

[illegible]

— 52

Syn

NTS

NTS

NTS

1994

NTS

NTS

NTS

NTS

NTS

NTS

NTS

NTS

NTS

NTS

NTS

NTS

NTS

NTS

NTS

NTS

10

10

NTS

NT

NT
NT

1

100

100

10

10

1

```
RRRRRRRR  MM      MM      000000  SSSSSSSS  TTTTTTTTTT  AAAAAA  LL      LL
RRRRRRRR  MM      MM      000000  SSSSSSSS  TTTTTTTTTT  AAAAAA  LL      LL
RR      RR  MMMM  MMMM  00      00  SS      TT      AA      AA  LL      LL
RR      RR  MMMM  MMMM  00      00  SS      TT      AA      AA  LL      LL
RR      RR  MM  MM  MM  00      0000  SS      TT      AA      AA  LL      LL
RRRRRRRR  MM      MM      00  00  00  SSSSSS  TT      AA      AA  LL      LL
RRRRRRRR  MM      MM      00  00  00  SSSSSS  TT      AA      AA  LL      LL
RR      RR  MM      MM      0000  00  SS      TT      AAAAAAAAAA  LL      LL
RR      RR  MM      MM      0000  00  SS      TT      AAAAAAAAAA  LL      LL
RR      RR  MM      MM      00      00  SS      TT      AA      AA  LL      LL
RR      RR  MM      MM      00      00  SS      TT      AA      AA  LL      LL
RR      RR  MM      MM      000000  SSSSSSSS  TT      AA      AA  LLLLLLLLLL  LLLLLLLLLL
RR      RR  MM      MM      000000  SSSSSSSS  TT      AA      AA  LLLLLLLLLL  LLLLLLLLLL
```

```
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
```


(2)	102
(3)	130
(8)	435
(12)	710

DECLARATIONS
RMSSTALL - STALL FOR I/O COMPLETION ROUTINE
RMSSTALLAST - AST ENTRY POINT FOR I/O COMPLETE
RMSCHKAST - CHECK FOR ASTS INHIBITED

```
0000 1          $BEGIN RMOSTALL,000,RMSRMS0,<STALL FOR I/O COMPLETION>,<NOWRT,QUAD>
0000 2
0000 3
0000 4 *****
0000 5 *
0000 6 *  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 7 *  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 8 *  ALL RIGHTS RESERVED.
0000 9 *
0000 10 *  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 11 *  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 12 *  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 13 *  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 14 *  OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 15 *  TRANSFERRED.
0000 16 *
0000 17 *  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 18 *  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 19 *  CORPORATION.
0000 20 *
0000 21 *  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 22 *  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 23 *
0000 24 *
0000 25 *****
0000 26
0000 27 ++
0000 28 Facility: rms32
0000 29
0000 30 Abstract:
0000 31 this module includes the various routines to
0000 32 handle required i/o stalls and the restarting
0000 33 of a thread upon i/o completion.
0000 34
0000 35 Environment:
0000 36 star processor running starlet exec.
0000 37
0000 38 Author: l f laverdure,          creation date: 4-FEB-1977
0000 39
0000 40 Modified By:
0000 41
0000 42 V03-014 RAS0269          Ron Schaefer          14-Mar-1984
0000 43 A little performance boost by re-arranging some code
0000 44 and branches and some instruction optimization.
0000 45 Correctly probe arglist before asynch copy and
0000 46 set -1 addr if not accessible so that RMSEXRRMS can
0000 47 give the user an error.
0000 48
0000 49 V03-013 DAS0004          David Solomon          02-Feb-1984
0000 50 In RM$STALL, don't call RM$LOWER_LOCK unless sharing.
0000 51
0000 52 V03-012 KPL0001          Peter Lieberwirth      13-May-1983
0000 53 Change byte immediate MOV to word immediate to account
0000 54 for increased size of FAB-related ASB.
0000 55
0000 56 V03-011 SHZ0001          Stephen H. Zalewski    13-Apr-1983
0000 57 If we enter stall via RM$STALL_LOCK, set a flag to prevent
```



```
0000 58 : us from reenqueuing for the lock after it was granted.
0000 59 :
0000 60 : V03-010 DAS0003 David Solomon 21-Feb-1983
0000 61 : Add entry point RMS$CHKAST_ANY the same as RMS$BLKFINCHK
0000 62 : for use by any RMS AST routine (e.g. it doesn't validate
0000 63 : the ASTPRM).
0000 64 :
0000 65 : V03-009 KBT0366 Keith B. Thompson 11-Oct-1982
0000 66 : Check for stack fit with new asb$w_sktlen field
0000 67 :
0000 68 : V03-008 KBT0362 Keith B. Thompson 6-Oct-1982
0000 69 : asb$b_stksiz is now a word field
0000 70 :
0000 71 : V03-007 KBT0360 Keith B. Thompson 6-Oct-1982
0000 72 : Fix check before calling restore_lock
0000 73 :
0000 74 : V03-006 KBT0323 Keith B. Thompson 9-Sep-1982
0000 75 : Remove all SO sharing code and add new STALL_LOCK test
0000 76 : on return from stall
0000 77 :
0000 78 : V03-005 JWH0002 Jeffrey W. Horn 07-Sep-1982
0000 79 : Remove test definition of IMP$V_RUH accidentally
0000 80 : left in JWH0001. Also fix bugs in JWH0001 in RUSTALL
0000 81 : logic by restoring R4, R5 before returning to caller and
0000 82 : correcting AST handling.
0000 83 :
0000 84 : V03-004 KBT0217 Keith B. Thompson 23-Aug-1982
0000 85 : Reorganize psepts
0000 86 :
0000 87 : V03-003 JWH0001 Jeffrey W. Horn 5-Aug-1982
0000 88 : Add logic to not stall if called from within
0000 89 : the RMS recovery unit handler, but to simply wait
0000 90 : in exec mode until I/O completes.
0000 91 :
0000 92 : V03-002 KBT0080 Keith B. Thompson 9-Jul-1982
0000 93 : Add stall_lock entry point
0000 94 :
0000 95 : V03-001 KDM0002 Kathleen D. Morse 28-Jun-1982
0000 96 : Added $PCBDEF.
0000 97 :
0000 98 : --
0000 99 :
0000 100 :
```



```

0000 102      .SBTTL DECLARATIONS
0000 103
0000 104      :
0000 105      : Include Files:
0000 106      :
0000 107      :
0000 108      :
0000 109      : Macros:
0000 110      :
0000 111
0000 112      $SETDEF      ; system service $setef definitions
0000 113      $IFBDEF
0000 114      $IRBDEF
0000 115      $ASBDEF
0000 116      $FABDEF
0000 117      $RABDEF
0000 118      $BDBDEF
0000 119      $PIODEF
0000 120      $PCBDEF
0000 121      $IMPDEF
0000 122      $RMSDEF
0000 123
0000 124      :
0000 125      : equated symbols
0000 126      :
0000 127
00000020 0000 128      BKP=IRB$L_BKPBITS*8      ; bit offset to bookkeeping bits

