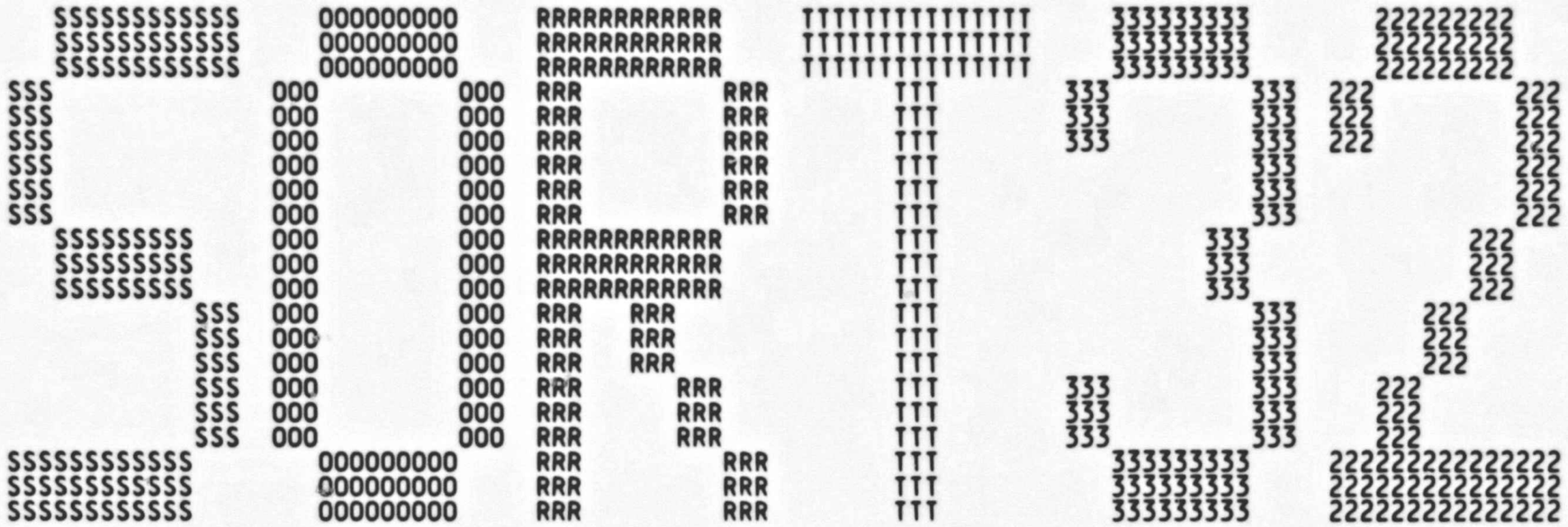
SSSSSSSSSSS 000000000 RRRRRRRRRRR TTTTTTTTTTTTTTT 333333333 222222222
SSSSSSSSSSS 000000000 RRRRRRRRRRR TTTTTTTTTTTTTTT 333333333 222222222
SSSSSSSSSSS 000000000 RRRRRRRRRRR TTTTTTTTTTTTTTT 333333333 222222222
SSS 000 000 RRR RRR TTT 333 333 222 222
SSS 000 000 RRR RRR TTT 333 333 222 222
SSS 000 000 RRR RRR TTT 333 222 222
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 RRRRRRRRRRR TTT 333 222
SSSSSSSSS 000 000 RRRRRRRRRRR TTT 333 222
SSSSSSSSS 000 000 RRRRRRRRRRR 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
SSSSSSSSSSS 000000000 RRR RRR TTT 333333333 222222222222222
SSSSSSSSSSS 000000000 RRR RRR TTT 333333333 222222222222222
SSSSSSSSSSS 000000000 RRR RRR TTT 333333333 222222222222222

FILEID**SORLIB

```
  SSSSSSSS    OOOOOO    RRRRRRRR    LL           IIIIII    BBBBBBBB
  SSSSSSSS    OOOOOO    RRRRRRRR    LL           IIIIII    BBBBBBBB
SS           OO      OO  RR      RR  LL             II      BB      BB
SS           OO      OO  RR      RR  LL             II      BB      BB
SS           OO      OO  RR      RR  LL             II      BB      BB
SS           OO      OO  RR      RR  LL             II      BB      BB
  SSSSSS     OO      OO  RRRRRRRR    LL             II      BBBBBBBB
  SSSSSS     OO      OO  RRRRRRRR    LL             II      BBBBBBBB
        SS   OO      OO  RR   RR      LL             II      BB      BB
        SS   OO      OO  RR   RR      LL             II      BB      BB
        SS   OO      OO  RR     RR    LL             II      BB      BB
        SS   OO      OO  RR     RR    LL             II      BB      BB
SSSSSSSS      OOOOOO    RR       RR  LLLLLLLLLL   IIIIII    BBBBBBBB
SSSSSSSS      OOOOOO    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
```

```
0001  0     ! File: SORLIB.REQ IDENT = 'V04-000'    ! File: SORLIB.REQ Edit: PDG3034
0002  0     !
0003  0     !****************************************************************************
0004  0     !*                                                                          *
0005  0     !*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
0006  0     !*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
0007  0     !*  ALL RIGHTS RESERVED.                                                    *
0008  0     !*                                                                          *
0009  0     !*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
0010  0     !*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0011  0     !*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
0012  0     !*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
0013  0     !*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
0014  0     !*  TRANSFERRED.                                                            *
0015  0     !*                                                                          *
0016  0     !*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
0017  0     !*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
0018  0     !*  CORPORATION.                                                            *
0019  0     !*                                                                          *
0020  0     !*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
0021  0     !*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
0022  0     !*                                                                          *
0023  0     !*                                                                          *
0024  0     !****************************************************************************
0025  0     !
0026  0     !
0027  0     !++
0028  0     !
0029  0     ! FACILITY:     VAX-11 SORT / MERGE
0030  0     !
0031  0     ! ABSTRACT:
0032  0     !
0033  0     !       This is the common definition file for VAX-11 SORT / MERGE.
0034  0     !       All definitions of interest to more than one module are in this file.
0035  0     !       This file is used as a library source.
0036  0     !
0037  0     ! ENVIRONMENT:  VAX/VMS user mode
0038  0     !
0039  0     ! AUTHOR: P. Gilbert, CREATION DATE: 07-Dec-1981
0040  0     !
0041  0     ! MODIFIED BY:
0042  0     !
0043  0     !       T03-015         Original
0044  0     !       T03-016 Add section on pad characters, and correct the extension for
0045  0     !               specification files (.SRT).  PDG 13-Dec-1982
0046  0     !       T03-017 Add WF_NAMES, CFT indices of work file names.  PDG 26-Dec-1982
0047  0     !       T03-018 Added DDB_CHAN.  PDG 28-Dec-1982
0048  0     !       T03-019 Make work-file description blocks (WFBs) distinct from DDBs.
0049  0     !               PDG 31-Dec-1982
0050  0     !       T03-020 Add clean-up routines.  PDG 4-Jan-1983
0051  0     !       T03-021 Add WFB_DEV.  PDG 6-Jan-1983
0052  0     !       T03-022 Removed PT/ST_ADR; added BS_DECM, WRK_SIZ.  PDG 26-Jan-1983
0053  0     !       T03-023 Change STAT_K_WRK_USE to STAT_K_WRK_ACQ.  Added WFB_USE field.
