

Table of contents

5-	1	SCHED	-- Suspend job and look for another
8-	1	SWPCHK	-- See if job swapping is needed
9-	1	GETMEM	-- Try to get free memory for a job
10-	1	TRYMEM	-- Try to allocate memory space for a job
11-	1	FREMEM	-- Free a memory region
12-	1	TRYPLS	-- Determine # memory pages available for PLAS
13-	1	TRYRGN	-- Try to allocate memory for global region
14-	1	SETMAP	-- Set up memory mapping registers for a job
15-	1	LODMAP	-- Load memory mapping from context block
16-	1	MAPSYS	-- Map to the system code regions
17-	1	PKSTAT	-- Pack user status into context block
18-	1	UPSTAT	-- Unpack user status from context block
19-	1	DOCMPL	-- Call job's completion routines
20-	1	SUTOP	-- Set top of memory for a job
21-	1	MEMXPN	-- Do job swap to expand memory size
22-	1	CXBMOV	-- Move job context data into buffer
23-	1	ENQHD	-- Put user at head of queue
24-	1	ENQTL	-- Add user to tail of execution queue
25-	1	DEQ	-- Remove user from run queue
26-	1	QSRCH	-- Look for 1st user with some execution state
27-	1	QNSPND	-- Put job in wait state
28-	2	FORCEX	-- Force user execution
28-	23	CHKABT	-- Check for abort condition
29-	1	UREGO	-- Restart user at head of wait queue
30-	1	QHIPRI	-- Put user in high priority queue
31-	1	QCPU	-- Place job in CPU-bound run queue
32-	1	GTSYMB	-- Get system message buffer
33-	1	TSXTRP	-- Catch traps
34-	1	TRPMAP	-- High-performance memory mapping service
35-	1	UEXINT	-- Unexpected interrupt
37-	1	CLKINT	-- Clock interrupt routine
38-	1	ENSY	-- Enter system state
39-	1	INTEN	-- Interrupt entry processing
42-	1	FORK	-- Queue a fork request
43-	1	FRKGET	-- Get a free Fork block
44-	1	FORKQ	-- Queue a fork request
45-	1	SYNCH	-- Queue a synch request
46-	1	QUNSIG	-- Signal quantum expiration
47-	1	SYSHLT	-- Fatal system halt
48-	1	INIJMP	-- Final system initialization

```

1          .TITLE  TSEXEC -- TSX-Plus Executive Module
2 000000   .CSECT  TSEXEC
3          .ENABL  LC
4          .ENABL  AMA
5          .DSABL  QBL
6 000000   TSEXEC:
7          ;
8          ; TSEXEC is the executive control module of TSX-Plus.
9          ; It contains routines to perform job scheduling, memory management,
10         ; clock interrupt processing, etc.
11         ;
12         ; Copyright (c) 1980, 1981, 1982, 1983, 1984, 1985.
13         ; S&H Computer Systems, Inc.  Nashville, Tn.
14         ;
15         ; All rights reserved.
16         ;
17         ; Written by Phil Sherrod.
18         ;
19         ; Global definitions
20         ;
21         .GLOBL  SCHED, EXEC, STOP, CLKINT, INTLVL, CLKRTI, DIEARG, DIEMSG
22         .GLOBL  DIEPC, DIESP, SYSHL1, TRPAR5, USP, INBSY, OUTBSY, EXCBUF
23         .GLOBL  ENQHD, ENQTL, DEQ, FORCEX, UREGO, TRP250, QSRCH
24         .GLOBL  CHKABT, QHIPRI, FPTRAP, TRP4, TRP10, STKLVL, ABRTOV
25         .GLOBL  DOSCHD, INTEN, SYSXIT, SUTOP, MRKTHD, INIJMP
26         .GLOBL  TSEXEC, FORK, KMNSTR, KMNSTK, KMNPGS
27         .GLOBL  KMNCHN, QNSPND, INTPRI, TRP14, TRP20, TRP24
28         .GLOBL  TRP34, EXCINI, INITFL, QTSYMB, QHDSPN, MBFFLG
29         .GLOBL  QDTTRP, DATIML, DATIMH, UIOCNT, SS, ENSYS
30         .GLOBL  PMUSER, PMFLGS, PMBASE, PMTOP, PMNBPC, INTENX
31         .GLOBL  PMPAR, PMRUN, PMCELS, SYNCH, UEXINT, MEMPAR
32         .GLOBL  SETMAP, JMPO, LOKSWP, MEMSWP, DTLX, SYPNCR, CKUSP2
33         .GLOBL  CHKUSP, EXCJOB, EM$SOF, MAPSYS, SYSMAP, UEXRTN
34         .GLOBL  CLKCNT, TIKCNT, TK5CNT, MINCTR, CLKPS, CLKPC
35         .GLOBL  RUNQHD, RUNQTL, QCPU, QNSPNX, QHDSPX
36         .GLOBL  BRKPT, USRJOB, SPDJOB, GETMEM, TRYPLS, TRYRGN
37         .GLOBL  INTPRI, FRKQGE, RUNQHD, SWPCOT, TRYMEM, MEMXPN
38         .GLOBL  FREMEM, FPUUSE, UMSPSV, CURVC
39         .GLOBL  INTSTK, INTSND, SS, SSEND, QUNSIG
40         .GLOBL  FRKPRI, FRKGET, FORKQ, SWPPOS, SWPJOB, CURFRK
41         .GLOBL  CXTWDS, CXTPAG, CXTPDR, CXTRMN, SLTSIZ
42         .GLOBL  CXTBUF, CXBOWN, CXBJOB, CXBBAS, CXBSIZ, CXBMOV
43         ;
44         ; Global references
45         ;
46         .GLOBL  $GEMAR
47         .GLOBL  CFLAG, MAXSRD, SR$PAR, SR$PDR, SR$PX, DOTRMP
48         .GLOBL  $DILUP, $INIT, $DISCN, LIOCNT, IOHLM, S$SPND
49         .GLOBL  $VNOTT, GETRTQ, $WDISP, WINDSP
50         .GLOBL  $INKMN, $SETCC, NEWUSR, VSWPFL, $RDSAV, LSW11
51         .GLOBL  S$IOFN, S$CPU, $FPUEX, PODB, PRIVCO
52         .GLOBL  EMTCAD, SPCPS, CPLEMT, MAPPAR, $SUSPN
53         .GLOBL  S$$RUN, QUECHR, $DBGBK, $DBGRK, SWAPER
54         .GLOBL  QF$SYN, QF$IOT, CQ$JOB, S$TWFN, QCOMPL
55         .GLOBL  MMENBL, SRMMR, MEM256, EMMAP, IOMAP, SR3MMR
56         .GLOBL  S$INWT, S$TMWT, $DEBUG, BRKENT, $NOUCR
57         .GLOBL  LSW, LSW4, LJSW, LBSPRI, VPRIVR, CURCP

```

```

58 . GLOBL LSW8, $SQIID, $SQHID, CQ$FLG, QF$SCR
59 . GLOBL FREIOQ, LCMPL, SYSHLT, CQ$CP
60 . GLOBL LSTATE, FQ$PRI, FP$MAX, FP$DEF, CP$SYN
61 . GLOBL CQ$LNK, CQ$RTN, CQ$RO, CQ$R1, CQ$PA5
62 . GLOBL SB$TXT, SB$PNT, CCFLG, LSCCA, FREFRK
63 . GLOBL USRJOB, INITGD
64 . GLOBL RPAR, RPDR, D. FLAG, D$DMON
65 . GLOBL ERRLOC, JSWLOC, UFPTRP
66 . GLOBL EM$PFT
67 . GLOBL EM$UEI, EM$MPR, EM$JMO, EM$FRK
68 . GLOBL TSXTX, TRPCOM, SYSDIE, LPRI, VPRIHI, VPRILO
69 . GLOBL $CTRLC, LSW7, $VIRJB, TRPBPT
70 . GLOBL LQLINK, $NDABT, CXTBAS, VPAR5
71 . GLOBL CORUSR, MA$RGN
72 . GLOBL MINTIM, FP$CKT
73 . GLOBL CW$FPU, CONFIG, LIOHLD
74 . GLOBL S$$HIP, UPMODE
75 . GLOBL $INCOR, SYSDIE, FPTRPX
76 . GLOBL LMEMIN, LBASE, SPSAVE, PCCCR2, PROFLG
77 . GLOBL QFREE, $NDMEM, LSTSL, CONFIG
78 . GLOBL UPARO, UPDR0, UPAR1, UPDR1, KPAR6, PSW
79 . GLOBL S$WFM, LNBLKS, LPARBS, $MLOCK, $NLOCK
80 . GLOBL FQ$LNK, FQ$R4, FQ$R5, FQ$RTN, FQ$PA6
81 . GLOBL RMNPDR, UPAR6, UPDR6, UPAR7, UPDR7
82 . GLOBL UMODE, $IOMAP, S$$RT, LCXPAR, FQ$PA5
83 . GLOBL UHIMEM, FREIOQ, FQ$UFB
84 . GLOBL LITIME, FQ$R1, FQ$R2, FQ$R3
85 . GLOBL S$HICP
86 . GLOBL CW$FPU, JSTKND
87 . GLOBL MINTIM
88 . GLOBL VINTIO
89 . GLOBL SNMSHD, SB$LNK, $KINIT
90 . GLOBL NMUMB
91 . GLOBL LQUAN
92 . GLOBL BASMAP, LOMAP, HIMAP, FREPGS, $MAPOK, LSW7
93 . GLOBL S$TMWT, S$INWT
94 . GLOBL CQ$PRI, CQ$RNS
95 . GLOBL MAPUSR
96 . GLOBL CS$ERR, CS$EOF
97 . GLOBL PMSIZE
98 . GLOBL R$UBAS
99 . GLOBL LSW6
100 . GLOBL SN$RTN, SN$JOB, SN$ID
101 . GLOBL OVRHC, OVRADD, D. PAR, KPAR5
102 . GLOBL CLKRUN, IOPAGE
103 . GLOBL LNSBLK
104 . GLOBL LHIPCT, VHIPCT, S$RT, S$LOW
105 . GLOBL CUPARO, CUPAR1, CUPAR6, CUPAR7
106 . GLOBL CUPDRO, CUPDR1, CUPDR6, CUPDR7

```

```

1          ; -----
2          ; Misc. data cells
3          ;
4          001000  SS      =      1000      ; Top of system stack area
5          000600  SEND    =      600      ; Base of system stack
6          ;
7          ; Ascii characters
8          ;
9          000015  CR      =      15      ; Carriage-return
10         000012  LF      =      12      ; Line-feed
11         000007  BELL    =      7       ; Bell
12         ;
13 000000      000  RUNQHD: .BYTE 0      ; Head of execution state list
14 000001      000  RUNQTL: .BYTE 0      ; Tail of execution state list
15 000002      000  USRJOB: .BYTE 0      ; Number of job that owns usr data base
16 000003      000  SPDJOB: .BYTE 0      ; Number of job that owns SPD data base
17 000004      000  DOSCHD: .BYTE 0      ; Non-zero ==> need to do job scheduling
18 000005      000  SWPCOT: .BYTE 0      ; Non-zero ==> Reschedule when CORTIM expires
19 000006      000  EXCJOB: .BYTE 0      ; # of job with exclusive access to system
20 000007      377  INTLVL: .BYTE -1     ; Interrupt level counter
21 000010      377  STKLVL: .BYTE -1     ; .GE. 0 ==> Running on interrupt stack
22 000011      000  FRKPRI: .BYTE 0      ; Current fork priority
23 000012      001  INITFL: .BYTE 1      ; Non-zero ==> system initialization being done
24 000013      000  FPUUSE: .BYTE 0      ; Non-zero ==> FPU unit in use by current job
25 000014      020  PROSKP: .BYTE 16     ; Counter used for PRO-350 clock interrupts
26 000015      000  CXBOWN: .BYTE 0      ; Number of job that owns CXTBUF
27 000016      000  CXBJOB: .BYTE 0      ; Number of job whose cxt blk is in CXTBUF
28 000017      000  INBSY: .BYTE 0      ; Number of job currently being inswapped
29 000020      000  OUTBSY: .BYTE 0      ; Number of job currently being outswapped
30         ;
31         ;
32 000022      000000 INTSTK: .WORD 0      ; Pointer to start of interrupt stack
33 000024      000000 INTSND: .WORD 0      ; Pointer to last word in interrupt stack
34 000026      000000 DTLX: .WORD 0      ; # minutes of uptime for demo system
35 000030      000000 ODTTRP: .WORD 0      ; Address of breakpoint entry into system ODT
36 000032      000000 SYSMAP: .WORD 0      ; number of system segment currently mapped
37 000034      000000 TRPAR5: .WORD 0      ; Kernel PAR5 content when trap occurred
38 000036      000000 DIEMSG: .WORD 0      ; Pointer to system crash message
39 000040      000000 DIEARG: .WORD 0      ; Argument value for system crash
40 000042      000000 DIEPC: .WORD 0      ; Address of call to SYSHLT
41 000044      000000 DIESP: .WORD 0      ; Kernel par 5 mapping at time of crash
42 000046      000000 KMNSTK: .WORD 0      ; Address of Kmon stack
43 000050      000000 KMNSTR: .WORD 0      ; Starting address of Kmon
44 000052      000000 KMNPGS: .WORD 0      ; # 256-word memory pages needed to run TSKMON
45 000054      000000 KMNCHN: .BLKW 5      ; Save status for Kmon file channel
46 000066      000000 MRKTHD: .WORD 0      ; Head of mark-time queue list
47 000070      000000 MEMSWP: .WORD 0      ; # jobs needing memory-expansion outswap
48 000072      000000 LOKSWP: .WORD 0      ; # jobs needing to be locked in memory
49 000074      000000 MBFFLG: .WORD 0      ; Non-zero==>Message buffer was freed
50 000076      177777 CLKCNT: .WORD -1
51 000100      177777 TIKCNT: .WORD -1
52 000102      000000 CLKPC: .WORD 0
53 000104      000000 CLKPS: .WORD 0
54 000106      000000 DATIML: .WORD 0
55 000110      000000 DATIMH: .WORD 0
56 000112      000000 TK5CNT: .WORD 0
57 000114      000000 UIOCNT: .WORD 0      ; # USER I/O OPERATIONS IN PROGRESS

```

```

58 000116 001130      MINCTR: .WORD 600.
59 000120 000000      USP: .WORD 0
60 000122 000340      INTPRI: .WORD 340
61 000124 000000      FRKCQE: .WORD 0 ;FORK queue head
62 000126 000000      SYPNCR: .WORD 0 ;Pending system mark-time compl requests
63 000130 000000      CURVC: .WORD 0 ;Addr of current direct interrupt control blk
64 000132 000000      CURFRK: .WORD 0 ;Address of currently running fork routine
65 000134 000000      SWPPOS: .WORD 0 ;Ptr to table with swap file slot positions
66 000136 000000      SWPJOB: .WORD 0 ;Ptr to table with #'s of jobs in swap slots
67 000140 000000      SLTSIZ: .WORD 0 ;# blocks used by each slot in swap file
68 000142 000000      PMUSER: .WORD 0 ;# of user doing performance monitoring
69 000144 000000      PMFLGS: .WORD 0 ;PF# flags for performance monitor
70 000146 000000      PMBASE: .WORD 0 ;Base address being monitored
71 000150 000000      PMTOP: .WORD 0 ;Top address of region being monitored
72 000152 000000      PMNBPC: .WORD 0 ;Number of bytes per p.m. cell
73 000154 000000      PMPAR: .WORD 0 ;Address of pm vector area
74 000156 000000C     PMCELS: .WORD PMSIZE/2 ;Number of cells in pm vector area
75 000160 000000      PMRUN: .WORD 0 ;Non-zero if performance mon in progress
76 000162 000000      CXTWDS: .WORD 0 ;# words for job context block
77 000164 000000      CXTPAG: .WORD 0 ;# 512-byte pages for job context block
78 000166 000000      CXTPDR: .WORD 0 ;PDR value to map job context block
79 000170 000000      CXTRMN: .WORD 0 ;Address in context area of simulated RMON
80 000172 000000      CXTBUF: .WORD 0 ;Addr of buffer used for accessing cxt blk
81 000174 000000      CXBBAS: .WORD 0 ;Addr of data currently in CXTBUF
82 000176 000000      CXBSIZ: .WORD 0 ;Amt of data currently in CXTBUF
83 000200 000000      RMNPDR: .WORD 0 ;PDR value to map to simulated RMON
84 ;
85 ; End of TSEXEC data area.
86 ; Begin 1024 byte buffer used by TSINIT at this point.
87 ;
88 000202      EXCBUF: ;Base of area used by TSINIT for buffer

```

```

1      ;
2      ; Macro to print an error message when a system crash occurs.
3      ;
4      ; Arguments:
5      ;   MSG = Name of error message to print.
6      ;   ARG = (Optional) argument value to display with error message.
7      ;
8      .MACRO  DIE      MSG,ARG
9      MOV    MSG, @DIEMSG
10     .IF    NB,ARG
11     MOV    ARG, @DIEARG
12     .ENDC
13     CALL  @SYSHLT
14     .ENDM  DIE
15     ;
16     ; Macro definition for call global routines residing in mapped system regions
17     ;
18     .MACRO  OCALL  ENTADD
19     .IF    B, ENTADD
20     .ERROR ; OCALL SPECIFIED WITH NO ENTRY ADDRESS
21     .MEXIT
22     .ENDC
23     CALL  OVRHC      ; CALL THE OVERLAY HANDLER
24     .WORD ENTADD    ; SPECIFY THE ENTRY POINT
25     .ENDM
26     ;
27     ; -----
28     ; MACROS TO ENABLE AND DISABLE INTERRUPTS.
29     ; 'DISABL' RAISES THE CPU PRIORITY LEVEL TO 7.
30     ; 'ENABL' LOWERS THE PRIORITY TO THE CURRENT OPERATING
31     ; PRIORITY WHICH IS STORED IN INTPRI.
32     ;
33     .MACRO  DISABL      ; DISABLE INTERRUPTS
34     BIS    #340, @PSW
35     .ENDM  DISABL
36     ;
37     .MACRO  ENABL      ; RESTORE INTERRUPT STATUS
38     BIC    INTPRI, @PSW
39     .ENDM  ENABL

```

```
1 ;  
2 ; Starting point of execution of TSX-Plus.  
3 ;  
4 ; Enter initialization module.  
5 ;  
6 000202 000137 0000000 START: JMP INITGD  
7 ;  
8 ; Call from TSTTY for ^R to allow breakpoint capture for debugging.  
9 ;  
10 000206 000207 BRKPT: RETURN ;return to TSTTY for echo processing
```

```

1          .SBTTL  SCHED  -- Suspend job and look for another
2          ;-----
3          ; The SCHED routine is called to suspend the execution of the
4          ; current user and to look for another user to execute.
5          ; SCHED selects the highest priority user who is ready to run
6          ; (or waits for a user to get ready to run), gets that user into
7          ; main memory, and then returns control over to the user.
8          ; On return from SCHED, the user is ready to run and the execution
9          ; time quantum has been set up.
10         ; When called, the user's stack must be in SP.
11         ; All registers are preserved.
12         ;
13         ; Save all of the user's registers on his stack
14         ;
15 000210 010046 SCHED:  MOV     R0,-(SP)
16 000212 010146      MOV     R1,-(SP)
17 000214 010246      MOV     R2,-(SP)
18 000216 010346      MOV     R3,-(SP)
19 000220 010446      MOV     R4,-(SP)
20 000222 010546      MOV     R5,-(SP)
21         ;
22         ; Check for user stack overflow.
23         ;
24 000224 020627 0000009      CMP     SP,#JSTKND      ;check the stack limit
25 000230 101010      BHI     1$              ;br if stack is ok
26 000232      DIE     #EM$SOF,#3      ;stack overflow
27         ;
28         ; Say user is no longer executing.
29         ;
30 000252 010603 1$:      MOV     SP,R3          ;Save user's stack pointer
31 000254 012706 001000      MOV     #SS,SP        ;Switch to system stack
32 000260 004737 002416'      CALL    PKSTAT        ;Pack status into job context block
33 000264 010337 0000000      MOV     R3,SPSAVE     ;Save user's stack pointer in his context area
34 000270 105037 0000000      CLRB   CORUSR        ;User is no longer running

```

SCHED -- Suspend job and look for another

```

1      ;
2      ; Find a job to run.
3      ; The EXEC module is entered from SCHED to find the next user to run.
4      ; EXEC locates the highest priority user and if that user is
5      ; ready to run starts execution of the user.
6      ; EXEC is also called at the end of TSX initialization to begin
7      ; operation of the system.
8      ;
9      ; If current job is a real-time job that has acquired exclusive
10     ; access to the system, continue to run it and bypass the normal
11     ; swapping and scheduling process.
12     ;
13     000274 113701 000006'      MOVB   EXCJOB,R1      ;Is there an exclusive real-time job?
14     000300 001031              BNE    GTRDY         ;If yes then go try to run it
15     ;
16     ; Call the swapper and see if it has anything to do.
17     ;
18     000302 004737 000626'      EXEC:  CALL   SWPCHK      ;Give swapper a chance to run
19     ;
20     ; Now search down the job run queue from highest priority to lowest
21     ; priority looking for a job that is in memory and wants to run.
22     ;
23     000306              DISABL          ;;;Disable interrupts
24     000314 105037 000004'      CLRB   DOSCHD        ;;;Say a scheduler cycle has been done
25     000320 113701 000000'      MOVB   RUNQHD,R1     ;;;Point to 1st job in run queue
26     000324 001413              BEQ    2$             ;;;Br if no jobs (system must be idle)
27     000326 026127 000000G 000000G 1$:  CMP    LSTATE(R1),#S$RUN ;;;Does this job want to run?
28     000334 101007              BHI    2$             ;;;Br if there are no jobs that want to run
29     000336 032761 000000G 000000G      BIT    #$INCOR,LSW(R1) ;;;Is this job in memory now?
30     000344 001007              BNE    GTRDY         ;;;Br if yes
31     000346 116101 000000G      MOVB   LQLINK(R1),R1  ;;;Try next job in list
32     000352 001365              BNE    1$             ;;;Br if there is another to test
33     ;
34     ; There are no in-core jobs that want to run.
35     ; Loop in scheduler until one becomes available.
36     ;
37     000354 2$:  ENABL          ;Enable interrupts
38     000362 000747              BR     EXEC          ;Keep looking for a job to run

