

## Table of contents

2-	1	EMTPLS	-- Dispatch PLAS EMT's
3-	1	.CRRG	-- Create PLAS region
4-	1	PVTRGN	-- Create a private region
5-	1	SHRRGN	-- Create a shared region
6-	1	.ELRG	-- Eliminate a PLAS region
7-	1	ELMPVT	-- Eliminate a private region
8-	1	ELMSHR	-- Eliminate a shared region
9-	1	.CRAW	-- Create a virtual address window
10-	1	.MAP	-- Map virtual window to physical region
11-	1	.ELAW	-- Eliminate a virtual address window
12-	1	.UNMAP	-- Unmap a virtual window
13-	1	.GMCX	-- Get window mapping status
14-	1	FLRCB	-- Find local RCB for private named region
15-	1	FLCRCB	-- Find local RCB for shared region
16-	1	FGRCB	-- Find global RCB for shared named region
17-	1	GLRCB	-- Get a free local RCB
18-	1	GGRCB	-- Get a free global RCB
19-	1	INIRCB	-- Initialize a new RCB
20-	1	CHKATT	-- Check valid attachment to named region
21-	1	RELRCB	-- Release attachment to a region
22-	1	WCBMAP	-- Do actual window mapping
23-	1	WCBUMP	-- Do actual unmapping of a PLAS window
24-	1	GETUDB	-- Get address of user definition block
25-	1	CHKRCB	-- Check address of Region Control Block
26-	1	PLSERR	-- Report error on PLAS EMT
27-	1	SEGOUT	-- Outswap job regions
30-	1	IASEGS	-- Allocate memory for regions during inswap
31-	1	MAPPLS	-- Set up mapping for all PLAS windows
32-	1	MAPSEQ	-- Set up window mapping during an inswap
33-	1	SEGIN	-- Inswap job regions
36-	1	PLSXIT	-- Do PLAS cleanup on job exit
37-	1	PLSOFF	-- Do PLAS cleanup at job logoff
38-	1	FRERCB	-- Free memory and swap space for a region
39-	1	SEGFRE	-- Free memory space for a region
40-	1	SWPGET	-- Find space in region swap file
41-	1	SWPFRE	-- Free space in region swap file
42-	1	PLSINI	-- Initialize PLAS system

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

000000  
000000 062763

```

.TITLE TSPLAS -- Programmed Logical Address Space Support
.ENABLE LC
.DSABLE GBL
.ENABL AMA
-----
; This module contains the routines to implement PLAS support for TSX-Plus.
;
; Copyright (c) 1983, 1984, 1985.
; S&H Computer Systems, Inc.
; Nashville, Tennessee USA
; All rights reserved
;
.PSECT TSPLAS
TSPLAS: .RAD50 /PLS/
-----
; Global definitions
;
.GLOBL BEGIN, SEGOOT, IASEGS, EMTPLS, PLSXIT, PLSOFF, MAPPLS, PLSINI
-----
; Global references
;
.GLOBL LBASE, KPAR6, RC$INM, RC$FLG, RC$LEN, RC$$SZ, DVRHC
.GLOBL RC$BAS, Q. BLKN, Q. PAR, Q. COMP, SEGCHN, GETSYQ
.GLOBL Q. WCNT, SYQID, CS$ERR, CS$EOF, EM$SSE, OUTBSY, OUTSWP
.GLOBL INBSY, RC$BLK, RC$USE, KEYRCB, KEYPAR
.GLOBL LBASE, LNBLKS, VPLAS, $INDRN, LSW5
.GLOBL EMTBLK, BADEMT, RCBBAS, RCBEND, SETERR, R. GSIZ, TRYMEM
.GLOBL VSWPFL, LNSBLK, R. GID, R. GSTS, EMTXIT
.GLOBL LSW6, RS. CRR, TRYPLS, INSFIN, TRYRGN, PRIVC2, P2$CGR
.GLOBL SR$PX, SR$PAR, SR$PDR, SR$FLG, SR. MOR, WC$MAP, WC. MAP, WC. FST
.GLOBL MEMXPB, URO, WCBBAS, WC$RCB, WC$$SZ, WCBEND, VALADW
.GLOBL WC$SIZ, WC$PAR, WC$NPR, WC$LEN, WC$VLD, WC$VHI, WC$TRP
.GLOBL W. NID, W. NBAS, W. NSIZ, WS. CRW, WS. ELW, WS. UNM, WS. OVR
.GLOBL W. NSTS, WS. MAP, NUMWCB, W. NOFF, W. NLEN, UPDRO, UPARO
.GLOBL RPDR, RPAR, SETMAP, WC$OFF, W. NRID, CXTAG, CORUSR
.GLOBL RC$PAG, LMEMIN, VCORTM, LMING, LQUAN, RS. NEW, RS. NAL
.GLOBL RC$SFA, LCXPAR, FREMEM, CUPARO, CUPDRO, RC$OFF
.GLOBL RS. GBL, RS. CGR, RS. PVT, R. NAME, RC$NAM, RC$EXI, RC$GBL
.GLOBL RC$PVT, SHRRCB, SHRRCN, RC$EXC, RC$DOWN, RS. AGE, RC$AEP
.GLOBL RS. EXI, RC$AGE, SWPFRC, SCPGET, SP$CMD, SP$JOB, SP$DW1
.GLOBL SA$RGN, S$WFM, QNSPND, CHKABT, RC$CNT
.GLOBL RS. EGR, RS. UNM, RC$LCG, RC$UNM
.GLOBL RS. DSP, RC$DSP, IDSFLG, $UDSPC, LSW11
.GLOBL RDDR, UDDRO, CUDDRO, RDAR, UDARO, CUDARO
-----
;
; Macro to print an error message when a system crash occurs.
;
; Arguments:
; MSG = Name of error message to print.
; ARG = (Optional) argument value to display with error message.
;
.GLOBL DIEMSG, DIEARG, SYSHLT
.MACRO DIE MSG, ARG

```

```

58          MOV      MSG, @#DIEMSG
59          .IF      NB, ARG
60          MOV      ARG, @#DIEARG
61          .ENDC
62          CALL     @#SYSHLT
63          .ENDM    DIE
64          ;
65          ; Macro to call a system overlay
66          ;
67          .MACRO   OCALL   ENTADD
68          .IF      B, ENTADD
69          .ERROR   ;OCALL without entry address
70          .ENDC
71          CALL     OVRHC
72          .WORD    ENTADD
73          .ENDM    OCALL
74          ;
75          -----
76          ; Symbolic equates for error codes
77          ;
78          000000   EC0    =      0
79          000001   EC1    =      1
80          000002   EC2    =      2
81          000003   EC3    =      3
82          000004   EC4    =      4
83          000005   EC5    =      5
84          000006   EC6    =      6
85          000007   EC7    =      7
86          000010   EC10   =     10
87          000011   EC11   =     11
88          000012   EC12   =     12
89          000014   EC14   =     14
90          000015   EC15   =     15
91          000020   EC20   =     20
92          000021   EC21   =     21
93          ;
94          -----
95          ; Local data areas
96          ;
97          000002   000000   SCBRCB: .WORD    0      ;Address of region control block
98          000004   000000   SCBBAS: .WORD    0      ;Memory address for region inswap/outswap
99          000006   000000   SCBLEN: .WORD    0      ;Number of 64-byte blocks to read/write
100         000010   000000   SCBBLK: .WORD    0      ;Block number in region swap file
101         000012   000000   SSMAP:  .WORD    0      ;Address of base of PLAS swap file free block bit tbl

```

```

1
2
3
4
5
6
7 000014 113700 000000G
8 000020 020027 000007
9 000024 103402
10 000026 000137 000000G
11 000032 006300
12 000034 000170 000040'
13
14
15
16 000040 000056'
17 000042 001510'
18 000044 001702'
19 000046 002564'
20 000050 002320'
21 000052 002630'
22 000054 002702'
23
24 000007
    
```

```

.SBTTL EMTPLS -- Dispatch PLAS EMT's
-----
; EMTPLS is jumped to from TSEMT when a PLAS EMT is executed.
; It dispatches control to the appropriate routine based on the
; EMT sub-function code.
;
EMTPLS: MOVB EMTBLK,RO ;Get sub-function code
        CMP RO,#MXPLAS ;Make sure it's not too big?
        BLD 1$ ;Br if ok
        JMP BADEMT ;Invalid sub-function code
1$: ASL RO ;Convert to word-table index
    JMP @PLSJMP(RO) ;Dispatch to processing routine
;
; PLAS EMT jump vector
;
PLSJMP: .WORD CRRG ; 0 .CRRG
        .WORD ELRG ; 1 .ELRG
        .WORD CRAW ; 2 .CRAW
        .WORD ELAW ; 3 .ELAW
        .WORD MAP ; 4 .MAP
        .WORD UNMAP ; 5 .UNMAP
        .WORD GMCX ; 6 .GMCX
;
MXPLAS = <.-PLSJMP>/2
    
```

.CRRG -- Create PLAS region

```

1          .SBTTL .CRRG -- Create PLAS region
2          ;-----
3          ; The .CRRG EMT is used to create a new PLAS region.
4          ;
5 000056   CRRG:
6          ;
7          ; Set up pointer to user's Region Definition Block (in R4)
8          ;
9 000056   004737   004632'   CALL   GETUDB           ;Get addr of user's RDB in R4
10         ;
11        ; Initialize status flags in user's Region Definition Block (RDB)
12        ;
13 000062   106564   000000G   MFPD   R.GSTS(R4)       ;Get status flags from user's RDB
14 000066   042716   000000C   BIC   #<RS.CRR!RS.UNM!RS.NAL!RS.NEW>,(SP) ;Clear some status flags
15 000072   011602                   MOV   (SP),R2          ;Save the status flags
16 000074   106664   000000G   MTPD   R.GSTS(R4)       ;Store flags back into RDB
17        ;
18        ; If this is a request to create a D-space region, see if it is allowed
19        ;
20 000100   113701   000000G   MOVB   CORUSR,R1       ;Get job index number
21 000104   032702   000000G   BIT   #RS.DSP,R2       ;Is this a D-space request?
22 000110   001416                   BEQ   1$                ;Skip test if not
23 000112   105737   000000G   TSTB   IDSFLG          ;Does machine support separate D-space?
24 000116   001003                   BNE   2$                ;Br if yes
25 000120   012700   000020   MOV   #EC20,R0          ;If not, say machine can't do it
26 000124   000406                   BR   3$                 ;Go abort
27 000126   032761   000000G 000000G 2$:   BIT   #UDSPC,LSW11(R1) ;Has job enabled separate I- and D-space?
28 000134   001004                   BNE   1$                ;OK, D-space PLAS region is allowed
29 000136   012700   000021   MOV   #EC21,R0          ;If not, say it hasn't been enabled yet
30 000142   000137   004764'   3$:   JMP   PLSERR          ;abort request
31        ;
32        ; Determine if we are creating a private or a shared region
33        ;
34 000146   032702   000000C   1$:   BIT   #<RS.GBL!RS.CGR>,R2 ;Creating a named region?
35 000152   001411                   BEQ   PVTRGN            ;Br if not
36 000154   032761   000000G 000000G   BIT   #INDRN,LSW5(R1) ;Is IND running?
37 000162   001005                   BNE   PVTRGN            ;Force its named regions to be private
38 000164   032702   000000G   BIT   #RS.PVT,R2       ;Private named region?
39 000170   001002                   BNE   PVTRGN            ;Br if yes
40 000172   000137   000666'   JMP   SHRRGN           ;Go create shared region

```

```

1          .SBTTL  PVTRGN -- Create a private region
2          ;-----
3          ; Create a region that is private to the creating job.
4          ; Note, the region may be either a named ("global") region or an unnamed
5          ; PLAS region.
6          ;
7          ; Inputs:
8          ;   R2 = Status flags from user's Region Definition Block.
9          ;   R4 = Pointer to RDB.
10         ;
11         000176 PVTRGN:
12         ;
13         ; See if this is a named or unnamed region.
14         ;
15         000176 032702 000000C      BIT    #<RS.GBL!RS.CGR>,R2 ;Creating a named region?
16         000202 001416              BEQ    15$                ;Br if not
17         ;
18         ; This is a named region.
19         ; Try to find an existing local RCB with the same name.
20         ;
21         000204 004737 003222'      CALL   FLRCB          ;Look for local RCB with the same name
22         000210 103404              BCS    2$                ;Br if not found
23         ;
24         ; The requested region already exists.
25         ;
26         000212 004737 003722'      CALL   CHKATT        ;Make sure attachment is valid
27         000216 000137 000424'      JMP    13$           ;Return info to user and exit
28         ;
29         ; We want to create a new region.
30         ; See if that is allowed.
31         ;
32         000222 032702 000000G      2$:   BIT    #RS.CGR,R2      ;Can we create a new region?
33         000226 001004              BNE    15$                ;Br if yes
34         000230 012700 000012      MOV    #EC12,R0      ;Return error code 12
35         000234 000137 004764'      JMP    PLSERR
36         ;
37         ; Get a free local RCB and initialize it.
38         ;
39         000240 004737 003476'      15$:  CALL   GLRCB          ;Get a free local RCB
40         ;
41         ; The address of a free Region Control Block is in R5.
42         ; Do some initialization of the RCB.
43         ;
44         000244 052765 000000C 000000G  BIS    #<RC$PVT!RC$EXC!RC$OFF>,RC$FLG(R5) ;Private region
45         ;
46         ; See if we can find free memory space for the region.
47         ;
48         000252 016503 000000G      MOV    RC$PAG(R5),R3 ;Get # 512-byte pages needed by region
49         000256 001004              BNE    18$                ;Br if specified size not zero
50         000260 012700 000010      MOV    #EC10,R0      ;Return error code 10
51         000264 000137 004764'      JMP    PLSERR
52         000270 010300      18$:  MOV    R3,R0
53         000272 004737 000000G      CALL   TRYMEM        ;See if we can get free memory for the request
54         000276 103520              BCS    4$                ;Br if unable to get a free memory area
55         ;
56         ; We got a free memory space for the region
57         ;

```

PVTRGN -- Create a private region

```

58 000300 072227 000003          ASH      #3,R2          ;Convert 512-byte block # to 64-byte block #
59 000304 010265 000000G        MOV      R2,RC$BAS(R5) ;Set physical memory base of region
60 000310 052765 000000G 000000G  BIS      #RC$INM,RC$FLG(R5);Set flag saying region is in memory
61                                     ;
62                                     ; Allocate space in region swap file for this region
63                                     ;
64 000316 105737 000000G        TSTB     VSWPFL        ;Is this a non-swapping system?
65 000322 001421                BEQ      6$           ;Br if non-swapping system
66 000324 010300                MOV      R3,R0        ;Get number of blocks needed
67 000326 004737 006670'        CALL     SWPGET        ;Try to allocate space in swap file
68 000332 103010                BCC      5$           ;Br if allocation was successful
69 000334 072027 000003          ASH      #3,R0        ;Convert largest swap file space to 64-byte
70 000340 010037 000000G        MOV      R0,URO        ;Return largest size to user in R0
71 000344 010502                MOV      R5,R2        ;Get address of RCB to R2 for SEGFRE
72 000346 004737 006610'        CALL     SEGFRE        ;Release memory allocated for the region
73 000352 000541                BR       14$          ;Return error to user
74 000354 010265 000000G        5$:     MOV      R2,RC$BLK(R5) ;Save swap file block number for region
75 000360 052765 000000G 000000G  BIS      #RC$SFA,RC$FLG(R5) ;Remember that space has been allocated
76                                     ;
77                                     ; Allocation is complete.
78                                     ;
79 000366 060361 000000G        6$:     ADD      R3,LNSBLK(R1) ;Increase # pages allocated to regions
80 000372 060361 000000G        ADD      R3,LMEMIN(R1) ;Increase total number of pages used by job
81 000376 106564 000000G        MFPD    R.GSTS(R4)    ;Get RDB status flags
82 000402 052716 000000G        BIS      #RS.NAL,(SP) ;Say this is a new allocation
83 000406 032716 000000G        BIT      #RS.GBL,(SP) ;Is this a named region?
84 000412 001402                BEQ      19$          ;Br if not
85 000414 052716 000000G        BIS      #RS.NEW,(SP) ;We created a new named region
86 000420 106664 000000G        19$:    MTPD    R.GSTS(R4) ;Store flags back into RDB
87                                     ;
88                                     ; Set some status flags in RCB
89                                     ;
90 000424 112765 000001 000000G 13$:    MOVVB   #1,RC$CNT(R5) ;Set attachment count to 1
91 000432 016500 000000G        MOV      RC$FLG(R5),R0 ;Get current RCB status flags
92 000436 052700 000000G        BIS      #RC$USE,R0   ;Say the RCB is active
93 000442 106564 000000G        MFPD    R.GSTS(R4)    ;Get status flags from RDB
94 000446 012602                MOV      (SP)+,R2
95 000450 032702 000000G        BIT      #RS.AGE,R2   ;Automatic elimination wanted?
96 000454 001402                BEQ      7$           ;Br if not
97 000456 052700 000000G        BIS      #RC$AEP,R0   ;Set automatic-elimination-pending flag
98 000462 032702 000000G        7$:     BIT      #RS.EXI,R2 ;Eliminate region on exit?
99 000466 001402                BEQ      8$           ;Br if not
100 000470 052700 000000G       BIS      #RC$EXI,R0   ;Set flag
101 000474 032702 000000G       8$:     BIT      #RS.DSP,R2 ;Is this a D-space region?
102 000500 001402                BEQ      20$          ;Br if not, I-space is default
103 000502 052700 000000G       BIS      #RC$DSP,R0   ;If so, remember for mapping time
104 000506 010065 000000G       20$:    MOV      R0,RC$FLG(R5) ;Store new flags back into RCB
105                                     ;
106                                     ; Return information to user's region definition block
107                                     ;
108 000512 010546                MOV      R5,-(SP)     ;Return address of RCB to user
109 000514 106664 000000G       MTPD    R.GID(R4)
110 000520 106564 000000G       MFPD    R.GSTS(R4)    ;Get old status flags from RDB
111 000524 052716 000000G       BIS      #RS.CRR,(SP) ;Set created-region flag
112 000530 106664 000000G       MTPD    R.GSTS(R4)    ;Store back into RDB
113 000534 000137 000000G       JMP      EMTXIT      ;Finished with EMT
114

```

PVTRGN -- Create a private region

```

115 ; Cannot get a free memory region large enough for region.
116 ; Determine the size of the largest region we will allow to be created.
117 ;
118 000540 105737 000000G 4$: TSTB VSWPFL ; Is this a non-swapping system?
119 000544 001440 BEQ 12$ ; Br if yes -- we can't consolidate memory
120 000546 004737 000000G CALL TRYPLS ; Determine # user pages we could get
121 000552 010046 MOV RO, -(SP) ; Save free memory space
122 000554 013700 000000G MOV VPLAS, RO ; Get # blocks allocated for PLAS swap file
123 000560 005200 INC RO ; Get more than total possible size
124 000562 004737 006670' CALL SWPGET ; See how big largest swap file region is
125 000566 012602 MOV (SP)+, R2 ; Get free memory space size
126 000570 020002 CMP RO, R2 ; Save smaller of free memory or swap region
127 000572 101401 BLOS 10$
128 000574 010200 MOV R2, RO
129 ;
130 ; We now know the size of the largest possible region that
131 ; we can allocate (Size in terms of 512-blocks is now in RO).
132 ; See if it is possible to satisfy this request.
133 ;
134 000576 020300 10$: CMP R3, RO ; Is request larger than largest possible?
135 000600 101022 BHI 12$ ; Br if yes -- Can't possibly do allocation
136 ;
137 ; It is possible to allocate this region, but we will have to do
138 ; some job swapping to collect free memory.
139 ; Go ahead and allocate space for the region in the region swap file.
140 ;
141 000602 010300 MOV R3, RO ; Get # blocks needed for the region
142 000604 004737 006670' CALL SWPGET ; Allocate space in the swap file
143 000610 103416 BCS 12$ ; This branch should never be taken
144 000612 010265 000000G MOV R2, RC$BLK(R5) ; Save block number for region in swap file
145 000616 052765 000000C 000000G BIS #<RC$USE!RC$SFA>, RC$FLG(R5) ; Region active and swap space
146 ;
147 ; Swap job to expand memory and allocate space for region
148 ;
149 000624 066100 000000G ADD LNSBLK(R1), RO ; Add num blocks already used by plas regions
150 000630 010061 000000G MOV RO, LNSBLK(R1) ; This is total space for all regions
151 000634 066100 000000G ADD LNBLKS(R1), RO ; Add space used by base portion of job
152 000640 004737 000000G CALL MEMXPN ; Do job swapping to expand memory space
153 000644 000667 BR 13$ ; Go finish up
154 ;
155 ; We are unable to satisfy the request.
156 ; Return largest possible region to user in RO.
157 ;
158 000646 072027 000003 12$: ASH #3, RO ; Convert # 512-byte blocks to # 64-byte blocks
159 000652 010037 000000G MOV RO, URO ; Return size of largest region in RO
160 000656 012700 000007 14$: MOV #EC7, RO ; Return error code # 7
161 000662 000137 004764' JMP PLSERR

```

SHRRGN -- Create a shared region

```

1          .SBTTL  SHRRGN -- Create a shared region
2          ;-----
3          ; SHRRGN is called to create a shared memory region.
4          ;
5          ; Inputs:
6          ; R2 = Status flags from user's Region Definition Block.
7          ; R4 = Pointer to RDB.
8          ;
9          000666 SHRRGN:
10         ;
11         ; See if we are authorized to access global regions
12         ;
13         000666 032737 000000G 000000G      BIT    #P2$CGR,PRIVC2 ;Are we authorized for global regions?
14         000674 001006                      BNE    23$          ;Br if yes
15         000676 012700 000007              MOV    #EC7,R0       ;Return error code 7 if not
16         000702 005037 000000G            CLR    URO          ;Max size of region = 0
17         000706 000137 004764'            JMP    PLSERR
18         ;
19         ; Try to find a local RCB that is connected to a shared region with
20         ; the same name.
21         ;
22         000712 004737 003322'            23$:   CALL    FLCRCB      ;Look for local RCB connected to global
23         000716 103406                      BCS    21$          ;Br if cannot find local RCB
24         000720 016305 000000G            MOV    RC$BLK(R3),R5 ;Get pointer to associated global RCB
25         000724 004737 003722'            CALL    CHKATT      ;See if we can attach to this region
26         000730 000137 001424'            JMP    20$          ;Found local RCB
27         ;
28         ; We cannot find a local RCB that is attached to the region.
29         ; See if there is a global RCB for the region.
30         ;
31         000734 004737 003414'            21$:   CALL    FGRCB      ;Look for global RCB for region
32         000740 103424                      BCS    16$          ;Br if can't find global RCB either
33         ;
34         ; We found a global RCB for the region but do not have a local RCB.
35         ; Create a local RCB for the region.
36         ;
37         000742 004737 003722'            CALL    CHKATT      ;See if we can attach to this region
38         000746 010546                      MOV    R5,-(SP)     ;Save address of global RCB
39         000750 004737 003476'            CALL    GLRCB      ;Get a free local RCB (R5 = address)
40         000754 010503                      MOV    R5,R3       ;Get address of local RCB
41         000756 012605                      MOV    (SP)+,R5    ;Get address of global RCB
42         000760 052763 000000C 000000G      BIS    #<RC$GBL!RC$EXC!RC$LCG>,RC$FLG(R3) ;Init local flags
43         000766 005763 000000G            TST    RC$LEN(R3)  ;Was specified region size 0?
44         000772 001165                      BNE    4$          ;Br if not
45         000774 016563 000000G 000000G      MOV    RC$LEN(R5),RC$LEN(R3) ;Copy length info from global RCB
46         001002 016563 000000G 000000G      MOV    RC$PAG(R5),RC$PAG(R3)
47         001010 000556                      BR     4$          ;Go transfer info to local RCB
48         ;
49         ; The named region does not now exist.
50         ; See if we want to create a new one.
51         ;
52         001012 032702 000000G            16$:   BIT    #RS.CGR,R2 ;Can we create a new region?
53         001016 001004                      BNE    17$          ;Br if yes
54         001020 012700 000012              MOV    #EC12,R0    ;Return error code 12
55         001024 000137 004764'            JMP    PLSERR
56         ;
57         ; We want to create a new region.

