         0         581        00:01.0
;  _$255$DUA28:[SYSLIB]XPORT.L32;1                    590        35         5         252        00:00.6




;                                       COMMAND QUALIFIERS

;       BLISS SRC$:SORLIB/LIS=LIS$:SORLIB/LIB=SRC$:SORLIB

; Run Time:          00:50.4
; Elapsed Time:      02:37.6
; Lines/CPU Min:       1532
; Lexemes/CPU-Min: 95546
; Memory Used:  138 pages
; Library Precompilation Complete
```

0365 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

SORRMSIO
LIS

SORMSG
LIS

SORSCRIO
LIS

SOROUTPUT
LIS

SORLIB
LIS