

SSSSSSSSSSSS	00000000	RRRRRRRRRR	TTTTTTTTTTTT	33333333	22222222
SSSSSSSSSSSS	00000000	RRRRRRRRRR	TTTTTTTTTTTT	33333333	22222222
SSSSSSSSSSSS	00000000	RRRRRRRRRR	TTTTTTTTTTTT	33333333	22222222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSSSSSSSSS	000	RRRRRRRRRR	TTT	333	222
SSSSSSSSSS	000	RRRRRRRRRR	TTT	333	222
SSSSSSSSSS	000	RRRRRRRRRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSSSSSSSSSSS	00000000	RRR	TTT	33333333	22222222
SSSSSSSSSSSS	00000000	RRR	TTT	33333333	22222222
SSSSSSSSSSSS	00000000	RRR	TTT	33333333	22222222

SSSSSSSS	000000	RRRRRRRR	SSSSSSSS	000000	RRRRRRRR	TTTTTTTTTT	
SSSSSSSS	000000	RRRRRRRR	SSSSSSSS	000000	RRRRRRRR	TTTTTTTTTT	
SS	00	RR	SS	00	RR	TT	
SS	00	RR	SS	00	RR	TT	
SS	00	RR	SS	00	RR	TT	
SS	00	RR	SS	00	RR	TT	
SSSSSS	00	RRRRRRRR	SSSSSS	00	RRRRRRRR	TT	
SSSSSS	00	RRRRRRRR	SSSSSS	00	RRRRRRRR	TT	
	00	RR RR		00	RR RR	TT	
	00	RR RR		00	RR RR	TT	
	00	RR RR		00	RR RR	TT	
	00	RR RR		00	RR RR	TT	
	00	RR RR		00	RR RR	TT	....
SSSSSSSS	000000	RR RR	SSSSSSSS	000000	RR RR	TT	....
SSSSSSSS	000000	RR RR	SSSSSSSS	000000	RR RR	TT	....

LL	IIIIII	SSSSSSSS
LL	IIIIII	SSSSSSSS
LL	II	SS
LL	II	SS
LL	II	SS
LL	II	SS
LL	II	SSSSSS
LL	II	SSSSSS
LL	II	SS
LL	II	SS
LL	II	SS
LL	II	SS
LLLLLLLLLL	IIIIII	SSSSSSSS
LLLLLLLLLL	IIIIII	SSSSSSSS



1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57

```

0001 0  MODULE SORSSORT (
0002 0      IDENT = 'V04-000'      ! File: SORSCRIO.B32 Edit: PDG3025
0003 0  ) =
0004 1  BEGIN
0005 1
0006 1  *****
0007 1  *
0008 1  *  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0009 1  *  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0010 1  *  ALL RIGHTS RESERVED.
0011 1  *
0012 1  *  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0013 1  *  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0014 1  *  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0015 1  *  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0016 1  *  OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0017 1  *  TRANSFERRED.
0018 1  *
0019 1  *  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0020 1  *  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0021 1  *  CORPORATION.
0022 1  *
0023 1  *  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0024 1  *  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0025 1  *
0026 1  *
0027 1  *****
0028 1
0029 1
0030 1  **
0031 1
0032 1  FACILITY:      VAX-11 SORT/MERGE
0033 1
0034 1  ABSTRACT:
0035 1
0036 1          This module contains routines that control the sorting process,
0037 1          such as handling the internal sort tree, switching between runs,
0038 1          and the merge phase.
0039 1
0040 1  ENVIRONMENT:  VAX/VMS user mode
0041 1
0042 1  AUTHOR: Peter D Gilbert, CREATION DATE: 07-Jan-1982
0043 1
0044 1  MODIFIED BY:
0045 1
0046 1          T03-015      Original
0047 1          T03-016      Added clean-up routines.  PDG 4-Jan-1983
0048 1          T03-017      Fix possible unlimited recursion in READ_INSERT.
0049 1          Identified the section of code relevant to the merge problems
0050 1          of sequence checking and record deletion.
0051 1          PDG 14-Jan-1983
0052 1          T03-018      Implement merge stuff mentioned in T03-017.  PDG 26-Jan-1983
0053 1          T03-019      Use COM_MERGE (rather than COM_MRG_ORDER) to indicate a merge.
0054 1          PDG 31-Jan-1983
0055 1          T03-020      Changes for hostile environment.  PDG 3-Feb-1983
0056 1          T03-021      Add missing dot in TREE_OUTPUT.  PDG 8-Mar-1983
0057 1          T03-022      Fix check for [KEY] being [0,0,0,0].  PDG 10-Mar-1983

```

SORSORT  
V04-000

L 4  
16-Sep-1984 00:38:27  
14-Sep-1984 13:10:49

VAX-11 Bliss-32 V4.0-742  
[SORT32.SRC]SORSORT.B32;1

Page 2  
(1)

SO  
VO

:	58	0058	1	:	T03-023	Remove SOR\$\$WORK DELRUN. Remove extra parameter from
:	59	0059	1	:		call to WORK MERGE. PDG 14-Apr-1983
:	60	0060	1	:	T03-024	Use CTX BLOCK macro to use additional fields. PDG 18-Apr-1983
:	61	0061	1	:	T03-025	Some coding changes to get best generated code.
:	62	0062	1	:		Compile-time check on the size of TREE_INSERT.
:	63	0063	1	!--		

.....

```

65 0064 1 LIBRARY 'SYSS$LIBRARY:STARLET';
66 0065 1 LIBRARY 'SYSS$LIBRARY:XPORT';
67 0066 1 REQUIRE 'SRC$:COM';
68 0136 1
69 0137 1 %IF %DECLARED(%QUOTE $DESCRIPTOR) %THEN UNDECLARE %QUOTE $DESCRIPTOR; %FI
70 0138 1
71 0139 1 FORWARD ROUTINE
72 0140 1     SOR$$TREE_INIT:      CAL_CTXREG,      ! Initialize the sort tree
73 0141 1     SOR$$TREE_INSERT:   JSB_INSERT,      ! Insert a record into sort tree
74 0142 1     SOR$$TREE_EXTRACT:  JSB_EXTRACT,    ! Extract a record from the sort
75 0143 1     TREE_OUTPUT:        JSB_OUTPUT,      ! Output routine
76 0144 1     READ_INSERT:        JSB_READINS NOVALUE, ! Read a record, insert on queue
77 0145 1     MERGE_PASSES:       CAL_CTXREG NOVALUE, ! Perform the merge passes
78 0146 1     CLEAN_UP:          CAL_CTXREG NOVALUE;
79 0147 1
80 0148 1 SOR$$END_ROUTINE_(CLEAN_UP);
81 0149 1
82 0150 1 LINKAGE
83 0151 1     JSB_SCOFY_R_DX6=    JSB (REGISTER=0,REGISTER=1,REGISTER=2):
84 0152 1                     NOPRESERVE (0,1,2,3,4,5,6)
85 0153 1                     NOTUSED (7,8,9,10,11);
86 0154 1 %IF HOSTILE
87 0155 1 %THEN
88 0156 1     MACRO
89 0157 1     LIB$GET_VM = SOR$LIB$GET_VM %;
90 0158 1 %FI
91 0159 1
92 0160 1 EXTERNAL ROUTINE
93 0161 1     SOR$$WORK_NEWRUN:   JSB_NEWRUN NOVALUE,      ! Indicate start of run
94 0162 1     SOR$$WORK_MERGE:   CAL_CTXREG,              ! Decide runs to merge
95 0163 1     SOR$$WORK_WRITE:   JSB_OUTPUT,              ! Write to work file
96 0164 1     SOR$$WORK_READ:    JSB_INPUT,               ! Read from a run
97 0165 1     SOR$$ALLOCATE:     CAL_CTXREG,              ! Allocate storage
98 0166 1     SOR$$DEALLOCATE:   CAL_CTXREG NOVALUE,      ! Deallocate storage
99 0167 1     %IF NOT HOSTILE %THEN
100 0168 1     LIB$SCOPY_R_DX6:   JSB_SCOFY_R_DX6
101 0169 1                     ADDRESSING_MODE(GENERAL), ! Copy string
102 0170 1     %FI
103 0171 1     LIB$GET_VM:        ADDRESSING_MODE(GENERAL),
104 0172 1     SOR$$ERROR:       ! Error routine
105 0173 1
106 0174 1 MACRO
107 0175 1     SWAP_(X,Y) = (LOCAL Z; Z=.X; X=.Y; Y=.Z) %;
108 0176 1
109 0177 1
110 0178 1 ASSERT_(TUN_K_CALC_FI LEQU 1) ! Must be 0 or 1, as these values ...
111 0179 1 ASSERT_(TUN_K_CALC_FE LEQU 1) ! ... are used in calculations
112 0180 1
113 0181 1
114 0182 1 ! Macros to define fields in the replacement selection tree.
115 0183 1 !
116 0184 1 LITERAL
117 0185 1 !
118 0186 1 ! Offset within node of where pointers to this node point.
119 0187 1 ! Having this the same as the offset to KEY is worthwhile, so that the
120 0188 1 ! address of the key portion is the 'pointer' to the node, thus making it
121 0189 1 ! easier to keep the address of the key in COM_REG_SRC2.

```

U  
U  
U

```

122 0190 1 | If so, RN and LOSER are negative offsets from the pointers.
123 0191 1 |
124 0192 1 | This should be either 0, or 4-TUN_K_CALC_FI-TUN_K_CALC_FE, although the
125 0193 1 | code will work for any value.
126 0194 1 |
127 0195 1 | K_ROOT= 4-TUN_K_CALC_FI-TUN_K_CALC_FE;
128 0196 1 | FIELD
129 0197 1 | NODE_FIELDS =
130 0198 1 | SET
131 0199 1 |
132 0200 1 | Run number of the record pointed to by LOSER
133 0201 1 |
134 0202 1 | RN= [0-K_ROOT, L_],
135 0203 1 |
136 0204 1 | Pointer to "loser" stored in this int node
137 0205 1 |
138 0206 1 | LOSER= [1-K_ROOT, L_],
139 0207 1 |
140 0208 1 | Pointer to int node above this int node
141 0209 1 |
142 U 0210 1 | %IF NOT TUN_K_CALC_FI %THEN
143 0211 1 | FI= [2-R_ROOT, L_], %FI
144 0212 1 |
145 0213 1 | Pointer to int node above this ext node
146 0214 1 |
147 U 0215 1 | %IF NOT TUN_K_CALC_FE %THEN
148 0216 1 | FE= [3-R_ROOT-TUN_K_CALC_FI, L_], %FI
149 0217 1 |
150 0218 1 | Key and record
151 0219 1 |
152 0220 1 | KEY= [4-K_ROOT-TUN_K_CALC_FI-TUN_K_CALC_FE, A_]
153 0221 1 | TES;
154 0222 1 | MACRO
155 0223 1 | NODE_BLOCK= BLOCK FIELD(NODE_FIELDS) %;
156 0224 1 |
157 0225 1 | LITERAL
158 0226 1 |
159 0227 1 | Number of extra bytes in a node
160 0228 1 |
161 0229 1 | K_NODE= (4-TUN_K_CALC_FI-TUN_K_CALC_FE)*%UPVAL;
162 0230 1 |
163 0231 1 |
164 0232 1 | ! Define fields within COM_TREE_INSERT
165 0233 1 | !
166 0234 1 | $FIELD
167 0235 1 | S_FIELDS =
168 0236 1 | SET
169 0237 1 | $OVERLAY(COM_TREE_INSERT)
170 0238 1 | S_O= [$BYTES(0)],
171 0239 1 | S_RQ= [XLONG], ! Saved value of RQ
172 0240 1 | S_RMAX= [XLONG], ! Saved value of RMAX
173 0241 1 | S_RC= [XLONG], ! Saved value of RC
174 0242 1 | S_X= [XLONG], ! Saved value of X
175 0243 1 | S_Q= [XLONG], ! Saved value of Q
176 0244 1 | S_DESC= [XLONG], ! Saved parameter for COM_OUTPUT
177 0245 1 | S_LEN= [XLONG], ! Saved parameter for COM_OUTPUT
178 0246 1 | S_LAST= [XLONG], ! Pointer to the LASTKEY area