```

SCHED -- Suspend job and look for another

```

1
2 ; We found a job to run.
3 ; The job index number is in R1.
4 ;
5 000364 110137 0000000 GTRDY: MOVB R1,CORUSR ;;;Say this job is running
6 000370 ENABL ;Enable interrupts
7 ;
8 ; See if we are reactivating an old job or starting a new job.
9 ;
10 000376 032761 0000000 0000000 BIT ##INIT,LSW(R1) ;Is this an old or new job?
11 000404 001002 BNE 2$ ;Br if old job
12 000406 000137 0000000 JMP NEWUSR ;Go initialize a new job
13 ;
14 ; We are reactivating an old job.
15 ; If the execution of this job has been suspended, put job back to sleep.
16 ;
17 000412 032761 0000000 0000000 2$: BIT ##SUSPN,LSW(R1) ;Has job's execution been suspended?
18 000420 001415 BEQ 6$ ;Br if not
19 000422 105761 0000000 TSTB LIOCNT(R1) ;Does job have any I/O active now?
20 000426 001404 BEQ 7$ ;Br if not
21 000430 012761 0000000 0000000 MOV #IOHLM,LIOHLD(R1) ;Hold I/O for this job
22 000436 000406 BR 6$ ;But let it run till I/O stops
23 000440 012700 0000000 7$: MOV #S$SPND,R0 ;Put job in suspended state
24 000444 004737 004424' CALL ENQTL ;Put job in suspended state
25 000450 000137 000302' JMP EXEC ;Go back to scheduler to find another job
26 ;
27 ; Set up kernel-mode PAR6 to point to job's context block
28 ;
29 000454 016137 0000000 0000000 6$: MOV LCXPAR(R1),@#KPAR6 ;Map kpar6 to job's context block
30 ;
31 ; Set up memory mapping registers for this job.
32 ; Check to see if the memory mapping registers are already loaded
33 ; for the job we want to run.
34 ;
35 000462 120137 0000000 CMPB R1,MAPUSR ;Is memory mapping set for job we want to run?
36 000466 001411 BEQ 5$ ;Br if yes
37 ;
38 ; Memory mapping is not set up correctly for the job we want to run.
39 ; Determine if mapping information in job's context block is set up
40 ; correctly. If so, just load it into PAR registers, otherwise
41 ; call SETMAP to compute memory mapping information.
42 ;
43 000470 032761 0000000 0000000 BIT ##MAPOK,LSW7(R1);Is memory mapping info in context block ok?
44 000476 001003 BNE 3$ ;Br if yes
45 000500 004737 001534' CALL SETMAP ;Set up memory mapping registers for the job
46 000504 000402 BR 5$
47 000506 004737 002276' 3$: CALL LODMAP ;Load par registers from context block info
48 ;
49 ; See if double Ctrl-C occurred while we were asleep and user
50 ; did a .SCCA.
51 ; If so, set status flag in SCCA status word.
52 ;
53 000512 032761 0000000 0000000 5$: BIT ##SETCC,LSW4(R1);Do we need to set control-c status flag?
54 000520 001415 BEQ 1$ ;Br if not
55 000522 016100 0000000 MOV LSCCA(R1),R0 ;Does user want control-c trap control?
56 000526 001407 BEQ 4$ ;Br if not
57 000530 052737 0000000 0000000 BIS #UPMODE,@#PSW ;Set user-previous-mode in PS

```

```

58 000536 106510          MFPD  @RO          ;Get current contents of user's flag cell
59 000540 052716 0000000  BIS   #CCFLG,(SP)    ;Set ctrl-c flag
60 000544 106610          MTPD  @RO          ;Store into cell in user's program space
61 000546 042761 0000000 0000000 4#:  BIC   ##SETCC,LSW4(R1);Say it has been done
62
63                      ;  Unpack job status information from context block.
64
65 000554 013703 0000000 1#:  MOV   SPSAVE,R3    ;Get pointer to job context area stack
66 000560 004737 002562'  CALL  UPSTAT        ;Unpack status for job
67
68                      ;  Switch to job's internal stack.
69
70 000564 010306          MOV   R3,SP         ;Run on stack in context block
71
72                      ;  See if we need to call any completion routines for this job.
73
74 000566 004737 002732'  CALL  DDCMPL        ;Run any pending completion routines
75
76                      ;  See if we need to redisplay display window for job
77
78 000572 032761 0000000 0000000  BIT   ##WDISP,LSW6(R1);Do we need to redisplay window?
79 000600 001403          BEQ   9#           ;Br if not
80 000602          DCALL  WINDSP        ;Redisplay window for job
81
82                      ;  Restore user's registers
83
84 000610 012605          9#:  MOV   (SP)+,R5
85 000612 012604          MOV   (SP)+,R4
86 000614 012603          MOV   (SP)+,R3
87 000616 012602          MOV   (SP)+,R2
88 000620 012601          MOV   (SP)+,R1
89 000622 012600          MOV   (SP)+,R0
90
91                      ;  Return to life!
92
93 000624 000207          RETURN          ;Return from SCHED

```

SWPCHK -- See if job swapping is needed

```

1          .SBTTL  SWPCHK -- See if job swapping is needed
2
3          ;-----
4          ; SWPCHK is called to see if jobs should be swapped into or out of memory.
5 000626 105737 000000G SWPCHK: TSTB  VSWPFL          ;Is this a non-swapping system?
6 000632 001407          BEQ      9$          ;Br if non-swapping system
7
8          ; Return quickly if the swapper is currently busy
9
10 000634 113700 000020'      MOVB  OUTBSY,R0      ;Outswap busy flag
11 000640 153700 000017'      BISB  INBSY,R0      ;Inswap busy flag
12 000644 001002          BNE  9$          ;Br if swapper is busy now
13
14          ; The swapper is not currently busy.
15          ; Call the swapper to see if it needs to do any swapping.
16
17 000646 004737 000000G      CALL  SWAPER          ;Try to do any job swapping
18
19          ; Finished
20
21 000652 000207          9$:  RETURN

```

```

1          .SBTTL  GETMEM -- Try to get free memory for a job
2          ;-----
3          ; GETMEM is called to attempt to obtain a memory region for a job.
4          ; If the memory space is available it is claimed for the job.
5          ;
6          ; Inputs:
7          ; R1 = Job index number
8          ; LMEMIN(R1) = Number of pages of memory needed for job
9          ;
10         ; Outputs:
11         ; MEMMAP updated to show memory space allocated to job.
12         ; LBASE(R1) = Base page number allocated to job.
13         ; LPARBS(R1) = PAR relocation value for base of job (above context block)
14         ; LCXPAR(R1) = PAR relocation value for job context block.
15         ; C-flag set on return if not enough free space was available.
16         ;
17 000654 010246 GETMEM: MOV      R2,-(SP)
18         ;
19         ; See if the total number of free pages is adequate for the request
20         ;
21 000656 016100 000000G      MOV      LMEMIN(R1),R0      ;Get total number of pages needed for job
22 000662 020037 000000G      CMP      R0,FREPGS      ;Are there enough total free pages?
23 000666 101402              BLOS    2$              ;Br if there are enough total free pages
24 000670 000261              SEC                      ;Signal failure on return
25 000672 000421              BR      9$
26         ;
27         ; There are enough total free pages.
28         ; Now see if there are enough contiguous free pages.
29         ;
30 000674 004737 000742' 2$:   CALL    TRYMEM      ;Try to find contiguous space for job
31 000700 103416          BCS    9$              ;Br if cannot find free space for job
32         ;
33         ; We got enough free space for the job.
34         ; Set up information about memory for the job.
35         ;
36 000702 010261 000000G      MOV      R2,LBASE(R1)      ;Base 512-byte page number assigned to job
37 000706 072227 0000003      ASH     #3,R2          ;Convert to 64-byte page number
38 000712 010261 000000G      MOV      R2,LCXPAR(R1)    ;This is value for PAR to map to job context
39 000716 013700 000164'      MOV      CXPAG,R0        ;# 512-byte pages used by job context block
40 000722 072027 0000003      ASH     #3,R0          ;Convert to # 64-byte pages
41 000726 060002              ADD     R0,R2          ;Add # pages used by job context block
42 000730 010261 000000G      MOV      R2,LPARBS(R1)   ;This is value for 1st PAR register for job
43 000734 000241              CLC                      ;Signal success on return
44         ;
45         ; Finished
46         ;
47 000736 012602 9$:   MOV      (SP)+,R2
48 000740 000207          RETURN

```

TRYMEM -- Try to allocate memory space for a job

```

1          .SBTTL  TRYMEM -- Try to allocate memory space for a job
2          ;-----
3          ; TRYMEM is called to attempt to locate a free memory region of
4          ; a specified size.  If the region is found, it is claimed for the job.
5          ;
6          ; Inputs:
7          ; R0 = Number of 512-byte pages wanted.
8          ; R1 = Job index number.
9          ;
10         ; Outputs:
11         ; C-flag cleared ==> Successfully got the region.
12         ; C-flag set      ==> Could not locate a large enough region.
13         ; R0 = Largest free region available.
14         ; R2 = Base page number of region gotten (only if C-flag cleared).
15         ;
16 000742 010346 TRYMEM: MOV     R3, -(SP)
17 000744 010446      MOV     R4, -(SP)
18 000746 010546      MOV     R5, -(SP)
19 000750 013746 000000G  MOV     @#KPAR5, -(SP) ; Save system PAR 5 mapping
20 000754 010046      MOV     R0, -(SP) ; Save request size on top of stack
21         ;
22         ; Set up pointers to memory map table
23         ;
24 000756 013737 000000G 000000G  MOV     MAPPAR, @#KPAR5 ; Map PAR 5 to the mem allocation table
25 000764 013704 000000G      MOV     LOMAP, R4 ; Pointer to 1st user available entry in MEMMAP
26 000770 005000      CLR     R0 ; Save largest free region in R0
27         ;
28         ; Search for the start of a free region
29         ;
30 000772 013703 000000G 8$:     MOV     HIMAP, R3 ; Pointer past last user page entry in MEMMAP
31 000776 020403      CMP     R4, R3 ; Are we beyond end of last possible page?
32 001000 103036      BHS    10$ ; Br if beyond end of user memory region
33 001002 160403      SUB     R4, R3 ; Calc # pages remaining to be checked
34 001004 105724 1$:     TSTB   (R4)+ ; Is this page free?
35 001006 001402      BEQ    2$ ; Br if the page is free
36 001010 077303      SOB    R3, 1$ ; Loop if more pages remain to be checked
37 001012 000431      BR     10$ ; Br if no free region
38         ;
39         ; Found a free region -- Determine how large it is.
40         ;
41 001014 011605 2$:     MOV     (SP), R5 ; Get requested number of pages
42 001016 005304      DEC     R4 ; Point to 1st entry in page table
43 001020 010402      MOV     R4, R2 ; Remember pointer to 1st page entry
44 001022 105724 4$:     TSTB   (R4)+ ; Is this page free?
45 001024 001003      BNE    3$ ; Br if not
46 001026 005305      DEC     R5 ; Does this satisfy the request?
47 001030 001407      BEQ    5$ ; Br if yes
48 001032 077305      SOB    R3, 4$ ; Loop if more pages left to check
49         ;
50         ; This region is not large enough.
51         ; Remember size of largest free region seen.
52         ;
53 001034 005402 3$:     NEG     R2 ; Calculate size of this free region
54 001036 060402      ADD     R4, R2
55 001040 020200      CMP     R2, R0 ; Is this the largest free region seen so far?
56 001042 101753      BLOS   8$ ; Br if not
57 001044 010200      MOV     R2, R0 ; Remember size of largest free region

```

```
58 001046 000751          BR      B#          ;Continue searching
59                          ;
60                          ; We found a free region of adequate size.
61                          ; Claim the region for this job.
62                          ;
63 001050 010203          5#:      MOV      R2,R3          ;Get base page number of region
64 001052 011605          MOV      (SP),R5        ;Get request size
65 001054 010500          MOV      R5,R0          ;Return request size in R0
66 001056 160537 0000000 SUB      R5,FREPGS      ;Decrease total number of free pages
67 001062 110123          7#:      MOVB   R1,(R3)+        ;Mark pages as in use by this job
68 001064 077502          SOB      R5,7#          ;
69 001066 163702 0000000 SUB      BASMAP,R2      ;Get page number of start of region
70 001072 000241          CLC                      ;Signal success on return
71 001074 000401          BR      12#          ;
72                          ;
73                          ; Could not find a large enough region
74                          ;
75 001076 000261          10#:     SEC                      ;Signal failure on return
76                          ;
77                          ; Finished
78                          ;
79 001100 012605          12#:     MOV      (SP)+,R5      ;Pop request size (don't change c-flag)
80 001102 012637 0000000 MOV      (SP)+,@#KPAR5  ;Restore system PAR mapping
81 001106 012605          MOV      (SP)+,R5
82 001110 012604          MOV      (SP)+,R4
83 001112 012603          MOV      (SP)+,R3
84 001114 000207          RETURN
```

FREMEM -- Free a memory region

```

1          .SBTTL  FREMEM -- Free a memory region
2          ;-----
3          ; FREMEM is called to free an area of memory.
4          ;
5          ; Inputs:
6          ; R2 = Base 512-byte page # of start of region.
7          ; R0 = Number of 512-byte pages in region being freed.
8          ;
9          ; Outputs:
10         ; FREPGS is increased by number of pages being freed.
11         ; MEMMAP is altered to show that the pages are free.
12         ;
13 001116 010246 FREMEM: MOV     R2,-(SP)
14 001120 013746 000000G      MOV     @#KPAR5,-(SP) ; Save system PAR 5 mapping
15 001124 010046              MOV     R0,-(SP)
16 001126 001411              BEQ     9$ ; Br if no pages being freed
17         ;
18         ; Map PAR 5 to the memory allocation table
19         ;
20 001130 013737 000000G 000000G      MOV     MAPPAR,@#KPAR5 ; Map PAR 5 to the memory alloc table
21         ;
22         ; Indicate that we have more free memory
23         ;
24 001136 060037 000000G      ADD     R0,FREPGS ; Increase # free pages
25         ;
26         ; Mark pages as free in MEMMAP
27         ;
28 001142 063702 000000G      ADD     BASMAP,R2 ; Point to entry for first page
29 001146 105022 1$: CLRB   (R2)+ ; Mark pages in region as free
30 001150 077002              SOB    R0,1$
31         ;
32         ; Finished
33         ;
34 001152 012600 9$:  MOV     (SP)+,R0
35 001154 012637 000000G      MOV     (SP)+,@#KPAR5 ; Restore system PAR 5 mapping
36 001160 012602              MOV     (SP)+,R2
37 001162 000207              RETURN

```

```

1          .SBTTL  TRYPLS -- Determine # memory pages available for PLAS
2          ;-----
3          ; TRYPLS is called to determine the total number of user memory pages
4          ; which could be used for a PLAS region.
5          ; This routine is placed in Exec because we must use PAR 5 to access
6          ; the memory allocation table.
7          ;
8          ; Inputs:
9          ; R1 = Current job index number.
10         ;
11         ; Outputs:
12         ; R0 = Number of 512-byte pages available for a PLAS region.
13         ;
14 001164 010246 TRYPLS: MOV     R2,-(SP)
15 001166 013746 000000G      MOV     @#KPAR5,-(SP) ;Save system PAR 5 mapping
16         ;
17         ; Map PAR 5 to the memory allocation table
18         ;
19 001172 013737 000000G 000000G      MOV     MAPPAR,@#KPAR5 ;Map PAR 5 to the memory alloc table
20         ;
21         ; Count number of pages available for a PLAS region
22         ;
23 001200 013702 000000G      MOV     LOMAP,R2 ;Point to base of memory map
24 001204 005000      CLR     R0 ;Count pages in R0
25 001206 105722      15$: TSTB   (R2)+ ;Is this page available for user jobs?
26 001210 002401      BLT    16$ ;Br if not
27 001212 005200      INC    R0 ;Count number of user pages
28 001214 020237 000000G      16$: CMP    R2,HIMAP ;Checked all pages?
29 001220 103772      BLO    15$ ;Br if not
30 001222 166100 000000G      SUB    LNBLKS(R1),R0 ;Subtract space already allocated to this job
31 001226 166100 000000G      SUB    LNSBLK(R1),R0 ;Including other plas regions
32 001232 012702 000000G      MOV    #LSTSL,R2 ;Get index number of last job
33 001236 020201      7$:  CMP    R2,R1 ;Is this our job?
34 001240 001410      BEQ    8$ ;Don't count our job again
35 001242 032762 000000C 000000G      BIT    #MLOCK!$NLOCK,LSW6(R2) ;Is this job locked in memory?
36 001250 001404      BEQ    8$ ;Br if not
37 001252 166200 000000G      SUB    LNBLKS(R2),R0 ;If locked, we can't use its space
38 001256 166200 000000G      SUB    LNSBLK(R2),R0
39 001262 162702 0000002      8$:  SUB    #2,R2 ;More jobs to check?
40 001266 001363      BNE    7$ ;Br if yes
41         ;
42         ; Finished
43         ;
44 001270 012637 000000G      MOV    (SP)+,@#KPAR5 ;Restore system PAR 5 mapping
45 001274 012602      MOV    (SP)+,R2
46 001276 000207      RETURN

```

TRYRGN -- Try to allocate memory for global region

```

1          .SBTTL  TRYRGN -- Try to allocate memory for global region
2          ;-----
3          ; TRYRGN is called to attempt to locate a free memory area for
4          ; a shared global (PLAS) region.  The allocation is made in
5          ; the highest available section of meory.
6          ; If a free area of adequate size is found, it is claimed for the
7          ; region.  If it is necessary to swap jobs to free up a large
8          ; enough area, the base of the area to be gotten is returned.
9          ;
10         ; Inputs:
11         ; RO = Number of 512-byte pages needed for region.
12         ;
13         ; Outputs:
14         ; C-flag cleared ==> Free area found and allocated for region.
15         ; R2 = Base page # of region gotten.
16         ; C-flag set ==> Could not find free area.
17         ; R2 = 0 ==> Impossible to collect that much free space.
18         ; RO = Largest available area.
19         ; R2 non-zero ==> Swapping needed, R2=base page # of region.
20         ;
21 001300 010146 TRYRGN: MOV     R1,-(SP)
22 001302 010346      MOV     R3,-(SP)
23 001304 010446      MOV     R4,-(SP)
24 001306 010546      MOV     R5,-(SP)
25 001310 013746 0000000 MOV     @#KPAR5,-(SP)
26 001314 010046      MOV     RO,-(SP)      ; Save requested memory size on top of stack
27         ;
28         ; Set up pointers to memory map table
29         ;
30 001316 013737 0000000 0000000 MOV     MAPPAR,@#KPAR5 ; Map PAR 5 to memory allocation table
31 001324 013704 0000000      MOV     HIMAP,R4      ; Pointer past last entry in memory map
32 001330 005000      CLR     RO      ; Save largest area found in RO
33         ;
34         ; Search for the start of a free region
35         ;
36 001332 010403 B$:     MOV     R4,R3      ; Get pointer above top of next area to check
37 001334 163703 0000000      SUB     LOMAP,R3      ; Get # pages left to check
38 001340 003463      BLE     10$      ; Br if we have checked all pages
39 001342 114401 1$:     MOVB  -(R4),R1      ; Is this page in use?
40 001344 001407      BEQ     2$      ; Br if page is free
41 001346 002404      BLT     6$      ; Br if page not available to user jobs
42 001350 032761 0000000 0000000 BIT     #MLOCK,LSW6(R1); Page belong to job that's locked in memory?
43 001356 001402      BEQ     2$      ; Br if not
44 001360 077310 6$:     SOB     R3,1$      ; Continue searching for start of free area
45 001362 000452      BR      10$      ; Br if no free area
46         ;
47         ; Found the start of an available area.
48         ; Determine how large it is.
49         ;
50 001364 011605 2$:     MOV     (SP),R5      ; Get requested number of pages
51 001366 010402      MOV     R4,R2      ; Save pointer to highest avail page in area
52 001370 005305      DEC     R5      ; Do we only need 1 page?
53 001372 001424      BEQ     5$      ; Br if yes
54 001374 020437 0000000 4$:     CMP     R4,LOMAP      ; Are we down to 1st page?
55 001400 101413      BLOS   13$      ; Br if yes
56 001402 114401      MOVB  -(R4),R1      ; Is this page in use?
57 001404 001405      BEQ     14$      ; Br if not

```

```

58 001406 002411          BLT      3$          ;Br if page not available to user jobs
59 001410 032761 000000G 000000G    BIT      ##MLOCK,LSW6(R1);Page belong to job that's locked in memory?
60 001416 001005          BNE      3$          ;Br if yes
61 001420 005305          14$:    DEC      R5          ;Does this satisfy the request?
62 001422 001410          BEQ      5$          ;Br if yes
63 001424 077315          SOB      R3,4$        ;Loop if more pages need to be checked
64 001426 000401          BR       3$          ;This area is not large enough
65
66          ; This region is not large enough.
67          ; Remember size of largest free region seen.
68
69 001430 005202          13$:    INC      R2
70 001432 160402          3$:    SUB      R4,R2        ;Calc # pages in area
71 001434 020200          CMP      R2,R0        ;Is this the largest free area seen so far?
72 001436 101735          BLOS    8$          ;Br if not
73 001440 010200          MOV      R2,R0        ;Remember size of largest free area
74 001442 000733          BR       8$          ;Continue searching
75
76          ; We found an available area of adequate size.
77          ; See if the area is completely free or if job swapping will be
78          ; required to free it.
79
80 001444 011605          5$:    MOV      (SP),R5    ;Get # pages needed for region
81 001446 010500          MOV      R5,R0        ;Return request size in R0
82 001450 010402          MOV      R4,R2        ;Get pointer to lowest entry in free area
83 001452 163702 000000G    SUB      BASMAP,R2    ;Convert to page number
84 001456 010403          MOV      R4,R3        ;Get pointer to 1st page in area
85 001460 105723          16$:    TSTB    (R3)+    ;Are all pages free now?
86 001462 001013          BNE     15$          ;Br if not -- Swapping will be required
87 001464 077503          SOB     R5,16$      ;Test all pages in region
88
89          ; The area is free. Claim it for the region.
90
91          MOV      R4,R3        ;Get pointer to base of area
92 001470 011605          MOV      (SP),R5    ;Get # pages in region
93 001472 160537 000000G    SUB      R5,FREPGS    ;Decrease total number of free pages
94 001476 112723 000000G    17$:    MOVB    #MA$RGN,(R3)+ ;Say area used for shared region
95 001502 077503          SOB     R5,17$      ;Claim entire area
96 001504 000241          CLC
97 001506 000402          BR      12$        ;Signal success on return
98
99          ; Could not find a large enough area.
100         ; Return in R0 the size of the largest area found.
101
102 001510 005002          10$:    CLR      R2          ;Say could not find an area
103 001512 000261          15$:    SEC
104
105         ; Finished
106
107 001514 012605          12$:    MOV      (SP)+,R5    ;Pop request size (don't alter C-flag)
108 001516 012637 000000G    MOV      (SP)+,@#KPAR5 ;Restore PAR 5 mapping
109 001522 012605          MOV      (SP)+,R5
110 001524 012604          MOV      (SP)+,R4
111 001526 012603          MOV      (SP)+,R3
112 001530 012601          MOV      (SP)+,R1
113 001532 000207          RETURN

```

SETMAP -- Set up memory mapping registers for a job

```

1          .SBTTL  SETMAP -- Set up memory mapping registers for a job
2          ;-----
3          ; SETMAP is called to load the memory mapping registers for a job.
4          ; The user mode registers are set to point to the user program space.
5          ;
6          ; Inputs
7          ; R1 = Job index number.
8          ; LNBLKS = # 256-word blocks assigned to job.
9          ; LPARBS = Physical address (32-word block #) of job's virtual address 0.
10         ;
11         ; Outputs:
12         ; User-mode PAR and PDR registers are loaded.
13         ; Mapping information is stored in CUPAR and CUPDR cells in job's
14         ; context block.
15         ;
16 001534 010246 SETMAP: MOV      R2,-(SP)
17 001536 010346      MOV      R3,-(SP)
18 001540 010446      MOV      R4,-(SP)
19 001542 010546      MOV      R5,-(SP)
20         ;
21         ; See if we are setting up mapping for Kmon or a regular user job.
22         ;
23 001544 032761 000000G 000000G      BIT      #$INKMN,LSW4(R1); Is Kmon running?
24 001552 001433      BEQ      7$          ;Br if not
25         ;
26         ; Set up mapping registers for Kmon as follows:
27         ; 000000-037777      --> 000000-037777 (Map over TSGEN)
28         ; 040000-(top of Kmon) --> Job program space
29         ; 140000-157777      --> Job context area
30         ; 160000-177777      --> Simulated Monitor vector table
31         ;
32         ; Map virtual page 0 (000000-017777) to physical TSGEN; allow write access.
33         ;
34 001554 005037 000000G      CLR      @#UPARO      ;Map virtual page 0 to physical page 0
35 001560 005037 000000G      CLR      CUPARO      ;Save info in context block
36 001564 012737 077406 000000G  MOV      #77406,@#UPDRO ;4kw window, allow read & write access
37 001572 012737 077406 000000G  MOV      #77406,CUPDRO ;Save info in context block
38 001600 012737 000200 000000G  MOV      #200,@#UPAR1 ;Map virtual page 1 to physical page 1
39 001606 012737 000200 000000G  MOV      #200,CUPAR1
40 001614 012737 077406 000000G  MOV      #77406,@#UPDR1 ;4kw window, allow read & write access
41 001622 012737 077406 000000G  MOV      #77406,CUPDR1
42         ; Now enter code to map virtual pages starting with # 1 to job program area.
43 001630 012704 000004      MOV      #4,R4          ;Start mapping with page # 2
44 001634 012705 000016      MOV      #14,R5         ;End mapping with page # 7
45 001640 000411      BR      6$
46         ;
47         ; We are mapping an ordinary user job.
48         ; Set up mapping as follows:
49         ; 000000-(top of program) --> Job program space
50         ; 160000-177777      --> Monitor vector table
51         ;
52         ; Set up user mode mapping registers.
53         ;
54 001642 005004 7$: CLR      R4          ;Start mapping with page # 0
55 001644 012705 000016      MOV      #14,R5         ;Assume we should map through page 7
56 001650 032761 000000G 000000G  BIT      #$IDMAP,LSW6(R1); Does job want page 7 mapped to I/O page?
57 001656 001402      BEQ      6$          ;Br if not

```

```

58 001660 012705 000014      MOV      #12.,R5      ;If yes, only set up mapping through page 6
59 001664 016102 000000G    6$:      MOV      LNBLKS(R1),R2 ;Get # 256-word blocks assigned to job
60 001670 163702 000164'    SUB      CXTPAG,R2    ;Subtract # blocks used by context area
61 001674 072227 000003      ASH      #3,R2        ;Cvt to # 32-word blocks for job
62 001700 016103 000000G      MOV      LPARBS(R1),R3 ;Get base block # of job area
63 001704 010200      2$:      MOV      R2,R0        ;Get # 32-word blocks left to be assigned
64 001706 001432      BEQ      8$          ;Br if finished
65 001710 020027 000200      CMP      R0,#128.    ;Max of 128 blocks per page
66 001714 101402      BLOS     1$          ;Br if ok
67 001716 012700 000200      MOV      #128.,R0    ;Assign 128 32-word blocks to this page
68 001722 160002      1$:      SUB      R0,R2        ;Get # blocks left after this page
69 001724 005300      DEC      R0          ;Get # blocks - 1
70 001726 000300      SWAB    R0          ;Put # blocks in left byte
71 001730 052700 000006      BIS      #6,R0       ;Allow read & write access to the page
72 001734 042700 100261      BIC      #100261,R0  ;Make sure unused PDR bits are zero
73 001740 010064 000000G      MOV      R0,UPDR0(R4) ;Set PDR for page
74 001744 010064 000000G      MOV      R0,CUPDR0(R4) ;Save info in context block
75 001750 010364 000000G      MOV      R3,UPAR0(R4) ;Set PAR for page
76 001754 010364 000000G      MOV      R3,CUPAR0(R4) ;Save info in context block
77 001760 062703 000200      ADD      #128.,R3    ;Advance page base block number
78 001764 062704 000002      ADD      #2,R4       ;Advance register pointer
79 001770 020405      CMP      R4,R5       ;Check for last PAR register mapped
80 001772 003744      BLE     2$          ;Go do next page
81      ;
82      ; Finished mapping all pages in user program space.
83      ; Disallow access to other pages.
84      ;
85 001774 020405      8$:      CMP      R4,R5       ;Have we done all pages?
86 001776 101007      BHI     5$          ;Br if yes
87 002000 005064 000000G      CLR     UPDR0(R4)    ;Disallow all access to the page
88 002004 005064 000000G      CLR     CUPDR0(R4)  ;Save info in context block
89 002010 062704 000002      ADD     #2,R4        ;Move on to next page
90 002014 000767      BR      8$          ;
91      ;
92      ; See if we should map user par7 to simulated Rmon or to I/O page.
93      ;
94 002016 032761 000000G 000000G 5$:      BIT     ##IOMAP,LSW6(R1); Does job want to access I/O page?
95 002024 001415      BEQ     11$         ;Br if not
96      ;
97      ; Map user par7 to I/O page.
98      ;
99 002026 012737 000000G 000000G      MOV     #IOPAGE,@#UPAR7 ;Set address for PAR7
100 002034 012737 000000G 000000G      MOV     #IOPAGE,CUPAR7 ;Set info in context block
101 002042 012737 077406 000000G      MOV     #77406,@#UPDR7 ;Set read/write access and full page size
102 002050 012737 077406 000000G      MOV     #77406,CUPDR7
103 002056 000426      BR      4$          ;
104      ;
105      ; Map user page 7 (160000-177777) to simulated monitor vector table.
106      ;
107 002060 032761 000000G 000000G 11$:     BIT     ##VIRJB,LSW9(R1); Is this a virtual job?
108 002066 001022      BNE     4$          ;If yes then don't alter user's PAR 7
109 002070 013702 000170'      MOV     CXTRMN,R2    ;Get virtual address of RMON in context blk
110 002074 162702 000000G      SUB     #CXTBAS,R2   ;Get offset to RMON within context block
111 002100 072227 177772      ASH     #-6.,R2      ;Convert offset to 64-byte units
112 002104 066102 000000G      ADD     LCXPAR(R1),R2 ;Add PAR base for context area
113 002110 010237 000000G      MOV     R2,@#UPAR7  ;Map user page 7 to table
114 002114 010237 000000G      MOV     R2,CUPAR7   ;Save info in context block

```

SETMAP -- Set up memory mapping registers for a job

```

115 002120 013737 000200' 0000000  MOV      RMNPDR,@#UPDR7 ;Set up mapping control
116 002126 013737 000200' 0000000  MOV      RMNPDR,CUPDR7 ;Save info in context block
117                                     ;
118                                     ; Map user page 6 to context block if KMON is running
119                                     ;
120 002134 032761 0000000 0000000 4$:  BIT      ##INKMN,LSW4(R1); Is Kmon running?
121 002142 001414                                     BEQ      10$ ;Br if not
122 002144 016100 0000000                                     MOV      LCXPAR(R1),R0 ;Get PAR value to map context block
123 002150 010037 0000000                                     MOV      R0,@#UPAR6 ;Map user page 6 to job context area
124 002154 010037 0000000                                     MOV      R0,CUPAR6 ;Save info in context block
125 002160 013737 000166' 0000000  MOV      CXTADR,@#UPDR6 ;Allow read and write access
126 002166 013737 000166' 0000000  MOV      CXTADR,CUPDR6
127                                     ;
128                                     ; See if job has any associated shared run-time systems
129                                     ;
130 002174 005002                                     10$:  CLR      R2 ;Init table index for PAR0
131 002176 016200 0000000 13$:  MOV      RPDR(R2),R0 ;Do we need to load this PAR?
132 002202 001412                                     BEQ      14$ ;Br if not
133 002204 010062 0000000                                     MOV      R0,UPDR0(R2) ;Set PDR control flags
134 002210 010062 0000000                                     MOV      R0,CUPDR0(R2) ;Save info in context block
135 002214 016262 0000000 0000000  MOV      RPAR(R2),UPAR0(R2); Set PAR value
136 002222 016262 0000000 0000000  MOV      RPAR(R2),CUPAR0(R2); Save info in context block
137 002230 062702 0000002 14$:  ADD      #2,R2 ;Advance PAR table index
138 002234 020227 000016                                     CMP      R2,#2*7 ;Have we done all 7 PAR's?
139 002240 101756                                     BLOS    13$ ;Br if more to load
140                                     ;
141                                     ; Set flag that says mapping information in job context block is valid
142                                     ;
143 002242 110137 0000000                                     MOV      R1,MAPUSR ;Say memory mapping set up for this job
144 002246 032761 0000000 0000000  BIT      ##KINIT,LSW(R1) ;Has Kmon finished initializing context block?
145 002254 001403                                     BEQ      9$ ;Br if context block not initialized yet
146 002256 052761 0000000 0000000  BIS      ##MAPOK,LSW7(R1); Mapping info in context block is valid
147                                     ;
148                                     ; Finished
149                                     ;
150 002264 012605 9$:  MOV      (SP)+,R5
151 002266 012604                                     MOV      (SP)+,R4
152 002270 012603                                     MOV      (SP)+,R3
153 002272 012602                                     MOV      (SP)+,R2
154 002274 000207                                     RETURN

```

```

1          . SBTTL  LODMAP -- Load memory mapping from context block
2          ;-----
3          ; LODMAP is called to load the user-mode memory management
4          ; PAR and PDR registers from the mapping information in the job's
5          ; context block. The context block information was set up by the
6          ; last call to the SETMAP routine.
7          ;
8 002276 010146 LODMAP: MOV      R1,-(SP)
9 002300 010246      MOV      R2,-(SP)
10 002302 010346      MOV      R3,-(SP)
11 002304 010446      MOV      R4,-(SP)
12          ;
13          ; Set up pointers to the registers
14          ;
15 002306 012701 0000000 MOV      #UPARO,R1      ;Point to hardware PAR register base
16 002312 012702 0000000 MOV      #CUPARO,R2     ;Point to PAR data cells in context block
17 002316 012703 0000000 MOV      #UPDRO,R3     ;Point to hardware PDR register base
18 002322 012704 0000000 MOV      #CUPDRO,R4    ;Point to PDR data cells in context block
19          ;
20          ; Load the PAR and PDR registers
21          ;
22 002326 012700 000010  MOV      #8,R0          ;Load 8 PAR and PDR registers
23 002332 012221 1$:    MOV      (R2)+,(R1)+    ;Load a user-mode PAR register
24 002334 012423      MOV      (R4)+,(R3)+    ;Load a user-mode PDR register
25 002336 077003      SOB      R0,1$          ;Loop if more to load
26          ;
27          ; Say memory mapping is set up for the current job
28          ;
29 002340 113737 0000000 0000000 MOVB    CORUSR,MAPUSR   ;Say memory mapping set up for current job
30          ;
31          ; Finished
32          ;
33 002346 012604      MOV      (SP)+,R4
34 002350 012603      MOV      (SP)+,R3
35 002352 012602      MOV      (SP)+,R2
36 002354 012601      MOV      (SP)+,R1
37 002356 000207      RETURN

```

```

1          .SBTTL  MAPSYS -- Map to the system code regions
2          ;-----
3          ;
4          ; MAPSYS actually performs the mapping required to execute code in
5          ; the mapped system regions.  The mapped system code includes TSUSR,
6          ; TSEMT, TSLOCK, and TSSPOL.  The correct calling interface is
7          ;
8          ;     CALL  MAPSYS
9          ;     with R5 containing the <segment number>#6
10         ;
11         ; The output of the subroutine actually changes the KPAR5 contents to
12         ; get to the correct physical memory locations.
13         ; This routine must preserve the C-bit because it will be called
14         ; from the DCALL (inter-overlay call) handler.
15         ;
16         ;
17         ; Overlay table structure:
18         ;
19         ;     loc 64 --> #OVTAB:
20         ;         .WORD  <IDENTIFIER>,<KPAR5>,<WORD COUNT>
21         ;         DUMMY SUBROUTINES FOR ALL OVERLAY SEGMENTS
22         ;
23 002360  MAPSYS:
24 002360  011646  MOV      (SP),-(SP)      ;reposition return address
25 002362  013766  000000G 000002  MOV      @#PSW,2(SP)    ;save psw with current interrupt priority
26 002370  052737  000340  000000G  BIS      #340,@#PSW    ;disable interrupts
27 002376  010537  000032'  MOV      R5,SYSMAP     ;store the mapped region number
28 002402  063705  000000G  ADD      OVRADD,R5     ;add the pointer to the overlay table
29 002406  016537  177772G 000000G  MOV      0.PAR-6(R5),@#KPAR5;enter the physical memory location
30 002414  000002  RTI                      ;return with previous psw and priority
  
```

PKSTAT -- Pack user status into context block

```

1          .SBTTL  PKSTAT -- Pack user status into context block
2          ;-----
3          ; PKSTAT is called to pack all of the status information about the
4          ; current user into the user's context block.
5          ; After calling PKSTAT the user is ready to be outswapped.
6          ; All registers are preserved.
7          ; When called, we must be using the system stack.
8          ;
9          ; Inputs:
10         ; R3 = Pointer to top of user stack in job context area.
11         ;
12         ; Outputs:
13         ; R3 = Updated user stack pointer (after pushing job status).
14         ;
15 002416 010146 PKSTAT: MOV      R1,-(SP)
16 002420 113701 0000000 MOVB   CORUSR,R1      ;Get job index number
17         ;
18         ; Save PSW and user mode SP.
19         ;
20 002424 013743 0000000 MOV     @#PSW,-(R3)    ;Save PSW
21 002430 052737 0000000 0000000 BIS     #UPMODE,@#PSW ;Make sure previous-mode = user
22 002436 106506 MFPD   SP            ;Get user-mode SP
23 002440 011643 MOV     (SP),-(R3)    ;Push user SP
24 002442 012637 0000000 MOV     (SP)+,UMSPSV  ;Save it in context block cell also
25         ;
26         ; Save some cells in program space
27         ;
28 002446 032761 0000000 0000000 BIT     #RDSAV,LSW11(R1);Are we reading in SAV file now?
29 002454 001007 BNE     1$           ;Br if yes -- Don't save info from memory
30 002456 106537 0000000 MFPD   @#ERRLOC      ;Save error cells
31 002462 012643 MOV     (SP)+,-(R3)
32 002464 106537 0000000 MFPD   @#JSWLOC      ;Job status word
33 002470 012661 0000000 MOV     (SP)+,LJSW(R1)
34         ;
35         ; Save the current kernel PAR 5 region mapping
36         ;
37 002474 013743 0000000 1$: MOV     @#KPAR5,-(R3) ;Save kernel par 5 mapping
38         ;
39         ; See if we need to save state of FPU
40         ;
41 002500 005737 0000000 TST     UFPTRP        ;Is user using the FPU?
42 002504 001424 BEQ     2$           ;Br if not
43 002506 032737 0000000 0000000 BIT     #CW#FPU,CONFIG ;Does system have an FPU?
44 002514 001420 BEQ     2$           ;Br if not
45         ; Save contents of FPU accumulators
46 002516 010246 MOV     R2,-(SP)
47 002520 170202 STFPS  R2            ;Hold floating point status in R2
48 002522 010243 MOV     R2,-(R3)    ;Save floating point status register
49 002524 170011 SETD   ;Set to 64-bit mode
50 002526 174043 STD     R0,-(R3)    ;AC0
51 002530 174143 STD     R1,-(R3)    ;AC1
52 002532 174243 STD     R2,-(R3)    ;AC2
53 002534 174343 STD     R3,-(R3)    ;AC3
54 002536 172404 LDD     R4,R0        ;AC4-->AC0
55 002540 174043 STD     R0,-(R3)    ;(AC4)
56 002542 172405 LDD     R5,R0        ;AC5-->AC0
57 002544 174043 STD     R0,-(R3)    ;(AC5)

```

PKSTAT -- Pack user status into context block

```
58 002546 170102          LDFPS  R2          ;Leave FPU status same as on entry
59 002550 105037 000013'  CLRB   FPUUSE     ;FPU is no longer in use
60 002554 012602          MOV    (SP)+,R2
61                               ;
62                               ; Finished saving status
63                               ;
64 002556 012601 2$:     MOV    (SP)+,R1
65 002560 000207          RETURN
```