```

```

58 ; Get a free local and global RCB.
59 ;
60 001030 004737 003476' 17$: CALL GLRCB ;Get a local RCB (R5 = address)
61 001034 010503 ; MOV R5,R3 ;Get address of local RCB
62 001036 004737 003542' ; CALL GGRCB ;Get a global RCB (R5 = address)
63 001042 005765 000000G ; TST RC$LEN(R5) ;Was specified region size 0?
64 001046 001004 ; BNE 2$ ;Br if not
65 001050 012700 000010 ; MOV #EC10,R0 ;Return error code 10
66 001054 000137 004764' ; JMP PLSERR
67 ;
68 ; R3 = Address of free local RCB.
69 ; R5 = Address of free global RCB.
70 ; Initialize the RCB's.
71 ;
72 001060 052765 000000C 000000G 2$: BIS #<RC$GBL!RC$EXC>,RC$FLG(R5) ;Initialize status flags
73 001066 052763 000000C 000000G ; BIS #<RC$GBL!RC$EXC!RC$LCG>,RC$FLG(R3) ;Init local flags
74 001074 032702 000000G ; BIT #RS.AGE,R2 ;Is automatic elimination wanted?
75 001100 001403 ; BEQ 24$ ;Br if not
76 001102 052765 000000G 000000G ; BIS #RC$AEP,RC$FLG(R5) ;Remember automatic elim pending
77 001110 032702 000000G 24$: BIT #RS.DSP,R2 ;Is this to be a D-space region?
78 001114 001403 ; BEQ 5$ ;Br if not
79 001116 052765 000000G 000000G ; BIS #RC$DSP,RC$FLG(R5) ;Remember to use D-space mapping
80 ;
81 ; Determine how much memory space can possibly be allocated for
82 ; a shared region.
83 ;
84 001124 113701 000000G 5$: MOV#B CORUSR,R1 ;Get current job index number
85 001130 105737 000000G ; TSTB VSWPFL ;Is this a swapping system?
86 001134 001006 ; BNE 7$ ;Br if yes
87 ;
88 ; Do memory allocation for non-swapping system
89 ;
90 001136 016500 000000G ; MOV RC$PAG(R5),R0 ;Get requested # of pages
91 001142 004737 000000G ; CALL TRYMEM ;See if a large enough area is available
92 001146 103550 ; BCS 10$ ;Br if not
93 001150 000412 ; BR 13$
94 ;
95 ; Do memory allocation for swapping system
96 ;
97 001152 004737 000000G 7$: CALL TRYPLS ;Determine largest possible PLAS region
98 001156 026500 000000G ; CMP RC$PAG(R5),R0 ;Do we want more than is possible?
99 001162 101142 ; BHI 10$ ;Br if yes
100 001164 016500 000000G ; MOV RC$PAG(R5),R0 ;Get requested number of pages
101 001170 004737 000000G ; CALL TRYRGN ;See if area is available now
102 001174 103405 ; BCS 8$ ;Br if not available now
103 ;
104 ; We were able to allocate the region.
105 ; Set up information in the Region Control Block.
106 ;
107 001176 072227 000003 13$: ASH #3,R2 ;Convert 512-byte page # to 64-byte block #
108 001202 010265 000000G ; MOV R2,RC$BAS(R5) ;Remember base of allocated area
109 001206 000446 ; BR 15$
110 ;
111 ; The requested memory space is not currently available.
112 ;
113 001210 005702 8$: TST R2 ;Is it possible to allocate that much space?
114 001212 001526 ; BEQ 10$ ;Br if not (R0 contains largest space poss.)

```

SHRRGN -- Create a shared region

```

115 ;
116 ; It is possible to get the requested space but we will have to
117 ; do some job swapping to consolidate the free space.
118 ;
119 001214 052765 000000G 000000G BIS #RC$USE,RC$FLG(R5) ;Remember this RCB is in use
120 001222 005065 000000G CLR RC$BAS(R5) ;Say no memory allocated yet
121 001226 OCALL SWPFRG ;Outswap any jobs in area we want
122 001234 010546 MOV R5,-(SP) ;Save address of RCB
123 001236 OCALL SCPGET ;Get a swap command packet
124 001244 112765 000000G 000000G MOVB #SA$RGN,SP$CMD(R5);Set command for swapper
125 001252 110165 000000G MOVB R1,SP$JOB(R5) ;Set our job number
126 001256 011665 000000G MOV (SP),SP$DW1(R5) ;Set address of RCB in SCP
127 001262 012605 MOV (SP)+,R5 ;Get back address of RCB
128 ;
129 ; Suspend our job until the job swapping is completed
130 ;
131 001264 012700 000000G 14$: MOV #S$WFM,R0 ;Waiting-for-memory state
132 001270 004737 000000G CALL QNSPND ;Suspend job till swapping completes
133 001274 004737 000000G CALL CHKABT ;See if we were aborted while asleep
134 001300 016500 000000G MOV RC$BAS(R5),R0 ;Was memory allocated for the region?
135 001304 001767 BEQ 14$ ;Br if not
136 001306 020027 177777 CMP R0,#-1 ;Was memory actually allocated by swapper?
137 001312 001004 BNE 15$ ;Br if yes
138 001314 042765 000000G 000000G BIC #RC$USE,RC$FLG(R5) ;Say RCB is not in use
139 001322 000713 BR 7$ ;Go back and try the whole process again
140 ;
141 ; We have finished allocating space for the region
142 ;
143 001324 052765 000000C 000000G 15$: BIS #<RC$INM!RC$USE>,RC$FLG(R5);Region in memory & in use
144 001332 106564 000000G MFPD R.GSTS(R4) ;Get RDB status flags
145 001336 052716 000000C BIS #<RS.NEW!RS.NAL>,(SP) ;Say new region was created
146 001342 106664 000000G MTPD R.GSTS(R4) ;Store flags back into RDB
147 ;
148 ; Transfer some information from global RCB to local RCB
149 ;
150 001346 052763 000000C 000000G 4$: BIS #<RC$INM!RC$USE>,RC$FLG(R3) ;Set flags in local RCB too
151 001354 106564 000000G MFPD R.GSTS(R4) ;Get flags from user's RDB
152 001360 032726 000000G BIT #RS.EXI,(SP)+ ;Is elimination on program exit wanted?
153 001364 001403 BEQ 25$ ;Br if not
154 001366 052763 000000G 000000G BIS #RC$EXI,RC$FLG(R3) ;If yes, set automatic elimination flag
155 001374 032765 000000G 000000G 25$: BIT #RC$DSP,RC$FLG(R5) ;Does global region use D-space?
156 001402 001403 BEQ 6$ ;Br if not
157 001404 052763 000000G 000000G BIS #RC$DSP,RC$FLG(R3) ;If yes, remember to map local thru D-space
158 001412 016563 000000G 000000G 6$: MOV RC$BAS(R5),RC$BAS(R3) ;Base memory page for region
159 001420 010563 000000G MOV R5,RC$BLK(R3) ;Set pointer to global RCB in local RCB
160 ;
161 ; Increment attachment counts
162 ;
163 001424 105763 000000G 20$: TSTB RC$CNT(R3) ;Are we already attached to local RCB?
164 001430 001004 BNE 22$ ;Br if yes
165 001432 005263 000000G INC RC$CNT(R3) ;Say local attachment count = 1
166 001436 005265 000000G INC RC$CNT(R5) ;Increment global attachment count
167 ;
168 ; Return information to user's Region Definition Block
169 ;
170 001442 010346 22$: MOV R3,-(SP) ;Return address of local RCB to user
171 001444 106664 000000G MTPD R.GID(R4)

```

SHRRGN -- Create a shared region

```

172 001450 106564 000000G      MFPD  R.GSTS(R4)      ;Get old status flags from RDB
173 001454 052716 000000G      BIS   #RS.CRR,(SP)    ;Set created-region flag
174 001460 106664 000000G      MTPD  R.GSTS(R4)      ;Store back into RDB
175 001464 000137 000000G      JMP   EMTXIT         ;Finished with EMT
176                               ;
177                               ; We cannot allocate as large a region as was requested.
178                               ; Return the size of the largest possible region.
179                               ;
180 001470 072027 000003      10#:  ASH   #3,RO      ;Convert # 512-byte pages to # 64-byte pages
181 001474 010037 000000G      MOV   RO,URO         ;Return largest region size in RO
182 001500 012700 000007      MOV   #EC7,RO        ;Return error code 7
183 001504 000137 004764'      JMP   PLSERR

```

.ELRG -- Eliminate a PLAS region

```

1          .SBTTL .ELRG -- Eliminate a PLAS region
2          ;-----
3          ; .ELRG EMT eliminates a PLAS region.
4          ;
5 001510 004737 004632' ELRG:  CALL  GETUDB          ;Get addr of user's Region Definition Blk (R4)
6          ;
7          ; Get address of Region Control Block
8          ;
9 001514 106564 000000G          MFPD  R.GID(R4)      ;Get address of region control block
10 001520 012602          MOV   (SP)+,R2
11 001522 010200          MOV   R2,R0          ;Get RCB address to R0 for CHKRCB
12 001524 004737 004714'          CALL  CHKRCB          ;Make sure RCB address is valid
13          ;
14          ; Determine if this is a private or shared region.
15          ;
16 001530 032762 000000G 000000G BIT   #RC$LCG,RC$FLG(R2) ;Is this a shared region?
17 001536 001402          BEQ   ELMPVT          ;Br if not
18 001540 000137 001576'          JMP   ELMSHR          ;Eliminate a shared region

```

```

1          .SBTTL  ELMPVT -- Eliminate a private region
2          ;-----
3          ; Eliminate a private (non-shared) region.
4          ;
5          ; Inputs:
6          ;   R2 = Pointer to Region Control Block
7          ;   R4 = Pointer to User's Region Definition Block
8          ;
9 001544   ELMPVT:
10         ;
11         ; Always eliminate unnamed regions
12         ;
13 001544   032762   000000G 000000G       BIT    #RC#OBL,RC#FLG(R2) ;Is this a named region?
14 001552   001405                BEQ    2$          ;Br if not named
15         ;
16         ; This is a named region that is being eliminated.
17         ; Free named region if the RS.EGR flag is set
18         ;
19 001554   106564   000000G                MFPD   R.GSTS(R4)      ;Get status flags from user's RDB
20 001560   032726   000000G                BIT    #RS.EGR,(SP)+  ;Eliminate global region flag set?
21 001564   001426                BEQ    ELMCOM        ;Br if not -- Just detach from region
22         ;
23         ; We want to eliminate this region.
24         ;
25 001566   052762   000000G 000000G 2$:   BIS    #RC#AGE,RC#FLG(R2) ;Force elimination of the region
26 001574   000422                BR     ELMCOM        ;Now go detach from region

```

ELMSHR -- Eliminate a shared region

```

1          .SBTTL  ELMSHR -- Eliminate a shared region
2          ;-----
3          ; Eliminate a shared region.
4          ;
5          ; Inputs:
6          ; R2 = Pointer to local copy of RCB for region.
7          ; R4 = Pointer to User's Region Definition Block
8          ;
9 001576 016203 000000G  ELMSHR: MOV      RC$BLK(R2),R3  ;Get pointer to global RCB
10         ;
11         ; See if we are detaching from the region or eliminating it.
12         ;
13 001602 106564 000000G      MFPD   R.GSTS(R4)      ;Get status flags from user's RDB
14 001606 032726 000000G      BIT    #RS.EGR,(SP)+  ;Eliminate global region flag set?
15 001612 001413              BEQ    ELMCOM          ;Br if not
16         ;
17         ; We are to eliminate the region.
18         ; Make sure no one else is attached to region.
19         ;
20 001614 126327 000000G 000001  CMPB   RC$CNT(R3),#1  ;Are we the only attached job?
21 001622 001404              BEQ    1$           ;Br if yes
22 001624 012700 000014      MOV    #EC14,R0       ;Return error code 14
23 001630 000137 004764'     JMP    PLSERR
24 001634 052763 000000G 000000G 1$:  BIS    #RC$AGE,RC$FLG(R3) ;Force elimination of global region
25         ;
26         ; Call RELRGN to do the actual release
27         ;
28 001642 004737 004010'     ELMCOM: CALL  RELRGN          ;Release the region
29         ;
30         ; Clean out user's RDB
31         ;
32 001646 005046              CLR    -(SP)
33 001650 106664 000000G      MTPD   R.GID(R4)      ;Zero Region ID number
34 001654 005046              CLR    -(SP)      ;Zero region status word
35 001656 032762 000000G 000000G  BIT    #RC$UNM,RC$FLG(R2) ;Did we unmap any windows?
36 001664 001402              BEQ    9$           ;Br if not
37 001666 052716 000000G      BIS    #RS.UNM,(SP)  ;Report to user if we did
38 001672 106664 000000G 9$:  MTPD   R.GSTS(R4)      ;Return status word
39         ;
40         ; Finished
41         ;
42 001676 000137 000000G      JMP    EMTXIT

```

.CRAW -- Create a virtual address window

```

1          .SBTTL .CRAW -- Create a virtual address window
2          ;-----
3          ; The .CRAW EMT is used to create a PLAS virtual address window.
4          ;
5          001702 004737 004632' CRAW: CALL GETUDB ;Get address of window def block to R4
6          ;
7          ; Initialize status flags in W.NSTS
8          ;
9          001706 106564 000000G MFPD W.NSTS(R4) ;Get current status flags
10         001712 042716 000000C BIC #WS.ELW!WS.UNM!WS.CRW,(SP) ;Clear misc flags
11         001716 106664 000000G MTPD W.NSTS(R4) ;Return flags to user
12         ;
13         ; Get and check the APR value
14         ;
15         001722 106514 MFPD (R4) ;Get APR value (high byte)
16         001724 012602 MOV (SP)+,R2
17         001726 105002 CLRB R2 ;Clear low-byte
18         001730 010200 MOV R2,R0 ;Save word with APR number in high-byte
19         001732 000302 SWAB R2 ;Get APR number to low byte
20         001734 020227 000007 CMP R2,#7 ;APR must be in the range 0 to 7
21         001740 101070 BHI 2$ ;Br if invalid APR number
22         ;
23         ; Find a free window control block
24         ;
25         001742 012705 000000G MOV #WCBBAS,R5 ;Point to 1st WCB
26         001746 052700 000001 BIS #1,R0 ;First WCB is # 1
27         001752 005765 000000G 1$: TST WC$SIZ(R5) ;Is this WCB in use?
28         001756 001412 BEQ 3$ ;Br if not
29         001760 005200 INC R0 ;Keep track of the WCB number
30         001762 062705 000000G ADD #WC$$SZ,R5 ;Point to next WCB
31         001766 020527 000000G CMP R5,#WCBEND ;Have we checked all WCB's?
32         001772 103767 BLD 1$ ;Br if not
33         ;
34         ; Error -- There are no free window control blocks
35         ;
36         001774 012700 000001 MOV #EC1,R0 ;Return error code 1
37         002000 000137 004764' JMP PLSERR
38         ;
39         ; We found a free WCB (its address is in R5)
40         ; Init some cells
41         ; Return ID number to user
42         ;
43         002004 112765 177777 000000G 3$: MOVB #-1,WC$TRP(R5) ;Say not associated with any fast maps yet
44         002012 010046 MOV R0,-(SP) ;Push WCB id number and APR number
45         002014 106664 000000G MTPD W.NID(R4) ;Return to user
46         002020 106564 000000G MFPD W.NOFF(R4) ;Get specified offset into region
47         002024 012665 000000G MOV (SP)+,WC$OFF(R5) ;And set into wcb
48         002030 106564 000000G MFPD W.NRID(R4) ;Get region identifier
49         002034 012600 MOV (SP)+,R0 ; from user wdb
50         002036 004737 004714' CALL CHKRCB ;Verify that it is valid
51         002042 010065 000000G MOV R0,WC$RCB(R5) ;And set into wcb
52         ;
53         ; Set up information about base virtual address of the window
54         ;
55         002046 006302 ASL R2 ;Get 2*APR number
56         002050 110265 000000G MOVB R2,WC$PAR(R5) ;Save APR index
57         002054 072227 000014 ASH #12.,R2 ;Convert APR # to virtual address

```

.CRAW -- Create a virtual address window

```

58 002060 010265 000000G      MOV      R2,WC$VLD(R5)      ; Save in WCB
59 002064 010246              MOV      R2,-(SP)          ; Return virtual address to user
60 002066 106664 000000G      MTPD    W.NBAS(R4)
61                               ;
62                               ; Set up information about window size
63                               ;
64 002072 106564 000000G      MFPD    W.NSIZ(R4)        ; Get window size (64-byte units)
65 002076 012603              MOV      (SP)+,R3
66 002100 001410              BEQ     2$                ; Not legal to have 0 size
67 002102 000241              CLC
68 002104 006002              ROR     R2                ; Convert base virtual addr to 64-byte units
69 002106 072227 177773      ASH     #-5,R2
70 002112 060302              ADD     R3,R2            ; Get address above top of region (64-byte u)
71 002114 020227 002000      CMP     R2,#2000         ; Is it greater than 64Kb?
72 002120 101404              BLOS   4$                ; Br if not too big
73 002122 012700 000000      2$:    MOV     #ECO,R0      ; Return error code 0
74 002126 000137 004764'     JMP     PLSERR
75 002132 010365 000000G      4$:    MOV     R3,WC$SIZ(R5) ; Set size of window (64-byte units)
76 002136 072227 000006      ASH     #6,R2            ; Get virtual address of top of window
77 002142 005302              DEC     R2                ; Get address of last byte in window
78 002144 010265 000000G      MOV     R2,WC$VHI(R5)    ; Set high virtual address for window
79                               ;
80                               ; See if new window overlaps with any existing windows
81                               ;
82 002150 106564 000000G      MFPD    W.NSTS(R4)        ; Get user's window status word
83 002154 012603              MOV     (SP)+,R3
84 002156 052703 000000G      BIS     #WS.CRW,R3       ; Set in successful-creation flag
85 002162 032703 000000G      BIT     #WS.OVR,R3       ; Should we permit windows to overlap?
86 002166 001044              BNE    7$                ; Skip overlap checking if so
87                               ;
88 002170 012702 000000G      MOV     #WCBAS,R2        ; Point to 1st window control block
89 002174 005762 000000G      5$:    TST     WC$SIZ(R2)    ; Is this window in use?
90 002200 001432              BEQ     6$                ; Br if not
91 002202 020205              CMP     R2,R5            ; Is this the WCB for the new window?
92 002204 001430              BEQ     6$                ; Br if yes
93 002206 026265 000000G 000000G  CMP     WC$VHI(R2),WC$VLD(R5) ; Is old window below new window?
94 002214 101424              BLOS   6$                ; Br if yes
95 002216 026265 000000G 000000G  CMP     WC$VLD(R2),WC$VHI(R5) ; Is old window above new window?
96 002224 103020              BHIS   6$                ; Br if yes
97                               ;
98                               ; We have found a window overlap.
99                               ; Eliminate and possibly unmap the old window.
100                              ;
101 002226 052703 000000G      BIS     #WS.ELW,R3       ; Set flag saying we eliminated a window
102 002232 005062 000000G      CLR     WC$SIZ(R2)       ; Eliminate the window
103 002236 105762 000000G      TSTB   WC$MAP(R2)       ; Is window mapped now?
104 002242 001407              BEQ     5B$              ; Br if not
105 002244 010546              MOV     R5,-(SP)        ; Save address of WCB for new window
106 002246 010205              MOV     R2,R5           ; Get address of WCB of window being unmapped
107 002250 004737 004500'     CALL   WCBUMP           ; Unmap the window
108 002254 012605              MOV     (SP)+,R5        ; Get back address of WCB of new window
109 002256 052703 000000G      BIS     #WS.UNM,R3       ; Say we had to unmap a window
110 002262 005062 000000G      5B$:   CLR     WC$RCB(R2)   ; Dissociate window from region
111 002266 062702 000000G      6$:    ADD     #WC$$SZ,R2    ; Point to next WCB
112 002272 020227 000000G      CMP     R2,#WCBEND      ; Checked all windows?
113 002276 103736              BLO    5$                ; Br if more to check
114 002300 010346      7$:    MOV     R3,-(SP)        ; Return status flags