```



```

0000 130      .SBTTL  RMSSTALL - STALL FOR I/O COMPLETION ROUTINE
0000 131
0000 132      :++
0000 133      :
0000 134      RMSSTALL      stall for I/O completion routine
0000 135      RMSSTALL_LOCK  alternate entry point for stall for file lock
0000 136      RM$ENBAST      re-enable ASTs
0000 137
0000 138      this routine is called whenever a stream must stall for either an i/o
0000 139      completion or for access to the shared file database (or part thereof).
0000 140
0000 141      this routine first checks if the stalling stream is for a shared file
0000 142      and if so, the shared ifab is released.  next it checks to see whether
0000 143      an asb (asynchronous context block) exists, and if not, the stalling stream
0000 144      is for a fab function, and it allocates an asb, saving its address in the
0000 145      ifab.
0000 146
0000 147      if this is an asynchronous rab operation, copies the argument list into
0000 148      the asb, changes the arglist pointer to point to the saved copy, and sets
0000 149      the status code to rms$_pending.
0000 150
0000 151      if not an asynchronous rab operation, sets the status code to rms$_stall.
0000 152
0000 153      the routine then saves registers r4 thru r11, the stack along
0000 154      with the return pc, and the stack size in the asb.
0000 155
0000 156      finally the routine checks for running at exec ast level,
0000 157      and if so, merely returns (i.e., it exits from the ast), otherwise
0000 158      it re-enables asts, sets the status code into r0, and returns to the
0000 159      rms user possibly waiting at user's access mode).
0000 160
0000 161      return sequence depends upon following registers not being
0000 162      destroyed by the return thru the change mode dispatcher to
0000 163      the rms synchronization code:
0000 164
0000 165      r8      structure address
0000 166      r4      $wait type flag (0=same rab, 1=different rab)
0000 167      r3      efn to synchronize on
0000 168
0000 169      Calling sequence:
0000 170
0000 171      BSBW  RMSSTALL
0000 172
0000 173      Input Parameters:
0000 174
0000 175      r11      impure area address
0000 176      r10      ifab address if r9 is an irab address
0000 177      r9      ifab/irab address
0000 178
0000 179      Implicit Inputs:
0000 180
0000 181      the contents of the ifab/irab and impure area.
0000 182
0000 183      Output Parameters:
0000 184
0000 185      This routine does not return directly to the caller,
0000 186      exiting from rms instead.  return occurs via the routine

```



```

0000 187 : rm$stallast, which is entered via the ast signaling the
0000 188 : completion of the i/o being awaited by rm$stall. upon
0000 189 : return to the caller his entire context with the exception of
0000 190 : r0 thru r3 and ap is restored.
0000 191 :
0000 192 : Implicit Outputs:
0000 193 :
0000 194 : an asb is allocated, if required, and filled in.
0000 195 : the rms event flag may be cleared.
0000 196 :
0000 197 : Completion Codes:
0000 198 :
0000 199 : if returning to caller of rms, r0 will be set to
0000 200 : either rms$_pending (async) or rms$_stall (sync).
0000 201 : if rms$_stall, this code is intercepted by rms
0000 202 : code running in the caller's mode which awaits the
0000 203 : completion of the rms operation.
0000 204 :
0000 205 : if exiting from an ast, r0 is undefined.
0000 206 :
0000 207 : Side Effects:
0000 208 :
0000 209 : rms asts are reenabled.
0000 210 :
0000 211 : --
0000 212 :