0054  0     !               Added COM_MRG_STREAM for stable merges.  PDG 27-Jan-1983
0055  0     !       T03-024 Remove section on pad characters.  Add COM_PAD.  PDG 8-Feb-1983
0056  0     !       T03-025 Remove unreferenced fields.  Change linkage declarations so
0057  0     !               register information is available to SOR$$KEY_SUB at run time.
```

```
;    0058  0     !                   Define the macro SOR$$FATAL.  PDG 16-Mar-1983
;    0059  0     !           T03-026 Give the SOR$RO_CODE_0 PSECTs the EXE attr.  PDG 7-Apr-1983
;    0060  0     !           T03-027 Information hiding of WFB structure.  PDG 12-Apr-1983
;    0061  0     !           T03-028 Move definitions of fields specific to scratch-i/o to SORSCRIO
;    0062  0     !                   from this module.  PDG 18-Apr-1983
;    0063  0     !           T03-029 Reduce COM_K_SCRATCH.  PDG 22-Apr-1983
;    0064  0     !           T03-030 Correct size of COM_WF_NAMES.  PDG 17-May-1983
;    0065  0     !           T03-031 Add COM_ARCHFLAG.  PDG 31-Jan-1984
;    0066  0     !           T03-032 Add COL[_BLOCK stuff.  PDG 22-Feb-1984
;    0067  0     !           T03-033 Change TON_K_BUFSIZE to 5 blocks for VAXELN.
;    0068  0     !                   Add support for VAXELN.  Jeff East 3/13/84
;    0069  0     !           T03-034 Change COM_RHB to COM_RHB_INP and COM_RHB_OUT.
;    0070  0     !                   This is to avoid problems with merge, where an incoming
;    0071  0     !                   record overwrites the VFC area for the outgoing record.
;    0072  0     !                   PDG 24-Jul-1984
;    0073  0     !--
;    0074  0
;    0075  0     LIBRARY 'SYS$LIBRARY:STARLET';
;    0076  0     LIBRARY 'SYS$LIBRARY:XPORT';
```

```
;    0077  0     !                          X P O R T
;    0078  0     !
;    0079  0     !       The use of XPORT causes some problems, most notably with alignment,
;    0080  0     !       and the default sign extension.  The following macros are used.
;    0081  0     !
;    0082  0     MACRO
;    0083  0         XBYTE =     $ALIGN(BYTE) %EXPAND $BITS(8) %,
;    0084  0         XWORD =     $ALIGN(WORD) %EXPAND $BITS(16) %,
;    0085  0         XLONG =     $ALIGN(FULLWORD) %EXPAND $BITS(32) %,
;    0086  0         XDESC =     $ALIGN(FULLWORD) $SUB_BLOCK(2) %,
;    0087  0         XADDR =     $ALIGN(FULLWORD) $ADDRESS %;
;    0088  0     $SHOW(FIELDS)
```

```
;    0089  0      !                          POSITION AND SIZE MACROS
;    0090  0      !
;    0091  0      !
;    0092  0      MACRO
;    0093  0      !       Macros used for field references
;    0094  0      !
;    0095  0      !       A_=                 0,0,0 %,
;    0096  0      !       L_=                 0,32,0 %,
;    0097  0      !       BASE_=            0,0, 0,0 %,
;    0098  0      !
;    0099  0      !
;    0100  0      !       Macros to construct a bit mask from a standard four-component field
;    0101  0      !       definition (offset, position, size, extension).  The result has set
;    0102  0      !       bits in those positions that belong to the field.  A list of field
;    0103  0      !       definitions can be specified.
;    0104  0      !
;    0105  0      !       Example:
;    0106  0      !
;    0107  0      !             MACRO
;    0108  0      !                 A=0,2,4,0%,
;    0109  0      !                 B=0,9,1,0%;
;    0110  0      !
;    0111  0      !             MASK_(A,B) is equal to %B'1000111100'
;    0112  0      !
;M   0113  0      !       XMASK_[O,P,S,E]=
;    0114  0      !           (1^((P)+(S))) - (1^(P)) %,
;    0115  0      !
;    0116  0      !
;M   0117  0      !       MASK_[]=
;    0118  0      !           (0 OR XMASK_(%REMAINING)) %,
;    0119  0      !
;    0120  0      !
;    0121  0      !       Macros to align a specified value at the bit position specified by a
;    0122  0      !       standard four-component field definition (offset, position, size,
;    0123  0      !       extension).  A list of values and field definitions can be specified.
;    0124  0      !
;    0125  0      !       Example:
;    0126  0      !
;    0127  0      !             MACRO
;    0128  0      !                 A=0,2,4,0%,
;    0129  0      !                 B=0,9,1,0%;
;    0130  0      !
;    0131  0      !             ALIGN_(7,A,1,B) is equal to 7^2 OR 1^9
;    0132  0      !
;M   0133  0      !       XALIGN_[V,O,P,S,E]=
;    0134  0      !           ((V)^(P)) %,
;    0135  0      !
;    0136  0      !
;M   0137  0      !       ALIGN_[]=
;    0138  0      !           (0 OR XALIGN_(%REMAINING)) %;
```

```
;    0139  0      !                       G E N E R A L
;    0140  0      !
;    0141  0      !
;    0142  0      LITERAL
;    0143  0              TRUE=           1,
;    0144  0              FALSE=          0;
;    0145  0
;    0146  0
;    0147  0      MACRO
;    0148  0              ELIF=           ELSE IF %;
;    0149  0
;    0150  0
;    0151  0      MACRO
;    0152  0              ! Macro to round a value to the next higher multiple of a number.
;    0153  0              !
;    0154  0              ! The first parameter is the number which is to be rounded.
;    0155  0              ! The second parameter is the multiple up to which we round.
;    0156  0              !       If omitted, the default for the second parameter is %UPVAL
;    0157  0              !       The second parameter should be a literal, and a power of 2.
;    0158  0              !
;M   0159  0              ROUND_(A,B) =
;M   0160  0                  %IF %NULL(B)
;M   0161  0                  %THEN (((A) + %UPVAL-1) AND NOT (%UPVAL-1))
;M   0162  0                  %ELSE (((A)+  (B)   -1) AND NOT ((B)   -1))
;    0163  0                  %FI %;
;    0164  0
;    0165  0
;    0166  0      MACRO
;    0167  0              ! Macro to calculate floor(log2(constant))
;    0168  0              !
;M   0169  0              LN2_(A)=
;    0170  0                  (%NBITSU(A)-1) %;
;    0171  0
;    0172  0      MACRO
;    0173  0              ! Macro to signal an internal consistency check.
;    0174  0              !
;M   0175  0              BUGCHECK(A)=
;M   0176  0                  BEGIN BUILTIN CHMU;
;M   0177  0                  CHMU(%REF(0));
;M   0178  0                  0
;    0179  0                  END %;
;    0180  0
;    0181  0      MACRO
;    0182  0              ! Macro to establish a condition handler.