```

```

179      0247 1      S_QUEUE=[XLONG],      ! Pointer to queue of runs
180      L 0248 1 %IF TUN_K_CALC_FI OR TUN_K_CALC_FE
181      0249 1 %THEN
182      0250 1      S_BIT= [XLONG],
183      0251 1      S_ADJ= [XLONG],
184      0252 1 %FI
185      L 0253 1 %IF TUN_K_CALC_FI
186      0254 1 %THEN
187      0255 1      S_FIK= [XLONG],
188      0256 1 %FI
189      L 0257 1 %IF TUN_K_CALC_FE
190      0258 1 %THEN
191      0259 1      S_FEK= [XLONG],
192      0260 1 %FI
193      0261 1      S_1= [$BYTES(0)]
194      0262 1      TES;
195      0263 1
196      0264 1 ! Make sure everything fits within the size of our portion
197      0265 1 !
198      0266 1 ASSERT (%FIELDEXPAND(S_1,0)-%FIELDEXPAND(S_0,0) LEQ COM_K_TREE)
199      L 0267 1 %IF %FIELDEXPAND(S_1,0)-%FIELDEXPAND(S_0,0) LSS COM_K_TREE
200      0268 1 %THEN %INFORM('COM_K_TREE can be made smaller') %FI
201      0269 1
202      0270 1 MACRO
203      M 0271 1      FE (X,Y) =      ! Y = .X[FE]
204      M 0272 1      %IF NOT TUN_K_CALC_FE
205      M 0273 1      %THEN
206      M 0274 1      Y = .X[FE]
207      M 0275 1      %ELSE
208      M 0276 1      Y = .X ^ -1;
209      M 0277 1      IF .BITVECTOR[Y,.CTX[S_BIT];%BPVAL] THEN Y = .Y + .CTX[S_ADJ];
210      M 0278 1      Y = .Y + .CTX[S_FEK]
211      0279 1 %FI %;
212      0280 1 MACRO
213      M 0281 1      FI (X,Y) =      ! Y = .X[FI]
214      M 0282 1      %IF NOT TUN_K_CALC_FI
215      M 0283 1      %THEN
216      M 0284 1      Y = .X[FI]
217      M 0285 1      %ELSE
218      M 0286 1      Y = .X ^ -1;
219      M 0287 1      IF .BITVECTOR[Y,.CTX[S_BIT];%BPVAL] THEN Y = .Y + .CTX[S_ADJ];
220      M 0288 1      Y = .Y + .CTX[S_FIK]
221      0289 1 %FI %;
222      0290 1 !MACRO
223      0291 1      CHECK(X,K) =
224      0292 1      BEGIN
225      0293 1      LOCAL Z;
226      0294 1      %NAME(K,' ') (X,Z);
227      0295 1      IF .X[%NAME(K)] NEQ .Z THEN BUGCHECK();
228      0296 1      END %;
229      0297 1
230      0298 1 ! Macros to define the format of a queue entry
231      0299 1 !
232      0300 1 MACRO
233      0301 1      QUE_FWD = 0, L_ %;      ! Forward pointer
234      0302 1      QUE_BWD = 1, L_ %;      ! Backward pointer
235      0303 1      QUE_REC = 2, L_ %;      ! Address of the internal format record

```

```
: 236      0304 1      QUE_RUN = 3, L_ %;          ! Pointer to the run description block
: 237      0305 1 LITERAL
: 238      0306 1      QUE_K_SIZE= 4;          ! Size in longwords
: 239      0307 1
: 240      0308 1 ! The queue header for the queue is somewhat special. Normally, its QUE_REC
: 241      0309 1 ! and QUE_RUN fields are zero. For sequence-checking (which is allowed only
: 242      0310 1 ! for merges), the QUE_REC field contains the address of an internal format
: 243      0311 1 ! record which has not yet been written, and QUE_PRESENT indicates that this
: 244      0312 1 ! record has not been deleted. QUE_PRESENT overlays the QUE_RUN field.
: 245      0313 1
: 246      0314 1 MACRO
: 247      0315 1      QUE_PRESENT = %EXPAND QUE_RUN %;
```



```

249 0316 1 GLOBAL ROUTINE SOR$$TREE_INIT
250 0317 1 (
251 0318 1     PAGES,           ! Pages available for the sort tree
252 0319 1     EXP_RECS      ! Expected number of input records
253 0320 1 ):          CAL_CTXREG =
254 0321 1 ++
255 0322 1
256 0323 1 FUNCTIONAL DESCRIPTION:
257 0324 1
258 0325 1     This routine initializes the internal replacement selection tree.
259 0326 1
260 0327 1 FORMAL PARAMETERS:
261 0328 1
262 0329 1     PAGES           Number of pages available for the sort tree
263 0330 1     EXP_RECS      Expected number of input records
264 0331 1     CTX           Longword pointing to work area (passed in COM_REG_CTX)
265 0332 1
266 0333 1 IMPLICIT INPUTS:
267 0334 1
268 0335 1     The memory for the sort tree has already been allocated.
269 0336 1
270 0337 1 IMPLICIT OUTPUTS:
271 0338 1
272 0339 1     NONE
273 0340 1
274 0341 1 ROUTINE VALUE:
275 0342 1
276 0343 1     Status code.
277 0344 1
278 0345 1 SIDE EFFECTS:
279 0346 1
280 0347 1     NONE
281 0348 1
282 0349 1 NOTES:
283 0350 1
284 0351 1     The following routines are accessed through the context area:
285 0352 1
286 0353 1     CTX[COM_COMPARE]      ! Compare records
287 0354 1     (see SOR$$KEY_SUB)
288 0355 1     CTX[COM_INPUT]       ! Convert and copy routine
289 0356 1     (see SOR$$KEY_SUB)
290 0357 1     CTX[COM_NEWRUN]      ! Indicate a new run
291 0358 1     SOR$$WORK_NEWRUN
292 0359 1     RSB                  ! Nothing special if fits in tree
293 0360 1     CTX[COM_OUTPUT]     ! Output a record to a temp file
294 0361 1     SOR$$WORK_WRITE    ! Output to a work file
295 0362 1     TREE_OUTPOT       ! Output to a descriptor
296 0363 1 --
297 0364 2 BEGIN
298 0365 2 EXTERNAL REGISTER
299 0366 2     CTX = COM_REG_CTX: REF CTX_BLOCK_(S_FIELDS);
300 0367 2 LITERAL
301 0368 2     N_TREE = 2,      ! Nodes needed in the tree
302 0369 2     N_LASTKEY = 0,  ! Another node to hold the last key output
303 0370 2     N_ROUND = 2,    ! Truncate number of nodes to a mult of N_ROUND
304 0371 2
305 0372 2     ! Minimum number of nodes needed from GET_VM

```

306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362

0373  
0374  
0375  
0376  
0377  
0378  
0379  
0380  
0381  
0382  
0383  
0384  
0385  
0386  
0387  
0388  
0389  
0390  
0391  
0392  
0393  
0394  
0395  
0396  
0397  
0398  
0399  
0400  
0401  
0402  
0403  
0404  
0405  
0406  
0407  
0408  
0409  
0410  
0411  
0412  
0413  
0414  
0415  
0416  
0417  
0418  
0419  
0420  
0421  
0422  
0423  
0424  
0425  
0426  
0427  
0428  
0429

```

: This is to guarantee that:
: (k_minp - n_lastkey) and not (n_round-1) >= n_tree
K_MINP = ROUND_(N_TREE, N_ROUND) + N_LASTKEY;

LOCAL
P,           ! Number of nodes in the tree
X: REF BLOCK, ! Pointer to root of the tree
INTLEN,     ! Length of internal node
STATUS;     ! Status

! Calculate the length of an internal node, including the LOSER pointer,
! the run number, et.al. Round up to align on an addressing boundary.
INTLEN = ROUND_(.CTX[COM_LRL_INT] + K_NODE, 1^TUN_K_ALIGN_NODE);

! Determine the number of records to have in the tree, based on either:
! the expected number of records to be sorted plus, an extra K_MINP
! or
! the number of pages allowed for the sort tree.
P = MINU( .EXP RECS + K_MINP,
COM_K_BPERPAGE * .PAGES / .INTLEN + TUN_K_MRG COST );

IF .P LSS K_MINP THEN P = K_MINP;           ! Get at least this many nodes

WHILE TRUE DO
BEGIN
CTX[COM_TREE_LEN] = ROUND_(.P*.INTLEN, 1^TUN_K_ALIGN_TREE);
STATUS = LIB$GET_VM( CTX[COM_TREE_LEN], CTX[COM_TREE_ADR] );
! If we got the memory, exit the loop, so we don't retry.
IF .STATUS THEN EXITLOOP;
! Complain, and then try asking for less memory.
SOR$$ERROR(SORS_SHR_SYSERROR AND NOT STS$$SEVERITY OR STS$$WARNING,
0, .STATUS);
P = .P * 7 / 8;
IF .P LSS K_MINP THEN RETURN SOR$$ERROR(SORS_SHR_INSVIRMEM);
END;
X = .CTX[COM_TREE_ADR];
P = .CTX[COM_TREE_LEN] / .INTLEN;           ! Divvy up the space

! Truncate P to an even number to speed up the initialization loop
P = .P AND NOT (N_ROUND-1);

! Initialize variables
CTX[S_RMAX] = 0;
CTX[S_RC] = 0;
CTX[S_RQ] = 0;

```

```

363      0430      2      CTX[S_LAST] = 0;
364      0431      2      CTX[S_X] = X[K_ROOT,A_];
365      0432      2      CTX[S_Q] = X[K_ROOT,A_];
366      0433      2
367      0434      2      CTXLCOM_STAT_NODES] = .P;
368      0435      2
369      0436      2      ! Compute constants for calculating Y[FI] and Y[FE] with FI_ and FE_ macros
370      0437      2      !
371      L 0438      2      %IF TUN_K_CALC_FI OR TUN_K_CALC_FE
372      0439      2      %THEN
373      0440      3      BEGIN
374      0441      3      BUILTIN
375      0442      3      FFS;
376      0443      3      LOCAL
377      0444      3      B;
378      0445      3      FFS(%REF(0), %REF(%BPVAL), INTLEN, CTX[S_BIT]); ! INTLEN must be even!
379      0446      3      B = .BITVECTOR[ CTX[S_X], .CTX[S_BIT] ];
380      0447      3      CTX[S_BIT] = .CTX[S_BIT] - 1; ! Since we first divide by 2
381      L 0448      3      %IF TUN_K_CALC_FI
382      0449      3      %THEN
383      0450      3      CTX[S_FIK] = .CTX[S_X]/2 - .B*.INTLEN/2;
384      0451      3
385      L 0452      3      %IF TUN_K_CALC_FE
386      0453      3      %THEN
387      0454      3      CTX[S_FEK] = (.P*.INTLEN+.CTX[S_X])/2 - .B*.INTLEN/2; ! .P must be even!
388      0455      3
389      0456      3      CTX[S_ADJ] = - .INTLEN/2 + .B*.INTLEN;
390      0457      2      END;
391      0458      2      %FI
392      0459      2
393      0460      2
394      0461      2      ! NODE[J,LOSER] = NODE[J]
395      0462      2      ! NODE[J,RN] = 0
396      0463      2      ! NODE[J,FE] = NODE[(P+J)/2]
397      0464      2      ! NODE[J,FI] = NODE[(J/2)]
398      0465      2
399      0466      3      BEGIN
400      0467      3      LOCAL
401      0468      3      Y: REF NODE_BLOCK,
402      0469      3      J2LEN,
403      0470      3      P2LEN;
404      0471      3      Y = .CTX[S_X];
405      0472      3      P2LEN = (.P/2)*.INTLEN;
406      0473      3      J2LEN = 0;
407      0474      3      DECR K FROM .P/2-1 TO 0 DO ! INCR J FROM 0 TO .P-1
408      0475      4      BEGIN
409      0476      4      Y[RN] = 0; ! Set run number to zero
410      0477      4      Y[LOSER] = Y[BASE_]; ! Set loser to point to self
411      L 0478      4      %IF NOT TUN_K_CALC_FI
412      U 0479      4      %THEN
413      U 0480      4      Y[FI] = Y[BASE_] + .J2LEN;
414      0481      4      %FI
415      L 0482      4      %IF NOT TUN_K_CALC_FE
416      U 0483      4      %THEN
417      U 0484      4      Y[FE] = Y[BASE_] + .J2LEN + .P2LEN;
418      0485      4      %FI
419      0486      4      Y = Y[BASE_] + .INTLEN; ! Advance to next node

```

```

420 L 0487 4 %IF NOT TUN_K_CALC_FI OR NOT TUN_K_CALC_FE
421 U 0488 4 %THEN
422 U 0489 4 J2LEN = .J2LEN - .INTLEN;
423 U 0490 4 %FI
424 U 0491 4 Y[RN] = 0; ! Set run number to zero
425 U 0492 4 Y[LOSER] = Y[BASE_]; ! Set loser to point to self
426 L 0493 4 %IF NOT TUN_K_CALC_FI
427 U 0494 4 %THEN
428 U 0495 4 Y[FI] = Y[BASE_] + .J2LEN;
429 U 0496 4 %FI
430 L 0497 4 %IF NOT TUN_K_CALC_FE
431 U 0498 4 %THEN
432 U 0499 4 Y[FE] = Y[BASE_] + .J2LEN + .P2LEN;
433 U 0500 4 %FI
434 U 0501 4 Y = Y[BASE_] + .INTLEN; ! Advance to next node
435 U 0502 3 END;
436 U 0503 2 END;
437 U 0504 2
438 U 0505 2 CTX[COM_NEWRUN] = SOR$$WORK_NEWRUN; ! Indicate a new run
439 U 0506 2 CTX[COM_OUTPUT] = 0; ! Output a record to a temp file
440 U 0507 2 ! (access violate if we try to output
441 U 0508 2 ! a record before calling NEWRUN).
442 U 0509 2 RETURN S$$_NORMAL;
443 U 0510 1 END;