```



```
0000 214
0000 215 :++
0000 216 : note: the following code is not an entry point into this routine
0000 217 :
0000 218 : must allocate an asb for a stalled fab operation.
0000 219 : first check for release of sifab.
0000 220 : point r11 to pio segment so that if a free page is required it will be
0000 221 : allocated there.
0000 222 :--
0000 223
0000 224 STALLAL:
0000 225 STSTPT STALLAL
0006 226 PUSHL R11 ; save impure area addr
0008 227 MOVAL @#PIO$GW_PIOIMPA,R11 ; point to process i/o segment
000F 228 MOVL R11,R1 ; allocate space in control page
0012 229 MOVZBL #ASB$C_BLN_FAB/4,R2 ; size required
0016 230 BSBW RMSGETBLK ; go allocate space (r1=addr)
0019 231 MOVL (SP)+,R11 ; restore impure area addr
001C 232 BLBC R0,ERRDME
001F 233 MOVB #ASB$C_BID,ASB$B_BID(R1) ; make it a real asb
0023 234
0023 235 ASSUME ASB$W_STKLEN EQ 0
0023 236
0023 237 MOVW #<ASB$C_BLN_FAB-ASB$C_BLN_FIX>,-; stuff the size of the
0027 238 (R1) ; save stack area
0028 239 MOVL R1,R2 ; copy address to right reg
002B 240 MOVL R1,IFB$L_ASBADDR(R9) ; save the asb address
002F 241 BRW SYNCOP ; join sync operation code
0032 242
0032 243 :
0032 244 : couldn't allocate space for an asb
0032 245 :
0032 246
0032 247 ERRDME: RMSTBUG FTL$_ASBALLFAIL
0039 248
0039 249 :
0039 250 : save arglist for async rab operation (first stall only)
0039 251 :
0039 252
0039 253 ASYNCOPT:
0039 254 MOVL #RMS$_PENDING,AP ; async status code
0040 255 MOVL IRB$L_ARGLST(R9),R0 ; restore arglist addr
0044 256 ; (note: already probed 1st 2 longwords)
0044 257 MOVAL ASB$L_ARGLST(R2),IRB$L_ARGLST(R9) ; point at temp arglist
0049 258 MOVZBL (R0),R1 ; get arg count
004C 259 MOVQ (R0)+,ASB$L_ARGLST(R2) ; save count and FAB/RAB addr
0050 260 DECL R1 ; at most 3 args are of interest
0052 261 BLEQU 15$ ; branch if o.k.
0054 262 IFNORD #4,(R0),20$ ; can't read remainder
005A 263 MOVL (R0)+,ASB$L_ARGLST+8(R2) ; copy ERR= addr
005E 264 DECL R1 ; count ERR=
0060 265 BLEQU 15$ ; all there is
0062 266 IFNORD #4,(R0),25$ ; can't read remainder
0068 267 MOVL (R0),ASB$L_ARGLST+12(R2) ; copy SUC= addr
006C 268 15$: BRB CTXSAV
006E 269
006E 270 20$: MNEGL #1,ASB$L_ARGLST+8(R2) ; bad ERR= addr
```



```
18 A2 01 CE 0072 271 25$: MNEGL #1,ASB$L_ARGLST+12(R2) ; bad SUC= addr
      74 11 0076 272 BRB CTXSAV
      0078 273
      C078 274 RUSTALL:
18 A9 01 CE 0078 275 MNEGL #1,IRB$L_ARGLST(R9) ; indicate RU hand stall
      007C 276 $CLREF_S #IMP$C_ASYEFN ; clear event flag
      0085 277 CSB #PIO$V_INHAST,@#PIO$GW_STATUS ; check for AST disabled
      008D 278 $SETAST_S #1 ; enable them if so
      0096 279 $WAITFR_S #IMP$C_ASYEFN ; wait for event flag
50 0C A9 3C 009F 280 MOVZWL IRB$L_IOS(R9),R0 ; get status
 54 8E 7D 00A3 281 MOVQ (SP)+,R4 ; restore R4,R5
      05 00A6 282 RSB ; go back to thread
      00A7 283
```



```
00A7 285
00A7 286 :++
00A7 287 :
00A7 288 : entry point for stall for file lock
00A7 289 :
00A7 290 : NOTE: This entry point assumes that R10 contains the address of the IFAB.
00A7 291 :
00A7 292 :--
00A7 293 :
00A7 294 RMSSTALL_LOCK::
00A7 295 $TSTPT STALLLOCK
00AD 296 SSB #IFB$V_STALL_LOCK,(R10) ; Do not retake the lock once it is
00B1 297 BRB STALL
00B3 298
00B3 299 :++
00B3 300 :
00B3 301 : entry point for this routine
00B3 302 :
00B3 303 :--
00B3 304 :
00B3 305 RMSSTALL::
00B3 306 $TSTPT STALL
00B9 307 :
00B9 308 :
00B9 309 : If sharing, lower file lock to CR.
00B9 310 :
00B9 311 :
07 08 A9 E8 00B9 312 BLBS IFB$B_BID(R9),10$ ; branch if R9 ->IFAB (else R10 ->IFAB)
78 AA D5 00BD 313 TSTL IFB$L_SF$B_PTR(R10) ; are we sharing?
0A 13 00C0 314 BEQL STALL ; no, skip call to RM$LOWER_LOCK
05 11 00C2 315 BRB 20$ ; yes, lower lock on file
78 A9 D5 00C4 316 10$: TSTL IFB$L_SF$B_PTR(R9) ; are we sharing?
03 13 00C7 317 BEQL STALL ; no, skip call to RM$LOWER_LOCK
FF34' 30 00C9 318 20$: BSBW RM$LOWER_LOCK ; lower file lock to CR
00CC 319 :