;    0183  0              !
;M   0184  0              ESTABLISH_(X) =
;M   0185  0                  BEGIN BUILTIN FP;
;M   0186  0                  .FP = X;
;    0187  0                  END %;
;    0188  0
;    0189  0      MACRO
;    0190  0              ! Macro to produce a list of names
;    0191  0              !
;    0192  0              PREFIX_(A)[B] = %NAME(A,B) %;
```

```
;    0193  0      MACRO
;    0194  0
;    0195  0              ! Macros to determine if the value of an expression is one of a set of
;    0196  0              ! specified small-integer values.  These macros can be used only if the
;    0197  0              ! following conditions are met:
;    0198  0              !
;    0199  0              !       The value to be tested is in the range 0 through 127.
;    0200  0              !
;    0201  0              !       The values to be tested for are all in the range 0 through 31.
;    0202  0              !
;    0203  0              ! Example:
;    0204  0              !
;    0205  0              !       IF ONEOF_(.X, BMSK_(1,3,5)) ...
;    0206  0              !
;    0207  0              ! The code generated is much more efficient than a series of comparisons
;    0208  0              ! (provided that the parameters of BMSK_ are all compile-time constant).
;    0209  0              !
;  M 0210  0              XBMSK_[A]=
;  M 0211  0                  %IF (A) GTRU 31 %THEN %WARN('ONEOF won''t work') %FI
;    0212  0                  (1 ^ (31 - (A))) %,
;    0213  0
;    0214  0
;  M 0215  0              BMSK_[]=
;    0216  0                  (0 OR XBMSK_(%REMAINING)) %,
;    0217  0
;    0218  0
;  M 0219  0              ONEOF_(A,B)=
;    0220  0                  ((?(B) ^ (A)) LSS 0) %;
;    0221  0
;    0222  0      MACRO
;    0223  0
;    0224  0              ! Macros to create initialized, read-only bit-vectors.
;    0225  0              ! The first parameter to BV_ is the largest element which will be
;    0226  0              ! accessed in the bit-vector.
;    0227  0              !
;    0228  0              ! For example:
;    0229  0              !
;    0230  0              ! OWN   PRIMES: BV_( 51, 2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,51 );
;    0231  0              ! IF .PRIMES[.I]
;    0232  0              ! THEN  %( I is Prime )%
;    0233  0              ! ELSE  %( I is Composite )%
;    0234  0              !
;    0235  0
;    0236  0              BV_1_[A] = [A] = 1 %,
;    0237  0
;  M 0238  0              BV_(M) = BITVECTOR[M+1]
;    0239  0                  PSECT(SOR$RO_CODE) PRESET( BV_1_(%REMAINING) ) %;
;    0240  0
;    0241  0      MACRO
;    0242  0              ! Macros to distinguish whether the value of an expression is amoung
;    0243  0              ! one set of values, or another set of values, based on a single bit.
;    0244  0              ! An error diagnostic is issued if a single bit will not suffice.
;    0245  0              !
;  M 0246  0              DIST_(X,Y,Z) =
;  M 0247  0                  BEGIN
;  M 0248  0                  LITERAL
;  M 0249  0                      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) %;
```

```
;  0261  0     !                       DEBUGGING CODE
;  0262  0     !
;  0263  0     !
;  0264  0     !       This section defines macros to aid in writing debugging code.
;  0265  0     !
;  0266  0     !       The %VARIANT switch is used to conditionally include compiler debugging
;  0267  0     !       code.  When %VARIANT is true, debugging code is included.  When it is
;  0268  0     !       false, debugging code is omitted.  The macro DEB_CODE is provided to
;  0269  0     !       bracket debugging code that is to be unconditionally executed.
;  0270  0     !
;  0271  0     !       In addition, the global variable "SOR$$D" in the COMENTRY module can be
;  0272  0     !       used to obtain conditional execution of debugging code.  This variable
;  0273  0     !       is initialized to zero, but may be altered during the initial DEBUG
;  0274  0     !       dialogue, before the compiler is started:
;  0275  0     !
;  0276  0     !               DBG>D SOR$$D=%X'D6003FFF'       (for example)
;  0277  0     !               DBG>D SOR$$D=1                  (for example)
;  0278  0     !               DBG>G
;  0279  0     !
;  0280  0     !       The bits in the variable "SOR$$D" are allocated as follows:
;  0281  0     !                 0       %X'00000001'    Dump run information
;  0282  0     !                 1       %X'00000002'    Dump incremental statistics
;  0283  0     !                 2       %X'00000004'    Dump allocation information
;  0284  0     !                 .       .
;  0285  0     !                 .       .               .
;  0286  0     !                30       %X'40000000'    Unassigned
;  0287  0     !                31       %X'80000000'    Unassigned
;  0288  0     !
;  0289  0     !       The macro DEB_SWITCH is provided to bracket conditionally executed
;  0290  0     !       debugging code.
;  0291  0     !
;  0292  0     MACRO
;  0293  0     !
;  0294  0     !       Macro to bracket unconditional debugging code.  The parameter is an
;  0295  0     !       expression that will be compiled if %VARIANT is true.
;  0296  0     !
M 0297  0     DEB_CODE(A)=
M 0298  0         %IF %VARIANT
M 0299  0         %THEN
M 0300  0             A
          0         %FI %,
;  0301  0     !
;  0302  0     !
;  0303  0     !       Macro to bracket conditional debugging code.  The first parameter is
;  0304  0     !       a bit number in the variable SOR$$D, and the second parameter is an
;  0305  0     !       expression that will be evaluated if that bit is set.  The entire
;  0306  0     !       expansion is compiled only if %VARIANT is true.
;  0307  0     !
M 0308  0     DEB_SWITCH(A,B)=
M 0309  0         %IF %VARIANT
M 0310  0         %THEN
M 0311  0             BEGIN EXTERNAL SOR$$D;
M 0312  0             IF .SOR$$D<A,1> THEN B;
M 0313  0             END
          0         %FI %,
;  0314  0     !
;  0315  0     !
;  0316  0     !
;  0317  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 %;
```

```
;    0324  0      !                       MAXIMUM VALUES
;    0325  0      !
;    0326  0      !
;    0327  0      LITERAL
;    0328  0              MAX_KEYS=        255,              ! Maximum number of sort keys allowed
;    0329  0              MAX_FILES=       10,               ! Maximum number of input files.