```

.CRAW -- Create a virtual address window

```
115 002302 106664 000000G          MTPD   W.NSTS(R4)      ;Move back to user's area
116                               ;
117                               ; If mapping is requested, go do that now
118                               ;
119 002306 032703 000000G          BIT    #WS.MAP,R3     ;Is mapping of window wanted?
120 002312 001015                    BNE    MAP2           ;Br if yes
121 002314 000137 000000G          JMP    EMTXIT        ;Finished
```

.MAP -- Map virtual window to physical region

```

1          .SBTTL .MAP -- Map virtual window to physical region
2          ;-----
3          ; The .MAP EMT is used to map a virtual window to a physical region.
4          ;
5 002320 004737 004632' MAP: CALL GETUDB ;Get address of window definition block (R4)
6          ;
7          ; Get window id, check that it is valid and convert it to the address
8          ; of the window control block.
9          ;
10 002324 004737 004646' CALL GETWCB ;Get address of window control block to R5
11 002330 005765 000000G TST WC$SIZ(R5) ;Has this window been allocated?
12 002334 001004 BNE MAP2 ;Br if yes
13 002336 012700 000003 MOV #EC3,R0 ;Return error code 3 if window not allocated
14 002342 000137 004764' JMP PLSERR
15          ;
16          ; Get address of associated Region Control Block
17          ;
18 002346 106564 000000G MAP2: MFPD W.NRID(R4) ;Get address of region control block
19 002352 012602 MOV (SP)+,R2
20 002354 010200 MOV R2,R0 ;Get RCB address to R0 for CHKRCB
21 002356 004737 004714' CALL CHKRCB ;Make sure RCB address is ok
22          ;
23          ; At this point, the following registers are set up:
24          ; R2 = Address of Region Control Block
25          ; R4 = Address of Window Definition Block in user's area.
26          ; R5 = Address of Window Control Block.
27          ;
28          ; If this window is already mapped set flag saying it is being unmapped.
29          ;
30 002362 105765 000000G 2$: TSTB WC$MAP(R5) ;Is window currently mapped?
31 002366 001406 BEQ 3$ ;Br if not
32 002370 106564 000000G MFPD W.NSTS(R4) ;Get current status flags
33 002374 052716 000000G BIS #WS.UNM,(SP) ;Set flag saying a window was unmapped
34 002400 106664 000000G MTPD W.NSTS(R4) ;Return new flags
35          ;
36          ; Determine length of window to map
37          ;
38 002404 106564 000000G 3$: MFPD W.NLEN(R4) ;Get specified mapping length
39 002410 012603 MOV (SP)+,R3
40 002412 001403 BEQ 4$ ;If zero, map as much as possible
41 002414 020365 000000G CMP R3,WC$SIZ(R5) ;Constrain length to size of window
42 002420 101402 BLOS 5$ ;Br if ok
43 002422 016503 000000G 4$: MOV WC$SIZ(R5),R3 ;Try to map full size of window
44 002426 106564 000000G 5$: MFPD W.NOFF(R4) ;Get window mapping offset
45 002432 012600 MOV (SP)+,R0
46 002434 010065 000000G MOV R0,WC$OFF(R5) ;And copy into window control block
47 002440 020062 000000G CMP R0,RC$LEN(R2) ;Is offset greater than region length?
48 002444 103406 BLO 6$ ;Br if not
49 002446 004737 004500' CALL WCBUMP ;Unmap the window
50 002452 012700 000004 MOV #EC4,R0 ;If yes then return error code 4
51 002456 000137 004764' JMP PLSERR
52 002462 005400 6$: NEG R0 ;Calculate how much of region is above
53 002464 066200 000000G ADD RC$LEN(R2),R0 ; offset
54 002470 020300 CMP R3,R0 ;Constrain mapped length to region size
55 002472 101401 BLOS 7$ ;Br if ok
56 002474 010003 MOV R0,R3 ;Get length of region
57 002476 010365 000000G 7$: MOV R3,WC$LEN(R5) ;Save length of window being mapped

```

.MAP -- Map virtual window to physical region

```

58 002502 106564 000000G      MFPD  W.NLEN(R4)      ;Get user specified length value
59 002506 005726              TST   (SP)+          ;Was specified value 0?
60 002510 001003              BNE   8$             ;Br if not -- don't change if non-zero
61 002512 010346              MOV   R3, -(SP)      ;Return actual length to user
62 002514 106664 000000G      MTPD  W.NLEN(R4)
63 002520 062703 000177      8$:  ADD   #177, R3    ;Determine # PAR's affected by window
64 002524 072327 177771      ASH   #-7, R3
65 002530 120365 000000G      CMPB  R3, WC$NPR(R5) ;Are we reducing the # of PAR's mapped?
66 002534 103002              BHIS  9$             ;Br if not
67 002536 004737 004500'      CALL  WCBUMP        ;Unmap all PAR's before setting new mapping
68 002542 110365 000000G      9$:  MOVB  R3, WC$NPR(R5) ;Number of PAR's to change with window
69 002546 112765 000000G 000000G  MOVB  #WC.MAP, WC$MAP(R5) ;Say window mapped normally (not fastmap)
70                               ;
71                               ; Now do the actual mapping
72                               ;
73 002554 004737 004252'      CALL  WCBMAP        ;Do actual window mapping
74                               ;
75                               ; Finished with EMT
76                               ;
77 002560 000137 000000G      JMP   EMTXIT

```

.ELAW -- Eliminate a virtual address window

```

1          .SBTTL .ELAW -- Eliminate a virtual address window
2          ;-----
3          ; The .ELAW EMT eliminates a virtual address window and frees the
4          ; associated window control block.
5          ;
6 002564 004737 004632' ELAW: CALL GETUDB ;Get address of window def block to R4
7 002570 004737 004646'      CALL GETWCB ;Get address of window control block to R5
8          ;
9          ; Unmap the window
10         ;
11 002574 004737 004500'      CALL WCBUMP ;Unmap the window
12         ;
13         ; Eliminate the window
14         ;
15 002600 005065 000000G      CLR WC$SIZ(R5) ;Eliminate window and say WCB is free
16 002604 005065 000000G      CLR WC$RCB(R5) ;And not associated with any region
17         ;
18         ; Return WS.ELW status flag
19         ;
20 002610 106564 000000G      MFPD W.NSTS(R4) ;Get current status flags
21 002614 052716 000000G      BIS #WS.ELW,(SP) ;Set flag
22 002620 106664 000000G      MTPD W.NSTS(R4) ;Return flags
23         ;
24         ; Finished
25         ;
26 002624 000137 000000G      JMP EMTXIT

```

.UNMAP -- Unmap a virtual window

```

1          .SBTTL .UNMAP -- Unmap a virtual window
2          ;-----
3          ; The .UNMAP EMT unmaps but does not eliminate a virtual window.
4          ;
5 002630 004737 004632' UNMAP: CALL GETUDB ;Get address of window def block to R4
6 002634 004737 004646'          CALL GETWCB ;Get address of window control block to R5
7          ;
8          ; Make sure window is currently mapped
9          ;
10 002640 105765 000000G          TSTB WC$MAP(R5) ;Is window currently mapped?
11 002644 001004          BNE 1$ ;Br if yes
12 002646 012700 000005          MOV #EC5,R0 ;Error 5 if not mapped
13 002652 000137 004764'          JMP PLSERR
14          ;
15          ; Unmap the window
16          ;
17 002656 004737 004500' 1$: CALL WCBUMP ;Unmap the window
18          ;
19          ; Set WS.UNM status flag
20          ;
21 002662 106564 000000G          MFPD W.NSTS(R4) ;Get current status flags
22 002666 052716 000000G          BIS #WS.UNM,(SP) ;Set flag
23 002672 106664 000000G          MTPD W.NSTS(R4) ;Return flags
24          ;
25          ; Finished
26          ;
27 002676 000137 000000G          JMP EMTXIT

```

.GMCX -- Get window mapping status

```

1          .SBTTL .GMCX -- Get window mapping status
2          ;-----
3          ; The .GMCX EMT is used to get the mapping status of a window.
4          ; If a window id of 0 is specified, the status of the root portion of the
5          ; program is returned.
6          ;
7 002702 004737 004632' GMCX: CALL GETUDB ;Get address of window def block to R4
8          ;
9          ; Get window id and see if it is zero
10         ;
11 002706 106564 000000G MFPD W.NID(R4) ;Get word with window id
12 002712 012605 MOV (SP)+,R5
13 002714 042705 177400 BIC #177400,R5 ;Clear all but window id
14 002720 001500 BEQ GMCXO ;Br if window id = 0
15         ;
16         ; We are getting the status of a PLAS window
17         ;
18 002722 004737 004646' CALL GETWCB ;Get address of window control block to R5
19 002726 005765 000000G TST WC$SIZ(R5) ;Is this window currenty allocated?
20 002732 001004 BNE 1$ ;Br if yes
21 002734 012700 000003 MOV #EC3,R0 ;Return error code 3 if not allocated
22 002740 000137 004764' JMP PLSERR
23         ; Set APR number
24 002744 106564 000000G 1$: MFPD W.NID(R4) ;Get word with APR value and window id
25 002750 116566 000000G 000001 MOVB WC$PAR(R5),1(SP);Set APR index in high-order byte
26 002756 106266 000001 ASRB 1(SP) ;Convert index into actual APR number
27 002762 106664 000000G MTPD W.NID(R4) ;Return to window def block
28         ; Set window virtual address
29 002766 016546 000000G MOV WC$VLO(R5),-(SP);Window virtual address base
30 002772 106664 000000G MTPD W.NBAS(R4)
31         ; Set window size
32 002776 016546 000000G MOV WC$SIZ(R5),-(SP);Window size (64-byte units)
33 003002 106664 000000G MTPD W.NSIZ(R4)
34         ; Set region identifier
35 003006 016500 000000G MOV WC$RCB(R5),R0 ;Get address of region control block
36 003012 010046 MOV R0,-(SP)
37 003014 106664 000000G MTPD W.NRID(R4)
38 003020 105765 000000G TSTB WC$MAP(R5) ;Is window currently mapped?
39 003024 001420 BEQ 2$ ;Br if window is not mapped
40         ;
41         ; Window is mapped to a region.
42         ; Return offset.
43         ;
44 003026 016546 000000G MOV WC$OFF(R5),-(SP)
45 003032 106664 000000G MTPD W.NOFF(R4)
46         ; Length of mapped portion of window
47 003036 016546 000000G MOV WC$LEN(R5),-(SP)
48 003042 106664 000000G MTPD W.NLEN(R4)
49         ; Status
50 003046 106564 000000G MFPD W.NSTS(R4) ;Get current status flags
51 003052 052716 000000G BIS #WS.MAP,(SP) ;Say window is mapped
52 003056 106664 000000G MTPD W.NSTS(R4)
53 003062 000137 000000G JMP EMTXIT ;Finished
54         ;
55         ; Window is not mapped
56         ;
57 003066 005046 2$: CLR -(SP)

```

.GMCX -- Get window mapping status

```

58 003070 106664 000000G          MTPD  W.NOFF(R4)      ; Offset
59 003074 005046                   CLR   -(SP)
60 003076 106664 000000G          MTPD  W.NLEN(R4)      ; Mapped length
61 003102 106564 000000G          MFPD  W.NSTS(R4)      ; Get status flags
62 003106 042716 000000G          BIC   #WS.MAP,(SP)   ; Clear mapped flag
63 003112 106664 000000G          MTPD  W.NSTS(R4)
64 003116 000137 000000G          JMP   EMTXIT          ; Finished
65
66                               ; Window id = 0.
67                               ; Return status of root portion of job.
68
69 003122 005046                   GMCX0: CLR   -(SP)
70 003124 106664 000000G          MTPD  W.NID(R4)      ; Say APR = 0
71                               ; Virtual base address
72 003130 005046                   CLR   -(SP)          ; Say base address = 0
73 003132 106664 000000G          MTPD  W.NBAS(R4)
74                               ; Region size and length
75 003136 016100 000000G          MOV   LNBLKS(R1),RO  ; Get # 512-byte blocks assigned to job root
76 003142 163700 000000G          SUB   CXTPAG,RO      ; Subtract # blocks used by context block
77 003146 072027 0000003          ASH   #3,RO          ; Convert to # 64-byte blocks
78 003152 010046                   MOV   RO,-(SP)
79 003154 106664 000000G          MTPD  W.NSIZ(R4)     ; Region size
80 003160 010046                   MOV   RO,-(SP)
81 003162 106664 000000G          MTPD  W.NLEN(R4)     ; Region mapped length
82                               ; Region id
83 003166 005046                   CLR   -(SP)
84 003170 106664 000000G          MTPD  W.NRID(R4)
85                               ; Offset
86 003174 005046                   CLR   -(SP)
87 003176 106664 000000G          MTPD  W.NOFF(R4)
88                               ; Status
89 003202 106564 000000G          MFPD  W.NSTS(R4)
90 003206 052716 000000G          BIS   #WS.MAP,(SP)
91 003212 106664 000000G          MTPD  W.NSTS(R4)
92 003216 000137 000000G          JMP   EMTXIT          ; Finished

```

```

1          .SBTTL  FLRCB  -- Find local RCB for private named region
2          ;-----
3          ; Try to locate a local RCB for a private region whose name matches
4          ; that specified in the user's RDB.
5          ;
6          ; Inputs:
7          ;   R4 = Pointer to user's RDB.
8          ;
9          ; Outputs:
10         ;   C-flag cleared ==> Found RCB.
11         ;   C-flag set ==> Could not find RCB.
12         ;   R5 = Pointer to RCB that was found.
13         ;
14 003222  012705  000000G  FLRCB:  MOV      #RCBBAS,R5      ;Point to first local RCB
15         ;
16         ; See if this RCB is for a named, private region
17         ;
18 003226  016500  000000G  1$:   MOV      RC$FLG(R5),R0    ;Get RCB flags
19 003232  032700  000000G        BIT      #RC$USE,R0      ;Is this RCB active?
20 003236  001420                BEQ      2$                ;Br if not
21 003240  032700  000000G        BIT      #RC$GBL,R0      ;Is this for a named region?
22 003244  001415                BEQ      2$                ;Br if not
23 003246  032700  000000G        BIT      #RC$PVT,R0     ;Is this for a private region?
24 003252  001412                BEQ      2$                ;Br if not
25         ;
26         ; Compare specified name with name in RCB
27         ;
28 003254  106564  000000G        MFPD     R.NAME(R4)      ;Get 1st 3 chars of name
29 003260  022665  000000G        CMP      (SP)+,RC$NAM(R5) ;Does first part of name match?
30 003264  001005                BNE     2$                ;Br if not
31 003266  106564  000002G        MFPD     R.NAME+2(R4)    ;Get 2nd 3 chars of name
32 003272  022665  000002G        CMP      (SP)+,RC$NAM+2(R5) ;Do names match?
33 003276  001407                BEQ     3$                ;Br if yes -- Found the RCB
34         ;
35         ; Try next RCB
36         ;
37 003300  062705  000000G  2$:   ADD      #RC$$SZ,R5      ;Point to next RCB
38 003304  020527  000000G        CMP      R5,#RCBEND    ;Checked all RCB's?
39 003310  103746                BLO     1$                ;Br if not
40         ;
41         ; There is no matching RCB
42         ;
43 003312  000261                SEC                      ;Signal failure on return
44 003314  000401                BR      9$
45         ;
46         ; We found a matching RCB
47         ;
48 003316  000241  3$:   CLC                      ;Signal success on return
49         ;
50         ; Finished
51         ;
52 003320  000207  9$:   RETURN

```

FLCRCB -- Find local RCB for shared region

```

1          .SBTTL  FLCRCB -- Find local RCB for shared region
2          ;-----
3          ; Try to locate a local RCB for a shared region whose name matches
4          ; that specified in the user's RDB.
5          ;
6          ; Inputs:
7          ;   R4 = Pointer to user's RDB.
8          ;
9          ; Outputs:
10         ;   C-flag cleared ==> Found RCB.
11         ;   C-flag set ==> Could not find RCB.
12         ;   R3 = Pointer to RCB that was found.
13         ;
14 003322 012703 000000G FLCRCB: MOV      #RCBBAS,R3      ;Point to first local RCB
15         ;
16         ; See if this RCB is for a shared region
17         ;
18 003326 016300 000000G 1$:      MOV      RC$FLG(R3),R0    ;Get RCB flags
19 003332 032700 000000G          BIT      #RC$USE,R0      ;Is this RCB active?
20 003336 001415          BEQ      2$                ;Br if not
21 003340 032700 000000G          BIT      #RC$LCC,R0      ;Is this for a shared region?
22 003344 001412          BEQ      2$                ;Br if not
23         ;
24         ; Compare specified name with name in RCB
25         ;
26 003346 106564 000000G          MFPD     R.NAME(R4)      ;Get 1st 3 chars of name
27 003352 022663 000000G          CMP      (SP)+,RC$NAM(R3) ;Does first part of name match?
28 003356 001005          BNE      2$                ;Br if not
29 003360 106564 000002G          MFPD     R.NAME+2(R4)    ;Get 2nd 3 chars of name
30 003364 022663 000002G          CMP      (SP)+,RC$NAM+2(R3) ;Do names match?
31 003370 001407          BEQ      3$                ;Br if yes -- Found the RCB
32         ;
33         ; Try next RCB
34         ;
35 003372 062703 000000G 2$:      ADD      #RC$$SZ,R3      ;Point to next RCB
36 003376 020327 000000G          CMP      R3,#RCBEND    ;Checked all RCB's?
37 003402 103751          BLO      1$                ;Br if not
38         ;
39         ; There is no matching RCB
40         ;
41 003404 000261          SEC                          ;Signal failure on return
42 003406 000401          BR       9$
43         ;
44         ; We found a matching RCB
45         ;
46 003410 000241 3$:      CLC                          ;Signal success on return
47         ;
48         ; Finished
49         ;
50 003412 000207 9$:      RETURN

```

```

1          .SBTTL  FGRCB  -- Find global RCB for shared named region
2          ;-----
3          ; Try to locate a global RCB for a region whose name matches
4          ; that specified in the user's RDB.
5          ;
6          ; Inputs:
7          ;   R4 = Pointer to user's RDB.
8          ;
9          ; Outputs:
10         ;   C-flag cleared ==> Found RCB.
11         ;   C-flag set ==> Could not find RCB.
12         ;   R5 = Pointer to RCB that was found.
13         ;
14 003414 013705 000000G FGRCB:  MOV      SHRRCB,R5      ;Point to first shared RCB
15         ;
16         ; See if this RCB is for an active region
17         ;
18 003420 032765 000000G 000000G 1$:  BIT      #RC$USE,RC$FLG(R5) ;Is this RCB active?
19 003426 001412          BEQ      2$              ;Br if not
20         ;
21         ; Compare specified name with name in RCB
22         ;
23 003430 106564 000000G          MFPD     R.NAME(R4)      ;Get 1st 3 chars of name
24 003434 022665 000000G          CMP      (SP)+,RC$NAM(R5); Does first part of name match?
25 003440 001005          BNE      2$              ;Br if not
26 003442 106564 000002G          MFPD     R.NAME+2(R4)    ;Get 2nd 3 chars of name
27 003446 022665 000002G          CMP      (SP)+,RC$NAM+2(R5); Do names match?
28 003452 001407          BEQ      3$              ;Br if yes -- Found the RCB
29         ;
30         ; Try next RCB
31         ;
32 003454 062705 000000G 2$:  ADD      #RC$$SZ,R5      ;Point to next RCB
33 003460 020537 000000G          CMP      R5,SHRRCN      ;Checked all RCB's?
34 003464 103755          BLD      1$              ;Br if not
35         ;
36         ; There is no matching RCB
37         ;
38 003466 000261          SEC                      ;Signal failure on return
39 003470 000401          BR      9$
40         ;
41         ; We found a matching RCB
42         ;
43 003472 000241 3$:  CLC                      ;Signal success on return
44         ;
45         ; Finished
46         ;
47 003474 000207 9$:  RETURN

```

GLRCB -- Get a free local RCB

```

1
2
3
4
5
6
7
8
9
10
11
12 003476 012705 000000G
13 003502 032765 000000G 000000G
14 003510 001411
15 003512 062705 000000G
16 003516 020527 000000G
17 003522 103767
18
19
20
21 003524 012700 000006
22 003530 000137 004764'
23
24
25
26
27 003534 004737 003606'
28
29
30
31 003540 000207

                .SBTTL  GLRCB  -- Get a free local RCB
                -----
                ; Get a free local RCB and initialize it.
                ;
                ; Inputs:
                ;   R4 = Pointer to user's RCB.
                ;   R2 = Status flags from user's RCB
                ;
                ; Outputs:
                ;   R5 = Pointer to free RCB.
                ;
GLRCB:  MOV      #RCBBAS,R5      ;Point to 1st local RCB
1$:    BIT      #RC$USE,RC$FLG(R5) ;Is this RCB free?
        BEQ     2$              ;Br if yes
        ADD     #RC$$SZ,R5      ;Point to next RCB
        CMP     R5,#RCBEND      ;Checked all?
        BLO    1$              ;Loop if not
                ;
                ; Error -- No free RCB's
                ;
                MOV     #EC6,R0      ;Return error code 6
                JMP     PLSERR
                ;
                ; We found a free RCB.
                ; Initialize it
                ;
2$:    CALL     INIRCB          ;Initialize the RCB
                ;
                ; Finished
                ;
                RETURN
    
```