UPSTAT -- Unpack user status from context block

```

1          .SBTTL  UPSTAT -- Unpack user status from context block
2          ;-----
3          ; UPSTAT is called to unpack the status information stored in the
4          ; user's context area and get the user ready to run.
5          ; When called, R1 must contain the user index #.
6          ; All registers are preserved except R0.
7          ;
8          ; Inputs:
9          ;   R1 = Job index number.
10         ;   R3 = User context area stack pointer
11         ;
12         ; Outputs:
13         ;   R3 = User context area stack pointer after job status is popped.
14         ;
15 002562 010446  UPSTAT: MOV     R4,-(SP)
16 002564 010546          MOV     R5,-(SP)
17 002566 052737 0000000 0000000  BIS     #UPMODE,@#PSW ;Make sure previous-mode = user
18         ;
19         ; See if FPU unit needs to be restored
20         ;
21 002574 005737 0000000          TST     UFPTRP          ;Is user using FPU?
22 002600 001420          BEQ     3$          ;Br if not
23 002602 032737 0000000 0000000  BIT     #CW$FPU,CONFIG ;Does system have an FPU?
24 002610 001414          BEQ     3$          ;Br if not
25         ; Restore the FPU registers and status
26 002612 110137 000013'          MOVB   R1,FPUUSE          ;Set flag saying FPU unit is in use by job
27 002616 170011          SETD   ;Set 64-bit mode
28 002620 172423          LDD    (R3)+,R0          ;Get AC5
29 002622 174005          STD    R0,R5
30 002624 172423          LDD    (R3)+,R0          ;Get AC4
31 002626 174004          STD    R0,R4
32 002630 172723          LDD    (R3)+,R3          ;AC3
33 002632 172623          LDD    (R3)+,R2          ;AC2
34 002634 172523          LDD    (R3)+,R1          ;AC1
35 002636 172423          LDD    (R3)+,R0          ;AC0
36 002640 170123          LDFPS  (R3)+          ;Load FPU status register
37         ;
38         ; Restore the mapping for kernel PAR 5
39         ;
40 002642 012337 0000000 3$:    MOV     (R3)+,@#KPAR5 ;Restore mapping for PAR 5
41         ;
42         ; Restore some cells in program space
43         ;
44 002646 032761 0000000 0000000  BIT     #RDSAV,LSW11(R1);Are we reading in SAV file now?
45 002654 001007          BNE    1$          ;Br if yes -- Don't alter memory image
46 002656 016146 0000000          MOV     LJSW(R1),-(SP) ;Job status word
47 002662 106637 0000000          MTPD   @#JSWLOC
48 002666 012346          MOV     (R3)+,-(SP)    ;Error cells
49 002670 106637 0000000          MTPD   @#ERRLOC
50         ;
51         ; Restore PSW mode and user-mode SP.
52         ;
53 002674 012346 1$:    MOV     (R3)+,-(SP)    ;User-mode SP
54 002676 106606          MTPD   SP
55 002700 005037 0000000          CLR    UMSPSV          ;Say user SP has been loaded
56 002704 012300          MOV     (R3)+,R0          ;Get saved PS
57 002706 042700 147777          BIC    #147777,R0      ;Clear all but previous-mode field

```

UPSTAT -- Unpack user status from context block

```
58 002712 042737 0000000 0000000      BIC    #UPMODE,@#PSW    ;Clear previous-mode field in PS
59 002720 050037 0000000                BIS    RO,@#PSW        ;Possibly set previous-mode field
60                                     ;
61                                     ; Finished
62                                     ;
63 002724 012605                MOV    (SP)+,R5
64 002726 012604                MOV    (SP)+,R4
65 002730 000207                RETURN
```

DOCMPL -- Call job's completion routines

```

1          .SBTTL  DOCMPL -- Call job's completion routines
2          ;-----
3          ; DOCMPL is called from the job scheduler just before returning control
4          ; to a job.  On entry we are running on the user context-area stack.
5          ; If there are pending completion routine requests for this job and we
6          ; are not already executing in a completion routine, we call the completion
7          ; routines.
8          ;
9          ; Inputs:
10         ; R1 = Job index number
11         ;
12 002732  DOCMPL:
13         ;
14         ; See if any completion routine is pending for this job.
15         ;
16 002732  016100  0000000  MOV      LCMPL(R1),R0      ;Is any completion request pending for job?
17 002736  001410          BEQ      12$              ;Br if not
18         ;
19         ; There is a pending completion routine for the job.
20         ; Don't enter a completion routine if one of the same or higher class
21         ; priority is already in execution for the job.
22         ; However, allow a higher class completion routine to interrupt a
23         ; lower class routine.
24         ;
25 002740  126037  0000000  0000000  CMPB    CQ#CP(R0),CURCP ;Is this request of higher priority?
26 002746  101404          BLOS    12$              ;Br if not
27         ;
28         ; We want to enter this completion routine.
29         ; Don't enter any synch or completion routines if no-abort flag is set.
30         ;
31 002750  032761  0000000  0000000  BIT     #$NOABT,LSW9(R1);Was the no-abort flag set?
32 002756  001401          BEQ     13$              ;Br if not
33 002760  000207          12$:   RETURN          ;Don't do any completion routine processing
34         ;
35         ; There is a pending completion routine request and we are not currently
36         ; executin a completion routine so we should call the pending routines.
37         ;
38         ; Save status of job on context-block stack.
39         ;
40 002762  010046          13$:   MOV     R0,-(SP)
41 002764  010146          MOV     R1,-(SP)
42 002766  010246          MOV     R2,-(SP)
43 002770  010346          MOV     R3,-(SP)
44 002772  010446          MOV     R4,-(SP)
45 002774  010546          MOV     R5,-(SP)
46 002776  013746  0000000  MOV     @#KPAR5,-(SP) ;Save kernel PAR 5 mapping
47         ;
48         ; Switch to system stack so PKSTAT can push job status on context-block
49         ; stack.
50         ;
51 003002  010603          MOV     SP,R3          ;Save current job-context-block stack pointer
52 003004  012706  001000  MOV     #SS,SP         ;Switch to system stack
53 003010  113705  000013'  MOVB   FPUUSE,R5       ;Remember if job is using FPU
54 003014  004737  002416'  CALL   PKSTAT          ;Save job status on context-block stack
55 003020  110537  000013'  MOVB   R5,FPUUSE       ;Tell system if job is using FPU
56 003024  010306          MOV     R3,SP         ;Switch back to context stack
57 003026  005037  0000000  CLR    UMSPSV          ;Say user SP is active- used for real-time int

```

```

58 ;
59 ; Our status has been saved.
60 ; Save current job execution priority on stack so we can restore it after
61 ; calling all completion routines.
62 ;
63 003032 116146 0000000 MOVBL LPRI(R1),-(SP) ; Save job execution priority
64 ;
65 ; Process next completion request.
66 ;
67 003036 1#: DISABL ; ** Disable interrupts **
68 003044 012700 0000000 MOV #LCMPL-CQ#LNK,RO ; Get fake pointer to compl Q head
69 003050 060100 ADD R1,RO ; Point to list head for our job
70 003052 016005 0000000 17#: MOV CQ#LNK(RO),R5 ; Get address of next compl Q element
71 003056 001540 BEQ 2# ; Br if no more elements left to process
72 003060 126537 0000000 0000000 CMPBL CQ#CP(R5),CURCP ; Is this request class higher than current
73 003066 101534 BLOS 2# ; Br if not -- No more requests to process now
74 003070 132765 0000000 0000000 BITB #QF#SCR,CQ#FLG(R5) ; Is this a system or user compl routine?
75 003076 001005 BNE 16# ; Br if system compl routine
76 003100 005761 0000000 TST LIOHLD(R1) ; Should we hold user compl routines?
77 003104 001402 BEQ 16# ; Br if not
78 003106 010500 MOV R5,RO ; Link to next element
79 003110 000760 BR 17# ; Skip over user compl requests in list
80 ;
81 ; Unlink completion queue request from list.
82 ;
83 003112 016560 0000000 0000000 16#: MOV CQ#LNK(R5),CQ#LNK(RO) ; Remove compl request from list
84 003120 ENABL ; ** Enable interrupts **
85 ;
86 ; Set job execution priority to that specified in completion queue element
87 ;
88 003126 113746 0000000 MOVBL CURCP, -(SP) ; Save current completion rtn class priority
89 003132 116537 0000000 0000000 MOVBL CQ#CP(R5),CURCP ; Remember class priority of compl rtn
90 003140 116500 0000000 MOVBL CQ#PRI(R5),RO ; Get execution priority for compl queue
91 003144 120061 0000000 CMPBL RO,LPRI(R1) ; Do we need to change job priority?
92 003150 001004 BNE 7# ; Br if not
93 003152 110061 0000000 MOVBL RO,LPRI(R1) ; Set new execution priority
94 003156 105237 000004' INCB DOSCHD ; Say a job scheduler cycle is needed
95 ;
96 ; Get some information out of the completion queue element and then
97 ; free the queue element.
98 ;
99 003162 010103 7#: MOV R1,R3 ; Save job index number in R3
100 003164 016504 0000000 MOV CQ#RTN(R5),R4 ; Address of completion routine to be called
101 003170 016546 0000000 MOV CQ#RO(R5),-(SP) ; Save info on stack for now
102 003174 016546 0000000 MOV CQ#R1(R5),-(SP)
103 003200 116502 0000000 MOVBL CQ#FLG(R5),R2 ; Control flags
104 003204 016537 0000000 0000000 MOV CQ#PA5(R5),@#KPAR5 ; Set up mapping for PAR 5
105 003212 010501 MOV R5,R1 ; Get address of queue element to R1
106 003214 004737 0000000 CALL QFREE ; Free the queue element
107 003220 012601 MOV (SP)+,R1 ; Recover values to pass in RO & R1
108 003222 012600 MOV (SP)+,RO
109 ;
110 ; Determine if completion routine is a system routine that should
111 ; be called in kernel mode or a user completion routine to be
112 ; called in user mode.
113 ;
114 003224 032702 0000000 BIT #QF#SCR,R2 ; Is this a system or user completion routine?

```

```

115 003230 001406          BEQ      6$          ;Br if user completion routine
116
117          ; We are calling a system completion routine in kernel mode
118
119 003232 032702 000000C    BIT      #QF$SYN!QF$IOT,R2 ;Is this a .SYNCH or .TIMID routine?
120 003236 001401          BEQ      15$          ;Br if not
121 003240 005011          CLR      (R1)          ;Clear cell in original call argument block
122 003242 004714    15$:    CALL     (R4)          ;Call system completion routine
123 003244 000440          BR       5$          ;Finished with completion routine
124
125          ; We are calling a user completion routine.
126          ; See if flag is set indicating that we are not to call any user
127          ; completion routines. This is set during job exit/abort cleanup.
128
129 003246 032763 000000G 000000G 6$:    BIT      ##NOUCR,LSW9(R3);Should we ignore user-mode completion rtns?
130 003254 001034          BNE      5$          ;Br if yes
131
132          ; Use special EMT to regain control at end of completion routine.
133
134 003256 052737 000000G 000000G    BIS      #UPMODE,@#PSW    ;Make sure previous mode = user
135 003264 013702 000000G          MOV      EMTCAD,R2      ;Get pointer to top of return addr stack
136 003270 012742 003346'          MOV      #5$,-(R2)     ;Push our return address
137 003274 010237 000000G          MOV      R2,EMTCAD     ;Save updated stack pointer
138 003300 106506          MFPD     SP            ;Get user's stack pointer
139 003302 012602          MOV      (SP)+,R2     ;
140 003304 012746 000000G          MOV      #CPLEMT,-(SP) ;Store address of exit EMT on user's stack
141 003310 106642          MTPD     -(R2)        ;
142 003312 010246          MOV      R2,-(SP)     ;Restore updated user's stack pointer
143 003314 106606          MTPD     SP            ;
144
145          ; Now push fake PS & PC on stack so RTI will enter compl routine
146          ; in user-mode.
147
148 003316 012746 000000C          MOV      #UMODE!UPMODE,-(SP);User-mode PS
149 003322 032737 000000G 000000G    BIT      #D$DMON,D.FLAG ;Is debugger doing data monitoring?
150 003330 001402          BEQ      14$          ;Br if not
151 003332 052716 000020          BIS      #20,(SP)     ;Set trap flag in PSW
152 003336 010446    14$:    MOV      R4,-(SP)     ;Address of completion routine
153 003340 012705 000024          MOV      #24,R5      ;Make R5 point to word with 0 for FORTRAN subs
154 003344 000002          RTI              ;Enter completion routine in user mode
155
156          ; Return here from the completion routine (when user does EMT instruction).
157
158          ; See if there are more completion routines to call.
159
160 003346 113701 000000G    5$:    MOVVB   CORUSR,R1      ;Get back job index number
161 003352 112637 000000G          MOVVB   (SP)+,CURCP   ;Restore compl routine class priority
162 003356 000627          BR       1$          ;Check for more pending completion requests
163
164          ; There are no more pending completion requests.
165          ; Reset job priority to base priority for job.
166
167 003360    2$:    ENABL     ;** Enable interrupts **
168 003366 112603          MOVVB   (SP)+,R3      ;Get saved job execution priority
169 003370 105737 000000G          TSTB   CURCP         ;Are we still in a completion routine?
170 003374 001025          BNE      8$          ;Br if yes
171 003376 116103 000000G          MOVVB   LBSPRI(R1),R3 ;Get base priority for job

```

DOCMPL -- Call job's completion routines

```

172 003402 032761 0000000 0000000 BIT    #$VNDOT,LSW(R1) ; Is this a disconnected virtual job?
173 003410 001417          BEQ     B$           ; Br if not
174 003412 120337 0000000          CMPB   R3,VPRIHI    ; Does job have a fixed high priority?
175 003416 103014          BHIS   B$           ; Br if yes
176 003420 120337 0000000          CMPB   R3,VPRILO    ; Does job have a fixed low priority?
177 003424 101411          BLOS   B$           ; Br if yes
178 003426 113700 0000000          MOVB  VPRIVR,R0    ; Get prio reduction for virtual jobs
179 003432 160003          SUB    R0,R3       ; Reduce priority for detached virtual jobs
180 003434 120337 0000000          CMPB   R3,VPRILO    ; But don't go into special low prio range
181 003440 003003          BGT    B$         ; Br if ok
182 003442 113703 0000000          MOVB  VPRILO,R3    ; Force above fixed low priority
183 003446 005203          INC   R3          ;
184 003450 120361 0000000      B$:    CMPB   R3,LPRI(R1) ; Are we changing the job's priority?
185 003454 001404          BEQ    10$        ; Br if not
186 003456 110361 0000000          MOVB  R3,LPRI(R1) ; Set new priority for job
187 003462 105237 000004'          INCB  DOSCHD      ; Say a job scheduler cycle is needed
188 003466 026127 0000000 0000000 10$:    CMP    LSTATE(R1),#S$#RT ; Is job in a real-time state?
189 003474 101005          BHI    3$         ; Br if not
190 003476 120337 0000000          CMPB   R3,VPRIHI    ; Does job have a real-time priority?
191 003502 103002          BHIS   3$         ; Br if yes
192 003504 004737 005206'          CALL  QHIPRI      ; Requeue job in normal high-prio state
193
194          ; Unpack job status.
195
196 003510 010603      3$:    MOV    SP,R3          ; Save user context-block stack pointer
197 003512 012706 001000          MOV    #SS,SP      ; Switch to system stack
198 003516 004737 002562'          CALL  UPSTAT       ; Unpack job status
199 003522 010306          MOV    R3,SP       ; Switch back to context-block stack
200
201          ; Finished
202
203 003524 012637 0000000          MOV    (SP)+,@#KPAR5 ; Restore mapping for kernel PAR 5
204 003530 012605          MOV    (SP)+,R5
205 003532 012604          MOV    (SP)+,R4
206 003534 012603          MOV    (SP)+,R3
207 003536 012602          MOV    (SP)+,R2
208 003540 012601          MOV    (SP)+,R1
209 003542 012600          MOV    (SP)+,R0
210 003544 000207          RETURN

```

SUTOP -- Set top of memory for a job

```

1          .SBTTL  SUTOP  -- Set top of memory for a job
2          ;-----
3          ; SUTOP is called to set the top of memory for the current job.
4          ; If memory expansion is being requested and the required memory space
5          ; is not available, the job is suspended and outswapped until memory
6          ; becomes available.
7          ; If a memory contraction is being done, the memory area being freed
8          ; is returned to an unused state.
9          ;
10         ; Inputs:
11         ; RO = Address above desired top of program.
12         ;
13 003546 010046 SUTOP:  MOV    RO,-(SP)
14 003550 010146      MOV    R1,-(SP)
15 003552 010446      MOV    R4,-(SP)
16 003554 010546      MOV    R5,-(SP)
17 003556 013746 0000000  MOV    @#KPAR5,-(SP) ; Save system PAR 5 mapping
18 003562 013737 0000000 0000000  MOV    MAPPAR,@#KPAR5 ; Map PAR 5 to memory allocation table
19 003570 113701 0000000      MOVVB  CORUSR,R1 ; Get job index #
20         ;
21         ; Set highest legal address for job and set base of USR
22         ;
23 003574 010037 0000000      MOV    RO,UHIMEM ; SET TOP OF MEMORY FOR JOB
24 003600 013705 000170'      MOV    CXTRMN,R5 ; GET ADDRESS OF JOB'S SIMULATED MON VEC TABLE
25 003604 010065 0000000      MOV    RO,R#UBAS(R5) ; SAY BASE OF USR = TOP OF JOB
26         ;
27         ; Convert top-of-memory address to 512-byte page number.
28         ;
29 003610 020027 177000      CMP    RO,#177000 ; DOES JOB NEED 64KB?
30 003614 101403      BLOS  7$ ; BR IF NOT
31 003616 012700 000200      MOV    #128.,RO ; 128 PAGES = 64KB
32 003622 000407      BR    8$
33 003624 062700 000777 7$:  ADD    #511.,RO ; BOUND UP TO PAGE BOUNDARY
34 003630 000241      CLC ; CONVERT # BYTES TO # WORDS
35 003632 006000      ROR    RO
36 003634 000300      SWAB  RO ; CONVERT TO # PAGES
37 003636 042700 177400      BIC    #^C377,RO ; MASK OUT ALL BUT # PAGES
38 003642 063700 000164' 8$:  ADD    CXTPAG,RO ; ADD # PAGES NEEDED FOR CONTEXT BLOCK
39         ;
40         ; Compare new memory request with current memory allocation for this job.
41         ;
42 003646 020061 0000000 10$:  CMP    RO,LNBLKS(R1) ; COMPARE NEW REQUEST WITH CURRENT ALLOCATION
43 003652 001501      BEQ   3$ ; BR IF NO CHANGE IN SIZE
44 003654 101026      BHI   1$ ; BR IF EXPANDING MEMORY SIZE
45         ;
46         ; We are decreasing the size of the job.
47         ; Free the memory pages above the new top of the job.
48         ;
49 003656 016104 0000000      MOV    LBASE(R1),R4 ; GET BASE PAGE # ASSIGNED TO THE JOB
50 003662 016105 0000000      MOV    LNBLKS(R1),R5 ; GET # PAGES ASSIGNED TO JOB NOW
51 003666 060504      ADD    R5,R4 ; GET # OF PAGE ABOVE TOP OF JOB AREA
52 003670 063704 0000000      ADD    BASMAP,R4 ; POINT TO ENTRY IN MEMMAP TABLE
53 003674 010061 0000000      MOV    RO,LNBLKS(R1) ; SET NEW # PAGES ASSIGNED TO JOB
54 003700 010061 0000000      MOV    RO,LMEMIN(R1) ; SET # BLOCKS NEEDED BY INSWAP
55 003704 066161 0000000 0000000  ADD    LNSBLK(R1),LMEMIN(R1) ; ADD MEMORY SPACE NEEDED FOR PLAS REGNS
56 003712 160005      SUB    RO,R5 ; GET # PAGES BEING FREED
57 003714 060537 0000000      ADD    R5,FREPGS ; KEEP TRACK OF # FREE PAGES

```

```

58 003720 105044          2$:   CLRB   -(R4)           ;FREE A PAGE
59 003722 077502          SOB    R5,2$           ;LOOP TO FREE MORE
60 003724 105237 000004'  INCB   DOSCHD         ;REQUEST A JOB SCHEDULER CYCLE
61 003730 000452          BR     3$
62                          ;
63                          ; We are increasing the size of the job.
64                          ;
65                          ; See if desired memory is available now.
66                          ;
67 003732 016104 0000000 1$:   MOV    LBASE(R1),R4   ;GET BASE PAGE # ASSIGNED TO THIS JOB
68 003736 016105 0000000   MOV    LNBLKS(R1),R5   ;GET # PAGES CURRENTLY ASSIGNED TO THIS JOB
69 003742 060504          ADD    R5,R4           ;GET # OF PAGE ABOVE TOP OF JOB AREA
70 003744 063704 0000000   ADD    BASMAP,R4      ;POINT INTO MEMMAP TABLE
71 003750 160500          SUB    R5,R0           ;GET # PAGES TO BE ADDED
72 003752 010046          MOV    R0,-(SP)
73 003754 105724          5$:   TSTB   (R4)+         ;IS THIS PAGE AVAILABLE?
74 003756 001025          BNE   4$              ;BR IF NOT
75 003760 077003          SOB    R0,5$          ;CHECK ALL PAGES WE NEED
76                          ;
77                          ; The desired memory space is available. Claim it for our job.
78                          ;
79 003762 012600          MOV    (SP)+,R0       ;GET # PAGES BEING ADDED
80 003764 016104 0000000   MOV    LBASE(R1),R4   ;GET BASE PAGE # ASSIGNED TO THIS JOB
81 003770 066104 0000000   ADD    LNBLKS(R1),R4   ;GET # OF PAGE ABOVE CURRENT TOP OF JOB
82 003774 060061 0000000   ADD    R0,LNBLKS(R1)  ;INCREASE # PAGES ASSIGNED TO THIS JOB
83 004000 016161 0000000 0000000   MOV    LNBLKS(R1),LMEMIN(R1);SET UP LMEMIN FOR SWAPPER
84 004006 066161 0000000 0000000   ADD    LNSBLK(R1),LMEMIN(R1);ADD SPACE NEEDED FOR PLAS REGIONS
85 004014 063704 0000000   ADD    BASMAP,R4      ;POINT INTO MEMMAP TABLE
86 004020 160037 0000000   SUB    R0,FREPGS     ;KEEP TRACK OF # FREE PAGES
87 004024 110124          6$:   MOVB   R1,(R4)+       ;CLAIM PAGES FOR OUR JOB
88 004026 077002          SOB    R0,6$
89 004030 000412          BR     3$
90                          ;
91                          ; The memory space we need is not now available.
92                          ; Force the job to be suspended and outswapped.
93                          ; The subsequent inswap will do the memory expansion for us.
94                          ;
95 004032 012600          4$:   MOV    (SP)+,R0       ;GET # PAGES BEING ADDED
96 004034 066100 0000000   ADD    LNBLKS(R1),R0   ;GET NEW TOTAL # PAGES FOR JOB ROOT
97 004040 010046          MOV    R0,-(SP)       ;SAVE SIZE OF JOB ROOT
98 004042 066100 0000000   ADD    LNSBLK(R1),R0  ;ADD SPACE NEEDED FOR PLAS REGIONS
99 004046 004737 004100'   CALL   MEMXPN         ;DO JOB SWAP TO EXPAND MEMORY SPACE
100 004052 012600          MOV    (SP)+,R0       ;GET BACK # PAGES NEEDED BY JOB ROOT
101 004054 000674          BR     10$           ;WE SHOULD NOW HAVE ALL NEEDED
102                          ;
103                          ; Memory allocation has been done.
104                          ; Load the memory management registers for the job.
105                          ;
106 004056 004737 001534'  3$:   CALL   SETMAP        ;LOAD MEMORY MANAGEMENT REGISTERS FOR THE JOB
107                          ;
108                          ; Finished
109                          ;
110 004062 012637 0000000   MOV    (SP)+,@#KPAR5  ;Restore system PAR 5 mapping
111 004066 012605          MOV    (SP)+,R5
112 004070 012604          MOV    (SP)+,R4
113 004072 012601          MOV    (SP)+,R1
114 004074 012600          MOV    (SP)+,R0

```

115 004076 000207

RETURN

MEMXPN -- Do job swap to expand memory size

```

1          .SBTTL  MEMXPN -- Do job swap to expand memory size
2          ;-----
3          ; MEMXPN is called when we want to expand the size of a job
4          ; but are unable to do so because we don't have free memory space
5          ; above the top of the job.
6          ; MEMXPN outswaps the job and then swaps the job back into the
7          ; larger memory region.
8          ;
9          ; Inputs:
10         ; R1 = Job index number.
11         ; R0 = Total number of 512-byte pages wanted for job after expansion.
12         ;
13 004100 010061 000000G MEMXPN: MOV      R0,LMEMIN(R1) ;Set job size needed
14 004104 032761 000000G 000000G BIT      ##NDMEM,LSW(R1) ;Is job already waiting for memory expansion?
15 004112 001005          BNE      1$          ;Br if yes
16 004114 052761 000000G 000000G BIS      ##NDMEM,LSW(R1) ;Set memory-needed flag for the job
17 004122 005237 000070' INC      MEMSWP      ;Tell swapper than memory-swap is needed
18 004126 012700 000000G 1$: MOV     #S$WFM,R0    ;Put job in waiting-for-memory state
19 004132 004737 004736' CALL     QNSPNX      ;Suspend and do the swap
20 004136 004737 005122' CALL     CHKABT      ;Was job aborted while suspended?
21         ;
22         ; Finished
23         ;
24 004142 000207          RETURN

```

CXBMOV -- Move job context data into buffer

```

1          .SBTTL  CXBMOV -- Move job context data into buffer
2          ;-----
3          ; This routine is called to move data from some area of physical memory
4          ; into the job context block access buffer (CXTBUF).
5          ; This routine is placed in the root because it uses PAR 5 to access
6          ; the physical memory area.
7          ;
8          ; Inputs:
9          ; R5 = Base 64-byte block number of start of data in physical memory.
10         ; R3 = Offset within data area of item being accessed.
11         ; R0 = Number of bytes of data to move (512 maximum).
12         ;
13 004144 010346 CXBMOV: MOV     R3,-(SP)
14 004146 010446      MOV     R4,-(SP)
15 004150 013746 0000000      MOV     @#KPAR5,-(SP) ; Save PAR 5 mapping
16         ;
17         ; Map PAR 5 to area being accessed
18         ;
19 004154 010537 0000000      MOV     R5,@#KPAR5 ; Map par 5 to context block
20         ;
21         ; Set up registers for the move
22         ;
23 004160 062703 0000000      ADD     #VPAR5,R3 ; Get mapped address of start of data
24 004164 013704 000172'      MOV     CXTBUF,R4 ; Point to buffer where data is to go
25 004170 006200      ASR     R0 ; Get # words to move
26         ;
27         ; Move the data
28         ;
29 004172 006200      ASR     R0 ; Get number of double-words to move
30 004174 001403      BEQ     2$ ; Br if less than 2 words to move
31 004176 012324 1$:      MOV     (R3)+,(R4)+ ; Move a word
32 004200 012324      MOV     (R3)+,(R4)+ ; Move second word of pair
33 004202 077003      SDB     R0,1$ ; Loop till all moved
34 004204 103001 2$:      BCC     9$ ; Br if don't need to move odd word at end
35 004206 011314      MOV     (R3),(R4) ; Move last word
36         ;
37         ; Finished
38         ;
39 004210 012637 0000000 9$:      MOV     (SP)+,@#KPAR5 ; Restore par 5 mapping
40 004214 012604      MOV     (SP)+,R4
41 004216 012603      MOV     (SP)+,R3
42 004220 000207      RETURN

```

ENQHD -- Put user at head of queue

```

1          .SBTTL  ENQHD  -- Put user at head of queue
2          ;-----
3          ; ENQHD is called to place a user in the run queue at the front
4          ; of the list of users of equal or lower priority.
5          ; When called, R1 must contain the user index number and
6          ; R0 must contain the execution state (S$----).
7          ; All registers are preserved.
8          ;
9 004222  010246  ENQHD:  MOV     R2,-(SP)
10 004224  010346      MOV     R3,-(SP)
11 004226  010446      MOV     R4,-(SP)
12          ;
13          ; If job is being placed in an executable state and the priority
14          ; of the job is one of the fixed priorities (very low or very high),
15          ; then force job to be placed in the S$LOW or S$RT queue.
16          ;
17 004230  116104  0000000  MOVB   LPRI(R1),R4      ;Get current priority for job
18 004234  020027  0000000  CMP    R0,#S$RUN      ;Is job being placed in an executable state?
19 004240  101013      BHI    3$              ;Br if not
20 004242  120437  0000000  CMPB  R4,VPRIHI      ;Does job have a real-time priority?
21 004246  103403      BLO   4$              ;Br if not
22 004250  012700  0000000  MOV   #S$RT,R0       ;Force real-time jobs into S$RT state
23 004254  000405      BR    3$
24 004256  120437  0000000  4$:   CMPB  R4,VPRILO  ;Is this a low-priority job?
25 004262  101002      BHI   3$              ;Br if not
26 004264  012700  0000000  MOV   #S$LOW,R0      ;Force into low-priority queue
27          ;
28          ; Remove user from queue he is in currently.
29          ;
30 004270  004737  004534'  3$:   CALL   DEQ          ;REMOVE FROM QUEUE ** DISABLE **
31          ;
32          ; Search down queue looking for right place to insert user.
33          ;
34 004274  113702  000000'  MOVB  RUNQHD,R2       ;POINT TO FIRST USER IN QUEUE
35 004300  001432      BEQ   ADQ1            ;BR IF QUEUE IS EMPTY
36 004302  020062  0000000  1$:   CMP    R0,LSTATE(R2) ;COMPARE EXECUTION STATE PRIO WITH NEXT JOB
37 004306  101004      BHI   2$              ;BR IF OUR EXECUTION STATE IS LOWER PRIO
38 004310  103412      BLO  ADQMID           ;BR IF OUR EXECUTION STATE IS HIGHER PRIO
39 004312  120462  0000000  CMPB  R4,LPRI(R2)    ;EQUAL EXECUTION STATES, COMPARE PRIORITIES
40 004316  103007      BHIS  ADQMID         ;BR IF PRIO IS EQUAL TO OR HIGHER THAN NEXT
41 004320  010203  2$:   MOV   R2,R3        ;CHAIN ON TO NEXT USER IN LIST
42 004322  116302  0000000  MOVB  LQLINK(R3),R2
43 004326  001365      BNE   1$              ;BR IF MORE USERS IN QUEUE
44          ;
45          ; Add user to tail of queue
46          ;
47 004330  110137  000001'  ADQTL: MOVB  R1,RUNQTL  ;SAY WE ARE LAST USER IN LIST
48 004334  000404      BR    ADQT
49          ;
50          ; Link in front of user whose index # is in R2
51          ;
52 004336  116203  0000010  ADQMID: MOVB  LQLINK+1(R2),R3 ;GET INDEX OF EARLIER USER
53 004342  110162  0000010      MOVB  R1,LQLINK+1(R2) ;SAY WE ARE PREDECESSOR TO R2 USER
54 004346  110261  0000000  ADQT:  MOVB  R2,LQLINK(R1) ;SAY R2 USER FOLLOWS US
55 004352  110361  0000010      MOVB  R3,LQLINK+1(R1) ;SAY R3 USER IS OUR PREDECESSOR
56 004356  001405      BEQ   ADQHD          ;BR IF WE ARE AT HEAD OF LIST
57 004360  110163  0000000      MOVB  R1,LQLINK(R3)  ;SAY WE FOLLOW R3 USER

```

```
58 004364 000404          BR      ADQXIT
59
60          ; Set us as only entry in queue
61          ;
62 004366 110137 000001'  ADQ1:  MOV  R1,RUNQTL      ; MAKE QUEUE TAIL POINT TO US
63 004372 110137 000000'  ADQHD: MOV  R1,RUNQHD     ; MAKE QUEUE HEAD POINT TO US
64 004376 010061 0000000  ADQXIT: MOV  R0,LSTATE(R1) ; SET OUR EXECUTION STATE
65          ;
66          ; Finished. Request a job scheduler cycle.
67          ;
68 004402 105237 000004'          INCB   DOSCHD      ; REQUEST A JOB SCHEDULER CYCLE
69 004406          ENABL          ; ** ENABLE **
70 004414 012604          MOV    (SP)+,R4
71 004416 012603          MOV    (SP)+,R3
72 004420 012602          MOV    (SP)+,R2
73 004422 000207          RETURN
```

ENQTL -- Add user to tail of execution queue

```

1          .SBTTL  ENQTL  -- Add user to tail of execution queue
2          ;-----
3          ; ENQTL is called when it is desired to add the user whose
4          ; index number is in R1 to the end of the list of users with
5          ; the execution state whose code is in R0.  If there are no other
6          ; users with this state in the queue, the user is linked in
7          ; in front of any lower priority users.
8          ; All registers are preserved.
9          ;
10         004424  010246  ENQTL:  MOV      R2,-(SP)
11         004426  010346          MOV      R3,-(SP)
12         004430  010446          MOV      R4,-(SP)
13         ;
14         ; If job is being placed in an executable state and the priority
15         ; of the job is one of the fixed priorities (very low or very high),
16         ; then force job to be placed in the S$LOW or S$RT queue.
17         ;
18         004432  116104  0000000  MOVVB   LPRI(R1),R4      ;Get current priority for job
19         004436  020027  0000000  CMP     R0,#S$$RUN     ;Is job being placed in an executable state?
20         004442  101013          BHI     3$             ;Br if not
21         004444  120437  0000000  CMPB   R4,VPRIHI      ;Does job have a real-time priority?
22         004450  103403          BLO     4$             ;Br if not
23         004452  012700  0000000  MOV     #S$RT,R0      ;Force real-time jobs into S$RT state
24         004456  000405          BR      3$
25         004460  120437  0000000  4$:    CMPB   R4,VPRILO  ;Is this a low-priority job?
26         004464  101002          BHI     3$             ;Br if not
27         004466  012700  0000000  MOV     #S$LOW,R0     ;Force into low-priority queue
28         ;
29         ; Remove user from queue.
30         ;
31         004472  004737  004534'  3$:    CALL   DEQ          ;REMOVE FROM QUEUE ** DISABLE **
32         ;
33         ; Search for right place to insert user.
34         ;
35         004476  113702  000000'  MOVVB   RUNQHD,R2     ;POINT TO 1ST USER IN QUEUE
36         004502  001731          BEQ     ADQ1          ;BR IF QUEUE IS EMPTY
37         004504  020062  0000000  1$:    CMP     R0,LSTATE(R2) ;COMPARE EXECUTION STATES
38         004510  101004          BHI     2$             ;BR IF OUR EXECUTION STATE PRIO IS LOWER
39         004512  103711          BLO     ADQMID        ;BR IF OUR EXECUTION STATE PRIO IS HIGHER
40         004514  120462  0000000  CMPB   R4,LPRI(R2)   ;EQUAL EXECUTION STATES, COMPARE PRIORITIES
41         004520  101306          BHI     ADQMID        ;BR IF OUR PRIORITY IS HIGHER
42         004522  010203  0000000  2$:    MOV     R2,R3      ;CHAIN FORWARD TO NEXT USER IN LIST
43         004524  116302  0000000  MOVVB   LQLINK(R3),R2
44         004530  001365          BNE     1$             ;BR IF MORE USERS IN LIST
45         004532  000676          BR      ADQTL         ;ADD US TO TAIL OF LIST

```

DEQ -- Remove user from run queue

```

1          .SBTTL  DEQ  -- Remove user from run queue
2          ;-----
3          ; DEQ is called to remove from the run queue the user whose
4          ; index number is in R1.
5          ; On return, the user will be left unlinked from the run queue
6          ; and his state code (LSTATE) will be zeroed.
7          ; ** The interrupts are left disabled on return **
8          ; All registers are preserved.
9          ;
10         DEQ:  MOV     R2,-(SP)
11             MOV     R3,-(SP)
12             DISABL                ;** DISABLE **
13         004534 010246 00000000  TST     LQLINK(R1)  ; IS USER UNLINKED NOW?
14         004536 010346          BNE     1$                ; BR IF NOT
15         004540 005761 00000000  CMPB   R1,RUNQHD    ; IS USER ONLY ENTRY IN QUEUE?
16         004544 001010          BNE     2$                ; BR IF NOT -- MUST NOT BE IN QUEUE AT ALL
17         ; User is only entry in queue
18         004546 005761 00000000  CLRB   RUNQHD      ; REMOVE FROM QUEUE
19         004552 001010          CLRB   RUNQTL
20         004554 120137 00000000  BR     2$
21         ; Unlink from queue
22         004556 001030          1$:  MOVB  LQLINK+1(R1),R3 ; GET # OF USER IN FRONT OF US IN QUEUE
23         004562 105037 00000000  MOVB  LQLINK(R1),R2  ; GET # OF USER WHO FOLLOWS US IN QUEUE
24         004566 105037 00000001  BNE   4$                ; BR IF NOT AT TAIL OF QUEUE
25         004572 000423          MOVB  R3,RUNQTL      ; MAKE TAIL POINT TO OUR PREDECESSOR
26         004574 116103 00000010  BR    5$
27         004576 116102 00000000  4$:  MOVB  R3,LQLINK+1(R2) ; MAKE OUR SUCCESSOR POINT BACK OVER US
28         004582 005703          5$:  TST   R3                ; ARE WE AT HEAD OF QUEUE?
29         004584 001003          BNE   6$                ; BR IF NOT AT HEAD OF QUEUE
30         004586 110337 00000001  MOVB  R2,RUNQHD    ; MAKE QUEUE HEAD POINT TO OUR SUCCESSOR
31         004592 000402          BR    3$
32         004594 110237 00000000  6$:  MOVB  R2,LQLINK(R3)  ; MAKE OUR PREDECESSOR POINT TO OUR SUCCESSOR
33         ; Finished unlinking.
34         ; Say user is not in any queue
35         004596 005061 00000000  3$:  CLR   LQLINK(R1)    ; CLEAR BOTH OUR FORWARD & BACKWARD LINKS
36         004598 005061 00000000  2$:  CLR   LSTATE(R1)   ; SAY WE HAVE NO EXECUTION STATE
37         004604 012603          MOV   (SP)+,R3
38         004606 012602          MOV   (SP)+,R2
39         004608 000207          RETURN

```

QSRCH -- Look for 1st user with some execution state

```

1          .SBTTL  QSRCH  -- Look for 1st user with some execution state
2          ;-----
3          ; QSRCH is called to locate the highest priority user in a
4          ; certain execution state.
5          ; When called, R0 must contain the execution state code (S#----).
6          ; If a user is found with the state code, the user index number
7          ; is returned in R1 and the C-flag is cleared.
8          ; If no user is found with the specified state, the C-flag is
9          ; set on return.
10         ; ** Interrupts are disabled and left disabled if a user is found
11         ; with the specified state. If no user is found, the interrupts
12         ; are reenabled before returning. **
13         ; All registers are preserved except R1.
14         ;
15 004654  QSRCH:  DISABL                    ; ** DISABLE **
16 004662  113701 000000'  MOVB  RUNQHD,R1      ; GET # OF 1ST USER IN QUEUE
17 004666  001406          BEQ   1$              ; BR IF QUEUE IS EMPTY
18 004670  020061 0000000  3$:  CMP   R0,LSTATE(R1) ; IS THIS USER IN STATE OF INTEREST?
19 004674  001410          BEQ   2$              ; BR IF YES -- SUCCESS
20 004676  116101 0000000  MOVB  LQLINK(R1),R1 ; CHAIN FORWARD
21 004702  001372          BNE   3$              ; BR IF MORE TO CHECK
22         ; No user has the desired state
23 004704  1$:  ENABL                    ; ** ENABLE **
24 004712  000261          SEC                   ; SIGNAL FAILURE
25 004714  000207          RETURN
26         ; Found a user in the desired state
27         ; Leave the interrupts disabled on return
28 004716  000241          2$:  CLC                   ; INDICATE SUCCESS
29 004720  000207          RETURN

```

QNSPND -- Put job in wait state

```

1          .SBTTL  QNSPND -- Put job in wait state
2          ;-----
3          ; There are four routines that can be called to suspend the
4          ; execution of a job: QNSPND, QNSPNX, QHDSPN, QHDSPX.
5          ; All four routines perform the functions of changing the
6          ; job state to a specified wait state and then calling the
7          ; job scheduler to run some other job while the current job is waiting.
8          ; Before placing the job in the specified wait state, these routines
9          ; check to see if there is a pending completion routine for the job.
10         ; If there is a pending completion routine, the job state is changed
11         ; to the state associated with the completion routine to allow the
12         ; completion routine to run before the job is suspended.
13         ; The difference between the four routines is whether the job is
14         ; queued at the head or tail of the wait queue and whether the
15         ; job's time-slice quantum is reset.
16         ;
17         ; QNSPND -- Queue at tail of wait list,  reset quantum.
18         ; QNSPNX -- Queue at tail of wait list,  don't reset quantum.
19         ; QHDSPN -- Queue at head of wait list,  reset quantum.
20         ; QHDSPX -- Queue at head of wait list,  don't reset quantum.
21         ;
22         ; Inputs:
23         ;   RO = Job state to which job is to be set before calling scheduler.
24         ;
25         ; Outputs:
26         ;   Interrupts are enabled.
27         ;
28         ; Queue at tail of wait list, reset quantum.
29         ;
30 004722  010146  QNSPND: MOV     R1, -(SP)
31 004724  004737  005014'  CALL    QCKCPL      ; DO COMMON ENTRY SETUP  ** DISABLE **
32 004730  004737  004424'  CALL    ENQTL       ; PUT JOB IN WAIT QUEUE  ** ENABLE **
33 004734  000421          BR      DQSPNR      ; RESET QUANTUM AND SUSPEND JOB
34         ;
35         ; Queue at tail of wait list, don't reset quantum.
36         ;
37 004736  010146  QNSPNX: MOV     R1, -(SP)
38 004740  004737  005014'  CALL    QCKCPL      ; DO COMMON ENTRY SETUP  ** DISABLE **
39 004744  004737  004424'  CALL    ENQTL       ; PUT JOB IN WAIT QUEUE  ** ENABLE **
40 004750  000415          BR      DQSPNX      ; GO SUSPEND JOB
41         ;
42         ; Queue at head of wait list, don't reset quantum.
43         ;
44 004752  010146  QHDSPX: MOV     R1, -(SP)
45 004754  004737  005014'  CALL    QCKCPL      ; DO COMMON ENTRY SETUP  ** DISABLE **
46 004760  004737  004222'  CALL    ENQHD       ; PUT JOB IN WAIT QUEUE  ** ENABLE **
47 004764  000407          BR      DQSPNX      ; GO SUSPEND JOB
48         ;
49         ; Queue at head of wait list, reset quantum.
50         ;
51 004766  010146  QHDSPN: MOV     R1, -(SP)
52 004770  004737  005014'  CALL    QCKCPL      ; DO COMMON ENTRY SETUP  ** DISABLE **
53 004774  004737  004222'  CALL    ENQHD       ; PUT JOB IN WAIT QUEUE  ** ENABLE **
54 005000  005061  000000G  DQSPNR: CLR     LQUAN(R1) ; RESET JOB TIME-SLICE QUANTUM
55 005004  012601  DQSPNX: MOV     (SP)+, R1
56 005006  004737  000210'  CALL    SCHED       ; CALL JOB SCHEDULER TO RUN ANOTHER JOB
57 005012  000207          RETURN            ; RESUME EXECUTION OF OUR JOB

```

58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76 005014 010246
 77 005016 113701 0000000
 78 005022
 79
 80
 81
 82 005030 016102 0000000
 83 005034 001413
 84 005036 005761 0000000
 85 005042 001010
 86 005044 105737 0000000
 87 005050 001005
 88 005052 120062 0000000
 89 005056 101402
 90 005060 116200 0000000
 91
 92
 93
 94 005064 012602
 95 005066 000207

```

-----
; QCKCPL is a local subroutine called by the queue-and-suspend
; routines to perform common setup functions.
; The following operations are performed.
; 1) Interrupts are disabled.
; 2) The current job number is loaded into R1.
; 3) If there is a pending completion routine, the priority request
;    for the job is changed to the priority of the completion routine.
;
; Inputs:
;   RO = Requested wait state for job to be placed in.
;
; Outputs:
;   RO = Job state that job is actually to be put in.
;   R1 = Current job index number.
;   Interrupts are left disabled.
;
QCKCPL: MOV     R2, -(SP)
        MOVB   CORUSR, R1      ; Get current job index number
        DISABL                ; ** Disable **
;
; Check for pending completion routines
;
        MOV    LCMPL(R1), R2   ; Are there any pending completion rtns?
        BEQ    9$              ; Br if not
        TST   LIOHLD(R1)      ; Are we holding completion routines?
        BNE   9$              ; Br if yes
        TSTB  CURCP           ; Are we in a completion routine now?
        BNE   9$              ; Br if yes
        CMPB  RO, CQ$RNS(R2)  ; Does compl routine have higher priority?
        BLOS  9$              ; Br if not
        MOVB  CQ$RNS(R2), RO  ; Change to prio as specified by compl routine
;
; Finished
;
9$:     MOV    (SP)+, R2
        RETURN
  
```

```

1
2
3
4
5
6
7
8
9
10 005070 010046
11 005072 026127 0000000 0000000
12 005100 103404
13
14
15
16
17 005102 012700 0000000
18 005106 004737 004424'
19 005112 105237 000004'
20 005116 012600
21 005120 000207
22
23
24
25
26
27
28
29
30
31
32
33 005122 010146
34 005124 113701 0000000
35 005130 032761 0000000 0000000
36 005136 001411
37 005140 120137 000002'
38 005144 001406
39 005146 032761 0000000 0000000
40 005154 001002
41 005156 004737 0000000
42 005162 012601
43 005164 000207

          .SBTTL  FORCEX  -- Force user execution
          -----
; FORCEX is called to force execution of the user whose line
; index number is in R1.
; The user is taken out of any wait state and placed in
; a high priority execution queue.
; All registers are preserved.
;
FORCEX:  MOV     RO,-(SP)
;
; User is not running now.
; Put user in high priority run state
;
          MOV     $#IOFN,RO      ;GET HIGH-PRIORITY STATE CODE
          CALL    ENQTL          ;PUT USER AT TAIL OF QUEUE
1$:      INCB    DOSCHD          ;REQUEST A JOB SCHEDULER CYCLE
          MOV     (SP)+,RO
          RETURN

          .SBTTL  CHKABT  -- Check for abort condition
          -----
; CHKABT IS CALLED TO SEE IF AN ABORT CONDITION
; SUCH AS DOUBLE CTRL-C OR LINE DISCONNECT HAS
; OCCURED.  IF AN ABORT CONDITION IS PENDING
; THE USER IS ABORTED BY JUMPING DIRECTLY TO
; STOP.  IF NO ABORT CONDITION IS PENDING
; CHKABT RETURNS TO THE CALLING ROUTINE.
; ALL REGISTERS ARE PRESERVED.
;
CHKABT:  MOV     R1,-(SP)
          MOVB    CDRUSR,R1      ;GET USER INDEX NUMBER
          BIT     #<#CTRLC!#DISCN>,LSW(R1);ABORT PENDING?
          BEQ     1$             ;BRANCH IF NOT
          CMPB   R1,USRJOB       ;ARE WE CURRENTLY DOING A DIRECTORY OPERATION?
          BEQ     1$             ;IF YES DON'T ABORT NOW
          BIT     ##NOABT,LSW9(R1);Is the No-abort flag set for job?
          BNE    1$             ;Br if yes -- Don't abort now
2$:      CALL    STOP            ;ABORT THE USER
1$:      MOV     (SP)+,R1
          RETURN
  
```

UREGO -- Restart user at head of wait queue

```

1          .SBTTL  UREGO  -- Restart user at head of wait queue
2          ;-----
3          ;  UREGO is called to restart the user who
4          ;  is at the head of the wait queue whose state code is in RO.
5          ;  This user is removed from the wait queue and
6          ;  added to the tail of the S$IOFN queue.
7          ;  All registers are preserved.
8          ;  Interrupts are enabled on return.
9          ;
10         005166  010146          UREGO:  MOV     R1,-(SP)
11         005170  004737  004654'  CALL    QSRCH      ;FIND USER AT HEAD OF QUEUE * DISABLE *
12         005174  103402          BCS     1$           ;BR IF CAN'T FIND ANY USERS IN THAT STATE
13         005176  004737  005206'  CALL    QHIPRI     ;REQUEUE USER AT TAIL OF S$IOFN QUEUE
14         005202  012601          1$:   MOV     (SP)+,R1
15         005204  000207          RETURN

```

QHIPRI -- Put user in high priority queue

```

1          .SBTTL  QHIPRI -- Put user in high priority queue
2          ;-----
3          ; QHIPRI is called to place the user whose line index number is
4          ; in R1 at the tail of the S#IOFN high priority execution queue.
5          ; All registers are preserved.
6          ;
7 005206 010046 QHIPRI: MOV      RO,-(SP)
8          ;
9          ; If this is an interactive job doing I/O, put the job in the S#HICP
10         ; state.
11         ;
12 005210 005761 0000000  TST      LITIME(R1)      ; Is this job interactive or compute bound?
13 005214 001426         BEQ      3$              ; Br if compute bound
14 005216 005761 0000000  TST      LHIPCT(R1)     ; Used up all allowed I/O ops for interactive?
15 005222 001407         BEQ      4$              ; Br if yes -- no longer interactive
16 005224 005361 0000000  DEC      LHIPCT(R1)     ; Decrease remaining number of interactive I/O
17 005230 012700 0000000  MOV      #S#HICP,RO     ; Get interactive completion state code
18 005234 004737 004424'  CALL    ENQTL          ; Put job at tail of that queue
19 005240 000433         BR      2$
20         ;
21         ; Interactive job has performed maximum number of I/O operations while
22         ; in interactive state.  Reclassify the job as non-interactive.
23         ;
24 005242 005061 0000000 4$:  CLR      LITIME(R1)      ; Say job is no longer interactive
25 005246 004037 010040'  JSR      RO,QUNSIG      ; Signal that INTIOC has been used up
26 005252 0000000        .WORD    $SGIIO
27 005254 013700 0000000  MOV      VHIPCT,RO      ; Get # high-prio boosts allowed for CPU jobs
28 005260 163700 0000000  SUB      VINTIO,RO      ; Any remaining after number already used?
29 005264 003410         BLE      5$              ; Br if not
30 005266 010061 0000000  MOV      RO,LHIPCT(R1)  ; Set number of remaining high-prio boosts
31         ;
32         ; Job is compute bound.
33         ; Put job in I/O complete state or CPU state.
34         ;
35 005272 005761 0000000 3$:  TST      LHIPCT(R1)     ; HAS JOB USED UP ALL OF ITS HIGH-PRIO HITS?
36 005276 001006         BNE      1$              ; BR IF NOT
37 005300 004037 010040'  JSR      RO,QUNSIG      ; SIGNAL THAT HIPRCT WAS USED UP
38 005304 0000000        .WORD    $SGHIO
39 005306 004737 005334' 5$:  CALL    QCPU           ; QUEUE AS CPU-BOUND JOB IF YES
40 005312 000406         BR      2$
41 005314 005361 0000000 1$:  DEC      LHIPCT(R1)     ; ONE LESS HIGH-PRIORITY HIT REMAINING
42 005320 012700 0000000  MOV      #S#IOFN,RO     ; GET STATE CODE
43 005324 004737 004424'  CALL    ENQTL          ; PUT USER AT TAIL OF QUEUE
44 005330 012600        2$:  MOV      (SP)+,RO
45 005332 000207         RETURN

```

QCPU -- Place job in CPU-bound run queue

```

1          .SBTTL  QCPU  -- Place job in CPU-bound run queue
2          ;-----
3          ; QCPU is called to change the run-state of a job to be
4          ; compute bound.
5          ;
6          ; Inputs:
7          ; R1 = Index number of job to be affected.
8          ; Job's time quantum is reinitialized.
9          ;
10         005334  012700  0000000  QCPU:  MOV      #S$CPU,R0      ;PUT JOB IN CPU-BOUND EXECUTION QUEUE
11         005340  004737  004424'   CALL    ENQTL      ;REQUEUE JOB AT TAIL OF THAT QUEUE
12         005344  005061  0000000   CLR     LQUAN(R1)   ;REINITIALIZE JOB'S TIME QUANTUM
13         005350  013761  0000000  0000000  MOV     VHIPCT,LHIPCT(R1);REINIT NUMBER OF HIGH-PRIO HITS FOR JOB
14         005356  000207              RETURN

```

GTSYMB -- Get system message buffer

```

1          .SBTTL  GTSYMB -- Get system message buffer
2          ;-----
3          ; GTSYMB is called to get a system message buffer block.
4          ;
5          ; Outputs:
6          ; R4 = Address of message block acquired.
7          ; C-flag set if no free message blocks are available.
8          ;
9 005360   GTSYMB: DISABL          ;** DISABLE **
10 005366  013704 0000000   MOV     SNMSHD,R4      ;GET ADDRESS OF 1ST FREE MESSABE BLOCK
11 005372  001417          BEQ     1$              ;BR IF NO FREE BLOCKS
12          ;
13          ; Got a free message block.  Unlink from free list.
14          ;
15 005374  016437 0000000 0000000   MOV     SB$LNK(R4),SNMSHD;REMOVE BLOCK FROM FREE LIST
16 005402  005337 0000000          DEC     NMUMB          ;DECREASE # FREE BLOCKS
17 005406          ENABL          ;** ENABLE **
18          ;
19          ; Initialize pointer into text area of buffer.
20          ;
21 005414  010400          MOV     R4,R0          ;GET ADDRESS OF BUFFER
22 005416  062700 0000000   ADD     #SB$TXT,R0      ;POINT TO TEXT STORAGE AREA IN BUFFER
23 005422  010064 0000000   MOV     R0,SB$PNT(R4)  ;SET POINTER TO TEXT AREA
24 005426  000241          CLC          ;SIGNAL SUCCESS ON RETURN
25 005430  000404          BR     2$
26          ;
27          ; No free message blocks.
28          ;
29 005432   1$:  ENABL          ;** ENABLE **
30 005440  000261          SEC          ;SIGNAL FAILURE ON RETURN
31 005442  000207   2$:  RETURN

```

TSXTRP -- Catch traps

```

1          .SBTTL  TSXTRP -- Catch traps
2          ;-----
3          ; TSXT4 and TSXT10 catch traps to 4 and 10 respectively.
4          ; If the user did a .TRPSET his routine is entered.
5          ; Otherwise the job is aborted.
6          ;
7          ;
8          ; Trap to 250 (Memory management trap)
9          ;
10         005444  012737  0000000 0000000 TRP250: MOV      #MMENBL,@#SR0MMR; RESET ERROR FLAGS
11         ; Treat trap to 250 like trap to 4.
12         ;
13         ; Trap to 4.
14         ;
15         005452  010446          TRP4:  MOV      R4,-(SP)
16         005454  113704  0000000          MOVVB   CORUSR,R4      ;Get current job index number
17         005460  032764  0000000 0000000          BIT     #$GEMAR,LSW11(R4);Are we accessing user's argument block?
18         005466  001410          BEQ     1$          ;Br if not
19         005470  012604          MOV     (SP)+,R4     ;Pop R4
20         005472  011646          MOV     (SP),-(SP)   ;Move down PC
21         005474  016666  0000004 0000002          MOV     4(SP),2(SP)  ;Move down PS
22         005502  005066  0000004          CLR     4(SP)       ;Store 0 value to be returned
23         005506  000002          RTI     ;Return following MFPD with 0 on stack
24         005510  010546          1$:   MOV     R5,-(SP)
25         005512  013737  0000000 000034'          MOV     @#KPAR5,TRPAR5 ;SAVE KPAR5 FROM TRAP FOR ERROR HANDLING
26         005520  013737  120002  0000000          MOV     @#120002,ABRTOV ;SAVE RAD50 OVERLAY NAME
27         005526  012705  0000001          MOV     #1,R5       ;ERROR CODE FOR TRAP 4
28         005532  000137  0000000          JMP     TSXTX       ;ENTER TRAP HANDLER IN TSX OVERLAY
29         ;
30         ; Trap to 10
31         ;
32         005536  010446          TRP10: MOV     R4,-(SP)
33         005540  010546          MOV     R5,-(SP)
34         005542  013737  0000000 000034'          MOV     @#KPAR5,TRPAR5 ;SAVE KPAR5 FROM TRAP FOR ERROR HANDLING
35         005550  013737  120002  0000000          MOV     @#120002,ABRTOV ;SAVE RAD50 OVERLAY NAME
36         005556  012705  0000002          MOV     #2,R5       ;ERROR CODE FOR TRAP 10
37         005562  000137  0000000          JMP     TSXTX       ;ENTER TRAP HANDLER IN TSX OVERLAY
38         ;
39         ; Trap to 14 (Breakpoint trap)
40         ;
41         005566  032766  0000000 0000002 TRP14: BIT     #UMODE,2(SP) ;DID BREAKPOINT OCCUR IN USER OR KERNEL MODE?
42         005574  001002          BNE     1$          ;BR IF USER MODE
43         ; Breakpoint occurred in kernel mode.
44         ; Give control to system ODT.
45         005576  000177  172226          JMP     @ODTTRP     ;ENTER SYSTEM DEBUGGER
46         ; Breakpoint occurred in user mode.
47         ; Give control to user's debugger program.
48         005602  010446          1$:   MOV     R4,-(SP)
49         005604  113704  0000000          MOVVB   CORUSR,R4      ;Get current job index number
50         005610  032764  0000000 0000000          BIT     #$DEBUG,LSW9(R4);Is program being run with TSX debugger?
51         005616  001403          BEQ     2$          ;Br if not
52         ; Enter TSX-Plus debugger
53         005620  012604          MOV     (SP)+,R4
54         005622  000137  0000000          JMP     BRKENT       ;Enter TSX-Plus debugger
55         ; Enter user's debugger
56         005626  000137  0000000          2$:   JMP     TRPBPT     ;ENTER TRAP HANDLING ROUTINE IN TSX OVERLAY
57         ;

```

TSXTRP -- Catch traps

```

58          ; Trap to 20 (IOT)
59          ;
60 005632 010446 TRP20: MOV      R4,-(SP)
61 005634 010546      MOV      R5,-(SP)
62 005636 012705 000013      MOV      #13,R5      ;GET ERROR CODE
63 005642 012704 000020      MOV      #20,R4      ;GET TRAP LOCATION
64 005646 000137 00000000      JMP      TRPCOM      ;ENTER TRAP HANDLING ROUTINE IN TSX OVERLAY
65          ;
66          ; Trap to 24 (Power fail)
67          ;
68 005652 TRP24: DIE      #EM#PFT      ;SYSTEM HALT IF POWER FAIL TRAP
69          ;
70          ; Trap to 34 (TRAP instruction)
71          ;
72 005664 010446 TRP34: MOV      R4,-(SP)
73 005666 010546      MOV      R5,-(SP)
74 005670 105737 00000000      TSTB     DOTRMP      ;Using TRAP instruction for mapping?
75 005674 001017      BNE      TRPMAP      ;Br if yes
76 005676 012705 000015      MOV      #15,R5      ;GET ERROR CODE
77 005702 012704 000034      MOV      #34,R4      ;GET TRAP LOCATION
78 005706 000137 00000000      JMP      TRPCOM      ;ENTER TRAP HANDLING ROUTINE IN TSX OVERLAY
79          ;
80          ; Trap to 244 (Floating point exception interrupt).
81          ; Set #FPUEX flag and return through SYSXIT which will do actual
82          ; FPU exception processing when we are about to return to user mode.
83          ;
84 005712 004537 006404' FPTRAP: JSR      R5,INTEN      ;Standard interrupt entry
85 005716 000000      .WORD     0          ;Run at priority 7
86 005720 113704 00000000      MOVB     CORUSR,R4      ;Get current job index number
87 005724 052764 00000000 00000000      BIS      #FPUEX,LSW(R4) ;Set flag for job saying FPU interrupt
88 005732 000207      RETURN      ;Return and perform FPU exception code

```

TRPMAP -- High-performance memory mapping service

```

1          .SBTTL  TRPMAP -- High-performance memory mapping service
2          ;-----
3          ; This routine is jumped to when a TRAP instruction is executed and we
4          ; are are doing high-performance memory mapping.
5          ;
6          ; Inputs:
7          ;   RO = Mapping region index number.
8          ;
9          ; Stack:
10         ;   (SP) = R5
11         ;   2(SP) = R4
12         ;   4(SP) = PC
13         ;   6(SP) = PS
14         ;
15 005734  TRPMAP:
16         ;
17         ; Clear C-flag in PS
18         ;
19 005734  042766  0000000 0000006      BIC      #CFLAG,6(SP)      ;Clear C-flag in PS on stack
20         ;
21         ; Make sure the region index number is valid
22         ;
23 005742  020027  0000000      CMP      RO,#MAXSRD      ;Is region index valid?
24 005746  103404      BLO      1$              ;Br if ok
25 005750  052766  0000000 0000006      BIS      #CFLAG,6(SP)      ;Set carry flag for return
26 005756  000423      BR       9$              ;
27         ;
28         ; Get the PAR index
29         ;
30 005760  116005  0000000 1$:      MOVB     SR$PX(RO),R5      ;Get PAR index number
31         ;
32         ; Load the PAR value
33         ;
34 005764  006300      ASL      RO              ;Get word table index
35 005766  016004  0000000      MOV      SR$PAR(RO),R4      ;Get value to load into PAR registers
36 005772  010465  0000000      MOV      R4,RPAR(R5)      ;Shared run-time mapping for this PAR
37 005776  010465  0000000      MOV      R4,CUPARO(R5)    ;Set PAR value in context block
38 006002  010465  0000000      MOV      R4,UPARO(R5)    ;Set PAR value in hardware register
39         ;
40         ; Load the PDR value
41         ;
42 006006  016004  0000000      MOV      SR$PDR(RO),R4      ;Get value to load into PDR registers
43 006012  010465  0000000      MOV      R4,RPDR(R5)      ;Shared run-time mapping for this PDR
44 006016  010465  0000000      MOV      R4,CUPDRO(R5)    ;Set PDR value in context block
45 006022  010465  0000000      MOV      R4,UPDRO(R5)    ;Set PDR value in hardware register
46         ;
47         ; Finished
48         ;
49 006026  012605  9$:      MOV      (SP)+,R5
50 006030  012604      MOV      (SP)+,R4
51 006032  000002      RTI                      ;Return from TRAP instruction

```

```

1          .SBTTL  UEXINT -- Unexpected interrupt
2          ;-----
3          ; An interrupt occurred at an unexpected location.
4          ; On entry to UEXINT the interrupt vector address is encoded
5          ; in the PS that was set by the interrupt vector.
6          ; The address has the two low-order bits removed (they are assumed
7          ; to be zero) and the remainder of the address stored in the
8          ; PS fields priority and n-z-v-c (note the T field is not used).
9          ;
10         006034 013703 0000000 UEXINT: MOV    @#PSW,R3    ;GET CURRENT PROCESSOR STATUS VALUE
11         006040 006303          ASL    R3          ;ADD ONE LOW-ORDER ZERO BIT
12         006042 010302          MOV    R3,R2        ;COPY VALUE
13         006044 042703 177077    BIC    #^C700,R3    ;MASK OUT ALL BUT PRIO FIELD (SHIFTED)
14         006050 042702 177741    BIC    #^C36,R2     ;MASK OUT ALL BUT N-Z-V-C FIELDS (SHIFTED)
15         006054 006302          ASL    R2          ;ALIGN LOW-ORDER FIELD WITH HIGH-ORDER
16         006056 050203          BIS    R2,R3        ;COMBINE LOW- AND HIGH-ORDER FIELDS
17         006060          DIE    #EM#UEI,R3    ;SYSTEM CRASH -- ARG VALUE = INT LOCATION
18         ;
19         ; Enter at UEXRTN if we should ignore unexpected interrupts
20         ;
21         006076 000002 UEXRTN: RTI          ;Return from interrupt -- Ignore it
22         ;
23         ; Memory parity error
24         ;
25         006100          MEMPAR: DIE    #EM#MPR    ;MEMORY PARITY ERROR
26         ;
27         ; Jump occurred to location 0
28         ;
29         006112          JMPO:  DIE    #EM#JMO    ;FATAL SYSTEM HALT

```

UEXINT -- Unexpected interrupt

```

1 ;-----
2 ;   CHKUSP is called to determine if the current user-mode stack pointer (SP)
3 ;   is valid.  The SP is checked to make sure it is even and >400.
4 ;   CKUSP2 is an alternate entry point that is a little faster if it is
5 ;   already known that the previous mode was user.
6 ;
7 ;   Outputs:
8 ;   C-flag cleared if SP is valid.  Set if invalid.
9 ;   R5 = abort code for invalid stack if error detected, otherwise unaltered.
10 ;
11 006124  CHKUSP:
12 ;
13 ;   Return with carry cleared if previous mode is not user
14 ;
15 006124  032737  0000000 0000000      BIT      #UPMODE,@#PSW      ;PREVIOUS MODE = USER?
16 006132  001002                BNE      CKUSP2          ;BR IF YES
17 006134  000241                CLC                    ;CLEAR CARRY FOR RETURN
18 006136  000207                RETURN
19 ;
20 ;   Get the user mode SP
21 ;
22 006140  106506  CKUSP2: MFPD      SP          ;GET USER-MODE SP
23 ;
24 ;   Make sure the stack is in the right range
25 ;
26 006142  021627  000400      CMP      (SP),#400      ;DID A STACK OVERFLOW OCCUR?
27 006146  103405                BLO      2$           ;BR IF YES
28 006150  021637  0000000      CMP      (SP),UHIMEM   ;IS STACK ADDRESS TOO HIGH?
29 006154  103002                BHS      2$           ;BR IF TOO HIGH
30 ;
31 ;   Make sure the stack address is even
32 ;
33 006156  006016                ROR      (SP)          ;IS THE STACK ADDRESS EVEN?
34 006160  103003                BCC      3$           ;BR IF EVEN -- OK
35 ;
36 ;   User's stack pointer is invalid
37 ;
38 006162  012705  000011  2$:      MOV      #11,R5      ;LOAD ABORT ERROR CODE VALUE
39 006166  000261                SEC                    ;SIGNAL ERROR ON RETURN
40 006170  005226  3$:      INC      (SP)+      ;CLEAN OFF STACK (DON'T ALTER C-FLAG)
41 ;
42 ;   Finished
43 ;
44 006172  000207  9$:      RETURN

```

CLKINT -- Clock interrupt routine

```

1          .SBTTL  CLKINT -- Clock interrupt routine
2          ;-----
3          ; CLKINT is the interrupt service routine for clock interrupts.
4          ; It is entered directly from the interrupt (priority = 7).
5          ;
6 006174   CLKINT:
7          ;
8          ; If this is a PRO-350, access the CSR2 clock register to reenable
9          ; the interrupt.
10         ;
11 006174   105737   0000000   TSTB   PROFLG   ; Is this a PRO-350?
12 006200   001411   BEQ     1$      ; Br if not
13 006202   005737   0000000   TST    @#PCCR2   ; Acknowledge the interrupt
14         ;
15         ; Ignore every 16'th clock tick on a PRO-350 so that the effective
16         ; clock rate will be 60 Hz.
17         ;
18 006206   105337   000014'   DECB   PROSKP   ; Is this the 16'th tick?
19 006212   001004   BNE    1$      ; Br if not
20 006214   112737   000020   000014'   MOVB   #16.,PROSKP ; Reset the counter
21 006222   000420   BR     CLKRTI   ; Ignore this clock tick
22         ;
23         ; Count another clock tick
24         ;
25 006224   005237   000100'   1$:   INC    TIKCNT   ; ANOTHER TICK HAS OCCURED
26 006230   003015   BGT    CLKRTI   ; BR IF STILL PROCESSING LAST TICK
27         ;
28         ; We are not reentering the clock processing routine.
29         ; Save interrupted PC & PS for performance monitor to use.
30         ;
31 006232   011637   000102'   MOV    (SP),CLKPC ; INTERRUPTED PC
32 006236   016637   000002   000104'   MOV    2(SP),CLKPS ; INTERRUPTED PS
33         ;
34         ; Drop priority to 6 then fork.
35         ;
36 006244   004537   006404'   JSR    R5,INTEN   ; DROP RUNNING PRIORITY TO 6
37 006250   000040   .WORD  40        ; MASK TO SET PRIO TO 6
38 006252   004537   007332'   JSR    R5,FORK    ; NOW FORK TO GET TO PRIORITY 0
39 006256   0000000   .WORD  FP$CKT    ; Specify fork priority
40         ;
41         ; CLKRUN is entered to perform clock servicing in the system mapped region.
42         ;
43 006260   000137   0000000   JMP    CLKRUN     ; ENTER THE SYSTEM MAPPED REGION
44         ;
45         ; The clock processing routine is still running from the last tick.
46         ; Don't reenter it.
47         ;
48 006264   000002   CLKRTI: RTI      ; RETURN FROM INTERRUPT QUICKLY
49

```

ENSYS -- Enter system state

```

1          .SBTTL  ENSYS  -- Enter system state
2          ;-----
3          ; ENSYS is called to enter system state.  What this consists of is switching
4          ; to the interrupt stack and saving the kernel PAR6 value.
5          ; Basically, what we do is fake an interrupt and then do a .INTEN and .FORK.
6          ; On return from ENSYS we are running in system state at fork level using
7          ; the interrupt stack.
8          ; To exit from system state, do a RETURN.
9          ;
10         ; The form of the call to ENSYS is:
11         ;
12         ;     MOV     #return_address,R0      ;Get return address
13         ;     CALL   ENSYS                    ;Enter system state
14         ;     .WORD  fork_priority          ;Fork priority level to run at
15         ;
16         ; Inputs:
17         ;     R0 = Address of routine to be jumped to when a RETURN is done to exit
18         ;         from system state.
19         ;
20         ; Outputs:
21         ;     R0, R4 and R5 are destroyed.
22         ;     On return we are in system state running on the interrupt stack.
23         ;     All registers except R0 are preserved across the ENSYS.
24         ;
25 006266  012604  ENSYS:  MOV     (SP)+,R4      ;Get return address to R4
26         ;
27         ; Check to see if we are already running in system state.
28         ;
29 006270  105737  000010'  TSTB   STKLVL      ;Are we already in system state?
30 006274  002017      BGE     10$         ;Br if already in system state
31         ;
32         ; We are not currently in system state.
33         ; Put PC & PS on the stack to make it look like an interrupt occurred.
34         ;
35 006276  013746  0000000  MOV     @#PSW,-(SP) ;PS
36 006302  010046      MOV     R0,-(SP)    ;PC -- Return here when we exit system state
37         ;
38         ; Do .INTEN to enter system state.
39         ;
40 006304      DISABL      ;** Disable interrupts **
41 006312  004537  006404'  JSR     R5,INTEN    ;Enter system state
42 006316  000000      .WORD  0           ;Priority = 7
43         ;
44         ; We are now running in system state on the interrupt stack.
45         ; The processor priority level is 7.
46         ; Now do a .FORK so that we will not hold out interrupts.
47         ;
48 006320  012437  006330'  MOV     (R4)+,1$    ;Set fork priority
49 006324  004537  007332'  JSR     R5,FORK     ;Do a fork
50 006330  000000  1$:     .WORD  0     ;Fork priority is stored here
51         ;
52         ; We are now running in system state, fork level.
53         ; Return to caller in system state.
54         ; Caller should do a RETURN to exit from system state.
55         ;
56 006332  000114  3$:     JMP     (R4)     ;Call calling routine in system state
57         ;

```

ENSYS -- Enter system state

```

58          ; ENSYS was called while already running in system state.
59          ; Save context and set up stack so we will restore it on return.
60          ;
61 006334 010046 10$:  MOV     R0,-(SP)      ;Set ultimate return address
62 006336 010146      MOV     R1,-(SP)
63 006340 010246      MOV     R2,-(SP)
64 006342 010346      MOV     R3,-(SP)
65 006344 013746 0000000  MOV     @#KPAR6,-(SP)  ;Save kernel PAR 6
66 006350 013746 0000000  MOV     @#KPAR5,-(SP)  ;Save kernel PAR 5
67 006354 012746 006364'  MOV     #11$,-(SP)    ;Set address of routine for ENSYS exit
68 006360 000164 0000002  JMP     2(R4)          ;Enter user's routine
69          ;
70          ; Finished with routine in system state.
71          ; Drop down a level.
72          ;
73 006364 012637 0000000 11$:  MOV     (SP)+,@#KPAR5 ;Restore kernel PAR 5
74 006370 012637 0000000      MOV     (SP)+,@#KPAR6 ;Restore kernel PAR 6
75 006374 012603      MOV     (SP)+,R3
76 006376 012602      MOV     (SP)+,R2
77 006400 012601      MOV     (SP)+,R1
78 006402 000207      RETURN          ;Return

```

INTEN -- Interrupt entry processing

```

1          .SBTTL  INTEN  -- Interrupt entry processing
2
3          ;-----
4          ; INTEN performs the RT-11 .INTEN function which is used to begin
5          ; interrupt processing.  The form of the call to INTEN is:
6
7          ;       JSR      R5,INTEN
8          ;       .WORD   <^CPriority to run at>&340
9
10         ; INTEN switches to the TSX interrupt stack and then calls the calling
11         ; routine back as a coroutine.
12         ; When the interrupt processing task completes, it exits back to
13         ; INTEN by doing an RTS PC.
14         ; Before returning from interrupt processing INTEN calls any routines
15         ; queued as a result of .FORK requests and also may call the job scheduler
16         ; if any job scheduling event occurred during interrupt processing.
17
18         ; There are three "level indicators" that indicate the processing state.
19         ; INTLVL indicates the hardware interrupt level.
20         ; STKLVL indicates if we are running on the interrupt stack.
21         ; FRKPRI indicates the current fork processing priority.
22         ; The initial (non-interrupt, non-fork) value for all three is -1.
23         ; If INTLVL >= 0 we are in an interrupt routine.
24         ; If INTLVL < 0 we are not at interrupt level but may be at fork level.
25         ; If FRKPRI > 0 we are at fork level.
26 006404  010446  INTEN:  MOV      R4,-(SP)      ;R5 is already on stack, save R4 too.
27
28         ; We should already be running at processor priority level 7.
29         ; However, make sure we are at level 7.
30
31 006406          ;       DISABL          ;;;** Disable interrupts **
32
33         ; Increment interrupt level counter.
34
35 006414  105237  000007'  ;       INCB   INTLVL          ;;;Increment interrupt level counter
36 006420  105237  000010'  ;       INCB   STKLVL         ;;;Are we already running on interrupt stack?
37 006424  003004          ;       BGT    1$             ;;;Br if yes - Interrupting another int/fork
38
39         ; We were at level 0 when the interrupt occurred.
40         ; Save user's stack pointer and switch to TSX interrupt stack.
41
42 006426  010637  000120'  ;       MOV    SP,USP         ;;;Save user's stack pointer
43 006432  013706  000022'  ;       MOV    INTSTK,SP      ;;;Switch to interrupt stack
44
45         ; Drop running priority to that requested by caller.
46
47 006436  013746  000122'  1$:   MOV    INTPRI,-(SP)      ;;;Save current running priority
48 006442  113746  000011'  ;       MOVB   FRKPRI,-(SP)   ;;;Save current fork level priority
49 006446  112737  000000  000011' ;       MOVB   #0,FRKPRI     ;;;Not at fork level (preserve C-bit)
50 006454  013746  00000000 ;       MOV    @#KPAR6,-(SP)  ;;;Save kernel-mode PAR6 register
51 006460  013746  00000000 ;       MOV    @#KPAR5,-(SP)  ;;;Save kernel-mode PAR5 register
52 006464  011537  000122'  ;       MOV    (R5),INTPRI    ;;;Set new priority
53 006470  042537  00000000 ;       BIC    (R5)+,@#PSW    ;Drop running priority
54
55         ; We are now running at the requested priority.
56         ; Call our caller back as a coroutine.
57

```

58 006474 004710
59 006476

CALL @R5
INTENX:

;Call caller as a coroutine
;Must immediately follow CALL @R5

INTEN -- Interrupt entry processing

```

1      ;
2      ;   Interrupt processing routine is finished.
3      ;
4      ;   See if we are returning to level 0 or to a lower level interrupt routine.
5      ;
6 006476 INTXIT: DISABL      ;** Disable interrupts **
7 006504 012637 0000000  MOV      (SP)+,@#KPAR5 ;Restore kernel-mode PAR5 register
8 006510 012637 0000000  MOV      (SP)+,@#KPAR6 ;Restore kernel-mode PAR6 register
9 006514 112637 000011'  MOVB    (SP)+,FRKPRI ;Restore fork processing priority
10 006520 012637 000122'  MOV      (SP)+,INTPRI ;Reset interrupt priority
11 006524 105337 000007'  DECB    INTLVL      ;Are we returning to level 0?
12 006530 002405          BLT      1$          ;Br if yes
13      ;
14      ;   We are about to return to a lower-level interrupt.
15      ;   We go back to lower level interrupt routines before we check
16      ;   for pending fork requests. This is done to give all interrupt routines
17      ;   priority over all fork routines.
18      ;
19 006532 105337 000010'  DECB    STKLVL      ;We are going down one level on the stack
20 006536 012604          MOV      (SP)+,R4
21 006540 012605          MOV      (SP)+,R5
22 006542 000002          RTI          ;Continue processing lower-priority interrupt
23      ;
24      ;   We are returning to level 0.
25      ;
26      ;   See if there are any pending fork queue requests.
27      ;
28 006544 010346          1$:   MOV      R3,-(SP)
29 006546          2$:   DISABL      ;** Disable interrupts **
30 006554 013703 000124'  MOV      FRKCQE,R3 ;Are there any pending fork requests?
31 006560 001515          BEQ      6$          ;Br if not
32 006562 126337 0000000 000011'  CMPB    FQ$PRI(R3),FRKPRI;Is pending request higher prio than current?
33 006570 101511          BLOS    6$          ;Br if not
34      ;
35      ;   There is a fork request that needs to be processed
36      ;
37 006572 113746 000011'  MOVB    FRKPRI,-(SP) ;Save current fork priority
38 006576 116337 0000000 000011'  MOVB    FQ$PRI(R3),FRKPRI ;Set current fork priority
39 006604 016337 0000000 000124'  MOV      FQ$LNK(R3),FRKCQE ;Remove fork block from pending list
40 006612          ENABL      ;** Enable interrupts **
41      ;
42      ;   Save current context before entering the fork routine
43      ;
44 006620 013746 000132'  MOV      CURFRK,-(SP) ;Address of currently running fork routine
45 006624 010046          MOV      R0,-(SP)
46 006626 010146          MOV      R1,-(SP)
47 006630 010246          MOV      R2,-(SP)
48 006632 013746 0000000  MOV      @#KPAR6,-(SP) ;Save kernel-mode PAR6 register
49 006636 013746 0000000  MOV      @#KPAR5,-(SP) ;Save kernel-mode PAR5 register
50      ;
51      ;   Return fork request block to the free list
52      ;
53 006642          DISABL      ;** Disable interrupts **
54 006650 013763 0000000 0000000  MOV      FREFRK,FQ$LNK(R3);Put fork block back on free list
55 006656 010337 0000000  MOV      R3,FREFRK
56      ;
57      ;   See if fork request has been cancelled

```

INTEN -- Interrupt entry processing

```

58
59 006662 016337 0000000 0000000      MOV      FQ$PA5(R3),@#KPAR5 ;Set mapping for kernel PAR 5
60 006670 016337 0000000 0000000      MOV      FQ$PA6(R3),@#KPAR6 ;Set mapping for kernel PAR 6
61 006676 016304 0000000      MOV      FQ$UFB(R3),R4      ;Get address of FQ$LNK in user's fork block
62 006702 001403      BEQ      7$                ;Br if user did not specify a fork block
63 006704 005024      CLR      (R4)+             ;Say fork request has been processed
64 006706 005714      TST      (R4)              ;Has fork request been cancelled? (FQ$RTN)
65 006710 001422      BEQ      8$                ;Br if cancelled
66
67 ; Set up context based on information in fork block
68
69 006712 016304 0000000 7$:      MOV      FQ$R4(R3),R4      ;Restore R4 & R5 for fork routine
70 006716 016305 0000000      MOV      FQ$R5(R3),R5
71 006722 016302 0000000      MOV      FQ$R2(R3),R2
72 006726 016301 0000000      MOV      FQ$R1(R3),R1
73 006732 016300 0000000      MOV      FQ$RTN(R3),R0     ;Get address of fork routine to be called
74 006736 010037 000132'      MOV      R0,CURFRK         ;Remember address of current fork routine
75 006742 016303 0000000      MOV      FQ$R3(R3),R3
76 006746      ENABL                    ;** Enable interrupts **
77
78 ; Call the fork routine
79
80 006754 004710      CALL     @R0               ;Call routine at fork level
81
82 ; Restore context
83
84 006756 8$:      DISABL                    ;Make sure interrupts are disabled
85 006764 012637 0000000      MOV      (SP)+,@#KPAR5     ;Restore kernel-mode PAR5 register
86 006770 012637 0000000      MOV      (SP)+,@#KPAR6     ;Restore kernel-mode PAR6 register
87 006774 012602      MOV      (SP)+,R2
88 006776 012601      MOV      (SP)+,R1
89 007000 012600      MOV      (SP)+,R0
90 007002 012637 000132'      MOV      (SP)+,CURFRK      ;Address of currently running fork routine
91 007006 112637 000011'      MOVB     (SP)+,FRKPRI      ;Restore fork priority
92
93 ; See if there are more pending fork requests
94
95 007012 000655      BR       2$                ;See if there are more fork requests to do
96
97 ; We have processed all fork queue requests.
98
99 007014 012603 6$:      MOV      (SP)+,R3
100
101 ; Switch back to user's stack.
102
103 007016 105337 000010'      DECB     STKLVL            ;Going down one level on interrupt stack
104 007022 002002      BGE     9$                ;Br if still more levels on int stack
105 007024 013706 000120'      MOV      USP,SP           ;Switch back to user's stack
106
107 ; Completed interrupt processing
108
109 007030 012604 9$:      MOV      (SP)+,R4
110 007032 012605      MOV      (SP)+,R5         ;R5 was saved by JSR R5,INTEN

```

INTEN -- Interrupt entry processing

```

1
2 ; At this point we are about to do an RTI to return from an interrupt
3 ; or from an EMT. We are running on the user's stack in his context block
4 ; and all of his registers are intact.
5 ;
6 007034 032766 0000000 000002 SYSXIT: BIT #UMODE,2(SP) ;Are we about to return to user mode?
7 007042 001502 BEQ DORTI ;Br if not
8 007044 105737 000010' TSTB STKLVL ;Are we running on system stack?
9 007050 002077 BGE DORTI ;Br if running on system stack
10 ;
11 ; We are about to return to user mode.
12 ;
13 007052 ENABL ;** Enable **
14 007060 010146 MOV R1,-(SP)
15 007062 113701 0000000 MOVB CORUSR,R1 ;Get job index number
16 ;
17 ; See if a Floating Point Unit (FPU) exception interrupt occurred.
18 ;
19 007066 032761 0000000 0000000 BIT ##FPUEX,LSW(R1) ;Did a FPU exception interrupt occur?
20 007074 001402 BEQ 3$ ;Br if not
21 007076 000137 0000000 JMP FPTRPX ;Do FPU exception processing
22 ;
23 ; See if user typed ctrl-D to force entry to the debugger.
24 ;
25 007102 032761 0000000 0000000 3$: BIT ##DBGBK,LSW9(R1);Does user want to force a breakpoint?
26 007110 001413 BEQ 2$ ;Br if not
27 007112 032737 0000000 0000000 BIT #PO$DBG,PRIVCO ;Is user authorized to use debugger?
28 007120 001004 BNE 4$ ;Br if yes
29 007122 042761 0000000 0000000 BIC ##DBGBK,LSW9(R1);Clear effect of ctrl-D
30 007130 000403 BR 2$ ;Don't enter debugger
31 007132 012601 4$: MOV (SP)+,R1
32 007134 000137 0000000 JMP DBGBRK ;Enter debugger
33 ;
34 ; See if a job scheduler cycle was requested.
35 ;
36 007140 012601 2$: MOV (SP)+,R1
37 007142 105737 000004' TSTB DOSCHD ;DO WE NEED TO CALL THE JOB SCHEDULER?
38 007146 001406 BEQ 1$ ;BR IF NOT
39 ;
40 ; We need to call the job scheduler.
41 ;
42 007150 004737 005122' CALL CHKABT ;SEE IF JOB HAS BEEN ABORTED
43 007154 004737 000210' CALL SCHED ;CALL JOB SCHEDULER
44 007160 004737 005122' CALL CHKABT ;SEE IF WE WERE ABORTED WHILE ASLEEP
45 ;
46 ; See if user did a .SPCPS to alter return address from completion routine.
47 ;
48 007164 005737 0000000 1$: TST SPCPS ;DID USER DO A .SPCPS?
49 007170 001427 BEQ DORTI ;BR IF NOT
50 007172 105737 0000000 TSTB CURCP ;Is user still in a compl routine?
51 007176 001024 BNE DORTI ;DON'T TRIGGER .SPCPS UNTIL EXITING FROM COMPL
52 ;
53 ; User did a .SPCPS -- Set new PC for return.
54 ;
55 007200 010146 MOV R1,-(SP)
56 007202 013701 0000000 MOV SPCPS,R1 ;GET ADDRESS OF USER'S INFORMATION BLOCK
57 007206 005037 0000000 CLR SPCPS ;REMEMBER THAT WE HAVE DONE THE .SPCPS

```

INTEN -- Interrupt entry processing

```

58 007212 052737 0000000 0000000      BIS      #UPMODE,@#PSW      ; MAKE SURE PREVIOUS MODE = USER
59 007220 062701 0000004                ADD      #4,R1              ; POINT TO CELL WHERE OLD PS IS TO BE STORED
60 007224 016646 0000004                MOV      4(SP),-(SP)        ; GET OLD PS VALUE
61 007230 106611                MTPD     (R1)               ; STORE OLD PS IN USER'S INFO BLOCK
62 007232 016646 0000002                MOV      2(SP),-(SP)        ; GET OLD PC VALUE
63 007236 106641                MTPD     -(R1)              ; STORE INTO USER'S INFO BLOCK
64 007240 106541                MFPD     -(R1)              ; GET NEW PC FROM USER'S INFO BLOCK
65 007242 012666 0000002                MOV      (SP)+,2(SP)        ; SET NEW PC FOR RETURN
66 007246 012601                MOV      (SP)+,R1
67                                     ;
68                                     ;   See if a system stack overflow occurred
69                                     ;
70 007250 023727 000600 123456 DORTI:  CMP      SSEND,#123456    ; GENERAL SYSTEM STACK OK?
71 007256 001410                BEQ      1$                 ; BR IF OK
72 007260                DIE      #EM$SOF,#1       ; GENERAL SYSTEM STACK (SS) OVERFLOW
73 007300 027727 170520 123456 1$:  CMP      @INTSND,#123456    ; INTERRUPT STACK OK?
74 007306 001410                BEQ      2$                 ; BR IF OK
75 007310                DIE      #EM$SOF,#2       ; INTERRUPT STACK (INTSTK) OVERFLOW
76                                     ;
77                                     ;   Return to the user.
78                                     ;
79 007330 0000002                2$:    RTI                  ; RETURN FROM INTERRUPT OR EMT PROCESSING

```

FORK -- Queue a fork request

```

1          .SBTTL  FORK  -- Queue a fork request
2          ;-----
3          ; FORK is called to queue a fork request.
4          ; Note that INTEN must have been called before FORK is called and that
5          ; nothing may be pushed on the stack between the INTEN call and the FORK.
6          ; A fork request is held until the last active interrupt routine is ready
7          ; to return to the job that was originally interrupted then the fork
8          ; requests are processed in order by priority and, within the same priority,
9          ; by the order in which they were queued.
10         ; The form of the call to FORK is:
11         ;
12         ;     JSR     R5,FORK
13         ;     .WORD  <forkblock-.> or <priority>
14         ;
15         ; The TSX fork routine differs from the RT-11 fork routine in that TSX
16         ; uses an internal set of fork request blocks rather than using blocks
17         ; provided by the caller of FORK.
18         ; The word following the JSR R5,FORK may contain the address of a user
19         ; fork block, or it may contain a priority value in the range 1 to 127
20         ; which becomes the fork processing priority, or it may be zero (0) in which
21         ; case a default fork priority (FP$DEF) is used.
22         ;
23         ;
24 007332 010446 FORK:  MOV     R4,-(SP)
25         ;
26         ; Get a free fork block from the free list.
27         ;
28 007334 004737 007442' CALL   FRKGET      ;Get a free fork block
29         ;
30         ; We got a free fork block.  R4 = Address of block.
31         ; Set up information in fork request block.
32         ;
33 007340 012664 000000G MOV    (SP)+,FQ$R4(R4) ;Save R4 in fork block
34 007344 012664 000000G MOV    (SP)+,FQ$R5(R4) ;And R5
35 007350 010164 000000G MOV    R1,FQ$R1(R4)   ;Save other registers
36 007354 010264 000000G MOV    R2,FQ$R2(R4)
37 007360 010364 000000G MOV    R3,FQ$R3(R4)
38 007364 012502 MOV    (R5)+,R2       ;Get addr of user's fork block or fork prio.
39 007366 001416 BEQ    4$             ;Br if no user fork block or specified prio.
40 007370 003006 BGT    5$             ;Br if offset to a fork block
41 007372 020227 000000G CMP    R2,#FP$MAX     ;Is this a priority or an address?
42 007376 101003 BHI    5$             ;Br if it is the address of a user fork block
43 007400 110264 000000G MOV    R2,FQ$PRI(R4) ;Set fork priority
44 007404 000407 BR     4$
45 007406 060502 5$:   ADD    R5,R2       ;Get address of FQ$RTN in user's fork block
46 007410 162702 000002 SUB    #2,R2          ;Get pointer to start of user's fork block
47 007414 010264 000000G MOV    R2,FQ$UFB(R4) ;Save pointer to user's fork block
48 007420 010522 MOV    R5,(R2)+      ;Make FQ$LNK in user's fork block non-zero
49 007422 010512 MOV    R5,(R2)       ;Make FQ$RTN in user's fork block non-zero
50 007424 016402 000000G 4$:   MOV    FQ$R2(R4),R2 ;Recover R2
51 007430 010564 000000G MOV    R5,FQ$RTN(R4) ;Save address of routine to call
52         ;
53         ; Add fork block to queue of waiting fork blocks
54         ;
55 007434 004737 007534' CALL   FORKQ        ;Queue the fork request
56         ;
57         ; Finished -- Return to INTEN routine which will check for fork requests

```

58

59 007440 000207

RETURN

;Return to INTEN routine

```

1          .SBTTL  FRKGET -- Get a free Fork block
2          ;-----
3          ; FRKGET is called to get a free fork block.
4          ; If no free fork blocks are available, a system crash occurs.
5          ;
6          ; Outputs:
7          ; R4 = Address of fork block.
8          ; The following fields are initialized in the fork block:
9          ; FQ$PRI = Default fork priority (FP$DEF)
10         ; FQ$PA5 = Current KPAR5 mapping
11         ; FQ$PA6 = Current KPAR6 mapping
12         ; FQ$UFB = 0
13         ;
14 007442  FRKGET:
15         ;
16         ; Get a fork block from the free list
17         ;
18 007442          DISABL          ;** Disable interrupts **
19 007450 013704 000000G          MOV      FREFRK,R4          ;Get address of a free fork block
20 007454 001005          BNE      3$          ;Br if fork block is available
21 007456          DIE      #EM$FRK          ;System halt if no free fork blocks
22 007470 016437 000000G 000000G 3$:  MOV      FQ$LNK(R4),FREFRK;Remove fork block from the free list
23 007476          ENABL          ;** Enable interrupts **
24         ;
25         ; Set default values in the fork block
26         ;
27 007504 112764 000000G 000000G          MOVB     #FP$DEF,FQ$PRI(R4);Set default priority
28 007512 013764 000000G 000000G          MOV      @#KPAR5,FQ$PA5(R4);Save KPAR5 mapping
29 007520 013764 000000G 000000G          MOV      @#KPAR6,FQ$PA6(R4);And PAR6
30 007526 005064 000000G          CLR      FQ$UFB(R4)          ;No user fork block
31         ;
32         ; Finished
33         ;
34 007532 000207          RETURN
35

```

```

1          .SBTTL  FORKQ  -- Queue a fork request
2          ;-----
3          ; FORKQ is called to place a fork request block on the fork-pending
4          ; list. The queue entry is entered in the request queue based on its
5          ; priority as stored in the FQ$PRI field of the fork block.
6          ;
7          ; Inputs:
8          ; R4 = Address of fork request block to be queued.
9          ;
10         007534  010246  FORKQ:  MOV     R2,-(SP)
11         007536  010346          MOV     R3,-(SP)
12         ;
13         ; Do a linear search down the list of current fork entries and look
14         ; for the correct position to insert our new entry based on its priority.
15         ;
16         007540  012703  000000C      MOV     #FRKCQE-FQ$LNK,R3;Get pointer to dummy fork block at head
17         007544          DISABL          ;** Disable interrupts **
18         007552  016302  000000G      1$:   MOV     FQ$LNK(R3),R2 ;Get address of following fork block
19         007556  001406          BEQ     2$           ;Br if we should insert at end of chain
20         007560  126462  000000G 000000G  CMPB   FQ$PRI(R4),FQ$PRI(R2);Is next entry of lower priority?
21         007566  101002          BHI     2$           ;Br if yes
22         007570  010203          MOV     R2,R3       ;Link forward to next entry
23         007572  000767          BR     1$           ;And continue searching for insert point
24         ;
25         ; Insert following the entry pointed to by R3 and before the entry
26         ; pointed to by R2
27         ;
28         007574  010463  000000G      2$:   MOV     R4,FQ$LNK(R3) ;Make previous entry point to us
29         007600  010264  000000G      MOV     R2,FQ$LNK(R4) ;Make new entry point to following one
30         ;
31         ; Finished
32         ;
33         007604          ENABL          ;** Enable interrupts **
34         007612  012603          MOV     (SP)+,R3
35         007614  012602          MOV     (SP)+,R2
36         007616  000207          RETURN
  
```

SYNCH -- Queue a synch request

```

1          .SBTTL SYNCH -- Queue a synch request
2          ;-----
3          ; SYNCH is called to queue a synch request.
4          ; A synch request can be made by a handler when it reaches a point where
5          ; it must run in user state. The call to SYNCH simply queues a synch
6          ; request for the job.
7          ; The synch routine is called from the job scheduler at the point where
8          ; the job is selected and set up ready to run in user state.
9          ;
10         ; Inputs:
11         ; R4 = Address of 7-word synch control block.
12         ; R5 = Address following synch call (JSR R5,SYNCH)
13         ;
14 007620 010046 SYNCH: MOV R0,-(SP)
15 007622 010146 MOV R1,-(SP)
16 007624 010246 MOV R2,-(SP)
17 007626 010446 MOV R4,-(SP)
18 007630 005764 0000000 TST SN$RTN(R4) ; IS THIS SYNCH BLOCK FREE?
19 007634 001074 BNE 9$ ; BR IF NOT
20 007636 016402 0000000 MOV SN$JOB(R4),R2 ; GET JOB NUMBER FROM SYNCH BLOCK
21 007642 042702 177400 BIC #^C<377>,R2 ; KILL SIGN EXTENSION FROM HANDLER MOV#
22 007646 001467 BEQ 9$ ; ZERO IS INVALID
23 007650 006302 ASL R2 ; CONVERT TO WORD TABLE INDEX #
24 007652 020227 0000000 CMP R2,#LSTSL ; IS IT VALID LINE #?
25 007656 101063 BHI 9$ ; BR IF NOT
26 007660 032762 0000000 0000000 BIT ##DILUP,LSW(R2) ; IS JOB LOGGED ON?
27 007666 001457 BEQ 9$ ; BR IF NOT
28         ;
29         ; Synch block looks good.
30         ; Queue a completion request for the job.
31         ;
32 007670 004737 0000000 CALL GETRTQ ; Get a free queue element (address in R1)
33 007674 152761 0000000 0000000 BISB #<QF$SYN!QF$SCR>,CQ$FLG(R1); Synch routine, call in kernel mode
34 007702 110261 0000000 MOV# R2,CQ$JOB(R1) ; Set job index number for compl routine
35 007706 012700 0000000 MOV #S$TWFN,R0 ; Get compl priority for non-interactive jobs
36 007712 005762 0000000 TST LITIME(R2) ; Is this job interactive?
37 007716 001402 BEQ 1$ ; Br if not
38 007720 012700 0000000 MOV #S$HICP,R0 ; Get compl prio for interactive jobs
39 007724 110061 0000000 1$: MOV# R0,CQ$RNS(R1) ; Set execution state for compl routine
40 007730 116261 0000000 0000000 MOV# LPRI(R2),CQ$PRI(R1); Set execution priority
41 007736 112761 0000000 0000000 MOV# #CP$SYN,CQ$CP(R1); Set compl routine class priority
42 007744 005725 TST (R5)+ ; Point to successful return point for synch
43 007746 010561 0000000 MOV R5,CQ$RTN(R1) ; Set address of routine to call
44 007752 010564 0000000 MOV R5,SN$RTN(R4) ; Set flag saying synch block is busy
45 007756 016461 0000000 0000000 MOV SN$ID(R4),CQ$RO(R1); Set synch ID value to be passed in R0
46 007764 010461 0000000 MOV R4,CQ$R1(R1) ; Set address of cell to be cleared by call
47 007770 062761 0000000 0000000 ADD #CQ$RTN,CQ$R1(R1)
48 007776 013761 0000000 0000000 MOV @#KPAR5,CQ$PA5(R1); Set PAR 5 mapping to use for synch routine
49 010004 010104 MOV R1,R4 ; Get address of completion request block to R4
50 010006 004737 0000000 CALL QCOMPL ; Queue a completion request for the job
51         ;
52         ; Successful completion of synch request.
53         ; Do a RTS PC to return from handler interrupt.
54         ; Synch routine will be called from job scheduler.
55         ;
56 010012 012604 MOV (SP)+,R4
57 010014 012602 MOV (SP)+,R2

```

SYNCH -- Queue a synch request

```

58 010016 012601          MOV      (SP)+,R1
59 010020 012600          MOV      (SP)+,R0
60 010022 005726          TST      (SP)+          ;POP R5 VALUE
61 010024 000207          RETURN
62
63          ; Error -- Invalid synch control block.
64          ;          Return to word following .synch request.
65          ;
66 010026 012604          9#:    MOV      (SP)+,R4
67 010030 012602          MOV      (SP)+,R2
68 010032 012601          MOV      (SP)+,R1
69 010034 012600          MOV      (SP)+,R0
70 010036 000205          RTS      R5          ;ERROR RETURN FROM .SYNCH

```

QUNSIG -- Signal quantum expiration

```

1          .SBTTL  QUNSIG -- Signal quantum expiration
2          ;-----
3          ; QUNSIG is called when a time-slice quantum expires to see if the
4          ; user wants to have the terminal bell rung to signal that the time
5          ; slice has expired.
6          ; The SET SIGNAL command is used to control this feature.
7          ;
8          ; Form of the call:
9          ;     JSR      RO,QUNSIG
10         ;     .WORD   $SQxxx
11         ;
12         ; Where $SQxxx is the flag in LSWB that controls which quantum expired.
13         ;
14         ; Inputs:
15         ;     R1 = Job index number.
16         ;
17 010040  QUNSIG:
18         ;
19         ; See if used wants to be notified about this quantum running out
20         ;
21 010040  032061  0000009      BIT      (RO)+,LSWB(R1) ;Is notification wanted?
22 010044  001407              BEQ      9$          ;Br if not
23         ;
24         ; Ring terminal bell
25         ;
26 010046  010046              MOV      RO,-(SP)
27 010050  012700  0000007      MOV      #7,RO          ;Get bell character
28 010054              DCALL   QUECHR          ;Send bell to terminal
29 010062  012600              MOV      (SP)+,RO
30         ;
31         ; Finished
32         ;
33 010064  000200      9$:     RTS      RO          ;Return

```

SYSHLT -- Fatal system halt

```

1          .SBTTL SYSHLT -- Fatal system halt
2          ;-----
3          ; SYSHLT is entered when a fatal system error is detected.
4          ; An error message is printed and the system is halted.
5          ;
6          ; Inputs:
7          ; (SP) = Address of call to SYSHLT.
8          ; DIEMSG = Address of error message to print.
9          ; DIEARG = Argument value to print with error message.
10         ;
11 010066 013737 000000G 000034' SYSHLT: MOV     @#KPAR5,TRPAR5 ; Save current kernel par 5 mapping
12 010074          SYSHLT: DISABL ; DISABLE ALL INTERRUPTS
13 010102 012637 000042'          MOV     (SP)+,DIEPC ; GET ADDRESS OF CALL TO SYHLT
14 010106 010637 000044'          MOV     SP,DIESP ; Save stack pointer at time of crash
15 010112 020627 000000G          CMP     SP,#VPAR5 ; Are we running on context blk stack?
16 010116 103402          BLD     2$ ; Br if not
17 010120 012706 001000          MOV     #SS,SP ; RUN ON SYSTEM STACK NOW
18 010124 000005          2$: RESET ; RESET ALL DEVICES
19 010126 052737 000000G 000000G  BIS     #MMENBL,@#SR0MMR; MAKE SURE MEMORY MANAGEMENT ENABLED
20 010134 105737 000000G          TSTB   MEM256 ; DOES MACHINE HAVE AT LEAST 256 KB?
21 010140 001403          BEQ     1$ ; BR IF NOT
22 010142 052737 000000C 000000G  BIS     #EMMAP!IOMAP,@#SR3MMR ; TURN ON 22-BIT MEMORY MANAGEMENT
23 010150 000137 000000G          1$: JMP     SYSDIE ; ENTER OVERLAY TO PRINT ERROR MESSAGE

```

INIUMP --- Final system initialization

```
1          .SBTTL  INIUMP -- Final system initialization
2          ;-----
3          ; Turn on kernel mode memory management and perform final system
4          ; initialization.
5          ;
6 010154   INIUMP:
7          ;
8          ; Now jump to EXCINI in TSMISC overlay to complete initialization
9          ;
10 010154 000137 000000G      JMP      EXCINI
11          000202'          .END      START
```

Errors detected: 0

*** Assembler statistics

Work file reads: 0
Work file writes: 0
Size of work file: 8709 Words (35 Pages)
Size of core pool: 17920 Words (70 Pages)
Operating system: RT-11

Elapsed time: 00:00:56.48
DK:TSEXEC,LP:TSEXEC=DK:TSEXEC.MAC/C/N:SYM

S\$LOW	1-104	23-26	24-27							
S\$RT	1-104	23-22	24-23							
S\$SPND	1-48	7-23								
S\$TMWT	1-56	1-93								
S\$TWFN	1-54	45-35								
S\$WFM	1-79	21-18								
SB\$LNK	1-89	32-15								
SB\$PNT	1-62	32-23*								
SB\$TXT	1-62	32-22								
SCHED	1-21	5-15#	27-56	41-43						
SETMAP	1-32	7-45	14-16#	20-106						
SLTSIZ	1-41	2-67#								
SN\$ID	1-100	45-45								
SN\$JOB	1-100	45-20								
SN\$RTN	1-100	45-18	45-44*							
SNMSHD	1-89	32-10	32-15*							
SPCPS	1-52	41-48	41-56	41-57*						
SPDJOB	1-36	2-16#								
SPSAVE	1-76	5-33*	7-65							
SR\$PAR	1-47	34-35								
SR\$PDR	1-47	34-42								
SR\$PX	1-47	34-30								
SROMMR	1-55	33-10*	47-19*							
SR3MMR	1-55	47-22*								
SS	1-29	1-39	2-4#	5-31	19-52	19-197	47-17			
SSEND	1-39	2-5#	41-70							
START	4-6#	48-11								
STKLVL	1-24	2-21#	38-29	39-36*	40-19*	40-103*	41-8			
STOP	1-21	28-41								
SUTOP	1-25	20-13#								
SWAPER	1-53	8-17								
SWPCHK	6-18	8-5#								
SWPCOT	1-37	2-18#								
SWPJOB	1-40	2-66#								
SWPPOS	1-40	2-65#								
SYNCH	1-31	45-14#								
SYPNCR	1-32	2-62#								
SYSDIE	1-68	1-75	47-23							
SYSHL1	1-22	47-12#								
SYSHLT	1-59	5-26	33-68	35-17	35-25	35-29	41-72	41-75	43-21	47-11#
SYSMAP	1-33	2-36#	16-27*							
SYSXIT	1-25	41-6#								
TIKCNT	1-34	2-51#	37-25*							
TK5CNT	1-34	2-56#								
TRP10	1-24	33-32#								
TRP14	1-27	33-41#								
TRP20	1-27	33-60#								
TRP24	1-27	33-68#								
TRP250	1-23	33-10#								
TRP34	1-28	33-72#								
TRP4	1-24	33-15#								
TRPAR5	1-22	2-37#	33-25*	33-34*	47-11*					
TRPBPT	1-69	33-56								
TRPCOM	1-68	33-64	33-78							
TRPMAP	33-75	34-15#								
TRYMEM	1-37	9-30	10-16#							

TRYPLS	1-36	12-14#							
TRYRGN	1-36	13-21#							
TSEXEC	1-6#	1-26							
TSXTX	1-68	33-28	33-37						
UEXINT	1-31	35-10#							
UEXRTN	1-33	35-21#							
UFPTRP	1-65	17-41	18-21						
UHIMEM	1-83	20-23*	36-28						
UIDCNT	1-29	2-57#							
UMODE	1-82	19-148	33-41	41-6					
UMSPSV	1-38	17-24*	18-55*	19-57*					
UPARO	1-78	14-34*	14-75*	14-135*	15-15	34-38*			
UPAR1	1-78	14-38*							
UPAR6	1-81	14-123*							
UPAR7	1-81	14-99*	14-113*						
UPDRO	1-78	14-36*	14-73*	14-87*	14-133*	15-17	34-45*		
UPDR1	1-78	14-40*							
UPDR6	1-81	14-125*							
UPDR7	1-81	14-101*	14-115*						
UPMODE	1-74	7-57	17-21	18-17	18-58	19-134	19-148	36-15	41-58
UPSTAT	7-66	18-15#	19-19B						
UREGO	1-23	29-10#							
USP	1-22	2-59#	39-42*	40-105					
USRJOB	1-36	1-63	2-15#	28-37					
VHIPCT	1-104	30-27	31-13						
VINTIO	1-88	30-28							
VPAR5	1-70	22-23	47-15						
VPRIHI	1-68	19-174	19-190	23-20	24-21				
VPRILO	1-68	19-176	19-180	19-182	23-24	24-25			
VPRIVR	1-57	19-178							
VSWPFL	1-50	8-5							
WINDSP	1-49	7-80							