;    0330  0              MIN_WORK_FILES=  1,                ! Minimum number of work files
;    0331  0              DEF_WORK_FILES=  2,                ! Default number of work files
;    0332  0              MAX_WORK_FILES=  10,               ! Maximum number of work files
;    0333  0              MAX_MERGE_ORDER=10,                ! Maximum merge order
;    0334  0              MAX_SPC_LINE=    132,              ! Maximum length of spec file line
;    0335  0
;    0336  0              MAX_SEQ_RECLEN=  32767,            ! Maximum sequential file record length
;    0337  0              MAX_REL_RECLEN=  16384,            ! Maximum relative file record length
;    0338  0              MAX_IDX_RECLEN=  16384,            ! Maximum indexed file record length
;    0339  0              MAX_ISAMKEYLEN=  255,              ! Maximum index key data item length
;    0340  0              MAX_REFSIZE=     65535,            ! Maximum length of a referenceable data-item
;    0341  0              MAX_PSECTSIZE=   2147483647;       ! Maximum length of a PSECT
;    0342  0      LITERAL
;    0343  0              MIN_MBC=         7,                ! Minimum MBC count
;    0344  0              MAX_MBC=         16,               ! Maximum MBC count (for RP06)
;    0345  0              MIN_MBF=         0,                ! Minimum MBF count
;    0346  0              MAX_MBF=         2;                ! Maximum MBF count
;    0347  0      LITERAL
;    0348  0              DEF_FILE_ALLOC= 128*3,             ! Default file allocation
;    0349  0              DEF_TRM_ALLOC=  16;                ! Default allocation for terminals
;    0350  0      LITERAL
;    0351  0              COM_K_BPERPAGE= 512,               ! Bytes per page
;    0352  0              COM_K_BPERBLOCK= 512;              ! Bytes per disk block
;    0353  0      LITERAL
;    0354  0              ! Define a literal for the amount of work space to allocate
;    0355  0              ! for specification text, and another for the amount of work space
;    0356  0              ! to allocate if we only need to process a collating sequence.
;    0357  0              !
;    0358  0              WRK_K_ALLOC=    128 * COM_K_BPERPAGE,   ! Allocation for work area
;    0359  0              WRK_K_COLLATE=  6 * 256;                ! Alloc to process collating sequence
```

```
;  0360  0     !                       INTERFACE VALUES
;  0361  0     !
;  0362  0     !
;  0363  0     LITERAL
;  0364  0             ! Datatype values for use in the key definition buffer (KEY_BUFFER).
;  0365  0             !
;  0366  0             ! These are also used to define the global literals SOR$GK_xxx_KEY.
;  0367  0             !
;  0368  0             ! These are used only for compatability purposes.
;  0369  0             !
;  0370  0             KEY_K_CHAR=     1,              ! Character data
;  0371  0             KEY_K_BIN=      2,              ! Signed binary data
;  0372  0             KEY_K_ZONE=     3,              ! Zoned decimal
;  0373  0             KEY_K_PACK=     4,              ! Packed decimal
;  0374  0             KEY_K_USB=      5,              ! Unsigned binary
;  0375  0             KEY_K_DLO=      6,              ! Decimal leading overpunch
;  0376  0             KEY_K_DLS=      7,              ! Decimal leading separate
;  0377  0             KEY_K_DTO=      8,              ! Decimal trailing overpunch
;  0378  0             KEY_K_DTS=      9,              ! Decimal trailing separate
;  0379  0             KEY_K_FLT=      10,             ! Floating
;  0380  0             KEY_K_FLTD=     11,             ! D_floating
;  0381  0             KEY_K_FLTG=     12,             ! G_floating
;  0382  0             KEY_K_FLTH=     13,             ! H_floating
;  0383  0             KEY_K_MAX=      13;             ! Maximum
;  0384  0
;  0385  0
;  0386  0     LITERAL
;  0387  0             ! Values for sort types, passed to SOR$INIT_SORT.
;  0388  0             !
;  0389  0             ! These are also used to define the global literals SOR$GK_xxx.
;  0390  0             !
;  0391  0             TYP_K_RECORD=   1,              ! Record sort
;  0392  0             TYP_K_TAG=      2,              ! Tag sort
;  0393  0             TYP_K_INDEX=    3,              ! Index sort
;  0394  0             TYP_K_ADDRESS=  4,              ! Address sort
;  0395  0             TYP_K_MAX=      4;              ! Maximum sort type
;  0396  0
;  0397  0     MACRO
;  0398  0             ! Options flags, passed to SOR$INIT_SORT and SOR$INIT_MERGE.
;  0399  0             !
;  0400  0             ! These are used to define the global literals SOR$V_xxx and SOR$M_xxx.
;  0401  0             !
;  0402  0             OPT_STABLE=     0, 0, 1, 0 %,   ! Stable sort
;  0403  0             OPT_EBCDIC=     0, 1, 1, 0 %,   ! EBCDIC collating sequence
;  0404  0             OPT_MULTI=      0, 2, 1, 0 %,   ! MULTINATIONAL collating sequence
;  0405  0             OPT_NOSIGNAL=   0, 3, 1, 0 %,   ! Don't signal errors
;  0406  0             OPT_SEQ_CHECK=  0, 4, 1, 0 %,   ! Sequence check on merge input
;  0407  0     !       unused=         0, 5, 1, 0 %,
;  0408  0             OPT_NODUPS=     0, 6, 1, 0 %,   ! Delete records with duplicate keys
;  0409  0             OPT_FIXED=      0, 7, 1, 0 %,   ! Records are fixed length (NYUsed)
;  0410  0     !       OPT_LOCATE=     0, 8, 1, 0 %,   ! Use locate mode with RETURN_REC
;  0411  0             OPT_LOAD_FILL=  0, 9, 1, 0 %;   ! Use LOAD_FILL on output file
;  0412  0
;  0413  0     LITERAL
;  0414  0             ! Values to index the sort statistics
;  0415  0             !
;  0416  0             ! These are also used to define the global literals SOR$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:
;  L 0519  0          COM_SIGNAL=       [$BIT],         ! Sort/merge should signal errors
;  %PRINT:                                [22,25,1,0]     (+%X'5B')
;  L 0520  0          COM_NOCHKPNT=     [$BIT],         ! Checkpointing should not be done
;  %PRINT:                                [22,26,1,0]     (+%X'5B')
;  L 0521  0          COM_LOAD_FILL=    [$BIT],         ! Use load-fill on indexed files
;  %PRINT:                                [22,27,1,0]     (+%X'5B')
;  L 0522  0          COM_NODUPS=       [$BIT],         ! Delete records with duplicate keys
;  %PRINT:                                [22,28,1,0]     (+%X'5B')
;  L 0523  0          U_=               [$BIT],         ! Use locate mode with RETURN_REC
;  %PRINT:                                [22,29,1,0]     (+%X'5B')
;     0524  0          !
;     0525  0          ! Control flow flags
;     0526  0          !
;  L 0527  0          COM_FLO_SORT=     [$BIT],         ! May call Sort-Merge
;  %PRINT:                                [22,30,1,0]     (+%X'5B')
;  L 0528  0          COM_FLO_NOINIT=   [$BIT],         ! May not call Pass-Files, Init-Sort or Init-Merge
;  %PRINT:                                [22,31,1,0]     (+%X'5B')
;  L 0529  0          COM_FLO_RELEASE=  [$BIT],         ! May call Release-Rec
;  %PRINT:                                [23,0,1,0]      (+%X'5C')
;  L 0530  0          COM_FLO_RETURN=   [$BIT],         ! May call Return-Rec or End-Sort
;  %PRINT:                                [23,1,1,0]      (+%X'5C')
;  L 0531  0          COM_FLO_DOMERGE=  [$BIT],         ! May call Do-Merge
;  %PRINT:                                [23,2,1,0]      (+%X'5C')
;  L 0532  0          COM_FLO_ABORT=    [$BIT],         ! May only call End-Sort
;  %PRINT:                                [23,3,1,0]      (+%X'5C')
;     0533  0          !