GGRCB -- Get a free global RCB

```

1          .SBTTL  GGRCB  -- Get a free global RCB
2          ;-----
3          ; Get a free global RCB and initialize it.
4          ;
5          ; Inputs:
6          ;   R4 = Pointer to user's RCB.
7          ;   R2 = Status flags from user's RCB
8          ;
9          ; Outputs:
10         ;   R5 = Pointer to free RCB.
11         ;
12 003542 013705 000000G GGRCB:  MOV    SHRRCB,R5      ;Point to 1st shared RCB
13 003546 032765 000000G 000000G 1$:  BIT    #RC$USE,RC$FLG(R5) ;Is this RCB free?
14 003554 001411          BEQ    2$              ;Br if yes
15 003556 062705 000000G          ADD    #RC$$SZ,R5      ;Point to next RCB
16 003562 020537 000000G          CMP    R5,SHRRCN      ;Checked all?
17 003566 103767          BLD    1$              ;Loop if not
18         ;
19         ; Error -- No free RCB's
20         ;
21 003570 012700 000006          MOV    #EC6,RO        ;Return error code 6
22 003574 000137 004764'          JMP    PLSERR
23         ;
24         ; We found a free RCB.
25         ; Initialize it
26         ;
27 003600 004737 003606' 2$:  CALL  INIRCB      ;Initialize the RCB
28         ;
29         ; Finished
30         ;
31 003604 000207          RETURN

```

INIRCB -- Initialize a new RCB

```

1
2
3
4
5
6
7
8
9
10
11 003606 010346
12
13
14
15 003610 010503
16 003612 012700 000000C
17 003616 005023
18 003620 077002
19
20
21
22 003622 113765 000000G 000000G
23
24
25
26 003630 106564 000000G
27 003634 012600
28 003636 010065 000000G
29 003642 062700 000007
30 003646 000241
31 003650 006000
32 003652 072027 177776
33 003656 010065 000000G
34
35
36
37 003662 032702 000000C
38 003666 001413
39 003670 052765 000000G 000000G
40 003676 106564 000000G
41 003702 012665 000000G
42 003706 106564 000002G
43 003712 012665 000002G
44
45
46
47 003716 012603
48 003720 000207

                .SBTTL  INIRCB -- Initialize a new RCB
                -----
; INIRCB zeroes a new RCB and stores the name of the region into it
; if it is a named region.
;
; Inputs:
; R5 = Pointer to new RCB to be initialized.
; R4 = Pointer to user's RDB.
; R2 = Status flags from RDB.
;
INIRCB: MOV      R3, -(SP)
;
; Zero the RCB
;
                MOV      R5, R3          ;Point to the RCB
                MOV      #RC$$SZ/2, R0  ;Get # words to zero
1$: CLR      (R3)+          ;Zero the RCB
                SOB      R0, 1$
;
; Initialize the RCB
;
                MOVVB   CORUSR, RC$DOWN(R5) ;Save # of job creating region
;
; Set up information about the size of the region.
;
                MFPD    R, GSIZ(R4)      ;Get # 64-byte blocks needed for region
                MOV      (SP)+, R0
                MOV      R0, RC$LEN(R5)  ;Save region size in RCB
                ADD      #7, R0          ;Bound up to # 512-byte pages
                CLC      ;Clear carry for rotate
                ROR      R0              ;Convert # 64-byte blocks to # 512-byte blocks
                ASH      #-2, R0
                MOV      R0, RC$PAG(R5)  ;Save # pages needed for region
;
; Store the name into the RCB if it is a named region.
;
                BIT      #<RS. GBL!RS. CGR>, R2 ;Is this a named region?
                BEQ      9$              ;Br if not
                BIS      #RC$GBL, RC$FLG(R5);Remember this is a named region
                MFPD    R, NAME(R4)      ;Get 1st 3 chars of name
                MOV      (SP)+, RC$NAM(R5);Store into RCB
                MFPD    R, NAME+2(R4)    ;Get 2nd 3 chars of name
                MOV      (SP)+, RC$NAM+2(R5);Store into RCB
;
; Finished
;
9$: MOV      (SP)+, R3
                RETURN

```

CHKATT -- Check valid attachment to named region

```

1          .SBTTL  CHKATT -- Check valid attachment to named region
2          ;-----
3          ;  CHKATT is called when we are about to attach to an existing named
4          ;  region to make sure the size specified in the RDB is compatible with
5          ;  the size of the region.
6          ;
7          ;  Inputs:
8          ;  R4 = Pointer to user's RDB.
9          ;  R5 = Pointer to RCB for region.
10         ;
11 003722  CHKATT:
12         ;
13         ;  See if we are trying to attach to a region that is reserved for
14         ;  exclusive access by another job.
15         ;
16 003722  032765  000000C 000000G          BIT    #<RC$LCG!RC$PVT>,RC$FLG(R5) ;Private or local copy of global?
17 003730  001010          BNE    1$          ;Br if yes
18 003732  032765  000000G 000000G          BIT    #RC$EXC,RC$FLG(R5) ;Exclusive access to region reserved?
19 003740  001404          BEQ    1$          ;Br if not
20 003742  012700  000015          MOV    #EC15,R0          ;Return error code 15
21 003746  000137  004764'          JMP    PLSERR
22         ;
23         ;  Get size of region from RDB
24         ;
25 003752  106564  000000G          1$:    MFPD   R.GSIZ(R4)          ;Get requested size of region
26 003756  012600          MOV    (SP)+,R0
27 003760  001412          BEQ    9$          ;Size of zero is ok
28 003762  020065  000000G          CMP    R0,RC$LEN(R5)    ;Compare with existing length
29 003766  101407          BLOS  9$          ;Br if ok
30         ;
31         ;  Error -- Requested size is too big
32         ;
33 003770  016537  000000G 000000G          MOV    RC$LEN(R5),URO   ;Return actual region size to user in R0
34 003776  012700  000007          MOV    #EC7,R0          ;Return error code 7
35 004002  000137  004764'          JMP    PLSERR
36         ;
37         ;  Region size is ok
38         ;
39 004006  000207          9$:    RETURN

```

RELGRN -- Release attachment to a region

```

1          .SBTTL  RELGRN -- Release attachment to a region
2          ;-----
3          ; RELGRN is called to release an attachment by the current job to a region.
4          ;
5          ; Inputs:
6          ; R2 = Address of Region Control Block for region being released.
7          ;
8 004010 010246 RELGRN: MOV     R2,-(SP)
9 004012 010546      MOV     R5,-(SP)
10         ;
11         ; Unmap any windows that are mapped to this region
12         ;
13 004014 012705 000000G      MOV     #WCBBAS,R5      ;Point to first window control block
14 004020 020265 000000G 5$:  CMP     R2,WC$RCB(R5)  ;Is this window associated with this region?
15 004024 001011          BNE     6$           ;Br if not
16 004026 004737 004500'      CALL    WCBUMP        ;Unmap the window
17 004032 005065 000000G      CLR     WC$SIZ(R5)    ;Eliminate it
18 004036 005065 000000G      CLR     WC$RCB(R5)    ;And dissociate it from region
19 004042 052762 000000G 000000G  BIS     #RC$UNM,RC$FLG(R2) ;Remember a window was eliminated
20 004050 062705 000000G 6$:  ADD     #WC$$SZ,R5    ;Point to next WCB
21 004054 020527 000000G      CMP     R5,#WCBEND   ;Checked all windows?
22 004060 103757          BLO     5$           ;Br if not
23         ;
24         ; See if this is a private or shared region.
25         ;
26 004062 016200 000000G      MOV     RC$FLG(R2),R0 ;Get RCB status flags
27 004066 032700 000000G      BIT     #RC$LCG,R0   ;Private or shared region
28 004072 001022          BNE     3$           ;Br if shared region
29         ;
30         ; We are releasing a private region
31         ;
32 004074 032700 000000G      BIT     #RC$GBL,R0   ;Is this a named region?
33 004100 001457          BEQ     1$           ;Br if not -- Always eliminate
34 004102 032700 000000G      BIT     #RC$AGE,R0   ;Should we eliminate the region?
35 004106 001054          BNE     1$           ;Br if yes
36 004110 105062 000000G      CLRB   RC$CNT(R2)    ;Say attachment count = 0
37 004114 032700 000000G      BIT     #RC$AEP,R0   ;Is automatic elimination pending?
38 004120 001451          BEQ     9$           ;Br if not
39 004122 042762 000000G 000000G  BIC     #RC$AEP,RC$FLG(R2) ;Clear pending flag
40 004130 052762 000000G 000000G  BIS     #RC$AGE,RC$FLG(R2) ;Set automatic-elimination flag
41 004136 000442          BR      9$
42         ;
43         ; We are releasing a shared region.
44         ;
45 004140 042762 000000G 000000G 3$:  BIC     #RC$USE,RC$FLG(R2) ;Say local RCB is free
46 004146 016202 000000G      MOV     RC$BLK(R2),R2 ;Get address of global RCB
47 004152 032700 000000G      BIT     #RC$AGE,R0   ;Was AGE specified in local RCB?
48 004156 001403          BEQ     4$           ;Br if not
49 004160 052762 000000G 000000G  BIS     #RC$AGE,RC$FLG(R2) ;Transfer AGE option to global RCB
50 004166 016200 000000G 4$:  MOV     RC$FLG(R2),R0 ;Get status flags from global RCB
51 004172 105362 000000G      DECB   RC$CNT(R2)    ;Dec attachment count in global RCB
52 004176 001003          BNE     2$           ;Br if someone else still attached
53 004200 032700 000000G      BIT     #RC$AGE,R0   ;Is automatic elimination wanted?
54 004204 001015          BNE     1$           ;Br if yes
55 004206 042762 000000G 000000G 2$:  BIC     #RC$EXC,RC$FLG(R2) ;Clear exclusive-access flag
56 004214 032700 000000G      BIT     #RC$AEP,R0   ;Is automatic elimination pending?
57 004220 001411          BEQ     9$           ;Br if not

```

RELRCN -- Release attachment to a region

```

58 004222 052762 000000G 000000G      BIS      #RC$AGE,RC$FLG(R2)      ;Set automatic elimination flag
59 004230 042762 000000G 000000G      BIC      #RC$AEP,RC$FLG(R2) ;Clear automatic elimination pending
60 004236 000402                                     BR        9$
61                                     ;
62                                     ; Eliminate this region
63                                     ;
64 004240 004737 006456'      1$:      CALL      FRERCB      ;Release the region
65                                     ;
66                                     ; Finished
67                                     ;
68 004244 012605      9$:      MOV      (SP)+,R5
69 004246 012602                                     MOV      (SP)+,R2
70 004250 000207                                     RETURN
    
```

```

1                                     .SBTTL  WCBMAP -- Do actual window mapping
2                                     ;-----
3                                     ; WCBMAP is called to perform the actual mapping of a PLAS window to
4                                     ; a physical memory region. The information in the window control block
5                                     ; is used to set up the mapping. The user PAR and PDR registers are
6                                     ; loaded and the RPAR and RPDR cells are set up to reflect the mapping.
7                                     ;
8                                     ; Inputs:
9                                     ; R5 = Address of window control block.
10                                    ;
11 004252 010146 WCBMAP: MOV      R1,-(SP)
12 004254 010246      MOV      R2,-(SP)
13 004256 010346      MOV      R3,-(SP)
14 004260 010446      MOV      R4,-(SP)
15 004262 010546      MOV      R5,-(SP)
16                                    ;
17                                    ; Get information about region being mapped
18                                    ;
19 004264 016502 000000G      MOV      WC#RCB(R5),R2 ;Get address of region control block
20 004270 116503 000000G      MOVB     WC#PAR(R5),R3 ;Get PAR index number (2*base PAR #)
21 004274 016504 000000G      MOV      WC#LEN(R5),R4 ;Get # 64-byte blocks to be mapped
22 004300 016201 000000G      MOV      RC#BAS(R2),R1 ;Get base 64-byte block # of region
23 004304 066501 000000G      ADD      WC#OFF(R5),R1 ;Add window offset within region
24                                    ;
25                                    ; Determine if this window should be mapped thru I- or D-space
26                                    ;
27 004310 005005      CLR      R5 ;Assume I-space mapping
28 004312 032762 000000G 000000G      BIT      #RC#DSP,RC#FLG(R2) ;Should this region be in D-space?
29 004320 001404      BEQ      4$ ;Br if not
30 004322 105737 000000G      TSTB     IDSFLG ;Does the machine support separate space?
31 004326 001401      BEQ      4$ ;Br if not
32 004330 005205      INC      R5
33                                    ;
34                                    ; Begin loop to set up each PAR
35                                    ;
36 004332 012702 000200      4$:      MOV      #128.,R2 ;Carry 128 in R2
37 004336 010400      1$:      MOV      R4,R0 ;Get # 64-byte blocks left to map
38 004340 001451      BEQ      2$ ;Br if finished mapping
39 004342 020002      CMP      R0,R2 ;We can only map 128 blocks per PAR
40 004344 101401      BLOS     3$ ;Br if we can map all that is left
41 004346 010200      MOV      R2,R0 ;Map 128 blocks through this PAR
42 004350 160004      3$:      SUB      R0,R4 ;Get # 64-byte blocks left after this PAR
43 004352 005300      DEC      R0 ;Get # blocks -1
44 004354 000300      SWAB     R0 ;Put # blocks in high order byte
45 004356 052700 000000G      BIS      #6,R0 ;Allow read and write access to page
46 004362 042700 100261      BIC      #100261,R0 ;Make sure unused PDR bits are zero
47 004366 005705      TST      R5 ;Use I- or D-space registers?
48 004370 001015      BNE      5$ ;Br if D-space
49 004372 010063 000000G      MOV      R0,UPDR0(R3) ;Set actual PDR register
50 004376 010063 000000G      MOV      R0,RPDR(R3) ;Set shadow cell
51 004402 010063 000000G      MOV      R0,CUPDR0(R3) ;Set shadow cell in context block
52 004406 010163 000000G      MOV      R1,UPAR0(R3) ;Set actual PAR register
53 004412 010163 000000G      MOV      R1,RPAR(R3) ;Set shadow cell
54 004416 010163 000000G      MOV      R1,CUPAR0(R3) ;Set shadow cell in context block
55 004422 000414      BR       6$ ;Step up to next set
56 004424 010063 000000G      5$:      MOV      R0,UDDR0(R3) ;Set actual D-space PDR register
57 004430 010063 000000G      MOV      R0,RDDR(R3) ;Set shadow cell

```

WCBMAP -- Do actual window mapping

```

58 004434 010063 000000G      MOV      R0,CUDDRO(R3) ;Set shadow cell in context block
59 004440 010163 000000G      MOV      R1,UDARO(R3) ;Set actual D-space PAR register
60 004444 010163 000000G      MOV      R1,RDAR(R3) ;Set shadow cell
61 004450 010163 000000G      MOV      R1,CUDARO(R3) ;Set shadow cell in context block
62 004454 060201          6$:      ADD      R2,R1 ;Advance physical block number
63 004456 062703 0000002      ADD      #2,R3 ;Advance PAR index
64 004462 000725          BR       1$ ;Go load more PAR's
65                               ;
66                               ; Finished
67                               ;
68 004464 012605          2$:      MOV      (SP)+,R5
69 004466 012604          MOV      (SP)+,R4
70 004470 012603          MOV      (SP)+,R3
71 004472 012602          MOV      (SP)+,R2
72 004474 012601          MOV      (SP)+,R1
73 004476 000207          RETURN

```

WCBUMP -- Do actual unmapping of a PLAS window

```

1                                     .SBTTL  WCBUMP -- Do actual unmapping of a PLAS window
2                                     ;-----
3                                     ; WCBUMP is called to do the actual unmapping operation for a PLAS window.
4                                     ;
5                                     ; Inputs:
6                                     ; R5 = Address of window control block
7                                     ;
8 004500 010146 WCBUMP: MOV      R1, -(SP)
9 004502 010246      MOV      R2, -(SP)
10 004504 010346      MOV      R3, -(SP)
11
12 004506 105765 000000G      TSTB     WC$MAP(R5)      ; Is the window mapped now?
13 004512 001443      BEQ      9$              ; Br if not
14 004514 016503 000000G      MOV      WC$RCB(R5), R3      ; Get pointer to rcb
15 004520 116501 000000G      MOVVB    WC$PAR(R5), R1      ; Get first APR index number for the window
16 004524 032763 000000G 000000G  BIT      #RC$DSP, RC$FLG(R3) ; Was this mapped thru I- or D-space
17 004532 001003      BNE      4$              ; Br if thru D-space
18 004534 062701 000000G      ADD      #RPDR, R1          ; Index into I-space register set
19 004540 000402      BR       5$              ; Step over D-space offset
20 004542 062701 000000G 4$:      ADD      #RDDR, R1          ; Index into D-space register set
21 004546 116502 000000G 5$:      MOVVB    WC$TRP(R5), R2      ; Get trap region # associated with window
22 004552 006302      ASL      R2              ; Convert to index
23 004554 116500 000000G      MOVVB    WC$NPR(R5), R0      ; Get number of APR's affected by window
24 004560 001416      BEQ      2$              ; Br if no APR's to unmap
25 004562 005011      1$:      CLR      (R1)            ; Say no special mapping for this APR
26 004564 005702      TST     R2              ; Any trap region to clean up?
27 004566 100404      BMI     3$              ; Br if not
28 004570 005062 000000G      CLR     SR$PDR(R2)         ; Indicate that fast trap region not in use
29 004574 062702 000002      ADD     #2, R2            ; Advance trap region index
30 004600 062701 000002 3$:      ADD     #2, R1            ; Advance APR index
31 004604 077012      SOB     R0, 1$          ; Loop if more APR's to free
32                                     ;
33                                     ; We have reset the mapping information.
34                                     ; Call SETMAP to do the actual mapping for the job.
35                                     ;
36 004606 113701 000000G      MOVVB    CORUSR, R1         ; Get current job index number
37 004612 004737 000000G      CALL    SETMAP            ; Reload map registers for the job
38                                     ;
39                                     ; Clear WC$MAP to indicate window is unmapped
40                                     ;
41 004616 105065 000000G 2$:      CLRB    WC$MAP(R5)        ; Say window is now unmapped
42                                     ;
43                                     ; Finished
44                                     ;
45 004622 012603 9$:      MOV     (SP)+, R3
46 004624 012602      MOV     (SP)+, R2
47 004626 012601      MOV     (SP)+, R1
48 004630 000207      RETURN

```

GETUDB -- Get address of user definition block

```

1          .SBTTL  GETUDB -- Get address of user definition block
2          ;-----
3          ; GETUDB is called to get the address of a Region Definition Block or a
4          ; Window Definition Block from a user's EMT argument block.
5          ; The address is checked to make sure it is valid.
6          ;
7          ; Outputs:
8          ; R4 = Address of user's control block.
9          ;
10         004632 013700 000002G  GETUDB: MOV     EMTBLK+2,R0      ;Get address of control block
11         004636 010004          MOV     RO,R4          ;Return in R4
12         004640 004737 000000G          CALL    VALADW        ;Make sure address is valid
13         004644 000207          RETURN
14
15         ;-----
16         ; GETWCB is called to get the address of a Window Control Block
17         ; based on the value of the window id specified in a user's Window
18         ; Definition Block.
19         ; If the window id is not valid an emt error code of 3 is returned.
20         ;
21         ; Inputs:
22         ; R4 = Address of user's window definition block.
23         ;
24         ; Outputs:
25         ; R5 = Address of window control block.
26         ;
27         004646 106564 000000G  GETWCB: MFPD   W.NID(R4)      ;Get window id
28         004652 012600          MOV     (SP)+,R0
29         004654 042700 177400          BIC    #177400,R0        ;Clear high-order byte
30         004660 001411          BEQ    1$              ;Error if window = 0
31         004662 020027 000000G          CMP    RO,#NUMWCB      ;Make sure id is not too large
32         004666 101006          BHI    1$              ;Br if too big
33
34         ; Convert window id to address of window control block
35         ;
36         004670 012705 000000C  2$:   MOV     #WCBBAS-WC##SZ,R5 ;Point to WCB area
37         004674 062705 000000G  4$:   ADD     #WC##SZ,R5      ;Point to correct WCB
38         004700 077003          SOB    RO,4$           ;Based on ID number
39
40         ; Finished
41         ;
42         004702 000207          RETURN
43
44         ; Error -- Invalid window id
45         ;
46         004704 012700 000003  1$:   MOV     #EC3,R0        ;Return emt error code 3
47         004710 000137 004764'          JMP    PLSERR

```

CHKRCB -- Check address of Region Control Block

```

1          .SBTTL  CHKRCB -- Check address of Region Control Block
2          ;-----
3          ;  CHKRCB is called to verify that the address of a region control
4          ;  block is valid.  If the address is not valid, an emt error code
5          ;  number 2 is returned.
6          ;
7          ;  Inputs:
8          ;  RO = Region control block address to be checked
9          ;
10         CHKRCB: BIT    #1,RO          ;Make sure address is even
11         BNE     1$          ;Br if odd
12         CMP     RO,#RCBBAS        ;Make sure it is within proper region
13         BLD     2$
14         CMP     RO,#RCBEND
15         BLD     9$
16         2$:    CMP     RO,SHRRCB    ;See if this is a shared RCB
17         BLD     1$
18         CMP     RO,SHRRCN
19         BHI     1$
20         ;
21         ;  Address is ok
22         ;
23         9$:    RETURN
24         ;
25         ;  Error -- Invalid region control block address
26         ;
27         1$:    MOV     #EC2,RO      ;Return EMT error code 2
28         JMP     PLSERR

```

PLSERR -- Report error on PLAS EMT

```
1          .SBTTL  PLSERR -- Report error on PLAS EMT
2          ;-----
3          ; This routine is jumped to when any error occurs while processing
4          ; a PLAS EMT.  It returns the error code to the calling program
5          ; with the C-flag set on return from the EMT.
6          ;
7          ; Inputs:
8          ;   RO = EMT error code to be returned
9          ;
10 004764 000137 000000G  PLSERR: JMP      SETERR      ;Enter EMT error reporting routine
```