A5 7E 54 7D 00CC 320 STALL: MOVQ R4,-(SP) ; Save r4,r5
6B 06 E0 00CF 321 BBS #IMP$V_RUH,(R11),RUSTALL; branch if in RU hand
00D3 322 :
00D3 323 ASSUME IFB$L_ASBADDR EQ IRB$L_ASBADDR
00D3 324 :
52 14 A9 D0 00D3 325 MOVL IFB$L_ASBADDR(R9),R2 ; get asb address
03 12 00D7 326 BNEQ 10$ ; continue if we have one
FF24 31 00D9 327 BRW STALLAL ; stallal if we don't
00DC 328 :
00DC 329 :
00DC 330 : check for asynchronous rab operation and if so copy arglist into the asb
00DC 331 :
00DC 332 :
00DC 333 ASSUME IMP$W_RMSSTATUS EQ 0
00DC 334 :
0C 6B 01 E0 00DC 335 10$: BBS #IMP$V_AST,(R11),CTXSAV ; branch if at ast level
00E0 336 :
00E0 337 ASSUME IFB$V_ASYNC EQ IRB$V_ASYNC
00E0 338 :
03 69 23 E1 00E0 339 BBC #IRB$V_ASYNC,(R9),SYNCOPI ; continue if synch operation
FF52 31 00E4 340 BRW ASYNCOPI ; branch if async operation
00E7 341
```

```
00E7 342 :  
00E7 343 : synchronous operation first stall - set stall i/o status code  
00E7 344 :  
00E7 345 :  
00E7 346 SYNCOP: RMSSUC STALL,AP  
00EC 347 :  
00EC 348 :  
00EC 349 : save stack size, registers, and stack (including return pc)  
00EC 350 :  
53 14 AB 5E C3 00EC 351 :  
00EC 352 CTXSAV: SUBL3 SP,IMP$L_SAVED_SP(R11),R3 ; get stack size  
00F1 353 :  
00F1 354 :  
00F1 355 : verify stack fits into asb  
00F1 356 :  
00F1 357 :  
00F1 358 ASSUME ASB$W_STKLEN EQ 0  
00F1 359 :  
02 62 53 B1 00F1 360 CMPW R3,(R2) ; does stack fit?  
02 A2 44 1A 00F4 361 BGTRU ERRBUG ; branch if bad  
52 53 B0 00F6 362 MOVW R3,ASB$W_STKSIZ(R2) ; save the size  
82 1C C0 00FA 363 ADDL2 #ASB$L_REGS,R2 ; get addr of register save area  
82 56 D0 00FD 364 MOVL R6,(R2)+ ; save r6  
82 57 7D 0100 365 MOVQ R7,(R2)+ ; save r7 & r8  
0103 366 :  
0103 367 :  
0103 368 : note: r9 saved as ast parameter  
0103 369 :  
0103 370 :  
62 82 5A 7D 0103 371 MOVQ R10,(R2)+ ; save r10 & r11  
62 6E 53 28 0106 372 MOVQ R3,(SP),(R2) ; copy the stack including  
010A 373 : saved R4 & R5  
010A 374 :  
010A 375 :  
010A 376 : set the bit in the IFAB/IRAB which indicates that this RMS thread is  
010A 377 : currently stalled. This bit is cleared within RMSSTALLAST, when the  
010A 378 : stalled RMS thread resumes.  
010A 379 :  
010A 380 :  
010A 381 ASSUME IFB$V_RMS_STALL EQ IRB$V_RMS_STALL  
010A 382 :  
010A 383 SSB #IFB$V_RMS_STALL,(R9) ; set rms stall bit in IRAB/IFAB  
010E 384 :  
010E 385 :  
010E 386 : if really there (just return)  
010E 387 :  
010E 388 :  
1C 6B 01 E4 010E 389 BBSC #IMP$V_AST,(R11),RETURN ; clear at ast level and branch  
0D 69 23 E0 0112 390 BBS #IRB$V_ASYNC,(R9),30$ ; branch if asynchronous i/o  
0116 391 :  
0116 392 ASSUME IRB$B_EFN EQ IFB$B_EFN  
0116 393 :  
53 0B A9 9A 0116 394 MOVZBL IRB$B_EFN(R9),R3 ; set event flag on which to wait  
07 12 011A 395 BNEQ 30$ ; branch if non-zero (not rah/wbh)  
011C 396 :  
011C 397 :  
011C 398 ASSUME IFB$V_ASYNC EQ IRB$V_ASYNC  
ASSUME IFB$V_ASYNCWAIT EQ IRB$V_ASYNCWAIT
```



```

04 00000000'9F 00 E5 011C 399
53 04 18 88 011C 400 BISB2 #<1a<IRBSV_ASYNC-BKP>>!!<1a<IRBSV_ASYNCWAIT-BKP>>,-
53 1E D0 011E 401 IRBSL_BKPBITS(R9) ; show waiting on async efn
0120 402 MOVL #IMPSC_ASYEFN,R3 ; and wait on it
0123 403 30$:
0123 404
0123 405 :++
0123 406
0123 407 : at non-ast level - re-enable asts
0123 408 : entry here from $wait with:
0123 409
0123 410 : ap = status
0123 411 : r8 = rab address
0123 412 : r4 = $wait type flag
0123 413 : r3 = efn
0123 414 :--
0123 415
0123 416 RM$ENBAST::
0123 417 BBCC #PIO$V_INHAST,@#PIO$GW_STATUS,ENBAST ; clear ast inhibit
012B 418
012B 419 :
012B 420 : branching if clear
012B 421 :
012B 422
50 5C D0 012B 423 SETSTS: MOVL AP,R0 ; restore status code
04 04 012E 424 RETURN: RET ; exit rms
F1 11 012F 425 ENBAST: $SETAST_S #1 ; must re-enable asts
0138 426 BRB -SETSTS
013A 427
013A 428 :
013A 429 : Not enough space in asb for stack. The bad stack size is in R3.
013A 430 :
013A 431
013A 432 ERRBUG:
013A 433 RMSTBUG FTL$_STKTOOBIG