;     0534  0          ! Flags to amend for V3 compatability hacks
;     0535  0          !
;  L 0536  0          COM_HACK_2ARGS=   [$BIT],         ! Pass only 2 args to callback routines
;  %PRINT:                                [23,4,1,0]      (+%X'5C')
;  L 0537  0          COM_HACK_STRIP=   [$BIT],         ! Strip the keys
;  %PRINT:                                [23,5,1,0]      (+%X'5C')
;     0538  0          !
;     0539  0          ! Merge-specific fields
;     0540  0          !
;     0541  0          ! Note that COM_MRG_ORDER is non-zero iff this is a merge
;     0542  0          !
;  L 0543  0          COM_MERGE=        [$BIT],         ! Indicates a merge (not a sort)
;  %PRINT:                                [23,6,1,0]      (+%X'5C')
;  L 0544  0          COM_MRG_ORDER=    [XBYTE],        ! Order of the merge
;  %PRINT:                                [23,7,1,0]      (+%X'5C')
;     0545  0          !                                  (+%X'5D')
;     0546  0          ! Spec text processing stuff
;     0547  0          !
;  L 0548  0          COM_SPEC_TKS=     [XWORD],        ! Size of keys portion of internal node
;  %PRINT:                                [23,8,8,0]      (+%X'5D')
;     0549  0          !                                  (+%X'5E')
;     0550  0          ! Merge-specific fields
;     0551  0          !
;  L 0552  0          COM_MRG_INPUT=    [XADDR],        ! User-written merge input routine
;  %PRINT:                                [23,16,16,0]    (+%X'5E')
;  L 0553  0          COM_MRG_STREAM=   [XLONG],        ! Stream number for stable merges
;  %PRINT:                                [24,0,32,0]     (+%X'60')
;     0554  0          !                                  (+%X'64')
;     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             !
```

```
;    0630  0              ! Locking information
;    0631  0              !
;    0632  0      !       COM_LOCKED=     [XADDR],        ! List of locked code sections
;    0633  0              !
;    0634  0              ! Specification file stuff
;    0635  0              !
;L 0636  0                COM_SPC_TXT=    [XDESC],        ! Dymanic string for spec file text
;%PRINT:                                    [61,0,0,0]   (+%X'F4')
;    0637  0              !
;    0638  0              ! Specification file stuff
;    0639  0              !
;L 0640  0                COM_RDT_SIZ=    [XBYTE],
;%PRINT:                                    [63,0,8,0]   (+%X'FC')
;L 0641  0                COM_KFT_SIZ=    [XBYTE],
;%PRINT:                                    [63,8,8,0]   (+%X'FD')
;L 0642  0                COM_CFT_SIZ=    [XBYTE],
;%PRINT:                                    [63,16,8,0]   (+%X'FE')
;L 0643  0                COM_FDT_SIZ=    [XBYTE],
;%PRINT:                                    [63,24,8,0]   (+%X'FF')
;L 0644  0                COM_TDT_SIZ=    [XBYTE],
;%PRINT:                                    [64,0,8,0]   (+%X'100')
;L 0645  0                COM_PAD=        [XBYTE],        ! Pad character
;%PRINT:                                    [64,8,8,0]   (+%X'101')
;L 0646  0                U_=             [$BITS(16)],
;%PRINT:                                    [64,16,16,0]   (+%X'102')
;L 0647  0                COM_RDT_ADR=    [XADDR],        ! Record definition table
;%PRINT:                                    [65,0,32,0]   (+%X'104')
;L 0648  0                COM_KFT_ADR=    [XADDR],        ! Key/data field table
;%PRINT:                                    [66,0,32,0]   (+%X'108')
;L 0649  0                COM_CFT_ADR=    [XADDR],        ! Constant field table
;%PRINT:                                    [67,0,32,0]   (+%X'10C')
;L 0650  0                COM_FDT_ADR=    [XADDR],        ! Field definition table
;%PRINT:                                    [68,0,32,0]   (+%X'110')
;L 0651  0                COM_TDT_ADR=    [XADDR],        ! Test definition table
;%PRINT:                                    [69,0,32,0]   (+%X'114')
;L 0652  0                COM_CONST_AREA= [XADDR],        ! Constant area (address)
;%PRINT:                                    [70,0,32,0]   (+%X'118')
;L 0653  0                COM_PTAB=       [XADDR],        ! Pointer to 256-byte table
;%PRINT:                                    [71,0,32,0]   (+%X'11C')
;L 0654  0                U_=             [XADDR],
;%PRINT:                                    [72,0,32,0]   (+%X'120')
;L 0655  0                COM_WRK_SIZ=    [XLONG],        ! Length of work area
;%PRINT:                                    [73,0,32,0]   (+%X'124')
;L 0656  0                COM_WRK_ADR=    [XADDR],        ! Address of work area
;%PRINT:                                    [74,0,32,0]   (+%X'128')
;L 0657  0                COM_WRK_END=    [XADDR],        ! Address past end of work area
;%PRINT:                                    [75,0,32,0]   (+%X'12C')
;    0658  0              !
;    0659  0              ! Other stuff
;    0660  0              !
;L 0661  0                COM_WORST=      [XLONG],        ! Worst error we've ever seen
;%PRINT:                                    [76,0,32,0]   (+%X'130')
;    0662  0              COM_WF_NAMES=   ! Counted list of indices into CFT of work file names
;L 0663  0                                [$BYTES(1+MAX_WORK_FILES)],
;%PRINT:                                    [77,0,0,0]   (+%X'134')
;    0664  0              $ALIGN(FULLWORD)
;L 0665  0                COM_CDD=        [$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__;
```

```
;  0684  0     !                       RECORD FORMATS
;  0685  0     !
;  0686  0     !       This section describes the various record formats that are used
;  0687  0     !       throughout Sort/Merge.
;  0688  0     !
;  0689  0     !       INPUT RECORD FORMAT:
;  0690  0     !
;  0691  0     !               VAR (a word) is present only for variable length records
;  0692  0     !               VFC is present only for VFC files
;  0693  0     !               DATA is always present
;  0694  0     !
;  0695  0     !       INTERNAL RECORD FORMAT:
;  0696  0     !
;  0697  0     !           FORM KEY VAR VFC DATA STAB          ! Record sort
;  0698  0     !           FORM KEY RFA FILE STAB              ! Tag, address, index
;  0699  0     !
;  0700  0     !               VAR (a word) is present only for variable length records
;  0701  0     !               VFC is present only for VFC files
;  0702  0     !               KEY is present for keys or converted keys
;  0703  0     !               FORM (a byte) is present only for multiple record formats
;  0704  0     !               FILE (a byte) is present only for multi-file non-record sorts
;  0705  0     !               STAB (a longword) is present only for stable sorts
;  0706  0     !               RFA (RAB$S_RFA bytes) is present for non-record sorts
;  0707  0     !
;  0708  0     !       OUTPUT RECORD FORMAT:
;  0709  0     !
;  0710  0     !           VAR VFC DATA                        ! Record, tag sort
;  0711  0     !           RFA FILE                            ! Address sort
;  0712  0     !           RFA FILE OKEY STAB                  ! Index sort
;  0713  0     !