SEGOUT -- Outswap job regions

```

1          .SBTTL  SEGOUT -- Outswap job regions.
2          ;-----
3          ; SEGOUT is called to write all PLAS regions associated with a
4          ; particular job to the region swap file.
5          ;
6          ; Inputs:
7          ;   R1 = Job index number
8          ;
9          ; Outputs:
10         ;   C-flag set ==> No regions need to be outswapped for this job
11         ;   C-flag cleared ==> Beginning to swap regions.
12         ;
13 004770 010246 SEGOUT: MOV      R2, -(SP)
14         ;
15         ; Map PAR6 to job context block
16         ;
17 004772 016137 000000G 000000G      MOV      LCXPAR(R1), @KPAR6 ;Map kernel par6 to job context block
18         ;
19         ; Determine if any regions need to be swapped for this job
20         ;
21 005000 012702 000000G      MOV      #RCBBAS, R2      ;Point to first region descriptor
22 005004 032762 000000G 000000G 1$:  BIT      #RC$USE, RC$FLG(R2) ;Is this RCB active?
23 005012 001410      BEQ      3$          ;Br if not
24 005014 032762 000000G 000000G      BIT      #RC$LCG, RC$FLG(R2); Is this a shared region?
25 005022 001004      BNE      3$          ;Br if yes
26 005024 032762 000000G 000000G      BIT      #RC$INM, RC$FLG(R2); Is this region in memory now?
27 005032 001007      BNE      2$          ;Br if yes
28 005034 062702 000000G      3$:  ADD      #RC$$SZ, R2      ;Point to next region descriptor block
29 005040 020227 000000G      CMP      R2, #RCBEND      ;Checked all region descriptors?
30 005044 103757      BLO      1$          ;Br if not
31         ;
32         ; There are no regions that need to be outswapped
33         ;
34 005046 000261      SEC          ;Signal no regions to outswap
35 005050 000416      BR      9$
36         ;
37         ; We found a region that needs to be outswapped.
38         ; Set up information to describe the transfer.
39         ;
40 005052 010237 000002'      2$:  MOV      R2, SCBRCB      ;Save address of region control block
41 005056 016237 000000G 000004'      MOV      RC$BAS(R2), SCBBAS; Base 64-byte block # of region
42 005064 016237 000000G 000006'      MOV      RC$LEN(R2), SCBLEN; # 64-byte blocks to transfer
43 005072 016237 000000G 000010'      MOV      RC$BLK(R2), SCBBLK; Block # of start in swap file
44         ;
45         ; Initiate the transfer
46         ;
47 005100 004737 005112'      CALL     OSSGO      ;Start the I/O transfer
48 005104 000241      CLC          ;Signal that swapping is being done
49         ;
50         ; Finished
51         ;
52 005106 012602      9$:  MOV      (SP)+, R2
53 005110 000207      RETURN

```

SEGOOT -- Outswap job regions

```

1
2 ; -----
3 ; DSSGO is called to initiate an I/O operation to write a PLAS region
4 ; (or a portion of a region) to the region swap file.
5 ;
6 ; Inputs:
7 ; SCBBAS = Base 64-byte block # of start of region memory region
8 ; SCBLEN = Number of 64-byte blocks to transfer
9 ; SCBBLK = Block number in swap file where region is to be written
10 005112 010146 DSSGO:  MOV     R1, -(SP)
11 005114 010246      MOV     R2, -(SP)
12 ;
13 ; Get a system I/O queue element
14 ;
15 005116 012701 000000G      MOV     #SEGCHN, R1      ;Point to system swap file I/O channel
16 005122 004737 000000G      CALL    GETSYQ          ;Get I/O queue element (address in R1)
17 ;
18 ; Set up information in I/O queue element for the transfer
19 ;
20 005126 063761 000010' 000000G      ADD     SCBBLK, Q. BLKN(R1);Block number within file
21 005134 013761 000004' 000000G      MOV     SCBBAS, Q. PAR(R1);64-byte memory address
22 005142 012761 005234' 000000G      MOV     #OSSCMP, Q. COMP(R1);Address of completion routine
23 005150 013700 000006'      MOV     SCBLEN, R0      ;Get # 64-byte blocks to transfer
24 005154 020027 001770      CMP     R0, #1016.     ;Compare to max we can transfer at one time
25 005160 101402      BLUS   1$             ;Br if we can transfer all this time
26 005162 012700 001770      MOV     #1016., R0     ;Truncate amt to transfer
27 005166 160037 000006'      1$:   SUB     R0, SCBLEN  ;Get # blocks left to transfer
28 005172 060037 000004'      ADD     R0, SCBBAS    ;Advance address for next transfer
29 005176 010002      MOV     R0, R2
30 005200 072227 177775      ASH    #-3, R2        ;Convert # 64-byte blocks to # 512-byte blocks
31 005204 060237 000010'      ADD     R2, SCBBLK    ;Advance file block number
32 005210 072027 000005      ASH    #5, R0         ;Convert to # words to transfer
33 005214 005400      NEG     R0             ;Negative ==> Write operation
34 005216 010061 000000G      MOV     R0, Q. WCNT(R1);Set word count for transfer
35 ;
36 ; Queue the I/O request
37 ;
38 005222 004737 000000G      CALL    SYQIO         ;Initiate the I/O operation
39 ;
40 ; Return
41 ;
42 005226 012602      MOV     (SP)+, R2
43 005230 012601      MOV     (SP)+, R1
44 005232 000207      RETURN

```

SEGOUT -- Outswap job regions

```

1
2 ; -----
3 ; OSSCMP is the I/O completion routine for PLAS region out swapping.
4 005234 032737 000000C 000000G OSSCMP: BIT #CS$ERR!CS$EOF,SEGCHN ;Did an I/O error occur?
5 005242 001405 BEQ 1$ ;Br if not
6 005244 DIE #EM$SSE ;System crash if swapping error
7
8 ; Map PAR 6 to job context block
9
10 005256 113701 000000G 1$: MOVB OUTBSY,R1 ;Get index number of job being swapped
11 005262 016137 000000G 000000G MOV LCXPAR(R1),@#KPAR6 ;Map kernel par6 to job context block
12
13 ; See if there is more of this region to transfer
14
15 005270 005737 000006' TST SCBLEN ;Any more of region left to transfer?
16 005274 001403 BEQ 2$ ;Br if not
17
18 ; Transfer next portion of region
19
20 005276 004737 005112' CALL OSSGO ;Initiate next transfer
21 005302 000407 BR 3$
22
23 ; Finished swapping this region
24 ; Indicate that the region is no longer in memory
25
26 005304 013702 000002' 2$: MOV SCBRCB,R2 ;Get address of region control block
27 005310 004737 006610' CALL SEGFRE ;Free memory space used by the region
28
29 ; Start outswap of base portion of the job
30
31 005314 OCALL OUTSWP ;Start swap of base portion of the job
32
33 ; Finished
34
35 005322 000207 3$: RETURN

```

IASEGS -- Allocate memory for regions during inswap

```

1          .SBTTL  IASEGS -- Allocate memory for regions during inswap
2          ;-----
3          ; IASEGS is called to allocate memory regions for PLAS regions for
4          ; a job that is being inswapped.  The RPAR and RPDR cells are also
5          ; set up to reflect the current window mapping state.
6          ;
7          ; Inputs:
8          ; R1 = Job index number
9          ;
10         IASEGS: MOV      R2,-(SP)
11         MOV      R3,-(SP)
12         MOV      R5,-(SP)
13         ;
14         ; Map PAR6 to the job's context block
15         ;
16         005332  016137  000000G 000000G      MOV      LCXPAR(R1),@#KPAR6 ;Map kernel par6 to job context block
17         ;
18         ; Compute address of start of PLAS regions
19         ;
20         005340  016103  000000G      MOV      LBASE(R1),R3      ;Base 512-byte block # of job region
21         005344  066103  000000G      ADD      LNBLKS(R1),R3    ;Point past job base region
22         005350  072327  0000003      ASH      #3,R3           ;Convert to 64-byte page number
23         ;
24         ; Set base address for each region
25         ;
26         005354  012702  000000G      MOV      #RCBBAS,R2      ;Point to first region control block
27         005360  016200  000000G      1$:     MOV      RC$PAG(R2),R0    ;Get # 512-byte pages allocated to region
28         005364  001416                      BEQ      2$              ;Br if region is not allocated
29         005366  032762  000000G 000000G      BIT      #RC$LCG,RC$FLG(R2); Is this a shared region?
30         005374  001012                      BNE      2$              ;Br if yes
31         005376  072027  0000003      ASH      #3,R0           ;Convert to # 64-byte pages
32         005402  010362  000000G      MOV      R3,RC$BAS(R2)   ;Set base 64-byte page # of region
33         005406  020237  000000G      CMP      R2,KEYRCB       ;Is this region used for key definitions?
34         005412  001002                      BNE      3$              ;Br if not
35         005414  010337  000000G      MOV      R3,KEYPAR       ;Save pointer to base of key def region
36         005420  060003      3$:     ADD      R0,R3          ;Point past end of region's area
37         005422  062702  000000G      2$:     ADD      #RC$$SZ,R2    ;Point to next region control block
38         005426  020227  000000G      CMP      R2,#RCBEND      ;Have we checked all regions?
39         005432  103752                      BLO      1$              ;Loop if more to allocate
40         ;
41         ; Set up mapping information for each PLAS window
42         ;
43         005434  004737  005450'      CALL     MAPPLS          ;Set up mapping for PLAS regions
44         ;
45         ; Finished
46         ;
47         005440  012605                      MOV      (SP)+,R5
48         005442  012603                      MOV      (SP)+,R3
49         005444  012602                      MOV      (SP)+,R2
50         005446  000207                      RETURN

```

```
1                                     .SBTTL  MAPPLS -- Set up mapping for all PLAS windows
2                                     ;-----
3                                     ; MAPPLS is called to set up the memory mapping for all active PLAS
4                                     ; windows for the job.
5                                     ;
6 005450 010546 MAPPLS: MOV      R5, -(SP)
7                                     ;
8                                     ; Begin loop to process each window for the job
9                                     ;
10 005452 012705 000000G          MOV      #WCBBAS, R5          ;Point to first window control block
11 005456 105765 000000G          3$:   TSTB    WC$MAP(R5)          ;Is this window currently mapped?
12 005462 001402                  BEQ     4$              ;Br if not
13 005464 004737 005506'          CALL   MAPSEG          ;Set up mapping info for this window
14 005470 062705 000000G          4$:   ADD    #WC$$SZ, R5          ;Point to next window control block
15 005474 020527 000000G          CMP    R5, #WCBEND          ;Have we done all windows?
16 005500 103766                  BLO    3$              ;Br if more to do
17                                     ;
18                                     ; Finished
19                                     ;
20 005502 012605                  MOV    (SP)+, R5
21 005504 000207                  RETURN
```

MAPSEG -- Set up window mapping during an inswap

```

1          .SBTTL  MAPSEG -- Set up window mapping during an inswap
2          ;-----
3          ; MAPSEG is called to perform the actual mapping of a PLAS window to
4          ; a physical memory region. The information in the window control block
5          ; is used to set up the mapping.
6          ; The RPAR and RPDR cells are set up to reflect the mapping.
7          ;
8          ; Inputs:
9          ; R5 = Address of window control block.
10         ;
11         ; Outputs:
12         ; RPAR & RPDR cells set to to reflect correct mapping for windows.
13         ;
14 005506 010146 MAPSEG: MOV      R1, -(SP)
15 005510 010246      MOV      R2, -(SP)
16 005512 010346      MOV      R3, -(SP)
17 005514 010446      MOV      R4, -(SP)
18 005516 005046      CLR      -(SP)          ; Assume mapping will be thru I-space
19         ;
20         ; Get information about region being mapped
21         ;
22 005520 016502 000000G      MOV      WC$RCB(R5), R2      ; Get address of region control block
23 005524 116503 000000G      MOVVB   WC$PAR(R5), R3      ; Get PAR index number (2*base PAR #)
24 005530 016504 000000G      MOV      WC$LEN(R5), R4      ; Get # 64-byte blocks to be mapped
25 005534 016201 000000G      MOV      RC$BAS(R2), R1      ; Get base 64-byte block # of region
26 005540 066501 000000G      ADD      WC$OFF(R5), R1      ; Add window offset within region
27 005544 032762 000000G 000000G      BIT      #RC$DSP, RC$FLG(R2) ; Map this window thru D-space?
28 005552 001401      BEQ      6$          ; Br if not, use I-space
29 005554 005216      INC      (SP)          ; Set flag to use D-space registers
30 005556 116502 000000G      6$:  MOVVB   WC$TRP(R5), R2      ; See if this window has any fast map regions
31         ; Relies on sign extension if unused
32         ;
33         ; Begin loop to set up each PAR
34         ;
35 005562 010400      1$:  MOV      R4, R0          ; Get # 64-byte blocks left to map
36 005564 001457      BEQ      2$          ; Br if finished mapping
37 005566 020037 000200      CMP      R0, 128.      ; We can only map 128 blocks per PAR
38 005572 101402      BLOS    3$          ; Br if we can map all that is left
39 005574 013700 000200      MOV      128., R0      ; Map 128 blocks through this PAR
40 005600 160004      3$:  SUB      R0, R4          ; Get # 64-byte blocks left after this PAR
41 005602 005300      DEC      R0          ; Get # blocks -1
42 005604 000300      SWAB   R0          ; Put # blocks in high order byte
43 005606 052700 000006      BIS      #6, R0          ; Allow read and write access to page
44 005612 042700 100261      BIC      #100261, R0      ; Make sure unused PDR bits are zero
45 005616 005716      TST      (SP)          ; Are we mapping thru I- or D-space?
46 005620 001005      BNE      7$          ; Br if using D-space
47 005622 010063 000000G      MOV      R0, RPDR(R3)      ; Set shadow cell for PDR
48 005626 010163 000000G      MOV      R1, RPAR(R3)      ; Set shadow cell for PAR
49 005632 000404      BR      8$          ; Skip over D-space cells
50 005634 010063 000000G      7$:  MOV      R0, RDDR(R3)      ; Set shadow cell for D-space PDR
51 005640 010163 000000G      MOV      R1, RDAR(R3)      ; Set shadow cell for D-space PAR
52         ;
53         ; If this window is fast mapped, refresh the fast map cells
54         ;
55 005644 006302      8$:  ASL      R2          ; Convert trap # to word index
56 005646 100421      BMI      5$          ; Skip if no fast map region associated
57 005650 010062 000000G      MOV      R0, SR$PDR(R2)      ; Set PDR for fast map region

```

```
58 005654 010162 000000G      MOV    R1,SR$PAR(R2)    ;Set PAR base for fast map region
59 005660 006202              ASR    R2              ;Convert index back to trap #
60 005662 110362 000000G      MOVB   R3,SR$PX(R2)    ;Set PAR index
61 005666 005704              TST    R4              ;Any more left to map?
62 005670 001405              BEQ    4$              ;Br if not
63 005672 152762 000000G 000000G  BISB   #SR.MDR,SR$FLG(R2) ;There are more cells in this region
64 005700 005202              INC    R2              ;Step up to next trap cell for next set
65 005702 000403              BR     5$              ;
66 005704 142762 000000G 000000G 4$:  BICB   #SR.MDR,SR$FLG(R2) ;This is the last cell for this region
67                               ;
68                               ; Advance par and par index and repeat until done
69                               ;
70 005712 063701 000200      5$:   ADD    128,R1        ;Advance physical block number
71 005716 062703 000002      ADD    #2,R3          ;Advance PAR index
72 005722 000717              BR     1$              ;Go load more PAR's
73                               ;
74                               ; Finished
75                               ;
76 005724 005726      2$:   TST    (SP)+        ;Clean up I- vs. D-space flag on stack
77 005726 012604      MOV    (SP)+,R4
78 005730 012603      MOV    (SP)+,R3
79 005732 012602      MOV    (SP)+,R2
80 005734 012601      MOV    (SP)+,R1
81 005736 000207      RETURN
```

SEGIN -- Inswap job regions

```

1          .SBTTL  SEGIN  -- Inswap job regions
2          ;-----
3          ; SEGIN is called to swap into memory any PLAS regions associated with
4          ; a job.
5          ;
6          ; Inputs:
7          ;   R1 = Job index number
8          ;
9          ; Outputs:
10         ;   C-flag set ==> No regions need to be brought in
11         ;   C-flag cleared ==> Started inswap of a region
12         ;
13 005740  010246  SEGIN:  MOV      R2,-(SP)
14         ;
15         ; Map PAR6 to the job's context block
16         ;
17 005742  016137  000000G 000000G  MOV      LCXPAR(R1),@#KPAR6 ;Map kernel par6 to job context block
18         ;
19         ; Determine if there are any regions that need to be inswapped
20         ;
21 005750  012702  000000G          MOV      #RCBBAS,R2          ;Point to first region descriptor block
22 005754  032762  000000G 000000G 1$:  BIT      #RC$USE,RC$FLG(R2) ;Is this RCB active?
23 005762  001410          BEQ      2$              ;Br if not
24 005764  032762  000000G 000000G  BIT      #RC$LCG,RC$FLG(R2) ;Is this a shared region?
25 005772  001004          BNE      2$              ;Br if yes
26 005774  032762  000000G 000000G  BIT      #RC$INM,RC$FLG(R2);Is this region already in memory?
27 006002  001407          BEQ      3$              ;Br if not in memory
28 006004  062702  000000G          2$:  ADD      #RC$$SZ,R2          ;Point to next region descriptor block
29 006010  020227  000000G          CMP      R2,#RCBEND      ;Checked all region descriptors?
30 006014  103757          BLO      1$              ;Br if not
31         ;
32         ; No regions need to be inswapped
33         ;
34 006016  000261          SEC                      ;Signal no swapping needed on return
35 006020  000421          BR      9$
36         ;
37         ; We found a region that needs to be inswapped.
38         ; Set up information to describe the transfer.
39         ;
40 006022  010237  000002'          3$:  MOV      R2,SCBRCB          ;Remember which region is being swapped
41 006026  016237  000000G 000004'  MOV      RC$BAS(R2),SCBBAS ;Set base address for the transfer
42 006034  016237  000000G 000006'  MOV      RC$LEN(R2),SCBLEN ;Set # 64-byte blocks to transfer
43 006042  016237  000000G 000010'  MOV      RC$BLK(R2),SCBBLK ;Set base block number in swap file
44         ;
45         ; Set flag saying region is in memory
46         ;
47 006050  052762  000000G 000000G  BIS      #RC$INM,RC$FLG(R2);Set flag saying region is in memory
48         ;
49         ; Initiate the transfer
50         ;
51 006056  004737  006070'          CALL     ISSGO          ;Start the inswap
52 006062  000241          CLC                      ;Signal that we started a swap
53         ;
54         ; Finished
55         ;
56 006064  012602  9$:  MOV      (SP)+,R2
57 006066  000207          RETURN

```

SEGIN -- Inswap job regions

```

1          ; -----
2          ; ISSGO is called to initiate an inswap of a PLAS region.
3          ;
4          ; Inputs:
5          ; SCBBAS = Address in 64-byte units of memory where transfer is to begin
6          ; SCBLEN = Number of 64-byte blocks to transfer
7          ; SCBBLK = Block number in swap file
8          ;
9 006070 010146 ISSGO:  MOV     R1, -(SP)
10 006072 010246      MOV     R2, -(SP)
11          ;
12          ; Get a system I/O queue element
13          ;
14 006074 012701 000000G      MOV     #SEGCHN, R1      ;Point to region swapping channel
15 006100 004737 000000G      CALL    GETSYQ          ;Get a system I/O queue element (R1=address)
16          ;
17          ; Set up I/O queue element for the transfer
18          ;
19 006104 063761 000010' 000000G      ADD     SCBBLK, Q. BLKN(R1) ;File block number
20 006112 013761 000004' 000000G      MOV     SCBBAS, Q. PAR(R1) ;Base 64-byte memory address
21 006120 012761 006210' 000000G      MOV     #ISSCMP, Q. COMP(R1); Address of completion routine
22 006126 013700 000006'      MOV     SCBLEN, R0      ;Get # 64-byte units to transfer
23 006132 020027 001770      CMP     R0, #1016.     ;Compare to max we can transfer in 1 operation
24 006136 101402      BLOS   1$             ;Br if we can read all this time
25 006140 012700 001770      MOV     #1016., R0     ;Truncate the transfer
26 006144 160037 000006'      1$:   SUB     R0, SCBLEN   ;Decrease amt remaining to be transferred
27 006150 060037 000004'      ADD     R0, SCBBAS    ;Advance base address
28 006154 010002      MOV     R0, R2
29 006156 072227 177775      ASH    #-3, R2        ;Cvt # 64-byte blocks to # 512-byte blocks
30 006162 060237 000010'      ADD     R2, SCBBLK    ;Advance file block number
31 006166 072027 000005      ASH    #5, R0         ;Convert to # words to transfer
32 006172 010061 000000G      MOV     R0, Q. WCNT(R1) ;Set word count in queue element
33          ;
34          ; Queue the I/O request
35          ;
36 006176 004737 000000G      CALL    SYQIO         ;Initiate the I/O operation
37          ;
38          ; Finished
39          ;
40 006202 012602      MOV     (SP)+, R2
41 006204 012601      MOV     (SP)+, R1
42 006206 000207      RETURN

```

SEGIN -- Inswap job regions

```

1          ; -----
2          ;   ISSCMP is the I/O completion routine for PLAS region inswaps.
3          ;
4 006210   032737   000000C 000000G ISSCMP: BIT      #CS$ERR!CS$EOF,SEGCHN ;Did an I/O error occur?
5 006216   001405                   BEQ      1$              ;Br if not
6 006220                   DIE      #EM$SSE          ;System crash if region swap error
7          ;
8          ;   See if there is more of this region remaining to be transferred
9          ;
10 006232   113701   000000G      1$:   MOVB     INBSY,R1          ;Get index of job being swapped
11 006236   005737   000006'      TST     SCBLEN         ;Any portion of region left to transfer?
12 006242   001403                   BEQ     2$              ;Br if not
13          ;
14          ;   Start transfer of next portion of the region
15          ;
16 006244   004737   006070'      CALL    ISSGO          ;Start next transfer
17 006250   000406                   BR     3$
18          ;
19          ;   Finished region inswap
20          ;   See if there are more regions to inswap
21          ;
22 006252   004737   005740'      2$:   CALL    SEGIN          ;Try to start another region inswap
23 006256   103003                   BCC    3$              ;Br if started another transfer
24          ;
25          ;   Finished inswapping the last region
26          ;   Say job is completely in memory now
27          ;
28 006260                   OCALL   INSFIN          ;Completed inswap
29          ;
30          ;   Finished
31          ;
32 006266   000207      3$:   RETURN