```

```

.TITLE SORSSORT
.IDENT \V04-000\
.PSECT SOR$RO_CODE-----2,NOWRT, SHR, PIC,

```

0000000V 00000 \_CLEAN\_UP:

```

.LONG <CLEAN_UP-_CLEAN_UP>
.EXTRN SOR$$WORK_NEWRUN
.EXTRN SOR$$WORK_MERGE
.EXTRN SOR$$WORK_WRITE
.EXTRN SOR$$WORK_READ, SOR$$ALLOCATE
.EXTRN SOR$$DEALLOCATE
.EXTRN LIB$COPY R_DX6
.EXTRN LIB$GET_VM, SOR$$ERROR
.PSECT SOR$RO_CODE,NOWRT, SHR, PIC,2

```

```

56 0000000G 007C 0000 .ENTRY SOR$$TREE INIT, Save R2,R3,R4,R5,R6 : 0316
50 0088 CB 3C 00002 MOVAB SOR$$ERROR, R6 : 0388
50 08 CB C0 0000E ADDL2 #11, R0 :
53 50 03 CB 00011 BICL3 #3, R0, INTLEN : 0396
51 08 AC 02 C1 00015 ADDL3 #2, EXP RECS, R1 : 0397
50 04 AC 09 78 0001A ASHL #9, PAGES, R0 :
50 53 C6 0001F DIVL2 INTLEN, R0 :
51 50 D0 00022 CMPL R1, R0 :
52 51 D0 0002A 1$: MOVL R0, R1 : 0396
02 52 D1 0002D CMPL P, #2 : 0399

```

			03	18	00030	BGEQ	2\$		
		52	02	D0	00032	MOVL	#2, P		
		54	0C	9E	00035	MOVAB	196(CTX), R4	0403	
51		52	53	C5	0003A	MULL3	INTLEN, P, R1		
		51	C1	9E	0003E	MOVAB	511(R1), R1		
64		51	8F	CB	00043	BICL3	#511, R1, (R4)		
			CB	9F	0004B	PUSHAB	200(CTX)	0404	
			54	DD	0004F	PUSHL	R4		
	00000000G	00	02	FB	00051	CALLS	#2, LIB\$GET_VM		
		55	50	D0	00058	MOVL	R0, STATUS		
		24	55	E8	0005B	BLBS	STATUS, 4\$	0408	
			55	DD	0005E	PUSHL	STATUS	0413	
			7E	D4	00060	CLRL	-(SP)	0412	
			8F	DD	00062	PUSHL	#1839536		
		66	03	FB	00068	CALLS	#3, SOR\$\$ERROR		
51		52	07	C5	0006B	MULL3	#7, P, R1	0414	
52		51	08	C7	0006F	DIVL3	#8, R1, P		
		02	52	D1	00073	CMPL	P, #2	0415	
			C2	18	00076	BGEQ	3\$		
			8F	DD	00078	PUSHL	#1839860		
		66	01	FB	0007E	CALLS	#1, SOR\$\$ERROR		
			04	00081	RET				
		50	CB	D0	00082	MOVL	200(CTX), X	0417	
52		64	53	C7	00087	DIVL3	INTLEN, (R4), P	0418	
		52	01	8A	0008B	BICB2	#1, P	0423	
			AB	7C	0008E	CLRQ	36(CTX)	0427	
			AB	D4	00091	CLRL	32(CTX)	0429	
			AB	D4	00094	CLRL	60(CTX)	0430	
		54	AB	9E	00097	MOVAB	44(CTX), R4	0431	
		50	08	C0	0009B	ADDL2	#8, R0		
		64	50	D0	0009E	MOVL	R0, (R4)		
			AB	50	000A1	MOVL	R0, 48(CTX)	0432	
	30		52	D0	000A5	MOVL	P, 176(CTX)	0434	
	00B0		00	EA	000AA	FFS	#0, #32, INTLEN, 68(CTX)	0445	
44	AB	53	00	EA	000AA	FFS	#0, #32, INTLEN, 68(CTX)	0445	
	50	64	01	44	AB	EF	000B0	EXTZV	68(CTX), #1, (R4), B
			44	AB	D7	000B6	DECL	68(CTX)	0447
		51	02	C7	000B9	DIVL3	#2, (R4), R1	0450	
		55	53	C5	000BD	MULL3	INTLEN, B, R5		
		50	02	C7	000C1	DIVL3	#2, R5, R0		
4C	AB	51	50	C3	000C5	SUBL3	R0, R1, 76(CTX)		
	51	52	53	C5	000CA	MULL3	INTLEN, P, R1	0454	
		51	64	C0	000CE	ADDL2	(R4), R1		
		51	02	C6	000D1	DIVL2	#2, R1		
50	AB	51	50	C3	000D4	SUBL3	R0, R1, 80(CTX)		
	50	53	02	C7	000D9	DIVL3	#2, INTLEN, R0	0456	
48	AB	55	50	C3	000DD	SUBL3	R0, R5, 72(CTX)		
		50	64	D0	000E2	MOVL	(R4), Y	0471	
		52	02	C6	000E5	DIVL2	#2, R2	0472	
		51	53	C5	000E8	MULL3	INTLEN, R2, P2LEN		
			51	D4	000EC	CLRL	J2LEN	0473	
			14	11	000EE	BRB	6\$	0474	
			AB	D4	000F0	CLRL	-8(Y)	0476	
		FC	50	D0	000F3	MOVL	Y, -4(Y)	0477	
			53	C0	000F7	ADDL2	INTLEN, Y	0486	
			AB	D4	000FA	CLRL	-8(Y)	0491	
		FC	50	D0	000FD	MOVL	Y, -4(Y)	0492	
			53	C0	00101	ADDL2	INTLEN, Y	0501	

SORSORT  
V04-000

1 5  
16-Sep-1984 00:38:27  
14-Sep-1984 13:10:49

VAX-11 Bliss-32 V4.0-742  
[SORT32.SRC]SORSORT.B32;1

Page 12  
(3)

SO  
VO

14	E9		52	F4	00104	68:	SOBGEQ	K, 58	:	0474
	AB	00000000G	00	9E	00107		MOVAB	SORS\$WORK_NEWRUN, 20(CTX)	:	0505
		OC	AB	D4	0010F		CLRL	12(CTX)	:	0506
	50		01	D0	00112		MOVL	#1, R0	:	0509
				04	00115		RET		:	0510

; Routine Size: 278 bytes, Routine Base: SORSRO\_CODE + 0000

```

445      0511 1 MACRO
446      M 0512 1 COMPARE_LSS(X,Y,DELX,DELY,EQCOND) =
447      M 0513 1 BEGIN
448      M 0514 1
449      M 0515 1 This macro expands into a key comparison and either a call to
450      M 0516 1 the equal key routine or the output routine, depending on whether
451      M 0517 1 or not the DELX and DELY parameters are present. If not present,
452      M 0518 1 the second parameter (Y) is the record that is output.
453      M 0519 1
454      M 0520 1 The value of this macro expansion is (X LSS Y), unless the keys
455      M 0521 1 compare equal.
456      M 0522 1
457      M 0523 1 LOCAL
458      M 0524 1 CMP;
459      M 0525 1 REGISTER
460      M 0526 1 _X = COM_REG_SRC1;
461      M 0527 1 X = _X;
462      M 0528 1 CMP = JSB COMPARE(.CTX[COM_COMPARE], ._X, Y);
463      M 0529 1 IF .CMP LSS 0
464      M 0530 1 THEN
465      M 0531 1 TRUE
466      M 0532 1 ELIF .CMP EQL 0
467      M 0533 1 THEN
468      M 0534 1 BEGIN
469      M 0535 1
470      M 0536 1 The result of this macro is FALSE.
471      M 0537 1
472      M 0538 1 If there is an equal key routine, and we are comparing with
473      M 0539 1 a node in the tree, call the equal key routine.
474      M 0540 1
475      M 0541 1 If there is no equal key routine, and we are comparing with
476      M 0542 1 the last key output, we can output the record now.
477      M 0543 1
478      M 0544 1 IF .CTX[COM_EQUAL] NEQ 0
479      M 0545 1 THEN
480      M 0546 1 %IF %NULL(DELX)
481      M 0547 1 %THEN
482      M 0548 1 0 ! Don't do deletes
483      M 0549 1 %ELSE
484      M 0550 1 %IF NOT %NULL(EQCOND) %THEN IF EQCOND THEN %FI
485      M 0551 1 BEGIN
486      M 0552 1 LOCAL SS: BLOCK[1];
487      M 0553 1 SS = JSB_EQUAL(.CTX[COM_EQUAL], ._X, Y);
488      M 0554 1
489      M 0555 1 Check the returned status and delete
490      M 0556 1 records from the sort, as requested.
491      M 0557 1
492      M 0558 1 Note that if X is deleted, and Y is not deleted,
493      M 0559 1 this macro should really have the value "true".
494      M 0560 1 The effect of this inaccuracy is that there may be
495      M 0561 1 empty nodes in the tree, which could have been put
496      M 0562 1 to better use creating longer runs. If the equal-key
497      M 0563 1 routine has a choice, the second record should be
498      M 0564 1 deleted, rather than the first.
499      M 0565 1
500      M 0566 1 Finally, if the user supplied an equal-key routine,
501      M 0567 1 a routine is generated to call his routine.

```

```

502 M 0568 1
503 M 0569 1
504 M 0570 1
505 M 0571 1
506 M 0572 1
507 M 0573 1
508 M 0574 1
509 M 0575 1
510 M 0576 1
511 M 0577 1
512 M 0578 1
513 M 0579 1
514 M 0580 1
515 M 0581 1
516 M 0582 1
517 M 0583 1
518 M 0584 1
519 M 0585 1
520 M 0586 1
521 M 0587 1
522 M 0588 1
523 M 0589 1
524 M 0590 1
525 M 0591 1
526 M 0592 1
527 M 0593 1
528 M 0594 1
529 M 0595 1
530 M 0596 1
531 M 0597 1
532 M 0598 1
533 M 0599 1
534 M 0600 1
535 M 0601 1
536 M 0602 1
537 M 0603 1
538 M 0604 1
539 M 0605 1
540 M 0606 1
541 M 0607 1
542 M 0608 1
543 M 0609 1
544 M 0610 1
545 M 0611 1
546 M 0612 1
547 M 0613 1
548 M 0614 1
549 M 0615 1
550 M 0616 1
551 M 0617 1
552 M 0618 1
553 M 0619 1

```

```

! If an error is returned from the user routine, it
! is signalled from the generated routine, and
! SSS_NORMAL is returned here. Thus we need only
! distinguish between the expected returned statuses
! and SSS_NORMAL. The following code will cause a
! problem only if the equal-key routine returns an
! unexpected successful Sort status.
IF .SS[STSSV_FAC_NO] EQL SORTS_FACILITY
THEN
BEGIN
IF DIST_(.SS, (SORS_DELETE1, SORS_DELBOTH),
(SORS_DELETE2))
THEN (DELX);
IF DIST_(.SS, (SORS_DELETE2, SORS_DELBOTH),
(SORS_DELETE1))
THEN (DELY);
END;
%IF NOT %NULL(EQCOND) %THEN ELSE 0 %FI
ELSE %FI ! There is no equal-key routine
%IF %NULL(DELX)
%THEN
BEGIN
! The keys have been found to be equal, and there is
! no equal-key routine. This implies that we are
! comparing the record with the LASTKEY record.
! Thus, we can safely output the record now, and we
! need not update the LASTKEY record, since it's
! equal to the current record.
CTX[ES_LAST] = JSB_OUTPUT(.CTX[COM_OUTPUT], Y);
! We can safely return at this point, since no updates
! are made to the tree. The "keep on chugging" code
! which flushes the tree has no effect, since having
! this record implies we aren't at the end, and the
! comparison with LASTKEY is not in a loop.
RETURN SSS_NORMAL;
END
%ELSE 0 ! Put it in the tree
%FI;
FALSE
END
ELSE FALSE
END
%:

```



```
555 0620 1 GLOBAL ROUTINE SORS$TREE_INSERT
556 0621 1 (
557 0622 1     INP_DESC:      REF BLOCK[,BYTE] VOLATILE      ! Record to insert
558 0623 1     ):      JSB_INSERT =
559 0624 1 ++
560 0625 1
561 0626 1 FUNCTIONAL DESCRIPTION:
562 0627 1
563 0628 1     This routine inserts a record into the sort tree.
564 0629 1
565 0630 1 FORMAL PARAMETERS:
566 0631 1
567 0632 1     INP_DESC.rab.d  Descriptor of the record
568 0633 1                 Not really volatile, but saying it is prevents Bliss
569 0634 1                 from materializing (INP_DESC[BASE_] EQL 0).
570 0635 1     CTX             Longword pointing to work area (passed in COM_REG_CTX)
571 0636 1
572 0637 1     INP_DESC is optional, and may be absent if and only if the end of the
573 0638 1     input file has been reached.
574 0639 1
575 0640 1 IMPLICIT INPUTS:
576 0641 1
577 0642 1     The sort tree has been initialized by TREE_INIT.
578 0643 1     The address of a record output routine in the context area.
579 0644 1
580 0645 1 IMPLICIT OUTPUTS:
581 0646 1
582 0647 1     If the tree becomes full, records may be output to a scratch file.
583 0648 1
584 0649 1 ROUTINE VALUE:
585 0650 1
586 0651 1     False (SS$ ENDOFFILE) if we have completely emptied the tree.
587 0652 1     True (SS$_NORMAL) if the tree still contains elements.
588 0653 1
589 0654 1 SIDE EFFECTS:
590 0655 1
591 0656 1     NONE
592 0657 1
593 0658 1 NOTES:
594 0659 1
595 0660 1     Replacement selection is used for the dispersion pass.
596 0661 1     See Vol 3 of "The Art of Computer Programming", pages 256 & ff, for a
597 0662 1     description of the algorithm. Steps R2 and R3 are moved to the end of
598 0663 1     the loop. A flag is used to obviate testing against infinity.
599 0664 1     To avoid comparisons between uninitialized records, we test for R0
600 0665 1     equal to zero before step R6.
601 0666 1
602 0667 1     Additional code has been added for equal-key comparisons, and to avoid
603 0668 1     comparing records in uninitialized nodes, and nodes that have already
604 0669 1     been written.
605 0670 1
606 0671 1     Two special run numbers are used. These are zero and negative one.
607 0672 1     Zero is used to indicate an initially empty record; negative one
608 0673 1     indicates a record that was emptied during the final flush of records.
609 0674 1     Thus an unsigned comparison can be used for comparisons based on run
610 0675 1     numbers, while a signed comparison with zero can be used to avoid
611 0676 1     comparisons.
```

```

612 0677 1 !--
613 0678 2 BEGIN
614 0679 2 EXTERNAL REGISTER
615 0680 2 CTX = COM_REG_CTX: REF CTX_BLOCK_(S_FIELDS);
616 0681 2
617 0682 2 REGISTER
618 0683 2
619 0684 2 : Q_KEY is a pointer to the key portion of the node
620 0685 2
621 0686 2 Q_KEY = COM_REG_SRC2: REF BLOCK; ! Q[KEY] in a register
622 0687 2
623 L 0688 2 %IF %IDENTICAL( [%FIELDEXPAND(KEY)], [0,0,0,0] )
624 0689 2 %THEN
625 0690 2
626 0691 2 : The pointer to the node is the same as the KEY portion.
627 0692 2 : Thus, the same pointer can be used for both the node and the KEY.
628 0693 2
629 0694 2 BIND
630 0695 2 Q = Q_KEY: REF NODE_BLOCK
631 U 0696 2 %ELSE
632 0697 2
633 0698 2 : The pointer to the node is not the same as the KEY portion.
634 0699 2 : We must allocate another variable to point to the node itself.
635 0700 2
636 0701 2 %INFORM('This routine is non-optimal')
637 0702 2 LOCAL
638 U 0703 2 Q: REF NODE_BLOCK
639 0704 2
640 0705 2 %FI;
641 0706 2 Q = .CTX[S_Q]; ! Pointer to the node
642 0707 2 Q_KEY = Q[REY]; ! Address of key portion of node
643 0708 2
644 0709 2 : Input new record
645 0710 2
646 0711 2 IF INP_DESC[BASE_] EQL 0 ! Check for end-of-file
647 0712 2 THEN
648 0713 2 BEGIN
649 0714 2 CTX[S_RQ] = -1; ! To flush the remaining records
650 0715 2 END
651 0716 2 ELSE
652 0717 2 BEGIN
653 0718 2
654 0719 2 : Read another record into RECORD(Q)
655 0720 2
656 0721 2 : Check whether the record should be omitted.
657 0722 2
658 0723 2 IF NOT JSB_INPUT(.CTX[COM_INPUT], INP_DESC[BASE_], ! Convert, copy
659 0724 2 Q_KEY[BASE_])
660 0725 2 THEN
661 0726 2 BEGIN
662 0727 2 CTX[COM_OMI_RECNUM] = .CTX[COM_OMI_RECNUM] + 1;
663 0728 2 RETURN $$$_NORMAL;
664 0729 2 END;
665 0730 2
666 0731 2 IF
667 0732 2 BEGIN
668 0733 2 REGISTER
LAST_KEY = COM_REG_SRC1: REF BLOCK;

```

```

669 0734 4 CTX[S_RQ] = .CTX[S_RC]; ! Get current run number
670 0735 4 LAST KEY = .CTX[S_LAST];
671 0736 4 IF LAST KEY[BASE_] EQL 0 ! First time here?
672 0737 4 THEN TRUE
673 0738 5 ELSE NOT COMPARE_LSS(LAST_KEY[BASE_], Q_KEY[BASE_])
674 0739 4 END
675 0740 3 THEN
676 0741 4 BEGIN
677 0742 4 ! This new record does not belong to the current run
678 0743 4
679 0744 4 CTX[S_RQ] = .CTX[S_RQ] + 1; ! Belongs to next run
680 0745 4 IF .CTX[S_RMAX] LSS .CTX[S_RQ]
681 0746 4 THEN
682 0747 4 CTX[S_RMAX] = .CTX[S_RQ]; ! Maximize RMAX
683 0748 4
684 0749 3 END;
685 0750 2 END;
686 0751 2
687 0752 2 DO
688 0753 2 BEGIN
689 0754 3 LABEL
690 0755 3 TI;
691 0756 3 TI:
692 0757 4 BEGIN
693 0758 4
694 0759 4 ! Prepare to update
695 0760 4
696 0761 4 ! Now Q points to a new record, whose run number is RQ
697 0762 4
698 0763 4 REGISTER
699 0764 4 T: REF NODE_BLOCK;
700 0765 4 FE (Q,T); ! ? = .Q[FE];
701 0766 4 WHILE TRUE DO
702 0767 5 BEGIN
703 0768 5
704 0769 5 ! Determine which of these records is "smaller".
705 0770 5 ! First compare the winner run number with the one in the node
706 0771 5 ! (if the node is smaller, it's the new winner)
707 0772 5 ! (if the node is greater, keep the same winner),
708 0773 5 ! Then check whether the run number is 0 or -1
709 0774 5 ! (avoid comparing uninitialized or emptied records)
710 0775 5 ! (if 0, declare this node the winner, to save time),
711 0776 5 ! Then compare the keys themselves.
712 0777 5
713 0778 6 IF BEGIN
714 0779 6 IF .CTX[S_RQ] GTRU .T[RN] THEN TRUE
715 0780 6 ELIF .CTX[S_RQ] LSSU .T[RN] THEN FALSE
716 0781 6 ELIF .CTX[S_RQ] LEQ 0
717 0782 6 THEN
718 0783 7 BEGIN
719 0784 7 IF .CTX[S_RQ] EQL 0 THEN (LEAVE TI);
720 0785 7 FALSE
721 0786 7 END
722 0787 6 ELSE
723 P 0788 6 COMPARE_LSS(BLOCK[T[LOSER],KEY], Q_KEY[BASE_],
724 0789 7 T[RN] = 0, CTX[S_RQ] = 0)
725 0790 6 END

```

```

: 726      0791  5
: 727      0792  6
: 728      0793  6
: 729      0794  6
: 730      0795  6
: 731      0796  5
: 732      0797  5
: 733      0798  5
: 734      0799  4
: 735      0800  3
: 736      0801  3
: 737      0802  3
: 738      0803  3
: 739      0804  3
: 740      0805  3
: 741      0806  4
: 742      0807  4
: 743      0808  4
: 744      0809  4
: 745      0810  4
: 746      0811  4
: 747      0812  4
: 748      0813  4
: 749      0814  4
: 750      0815  4
: 751      0816  4
: 752      0817  4
: 753      0818  5
: 754      0819  5
: 755      0820  5
: 756      0821  5
: 757      0822  5
: 758      0823  4
: 759      0824  4
: 760      0825  4
: 761      0826  3
: 762      0827  3
: 763      0828  3
: 764      0829  3
: 765      0830  3
: 766      0831  3
: 767      0832  3
: 768      0833  3
: 769      0834  4
: 770      0835  4
: 771      0836  4
: 772      0837  4
: 773      0838  4
: 774      0839  4
: 775      0840  4
: 776      0841  4
: 777      0842  4
: 778      C 0843  4 X(
: 779      C 0844  4
: 780      C 0845  4
: 781      C 0846  4
: 782      C 0847  4

      THEN
      BEGIN
      SWAP(T[RN], CTX[S_RQ]);      ! RQ <--> T[RN]
      SWAP(T[LOSER], Q);          ! Q <--> T[LOSER]
      Q KEY = Q[KEY];
      END;
      FI(T,T);                    ! Go up (T = .T[FI])
      IF .T EQL .CTX[S_X] THEN LEAVE TI; ! Exit loop if we're at the top
      END;
END;

! End of run?
IF .CTX[S_RQ] GTRU .CTX[S_RC]
THEN
BEGIN
! We've just completed run number RC (which at first means run 0),
! and must prepare for the next run.

! Any special actions required by a merging pattern for subsequent
! passes of the sort should be done at this point.

! For what it's worth, at this point .RQ equals .RC + 1.
IF .CTX[S_RQ] LSS .CTX[S_RC]      ! Equiv to .CTX[S_RQ] EQL -1
THEN
BEGIN
! We've completed the initial dispersion.
RETURN SS$_ENDOFFILE;
END;
JSB NEWRUN(.CTX[COM_NEWRUN]);
CTX[S_RC] = .CTX[S_RC] + 1; ! = .CTX[S_RQ] ! Set current run number
END;

! Output top of tree
! Q points to the "champion", and RQ is its run number.
IF .CTX[S_RQ] GTR 0              ! First dummy run, or flushing?
THEN
BEGIN
! Output record pointed to by Q
CTX[S_LAST] = JSB_OUTPUT(.CTX[COM_OUTPUT], Q_KEY[BASE_]);
! ??? We don't need to exitloop if we are flushing to a work file.
EXITLOOP;

! Is this correct?
! Unless we are flushing the tree to a work file, exit the loop.

```

```

: 783 C 0848 4 IF .CTX[COM_RUNS] EQL 0
: 784 C 0849 4 THEN
: 785 C 0850 4 BEGIN
: 786 C 0851 4
: 787 C 0852 4 | We aren't writing to a work file, so leave now.
: 788 C 0853 4 |
: 789 C 0854 4 EXITLOOP;
: 790 C 0855 4 END;
: 791 C 0856 4 )%
: 792 C 0857 4 END;
: 793 C 0858 4
: 794 C 0859 4 ! If we are emptying the tree, keep looping until we output a record
: 795 C 0860 4 !
: 796 C 0861 4 !
: 797 C 0862 4 !
: 798 C 0863 4 !
: 799 C 0864 4 !
: 800 C 0865 4 ! We are emptying the tree
: 801 C 0866 4 !
: 802 C 0867 4 !
: 803 C 0868 4 !
: 804 C 0869 4 !
: 805 C 0870 4 ! Store value of 0
: 806 C 0871 4 !
: 807 C 0872 4 !
: 808 C 0873 4 !

```

5A	30	AB	D0	00000	SORS\$TREE_INSERT::		
					MOVL	48(CTX), 0	: 0706
					TSTL	INP_DESC	: 0711
					BNEQ	1\$	
					BRW	17\$	
					MOVL	INP_DESC, R9	: 0724
					JSB	@8(CTX)	
					BLBS	R0, 2\$	
					INCL	188(CTX)	: 0727
					BRB	3\$	: 0728
20	AB	28	AB	D0	0001C	2\$: MOVL 40(CTX), 32(CTX)	: 0734
					MOVL	60(CTX), LAST_KEY	: 0735
					BEQL	4\$	: 0736
					JSB	@0(CTX)	: 0738
					TSTL	CMP	
					BLSS	5\$	
					BNEQ	4\$	
					TSTL	4(CTX)	
					BNEQ	4\$	
					JSB	@12(CTX)	
3C	AB	50	D0	00038	MOVL	R0, 60(CTX)	
					BRW	19\$	
					INCL	32(CTX)	: 0745
20	AB	24	AB	D1	00042	4\$: CMPL 36(CTX), 32(CTX)	: 0746
					BGEQ	5\$	
24	AB	20	AB	D0	00049	MOVL 32(CTX), 36(CTX)	: 0748



: Routine Size: 261 bytes, Routine Base: SORSRO\_CODE + 0116

```
: 809      0874 1
: 810      0875 1 ! compile-time checks on the size of the SOR$$TREE_INSERT routine
: 811      0876 1 !
: 812      0877 1 OWN
: 813      0878 1     END__TREE_INSERT: BLOCK[0] PSECT(SORSRO_CODE) ALIGN(0);
: 814      0879 1 LITERAL
: 815      0880 1     SIZE_TREE_INSERT = END__TREE_INSERT - SOR$$TREE_INSERT,
: 816      0881 1     OLD__TREE_INSERT = 261;
: 817      U 0882 1 %IF SIZE_TREE_INSERT GTR OLD__TREE_INSERT %THEN
: 818      0883 1     %WARN('SIZE TREE INSERT has gotten larger') %FI
: 819      U 0884 1 %IF SIZE_TREE_INSERT LSS OLD__TREE_INSERT %THEN
: 820      0885 1     %INFORM('SIZE_TREE_INSERT has gotten smaller') %FI
: 821      0886 1 UNDECLARE
: 822      0887 1     END__TREE_INSERT,
: 823      0888 1     SIZE_TREE_INSERT,
: 824      0889 1     OLD__TREE_INSERT;
```

```

826 0890 1 GLOBAL ROUTINE SOR$$TREE_EXTRACT
827 0891 1 (
828 0892 1   OUT_DESC:      REF BLOCK[ BYTE],      ! Extracted record
829 0893 1   LEN:          REF VECTOR[1,WORD]
830 0894 1   ):          JSB_EXTRACT =
831 0895 1 ++
832 0896 1
833 0897 1 FUNCTIONAL DESCRIPTION:
834 0898 1
835 0899 1   This routine returns a record from the sort or merge.
836 0900 1
837 0901 1 FORMAL PARAMETERS:
838 0902 1
839 0903 1   OUT_DESC.rab.d  Descriptor of the record extracted
840 0904 1   LEN.waw.r      Address of returned length
841 0905 1   CTX           Longword pointing to work area (passed in COM_REG_CTX)
842 0906 1
843 0907 1 IMPLICIT INPUTS:
844 0908 1
845 0909 1   NONE
846 0910 1
847 0911 1 IMPLICIT OUTPUTS:
848 0912 1
849 0913 1   NONE
850 0914 1
851 0915 1 ROUTINE VALUE:
852 0916 1
853 0917 1   Status code
854 0918 1   $$$_ENDOFFILE indicates the end of the sorted records.
855 0919 1
856 0920 1 SIDE EFFECTS:
857 0921 1
858 0922 1   NONE
859 0923 1 --
860 0924 2 BEGIN
861 0925 2 EXTERNAL REGISTER
862 0926 2   CTX = COM_REG_CTX: REF CTX_BLOCK_(S_FIELDS);
863 0927 2 LITERAL
864 0928 2   OPC_RSB = 'X'05';
865 0929 2
866 0930 2
867 0931 2 ! If we haven't written to the work files, we can write directly to the
868 0932 2 ! user's buffer, via TREE_OUTPUT. Otherwise, we must first flush the
869 0933 2 ! tree into the work files, and do the merging.
870 0934 2
871 0935 2 ! Important! The output run should be included in the number of runs
872 0936 2 ! (CTX[COM_RUNS]), even if the output of the merge pass is going to the
873 0937 2 ! output file.
874 0938 2
875 0939 2 IF .CTX[COM_RUNS] EQL 0
876 0940 2 THEN
877 0941 2 BEGIN
878 0942 2   CTX[COM_NEWRUN] = UPLIT BYTE(OPC_RSB); ! Nothing special for new runs
879 0943 2   CTX[COM_OUTPUT] = TREE_OUTPUT;      ! Output to the user's buffer
880 0944 2   CTX[S_DESC] = OUT_DESC[BASE_];      ! User's buffer
881 0945 2   CTX[S_LEN] = LEN[0];                ! User's length
882 0946 2 RETURN SOR$$TREE_INSERT(0);

```



```

883 0947 3
884 0948 3
885 0949 3
886 0950 3
887 0951 3
888 0952 3
889 0953 3
890 0954 3
891 0955 3
892 0956 4
893 0957 4
894 0958 4
895 0959 4
896 0960 4
897 0961 3
898 0962 3
899 0963 3
900 0964 3
901 0965 3
902 0966 3
903 0967 3
904 0968 4
905 0969 4
906 0970 4
907 0971 4
908 0972 4
909 0973 4
910 0974 4
911 0975 4
912 0976 4
913 0977 4
914 0978 4
915 0979 5
916 0980 5
917 0981 5
918 0982 6
919 0983 6
920 0984 6
921 0985 6
922 0986 6
923 0987 6
924 0988 6
925 0989 6
926 0990 6
927 0991 6
928 P 0992 6
929 P 0993 6
930 P 0994 6
931 0995 7
932 0996 6
933 0997 6
934 0998 6
935 0999 6
936 1000 6
937 1001 6
938 1002 7
939 1003 7

```

```

END
ELSE
BEGIN
STACKLOCAL
QUEUE: REF BLOCKVECTOR[1+TUN_K_MAX_MERGE,QUE_K_SIZE],
Q_FWD: REF BLOCK;

IF .CTX[COM_STAT_PASSES] EQL 0 ! First time here?
THEN
BEGIN
! Call another routine for the merging, to keep this one simple.
MERGE_PASSES();
END;

! Get a pointer to the queue
QUEUE = .CTX[S_QUEUE];

WHILE TRUE DO
BEGIN
BUILTIN
TESTBITSS,
TESTBITSC;

Q_FWD = .QUEUE[0,QUE_FWD];

! Check for finishing the sort or merge
IF .Q_FWD NEQ QUEUE[0,BASE_]
THEN
BEGIN
IF .CTX[COM_SEQ_CHECK]
THEN
BEGIN
LOCAL
DEL;

! Check the sequence, if requested.
! Note that we make the call to the equal-key routine
! conditional on whether the "extra" record
! (at QUEUE[0,QUE_REC]) has already been deleted.
DEL = FALSE;
IF COMPARE_LSS(.Q_FWD[QUE_REC], .QUEUE[0,QUE_REC],
DEL = TRUE,
QUEUE[0,QUE_PRESENT] = FALSE,
.QUEUE[0,QUE_PRESENT] )
THEN
SOR$$ERROR(SORS_BAD_ORDER);
IF .DEL
THEN
READ_INSERT(Q_FWD[BASE_], QUEUE[0,BASE_])
ELSE
BEGIN
IF .QUEUE[0,QUE_PRESENT]

```

```

: 940 1004 7
: 941 1005 8
: 942 1006 8
: 943 1007 8
: 944 1008 8
: 945 1009 7
: 946 1010 7
: 947 1011 7
: 948 1012 7
: 949 1013 7
: 950 1014 7
: 951 1015 7
: 952 1016 7
: 953 1017 7
: 954 1018 6
: 955 1019 6
: 956 1020 5
: 957 1021 6
: 958 1022 6
: 959 1023 6
: 960 1024 6
: 961 1025 6
: 962 1026 6
: 963 1027 6
: 964 1028 6
: 965 1029 6
: 966 1030 6
: 967 1031 6
: 968 1032 5
: 969 1033 5
: 970 1034 4
: 971 1035 5
: 972 1036 5
: 973 1037 5
: 974 1038 5
: 975 1039 5
: 976 1040 5
: 977 1041 6
: 978 1042 6
: 979 1043 6
: 980 1044 6
: 981 1045 6
: 982 1046 6
: 983 1047 5
: 984 1048 5
: 985 1049 5
: 986 1050 4
: 987 1051 3
: 988 1052 3
: 989 1053 2
: 990 1054 1

```

```

THEN
  BEGIN
    CTX[S_DESC] = OUT_DESC[BASE_];
    CTX[S_LEN] = LEN[0];
    TREE_OUTPUT(.QUEUE[0],QUE_REC);
  END;
  CHSMOVE(.CTX[COM_LRL_INT],
    :Q_FWD[QUE_REC],
    :QUEUE[0],QUE_REC);
  READ_INSERT(Q_FWD[BASE_], QUEUE[0,BASE_]);
  IF TESTBITSS(QUEUE[0,QUE_PRESENT])
  THEN
    RETURN SSS_NORMAL;
    QUEUE[0,QUE_PRESENT] = TRUE;
  END;
ELSE
  END
  BEGIN
    ! Return the smallest record to the user
    ! Read another record, and insert onto the list
    CTX[S_DESC] = OUT_DESC[BASE_];
    CTX[S_LEN] = LEN[0];
    TREE_OUTPUT(.Q_FWD[QUE_REC]);
    READ_INSERT(Q_FWD[BASE_], QUEUE[0,BASE_]);
    RETURN SSS_NORMAL;
  END;
ELSE
  END
  BEGIN
    ! Proress the extra record that may be hanging around
    IF TESTBITSS(QUEUE[0,QUE_PRESENT])
    THEN
      BEGIN
        CTX[S_DESC] = OUT_DESC[BASE_];
        CTX[S_LEN] = LEN[0];
        TREE_OUTPUT(.QUEUE[0],QUE_REC);
        RETURN SSS_NORMAL;
      END
    ELSE
      RETURN SSS_ENDOFFILE;          ! We're finished
    END;
  END;
  END;
  RETURN SSS_NORMAL;
  END;
END;

```

```

0021B END__TREE_INSERT:
05 0021B P.AAA: .BLKB 0
               .BYTE 5

```

Label	Line	Code	Address	OpCode	Operand	Instruction	Address		
	7E		57	7D	00000	SOR\$TREE EXTRACT::			
	5E		08	C2	00003	MOVQ R7, -(SP)	0890		
		7A	AB	B5	00006	SUBL2 #8, SP			
			1B	12	00009	TSTW 122(CTX)	0939		
14	AB	F1	AF	9E	0000B	BNEQ 1\$			
0C	AB	0000V	CF	9E	00010	MOVAB P.AAA, 20(CTX)	0942		
34	AB	14	AE	7D	00016	MOVAB TREE_OUTPUT, 12(CTX)	0943		
			7E	D4	0001B	MOVQ OUT_DESC, 52(CTX)	0944		
			FEDA	30	0001D	CLRL -(SP)	0946		
	5E		04	C0	00020	BSBW SOR\$TREE_INSERT			
			00D3	31	00023	ADDL2 #4, SP			
		00B6	CB	B5	00026	BRW 15\$	0949		
			05	12	0002A	TSTW 182(CTX)	0954		
			00	FB	0002C	BNEQ 2\$			
0000V	CF		40	AB	00031	CALLS #0, MERGE_PASSES	0960		
04	AE		04	AE	00036	MOVL 64(CTX), QUEUE	0965		
	58			AE	00036	MOVL QUEUE, R8	0973		
	6E			68	0003A	MOVL (R8), Q_FWD			
	57			6E	0003D	MOVL Q_FWD, R7	0977		
	58			57	00040	CPL R7, R8			
				03	00043	BNEQ 4\$			
				0094	00045	BRW 12\$			
7B	5B	AB	01	E1	00048	BBC #1, 91(CTX), 11\$	0980		
			56	D4	0004D	CLRL DEL	0991		
	59	08	A7	D0	0004F	MOVL 8(R7), X	0995		
	5A	08	A8	D0	00053	MOVL 8(R8), R10			
		00	BB	16	00057	JSB @0(CTX)			
			50	D5	0005A	TSTL CMP			
			29	19	0005C	BLSS 6\$			
			34	12	0005E	BNEQ 7\$			
		04	AB	D5	00060	TSTL 4(CTX)			
			2F	13	00063	BEQL 7\$			
	2B	0C	AB	E9	00065	BLBC 12(R8), 7\$			
	5A	08	AB	D0	00069	MOVL 8(R8), R10			
		04	BB	16	0006D	JSB @4(CTX)			
1C	50	0C	10	ED	00070	CMPZV #16, #12, SS, #28			
			1D	12	00075	BNEQ 7\$			
	03	50	03	E1	00077	BBC #3, SS, 5\$			
		56	01	D0	0007B	MOVL #1, DEL			
	12	50	04	E1	0007E	BBC #4, SS, 7\$			
			0C	AB	00082	CLRL 12(R8)			
			0D	11	00085	BRB 7\$			
			001C80D0	8F	DD	00087	PUSHL #1867984	0997	
	00000000G	00	01	FB	0008D	CALLS #1, SOR\$ERROR			
		08	56	E9	00094	BLBC DEL, 9\$	0998		
		56	57	D0	00097	MOVL R7, R6	1000		
			0000V	30	0009A	BSBW READ_INSERT			
			9B	11	0009D	BRB 3\$			
		0C	0C	AB	E9	0009F	BLBC 12(R8), 10\$	1003	
	34	AB	14	AE	7D	000A3	MOVQ OUT_DESC, 52(CTX)	1006	
		5A	08	AB	D0	000A8	MOVL 8(R8), R10	1008	
			0000V	30	000AC	BSBW TREE_OUTPUT			
08	B8	08	B7	0088	CB	28	000AF	MOVC3 136(CTX), @8(R7), @8(R8)	1012
			56	57	D0	000B7	MOVL R7, R6	1013	

34	OC	A8	0000V	30	000BA	BSBW	READ_INSERT	:	1014
	OC	A8	00	E2	000BD	BBSS	#0, T2(R8), 14\$	:	1017
			01	D0	000C2	MOVL	#1, 12(R8)	:	0980
			D5	11	000C6	BRB	8\$	:	1026
	34	AB	14	AE	7D 000C8	11\$:	MOVQ	OUT_DESC, 52(CTX)	1028
		5A	08	A7	D0 000CD		MOVL	8(R7), R10	
					0000V	30	000D1	BSBW	TREE_OUTPUT
		56	57	D0	000D4	MOVL	R7, R6	:	1029
					0000V	30	000D7	BSBW	READ_INSERT
			1A	11	000DA	BRB	14\$	:	1031
OE	OC	A8	00	E5	000DC	12\$:	BBCC	#0, 12(R8), 13\$	1039
	34	AB	14	AE	7D 000E1		MOVQ	OUT_DESC, 52(CTX)	1042
		5A	08	AB	D0 000E6		MOVL	8(R8), R10	1044
					0000V	30	000EA	BSBW	TREE_OUTPUT
			07	11	000ED	BRB	14\$	:	1048
		50	0870	8F	3C 000EF	13\$:	MOVZWL	#2160, R0	
				03	11 000F4		BRB	15\$	
		50		01	D0 000F6	14\$:	MOVL	#1, R0	1052
		5E		08	C0 000F9	15\$:	ADDL2	#8, SP	1054
		57		8E	7D 000FC		MOVQ	(SP)+, R7	
				05	000FF	RSB		:	

; Routine Size: 256 bytes, Routine Base: SORSRO\_CODE + 021C

```

: 992      1055 1 ROUTINE TREE_OUTPUT
: 993      1056 1
: 994      1057 1      SRC_ADDR:      REF BLOCK      ! Address of internal format record
: 995      1058 1      ): JSB_OUTPUT =
: 996      1059 1
: 997      1060 1      ++
: 998      1061 1      FUNCTIONAL DESCRIPTION:
: 999      1062 1
: 1000     1063 1          This routine returns a record to the user.
: 1001     1064 1
: 1002     1065 1      FORMAL PARAMETERS:
: 1003     1066 1
: 1004     1067 1          SRC_ADDR.ral.v  Address of internal format record
: 1005     1068 1          CTX          Longword pointing to work area (passed in COM_REG_CTX)
: 1006     1069 1
: 1007     1070 1      IMPLICIT INPUTS:
: 1008     1071 1
: 1009     1072 1          NONE
: 1010     1073 1
: 1011     1074 1      IMPLICIT OUTPUTS:
: 1012     1075 1
: 1013     1076 1          NONE
: 1014     1077 1
: 1015     1078 1      ROUTINE VALUE:
: 1016     1079 1
: 1017     1080 1          NONE
: 1018     1081 1
: 1019     1082 1      SIDE EFFECTS:
: 1020     1083 1
: 1021     1084 1          NONE
: 1022     1085 1      --
: 1023     1086 1      BEGIN
: 1024     1087 1      EXTERNAL REGISTER
: 1025     1088 1          CTX = COM_REG_CTX:      REF CTX_BLOCK(S_FIELDS);
: 1026     1089 1      LOCAL
: 1027     1090 1          STATUS,
: 1028     1091 1          LEN,
: 1029     1092 1          ADR;
: 1030     1093 1
: 1031     1094 1          JSB_LENADR(.CTX[COM_LENADR], SRC_ADDR[BASE_]; LEN, ADR);
: 1032     1095 1      BEGIN
: 1033     1096 1      LOCAL
: 1034     1097 1          W: REF VECTOR[1,WORD];
: 1035     1098 1          IF (W = .CTX[S_LEN]) NEQ 0 THEN W[0] = .LEN;
: 1036     1099 1          END;
: 1037     1100 1
: 1038     L 1101 1          %IF NOT HOSTILE
: 1039     1102 1          %THEN
: 1040     1103 1              STATUS = LIB$COPY R_DX6(.LEN, .ADR, .CTX[S_DESC]);
: 1041     1104 1              IF NOT .STATUS THEN SORS$ERROR(SORS$_SHR_SYSERROR, 0, .STATUS);
: 1042     1105 1          %ELSE
: 1043     U 1106 1              BEGIN
: 1044     U 1107 1              BIND
: 1045     U 1108 1                  D = .CTX[S_DESC]: BLOCK[BYTE];
: 1046     U 1109 1              CH$COPY(.LEN, .ADR, 0, .D[DSC$W_LENGTH], .D[DSC$A_POINTER]);
: 1047     U 1110 1              END;
: 1048     1111 1          %FI

```

```

: 1049      1112  2
: 1050      1113  2
: 1051      1114  2  )(
: 1052      1115  2
: 1053      1116  2
: 1054      1117  2
: 1055      1118  2
: 1056      1119  2
: 1057      1120  2
: 1058      1121  2
: 1059      1122  2
: 1060      1123  2
: 1061      1124  2
: 1062      1125  2
: 1063      1126  2
: 1064      1127  2
: 1065      1128  2
: 1066      1129  2
: 1067      1130  2
: 1068      1131  2
: 1069      1132  2
: 1070      1133  2
: 1071      1134  1  )%

RETURN 0;

BIND
  D = CTX[S_DESC]: BLOCK[,BYTE];
  ASSERT(DSC$K_CLASS-Z LSSU 2)
  ASSERT(DSC$K_CLASS-S LSSU 2)
  IF .D[DSC$B_CLASS] [SSU 2]
  THEN
    CH$COPY(.LEN, .ADR, %' ', .D[DSC$W_LENGTH], .D[DSC$A_POINTER])
  ELIF
    .D[DSC$B_CLASS] EQL DSC$K_CLASS_D AND .LEN LEQU .D[DSC$W_LENGTH]
  THEN
    CH$MOVE(.LEN, .ADR, .D[DSC$A_POINTER])
  ELSE
    BEGIN
      LOCAL
        STATUS;
      STATUS = LIB$COPY R_DX6(.LEN, .ADR, D[BASE ]);
      IF NOT .STATUS THEN SORS$error(SORS$SHR_SYSERROR, 0, .STATUS);
    END;
  END;

```

	10	BB	16	00000	TREE_OUTPUT:		
					JSB	@16(CTX)	: 1094
52	38	AB	D0	00003	MOVL	56(CTX), W	: 1098
			03	13 00007	BEQL	1\$	
62		50	B0	00009	MOVW	LEN, (W)	
52	34	AB	D0	0000C	1\$:	MOVL	52(CTX), R2
	00000000G	00	16	00010	JSB	LIB\$COPY R_DX6	: 1103
11		50	E8	00016	BLBS	STATUS, 2\$	: 1104
		50	DD	00019	PUSHL	STATUS	
		7E	D4	0001B	CLRL	-(SP)	
	001C11B4	8F	DD	0001D	PUSHL	#1839540	
00000000G	00	03	FB	00023	CALLS	#3, SORS\$error	
		50	D4	0002A	2\$:	CLRL	: 1113
			05	0002C	RSB	R0	: 1134

; Routine Size: 45 bytes. Routine Base: SORS\$R0\_CODE + 031C



```

: 1130      1192      2          END %;
: 1131      1193      2
: 1132      1194      2          ! Deletion flags D_ENTRY and D_POINT
: 1133      1195      2
: 1134      1196      2
: 1135      1197      2          LOCAL
: 1136      1198      2          S:      BITVECTOR[%BPVAL] INITIAL(0);
: 1137      1199      2
: 1138      1200      2          REGISTER
: 1139      1201      2          ENTRY:  REF BLOCK;
: 1140      1202      2
: 1141      1203      2          ! Get a copy of PENTRY in a register
: 1142      1204      2
: 1143      1205      2          ENTRY = PENTRY[BASE_];
: 1144      1206      2
: 1145      1207      2
: 1146      1208      2          ! Remove ENTRY from its current place in the queue
: 1147      1209      2
: 1148      1210      2          REMQUE_(ENTRY[BASE_]);
: 1149      1211      2
: 1150      1212      2
: 1151      1213      2          ! Continue until we don't delete ENTRY
: 1152      1214      2
: 1153      1215      2          ! The deletion flags are always 0 at the start of the following loop, due
: 1154      1216      2          ! to the use of the INITIAL attribute and TESTBITSC and TESTBITCC builtins.
: 1155      1217      2
: 1156      1218      2          WHILE TRUE DO
: 1157      1219      2              BEGIN
: 1158      1220      2                  REGISTER
: 1159      1221      2                  POINT:  REF BLOCK;
: 1160      1222      2
: 1161      1223      2
: 1162      1224      2          ! Read the record
: 1163      1225      2
: 1164      1226      2          ENTRY[QUE_REC] = SORS$WORK_READ(.ENTRY[QUE_RUN]);
: 1165      1227      2
: 1166      1228      2
: 1167      1229      2          ! At the end of run, don't put ENTRY in the queue.
: 1168      1230      2          ! Thus, we will not read from this run again.
: 1169      1231      2
: 1170      1232      2          IF .ENTRY[QUE_REC] EQL 0 THEN RETURN;  ! Indicates end-of-file
: 1171      1233      2
: 1172      1234      2
: 1173      1235      2          ! Determine where the record belongs in the queue
: 1174      1236      2
: 1175      1237      2          POINT = .QUEUE[QUE_FWD];
: 1176      1238      2          WHILE POINT[BASE_] NEQ QUEUE[BASE_] DO
: 1177      1239      2              BEGIN
: 1178      1240      2                  IF COMPARE_LSS(.POINT[QUE_REC], .ENTRY[QUE_REC],
: 1179      1241      2                  S[D_POINT] = TRUE, S[D_ENTRY] = TRUE)
: 1180      1242      2                  THEN
: 1181      1243      2                      POINT = .POINT[QUE_FWD]
: 1182      1244      2                  ELSE
: 1183      1245      2                      EXITLOOP;
: 1184      1246      2                  END;
: 1185      1247      2
: 1186      1248      2          ! If keeping ENTRY, insert it in the queue just before POINT[BASE_]

```



```

: 1187 1249 3
: 1188 1250 3
: 1189 1251 3
: 1190 1252 4
: 1191 1253 4
: 1192 1254 4
: 1193 1255 4
: 1194 1256 4
: 1195 1257 4
: 1196 1258 4
: 1197 1259 4
: 1198 1260 4
: 1199 1261 4
: 1200 1262 4
: 1201 1263 4
: 1202 1264 5
: 1203 1265 5
: 1204 1266 5
: 1205 1267 5
: 1206 1268 5
: 1207 1269 4
: 1208 1270 4
: 1209 1271 4
: 1210 1272 3
: 1211 1273 4
: 1212 1274 4
: 1213 1275 4
: 1214 1276 4
: 1215 1277 4
: 1216 1278 4
: 1217 1279 4
: 1218 1280 4
: 1219 1281 4
: 1220 1282 4
: 1221 1283 4
: 1222 1284 4
: 1223 1285 4
: 1224 1286 4
: 1225 1287 4
: 1226 1288 3
: 1227 1289 3
: 1228 1290 2
: 1229 1291 2
: 1230 1292 2
: 1231 1293 1

```

```

!
IF TESTBITCC(S[D_ENTRY])
THEN
BEGIN
! Insert ENTRY in the queue just before POINT[BASE_]
INSQUE_(ENTRY[BASE_], POINT[BASE_]);
! To delete POINT, make ENTRY look like POINT and continue
! looping. This avoids a recursive call.
! If keeping both, return (hurrah, hurrah!).
IF TESTBITSC(S[D_POINT])
THEN
BEGIN
ENTRY = POINT[BASE_];
REMQUE_(ENTRY[BASE_]);
S<0,%BPUNIT*%ALLOCATION(S),0> = 0;
END
ELSE
RETURN;
END
ELSE
BEGIN
! To delete POINT, use READ_INSERT to read from its run.
! This is a recursive invocation of READ_INSERT.
! Note that the maximum recursion level is TUN_K_MAX_MERGE,
! because we've deleted a item from the queue, and have not yet
! inserted anything in the queue.
IF TESTBITSC(S[D_POINT])
THEN
READ_INSERT(POINT[BASE_], QUEUE[BASE_]);
! Continue looping, since we need to read another ENTRY.
O;
END;
END; ! Continue until we don't delete ENTRY
END;

```

```

57 DD 0000 READ_INSERT:
7E D4 0002 PUSHL R7
56 D0 0004 CLRL S
04 A0 04 67 D0 0007 MOVL PENTRY, ENTRY
04 B7 04 A7 D0 000A MOVL (ENTRY), R0
50 D0 000F MOVL 4(ENTRY), 4(R0)
MOVL R0, @4(ENTRY)

```

```

: 1135
: 1168
: 1205
: 1210
:

```

	59	0C	A7	D0	00013	1\$:	MOVL	12(ENTRY), R9	1226
		00000000G	00	16	00017		JSB	SOR\$\$WORK READ	
08	A7		50	D0	0001D		MOVL	R0, 8(ENTRY)	
	5A	08	A7	D0	00021		MOVL	8(ENTRY) R10	1232
			6D	13	00025		BEQL	8\$	
	56		68	D0	00027		MOVL	(QUEUE), POINT	1237
	58		56	D1	0002A	2\$:	CMPL	POINT, QUEUE	1238
			31	13	0002D		BEQL	5\$	
	59	08	A6	D0	0002F		MOVL	8(POINT), _X	1241
		00	BB	16	00033		JSB	@0(CTX)	
			50	D5	00036		TSTL	CMPL	
			21	19	00038		BLSS	4\$	
			24	12	0003A		BNEQ	5\$	
		04	AB	D5	0003C		TSTL	4(CTX)	
			1F	13	0003F		BEQL	5\$	
		04	BB	16	00041		JSB	@4(CTX)	
1C	50	0C	10	ED	00044		CMPZV	#16, #12, SS, #28	
			15	12	00049		BNEQ	5\$	
	03	50	03	E1	0004B		BBC	#3, SS, 3\$	
		6E	02	88	0004F		BISB2	#2, S	
	0A	50	04	E1	00052	3\$:	BBC	#4, SS, 5\$	
		6E	01	88	00056		BISB2	#1, S	
			05	11	00059		BRB	5\$	
		56	66	D0	0005B	4\$:	MOVL	(POINT), POINT	1243
			CA	11	0005E		BRB	2\$	
	27	6E	00	E4	00060	5\$:	BBSC	#0, S, 7\$	1250
		67	56	D0	00064		MOVL	POINT, (ENTRY)	1256
	04	A7	04	A6	D0	00067	MOVL	4(POINT), 4(ENTRY)	
	04	B7	57	D0	0006C		MOVL	ENTRY, @4(ENTRY)	
	04	A6	57	D0	00070		MOVL	ENTRY, 4(POINT)	
	1C	6E	01	E5	00074		BBCC	#1, S, 8\$	1262
		57	56	D0	00078		MOVL	POINT, ENTRY	1265
		50	67	D0	0007B		MOVL	(ENTRY), R0	1266
	04	A0	04	A7	D0	0007E	MOVL	4(ENTRY), 4(R0)	
	04	B7	50	D0	00083		MOVL	R0, @4(ENTRY)	
			6E	D4	00087		CLRL	S	1267
			88	11	00089	6\$:	BRB	1\$	1262
	84	6E	01	E5	0008B	7\$:	BBCC	#1, S, 1\$	1281
			FF6E	30	0008F		BSBW	READ_INSERT	1283
			F5	11	00092		BRB	6\$	1218
		5E	04	C0	00094	8\$:	ADDL2	#4, SP	1293
		57	8E	D0	00097		MOVL	(SP)+, R7	
			05	0009A			RSB		

; Routine Size: 155 bytes, Routine Base: SOR\$RO\_CODE + 0349

```

1233 1294 1 ROUTINE MERGE_PASSES:  CA_CTXREG NOVALUE =
1234 1295 1
1235 1296 1 +-+
1236 1297 1
1237 1298 1 FUNCTIONAL DESCRIPTION:
1238 1299 1
1239 1300 1     This routine performs the merge passes.
1240 1301 1
1241 1302 1 FORMAL PARAMETERS:
1242 1303 1
1243 1304 1     CTX             Longword pointing to work area (passed in COM_REG_CTX)
1244 1305 1
1245 1306 1 IMPLICIT INPUTS:
1246 1307 1
1247 1308 1     NONE
1248 1309 1
1249 1310 1 IMPLICIT OUTPUTS:
1250 1311 1
1251 1312 1     NONE
1252 1313 1
1253 1314 1 ROUTINE VALUE:
1254 1315 1
1255 1316 1     NONE
1256 1317 1
1257 1318 1 SIDE EFFECTS:
1258 1319 1
1259 1320 1     NONE
1260 1321 1 --
1261 1322 2 BEGIN
1262 1323 2 EXTERNAL REGISTER
1263 1324 2     CTX = COM_REG_CTX:  REF CTX_BLOCK_(S_FIELDS);
1264 1325 2 LOCAL
1265 1326 2     QUEUE: REF BLOCKVECTOR[1+TUN_K_MAX_MERGE,QUE_K_SIZE],
1266 1327 2     STATUS;
1267 1328 2
1268 1329 2
1269 1330 2     ! If this routine was called due to a sort (rather than a merge),
1270 1331 2     ! clean up the replacement selection tree.
1271 1332 2
1272 1333 2 IF NOT .CTX[COM_MERGE]
1273 1334 2 THEN
1274 1335 2 BEGIN
1275 1336 2     ! Flush the tree
1276 1337 2
1277 1338 2     WHILE (STATUS = SOR$$TREE_INSERT(0)) DO 0;
1278 1339 2     IF .STATUS NEQ SSS_ENDOFFILE THEN RETURN SOR$$ERROR(.STATUS);
1279 1340 2
1280 1341 2
1281 1342 2     ! Deallocate the replacement selection tree
1282 1343 2
1283 1344 2     SOR$$DEALLOCATE(.CTX[COM_TREE_LEN], CTX[COM_TREE_ADR]);
1284 1345 2 END;
1285 1346 2
1286 1347 2
1287 1348 2     ! Save the number of runs in the dispersion, for statistics
1288 1349 2
1289 1350 2

```

```

: 1290 1351 2
: 1291 1352 2
: 1292 1353 2
: 1293 1354 2
: 1294 1355 2
: 1295 1356 2
: 1296 1357 2
: 1297 1358 2
: 1298 1359 2
: 1299 1360 2
: 1300 1361 2
: 1301 1362 2
: 1302 1363 2
: 1303 1364 2
: 1304 1365 2
: 1305 1366 2
: 1306 1367 2
: 1307 1368 2
: 1308 1369 2
: 1309 1370 2
: 1310 1371 2
: 1311 1372 2
: 1312 1373 2
: 1313 1374 2
: 1314 1375 2
: 1315 1376 2
: 1316 1377 2
: 1317 1378 2
: 1318 1379 2
: 1319 1380 2
: 1320 1381 2
: 1321 1382 2
: 1322 1383 2
: 1323 1384 2
: 1324 1385 2
: 1325 1386 3
: 1326 1387 4
: 1327 1388 4
: 1328 1389 4
: 1329 1390 4
: 1330 1391 4
: 1331 1392 4
: 1332 1393 4
: 1333 1394 4
: 1334 1395 2
: 1335 1396 2
: 1336 1397 2
: 1337 1398 2
: 1338 1399 2
: 1339 1400 2
: 1340 1401 2
: 1341 1402 2
: 1342 1403 2
: 1343 1404 2
: 1344 1405 2
: 1345 1406 2
: 1346 1407 2

```

```

CTX[COM_STAT_RUNS] = .CTX[COM_RUNS];          ! Number of runs in dispersion

! Allocate storage to hold the queue.
! One entry per run, and one for the queue header.
QUEUE = CTX[S_QUEUE] = SOR$$ALLOCATE
      (-(1+TUN_K_MAX_MERGE) * QUE_K_SIZE * %UPVAL );

! While there are more runs than can be handled at once, ...
WHILE TRUE DO
  BEGIN
    LOCAL
      RUNS:      VECTOR[1+TUN_K_MAX_MERGE];

    ! One more merge pass
    CTX[COM_STAT_PASSES] = .CTX[COM_STAT_PASSES] + 1;

    ! Determine which runs to merge.
    ! This routine also initiates reading these runs.
    SOR$$WORK_MERGE(
      (.CTX[COM_RUNS]-2) MOD (TUN_K_MAX_MERGE-1) + 2,
      RUNS[0]);

    CTX[COM_STAT_MERGE] = MAXU(.RUNS[0], .CTX[COM_STAT_MERGE]);

    ! Initialize queue entries for each run
    DECR I FROM .RUNS[0] TO 1 DO
      BEGIN
        LOCAL
          P: REF BLOCK;
          P = QUEUE[.I, BASE_];
          P[QUE_FWD] = P[BASE_];          ! Point to self
          P[QUE_BWD] = P[BASE_];          ! Point to self
          P[QUE_RUN] = .RUNS[.I];        ! Init ptr to run info
        END;
        QUEUE[0,QUE_FWD] = QUEUE[0,BASE_];  ! Init queue header
        QUEUE[0,QUE_BWD] = QUEUE[0,BASE_];  ! Init queue header

    ! Read a record from each run
    DECR I FROM .RUNS[0] TO 1 DO
      READ_INSERT(QUEUE[.I,BASE_], QUEUE[0,BASE_]);

    ! If sequence checking, allocate and read an extra record
    IF .CTX[COM_SEQ_CHECK]
      THEN

```

```

: 1347      1408      4
: 1348      1409      4
: 1349      1410      4
: 1350      1411      4
: 1351      1412      4
: 1352      1413      4
: 1353      1414      4
: 1354      1415      5
: 1355      1416      5
: 1356      1417      5
: 1357      1418      5
: 1358      1419      5
: 1359      1420      5
: 1360      1421      4
: 1361      1422      3
: 1362      1423      3
: 1363      1424      3
: 1364      1425      3
: 1365      1426      3
: 1366      1427      3
: 1367      1428      3
: 1368      1429      3
: 1369      1430      3
: 1370      1431      3
: 1371      1432      3
: 1372      1433      3
: 1373      1434      3
: 1374      1435      3
: 1375      1436      3
: 1376      1437      3
: 1377      1438      4
: 1378      1439      4
: 1379      1440      4
: 1380      1441      4
: 1381      1442      4
: 1382      1443      4
: 1383      1444      4
: 1384      1445      3
: 1385      1446      3
: 1386      1447      2
: 1387      1448      2
: 1388      1449      1

```

```

BEGIN
LOCAL
  Q_FWD: REF BLOCK;
  QUEUE[0,QUE_REC] = SOR$$ALLOCATE(.CTX[COM_LRL_INT]);
  Q_FWD = .QUEUE[0,QUE_FWD];
  IF .Q_FWD NEQ QUEUE[0,BASE_]
  THEN
    BEGIN
      CHSMOVE(.CTX[COM_LRL_INT],
        .Q_FWD[QUE_REC],
        .QUEUE[0,QUE_REC]);
      QUEUE[0,QUE_PRESENT] = TRUE;
      READ_INSERT(Q_FWD[BASE_], QUEUE[0,BASE_]);
    END;
  END;

! If this is the final pass (indicated by comparing the number of
! active runs with the number of runs being merged), return now
! TREE_EXTRACT will use READ_INSERT to get the records.
! Note that for a merge, we will always return here.
IF .CTX[COM_RUNS] - .RUNS[0] LEQ 1      ! EQL suffices, LEQ is robust
THEN
  RETURN;

! Process all the runs
WHILE .QUEUE[0,QUE_FWD] NEQ QUEUE[0,BASE_] DO
  BEGIN
    ! Output the smallest record
    ! Read another record, and insert onto the list
    SOR$$WORK WRITE(.BLOCK[.QUEUE[0,QUE_FWD], QUE_REC]);
    READ_INSERT(.QUEUE[0,QUE_FWD], QUEUE[0,BASE_]);
  END;
END;
END;
END;

```

				07FC 0000 MERGE_PASSES:				
	5E	AC	AE	9E 00002	.WORD	Save R2,R3,R4,R5,R6,R7,R8,R9,R10		: 1294
		SC	AB	95 00006	MOVAB	-84(SP), SP		: 1333
			2D	19 00009	TSTB	92(CTX)		: 1339
			7E	D4 0000B 1\$:	BLSS	3\$		
			FD22	30 0000D	CLRL	-(SP)		
	5E		04	C0 00010	BSBW	SOR\$\$TREE_INSERT		
	F5		50	E8 00013	ADDL2	#4, SP		
00000870	8F		50	D1 00016	BLBS	STATUS, 1\$		
					CMPL	STATUS, #2160		: 1340

			0A	13	0001D	BEQL	2\$			
			50	DD	0001F	PUSHL	STATUS			
		00000000G	00	01	FB 00021	CALLS	#1, SOR\$\$ERROR			
					04 00028	RET				
				00C8	CB 9F 00029	2\$: PUSHAB	200(CTX)			1345
				00C4	CB DD 0002D	PUSHL	196(CTX)			
		00000000G	00	02	FB 00031	CALLS	#2, SOR\$\$DEALLOCATE			
		00B4	CB	7A	AB B0 00038	3\$: MOVW	122(CTX), 180(CTX)			1351
			7E	0150	8F 3C 0003E	MOVZWL	#336, -(SP)			1358
		00000000G	00	01	FB 00043	CALLS	#1, SOR\$\$ALLOCATE			
		40	AB	50	D0 0004A	MOVL	R0, 64(CTX)			
			58	50	D0 0004E	MOVL	R0, QUEUE			1357
				00B6	CB B6 00051	4\$: INCW	182(CTX)			1370
			50	5E	DD 00055	PUSHL	SP			1378
			7E	7A	AB 3C 00057	MOVZWL	122(CTX), R0			1377
	FFFFFFFE		8F	01	7A 0005B	EMUL	#1, R0, #-2, -(SP)			
	50		50	13	7B 00064	EDIV	#19, (SP)+, R0, R0			
			50	8E	A0 9F 00069	PUSHAB	2(R0)			
		00000000G	00	02	FB 0006C	CALLS	#2, SOR\$\$WORK_MERGE			
			50	02	6E D0 00073	MOVL	RUNS, R0			1381
			50	00	ED 00076	CMPZV	#0, #8, 184(CTX), R0			
	00B8		CB	05	1B 0007D	BLEQU	5\$			
			50	00B8	CB 9A 0007F	MOVZBL	184(CTX), R0			
			50	50	90 00084	5\$: MOVB	R0, 184(CTX)			
			50	01	C1 00089	ADDL3	#1, RUNS, I			1386
			51	13	11 0008D	BRB	7\$			
			50	04	78 0008F	6\$: ASHL	#4, I, R1			1390
			51	58	C0 00093	ADDL2	QUEUE, P			
			61	51	D0 00096	MOVL	P, (P)			1391
		04	A1	51	D0 00099	MOVL	P, 4(P)			1392
		0C	A1	6E40	D0 0009D	MOVL	RUNS[I], 12(P)			1393
			EA	50	F5 000A2	7\$: SOBGTR	I, 6\$			1386
			68	58	D0 000A5	MOVL	QUEUE, (QUEUE)			1395
		04	A8	58	D0 000A8	MOVL	QUEUE, 4(QUEUE)			1396
			57	01	C1 000AC	ADDL3	#1, RUNS, I			1401
			50	0B	11 000B0	BRB	9\$			
			56	04	78 000B2	8\$: ASHL	#4, I, R0			1402
			50	50	C1 000B6	ADDL3	R0, QUEUE, R6			
				FEA8	30 000BA	BSBW	READ INSERT			
			F2	57	F5 000BD	9\$: SOBGTR	I, 8\$			
			27	01	E1 000C0	BBC	#1, 91(CTX), 10\$			1406
			5B	00B8	CB 3C 000C5	MOVZWL	136(CTX), -(SP)			1411
		00000000G	00	01	FB 000CA	CALLS	#1, SOR\$\$ALLOCATE			
			08	50	D0 000D1	MOVL	R0, 8(QUEUE)			
			56	68	D0 000D5	MOVL	(QUEUE), Q FWD			1412
			58	56	D1 000D8	CMPL	Q FWD, QUEUE			1413
				0F	13 000DB	BEQL	10\$			
		08	B8	00B8	CB 28 000DD	MOVCS	136(CTX), 29(Q_FWD), 28(QUEUE)			1418
			0C	01	D0 000E5	MOVL	#1, 12(QUEUE)			1419
				FE79	30 000E9	BSBW	READ INSERT			1420
			6E	01	C1 000EC	10\$: ADDL3	#1, RUNS, R0			1430
			50	00	ED 000F0	CMPZV	#0, #16, 122(CTX), R0			
			7A	1D	15 000F6	BLEQ	13\$			
			57	68	D0 000F8	11\$: MOVL	(QUEUE), R7			1437
			58	57	D1 000FB	CMPL	R7, QUEUE			
				03	12 000FE	BNEQ	12\$			
				FF4E	31 00100	BRW	4\$			

SORSORT  
V04-000

H 7  
16-Sep-1984 00:38:27  
14-Sep-1984 13:10:49

VAX-11 Bliss-32 V4.0-742  
[SORT32.SRC]SORSORT.B32;1

Page 37  
(9)

5A	08	A7	DO	00103	12\$:	MOVL	8(R7), R10	:	1443
	00000000G	00	16	00107		JSB	SORS\$WORK_WRITE	:	
56		57	DO	0010D		MOVL	R7, R6	:	1444
		FE52	30	00110		BSBW	READ_INSERT	:	
		E3	11	00113		BRB	11\$	:	1437
			04	00115	13\$:	RET		:	1449

; Routine Size: 278 bytes,      Routine Base: SORS\$RO\_CODE + 03E4

SOR  
VO4

.....

```

1390 1450 1 ROUTINE CLEAN_UP: CAL_CTXREG NOVALUE =
1391 1451 1
1392 1452 1 +-
1393 1453 1
1394 1454 1 FUNCTIONAL DESCRIPTION:
1395 1455 1
1396 1456 1     Release resources allocated by this module.
1397 1457 1
1398 1458 1 FORMAL PARAMETERS:
1399 1459 1
1400 1460 1     NONE
1401 1461 1
1402 1462 1 IMPLICIT INPUTS:
1403 1463 1
1404 1464 1     NONE
1405 1465 1
1406 1466 1 IMPLICIT OUTPUTS:
1407 1467 1
1408 1468 1     NONE
1409 1469 1
1410 1470 1 ROUTINE VALUE:
1411 1471 1
1412 1472 1     NONE (signals errors)
1413 1473 1
1414 1474 1 SIDE EFFECTS:
1415 1475 1
1416 1476 1     NONE
1417 1477 1
1418 1478 1 --
1419 1479 2 BEGIN
1420 1480 2 EXTERNAL REGISTER
1421 1481 2     CTX = COM_REG_CTX: REF CTX_BLOCK(S_FIELDS);
1422 1482 2 LOCAL
1423 1483 2     QUEUE: REF BLOCKVECTOR[1+TUN_K_MAX_MERGE,QUE_K_SIZE];
1424 1484 2
1425 1485 2
1426 1486 2 ! Deallocate the extra storage used for sequence checking
1427 1487 2 !
1428 1488 2 IF (QUEUE = .CTX[S_QUEUE]) NEQ 0
1429 1489 2 THEN
1430 1490 2     SOR$$DEALLOCATE(.CTX[COM_LRL_INT], QUEUE[0,QUE_REC]);
1431 1491 2
1432 1492 2
1433 1493 2 ! Deallocate storage to hold the queue.
1434 1494 2 ! One entry per run, and one for the queue header.
1435 1495 2 !
1436 1496 2 SOR$$DEALLOCATE
1437 1497 2     ( (1+TUN_K_MAX_MERGE) * QUE_K_SIZE * %UPVAL, CTX[S_QUEUE] );
1438 1498 2
1439 1499 2
1440 1500 2 ! Deallocate the replacement selection tree
1441 1501 2 !
1442 1502 2 SOR$$DEALLOCATE(.CTX[COM_TREE_LEN], CTX[COM_TREE_ADR]);
1443 1503 2
1444 1504 2
1445 1505 1 END;

```



```

0004 00000 CLEAN_UP:
52 00000000G 00 9E 00002 .WORD Save R2
50 40 AB D0 00009 MOVAB SOR$$DEALLOCATE, R2
0B 13 0000D MOVL 64(CTX), QUEUE
08 AO 9F 0000F BEQL 1$
7E 0088 CB 3C 00012 PUSHAB 8(QUEUE)
62 02 FB 00017 MOVZWL 136(CTX), -(SP)
40 AB 9F 0001A 1$: CALLS #2, SOR$$DEALLOCATE
7E 0150 8F 3C 0001D PUSHAB 64(CTX)
62 02 FB 00022 MOVZWL #336, -(SP)
00C8 CB 9F 00025 CALLS #2, SOR$$DEALLOCATE
00C4 CB DD 00029 PUSHAB 200(CTX)
62 02 FB 0002D PUSHL 196(CTX)
04 00030 CALLS #2, SOR$$DEALLOCATE
RET
: 1450
: 1488
: 1490
: 1497
: 1502
: 1505

```

: Routine Size: 49 bytes, Routine Base: SOR\$RO\_CODE + 04FA

```

: 1446 1506 1
: 1447 1507 1 END
: 1448 1508 0 ELUDOM

```

PSECT SUMMARY

Name	Bytes	Attributes
SOR\$RO_CODE	4	NOVEC,NOWRT, RD ; EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)
SOR\$RO_CODE	1323	NOVEC,NOWRT, RD ; EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)

Library Statistics

File	Total	Symbols Loaded	Percent	Pages Mapped	Processing Time
_\$255\$DUA28:[SYSLIB]STARLET.L32;1	9776	6	0	581	00:01.1
_\$255\$DUA28:[SYSLIB]XPORT.L32;1	590	21	3	252	00:00.6
_\$255\$DUA28:[SORT32.SRC]SORLIB.L32;1	409	143	34	34	00:00.4

SORSORT  
V04-000

K 7  
16-Sep-1984 00:38:27  
14-Sep-1984 13:10:49

VAX-11 Bliss-32 V4.0-742  
[SORT32.SRC]SORSORT.B32;1

Page 40  
(10)

COMMAND QUALIFIERS

:  
: BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/NOTRACE/LIS=LIS\$:SORSORT/OBJ=OBJ\$:SORSORT MSRC\$:SORSORT/UPDATE=(ENH\$:SORSORT)

: Size: 1322 code + 5 data bytes  
: Run Time: 00:40.5  
: Elapsed Time: 01:53.9  
: Lines/CPU Min: 2235  
: Lexemes/CPU-Min: 36346  
: Memory Used: 220 pages  
: Compilation Complete



[Screenshot 1]	[Screenshot 2]	[Screenshot 3]	[Screenshot 4]	[Screenshot 5]	[Screenshot 6]	[Screenshot 7]	[Screenshot 8]	[Screenshot 9]	[Screenshot 10]	[Screenshot 11]	[Screenshot 12]
[Screenshot 13]	[Screenshot 14]	[Screenshot 15]	[Screenshot 16]	[Screenshot 17]	[Screenshot 18]	[Screenshot 19]	[Screenshot 20]	[Screenshot 21]	[Screenshot 22]	[Screenshot 23]	[Screenshot 24]
[Screenshot 25]	[Screenshot 26]	[Screenshot 27]	[Screenshot 28]	[Screenshot 29]	[Screenshot 30]	[Screenshot 31]	[Screenshot 32]	[Screenshot 33]	[Screenshot 34]	[Screenshot 35]	[Screenshot 36]
[Screenshot 37]	[Screenshot 38]	[Screenshot 39]	[Screenshot 40]	[Screenshot 41]	[Screenshot 42]	[Screenshot 43]	[Screenshot 44]	[Screenshot 45]	[Screenshot 46]	[Screenshot 47]	[Screenshot 48]
[Screenshot 49]	[Screenshot 50]	[Screenshot 51]	[Screenshot 52]	[Screenshot 53]	[Screenshot 54]	[Screenshot 55]	[Screenshot 56]	[Screenshot 57]	[Screenshot 58]	[Screenshot 59]	[Screenshot 60]
[Screenshot 61]	[Screenshot 62]	[Screenshot 63]	[Screenshot 64]	[Screenshot 65]	[Screenshot 66]	[Screenshot 67]	[Screenshot 68]	[Screenshot 69]	[Screenshot 70]	[Screenshot 71]	[Screenshot 72]
[Screenshot 73]	[Screenshot 74]	[Screenshot 75]	[Screenshot 76]	[Screenshot 77]	[Screenshot 78]	[Screenshot 79]	[Screenshot 80]	[Screenshot 81]	[Screenshot 82]	[Screenshot 83]	[Screenshot 84]
[Screenshot 85]	[Screenshot 86]	[Screenshot 87]	[Screenshot 88]	[Screenshot 89]	[Screenshot 90]	[Screenshot 91]	[Screenshot 92]	[Screenshot 93]	[Screenshot 94]	[Screenshot 95]	[Screenshot 96]
[Screenshot 97]	[Screenshot 98]	[Screenshot 99]	[Screenshot 100]	[Screenshot 101]	[Screenshot 102]	[Screenshot 103]	[Screenshot 104]	[Screenshot 105]	[Screenshot 106]	[Screenshot 107]	[Screenshot 108]
[Screenshot 109]	[Screenshot 110]	[Screenshot 111]	[Screenshot 112]	[Screenshot 113]	[Screenshot 114]	[Screenshot 115]	[Screenshot 116]	[Screenshot 117]	[Screenshot 118]	[Screenshot 119]	[Screenshot 120]
[Screenshot 121]	[Screenshot 122]	[Screenshot 123]	[Screenshot 124]	[Screenshot 125]	[Screenshot 126]	[Screenshot 127]	[Screenshot 128]	[Screenshot 129]	[Screenshot 130]	[Screenshot 131]	[Screenshot 132]