;  0714  0     !               VAR (a word) is present only for variable length records
;  0715  0     !               VFC is present only for VFC files
;  0716  0     !               FILE (a byte) is present only for multi-file non-record sorts
;  0717  0     !               OKEY is the unconverted keys
;  0718  0     !               STAB (a longword) is present only for stable index sorts
;  0719  0     !
;  0720  0     !
;  0721  0     ! Assertions can be made on the following literals to determine the relative
;  0722  0     ! ordering of fields within a record.
;  0723  0     !
;  0724  0     LITERAL
;  0725  0             COM_ORD_RFA     = 0,    ! RFA field
;  0726  0             COM_ORD_FILE    = 1,    ! File number field
;  0727  0             COM_ORD_FORM    = 2,    ! Format field
;  0728  0             COM_ORD_OKEY    = 3,    ! Original keys (for index sorts)
;  0729  0             COM_ORD_STAB    = 4,    ! Stable longword field
;  0730  0             COM_ORD_KEY     = 5,    ! Key or converted key field
;  0731  0             COM_ORD_VAR     = 6,    ! Length field
;  0732  0             COM_ORD_VFC     = 7,    ! VFC field
;  0733  0             COM_ORD_DATA    = 8,    ! Data field
;  0734  0             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;
```

```
;    0765  0     !                          L I N K A G E S
;    0766  0     !
;    0767  0     !
;    0768  0     !       Several internal routines use JSB linkages to improve performance.
;    0769  0     !       Common linkages are defined here.  Linkages to external routines
;    0770  0     !       are defined as LNK_routine_name.
;    0771  0     !
;    0772  0     LITERAL
;    0773  0             COM_REG_SRC1 =  9,
;    0774  0             COM_REG_SRC2 =  10,
;    0775  0             COM_REG_CTX =   11;
;    0776  0     MACRO
;    0777  0             %PRESERVE(X)    = %NAME(X,'_PR') %,
;    0778  0             %NOPRESERVE(X)  = %NAME(X,'_NP') %,
;    0779  0             %NOTUSED(X)     = %NAME(X,'_NU') %,
;    0780  0             XREGMASK_[P]    = 1^P %,
;    0781  0             REGMASK_[]      = 0 OR XREGMASK_(%REMAINING) %;
;    0782  0     KEYWORDMACRO
;  M 0783  0             JSB_DEFN_(NAM,PM,GL,PR,NP,NU) =
;  M 0784  0             LITERAL
;  M 0785  0                 %PRESERVE(NAM)      = REGMASK_(%REMOVE(PR)) + 0,
;  M 0786  0                 %NOPRESERVE(NAM)    = REGMASK_(%REMOVE(NP)) + 0,
;  M 0787  0                 %NOTUSED(NAM)       = REGMASK_(%REMOVE(NU)) + 0;
;  M 0788  0             LINKAGE NAM = JSB(%REMOVE(PM)):
;  M 0789  0                 %IF NOT %NULL(GL) %THEN GLOBAL(%REMOVE(GL)) %FI
;  M 0790  0                 %IF NOT %NULL(PR) %THEN PRESERVE(%REMOVE(PR)) %FI
;  M 0791  0                 %IF NOT %NULL(NP) %THEN NOPRESERVE(%REMOVE(NP)) %FI
;  M 0792  0                 %IF NOT %NULL(NU) %THEN NOTUSED(%REMOVE(NU)) %FI
;    0793  0             %;
;    0794  0
;  P 0795  0     JSB_DEFN_(
;  P 0796  0             NAM = JSB_INPUT,                    ! For COM_INPUT
;  P 0797  0             PM = <REGISTER=COM_REG_SRC1,REGISTER=COM_REG_SRC2>,
;  P 0798  0             PR = <COM_REG_SRC2>,
;  P 0799  0             NP = <0,1,2,3,4,5,6,COM_REG_SRC1>,      ! R6 holds the variable length
;  P 0800  0             NU = <7,8>,
;    0801  0             GL = <CTX=COM_REG_CTX> );
;    0802  0
;  P 0803  0     JSB_DEFN_(
;  P 0804  0             NAM = JSB_NEWRUN,                   ! For COM_NEWRUN
;  P 0805  0             NU = <4,5,6,7,8,10>,
;  P 0806  0             NP = <0,1>,
;  P 0807  0             PR = <2,3,9>,
;    0808  0             GL = <CTX=COM_REG_CTX> );
;    0809  0
;  P 0810  0     JSB_DEFN_(
;  P 0811  0             NAM = JSB_COMPARE,                  ! For COM_COMPARE
;  P 0812  0             PM = <REGISTER=COM_REG_SRC1,REGISTER=COM_REG_SRC2>,
;  P 0813  0             PR = <COM_REG_SRC1,COM_REG_SRC2>,
;  P 0814  0             NP = <0,1,2,3,4,5>,
;  P 0815  0             NU = <6,7,8>,                               ! Really???
;    0816  0             GL = <CTX=COM_REG_CTX> );
;    0817  0
;  P 0818  0     JSB_DEFN_(
;  P 0819  0             NAM = JSB_OUTPUT,                   ! For COM_OUTPUT
;  P 0820  0             PM = <REGISTER=COM_REG_SRC2>,
;  P 0821  0             PR = <COM_REG_SRC2>,
```

```
;  P 0822  0              NU = <7,8,9>,
;  P 0823  0              NP = <0,1,2,3,4,5,6>,                   ! R6 needed???
;    0824  0              GL = <CTX=COM_REG_CTX> );
;    0825  0
;  P 0826  0      JSB_DEFN (
;  P 0827  0              NAM = JSB_EQUAL,                        ! For COM_EQUAL
;  P 0828  0              PM = <REGISTER=COM_REG_SRC1,REGISTER=COM_REG_SRC2>,
;  P 0829  0              PR = <COM_REG_SRC1,COM_REG_SRC2>,
;  P 0830  0              NP = <0,15>,
;  P 0831  0              NU = <2,3,4,5,6,7,8>,
;    0832  0              GL = <CTX=COM_REG_CTX> );
;    0833  0
;  P 0834  0      JSB_DEFN (
;  P 0835  0              NAM = JSB_LENADR,                       ! For COM_LENADR
;  P 0836  0              PM = <REGISTER=COM_REG_SRC2;REGISTER=0,REGISTER=1>,
;  P 0837  0              PR = <COM_REG_SRC2>,
;  P 0838  0              NP = <0,15>,
;  P 0839  0              NU = <2,3,4,5,6,7,8,9>,
;    0840  0              GL = <CTX=COM_REG_CTX> );
;    0841  0
;  P 0842  0      JSB_DEFN (
;  P 0843  0              NAM = JSB_INSERT,                       ! For SOR$$TREE_INSERT
;  P 0844  0              PM = <STANDARD>,                        ! Can we use registers???