```

PLSXIT -- Do PLAS cleanup on job exit

```

1                                     .SBTTL PLSXIT -- Do PLAS cleanup on job exit
2                                     ;-----
3                                     ; PLSXIT is called from the code that does job cleanup when a job
4                                     ; exits or is aborted.
5                                     ;
6 006270 010246 PLSXIT: MOV      R2,-(SP)
7 006272 010546          MOV      R5,-(SP)
8                                     ;
9                                     ; See if any regions need to be freed on job exit or abort
10                                    ;
11 006274 012702 000000G          MOV      #RCBBAS,R2          ;Point to first private region control block
12 006300 016200 000000G 4$:     MOV      RC$FLG(R2),R0      ;Get region status flags
13 006304 032700 000000G          BIT      #RC$USE,R0          ;Is this RCB active?
14 006310 001410          BEQ      6$                      ;Br if not
15 006312 032700 000000G          BIT      #RC$EXI,R0          ;Eliminate region on exit?
16 006316 001403          BEQ      5$                      ;Br if not
17 006320 052762 000000G 000000G  BIS      #RC$AGE,RC$FLG(R2) ;Force elimination of region
18 006326 004737 004010' 5$:     CALL     RELRGN          ;Release the region
19 006332 062702 000000G 6$:     ADD      #RC$$SZ,R2          ;Point to next RCB
20 006336 020227 000000G          CMP      R2,#RCBEND        ;Checked all regions?
21 006342 103756          BLO      4$                      ;Loop if not
22                                    ;
23                                    ; Deallocate any windows
24                                    ;
25 006344 012705 000000G          MOV      #WCBBAS,R5          ;Point to first window control block
26 006350 004737 004500' 3$:     CALL     WCBUMP          ;Unmap the window
27 006354 005065 000000G          CLR      WC$SIZ(R5)        ;Say this window is free
28 006360 062705 000000G          ADD      #WC$$SZ,R5          ;Point to next window control block
29 006364 020527 000000G          CMP      R5,#WCBEND        ;Done all windows?
30 006370 103767          BLO      3$                      ;Loop if not
31                                    ;
32                                    ; Finished
33                                    ;
34 006372 012605          MOV      (SP)+,R5
35 006374 012602          MOV      (SP)+,R2
36 006376 000207          RETURN

```

PLSOFF -- Do PLAS cleanup at job logoff

```

1          .SBTTL  PLSOFF -- Do PLAS cleanup at job logoff
2          ;-----
3          ; PLSOFF is called from the code that does a job logoff to do
4          ; cleanup processing for PLAS regions.
5          ;
6 006400 010246 PLSOFF: MOV      R2,-(SP)
7          ;
8          ; Detach from all regions
9          ;
10 006402 012702 000000G          MOV      #RCBBAS,R2          ;Point to 1st RCB
11 006406 016200 000000G 1$:    MOV      RC$FLG(R2),R0      ;Get status flags from RCB
12 006412 032700 000000G          BIT      #RC$USE,R0          ;Is this RCB active?
13 006416 001410          BEQ      2$              ;Br if not
14 006420 032700 000000C          BIT      #RC$PVT!RC$OFF,R0 ;Private or release on logoff?
15 006424 001403          BEQ      3$              ;Br if not
16 006426 052762 000000G 000000G  BIS      #RC$AGE,RC$FLG(R2) ;Force release of region
17 006434 004737 004010' 3$:    CALL     RELRGN          ;Break the attachment
18 006440 062702 000000G 2$:    ADD      #RC$$SZ,R2          ;Point to next RCB
19 006444 020227 000000G          CMP      R2,#RCBEND          ;Checked all RCB's?
20 006450 103756          BLO      1$              ;Loop if not
21          ;
22          ; Finished
23          ;
24 006452 012602          MOV      (SP)+,R2
25 006454 000207          RETURN

```

FRERCB -- Free memory and swap space for a region

```

1          .SBTTL  FRERCB -- Free memory and swap space for a region
2          ;-----
3          ; FRERCB is called to deallocate memory space and swap file space
4          ; for a plas region.
5          ;
6          ; Inputs:
7          ; R2 = Address of Region Control Block
8          ;
9 006456 010546 FRERCB: MOV      R5, -(SP)
10         ;
11         ; Return immediately if the region is not allocated
12         ;
13 006460 016200 000000G      MOV      RC$FLG(R2),R0      ;Get status flags from RCB
14 006464 032700 000000G      BIT      #RC$USE,R0      ;Is this RCB active?
15 006470 001445              BEQ      9$              ;Br if not
16         ;
17         ; See if this RCB is a local copy of a shared RCB.
18         ;
19 006472 032700 000000G      BIT      #RC$LCG,R0      ;Is this a local copy of shared RCB?
20 006476 001405              BEQ      5$              ;Br if not
21 006500 042762 000000G 000000G  BIC      #RC$USE,RC$FLG(R2) ;Say local copy is free
22 006506 016202 000000G      MOV      RC$BLK(R2),R2      ;Get pointer to shared RCB
23         ;
24         ; Free swap file space allocated to the region.
25         ;
26 006512 004737 007132' 5$:  CALL      SWPFRE      ;Free swap file space
27         ;
28         ; Free memory space allocated for the region
29         ;
30 006516 004737 006610'      CALL      SEGFRE      ;Free memory space
31         ;
32         ; If this is a private region, reduce memory space used by job
33         ;
34 006522 032762 000000G 000000G  BIT      #RC$PVT,RC$FLG(R2) ;Is this a private region?
35 006530 001410              BEQ      3$              ;Br if not
36 006532 113705 000000G      MOVSB   CORUSR,R5      ;Get job index number
37 006536 166265 000000G 000000G  SUB      RC$PAG(R2),LNSBLK(R5) ;Reduce # pages used by regions for job
38 006544 166265 000000G 000000G  SUB      RC$PAG(R2),LMEMIN(R5) ;Reduce total # pages used by job
39         ;
40         ; Mark the region control block as free
41         ;
42 006552 005062 000000G 3$:  CLR      RC$LEN(R2)      ;Say no memory allocated for this region
43 006556 005062 000000G      CLR      RC$PAG(R2)
44 006562 005062 000000G      CLR      RC$FLG(R2)
45         ;
46         ; If this region was used for key definitions, say it has been deallocated
47         ;
48 006566 020237 000000G      CMP      R2,KEYRCB      ;Was this the key definition region?
49 006572 001004              BNE      9$              ;Br if not
50 006574 005037 000000G      CLR      KEYRCB      ;Say region is gone
51 006600 005037 000000G      CLR      KEYPAR      ;No user-defined keys
52         ;
53         ; Finished
54         ;
55 006604 012605 9$:  MOV      (SP)+,R5
56 006606 000207      RETURN

```

SEGFRE -- Free memory space for a region

```

1          .SBTTL  SEGFRE -- Free memory space for a region
2          ;-----
3          ; SEGFRE is called to deallocate all memory space used by a PLAS region.
4          ;
5          ; Inputs:
6          ; R2 = Address of region control block for the region.
7          ;
8 006610 010146 SEGFRE: MOV      R1, -(SP)
9 006612 010246      MOV      R2, -(SP)
10 006614 010201      MOV      R2, R1          ; Carry RCB address in R1
11          ;
12          ; Free the memory space used by the region
13          ;
14 006616 016100 000000G      MOV      RC$LEN(R1), R0      ; Get # 64-byte blocks allocated for region
15 006622 001412      BEQ      1$          ; Br if no memory allocated for the region
16 006624 016102 000000G      MOV      RC$BAS(R1), R2      ; Get base 64-byte block # for region
17 006630 000241      CLC          ; Convert to 512 byte block number
18 006632 006002      ROR      R2
19 006634 072227 177776      ASH      #-2, R2
20 006640 016100 000000G      MOV      RC$PAG(R1), R0      ; Get # 512-byte pages used by this region
21 006644 004737 000000G      CALL     FREMEM          ; Free the memory space used by this region
22          ;
23          ; Mark region as not in memory
24          ;
25 006650 042761 000000G 000000G 1$: BIC      #RC$INM, RC$FLG(R1) ; Say region is not in memory
26 006656 005061 000000G      CLR      RC$BAS(R1)
27          ;
28          ; Finished
29          ;
30 006662 012602      MOV      (SP)+, R2
31 006664 012601      MOV      (SP)+, R1
32 006666 000207      RETURN

```

```

1          .SBTTL  SWPGET -- Find space in region swap file
2          ;-----
3          ; Try to find space in the region swap file for a PLAS region.
4          ;
5          ; Inputs:
6          ;   R0 = Number of blocks needed
7          ;
8          ; Outputs:
9          ;   C-flag cleared ==> Found the requested space.
10         ;   C-flag set    ==> Could not find enough free space.
11         ;   R2 = Starting block number of allocated area.
12         ;   R0 = Size of largest available region if the request could not
13         ;         be satisfied, or the size of the request if the request
14         ;         was satisfied.
15         ;
16 006670 010146 SWPGET: MOV     R1,-(SP)
17 006672 010346      MOV     R3,-(SP)
18 006674 010446      MOV     R4,-(SP)
19 006676 010546      MOV     R5,-(SP)
20 006700 010046      MOV     R0,-(SP)      ; Save requested # blocks on top of stack
21         ;
22         ; Initialize pointers to bit map
23         ;
24 006702 013704 000012'      MOV     SSMAP,R4      ; Base of bit map table
25 006706 005005      CLR     R5          ; Initialize block number
26 006710 005000      CLR     R0          ; Initialize largest region size
27         ;
28         ; Rapidly skip over allocated blocks at the front of the file
29         ; Do this by checking 16 blocks at a time
30         ;
31 006712 013702 000000G      MOV     VPLAS,R2      ; Get total # blocks in swap file
32 006716 005724 15$:      TST     (R4)+      ; Any free blocks in this word?
33 006720 001005      BNE     16$      ; Br if yes
34 006722 062705 000020      ADD     #16.,R5      ; Advance block number
35 006726 020502      CMP     R5,R2      ; Have we gone past end of file?
36 006730 103772      BLD     15$      ; Keep looking if not
37 006732 000470      BR     12$      ; There are no free blocks
38 006734 005744 16$:      TST     -(R4)      ; Point back to word with 1st free block
39         ;
40         ; Search for the start of a free region
41         ;
42 006736 012701 000001      MOV     #1,R1      ; Get bit mask
43 006742 013702 000000G  8$:      MOV     VPLAS,R2      ; Get total number of blocks in swap file
44 006746 020502 1$:      CMP     R5,R2      ; Have we gone past end of the file?
45 006750 103061      BHIS   12$      ; Br if yes
46 006752 030114      BIT     R1,(R4)      ; Is this block free?
47 006754 001006      BNE     2$      ; Br if found a free block
48 006756 005205      INC     R5          ; Advance block number
49 006760 006301      ASL     R1          ; Shift bit mask for next block
50 006762 103371      BCC     1$      ; Br if didn't shift out of end of word
51 006764 006101      ROL     R1          ; Rotate bit back into right end of word
52 006766 005724      TST     (R4)+      ; Point to next word of bit map
53 006770 000766      BR     1$          ; Continue search for free block
54         ;
55         ; Found the start of a free region in swap file.
56         ; See if it is big enough to satisfy request.
57         ;

```

```

58 006772 010502      2$:   MOV     R5,R2      ;Save starting block number
59 006774 010503      MOV     R5,R3      ;Calculate block # beyond end of what we need
60 006776 061603      ADD     (SP),R3
61 007000 030114      5$:   BIT     R1,(R4)    ;Is this block free?
62 007002 001412      BEQ     7$         ;Br if not free
63 007004 005205      INC     R5         ;Advance block number
64 007006 006301      ASL     R1         ;Shift bit mask
65 007010 103002      BCC     3$         ;Br if did not shift bit out of word
66 007012 006101      ROL     R1         ;Shift bit into right end of word
67 007014 005724      TST     (R4)+      ;Point to next word in bit map
68 007016 020503      3$:   CMP     R5,R3      ;Have we found enough space to satisfy request
69 007020 103011      BHIS   4$         ;Br if yes
70 007022 020537 0000000 CMP     R5,VPLAS   ;Have we gone past end of the file?
71 007026 103764      BLO    5$         ;Keep searching if not
72
73 ; This region was not large enough to satisfy the request.
74 ; Save largest region size.
75 ;
76 007030 010503      7$:   MOV     R5,R3      ;Get number of block beyond end of region
77 007032 160203      SUB     R2,R3      ;Calc number of blocks in the region
78 007034 020300      CMP     R3,R0      ;Is new region larger than previous one?
79 007036 101741      BLOS   8$         ;Br if not
80 007040 010300      MOV     R3,R0      ;Save largest region size
81 007042 000737      BR     8$
82
83 ; We found a region large enough to satisfy the request.
84 ; Claim it.
85 ;
86 007044 010205      4$:   MOV     R2,R5      ;Get starting block number
87 007046 005004      CLR     R4         ;Clear high-order for divide
88 007050 071427 000020 DIV     #16.,R4    ;Convert block # to word and bit index
89 007054 012701 000001 MOV     #1,R1      ;Get bit mask
90 007060 072105      ASH     R5,R1      ;Position bit mask
91 007062 006304      ASL     R4         ;Convert word number to byte number
92 007064 063704 000012' ADD     SSMAP,R4   ;Point to word within bit map
93 007070 011603      MOV     (SP),R3    ;Get request size
94 007072 040114      10$:  BIC     R1,(R4)    ;Mark blocks as in use
95 007074 006301      ASL     R1         ;Shift bit mask
96 007076 103002      BCC     9$         ;Br if didn't shift out of word
97 007100 006101      ROL     R1         ;Shift bit back into right end of word
98 007102 005724      TST     (R4)+      ;Advance to next word of bit map
99 007104 077306      9$:   SOB     R3,10$   ;Mark all blocks in region as in use
100 007106 012600      MOV     (SP)+,R0   ;Return original request size in R0
101 007110 000241      CLC
102 007112 000402      BR     13$
103
104 ; Could not find a large enough region
105 ;
106 007114 005726      12$:  TST     (SP)+      ;Pop request size off of stack
107 007116 000261      SEC
108 ; Signal failure on return
109 ; Finished
110 ;
111 007120 012605      13$:  MOV     (SP)+,R5
112 007122 012604      MOV     (SP)+,R4
113 007124 012603      MOV     (SP)+,R3
114 007126 012601      MOV     (SP)+,R1

```

115 007130 000207

RETURN

SWPFRE -- Free space in region swap file

```

1          .SBTTL  SWPFRE -- Free space in region swap file
2          ;-----
3          ; Deallocate space in region swap file.
4          ;
5          ; Inputs:
6          ; R2 = Pointer to region descriptor
7          ;
8 007132 010146 SWPFRE: MOV     R1,-(SP)
9 007134 010446      MOV     R4,-(SP)
10 007136 010546      MOV     R5,-(SP)
11          ;
12          ; Calculate position in bit map of start of region being freed
13          ;
14 007140 032762 000000G 000000G      BIT     #RC$SFA,RC$FLG(R2) ;Has swap file space been allocated?
15 007146 001426      BEQ     9$          ;Br if not -- nothing to free
16 007150 016205 000000G      MOV     RC$BLK(R2),R5 ;Get starting block number
17 007154 005004      CLR     R4          ;Clear high-order for divide
18 007156 071427 000020      DIV     #16.,R4      ;Convert block number to word and bit index
19 007162 012701 000001      MOV     #1,R1        ;Get a bit mask
20 007166 072105      ASH     R5,R1        ;Position bit mask
21 007170 006304      ASL     R4          ;Convert word number to byte number
22 007172 063704 000012'      ADD     SSMAP,R4     ;Point to word in bit map
23 007176 016200 000000G      MOV     RC$PAG(R2),R0 ;Get # 512-byte pages used by region
24          ;
25          ; Set bits in bit map saying blocks are free
26          ;
27 007202 050114 1$:      BIS     R1,(R4)      ;Set bit saying block is free
28 007204 006301      ASL     R1          ;Shift bit mask
29 007206 103002      BCC     2$          ;Br if didn't shift out of word
30 007210 006101      ROL     R1          ;Shift bit back into right end of word
31 007212 005724      TST     (R4)+       ;Point to next word in bit table
32 007214 077006 2$:      SOB     R0,1$      ;Set all blocks free
33          ;
34          ; Say region has no allocated swap file blocks
35          ;
36 007216 042762 000000G 000000G      BIC     #RC$SFA,RC$FLG(R2) ;Say region has no swap file space
37          ;
38          ; Finished
39          ;
40 007224 012605 9$:      MOV     (SP)+,R5
41 007226 012604      MOV     (SP)+,R4
42 007230 012601      MOV     (SP)+,R1
43 007232 000207      RETURN

```

PLSINI -- Initialize PLAS system

```

1          .SBTTL  PLSINI -- Initialize PLAS system
2          ;-----
3          ; Initialize the PLAS system during system startup.
4          ;
5 007234   010246   PLSINI: MOV      R2,-(SP)
6 007236   010546           MOV      R5,-(SP)
7          ;
8          ; Possibly reduce size of PLAS swap file if we don't have room
9          ; enough for the required bit map.
10         ;
11 007240   012705   000000C   MOV      #137774-PLSTOP,R5 ;Get # bytes of available space in seg
12 007244   072527   0000003   ASH      #3,R5           ;Get max # blocks we can hold in map
13 007250   020537   000000G   CMP      R5,VPLAS       ;Compare with requested amt
14 007254   103002           BHIS     2$              ;Br if requested amt ok
15 007256   010537   000000G   MOV      R5,VPLAS       ;Reduce size of PLAS swap file
16         ;
17         ; Allocate a bit map to keep track of free blocks in the swap file
18         ;
19 007262   012705   007326'   2$:     MOV      #PLSTOP,R5 ;Get pointer past top of PLAS code
20 007266   010537   000012'   MOV      R5,SSMAP       ;Set pointer to base of table
21 007272   013702   000000G   MOV      VPLAS,R2       ;Get # blocks in PLAS swap file
22 007276   062702   000017   ADD      #15.,R2        ;Bound up to word
23 007302   072227   177774   ASH      #-4,R2         ;Get # words needed for bit map
24 007306   042702   170000   BIC      #170000,R2     ;Clear sign extension
25 007312   012725   177777   1$:     MOV      #177777,(R5)+ ;Turn all bits on in table
26 007316   077203           SOB      R2,1$
27         ;
28         ; Finished
29         ;
30 007320   012605           MOV      (SP)+,R5
31 007322   012602           MOV      (SP)+,R2
32 007324   000207           RETURN
33         ;
34         ; Top of PLAS code
35         ;
36 007326           PLSTOP:
37         .END

```

Errors detected: 0

\*\*\* Assembler statistics

Work file reads: 0  
 Work file writes: 0  
 Size of work file: 189 Words ( 1 Pages)  
 Size of core pool: 18176 Words ( 71 Pages)  
 Operating system: RT-11

Elapsed time: 00:01:00.11  
 ,LP: TSPLAS=DK: TSPLAS/C/N: SYM









WCBUMP	9-107	10-49	10-67	11-11	12-17	21-16	23-8#	36-26
WS. CRW	1-35	9-10	9-84					
WS. ELW	1-35	9-10	9-101	11-21				
WS. MAP	1-36	9-119	13-51	13-62	13-90			
WS. DVR	1-35	9-85						
WS. UNM	1-35	9-10	9-109	10-33	12-22			

DIE	1-57#	29-6	35-6		
OCALL	1-67#	5-121	5-123	29-31	35-28

Table of contents

2-	1	SWAPER -- Job swapper
3-	1	SWPPRI -- Do job swapping based on states and priorities
4-	1	SWPFUN -- Perform special swapping functions
5-	1	SWPLOCK -- Lock job in low memory area
6-	1	SWPRGN -- Create shared global PLAS region
7-	1	SWPMEM -- Do forced outswap of jobs
8-	1	OUTSWP -- Outswap a job
9-	1	OSWPGO -- Initiate outswap I/O operation
11-	1	INSWP -- Inswap a job
12-	1	ISWPGO -- Initiate inswap I/O operation
14-	1	INSFIN -- Finished inswap
15-	1	SCPGET -- Get a free swap command packet
16-	1	SCPFRE -- Free a swap command packet

```

1          .TITLE  TSSWAP -- TSX-Plus job swapper
2 000000   .CSECT  TSSWAP
3          .ENABL  LC
4          .ENABL  AMA
5          .DSABL  GBL
6 000000 075150 TSSWAP: .RAD50 /SWP/          ;Overlay ID
7          ;
8          ; TSSWAP contains the routines related to job swapping.
9          ;
10         ; Copyright (c) 1980,1981,1982,1983,1984,1985.
11         ; S&H Computer Systems, Inc.
12         ; Nashville, Tennessee USA
13         ; All rights reserved.
14         ;
15         ; Global definitions
16         ;
17         .GLOBL  TSSWAP
18         .GLOBL  SWAPER, INSFIN, OUTSWP
19         .GLOBL  SCPGET
20         ;
21         ; Global references
22         ;
23         .GLOBL  OVRHC, INBSY, OUTBSY
24         .GLOBL  RUNQHD, IOHLM, VSWPFL, LSW6, FREMEM, Q. COMP, SWPCOT
25         .GLOBL  LBASE, SWPCHN, LQLINK, GETSYQ, LQUAN, MEMSWP, $INCOR
26         .GLOBL  LNSBLK, S$CPU, SEGOUT, $MLOCK, INTPRI, EM$SIE
27         .GLOBL  IASEGS, EM$SFO, LSTSL, LMEMIN, LSW, LSW7, SEGIN, LMINQ
28         .GLOBL  LIOHLD, $MAPOK, ENQHD, GETMEM, S$RUN, S$RT, SCPFHD
29         .GLOBL  $NDMEM, CS$EOF, Q. WCNT, MAPUSR, LSWPBK, SWPJOB, PSW, LIOCNT
30         .GLOBL  Q. BLKN, S$IOWT, LNBLKS, RUNQTL, SYQID, VSWPSL, DOSCHD, Q. PAR
31         .GLOBL  LSTATE, SWPPOS, CS$ERR, VCORTM, SP$CMD, SP$JOB, EM$LMF
32         .GLOBL  EM$NSP, SP$LNK, RC$PAG, RC$BAS, TRYRON, SP$DW1
33         .GLOBL  FORCEX, $NLOCK, S$WFM
34         ;
35         ; Macro definitions
36         ;
37         ; Macro to print an error message when a system crash occurs.
38         ;
39         ; Arguments:
40         ; MSG = Name of error message to print.
41         ; ARG = (Optional) argument value to display with error message.
42         ;
43         .GLOBL  DIEMSG, DIEARG, SYSHLT
44         .MACRO  DIE      MSG, ARG
45         MOV     MSG, @DIEMSG
46         .IF    NB, ARG
47         MOV     ARG, @DIEARG
48         .ENDC
49         CALL   @SYSHLT
50         .ENDM  DIE
51         ;
52         ; Macro definition for call global routines residing in mapped system regions
53         ;
54         .MACRO  OCALL  ENTADD
55         .IF    B, ENTADD
56         .ERROR ;OCALL SPECIFIED WITH NO ENTRY ADDRESS
57         .MEXIT

```

```

58          . ENDC
59          CALL   OVRHC          ; CALL THE OVERLAY HANDLER
60          . WORD ENTADD        ; SPECIFY THE ENTRY POINT
61          . ENDM
62          ;
63          ;-----
64          ; Macros to enable and disable interrupts.
65          ; DISABL raises the CPU priority level to 7.
66          ; ENABL lowers the priority to the current operating
67          ; priority which is stored in INTPRI.
68          ;
69          . MACRO  DISABL          ; DISABLE INTERRUPTS
70          BIS     #340, @#PSW
71          . ENDM  DISABL
72          ;
73          . MACRO  ENABL          ; RESTORE INTERRUPT STATUS
74          BIC     INTPRI, @#PSW
75          . ENDM  ENABL
76          ;
77          ;-----
78          ; Data areas
79          ;
80 000002 000000 SCPQHD: . WORD 0          ; Head of pending swap command packets
81 000004 000000 SCPQTL: . WORD 0          ; Tail of pending swap command packets
82 000006 000000 SWPSIZ: . WORD 0          ; Number of 512-byte blocks being swapped
83 000010 000000 SWPBLK: . WORD 0          ; Block number in swap file where I/O is to go
84 000012 000000 SWPBAS: . WORD 0          ; Page number in memory of swap area
85          ;
86          ; Byte data
87          ;
88 000014 000    LSTIN: . BYTE 0          ; Last job inswapped
89 000015 000    LSTOUT: . BYTE 0         ; Last job outswapped
90 000016 000    LSTOPR: . BYTE 0        ; Last operation: 0==>Outswap, 1==>Inswap
91          . EVEN

```

1  
 2  
 3  
 4  
 5  
 6  
 7  
 8 000020  
 9  
 10  
 11  
 12 000020 005737 000000G  
 13 000024 001403  
 14 000026 004737 000524'  
 15 000032 103405  
 16  
 17  
 18  
 19  
 20  
 21 000034 004737 000322'  
 22 000040 103402  
 23  
 24  
 25  
 26 000042 004737 000050'  
 27  
 28  
 29  
 30 000046 000207

```

.SBTTL  SWAPER -- Job swapper
-----
; SWAPER is called to see if jobs should be swapped into or out of memory.
; The basic task of the swapper is simple:  Keep in memory the highest
; priority jobs that want to run and swap out as few of the lowest priority
; jobs as necessary to accomplish this.
;
SWAPER:
;
; See if we need to outswap any users who are waiting for memory expansion.
;
;          TST      MEMSWP      ;Any jobs need memory-expansion outswap?
;          BEQ      12$          ;Br if not
;          CALL     SWPMEM       ;Do memory expansion outswap
;          BCS      9$           ;Br if we started a swap
;
; We have finished doing all forced outswapping.
; See if there are any special swapper requests pending such as
; locking jobs in low memory or creating shared PLAS regions.
;
12$:      CALL     SWPFUN       ;Check for special swapper functions
;          BCS      9$           ;Br if special function started
;
; See if we need to do any job swapping based on job states and priorities.
;
;          CALL     SWPPRI       ;Do normal job swapping
;
; Finished
;
9$:      RETURN
  
```

SWPPRI -- Do job swapping based on states and priorities

```

1          .SBTTL  SWPPRI -- Do job swapping based on states and priorities
2          ;-----
3          ; SWPPRI is the portion of the job swapper that is responsible for doing
4          ; job swapping to keep in memory the jobs with the highest states and
5          ; priorities.
6          ;
7 000050 010146 SWPPRI: MOV      R1,-(SP)
8 000052 010246      MOV      R2,-(SP)
9          ;
10         ; Scan down the job list from highest priority to lowest priority
11         ; looking for a job that is ready to run but is not in memory.
12         ;
13 000054         DISABL          ;** Disable **
14 000062 113701 000000G      MOVB   RUNGHD,R1      ;Get head of job list
15 000066 001507         BEQ     8$              ;Br if no jobs in list
16 000070 026127 000000G 000000G 2$:    CMP     LSTATE(R1),#S#$RUN ;Does this job want to run?
17 000076 101103         BHI     8$              ;Br if there are no ready-to-run jobs
18 000100 032761 000000G 000000G      BIT     #$INCOR,LSW(R1) ;Is this job in memory now?
19 000106 001404         BEQ     1$              ;Br if not -- go try to get it in
20 000110 116101 000000G      MOVB   LQLINK(R1),R1 ;Advance to next job in list
21 000114 001365         BNE     2$              ;Br if more to check
22 000116 000473         BR      8$              ;No more jobs in list
23         ;
24         ; We found a job that is ready to run but is not now in memory.
25         ;
26         ; See if there is enough free memory available to bring in the job.
27         ;
28 000120 1$:      ENABL          ;** Enable **
29 000126 004737 000000G      CALL   GETMEM      ;See if there is enough memory for the job
30 000132 103403         BCS     5$              ;Br if there is not enough free memory
31         ;
32         ; There is enough free memory for the job.
33         ; Swap it in.
34         ;
35 000134 004737 001250'      CALL   INSWP      ;In swap the job
36 000140 000465         BR      9$              ;Finished
37         ;
38         ; There is not sufficient free memory space available for the job we want.
39         ; See if there are some lower priority jobs that should be outswapped to
40         ; make room.
41         ;
42 000142 010102 5$:      MOV     R1,R2          ;Save # of job we want to bring in
43         ;
44         ; Search job list from lowest priority job toward higher priorities looking
45         ; for a job to outswap.
46         ;
47 000144 10$:     DISABL          ;** Disable **
48 000152 113701 000000G      MOVB   RUNGTL,R1      ;Point to tail of job list
49 000156 001453         BEQ     8$              ;Br if list empty
50 000160 020102 11$:     CMP     R1,R2          ;Have we reached job we want to bring in?
51 000162 001451         BEQ     8$              ;If yes, there is no lower prio job to swap
52         ;
53         ; Found a lower priority job.
54         ; See if it is in memory now.
55         ;
56 000164 032761 000000G 000000G      BIT     #$INCOR,LSW(R1) ;Is the job in memory now?
57 000172 001434         BEQ     3$              ;Br if not

```

```

58 ;
59 ; We cannot outswap a job if it is locked in memory
60 ;
61 000174 032761 000000G 000000G BIT    #MLOCK,LSW6(R1);Is job locked in memory?
62 000202 001030 BNE    3$          ;Br if yes
63 ;
64 ; See if job has gotten minimum core residency time since last inswap.
65 ;
66 000204 005761 000000G TST    LMINQ(R1)      ;Has min core residency time expired?
67 000210 001417 BEQ    4$          ;Br if yes
68 ;
69 ; Job has not yet gotten its minimum core residency time.
70 ; If job we want to inswap is in a high-priority real-time state,
71 ; ignore the minimum core residency time and do the inswap anyway.
72 ;
73 000212 026227 000000G 000000G CMP    LSTATE(R2),#S#$RT;Is job we want to inswap in real-time state?
74 000220 101413 BLOS  4$          ;Br if yes
75 ;
76 ; If job is waiting for something other than I/O, ignore min core time.
77 ;
78 000222 016100 000000G MOV    LSTATE(R1),RO  ;Get current job state
79 000226 020027 000000G CMP    RO,#S#$RUN    ;Is job in an executable state?
80 000232 103403 BLO   22$         ;Br if yes -- honor min core time
81 000234 020027 000000G CMP    RO,#S#$IOWT   ;Is job in I/O wait state?
82 000240 001003 BNE   4$          ;Br if not -- ignore min core time
83 ;
84 ; We would like to outswap this job but its minimum core residency
85 ; time prevents the swap. Set a flag that will cause the clock
86 ; interrupt routine to recall the scheduler when the minimum core time
87 ; expires.
88 ;
89 000242 110137 000000G 22$:  MOVB   R1,SWPCOT   ;Set flag for clock interrupt routine
90 000246 000406 BR     3$          ;Don't outswap job
91 ;
92 ; We want to outswap this job.
93 ; If it has I/O active now, set flag to hold future I/O starts.
94 ;
95 000250 105761 000000G 4$:  TSTB   LIOCNT(R1)   ;Does job have any I/O active now?
96 000254 001407 BEQ   6$          ;Br if not -- swap it
97 000256 012761 000000G 000000G MOV    #IOHLM,LIOHLD(R1);Hold future I/O starts for job
98 ;
99 ; Job we looked at is not eligible to be outswapped.
100 ; Try next job in list.
101 ;
102 000264 116101 000001G 3$:  MOVB   LQLINK+1(R1),R1 ;Get # of next higher priority job
103 000270 001333 BNE   11$         ;Br if one exists
104 000272 000405 BR     8$
105 ;
106 ; We have found a job to outswap.
107 ;
108 000274 6$:  ENABL                   ;** Enable **
109 000302 004737 000634' CALL   OUTSWP       ;Initiate the outswap
110 ;
111 ; We've done all we can for now.
112 ;
113 000306 8$:  ENABL                   ;** Enable **
114 ;

```

```
115          ; Finished
116          ;
117 000314 012602 9$:      MOV      (SP)+, R2
118 000316 012601      MOV      (SP)+, R1
119 000320 000207      RETURN
```

SWPFUN -- Perform special swapping functions

```

1          .SBTTL  SWPFUN -- Perform special swapping functions
2          ;-----
3          ; SWPFUN is called after any forced outswapping is completed to perform
4          ; any special swapping functions such as locking jobs in memory or
5          ; allocating memory for shared PLAS regions.
6          ; Commands to SWPFUN are queued by use of swapper command packets.
7          ;
8          ; Outputs:
9          ;   C-flag cleared ==> No special function in progress.
10         ;   C-flag set ==> Special function in progress,
11         ;                   don't start any other swapping activity.
12         ;
13 000322  010546  SWPFUN: MOV      R5, -(SP)
14         ;
15         ;   Get pointer to 1st pending command packet
16         ;
17 000324  013705  000002' 2$:      MOV      SCPQHD, R5      ;Get pointer to 1st pending command packet
18 000330  001002          BNE      1$          ;Br if there is a pending command
19         ;
20         ;   There is no pending command
21         ;
22 000332  000241          CLC          ;Say to continue normal swapping
23 000334  000405          BR       9$
24         ;
25         ;   Call appropriate routine to process the command
26         ;
27 000336  116500  000000G 1$:      MOVB     SP$CMD(R5), R0 ;Get command value
28 000342  004770  000354'  CALL     @SWPVEC(R0) ;Call routine
29         ;
30         ;   The swapping action routines take care of freeing the command packet
31         ;   and return with the c-flag set or cleared.
32         ;   If the C-flag is cleared on return, go back and see if there is
33         ;   another command pending.
34         ;
35 000346  103366          BCC      2$          ;Br if we should resume swapping activities
36         ;
37         ;   Finished
38         ;
39 000350  012605  9$:      MOV      (SP)+, R5
40 000352  000207          RETURN
41         ;
42         ;   Vector of addresses for swapping action routines
43         ;
44 000354  000360'  SWPVEC: .WORD   SWPLOK          ;SA$LOK -- Lock job in low memory
45 000356  000440'          .WORD   SWPRGN          ;SA$RGN -- Allocate shared region

```

SWPLOCK -- Lock job in low memory area

```

1          .SBTTL  SWPLOCK -- Lock job in low memory area
2          ;-----
3          ; SWPLOCK is a swapper action routine called to swap a job into a low
4          ; memory region and lock the job in memory.
5          ;
6          ; Inputs:
7          ;   R5 = Pointer to swapper command packet.
8          ;
9          ; Outputs:
10         ;   C-flag set ==> Don't start any other swapping activities.
11         ;   C-flag cleared ==> OK to continue with normal swapping activities.
12         ;
13 000360 010146 SWPLOCK: MOV      R1,-(SP)
14         ;
15         ;   Get number of job that is to be locked in memory
16         ;
17 000362 116501 000000G      MOVVB  SP$JOB(R5),R1  ;Get # of job to lock in memory
18         ;
19         ;   Set flag saying job is locked in memory
20         ;
21 000366 052761 000000G 000000G      BIS      #$MLOCK,LSW6(R1);Say job is locked in memory
22 000374 042761 000000G 000000G      BIC      #$NLOCK,LSW6(R1);Clear needs-lock flag
23         ;
24         ;   Allocate memory space for the job.
25         ;   (Note, the memory space should be at the bottom of memory)
26         ;
27 000402 004737 000000G      CALL     GETMEM          ;Get memory for the job
28 000406 103005              BCC     1$                ;We should always branch
29 000410              DIE     #$EM$LMF          ;Should never happen
30         ;
31         ;   Swap the job into the space we allocated
32         ;
33 000422 004737 001250'      1$:   CALL     INSWP          ;Inswap the job
34         ;
35         ;   Free the swap command packet
36         ;
37 000426 004737 001716'      CALL     SCPFRE          ;Free the swap command packet
38         ;
39         ;   Finished
40         ;
41 000432 000261              SEC              ;Don't start any other swapping
42 000434 012601              MOV      (SP)+,R1
43 000436 000207              RETURN

```

```

1          .SBTTL  SWPRGN -- Create shared global PLAS region
2          ;-----
3          ; SWPRGN is a swapper special function routine which allocates memory
4          ; for a shared global PLAS region after swapping jobs out of
5          ; memory to free up space for the region.
6          ;
7          ; Inputs:
8          ;   R5 = Pointer to swapper command packet.
9          ;
10         ; Outputs:
11         ;   C-flag set ==> Don't start any other swapping activities.
12         ;   C-flag cleared ==> OK to continue with normal swapping activities.
13         ;
14 000440 010146 SWPRGN: MOV      R1,-(SP)
15 000442 010246      MOV      R2,-(SP)
16         ;
17         ; Get a pointer to the Region Control Block out of the swap command packet
18         ;
19 000444 016501 000000G      MOV      SP$DW1(R5),R1 ;Get pointer to Region Control Block
20         ;
21         ; Try to allocate space for the region
22         ;
23 000450 016100 000000G      MOV      RC$PAG(R1),R0 ;Get # 512-byte pages needed for region
24 000454 012761 177777 000000G      MOV      #-1,RC$BAS(R1) ;Say address =-1 in case allocation fails
25 000462 004737 000000G      CALL     TRYRGN          ;Try to allocate space for region
26 000466 103404          BCS     1$              ;Br if cannot allocate (should never happen)
27 000470 072227 0000003      ASH     #3,R2          ;Convert page # to 64-byte block #
28 000474 010261 000000G      MOV      R2,RC$BAS(R1) ;Save pointer to region base in RCB
29         ;
30         ; Make sure the job creating the region is in an executable state
31         ;
32 000500 116501 000000G 1$:  MOV     SP$JOB(R5),R1 ;Get # of job that is creating region
33 000504 004737 000000G      CALL     FORCEX          ;Make sure it is in executable state
34         ;
35         ; Return the swap command packet
36         ;
37 000510 004737 001716'      CALL     SCPFRE          ;Free the command packet
38         ;
39         ; Finished
40         ;
41 000514 000241          CLC              ;Say normal swapping may continue
42 000516 012602          MOV     (SP)+,R2
43 000520 012601          MOV     (SP)+,R1
44 000522 000207          RETURN

```

SWPMEM -- Do forced outswap of jobs

```

1          .SBTTL  SWPMEM -- Do forced outswap of jobs
2          ;-----
3          ; SWPMEM is called to swap out of memory all jobs that have the #NDMEM
4          ; flag set in LSW.
5          ;
6          ; Outputs:
7          ; C-flag cleared ==> All flagged jobs have been outswapped.
8          ; C-flag set ==> Some jobs remain to be swapped.
9          ;
10         000524 010146 SWPMEM: MOV      R1,-(SP)
11         000526 010246          MOV      R2,-(SP)
12         ;
13         ; Begin loop that checks each job to see if it needs to be outswapped
14         ;
15         000530 005002          CLR      R2          ;Say not waiting on I/O to stop
16         000532 012701 000000G MOV      #LSTSL,R1      ;Get index # of last line
17         ;
18         ; See if this job needs to be outswapped
19         ;
20         000536 032761 000000G 000000G 15$: BIT      #NDMEM,LSW(R1) ;Is this job waiting for memory expansion?
21         000544 001416          BEQ      13$          ;Br if not
22         ;
23         ; We found a job that needs to be outswapped.
24         ; See if it is still in memory.
25         ;
26         000546 032761 000000G 000000G          BIT      #INCOR,LSW(R1) ;Is this job still in memory?
27         000554 001412          BEQ      13$          ;Br if not
28         ;
29         ; The job is in memory, see if it has any I/O active.
30         ;
31         000556 105761 000000G          TSTB   LIOCNT(R1)      ;Does this job have any I/O active?
32         000562 001003          BNE      14$          ;Br if yes
33         ;
34         ; Outswap this job.
35         ;
36         000564 004737 000634'          CALL   OUTSWP          ;Start outswap on the job
37         000570 000411          BR      9$
38         ;
39         ; We cannot outswap this job because it has I/O active.
40         ; Hold future I/O starts for the job.
41         ;
42         000572 012761 000000G 000000G 14$: MOV      #IOHLM,LIOHLD(R1);Hold I/O for this job
43         000600 005202          INC      R2          ;Remember we are waiting for I/O to stop
44         ;
45         ; See if there are any other jobs that need to be outswapped.
46         ;
47         000602 162701 000002          13$: SUB      #2,R1          ;Go check next job
48         000606 001353          BNE      15$
49         000610 005702          TST     R2          ;Do we need to wait for I/O to stop?
50         000612 001402          BEQ     10$          ;Br if not
51         ;
52         ; We have not finished all outswaps.
53         ;
54         000614 000261          9$: SEC          ;Signal that outswapping still needed
55         000616 000403          BR      11$
56         ;
57         ; We have finished outswapping all flagged jobs

```

```
58 ;  
59 000620 005037 000000G 10$: CLR MEMSWP ; Say no more jobs to force out of memory  
60 000624 000241 CLC ; Signal that all outswapping is finished  
61 ;  
62 ; Finished  
63 ;  
64 000626 012602 11$: MOV (SP)+, R2  
65 000630 012601 MOV (SP)+, R1  
66 000632 000207 RETURN
```

```

1          .SBTTL  OUTSWP -- Outswap a job
2          ;-----
3          ; OUTSWP is called to swap a job out of memory.
4          ; The job is immediately marked as not in memory.
5          ; When the swap completes, the memory region used by the job
6          ; is marked as free.
7          ;
8          ; Inputs:
9          ; R1 = Job index number of job to be outswapped.
10         ; LBASE(R1) = Base page number allocated to job.
11         ; LNBLKS(R1) = Number of pages allocated to job.
12         ; LSWPBK(R1) = Swap file disk block # of swap area for job.
13         ;
14         ; Outputs:
15         ; SWPBSY = Number of job being outswapped.
16         ; MEMMAP = Updated to show area used by job is free.
17         ;
18 000634 110137 000000G OUTSWP: MOVB    R1,OUTBSY    ;Set flag saying outswap busy
19 000640 110137 000015'   MOVB    R1,LSTOUT   ;Save history of last job outswapped
20 000644 105037 000016'   CLRB    LSTOPR    ;Say outswap last thing done
21         ;
22         ; Mark job as no longer in memory so scheduler won't try to run it
23         ;
24 000650 042761 000000G 000000G   BIC     #$INCOR,LSW(R1) ;Clear in-memory flag for the job
25         ;
26         ; Indicate that memory mapping is not set up for this job
27         ;
28 000656 042761 000000G 000000G   BIC     #$MAPOK,LSW7(R1);Invalidate memory mapping data in ctxt blk
29 000664 120137 000000G           CMPB    R1,MAPUSR    ;Is memory map currently loaded for this job?
30 000670 001002           BNE     2$             ;Br if not
31 000672 105037 000000G           CLRB    MAPUSR     ;Say memory map no longer valid
32         ;
33         ; See if we have to outswap any PLAS regions for the job
34         ;
35 000676 005761 000000G 2$:     TST     LNSBLK(R1)   ;Any PLAS regions associated with job?
36 000702 001404           BEQ     1$             ;Br if no PLAS regions
37 000704           OCALL  SEGOUT    ;Start outswap of PLAS regions
38 000712 103036           BCC     9$             ;Br if PLAS outswap started
39         ;
40         ; Select a slot in the swap file for this job
41         ;
42 000714 013702 000000G 1$:     MOV     VSWPSL,R2    ;Get # job slots in swap file
43 000720 013700 000000G           MOV     SWPJOB,R0    ;Point to table with job #'s in each slot
44 000724 005720 4$:     TST     (R0)+      ;Is this slot free?
45 000726 001406           BEQ     3$             ;Br if yes
46 000730 077203           SOB     R2,4$       ;Loop if more slots to check
47 000732           DIE     #EM$SFO    ;Swap file overflow
48 000744 010140 3$:     MOV     R1,-(R0)   ;Claim swap file slot for our job
49 000746 163700 000000G           SUB     SWPJOB,R0    ;Calculate offset into table
50 000752 063700 000000G           ADD     SWPPOS,R0    ;Point into table with block #'s
51 000756 011061 000000G           MOV     (R0),LSWPBK(R1);Set block # in swap file for this job
52         ;
53         ; Set up information to control swap of job root and context block
54         ;
55 000762 016137 000000G 000006'   MOV     LNBLKS(R1),SWPSIZ ;Number of 512-byte blocks to swap
56 000770 016137 000000G 000010'   MOV     LSWPBK(R1),SWPBLK ;Block number in swap file
57 000776 016137 000000G 000012'   MOV     LBASE(R1),SWPBAS  ;Base 512-byte block number in memory

```

```
58 001004 004737 001012'          CALL    OSWPG0          ;Initiate the outswap
59                                     ;
60                                     ; Finished starting outswap
61                                     ;
62 001010 000207          9$:    RETURN
```

OSWPGO -- Initiate outswap I/O operation

```

1          .SBTTL  OSWPGO -- Initiate outswap I/O operation
2          ;-----
3          ; OSWPGO is called to initiate an outswap I/O operation.
4          ;
5          ; Inputs:
6          ;   SWPSIZ = Number of 512-byte pages to be written.
7          ;   SWPBAS = Starting page number within memory
8          ;   SWPBLK = Starting block number within swap file.
9          ;
10         OSWPGO: MOV      R1,-(SP)
11         ;
12         ; Get a system I/O queue element
13         ;
14         001012  010146          MOV      #SWPCHN,R1      ;Point to system swap channel
15         001020  004737  000000G  CALL      GETSYQ      ;Get a system I/O queue element (addr. in R1)
16         ;
17         ; Set up I/O queue element for the transfer
18         ;
19         001024  063761  000010' 000000G  ADD      SWPBLK,Q.BLKN(R1) ;Set swap file block number for transfer
20         001032  013700  000012'          MOV      SWPBAS,R0      ;Get base 512-byte block number within memory
21         001036  072027  000003          ASH      #3,R0        ;Convert to 64-byte block number
22         001042  010061  000000G          MOV      R0,Q.PAR(R1)  ;Set memory address in queue element
23         001046  013700  000006'          MOV      SWPSIZ,R0     ;Get # 512-byte blocks to be written
24         001052  020027  000177          CMP      R0,#127.     ;We can write a max of 127 per I/O operation
25         001056  101402          BLOS     1$           ;Br if we can write all that is left
26         001060  012700  000177          MOV      #127.,R0     ;Truncate write to 127 blocks this time
27         001064  160037  000006'          1$: SUB      R0,SWPSIZ   ;Calc # 512-byte blocks left to transfer
28         001070  060037  000012'          ADD      R0,SWPBAS    ;Calc memory page for next transfer
29         001074  060037  000010'          ADD      R0,SWPBLK    ;Calc swap file block # for next transfer
30         001100  000300          SWAB     R0           ;Convert # blocks to # words to write
31         001102  005400          NEG      R0           ;Negative ==> Write operation
32         001104  010061  000000G          MOV      R0,Q.WCNT(R1) ;Set word count in I/O queue element
33         001110  012761  001126' 000000G  MOV      #OUTCMP,Q.COMP(R1) ;Set address of I/O completion routine
34         ;
35         ; Queue the I/O request
36         ;
37         001116  004737  000000G          CALL     SYQIO        ;Queue the I/O request
38         ;
39         ; Return. I/O completion will call OUTCMP.
40         ;
41         001122  012601          9$: MOV      (SP)+,R1
42         001124  000207          RETURN

```

OSWPGO -- Initiate outswap I/O operation

```

1          ; -----
2          ;   Outswap I/O completion routine.
3          ;
4          ;   Check for swap file I/O errors
5          ;
6 001126   032737   000000C 000000G OUTCMP: BIT      #CS$ERR!CS$EOF,SWPCHN ;Did any swap file I/O error occur?
7 001134   001405                   BEQ      3$          ;Br if not
8 001136                   DIE      #EM$SIE          ;Abort if I/O error on swap
9          ;
10         ;   See if there is any part of the job remaining to be swapped
11         ;
12 001150   005737   000006' 3$:      TST      SWPSIZ          ;Any part of job remaining to be swapped?
13 001154   001403                   BEQ      4$          ;Br if not
14 001156   004737   001012'       CALL     OSWPGO          ;Initiate transfer of remainder of job
15 001162   000431                   BR       9$          ;We will be called again when transfer ends
16         ;
17         ;   Release the memory space occupied by the job just outswapped.
18         ;
19 001164   113701   000000G 4$:      MOV      OUTBSY,R1          ;Get job # being outswapped
20 001170   016102   000000G       MOV      LBASE(R1),R2        ;Get base page #
21 001174   016100   000000G       MOV      LNBLKS(R1),R0       ;Get # blocks assigned to job
22 001200   004737   000000G       CALL     FREMEM            ;Release the memory space
23 001204   005061   000000G       CLR      LBASE(R1)         ;Say job is no longer occupying memory
24 001210   042761   000000G 000000G BIC      #N$MEM,LSW(R1)     ;Say job no longer needs outswap for mem expn.
25         ;
26         ;   If the job is in a suspended state waiting for memory expansion,
27         ;   put in in an executable state now.
28         ;
29 001216   026127   000000G 000000G       CMP      LSTATE(R1),#S$WFM ;Is job waiting for memory expansion?
30 001224   001004                   BNE      2$          ;Br if not
31 001226   012700   000000G       MOV      #S$CPU,R0         ;Requeue job in run state
32 001232   004737   000000G       CALL     ENQHD
33         ;
34         ;   Finished. Request a job scheduler cycle.
35         ;
36 001236   105037   000000G 2$:      CLRB     OUTBSY          ;Say outswap finished
37 001242   105237   000000G       INCB     DOSCHD           ;Request a job scheduler cycle
38 001246   000207                   9$:      RETURN

```

INSWP -- Inswap a job

```

1          .SBTTL  INSWP  -- Inswap a job
2          ;-----
3          ; INSWP is called to initiate the inswap of a job.
4          ;
5          ; Inputs:
6          ; R1 = Job index number of job to be inswapped.
7          ; LBASE(R1) = Base page assigned to the job.
8          ; LNBLKS(R1)= # 256-word pages currently used by job.
9          ; LMEMIN(R1)= # 256-word pages to be assigned to job after inswap.
10         ; LSWPBJ(R1)= Disk file block number where job is stored.
11         ;
12         ; Outputs:
13         ; LNBLKS(R1) is set to LMEMIN(R1) after swap finishes (used for
14         ; memory expansion)
15         ;
16 001250 110137 000000G INSWP:  MOVB   R1,INBSY      ;Say an inswap is in progress
17 001254 110137 000014'      MOVB   R1,LSTIN      ;Remember last job inswapped
18 001260 112737 000001 000016'  MOVB   #1,LSTOPR     ;Remember inswap was last operation
19         ;
20         ; If there are no blocks assigned to the job, then we are initializing
21         ; the job and there is nothing to be read into memory.  In this case,
22         ; call the completion routine.
23         ;
24 001266 005761 000000G      TST    LNBLKS(R1)    ;Any blocks assigned to job?
25 001272 001500              BEQ    INCMP2      ;Br if not -- Go call completion routine
26         ;
27         ; We need to do an inswap for the job.
28         ; Set up information to describe the I/O transfer
29         ;
30 001274 016137 000000G 000006'  MOV    LNBLKS(R1),SWPSIZ ;Number of 512-byte blocks to read
31 001302 016137 000000G 000010'  MOV    LSWPBK(R1),SWPBLK ;Starting block number in swap file
32 001310 016137 000000G 000012'  MOV    LBASE(R1),SWPBAS  ;Starting page number in memory
33         ;
34         ; Initiate the I/O operation
35         ;
36 001316 004737 001324'      CALL   ISWPGO        ;Start the inswap
37         ;
38         ; Finished
39         ;
40 001322 000207              RETURN

```

ISWPGO -- Initiate inswap I/O operation

```

1          .SBTTL  ISWPGO -- Initiate inswap I/O operation
2          ;-----
3          ; ISWPGO is called to initiate an inswap I/O operation.
4          ;
5          ; Inputs:
6          ; SWPBAS = Base 512-byte page number of job in memory.
7          ; SWPSIZ = Number of 512-byte pages to read.
8          ; SWPBLK = Base block number in swap file.
9          ;
10         001324 010146 ISWPGO: MOV      R1,-(SP)
11         ;
12         ; Get a system I/O queue element
13         ;
14         001326 012701 000000G      MOV      #SWPCHN,R1      ;Get address of swap file channel
15         001332 004737 000000G      CALL     GETSYQ        ;Get a system I/O queue element
16         ;
17         ; Set up I/O queue element for the transfer
18         ;
19         001336 063761 000010' 000000G      ADD      SWPBLK,Q.BLKN(R1) ;Set base block number in swap file
20         001344 013700 000012'          MOV      SWPBAS,R0      ;Get memory page number
21         001350 072027 000003          ASH     #3,R0          ;Convert to 64-byte page number
22         001354 010061 000000G      MOV      RO,Q.PAR(R1)   ;Set memory address in I/O queue element
23         001360 013700 000006'          MOV      SWPSIZ,R0      ;Get # 512-byte pages to read
24         001364 020027 000177          CMP     RO,#127.       ;Can't read more than 127 per I/O operation
25         001370 101402          BLOS   1$             ;Br if can read remainder of program this time
26         001372 012700 000177          MOV     #127.,R0      ;Truncate and only read 127 this time
27         001376 160037 000006'          1$:    SUB     RO,SWPSIZ   ;Calc # blocks left to read next time
28         001402 060037 000012'          ADD     RO,SWPBAS     ;Calc memory base address for next transfer
29         001406 060037 000010'          ADD     RO,SWPBLK     ;Calc swap file block number for next transfer
30         001412 000300          SWAB   RO             ;Cvt # blocks to # words to read
31         001414 010061 000000G      MOV     RO,Q.WCNT(R1)  ;Set word count in I/O queue element
32         001420 012761 001436' 000000G      MOV     #INCMP,Q.COMP(R1);Set address of I/O completion routine
33         ;
34         ; Queue the I/O request
35         ;
36         001426 004737 000000G      CALL     SYQID        ;Initiate the I/O operation
37         ;
38         ; Return for now.
39         ; I/O completion will call INCMP.
40         ;
41         001432 012601          MOV     (SP)+,R1
42         001434 000207          RETURN

```

ISWPGO -- Initiate inswap I/O operation

```

1          ;-----
2          ;   Inswap I/O completion routine.
3          ;
4          ;   Check for swap file I/O errors
5          ;
6 001436 032737 000000C 000000G INCMP: BIT      #CS$ERR!CS$EOF,SWPCHN ;Did any swap file I/O error occur?
7 001444 001405          BEQ      1$              ;Br if not
8 001446          DIE      #EM$SIE              ;Abort system if swap file I/O error
9          ;
10         ;   See if there is more of job root remaining to be inswapped.
11         ;
12 001460 005737 000006' 1$:   TST      SWPSIZ              ;Is any part of job remaining to be read?
13 001464 001403          BEQ      INCMP2              ;Br if not
14 001466 004737 001324'          CALL     ISWPGO              ;Start read of rest of job
15 001472 000207          RETURN             ;We will be called again when that I/O ends
16         ;
17         ;   Set # of pages in use by job in case it was outswapped due to a
18         ;   memory expansion request.
19         ;
20 001474 113701 000000G INCMP2: MOVB     INBSY,R1              ;Get index number of job being inswapped
21 001500 016100 000000G          MOV      LMEMIN(R1),R0          ;Get total # 512-byte pages used by job
22 001504 166100 000000G          SUB      LNSBLK(R1),R0          ;Subtract # pages used by PLAS regions
23 001510 010061 000000G          MOV      R0,LNBLKS(R1)          ;Set # pages used by root of job
24         ;
25         ;   See if we need to inswap any PLAS regions for this job
26         ;
27 001514 005761 000000G          TST      LNSBLK(R1)          ;Any PLAS segments to inswap?
28 001520 001407          BEQ      2$              ;Br if not
29 001522          OCALL     IASEGS              ;Set up memory addresses for PLAS regions
30 001530          OCALL     BEGIN              ;Start inswap of PLAS regions
31 001536 103004          BCC      9$              ;Br if PLAS inswap started
32         ;
33         ;   Mark job as in memory now.
34         ;
35 001540 113701 000000G 2$:   MOVB     INBSY,R1              ;Get # of job being inswapped
36 001544 004737 001552'          CALL     INSFIN              ;Finished inswap
37 001550 000207          9$:   RETURN

```

```

1          .SBTTL  INSFIN -- Finished inswap
2          ;-----
3          ; INSFIN is called when we have finished doing an inswap for a job.
4          ;
5          ; Inputs:
6          ; R1 = Index number of job that was inswapped.
7          ;
8 001552 010246 INSFIN: MOV      R2, -(SP)
9          ;
10         ; Say job is in memory now
11         ;
12 001554 052761 000000G 000000G      BIS      #$INCOR, LSW(R1) ; Say it is in memory now
13         ;
14         ; Say slot in swap file is now free
15         ;
16 001562 013702 000000G      MOV      SWPJOB, R2      ; Point to table with job #'s in swap file
17 001566 013700 000000G      MOV      VSWPSL, R0      ; Get # slots in swap file
18 001572 020122      1$:  CMP      R1, (R2)+      ; Search for job # in swap file table
19 001574 001402      BEQ      2$              ; Br if found
20 001576 077003      SOB      R0, 1$              ; Loop if more to check
21 001600 000401      BR       3$              ; Job must have been created without swapping
22 001602 005042      2$:  CLR      -(R2)          ; Say swap file slot is free
23         ;
24         ; Set minimum in-core run-time for job.
25         ;
26 001604 013761 000000G 000000G      3$:  MOV      VCORTM, LMINQ(R1); Must run this much before outswap
27 001612 005061 000000G      CLR      LQUAN(R1)      ; Initialize job's time quantum
28         ;
29         ; Request a job scheduler cycle.
30         ;
31 001616 105037 000000G      CLRB     INBSY          ; Say inswap is finished
32 001622 105237 000000G      INCB     DOSCHD        ; Request a job scheduler cycle
33         ;
34         ; Finished
35         ;
36 001626 012602      MOV      (SP)+, R2
37 001630 000207      RETURN

```

SCPGET -- Get a free swap command packet

```

1          .SBTTL  SCPGET -- Get a free swap command packet
2          ;-----
3          ; SCPGET is called to get a free swap command packet.
4          ; A system crash occurs if there are no free packets.
5          ; The packet that is acquired is linked onto the queue of pending
6          ; packets.
7          ;
8          ; Outputs:
9          ;   R5 = Address of free packet.
10         ;
11 001632  SCPGET:
12         ;
13         ; Get 1st packet off of free list.
14         ;
15 001632  013705  000000G      MOV      SCPFHD,R5      ;Get pointer to 1st free packet
16 001636  001422          BEQ      10$      ;Br if there are no free packets
17         ;
18         ; Remove packet from free list
19         ;
20 001640  016537  000000G 000000G      MOV      SP$LNK(R5),SCPFHD ;Remove packet from free list
21         ;
22         ; Add packet to tail of pending list
23         ;
24 001646  013700  000004'      MOV      SCPQTL,R0      ;Point to last entry in queue
25 001652  001403          BEQ      1$      ;Br if queue is empty
26 001654  010560  000000G      MOV      R5,SP$LNK(R0) ;Make current end of tail point to us
27 001660  000402          BR      2$
28 001662  010537  000002'      1$:     MOV      R5,SCPQHD ;We are only entry in list
29 001666  010537  000004'      2$:     MOV      R5,SCPQTL ;We are new end of list
30 001672  005065  000000G      CLR      SP$LNK(R5)   ;Say no packet follows us
31 001676  105237  000000G      INCB    DOSCHD      ;Request a job scheduler cycle
32         ;
33         ; Finished
34         ;
35 001702  000207          RETURN
36         ;
37         ; Error -- There are no free swap command packets
38         ;
39 001704      10$:     DIE      #EM$NSP      ;System crash -- No free swap command packets

```

SCPFRE -- Free a swap command packet

```

1          .SBTTL  SCPFRE -- Free a swap command packet
2          ;-----
3          ; SCPFRE is called to return a swap command packet to the free list.
4          ;
5          ; Inputs:
6          ; R5 = Pointer to packet to be freed.
7          ;
8 001716   SCPFRE:
9          ;
10         ; Remove command packet from pending list
11         ;
12 001716 016537 000000G 000002'      MOV     SP$LNK(R5),SCPQHD ;Make next packet be 1st pending one
13 001724 001002                      BNE     1$              ;Br if there is another pending packet
14 001726 005037 000004'              CLR     SCPQTL          ;No packets in queue
15         ;
16         ; Add packet to free list
17         ;
18 001732 013765 000000G 000000G 1$:  MOV     SCPFHD,SP$LNK(R5) ;Add packet to free list
19 001740 010537 000000G              MOV     R5,SCPFHD
20 001744 000207                      RETURN
21         .END

```

Errors detected: 0

\*\*\* Assembler statistics

Work file reads: 0  
 Work file writes: 0  
 Size of work file: 246 Words ( 1 Pages)  
 Size of core pool: 18176 Words ( 71 Pages)  
 Operating system: RT-11

Elapsed time: 00:00:17.20  
 ,LP:TSSWAP=DK:TSSWAP/C/N:SYM

\$INCOR	1-25	3-18	3-56	7-26	8-24	14-12		
\$MAPOK	1-28	8-28						
\$MLOCK	1-26	3-61	5-21					
\$NDMEM	1-29	7-20	10-24					
\$NLOCK	1-33	5-22						
CS\$EOF	1-29	10-6	13-6					
CS\$ERR	1-31	10-6	13-6					
DIEARG	1-43							
DIEMSG	1-43	5-29*	8-47*	10-8*	13-8*	15-39*		
DOSCHD	1-30	10-37*	14-32*	15-31*				
EM\$LMF	1-31	5-29						
EM\$NSP	1-32	15-39						
EM\$SFO	1-27	8-47						
EM\$SIE	1-26	10-8	13-8					
ENQHD	1-28	10-32						
FORCEX	1-33	6-33						
FREMEM	1-24	10-22						
GETMEM	1-28	3-29	5-27					
GETSYQ	1-25	9-15	12-15					
IASEGS	1-27	13-29						
INBSY	1-23	11-16*	13-20	13-35	14-31*			
INCMP	12-32	13-6#						
INCMP2	11-25	13-13	13-20#					
IN\$FIN	1-18	13-36	14-8#					
IN\$WP	3-35	5-33	11-16#					
INTPRI	1-26	3-28	3-108	3-113				
IDHLM	1-24	3-97	7-42					
ISWPGO	11-36	12-10#	13-14					
LBASE	1-25	8-57	10-20	10-23*	11-32			
LIOCNT	1-29	3-95	7-31					
LIOHLD	1-28	3-97*	7-42*					
L\$MEMIN	1-27	13-21						
L\$MING	1-27	3-66	14-26*					
LNBLKS	1-30	8-55	10-21	11-24	11-30	13-23*		
LNSBLK	1-26	8-35	13-22	13-27				
LQLINK	1-25	3-20	3-102					
LQUAN	1-25	14-27*						
LSTATE	1-31	3-16	3-73	3-78	10-29			
L\$STIN	1-88#	11-17*						
L\$STOPR	1-90#	8-20*	11-18*					
L\$STOUT	1-89#	8-19*						
L\$STSL	1-27	7-16						
LSW	1-27	3-18	3-56	7-20	7-26	8-24*	10-24*	14-12*
LSW6	1-24	3-61	5-21*	5-22*				
LSW7	1-27	8-28*						
LSWPBK	1-29	8-51*	8-56	11-31				
MAPUSR	1-29	8-29	8-31*					
MEMSWP	1-25	2-12	7-59*					
OSWPGO	8-58	9-10#	10-14					
OUTBSY	1-23	8-18*	10-19	10-36*				
OUTCMP	9-33	10-6#						
OUTSWP	1-18	3-109	7-36	8-18#				
OVRHC	1-23	8-37	13-29	13-30				
PSW	1-29	3-13*	3-28*	3-47*	3-108*	3-113*		
Q. BLKN	1-30	9-19*	12-19*					
Q. COMP	1-24	9-33*	12-32*					

Q.PAR	1-30	9-22*	12-22*						
Q.WCNT	1-29	9-32*	12-31*						
RC#BAS	1-32	6-24*	6-28*						
RC#PAG	1-32	6-23							
RUNQHD	1-24	3-14							
RUNQTL	1-30	3-48							
S#\$RT	1-28	3-73							
S#\$RUN	1-28	3-16	3-79						
S#CPU	1-26	10-31							
S#IOWT	1-30	3-81							
S#WFM	1-33	10-29							
SCPFHD	1-28	15-15	15-20*	16-18	16-19*				
SCPFRE	5-37	6-37	16-8#						
SCPGET	1-19	15-11#							
SCPQHD	1-80#	4-17	15-28*	16-12*					
SCPQTL	1-81#	15-24	15-29*	16-14*					
SEGIN	1-27	13-30							
SEQOUT	1-26	8-37							
SP#CMD	1-31	4-27							
SP#DW1	1-32	6-19							
SP#JOB	1-31	5-17	6-32						
SP#LNK	1-32	15-20	15-26*	15-30*	16-12	16-18*			
SWAPER	1-18	2-8#							
SWPBAS	1-84#	8-57*	9-20	9-28*	11-32*	12-20	12-28*		
SWPBLK	1-83#	8-56*	9-19	9-29*	11-31*	12-19	12-29*		
SWPCHN	1-25	9-14	10-6	12-14	13-6				
SWPCOT	1-24	3-89*							
SWPFUN	2-21	4-13#							
SWPJOB	1-29	8-43	8-49	14-16					
SWPLOK	4-44	5-13#							
SWPMEM	2-14	7-10#							
SWPPOS	1-31	8-50							
SWPPRI	2-26	3-7#							
SWPRGN	4-45	6-14#							
SWPSIZ	1-82#	8-55*	9-23	9-27*	10-12	11-30*	12-23	12-27*	13-12
SWPVEC	4-28	4-44#							
SYQID	1-30	9-37	12-36						
SYSHLT	1-43	5-29	8-47	10-8	13-8	15-39			
TRYRGN	1-32	6-25							
TSSWAP	1-6#	1-17							
VCORTM	1-31	14-26							
VSWPFL	1-24								
VSWPSL	1-30	8-42	14-17						

DIE	1-44#	5-29	8-47	10-8	13-8	15-39
DISABL	1-69#	3-13	3-47			
ENABL	1-73#	3-28	3-108	3-113		
OCALL	1-54#	8-37	13-29	13-30		