```

```

0141 435      .SBTTL RMSSTALLAST - AST ENTRY POINT FOR I/O COMPLETE
0141 436
0141 437      :++
0141 438      :
0141 439      RMSSTALLAST: AST entry point for I/O complete
0141 440      RMSRAHWBHASt: For read ahead/write behind via ast
0141 441      RMSTHREADGO: With r9 already set (for multi buffering).
0141 442      :
0141 443      this routine is entered as a result of an ast delivery for i/o
0141 444      completion. its function is to restart the associated
0141 445      thread which stalled as a result of calling rm$stall. the
0141 446      following processing is performed:
0141 447      :
0141 448      1. checks for asts inhibited, and if so disables asts,
0141 449      redeclares the current ast, sets a flag to cause
0141 450      asts to be re-enabled, and exits.
0141 451      2. otherwise, restores r9 (ifab or irab address) from
0141 452      the ast parameter value, checking for a valid ifab
0141 453      or irab.
0141 454      3. the asb address is retrieved and the saved
0141 455      registers (r4-r11) and stack are restored.
0141 456      4. the user structure (fab or rab) is reprobbed.
0141 457      5. the indicators imp$l_saved_sp and imp$u_ast are set
0141 458      appropriately
0141 459      6. if this is a shared file the file lock
0141 460      is restored for the stream
0141 461      7. return is made to the routine that called rm$stall
0141 462      with nearly full context restored (r0-r3 and ap are
0141 463      destroyed, secondary user structures must be
0141 464      reprobbed, absolute stack addresses are different)
0141 465      :
0141 466      Calling sequence:
0141 467      :
0141 468      entered at rm$stallast via an ast.
0141 469      alternate entry at rm$rahwbhast for read ahead/write behind via ast
0141 470      alternate entry at rm$threadgo with r9 already set (for multi buffering).
0141 471      :
0141 472      Input Parameters:
0141 473      :
0141 474      astprm - the ifab or irab address
0141 475      (for rm$rahwbhast astprm = bdb address)
0141 476      :
0141 477      Implicit Inputs:
0141 478      :
0141 479      the contents of the ifab or irab and related structures.
0141 480      :
0141 481      Output Parameters:
0141 482      :
0141 483      r4-r11 contents before stall
0141 484      sp      addr of stack having same contents as before stall
0141 485      pc      restored to return in line after call to rm$stall
0141 486      r1-r3,ap destroyed
0141 487      r0      set to contents of 1st word of i/o status block
0141 488      :
0141 489      Implicit Outputs:
0141 490      :
0141 491      imp$u_ast      set