;  P 0845  0              PR = <7,8>,
;  P 0846  0              NP = <0,1,2,3,4,5,6,COM_REG_SRC1,COM_REG_SRC2>,
;    0847  0              GL = <CTX=COM_REG_CTX> );
;    0848  0
;  P 0849  0      JSB_DEFN (
;  P 0850  0              NAM = JSB_READINS,                      ! For READ_INSERT
;  P 0851  0              PM = <REGISTER=6,REGISTER=8>,
;  P 0852  0              PR = <7,8>,
;  P 0853  0              NP = <0,1,2,3,4,5,6,9,10>,
;    0854  0              GL = <CTX=COM_REG_CTX> );
;    0855  0
;  P 0856  0      JSB_DEFN (
;  P 0857  0              NAM = JSB_EXTRACT,                      ! For SOR$$TREE_EXTRACT
;  P 0858  0              PM = <STANDARD>,                        ! Can we use registers???
;  P 0859  0              PR = <7,8>,
;  P 0860  0              NP = <0,1,2,3,4,5,6,COM_REG_SRC1,COM_REG_SRC2>,
;    0861  0              GL = <CTX=COM_REG_CTX> );
;    0862  0
;    0863  0      LINKAGE
;    0864  0              CAL_ACCESS =    CALL (  STANDARD;               ! For SOR$$RFA_ACCESS
;    0865  0                                      REGISTER=0,
;    0866  0                                      REGISTER=1):
;    0867  0                                      GLOBAL(CTX=COM_REG_CTX);
;    0868  0      LINKAGE
;    0869  0              CAL_CTXREG =    CALL:   GLOBAL(CTX=COM_REG_CTX);
```

```
;    0870  0      !                          T U N I N G   P A R A M E T E R S
;    0871  0      !
;    0872  0      !
;    0873  0      !       These values are used to tune the sort.
;    0874  0      !
;    0875  0      LITERAL
;    0876  0          TUN_K_NONTREE =     192,    ! Number of pages to not use for the tree
;    0877  0          TUN_K_FALLBACK =    64,     ! Minimum pages for tree for a large sort
;    0878  0          TUN_K_CALC_FI =     TRUE,   ! True to calculate FI in sort tree
;    0879  0          TUN_K_CALC_FE =     TRUE,   ! True to calculate FE in sort tree
;    0880  0          TUN_K_OUT_PREALL =  TRUE,   ! True to preallocate output file
;    0881  0          TUN_K_WRK_PREALL =  FALSE,  ! True to preallocate work files
;    0882  0          TUN_K_ALIGN_NODE =  2,      ! Log2 of alignment for nodes (longword align)
;    0883  0          TUN_K_ALIGN_TREE =  9,      ! Log2 of alignment for sort tree (page align)
;    0884  0          TUN_K_MRGCOST =     0,      ! Cost of merge
;    0885  0          TUN_K_PURGWS =      FALSE,  ! True to purge working set before INIT_TREE
;    0886  0      !   TUN_K_LCK_CTX =     TRUE,   ! True to lock context area in WS
;    0887  0      !   TUN_K_LCK_TREE =    3,      ! Pages of tree to lock in WS
;    0888  0      !   TUN_K_LCK_CODE =    TRUE,   ! True to lock code in WS
;    0889  0          TUN_K_BINMOVE =     32,     ! Max number of bytes to move with binary moves
;    0890  0          TUN_K_MAX_MERGE =   20;     ! Maximum merge order for internal merges
;    0891  0      MACRO
;  M 0892  0          TUN_K_BUFSIZE =
;  M 0893  0              %IF NOT HOSTILE_ELAN
;  M 0894  0              %THEN           50 * COM_K_BPERPAGE     ! Bytes in a buffer
;  M 0895  0              %ELSE            5 * COM_K_BPERPAGE     ! Bytes in a buffer
;    0896  0              %FI %;
;    0897  0      LITERAL
;    0898  0          FUN_K_CHECKPOINT =  FALSE;  ! True to generate code for checkpointing
;    0899  0      ASSERT_(TUN_K_MAX_MERGE GEQ MAX_MERGE_ORDER)
;    0900  0
;  L 0901  0      %IF NOT FUN_K_CHECKPOINT
;    0902  0      %THEN
;    0903  0          UNDECLARE %QUOTE COM_NOCHKPNT, %QUOTE COM_COUNTDOWN;
;    0904  0      %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)) %;
```

```
;    1123  0      !                       T E X T U A L   I N F O R M A T I O N
;    1124  0      !
;    1125  0      !
;    1126  0      !       User-visible text is defined here.  This text may be translated or
;    1127  0      !       changed, subject to the restrictions described below.
;    1128  0      !
;    1129  0      ! Default file extension
;    1130  0      !
;    1131  0      MACRO
;    1132  0              STR_DEF_EXT =           '.DAT' %;
;    1133  0
;    1134  0
;    1135  0      ! Default specification file, and default specification file extension
;    1136  0      !
;    1137  0      MACRO
;    1138  0              STR_DEF_SPECFILE =      'SYS$INPUT' %,
;    1139  0              STR_SPC_EXT =           '.SRT' %;
;    1140  0
;    1141  0      ! These macroes define the external and internal representations of options for
;    1142  0      ! command line qualifiers.  The first parameter in each pair may be translated;
;    1143  0      ! the second, however, is used to define internal name for this option, and may
;    1144  0      ! not be translated.
;    1145  0      !
;    1146  0      MACRO
;M   1147  0              STR_OPT_OUTFMT =                ! outfile/FORMAT=(...)
;M   1148  0                  'FIXED',            'FIXE',
;M   1149  0                  'VARIABLE',         'VARI',
;M   1150  0                  'CONTROLLED',       'CONT',
;M   1151  0                  'SIZE',             'SIZE',
;    1152  0                  'BLOCK_SIZE',       'BLOC' %,
;    1153  0
;M   1154  0              STR_OPT_INPFMT =                ! inpfile/FORMAT=(...)
;M   1155  0                  'FILE_SIZE',        'FILE',
;    1156  0                  'RECORD_SIZE',      'RECO' %,
;    1157  0
;M   1158  0              STR_OPT_PROCESS =               ! /PROCESS=...
;M   1159  0                  'RECORD',           'RECO',
;M   1160  0                  'TAG',              'TAG',
;M   1161  0                  'ADDRESS',          'ADDR',
;    1162  0                  'INDEX',            'INDE' %,
;    1163  0
;M   1164  0              STR_OPT_KEY =                   ! /KEY=...