```



```

0141 492 :      imp$l_saved_sp      set appropriately for new stack
0141 493 :
0141 494 :      Completion Codes:
0141 495 :
0141 496 :      none
0141 497 :
0141 498 :      Side Effects:
0141 499 :
0141 500 :      running at ast level.
0141 501 :      secondary user structures require re-probing.
0141 502 :      absolute stack addresses different.
0141 503 :
0141 504 :--
0141 505 :

```

```
0141 507
0141 508 :++
0141 509 :
0141 510 : entry here via ast for rah/wbh io completion
0141 511 :
0141 512 :--
0141 513 :
0141 514 .ALIGN QUAD
0148 515 $ENTRY RMSRAHWBHA$T,^/^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>/
014A 516
07 54 04 AC D0 014A 517 MOVL 4(AP),R4 ; get bdb addr (astprm)
0A 0A A4 06 E3 014E 518 BBCS #BDB$V_AST_DCL,BDB$B_FLGS(R4),10$; set i/o done, branching
0153 519 ; if no one waiting
0153 520
04 AC 24 A4 D0 0153 521 MOVL BDB$L_WAIT(R4),4(AP) ; change astprm to irab
20 11 0158 522 BRB CHECKAST ; go join common code to restart
015A 523 ; stalled stream
04 015A 524 10$: RET ; dismiss ast
015B 525
015B 526 :++
015B 527 :
015B 528 : entry here via ast for recovery-unit io completion
015B 529 :
015B 530 :--
015B 531 :
015B 532 .ALIGN QUAD
0160 533 $ENTRY RMSRUSTALLAST,^/^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>/
00B3 30 0162 534 BSBW RMSCHKAST ; check for asts inhibited
0165 535
0165 536 :
0165 537 : (note this must be a bsbw and
0165 538 : must immediately follow the entry mask.)
0165 539 :
0165 540
0165 541 RUSTALLAST:
0165 542 $SETEF_S #IMP$C_ASYEFN ; set event flag
016E 543 SSB #PIO$V_INHAST,@#PIO$GW_STATUS ; disable ASTs again
04 0176 544 RET
0177 545
```



```
0177 547 :++
0177 548 :
0177 549 : entry here via normal i/o completion ast
0177 550 :
0177 551 :--
0177 552 :
0177 553 :
0178 554 :.ALIGN QUAD
017A 555 :$ENTRY RMSSTALLAST,^/^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>/
017A 556 CHECKAST:
009B 30 017A 557 BSBW RMSCHKAST ; check for asts inhibited
017D 558 :
017D 559 :
017D 560 : (note this must be a bsbw and
017D 561 : must immediately follow the entry mask.)
017D 562 :
017D 563 :
017D 564 :
017D 565 : See if we are within RU handler, if so, handle in RU ast routine
017D 566 :
017D 567 :
50 01 18 A9 C1 017D 568 ADDL3 IRB$L_ARGLST(R9),#1,R0 ; RU handler stall?
E1 13 0182 569 BEQL RUSTALLAST ; branch if so
0184 570 :
0184 571 RM$THREADGO::
0184 572 :
0184 573 STSTPT STALAST ; sets r9 = ifab or irab addr
018A 574 :
018A 575 :
018A 576 : clear the bit within the IRAB/IFAB indicating that this thread of RMS is
018A 577 : stalled, as it no longer is, and allow it to continue
018A 578 :
018A 579 :
018A 580 ASSUME IFB$V_RMS_STALL EQ IRB$V_RMS_STALL
018A 581 :
018A 582 CSB #IFB$V_RMS_STALL,(R9) ; clear rms stall bit in IRAB/IFAB
018E 583 :
018E 584 ASSUME IFB$L_ASBADDR EQ IRB$L_ASBADDR
018E 585 :
51 14 A9 D0 018E 586 MOVL IFB$L_ASBADDR (R9),R1 ; get asb addr
65 13 0192 587 BEQL ERRASB ; error if none
0194 588 :
0194 589 ASSUME IFB$V_BUSY EQ IRB$V_BUSY
0194 590 :
61 69 20 E1 0194 591 BBC #IRB$V_BUSY,(R9),ERRASB ; branch if stream not busy
50 02 A1 3C 0198 592 MOVZWL ASB$W_STKSIZ(R1),R0 ; get size of stack
51 1C C0 019C 593 ADDL2 #ASB$C_REGS,R1 ; move to register save area
56 81 D0 019F 594 :
57 81 7D 01A2 595 MOVL (R1)+,R6 ; restore r6
01A5 596 MOVQ (R1)+,R7 ; restore r7/r8
01A5 597 :
01A5 598 :
01A5 599 : note r9 already restored
01A5 600 :
01A5 601 :
14 5A 81 7D 01A5 602 MOVQ (R1)+,R10 ; restore r10/r11
AB 5E D0 01A8 603 MOVL SP,IMP$L_SAVED_SP(R11) ; save stack entry value
```

```
6E 5E 50 C2 01AC 604          SUBL2  R0,SP          ; allocate required size
    61 50 28 01AF 605          MOVCL  R0,(R1),(SP)    ; copy stack including return pc
    01B3 606
    01B3 607          ASSUME  IMP$W_RMSSTATUS EQ      0
    01B3 608
    6B 02 88 01B3 609          BISB2  #<1@IMP$V_AST>,(R11) ; set flag for at ast level
    01B6 610
    01B6 611          :++
    01B6 612          : restore the file lock mode
    01B6 613          :
    01B6 614          :
    01B6 615          :--
    01B6 616
    01B6 617          ASSUME  <IFB$C_BID&1> EQ      1
    01B6 618          ASSUME  <IRB$C_BID&1> EQ      0
    01B6 619          ASSUME  IFB$B_BID EQ      IRB$B_BID
    01B6 620
    0B 08 A9 E9 01B6 621          BLBC   IFB$B_BID(R9),10$ ; branch if irab
    13 69 37 E4 01BA 622          BBSC   #IFB$V_STALL_LOCK,(R9),30$ ; branch if stalled for lock and cle
    78 A9 D5 01BE 623          TSTL   IFB$L_SF$B_PTR(R9) ; is the file shared?
    0E 13 01C1 624          BEQL   30$ ; branch if not
    09 11 01C3 625          BRB     20$
    08 6A 37 E4 01C5 626 10$: BBSC   #IFB$V_STALL_LOCK,(R10),30$ ; branch if stalled for lock and cle
    78 AA D5 01C9 627          TSTL   IFB$L_SF$B_PTR(R10) ; is the file shared?
    03 13 01CC 628          BEQL   30$ ; branch if not
    FE2F' 30 01CE 629 20$: BSRW    RMS$RESTORE_LOCK ; restore previous lock mode
    01D1 630
    01D1 631          :+
    01D1 632          : reprobe user structure (user could have deleted it from ast or
    01D1 633          : async operation)
    01D1 634          :
    01D1 635          :
    01D1 636          :-
    01D1 637
    01D1 638          ASSUME  IFB$B_MODE EQ      IRB$B_MODE
    01D1 639          ASSUME  <IFB$C_BID&1> EQ      1
    01D1 640          ASSUME  <IRB$C_BID&1> EQ      0
    01D1 641          ASSUME  IFB$B_BID EQ      IRB$B_BID
    01D1 642
    0E 08 A9 E8 01D1 643 30$: BLBS   IFB$B_BID(R9),CHKFAB ; branch if ifab
    01D5 644
    01D5 645          : irab operation
    01D5 646          :
    01D5 647          :
    01D5 648
    01D5 649          ASSUME  IFB$B_MODE EQ      IRB$B_MODE
    01D5 650          ASSUME  RAB$C_BLN LE      FAB$C_BLN
    01D5 651
    01 68 91 01D5 652          IFNOWRT #RAB$C_BLN,(R8),ERRSTRUCT,IRB$B_MODE(R9)
    0E 13 01DE 653          CMPB   RAB$B_BID(R8),#RAB$C_BID ; it must be a rab
    01E3 654          BEQL   GETBACK ; branch if so
    01E3 655
    01E3 656          :
    01E3 657          : (it could be a forced disconnect, hence a fab)
    01E3 658          :
    01E3 659          :
    01E3 660          :
```



```
01E3 661 ; ifab operation
01E3 662 ;
01E3 663 ;
03 68 91 01E3 664 CHKFAB: IFNOWRT #FAB$C_BLN,(R8),ERRSTRUCT,IFB$B_MODE(R9)
OF 12 01EC 665 CMPB FAB$B_BID(R8),#FAB$C_BID ; it must be a fab
01EF 666 BNEQ ERRSTRUCT ; branch if ok.
01F1 667 ;
01F1 668 ;+
01F1 669 ;
01F1 670 ; set r0 to status from i/o status block and return to thread
01F1 671 ;
01F1 672 ; -
01F1 673 ;
01F1 674 GETBACK:
54 8E 7D 01F1 675 MOVQ (SP)+,R4 ; restore r4 and r5
01F4 676
01F4 677 ASSUME IRB$L_IOS EQ IFB$L_IOS
01F4 678
50 0C A9 3C 01F4 679 MOVZWL IRB$L_IOS(R9),R0 ; pick up i/o completion status
05 05 01F8 680 RSB ; restart thread
01F9 681
01F9 682
01F9 683 ;
01F9 684 ; handle errors
01F9 685 ;
01F9 686 ;
01F9 687 ;
01F9 688 ; no asb found in ifab/irab or stream not busy
01F9 689 ;
01F9 690
01F9 691 ERRASB:
01F9 692 RMSTBUG FTL$_NOASB
0200 693
0200 694 ;
0200 695 ; the user has been playing funny games with memory
0200 696 ;
0200 697
0200 698 ERRSTRUCT:
58 14 A9 1C C1 0200 699 ADDL3 #ASB$L_REGS,IFB$L_ASBADDR(R9),R8 ; point r8 into asb
0205 700
0205 701 ASSUME <ASB$C_BLN_FAB - ASB$L_REGS> GE FAB$C_BLN
0205 702 ASSUME FAB$C_BLN GE RAB$C_BLN
0205 703
0205 704 PUSHR #^M<R1,R2,R3> ; save regs clobbered by mov
0207 705 MOVCS #0,(SP),#0,#FAB$C_BLN,(R8) ; clear out fake fab/rab
020F 706 POPR #^M<R1,R2,R3> ; restore regs
0211 707 BRB GETBACK ; return to thread
0213 708
```



```
0213 710 .SBTTL RMSCHKAST - CHECK FOR ASTS INHIBITED
0213 711
0213 712 :++
0213 713
0213 714 : RMSCHKAST: Check for ASTs inhibited
0213 715 : RMSBLKFINCHK:
0213 716 : RMSCHKAST_ANY:
0213 717
0213 718 : This routine checks for asts inhibited, and if so disables
0213 719 : asts, redeclares the current ast, clears the flag
0213 720 : pio$y_inhast to cause asts to be reenabled when the
0213 721 : active non-ast code exits, and exits.
0213 722
0213 723 : If asts are not disabled, sets r9 to the value of the
0213 724 : ast parameter and checks that it is a valid ifab of
0213 725 : irab address, and returns to the caller.
0213 726
0213 727 : The RMSBLKFINCHK and RMSCHKAST_ANY entry points do not validate the AST
0213 728 : parameter.
0213 729
0213 730 : calling sequence
0213 731
0213 732 : BSBW RMSCHKAST
0213 733 : BSBW RMSBLKFINCHK
0213 734 : BSBW RMSCHKAST_ANY
0213 735
0213 736
0213 737 : Input Parameters:
0213 738
0213 739 : ap ast argument list address
0213 740
0213 741 : Implicit Inputs:
0213 742
0213 743 : it is assumed that rm$chkast was called via bsbw as
0213 744 : the first instruction of the ast routine.
0213 745
0213 746 : Output Parameters:
0213 747
0213 748 : If return is made to caller,
0213 749 : R9 = AST parameter, which is
0213 750 : ifab or irab address for RMSCHKAST, or
0213 751 : BLB address for RMSBLKFINCHK entry.
0213 752
0213 753 : Implicit outputs:
0213 754
0213 755 : may requeue the ast if currently inhibited.
0213 756
0213 757 : Condition Codes:
0213 758
0213 759 : none.
0213 760
0213 761 : Side Effects:
0213 762
0213 763 : asts may be disabled.
0213 764
0213 765 :--
0213 766
```



```
0213 767 .ALIGN QUAD
0218 768
0218 769 RMSCHKAST::
0218 770 MOVL 4(AP),R9 ; get ifab/irab address
14 00000000'9F 04 AC D0 E4 021C 771 BBSC #PIO$V_INHAST,a#PIO$GW_STATUS,DSBLAST ; branch if inhibited
0224 772
0224 773 ;
0224 774 ; o.k. to receive ast
0224 775 ; check r9 ifab or irab address for validity
0224 776 ;
0224 777
0224 778 ASSUME IFB$B_BID EQ IRB$B_BID
0224 779
0A 08 A9 91 0224 780 CMPB IRB$B_BID(R9),#IRB$C_BID; is it an irab?
0B 08 A9 13 0228 781 BEQL 10$ ; if so exit
0B 08 A9 91 022A 782 CMPB IFB$B_BID(R9),#IFB$C_BID; if not then it must be an ifab?
01 01 12 022E 783 BNEQ 20$ ; if not an ifab then we goofed
05 0230 784 10$: RSB ; exit
0231 785
0231 786 20$: RMSTBUG FTL$_BADASTPRM ; oops.
0238 787
0238 788 ;
0238 789 ; asts are inhibited
0238 790 ; disable asts and redeclare the current ast
0238 791 ;
0238 792
0238 793 DSBLAST:
0238 794 $TSTPT ASTDSA
023E 795 $SETAST_S #0 ; disable asts
51 8E 05 C3 0247 796 SUBL3 -#5,(SP)+,R1 ; compute ast address
024B 797
024B 798 ;
024B 799 ; (return pc - 3-byte bsw
024B 800 ; - 2-byte entry mask)
024B 801 ;
024B 802
024B 803 $DCLAST_S ASTADR=(R1),ASTPRM=R9; re-declare the ast
01 50 E9 0258 804 BLBC -R0,ERRAST
04 025B 805 RET ; and exit
025C 806
025C 807 ;
025C 808 ; no space to declare an ast
025C 809 ;
025C 810
025C 811 ERRAST: RMSTBUG FTL$_CANTDOAST
0263 812
0263 813 ;+
0263 814 ; Alternate entries.
0263 815 ;-
0263 816
0263 817 .ALIGN QUAD
0268 818 RMSBLKFINCHK:: ; Used in RMORELEAS.
0268 819 RMSCHKAST_ANY:: ; Note: do not validate ASTPRM.
0268 820 MOVL 4(AP),R9 ; get AST parameter.
C4 00000000'9F 04 AC D0 E4 026C 821 BBSC #PIO$V_INHAST,a#PIO$GW_STATUS,DSBLAST ; branch if inhibited
05 0274 822 RSB ; Return to caller if not inhibited.
0275 823
```