;M   1165  0                  'ASCENDING',        'ASCE',
;M   1166  0                  'BINARY',           'BINA',
;M   1167  0                  'CHARACTER',        'CHAR',
;M   1168  0                  'DECIMAL',          'DECI',
;M   1169  0                  'DESCENDING',       'DESC',
;M   1170  0                  'UNSIGNED',         'UNSI',
;M   1171  0                  'F_FLOATING',       'F_FL',
;M   1172  0                  'D_FLOATING',       'D_FL',
;M   1173  0                  'G_FLOATING',       'G_FL',
;M   1174  0                  'H_FLOATING',       'H_FL',
;M   1175  0                  'LEADING_SIGN',     'LEAD',
;M   1176  0                  'NUMBER',           'NUMB',         ! NUMBER:nn
;M   1177  0                  'OVERPUNCHED_SIGN', 'OVER',
;M   1178  0                  'POSITION',         'POSI',         ! POSITION:nn
;M   1179  0                  'PACKED_DECIMAL',   'PACK',
```

```
;  M 1180  0                'SI',                  'SI',                  ! SIZE:nn
;  M 1181  0                'SIGNED',              'SIGN',
;  M 1182  0                'SIZE',                'SIZE',                ! SI:nn
;  M 1183  0                'SEPARATE_SIGN',       'SEPA',
;  M 1184  0                'TRAILING_SIGN',       'TRAI',
;    1185  0                'ZONED',               'ZONE' %,
;    1186  0
;  M 1187  0            STR_OPT_COLL =
;  M 1188  0                'ASCII',               'ASCI',
;  M 1189  0                'EBCDIC',              'EBCD',
;    1190  0                'DEC_MULTINATIONAL','DEC_' %;
;    1191  0
;    1192  0
;    1193  0      ! String passed to CLI$GET_VALUE to get the command line.
;    1194  0      !
;    1195  0      !MACRO
;    1196  0      !       STR_CLI_LINE =  '$LINE' %;
;    1197  0
;    1198  0
;    1199  0      ! FAO string used to output statistics via SYS$PUTMSG.
;    1200  0      !
;    1201  0      ! The following text interacts closely with the code in PRINT_STATS.
;    1202  0      ! The text can, however, be changed (translated) independent of the code, if
;    1203  0      ! the control string still uses the same FAO parameters, and text expands to
;    1204  0      ! no more than 1024 characters (a restriction of the way that the text is
;    1205  0      ! output), and lines are separated by carraige-return/line-feed pairs.
;    1206  0      !
;    1207  0      ! Note that the use of tab character in the text is avoided, since
;    1208  0      ! some terminals may not have tab stops at multiples of eight.
;    1209  0      !
;    1210  0      MACRO
; L  1211  0          STR_STATS = %EXPAND %STRING(
; L  1212  0                '!/!18* VAX-11 SORT/MERGE !AC Statistics',
; L  1213  0                '!/',
; L  1214  0                '!/Records read:!12UL',          '!10* Longest record length:!7UL',
; L  1215  0                '!/Records sorted:!10UL',        '!10* Input multiblock count:!6UL',
; L  1216  0                '!/Records output:!10UL',        '!10* Output multiblock count:!5UL',
; L  1217  0                '!/Working set extent:!6UL',     '!10* Input multibuffer count:!5UL',
; L  1218  0                '!/Virtual memory:!10UL',        '!10* Output multibuffer count:!4UL',
; L  1219  0                '!/Direct I/O:!14UL',            '!10* Number of initial runs:!6UL',
; L  1220  0                '!/Buffered I/O:!12UL',          '!10* Maximum merge order:!9UL',
; L  1221  0                '!/Page faults:!13UL',           '!10* Number of merge passes:!6UL',
; L  1222  0                    '!+!+'
; L  1223  0                '!/Sort tree size:!10UL',        '!10* Work file size used:!9UL',
; L  1224  0                    '!-!-!-!-'
; L  1225  0                '!/Elapsed time: !14%T',         '!7* Elapsed CPU:!6* !14%T',
;    1226  0                '') %;
;    1227  0
;    1228  0
;    1229  0      ! Logical names to use for work file assignments.
;    1230  0      !       The nth logical name actually used is:
;    1231  0      !       %STRING(STR_LOG_WORKFILE, (n-1)th character of STR_LOG_WORKNUM)
;    1232  0      !
;    1233  0      MACRO
;    1234  0              STR_LOG_WORKFILE =      'SORTWORK' %,
;    1235  0              STR_LOG_WORKNUM =       '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ' %;
;    1236  0
```

```
;   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' %;
```

```
;    1241  0      !                       C L E A N - U P   R O U T I N E S
;    1242  0      !
;    1243  0      !
;    1244  0      !       Clean-up routines are called by SOR$$END_SORT.  To facilitate
;    1245  0      !       information-hiding, the following mechanism is used.  It allows
;    1246  0      !       each sub-system to declare a clean-up routine to clean up its data
;    1247  0      !       structures (so that SOR$END_SORT need not know the format of the
;    1248  0      !       data structures, or even the name of the clean-up routine).
;    1249  0      !
;    1250  0      !       A clean-up routine is declared by:
;    1251  0      !               FORWARD ROUTINE CLEAN_UP;
;    1252  0      !               SOR$$END_ROUTINE_(CLEAN_UP);
;    1253  0      !               ROUTINE CLEAN_UP: CAL_CTXREG NOVALUE = ...
;    1254  0      !
;    1255  0      MACRO   SOR$$END_PSECT_(X) =    %NAME(%EXACTSTRING(30,'_','SOR$RO_CODE'),X) %;
;M   1256  0      MACRO   SOR$$END_ROUTINE_(X) =
;M   1257  0      PSECT   NODEFAULT=      %EXPAND SOR$$END_PSECT_(2)(PIC,SHARE,NOWRITE,EXECUTE);
;M   1258  0      OWN     %NAME('_',X):   PSECT(%EXPAND SOR$$END_PSECT_(2))
;    1259  0                              INITIAL(X-%NAME('_',X)) %;
```

```
;    1260  0      !                          E X E C - M O D E   V A R I A N T
;    1261  0      !
;    1262  0      !       A variant of Sort/Merge is made available to the RDMS group for
;    1263  0      !       use in EXEC mode.  This is gotten by compiling the following
;    1264  0      !       modules with the /VARIANT=1 command qualifier.  Note that the /VARIANT
;    1265  0      !       qualifier will have no effect when compiling the require files.
;    1266  0      !       External references from these modules are named SOR$fac$name.
;    1267  0      !       For example, the following code would be in SORINTERF.
;    1268  0      !
;    1269  0      !       %IF HOSTILE
;    1270  0      !       %THEN
;    1271  0      !           MACRO
;    1272  0      !               LIB$GET_VM = SOR$LIB$GET_VM %,
;    1273  0      !               LIB$FREE_VM = SOR$LIB$FREE_VM %;
;    1274  0      !       %FI
;    1275  0      !
;    1276  0      !       Another variant of Sort/Merge is made available for JRD on ELAN.
;    1277  0      !       This variant is gotten by compiling with /VARIANT=3.
;    1278  0      !       The major distinction between this and the previous is that
;    1279  0      !       the address of the context longword passed to Sort/Merge is passed
;    1280  0      !       to several of the SOR$fac$name system services.
;    1281  0      !
;    1282  0      !       The following modules are needed for these variants:
;    1283  0      !           COM.REQ, SORLIB.REQ, OPCODES.REQ, SORMSG.MSG, SORINTERF.B32,
;    1284  0      !           SORKEYSUB.B32, SORSORT.B32, SORSCRIO.B32, SORFILNAM.B32
;    1285  0      !
;    1286  0      MACRO   HOSTILE = %VARIANT %;
;    1287  0      MACRO   HOSTILE_ELAN = (%VARIANT AND %VARIANT^-1) %;
```

```
;   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
```

SORRMSIO
LIS

SORMSG
LIS

SORSCRIO
LIS

SOROUTPUT
LIS

SORLIB
LIS