RMOSTALL
V04-000

STALL FOR I/O COMPLETION
RMSCHKAST - CHECK FOR ASTS INHIBITED

E 9

16-SEP-1984 00:39:27
5-SEP-1984 16:22:37

VAX/VMS Macro V04-00
[RMS.SRC]RMOSTALL.MAR;1

Page 19
(12)

0275 824 .END

RMO
V04

RMOSTALL
Symbol table

STALL FOR I/O COMPLETION

F 9

16-SEP-1984 00:39:27 VAX/VMS Macro V04-00
5-SEP-1984 16:22:37 [RMS.SRC]RMOSTALL.MAR;1

Page 20
(12)

```

$$PSECT_EP      = 00000000
$$ARGS          = 00000001
$$RMSTEST       = 0000001A
$$RMS_PBUGCHK   = 00000010
$$RMS_TBUGCHK   = 00000008
$$RMS_UMODE     = 00000004
$$T1            = 00000000
ASB$B_BID       = 00000008
ASB$C_BID       = 0000000D
ASB$C_BLN_FAB   = 00000160
ASB$C_BLN_FIX   = 00000030
ASB$L_ARGCST    = 0000000C
ASB$L_REGS      = 0000001C
ASB$W_STKLEN    = 00000000
ASB$W_STKSIZ    = 00000002
ASYNCOP         = 00000039 R      01
BDB$B_FLGS      = 0000000A
BDB$L_WAIT      = 00000024
BDB$V_AST_DCL   = 00000006
BKP             = 00000020
CHECKAST        = 0000017A R      01
CHKFAB         = 000001E3 R      01
CTXSAV          = 000000EC R      01
DSBLAST         = 00000238 R      01
ENBAST          = 0000012F R      01
ERRASB          = 000001F9 R      01
ERRAST          = 0000025C R      01
ERRBUG          = 0000013A R      01
ERRDME          = 00000032 R      01
ERRSTRUCT       = 00000200 R      01
FAB$B_BID       = 00000000
FAB$C_BID       = 00000003
FAB$C_BLN       = 00000050
FTLS_ASBALLFAIL = FFFFFFFF9
FTLS_BADASTPRM  = FFFFFFFF8
FTLS_CANTDOAST  = FFFFFFFF7
FTLS_NOASB      = FFFFFFFF5
FTLS_STKTOOBIG  = FFFFFFFFE
GETBACK         = 000001F1 R      01
IFB$B_BID       = 00000008
IFB$B_EFN       = 0000000B
IFB$B_MODE      = 0000000A
IFB$C_BID       = 0000000B
IFB$L_ASBADDR   = 00000014
IFB$L_IOS       = 0000000C
IFB$L_SFSB_PTR  = 00000078
IFB$V_ASYNC     = 00000023
IFB$V_ASYNCWAIT = 00000024
IFB$V_BUSY      = 00000020
IFB$V_RMS_STALL = 0000003A
IFB$V_STALL_LOCK = 00000037
IMP$C_ASYEFN    = 0000001E
IMP$L_SAVED_SP  = 00000014
IMP$V_AST       = 00000001
IMP$V_RUH       = 00000006
IMP$W_RMSSTATUS = 00000000
IRB$B_BID       = 00000008

```

```

IRB$B_EFN       = 0000000B
IRB$B_MODE      = 0000000A
IRB$C_BID       = 0000000A
IRB$L_ARGLST    = 00000018
IRB$L_ASBADDR   = 00000014
IRB$L_BKPBITS   = 00000004
IRB$L_IOS       = 0000000C
IRB$V_ASYNC     = 00000023
IRB$V_ASYNCWAIT = 00000024
IRB$V_BUSY      = 00000020
IRB$V_RMS_STALL = 0000003A
PIO$A_TRACE     = ***** X 01
PIO$G_PIOIMPA   = ***** X 01
PIO$G_STATUS    = ***** X 01
PIO$V_INHAST    = 00000000
RAB$B_BID       = 00000000
RAB$C_BID       = 00000001
RAB$C_BLN       = 00000044
RETURN          = 0000012E R      01
RMSBLKF INCHK   = 00000268 RG     01
RMSBUG          = ***** X 01
RMSCHKAST       = 00000218 RG     01
RMSCHKAST_ANY   = 00000268 RG     01
RMSENBA$T       = 00000123 RG     01
RMSGETBLK       = ***** X 01
RMSLOWER_LOCK   = ***** X 01
RMSRAHWB$AST    = 00000148 RG     01
RMSRESTORE_LOCK = ***** X 01
RMSRUSTALLAST   = 00000160 RG     01
RMSSTALL        = 000000B3 RG     01
RMSSTALLAST     = 00000178 RG     01
RMSSTALL_LOCK   = 000000A7 RG     01
RMS$THREADGO    = 00000184 RG     01
RMS$PENDING     = 00018009
RMS$STALL       = 00018001
RUSTALL         = 00000078 R      01
RUSTALLAST      = 00000165 R      01
SETEF$EFN       = 00000004
SETEF$NARGS     = 00000001
SET$TS         = 0000012B R      01
STALL           = 000000CC R      01
STALLAL         = 00000000 R      01
SYNCOP          = 000000E7 R      01
SYSS$CLREF      = ***** GX 01
SYSS$DCLAST     = ***** GX 01
SYSS$SETAST     = ***** GX 01
SYSS$SETEF      = ***** GX 01
SYSS$WAITFR     = ***** GX 01
TPT$L_ASTDSA    = ***** X 01
TPT$L_STALAST   = ***** X 01
TPT$L_STALL     = ***** X 01
TPT$L_STALLAL   = ***** X 01
TPT$L_STALLLOCK = ***** X 01

```


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

PSECT name	Allocation	PSECT No.	Attributes
ABS	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
RMSRMS0	00000275 (629.)	01 (1.)	PIC USR CON REL GBL NOSHR EXE RD NOWRT NOVEC QUAD
SABSS	00000000 (0.)	02 (2.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE

+-----+
! Performance indicators !
+-----+

Phase	Page faults	CPU Time	Elapsed Time
Initialization	35	00:00:00.10	00:00:01.26
Command processing	127	00:00:00.73	00:00:05.37
Pass 1	362	00:00:12.77	00:00:31.10
Symbol table sort	0	00:00:01.53	00:00:03.37
Pass 2	148	00:00:02.97	00:00:07.55
Symbol table output	13	00:00:00.12	00:00:00.34
Psect synopsis output	2	00:00:00.03	00:00:00.03
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	689	00:00:18.26	00:00:49.03

The working set limit was 1650 pages.

69965 bytes (137 pages) of virtual memory were used to buffer the intermediate code.

There were 70 pages of symbol table space allocated to hold 1219 non-local and 18 local symbols.

824 source lines were read in Pass 1, producing 16 object records in Pass 2.

39 pages of virtual memory were used to define 38 macros.

+-----+
! Macro library statistics !
+-----+

Macro library name	Macros defined
_\$255\$DUA28:[RMS.OBJ]RMS.MLB;1	17
_\$255\$DUA28:[SYS.OBJ]LIB.MLB;1	4
_\$255\$DUA28:[SYSLIB]STARLET.MLB;2	13
TOTALS (all libraries)	34

1406 GETS were required to define 34 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS\$:RMOSTALL/OBJ=OBJ\$:RMOSTALL MSRC\$:RMOSTALL/UPDATE=(ENH\$:RMOSTALL)+EXECML\$/LIB+LIB\$:RMS/LIB

0320

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY