Table of contents

```
                                        .TITLE  TSEXEC -- TSX-Plus Executive Module
000000                                  .CSECT  TSEXEC
                                        .ENABL  LC
                                        .ENABL  AMA
                                        .DSABL  GBL
000000                          TSEXEC:
                                ;
                                ;  TSEXEC is the executive control module of TSX-Plus.
                                ;  It contains routines to perform job scheduling, memory management,
                                ;  clock interrupt processing, etc.
                                ;
                                ;  Copyright (c) 1980, 1981, 1982, 1983, 1984, 1985.
                                ;  S&H Computer Systems, Inc.  Nashville, Tn.
                                ;
                                ;  All rights reserved.
                                ;
                                ;  Written by Phil Sherrod.
                                ;
                                ;  Global definitions
                                ;
                                        .GLOBL  SCHED,EXEC,STOP,CLKINT,INTLVL,CLKRTI,DIEARG,DIEMSG
                                        .GLOBL  DIEPC,DIESP,SYSHL1,TRPAR5,USP,INBSY,OUTBSY,EXCBUF
                                        .GLOBL  ENQHD,ENQTL,DEQ,FORCEX,UREGO,TRP250,QSRCH
                                        .GLOBL  CHKABT,QHIPRI,FPTRAP,TRP4,TRP10,STKLVL,ABRTOV
                                        .GLOBL  DOSCHD,INTEN,SYSXIT,SUTOP,MRKTHD,INIJMP
                                        .GLOBL  TSEXEC,FORK,KMNSTR,KMNSTK,KMNPGS
                                        .GLOBL  KMNCHN,QNSPND,INTPRI,TRP14,TRP20,TRP24
                                        .GLOBL  TRP34,EXCIN1,INITFL,GTSYMB,QHDSPN,MBFFLG
                                        .GLOBL  ODTTRP,DATIML,DATIMH,UIOCNT,SS,ENSYS
                                        .GLOBL  PMUSER,PMFLGS,PMBASE,PMTOP,PMNBPC,INTENX
                                        .GLOBL  PMPAR,PMRUN,PMCELS,SYNCH,UEXINT,MEMPAR
                                        .GLOBL  SETMAP,JMPO,LOKSWP,MEMSWP,DTLX,SYPNCR,CKUSP2
                                        .GLOBL  CHKUSP,EXCJOB,EM$SOF,MAPSYS,SYSMAP,UEXRTN
                                        .GLOBL  CLKCNT,TIKCNT,TK5CNT,MINCTR,CLKPS,CLKPC
                                        .GLOBL  RUNQHD,RUNQTL,QCPU,QNSPNX,QHDSPX
                                        .GLOBL  BRKPT,USRJOB,SPDJOB,GETMEM,TRYPLS,TRYRGN
                                        .GLOBL  INTPRI,FRKCQE,RUNQHD,SWPCOT,TRYMEM,MEMXPN
                                        .GLOBL  FREMEM,FPUUSE,UMSPSV,CURVC
                                        .GLOBL  INTSTK,INTSND,SS,SSEND,QUNSIG
                                        .GLOBL  FRKPRI,FRKGET,FORKQ,SWPPOS,SWPJOB,CURFRK
                                        .GLOBL  CXTWDS,CXTPAG,CXTPDR,CXTRMN,SLTSIZ
                                        .GLOBL  CXTBUF,CXBOWN,CXBJOB,CXBBAS,CXBSIZ,CXBMOV
                                ;
                                ;  Global references
                                ;
                                        .GLOBL  $GEMAR
                                        .GLOBL  CFLAG,MAXSRD,SR$PAR,SR$PDR,SR$PX,DOTRMP
                                        .GLOBL  $DILUP,$INIT,$DISCN,LIOCNT,IOHLTM,S$SPND
                                        .GLOBL  $VNOTT,GETRTQ,$WDISP,WINDSP
                                        .GLOBL  $INKMN,$SETCC,NEWUSR,VSWPFL,$RDSAV,LSW11
                                        .GLOBL  S$IOFN,S$CPU,$FPUEX,PO$DBG,PRIVCO
                                        .GLOBL  EMTCAD,SPCPS,CPLEMT,MAPPAR,$SUSPN
                                        .GLOBL  S$$RUN,QUECHR,$DBGBK,DBGBRK,SWAPER
                                        .GLOBL  QF$SYN,QF$IOT,CQ$JOB,S$TWFN,QCOMPL
                                        .GLOBL  MMENBL,SROMMR,MEM256,EMMAP,IOMAP,SR3MMR
                                        .GLOBL  S$INWT,S$TMWT,$DEBUG,BRKENT,$NOUCR
                                        .GLOBL  LSW,LSW4,LJSW,LBSPRI,VPRIVR,CURCP
```

```
                        .GLOBL  LSW8,$SGIIO,$SGHIO,CQ$FLG,QF$SCR
                        .GLOBL  FREIOQ,LCMPL,SYSHLT,CQ$CP
                        .GLOBL  LSTATE,FQ$PRI,FP$MAX,FP$DEF,CP$SYN
                        .GLOBL  CQ$LNK,CQ$RTN,CQ$R0,CQ$R1,CQ$PA5
                        .GLOBL  SB$TXT,SB$PNT,CCFLG,LSCCA,FREFRK
                        .GLOBL  USRJOB,INITGO
                        .GLOBL  RPAR,RPDR,D.FLAG,D$DMON
                        .GLOBL  ERRLOC,JSWLOC,UFPTRP
                        .GLOBL  EM$PFT
                        .GLOBL  EM$UEI,EM$MPR,EM$JMO,EM$FRK
                        .GLOBL  TSXTX,TRPCOM,SYSDIE,LPRI,VPRIHI,VPRILO
                        .GLOBL  $CTRLC,LSW9,$VIRJB,TRPBPT
                        .GLOBL  LQLINK,$NOABT,CXTBAS,VPAR5
                        .GLOBL  CORUSR,MA$RGN
                        .GLOBL  MINTIM,FP$CKT
                        .GLOBL  CW$FPU,CONFIG,LIOHLD
                        .GLOBL  S$$HIP,UPMODE
                        .GLOBL  $INCOR,SYSDIE,FPTRPX
                        .GLOBL  LMEMIN,LBASE,SPSAVE,PCCCR2,PROFLG
                        .GLOBL  QFREE,$NDMEM,LSTSL,CONFIG
                        .GLOBL  UPAR0,UPDR0,UPAR1,UPDR1,KPAR6,PSW
                        .GLOBL  S$WFM,LNBLKS,LPARBS,$MLOCK,$NLOCK
                        .GLOBL  FQ$LNK,FQ$R4,FQ$R5,FQ$RTN,FQ$PA6
                        .GLOBL  RMNPDR,UPAR6,UPDR6,UPAR7,UPDR7
                        .GLOBL  UMODE,$IOMAP,S$$RT,LCXPAR,FQ$PA5
                        .GLOBL  UHIMEM,FREIOQ,FQ$UFB
                        .GLOBL  LITIME,FQ$R1,FQ$R2,FQ$R3
                        .GLOBL  S$HICP
                        .GLOBL  CW$FPU,JSTKND
                        .GLOBL  MINTIM
                        .GLOBL  VINTIO
                        .GLOBL  SNMSHD,SB$LNK,$KINIT
                        .GLOBL  NMUMB
                        .GLOBL  LQUAN
                        .GLOBL  BASMAP,LOMAP,HIMAP,FREPGS,$MAPOK,LSW7
                        .GLOBL  S$TMWT,S$INWT
                        .GLOBL  CQ$PRI,CQ$RNS
                        .GLOBL  MAPUSR
                        .GLOBL  CS$ERR,CS$EOF
                        .GLOBL  PMSIZE
                        .GLOBL  R$UBAS
                        .GLOBL  LSW6
                        .GLOBL  SN$RTN,SN$JOB,SN$ID
                        .GLOBL  OVRHC,OVRADD,O.PAR,KPAR5
                        .GLOBL  CLKRUN,IOPAGE
                        .GLOBL  LNSBLK
                        .GLOBL  LHIPCT,VHIPCT,S$RT,S$LOW
                        .GLOBL  CUPAR0,CUPAR1,CUPAR6,CUPAR7
                        .GLOBL  CUPDR0,CUPDR1,CUPDR6,CUPDR7
```

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

```
000116  001130          MINCTR: .WORD   600.
000120  000000          USP:    .WORD   0
000122  000340          INTPRI: .WORD   340
000124  000000          FRKCQE: .WORD   0               ;FORK queue head
000126  000000          SYPNCR: .WORD   0               ;Pending system mark-time compl requests
000130  000000          CURVC:  .WORD   0               ;Addr of current direct interrupt control blk
000132  000000          CURFRK: .WORD   0               ;Address of currently running fork routine
000134  000000          SWPPOS: .WORD   0               ;Ptr to table with swap file slot positions
000136  000000          SWPJOB: .WORD   0               ;Ptr to table with #'s of jobs in swap slots
000140  000000          SLTSIZ: .WORD   0               ;# blocks used by each slot in swap file
000142  000000          PMUSER: .WORD   0               ;# of user doing performance monitoring
000144  000000          PMFLGS: .WORD   0               ;PF$ flags for performance monitor
000146  000000          PMBASE: .WORD   0               ;Base address being monitored
000150  000000          PMTOP:  .WORD   0               ;Top address of region being monitored
000152  000000          PMNBPC: .WORD   0               ;Number of bytes per p.m. cell
000154  000000          PMPAR:  .WORD   0               ;Address of pm vector area
000156  0000000C        PMCELS: .WORD   PMSIZE/2        ;Number of cells in pm vector area
000160  000000          PMRUN:  .WORD   0               ;Non-zero if performance mon in progress
000162  000000          CXTWDS: .WORD   0               ;# words for job context block
000164  000000          CXTPAG: .WORD   0               ;# 512-byte pages for job context block
000166  000000          CXTPDR: .WORD   0               ;PDR value to map job context block
000170  000000          CXTRMN: .WORD   0               ;Address in context area of simulated RMON
000172  000000          CXTBUF: .WORD   0               ;Addr of buffer used for accessing cxt blk
000174  000000          CXBBAS: .WORD   0               ;Addr of data currently in CXTBUF
000176  000000          CXBSIZ: .WORD   0               ;Amt of data currently in CXTBUF
000200  000000          RMNPDR: .WORD   0               ;PDR value to map to simulated RMON
                        ;
                        ;  End of TSEXEC data area.
                        ;  Begin 1024 byte buffer used by TSINIT at this point.
                        ;
000202                  EXCBUF:                         ;Base of area used by TSINIT for buffer
```

```
                        ;
                        ;  Macro to print an error message when a system crash occurs.
                        ;
                        ;  Arguments:
                        ;   MSG = Name of error message to print.
                        ;   ARG = (Optional) argument value to display with error message.
                        ;
                              .MACRO  DIE     MSG,ARG
                              MOV     MSG,@#DIEMSG
                              .IF     NB,ARG
                              MOV     ARG,@#DIEARG
                              .ENDC
                              CALL    @#SYSHLT
                              .ENDM   DIE
                        ;
                        ;  Macro definition for call global routines residing in mapped system regions
                        ;
                              .MACRO  OCALL   ENTADD
                              .IF     B,ENTADD
                                      .ERROR  ;OCALL SPECIFIED WITH NO ENTRY ADDRESS
                                      .MEXIT
                              .ENDC
                              CALL    OVRHC           ;CALL THE OVERLAY HANDLER
                              .WORD   ENTADD          ;SPECIFY THE ENTRY POINT
                              .ENDM
                        ;
                        ;-----------------------------------------------------
                        ;  MACROS TO ENABLE AND DISABLE INTERRUPTS.
                        ;  'DISABL' RAISES THE CPU PRIORITY LEVEL TO 7.
                        ;  'ENABL' LOWERS THE PRIORITY TO THE CURRENT OPERATING
                        ;  PRIORITY WHICH IS STORED IN INTPRI.
                        ;
                              .MACRO  DISABL          ;DISABLE INTERRUPTS
                              BIS     #340,@#PSW
                              .ENDM   DISABL
                        ;
                              .MACRO  ENABL           ;RESTORE INTERRUPT STATUS
                              BIC     INTPRI,@#PSW
                              .ENDM   ENABL
```

```
                                        ;
                                        ;  Starting point of execution of TSX-Plus.
                                        ;
                                        ;  Enter initialization module.
                                        ;
000202  000137  000000G                 START:  JMP     INITGO
                                        ;
                                        ;  Call from TSTTY for ^R to allow breakpoint capture for debugging.
                                        ;
000206  000207                          BRKPT:  RETURN                  ;return to TSTTY for echo processing
```

```
                                        .SBTTL  SCHED  -- Suspend job and look for another
                                ;-----------------------------------------------------------------
                                ;  The SCHED routine is called to suspend the execution of the
                                ;  current user and to look for another user to execute.
                                ;  SCHED selects the highest priority user who is ready to run
                                ;  (or waits for a user to get ready to run), gets that user into
                                ;  main memory, and then returns control over to the user.
                                ;  On return from SCHED, the user is ready to run and the execution
                                ;  time quantums have been set up.
                                ;  When called, the user's stack must be in SP.
                                ;  All registers are preserved.
                                ;
                                ;  Save all of the user's registers on his stack
                                ;
000210  010046                  SCHED:  MOV     R0,-(SP)
000212  010146                          MOV     R1,-(SP)
000214  010246                          MOV     R2,-(SP)
000216  010346                          MOV     R3,-(SP)
000220  010446                          MOV     R4,-(SP)
000222  010546                          MOV     R5,-(SP)
                                ;
                                ;  Check for user stack overflow.
                                ;
000224  020627  0000000                 CMP     SP,#JSTKND              ;check the stack limit
000230  101010                          BHI     1$                      ;br if stack is ok
000232                                  DIE     #EM$SOF,#3              ;stack overflow
                                ;
                                ;  Say user is no longer executing.
                                ;
000252  010603                  1$:     MOV     SP,R3           ;Save user's stack pointer
000254  012706  001000                  MOV     #SS,SP          ;Switch to system stack
000260  004737  002416'                 CALL    PKSTAT          ;Pack status into job context block
000264  010337  0000000                 MOV     R3,SPSAVE       ;Save user's stack pointer in his context area
000270  105037  0000000                 CLRB    CORUSR          ;User is no longer running
```

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

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

```
                                        MFPD    @R0             ;Get current contents of user's flag cell
                                        BIS     #CCFLG,(SP)     ;Set ctrl-c flag
                                        MTPD    @R0             ;Store into cell in user's program space
                        4$:             BIC     #$SETCC,LSW4(R1);Say it has been done
                                ;
                                ;  Unpack job status information from context block.
                                ;
                        1$:             MOV     SPSAVE,R3       ;Get pointer to job context area stack
                                        CALL    UPSTAT          ;Unpack status for job
                                ;
                                ;  Switch to job's internal stack.
                                ;
                                        MOV     R3,SP           ;Run on stack in context block
                                ;
                                ;  See if we need to call any completion routines for this job.
                                ;
                                        CALL    DOCMPL          ;Run any pending completion routines
                                ;
                                ;  See if we need to redisplay display window for job
                                ;
                                        BIT     #$WDISP,LSW6(R1);Do we need to redisplay window?
                                        BEQ     9$              ;Br if not
                                        OCALL   WINDSP          ;Redisplay window for job
                                ;
                                ;  Restore user's registers
                                ;
                        9$:             MOV     (SP)+,R5
                                        MOV     (SP)+,R4
                                        MOV     (SP)+,R3
                                        MOV     (SP)+,R2
                                        MOV     (SP)+,R1
                                        MOV     (SP)+,R0
                                ;
                                ;  Return to life!
                                ;
                                        RETURN                  ;Return from SCHED
```

```
      1                                       .SBTTL  SWPCHK -- See if job swapping is needed
      2                                ;-----------------------------------------------------------------------
      3                                ;  SWPCHK is called to see if jobs should be swapped into or out of memory.
      4                                ;
      5 000626  105737  0000000       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  0000000               CALL    SWAPER          ;Try to do any job swapping
     18                                ;
     19                                ;  Finished
     20                                ;
     21 000652  000207                9$:     RETURN
```

```
                                        .SBTTL  GETMEM -- Try to get free memory for a job
                                ;-----------------------------------------------------------------------
                                ;  GETMEM is called to attempt to obtain a memory region for a job.
                                ;  If the memory space is available it is claimed for the job.
                                ;
                                ;  Inputs:
                                ;   R1 = Job index number
                                ;   LMEMIN(R1) = Number of pages of memory needed for job
                                ;
                                ;  Outputs:
                                ;   MEMMAP updated to show memory space allocated to job.
                                ;   LBASE(R1)  = Base page number allocated to job.
                                ;   LPARBS(R1) = PAR relocation value for base of job (above context block)
                                ;   LCXPAR(R1) = PAR relocation value for job context block.
                                ;   C-flag set on return if not enough free space was available.
                                ;
000654  010246                  GETMEM: MOV     R2,-(SP)
                                ;
                                ;  See if the total number of free pages is adequate for the request
                                ;
000656  016100  000000G                 MOV     LMEMIN(R1),R0   ;Get total number of pages needed for job
000662  020037  000000G                 CMP     R0,FREPGS       ;Are there enough total free pages?
000666  101402                          BLOS    2$              ;Br if there are enough total free pages
000670  000261                          SEC                     ;Signal failure on return
000672  000421                          BR      9$
                                ;
                                ;  There are enough total free pages.
                                ;  Now see if there are enough contiguous free pages.
                                ;
000674  004737  000742'         2$:     CALL    TRYMEM          ;Try to find contiguous space for job
000700  103416                          BCS     9$              ;Br if cannot find free space for job
                                ;
                                ;  We got enough free space for the job.
                                ;  Set up information about memory for the job.
                                ;
000702  010261  000000G                 MOV     R2,LBASE(R1)    ;Base 512-byte page number assigned to job
000706  072227  000003                  ASH     #3,R2           ;Convert to 64-byte page number
000712  010261  000000G                 MOV     R2,LCXPAR(R1)   ;This is value for PAR to map to job context
000716  013700  000164'                 MOV     CXTPAG,R0       ;# 512-byte pages used by job context block
000722  072027  000003                  ASH     #3,R0           ;Convert to # 64-byte pages
000726  060002                          ADD     R0,R2           ;Add # pages used by job context block
000730  010261  000000G                 MOV     R2,LPARBS(R1)   ;This is value for 1st PAR register for job
000734  000241                          CLC                     ;Signal success on return
                                ;
                                ;  Finished
                                ;
000736  012602                  9$:     MOV     (SP)+,R2
000740  000207                          RETURN
```

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

```
      58 001046  000751                          BR      8$                ;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
```

```
                                        .SBTTL  FREMEM -- Free a memory region
                                ;-------------------------------------------------------------------
                                ;  FREMEM is called to free an area of memory.
                                ;
                                ;  Inputs:
                                ;   R2 = Base 512-byte page # of start of region.
                                ;   R0 = Number of 512-byte pages in region being freed.
                                ;
                                ;  Outputs:
                                ;   FREPGS is increased by number of pages being freed.
                                ;   MEMMAP is altered to show that the pages are free.
                                ;
001116  010246                  FREMEM: MOV     R2,-(SP)
001120  013746  000000G                 MOV     @#KPAR5,-(SP)   ;Save system PAR 5 mapping
001124  010046                          MOV     R0,-(SP)
001126  001411                          BEQ     9$              ;Br if no pages being freed
                                ;
                                ;  Map PAR 5 to the memory allocation table
                                ;
001130  013737  000000G 000000G         MOV     MAPPAR,@#KPAR5  ;Map PAR 5 to the memory alloc table
                                ;
                                ;  Indicate that we have more free memory
                                ;
001136  060037  000000G                 ADD     R0,FREPGS       ;Increase # free pages
                                ;
                                ;  Mark pages as free in MEMMAP
                                ;
001142  063702  000000G                 ADD     BASMAP,R2       ;Point to entry for first page
001146  105022                  1$:     CLRB    (R2)+           ;Mark pages in region as free
001150  077002                          SOB     R0,1$
                                ;
                                ;  Finished
                                ;
001152  012600                  9$:     MOV     (SP)+,R0
001154  012637  000000G                 MOV     (SP)+,@#KPAR5   ;Restore system PAR 5 mapping
001160  012602                          MOV     (SP)+,R2
001162  000207                          RETURN
```

```
                                        .SBTTL  TRYPLS -- Determine # memory pages available for PLAS
                                ;-------------------------------------------------------------------
                                ;  TRYPLS is called to determine the total number of user memory pages
                                ;  which could be used for a PLAS region.
                                ;  This routine is placed in Exec because we must use PAR 5 to access
                                ;  the memory allocation table.
                                ;
                                ;  Inputs:
                                ;   R1 = Current job index number.
                                ;
                                ;  Outputs:
                                ;   R0 = Number of 512-byte pages available for a PLAS region.
                                ;
001164  010246                  TRYPLS: MOV     R2,-(SP)
001166  013746  0000000                 MOV     @#KPAR5,-(SP)   ;Save system PAR 5 mapping
                                ;
                                ;  Map PAR 5 to the memory allocation table
                                ;
001172  013737  0000000 0000000         MOV     MAPPAR,@#KPAR5  ;Map PAR 5 to the memory alloc table
                                ;
                                ;  Count number of pages available for a PLAS region
                                ;
001200  013702  0000000                 MOV     LOMAP,R2        ;Point to base of memory map
001204  005000                          CLR     R0              ;Count pages in R0
001206  105722                  15$:    TSTB    (R2)+           ;Is this page available for user jobs?
001210  002401                          BLT     16$             ;Br if not
001212  005200                          INC     R0              ;Count number of user pages
001214  020237  0000000         16$:    CMP     R2,HIMAP        ;Checked all pages?
001220  103772                          BLO     15$             ;Br if not
001222  166100  0000000                 SUB     LNBLKS(R1),R0   ;Subtract space already allocated to this job
001226  166100  0000000                 SUB     LNSBLK(R1),R0   ;Including other plas regions
001232  012702  0000000                 MOV     #LSTSL,R2       ;Get index number of last job
001236  020201                  7$:     CMP     R2,R1           ;Is this our job?
001240  001410                          BEQ     8$              ;Don't count our job again
001242  032762  000000C 0000000         BIT     #$MLOCK!$NLOCK,LSW6(R2) ;Is this job locked in memory?
001250  001404                          BEQ     8$              ;Br if not
001252  166200  0000000                 SUB     LNBLKS(R2),R0   ;If locked, we can't use its space
001256  166200  0000000                 SUB     LNSBLK(R2),R0
001262  162702  000002          8$:     SUB     #2,R2           ;More jobs to check?
001266  001363                          BNE     7$              ;Br if yes
                                ;
                                ;  Finished
                                ;
001270  012637  0000000                 MOV     (SP)+,@#KPAR5   ;Restore system PAR 5 mapping
001274  012602                          MOV     (SP)+,R2
001276  000207                          RETURN
```

```
                                        .SBTTL  TRYRGN -- Try to allocate memory for global region
                                ;-----------------------------------------------------------------------
                                ;  TRYRGN is called to attempt to locate a free memory area for
                                ;  a shared global (PLAS) region.  The allocation is made in
                                ;  the highest available section of meory.
                                ;  If a free area of adequate size is found, it is claimed for the
                                ;  region.  If it is necessary to swap jobs to free up a large
                                ;  enough area, the base of the area to be gotten is returned.
                                ;
                                ;  Inputs:
                                ;   R0 = Number of 512-byte pages needed for region.
                                ;
                                ;  Outputs:
                                ;   C-flag cleared ==> Free area found and allocated for region.
                                ;       R2 = Base page # of region gotten.
                                ;   C-flag set ==> Could not find free area.
                                ;       R2 = 0 ==> Impossible to collect that much free space.
                                ;       R0 = Largest available area.
                                ;       R2 non-zero ==> Swapping needed, R2=base page # of region.
                                ;
001300  010146                  TRYRGN: MOV     R1,-(SP)
001302  010346                          MOV     R3,-(SP)
001304  010446                          MOV     R4,-(SP)
001306  010546                          MOV     R5,-(SP)
001310  013746  0000000                 MOV     @#KPAR5,-(SP)
001314  010046                          MOV     R0,-(SP)        ;Save requested memory size on top of stack
                                ;
                                ;  Set up pointers to memory map table
                                ;
001316  013737  0000000 0000000         MOV     MAPPAR,@#KPAR5  ;Map PAR 5 to memory allocation table
001324  013704  0000000                 MOV     HIMAP,R4        ;Pointer past last entry in memory map
001330  005000                          CLR     R0              ;Save largest area found in R0
                                ;
                                ;  Search for the start of a free region
                                ;
001332  010403                  8$:     MOV     R4,R3           ;Get pointer above top of next area to check
001334  163703  0000000                 SUB     LOMAP,R3        ;Get # pages left to check
001340  003463                          BLE     10$             ;Br if we have checked all pages
001342  114401                  1$:     MOVB    -(R4),R1        ;Is this page in use?
001344  001407                          BEQ     2$              ;Br if page is free
001346  002404                          BLT     6$              ;Br if page not available to user jobs
001350  032761  0000000 0000000         BIT     #$MLOCK,LSW6(R1);Page belong to job that's locked in memory?
001356  001402                          BEQ     2$              ;Br if not
001360  077310                  6$:     SOB     R3,1$           ;Continue searching for start of free area
001362  000452                          BR      10$             ;Br if no free area
                                ;
                                ;  Found the start of an available area.
                                ;  Determine how large it is.
                                ;
001364  011605                  2$:     MOV     (SP),R5         ;Get requested number of pages
001366  010402                          MOV     R4,R2           ;Save pointer to highest avail page in area
001370  005305                          DEC     R5              ;Do we only need 1 page?
001372  001424                          BEQ     5$              ;Br if yes
001374  020437  0000000         4$:     CMP     R4,LOMAP        ;Are we down to 1st page?
001400  101413                          BLOS    13$             ;Br if yes
001402  114401                          MOVB    -(R4),R1        ;Is this page in use?
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 001466  010403                          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                         ;Signal success on return
      97 001506  000402                          BR      12$
      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                         ;Signal failure on return
     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
```

```
                                        .SBTTL  SETMAP -- Set up memory mapping registers for a job
                                ;-----------------------------------------------------------------------
                                ;  SETMAP is called to load the memory mapping registers for a job.
                                ;  The user mode registers are set to point to the user program space.
                                ;
                                ;  Inputs
                                ;   R1 = Job index number.
                                ;   LNBLKS = # 256-word blocks assigned to job.
                                ;   LPARBS = Physical address (32-word block #) of job's virtual address 0.
                                ;
                                ;  Outputs:
                                ;   User-mode PAR and PDR registers are loaded.
                                ;   Mapping information is stored in CUPAR and CUPDR cells in job's
                                ;   context block.
                                ;
001534  010246                 SETMAP: MOV     R2,-(SP)
001536  010346                         MOV     R3,-(SP)
001540  010446                         MOV     R4,-(SP)
001542  010546                         MOV     R5,-(SP)
                                ;
                                ;  See if we are setting up mapping for Kmon or a regular user job.
                                ;
001544  032761  000000G 000000G         BIT     #$INKMN,LSW4(R1);Is Kmon running?
001552  001433                         BEQ     7$              ;Br if not
                                ;
                                ;  Set up mapping registers for Kmon as follows:
                                ;   000000-037777          --> 000000-037777  (Map over TSGEN)
                                ;   040000-(top of Kmon) --> Job program space
                                ;   140000-157777          --> Job context area
                                ;   160000-177777          --> Simulated Monitor vector table
                                ;
                                ;  Map virtual page 0 (000000-017777) to physical TSGEN; allow write access.
                                ;
001554  005037  0000000                 CLR     @#UPAR0         ;Map virtual page 0 to physical page 0
001560  005037  0000000                 CLR     CUPAR0          ;Save info in context block
001564  012737  077406  000000G         MOV     #77406,@#UPDR0  ;4kw window, allow read & write access
001572  012737  077406  000000G         MOV     #77406,CUPDR0   ;Save info in context block
001600  012737  000200  000000G         MOV     #200,@#UPAR1    ;Map virtual page 1 to physical page 1
001606  012737  000200  000000G         MOV     #200,CUPAR1
001614  012737  077406  000000G         MOV     #77406,@#UPDR1  ;4kw window, allow read & write access
001622  012737  077406  000000G         MOV     #77406,CUPDR1
                                ;  Now enter code to map virtual pages starting with # 1 to job program area.
001630  012704  000004                 MOV     #4,R4           ;Start mapping with page # 2
001634  012705  000016                 MOV     #14.,R5         ;End mapping with page # 7
001640  000411                         BR      6$
                                ;
                                ;  We are mapping an ordinary user job.
                                ;  Set up mapping as follows:
                                ;   000000-(top of program) --> Job program space
                                ;   160000-177777           --> Monitor vector table
                                ;
                                ;  Set up user mode mapping registers.
                                ;
001642  005004                 7$:     CLR     R4              ;Start mapping with page # 0
001644  012705  000016                 MOV     #14.,R5         ;Assume we should map through page 7
001650  032761  0000000 000000G         BIT     #$IOMAP,LSW6(R1);Does job want page 7 mapped to I/O page?
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  0000000          6$:      MOV     LNBLKS(R1),R2    ;Get # 256-word blocks assigned to job
     60 001670  163702  000164'                   SUB     CXTPAG,R2        ;Subtract # blocks used by context area
     61 001674  072227  000003                    ASH     #3,R2            ;Cvt to # 32-word blocks for job
     62 001700  016103  0000000                   MOV     LPARBS(R1),R3    ;Get base block # of job area
     63 001704  010200                    2$:      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$:      SUB     R0,R2            ;Get # blocks left after this page
     69 001724  005300                             DEC     R0               ;Get # blocks - 1
     70 001726  000300                             SWAB    R0               ;Put # blocks in left byte
     71 001730  052700  000006                     BIS     #6,R0            ;Allow read & write access to the page
     72 001734  042700  100261                     BIC     #100261,R0       ;Make sure unused PDR bits are zero
     73 001740  010064  0000000                    MOV     R0,UPDR0(R4)     ;Set PDR for page
     74 001744  010064  0000000                    MOV     R0,CUPDR0(R4)    ;Save info in context block
     75 001750  010364  0000000                    MOV     R3,UPAR0(R4)     ;Set PAR for page
     76 001754  010364  0000000                    MOV     R3,CUPAR0(R4)    ;Save info in context block
     77 001760  062703  000200                     ADD     #128.,R3         ;Advance page base block number
     78 001764  062704  000002                     ADD     #2,R4            ;Advance register pointer
     79 001770  020405                             CMP     R4,R5            ;Check for last PAR register mapped
     80 001772  003744                             BLE     2$               ;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  0000000                    CLR     UPDR0(R4)        ;Disallow all access to the page
     88 002004  005064  0000000                    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  0000000 0000000    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  0000000 0000000            MOV     #IOPAGE,@#UPAR7 ;Set address for PAR7
    100 002034  012737  0000000 0000000            MOV     #IOPAGE,CUPAR7  ;Set info in context block
    101 002042  012737  077406  0000000            MOV     #77406,@#UPDR7  ;Set read/write access and full page size
    102 002050  012737  077406  0000000            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  0000000 0000000   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  0000000                    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  0000000                    ADD     LCXPAR(R1),R2    ;Add PAR base for context area
    113 002110  010237  0000000                    MOV     R2,@#UPAR7       ;Map user page 7 to table
    114 002114  010237  0000000                    MOV     R2,CUPAR7        ;Save info in context block
```

```
    115 002120  013737  000200' 000000G          MOV     RMNPDR,@#UPDR7  ;Set up mapping control
    116 002126  013737  000200' 000000G          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  000000G 000000G 4$:      BIT     #$INKMN,LSW4(R1);Is Kmon running?
    121 002142  001414                           BEQ     10$             ;Br if not
    122 002144  016100  000000G                  MOV     LCXPAR(R1),R0   ;Get PAR value to map context block
    123 002150  010037  000000G                  MOV     R0,@#UPAR6      ;Map user page 6 to job context area
    124 002154  010037  000000G                  MOV     R0,CUPAR6       ;Save info in context block
    125 002160  013737  000166' 000000G          MOV     CXTPDR,@#UPDR6  ;Allow read and write access
    126 002166  013737  000166' 000000G          MOV     CXTPDR,CUPDR6
    127                                 ;
    128                                 ;  See if job has any associated shared run-time systems
    129                                 ;
    130 002174  005002                  10$:     CLR     R2              ;Init table index for PAR0
    131 002176  016200  000000G         13$:     MOV     RPDR(R2),R0     ;Do we need to load this PAR?
    132 002202  001412                           BEQ     14$             ;Br if not
    133 002204  010062  000000G                  MOV     R0,UPDR0(R2)    ;Set PDR control flags
    134 002210  010062  000000G                  MOV     R0,CUPDR0(R2)   ;Save info in context block
    135 002214  016262  000000G 000000G          MOV     RPAR(R2),UPAR0(R2);Set PAR value
    136 002222  016262  000000G 000000G          MOV     RPAR(R2),CUPAR0(R2);Save info in context block
    137 002230  062702  000002          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  000000G                  MOVB    R1,MAPUSR       ;Say memory mapping set up for this job
    144 002246  032761  000000G 000000G          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  000000G 000000G          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
```

```
                                        .SBTTL  LODMAP -- Load memory mapping from context block
                                ;-----------------------------------------------------------------
                                ;  LODMAP is called to load the user-mode memory management
                                ;  PAR and PDR registers from the mapping information in the job's
                                ;  context block.  The context block information was set up by the
                                ;  last call to the SETMAP routine.
                                ;
002276  010146                  LODMAP: MOV     R1,-(SP)
002300  010246                          MOV     R2,-(SP)
002302  010346                          MOV     R3,-(SP)
002304  010446                          MOV     R4,-(SP)
                                ;
                                ;  Set up pointers to the registers
                                ;
002306  012701  0000000                 MOV     #UPAR0,R1       ;Point to hardware PAR register base
002312  012702  0000000                 MOV     #CUPAR0,R2      ;Point to PAR data cells in context block
002316  012703  0000000                 MOV     #UPDR0,R3       ;Point to hardware PDR register base
002322  012704  0000000                 MOV     #CUPDR0,R4      ;Point to PDR data cells in context block
                                ;
                                ;  Load the PAR and PDR registers
                                ;
002326  012700  000010                  MOV     #8.,R0          ;Load 8 PAR and PDR registers
002332  012221                  1$:     MOV     (R2)+,(R1)+     ;Load a user-mode PAR register
002334  012423                          MOV     (R4)+,(R3)+     ;Load a user-mode PDR register
002336  077003                          SOB     R0,1$           ;Loop if more to load
                                ;
                                ;  Say memory mapping is set up for the current job
                                ;
002340  113737  0000000 0000000         MOVB    CORUSR,MAPUSR   ;Say memory mapping set up for current job
                                ;
                                ;  Finished
                                ;
002346  012604                          MOV     (SP)+,R4
002350  012603                          MOV     (SP)+,R3
002352  012602                          MOV     (SP)+,R2
002354  012601                          MOV     (SP)+,R1
002356  000207                          RETURN
```

```
                                                  .SBTTL  MAPSYS -- Map to the system code regions
                                          ;-------------------------------------------------------------
                                          ;
                                          ;  MAPSYS actually performs the mapping required to execute code in
                                          ;  the mapped system regions.  The mapped system code includes TSUSR,
                                          ;  TSEMT, TSLOCK, and TSSPOL.  The correct calling interface is
                                          ;
                                          ;       CALL    MAPSYS
                                          ;       with R5 containing the &lt;segment number&gt;*6
                                          ;
                                          ;  The output of the subroutine actually changes the KPAR5 contents to
                                          ;  get to the correct physical memory locations.
                                          ;  This routine must preserve the C-bit because it will be called
                                          ;  from the OCALL (inter-overlay call) handler.
                                          ;
                                          ;
                                          ;  Overlay table structure:
                                          ;
                                          ;       loc 64 --> $OVTAB:
                                          ;                       .WORD   <IDENTIFIER>,<KPAR5>,<WORD COUNT>
                                          ;                       DUMMY SUBROUTINES FOR ALL OVERLAY SEGMENTS
                                          ;
 002360                                   MAPSYS:
 002360  011646                                   MOV     (SP),-(SP)      ;reposition return address
 002362  013766  000000G 000002                   MOV     @#PSW,2(SP)     ;save psw with current interrupt priority
 002370  052737  000340  000000G                  BIS     #340,@#PSW      ;disable interrupts
 002376  010537  000032'                          MOV     R5,SYSMAP       ;store the mapped region number
 002402  063705  000000G                          ADD     OVRADD,R5       ;add the pointer to the overlay table
 002406  016537  177772G 000000G                  MOV     O.PAR-6(R5),@#KPAR5;enter the physical memory location
 002414  000002                                   RTI                     ;return with previous psw and priority
```

```
                                        .SBTTL  PKSTAT -- Pack user status into context block
                                ;-----------------------------------------------------------------------
                                ;  PKSTAT is called to pack all of the status information about the
                                ;  current user into the user's context block.
                                ;  After calling PKSTAT the user is ready to be outswapped.
                                ;  All registers are preserved.
                                ;  When called, we must be using the system stack.
                                ;
                                ;  Inputs:
                                ;   R3 = Pointer to top of user stack in job context area.
                                ;
                                ;  Outputs:
                                ;   R3 = Updated user stack pointer (after pushing job status).
                                ;
002416  010146                 PKSTAT: MOV     R1,-(SP)
002420  113701  0000000                 MOVB    CORUSR,R1       ;Get job index number
                                ;
                                ;  Save PSW and user mode SP.
                                ;
002424  013743  0000000                 MOV     @#PSW,-(R3)     ;Save PSW
002430  052737  0000000 0000000         BIS     #UPMODE,@#PSW   ;Make sure previous-mode = user
002436  106506                          MFPD    SP              ;Get user-mode SP
002440  011643                          MOV     (SP),-(R3)      ;Push user SP
002442  012637  0000000                 MOV     (SP)+,UMSPSV    ;Save it in context block cell also
                                ;
                                ;  Save some cells in program space
                                ;
002446  032761  0000000 0000000         BIT     #$RDSAV,LSW11(R1);Are we reading in SAV file now?
002454  001007                          BNE     1$              ;Br if yes -- Don't save info from memory
002456  106537  0000000                 MFPD    @#ERRLOC        ;Save error cells
002462  012643                          MOV     (SP)+,-(R3)
002464  106537  0000000                 MFPD    @#JSWLOC        ;Job status word
002470  012661  0000000                 MOV     (SP)+,LJSW(R1)
                                ;
                                ;  Save the current kernel PAR 5 region mapping
                                ;
002474  013743  0000000         1$:     MOV     @#KPAR5,-(R3)   ;Save kernel par 5 mapping
                                ;
                                ;  See if we need to save state of FPU
                                ;
002500  005737  0000000                 TST     UFPTRP          ;Is user using the FPU?
002504  001424                          BEQ     2$              ;Br if not
002506  032737  0000000 0000000         BIT     #CW$FPU,CONFIG  ;Does system have an FPU?
002514  001420                          BEQ     2$              ;Br if not
                                ;  Save contents of FPU accumulators
002516  010246                          MOV     R2,-(SP)
002520  170202                          STFPS   R2              ;Hold floating point status in R2
002522  010243                          MOV     R2,-(R3)        ;Save floating point status register
002524  170011                          SETD                    ;Set to 64-bit mode
002526  174043                          STD     R0,-(R3)        ;AC0
002530  174143                          STD     R1,-(R3)        ;AC1
002532  174243                          STD     R2,-(R3)        ;AC2
002534  174343                          STD     R3,-(R3)        ;AC3
002536  172404                          LDD     R4,R0           ;AC4-->AC0
002540  174043                          STD     R0,-(R3)        ;(AC4)
002542  172405                          LDD     R5,R0           ;AC5-->AC0
002544  174043                          STD     R0,-(R3)        ;(AC5)
```

```
     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
```

```
                                        .SBTTL  UPSTAT -- Unpack user status from context block
                                ;-----------------------------------------------------------------------
                                ;  UPSTAT is called to unpack the status information stored in the
                                ;  user's context area and get the user ready to run.
                                ;  When called, R1 must contain the user index #.
                                ;  All registers are preserved except R0.
                                ;
                                ;  Inputs:
                                ;   R1 = Job index number.
                                ;   R3 = User context area stack pointer
                                ;
                                ;  Outputs:
                                ;   R3 = User context area stack pointer after job status is popped.
                                ;
002562  010446                  UPSTAT: MOV     R4,-(SP)
002564  010546                          MOV     R5,-(SP)
002566  052737  0000000 0000000         BIS     #UPMODE,@#PSW   ;Make sure previous-mode = user
                                ;
                                ;  See if FPU unit needs to be restored
                                ;
002574  005737  0000000                 TST     UFPTRP          ;Is user using FPU?
002600  001420                          BEQ     3$              ;Br if not
002602  032737  0000000 0000000         BIT     #CW$FPU,CONFIG  ;Does system have an FPU?
002610  001414                          BEQ     3$              ;Br if not
                                ;  Restore the FPU registers and status
002612  110137  000013'                 MOVB    R1,FPUUSE       ;Set flag saying FPU unit is in use by job
002616  170011                          SETD                    ;Set 64-bit mode
002620  172423                          LDD     (R3)+,R0        ;Get AC5
002622  174005                          STD     R0,R5
002624  172423                          LDD     (R3)+,R0        ;Get AC4
002626  174004                          STD     R0,R4
002630  172723                          LDD     (R3)+,R3        ;AC3
002632  172623                          LDD     (R3)+,R2        ;AC2
002634  172523                          LDD     (R3)+,R1        ;AC1
002636  172423                          LDD     (R3)+,R0        ;AC0
002640  170123                          LDFPS   (R3)+           ;Load FPU status register
                                ;
                                ;  Restore the mapping for kernel PAR 5
                                ;
002642  012337  0000000         3$:     MOV     (R3)+,@#KPAR5   ;Restore mapping for PAR 5
                                ;
                                ;  Restore some cells in program space
                                ;
002646  032761  0000000 0000000         BIT     #$RDSAV,LSW11(R1);Are we reading in SAV file now?
002654  001007                          BNE     1$              ;Br if yes -- Don't alter memory image
002656  016146  0000000                 MOV     LJSW(R1),-(SP)  ;Job status word
002662  106637  0000000                 MTPD    @#JSWLOC
002666  012346                          MOV     (R3)+,-(SP)     ;Error cells
002670  106637  0000000                 MTPD    @#ERRLOC
                                ;
                                ;  Restore PSW mode and user-mode SP.
                                ;
002674  012346                  1$:     MOV     (R3)+,-(SP)     ;User-mode SP
002676  106606                          MTPD    SP
002700  005037  0000000                 CLR     UMSPSV          ;Say user SP has been loaded
002704  012300                          MOV     (R3)+,R0        ;Get saved PS
002706  042700  147777                  BIC     #147777,R0      ;Clear all but previous-mode field
```

```
     58 002712  042737  0000000 0000000         BIC     #UPMODE,@#PSW   ;Clear previous-mode field in PS
     59 002720  050037  0000000                 BIS     R0,@#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
```

```
 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  000000G                   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  000000G 000000G            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  000000G 000000G            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  000000G                   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  000000G                   CLR     UMSPSV          ;Say user SP is active- used for real-time int
```

```
                                        ;
                                        ;  Our status has been saved.
                                        ;  Save current job execution priority on stack so we can restore it after
                                        ;  calling all completion routines.
                                        ;
003032  116146  0000000                       MOVB    LPRI(R1),-(SP)  ;Save job execution priority
                                        ;
                                        ;  Process next completion request.
                                        ;
003036                                1$:     DISABL                  ;** Disable interrupts **
003044  012700  000000C                       MOV     #LCMPL-CQ$LNK,R0;Get fake pointer to compl Q head
003050  060100                                ADD     R1,R0           ;Point to list head for our job
003052  016005  0000000               17$:    MOV     CQ$LNK(R0),R5   ;Get address of next compl Q element
003056  001540                                BEQ     2$              ;Br if no more elements left to process
003060  126537  0000000 0000000             CMPB    CQ$CP(R5),CURCP ;Is this request class higher than current
003066  101534                                BLOS    2$              ;Br if not -- No more requests to process now
003070  132765  0000000 0000000             BITB    #QF$SCR,CQ$FLG(R5);Is this a system or user compl routine?
003076  001005                                BNE     16$             ;Br if system compl routine
003100  005761  0000000                       TST     LIOHLD(R1)      ;Should we hold user compl routines?
003104  001402                                BEQ     16$             ;Br if not
003106  010500                                MOV     R5,R0           ;Link to next element
003110  000760                                BR      17$             ;Skip over user compl requests in list
                                        ;
                                        ;  Unlink completion queue request from list.
                                        ;
003112  016560  0000000 0000000       16$:    MOV     CQ$LNK(R5),CQ$LNK(R0);Remove compl request from list
003120                                        ENABL                   ;** Enable interrupts **
                                        ;
                                        ;  Set job execution priority to that specified in completion queue element
                                        ;
003126  113746  0000000                       MOVB    CURCP,-(SP)     ;Save current completion rtn class priority
003132  116537  0000000 0000000             MOVB    CQ$CP(R5),CURCP ;Remember class priority of compl rtn
003140  116500  0000000                       MOVB    CQ$PRI(R5),R0   ;Get execution priority for compl queue
003144  120061  0000000                       CMPB    R0,LPRI(R1)     ;Do we need to change job priority?
003150  001004                                BNE     7$              ;Br if not
003152  110061  0000000                       MOVB    R0,LPRI(R1)     ;Set new execution priority
003156  105237  000004'                       INCB    DOSCHD          ;Say a job scheduler cycle is needed
                                        ;
                                        ;  Get some information out of the completion queue element and then
                                        ;  free the queue element.
                                        ;
003162  010103                        7$:     MOV     R1,R3           ;Save job index number in R3
003164  016504  0000000                       MOV     CQ$RTN(R5),R4   ;Address of completion routine to be called
003170  016546  0000000                       MOV     CQ$R0(R5),-(SP) ;Save info on stack for now
003174  016546  0000000                       MOV     CQ$R1(R5),-(SP)
003200  116502  0000000                       MOVB    CQ$FLG(R5),R2   ;Control flags
003204  016537  0000000 0000000             MOV     CQ$PA5(R5),@#KPAR5 ;Set up mapping for PAR 5
003212  010501                                MOV     R5,R1           ;Get address of queue element to R1
003214  004737  0000000                       CALL    QFREE           ;Free the queue element
003220  012601                                MOV     (SP)+,R1        ;Recover values to pass in R0 & R1
003222  012600                                MOV     (SP)+,R0
                                        ;
                                        ;  Determine if completion routine is a system routine that should
                                        ;  be called in kernel mode or a user completion routine to be
                                        ;  called in user mode.
                                        ;
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 .TIMIO 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  0000000 0000000 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  0000000 0000000         BIS     #UPMODE,@#PSW   ;Make sure previous mode = user
    135 003264  013702  0000000                 MOV     EMTCAD,R2       ;Get pointer to top of return addr stack
    136 003270  012742  003346'                 MOV     #5$,-(R2)       ;Push our return address
    137 003274  010237  0000000                 MOV     R2,EMTCAD       ;Save updated stack pointer
    138 003300  106506                          MFPD    SP              ;Get user's stack pointer
    139 003302  012602                          MOV     (SP)+,R2
    140 003304  012746  0000000                 MOV     #CPLEMT,-(SP)   ;Store address of exit EMT on user's stack
    141 003310  106642                          MTPD    -(R2)
    142 003312  010246                          MOV     R2,-(SP)        ;Restore updated user's stack pointer
    143 003314  106606                          MTPD    SP
    144                                 ;
    145                                 ;  Now push fake PS & PC on stack so RTI will enter compl routine
    146                                 ;  in user-mode.
    147                                 ;
    148 003316  012746  000000C                 MOV     #UMODE!UPMODE,-(SP);User-mode PS
    149 003322  032737  0000000 0000000         BIT     #D$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$:    MOV     R4,-(SP)        ;Address of completion routine
    153 003340  012705  000024                  MOV     #24,R5          ;Make R5 point to word with 0 for FORTRAN subs
    154 003344  000002                          RTI                     ;Enter completion routine in user mode
    155                                 ;
    156                                 ;  Return here from the completion routine (when user does EMT instruction).
    157                                 ;
    158                                 ;  See if there are more completion routines to call.
    159                                 ;
    160 003346  113701  0000000         5$:     MOVB    CORUSR,R1       ;Get back job index number
    161 003352  112637  0000000                 MOVB    (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$:     ENABL                   ;** Enable interrupts **
    168 003366  112603                          MOVB    (SP)+,R3        ;Get saved job execution priority
    169 003370  105737  0000000                 TSTB    CURCP           ;Are we still in a completion routine?
    170 003374  001025                          BNE     8$              ;Br if yes
    171 003376  116103  0000000                 MOVB    LBSPRI(R1),R3   ;Get base priority for job
```

```
                172 003402  032761  0000000 0000000         BIT     #$VNOTT,LSW(R1) ;Is this a disconnected virtual job?
                173 003410  001417                          BEQ     8$              ;Br if not
                174 003412  120337  0000000                 CMPB    R3,VPRIHI       ;Does job have a fixed high priority?
                175 003416  103014                          BHIS    8$              ;Br if yes
                176 003420  120337  0000000                 CMPB    R3,VPRILO       ;Does job have a fixed low priority?
                177 003424  101411                          BLOS    8$              ;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     8$              ;Br if ok
                182 003442  113703  0000000                 MOVB    VPRILO,R3       ;Force above fixed low priority
                183 003446  005203                          INC     R3
                184 003450  120361  0000000         8$:     CMPB    R3,LPRI(R1)     ;Are we changing the job's priority?
                185 003454  001404                          BEQ     10$             ;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
```

```
                                        .SBTTL  SUTOP  -- Set top of memory for a job
                                ;-----------------------------------------------------------------------
                                ;  SUTOP is called to set the top of memory for the current job.
                                ;  If memory expansion is being requested and the required memory space
                                ;  is not available, the job is suspended and outswapped until memory
                                ;  becomes available.
                                ;  If a memory contraction is being done, the memory area being freed
                                ;  is returned to an unused state.
                                ;
                                ;  Inputs:
                                ;   R0 = Address above desired top of program.
                                ;
003546  010046                  SUTOP:  MOV     R0,-(SP)
003550  010146                          MOV     R1,-(SP)
003552  010446                          MOV     R4,-(SP)
003554  010546                          MOV     R5,-(SP)
003556  013746  000000G                 MOV     @#KPAR5,-(SP)   ;Save system PAR 5 mapping
003562  013737  000000G 000000G         MOV     MAPPAR,@#KPAR5  ;Map PAR 5 to memory allocation table
003570  113701  000000G                 MOVB    CORUSR,R1       ;Get job index #
                                ;
                                ;  Set highest legal address for job and set base of USR
                                ;
003574  010037  000000G                 MOV     R0,UHIMEM       ;SET TOP OF MEMORY FOR JOB
003600  013705  000170'                 MOV     CXTRMN,R5       ;GET ADDRESS OF JOB'S SIMULATED MON VEC TABLE
003604  010065  000000G                 MOV     R0,R$UBAS(R5)   ;SAY BASE OF USR = TOP OF JOB
                                ;
                                ;  Convert top-of-memory address to 512-byte page number.
                                ;
003610  020027  177000                  CMP     R0,#177000      ;DOES JOB NEED 64KB?
003614  101403                          BLOS    7$              ;BR IF NOT
003616  012700  000200                  MOV     #128.,R0        ;128 PAGES = 64KB
003622  000407                          BR      8$
003624  062700  000777          7$:     ADD     #511.,R0        ;BOUND UP TO PAGE BOUNDARY
003630  000241                          CLC                     ;CONVERT # BYTES TO # WORDS
003632  006000                          ROR     R0
003634  000300                          SWAB    R0              ;CONVERT TO # PAGES
003636  042700  177400                  BIC     #^C377,R0       ;MASK OUT ALL BUT # PAGES
003642  063700  000164'          8$:     ADD     CXTPAG,R0       ;ADD # PAGES NEEDED FOR CONTEXT BLOCK
                                ;
                                ;  Compare new memory request with current memory allocation for this job.
                                ;
003646  020061  000000G          10$:    CMP     R0,LNBLKS(R1)   ;COMPARE NEW REQUEST WITH CURRENT ALLOCATION
003652  001501                          BEQ     3$              ;BR IF NO CHANGE IN SIZE
003654  101026                          BHI     1$              ;BR IF EXPANDING MEMORY SIZE
                                ;
                                ;  We are decreasing the size of the job.
                                ;  Free the memory pages above the new top of the job.
                                ;
003656  016104  000000G                 MOV     LBASE(R1),R4    ;GET BASE PAGE # ASSIGNED TO THE JOB
003662  016105  000000G                 MOV     LNBLKS(R1),R5   ;GET # PAGES ASSIGNED TO JOB NOW
003666  060504                          ADD     R5,R4           ;GET # OF PAGE ABOVE TOP OF JOB AREA
003670  063704  000000G                 ADD     BASMAP,R4       ;POINT TO ENTRY IN MEMMAP TABLE
003674  010061  000000G                 MOV     R0,LNBLKS(R1)   ;SET NEW # PAGES ASSIGNED TO JOB
003700  010061  000000G                 MOV     R0,LMEMIN(R1)   ;SET # BLOCKS NEEDED BY INSWAP
003704  066161  000000G 000000G         ADD     LNSBLK(R1),LMEMIN(R1) ;ADD MEMORY SPACE NEEDED FOR PLAS REGNS
003712  160005                          SUB     R0,R5           ;GET # PAGES BEING FREED
003714  060537  000000G                 ADD     R5,FREPGS       ;KEEP TRACK OF # FREE PAGES
```

```
                                2$:     CLRB    -(R4)           ;FREE A PAGE
                                        SOB     R5,2$           ;LOOP TO FREE MORE
                                        INCB    DOSCHD          ;REQUEST A JOB SCHEDULER CYCLE
                                        BR      3$
                                ;
                                ;  We are increasing the size of the job.
                                ;
                                ;  See if desired memory is available now.
                                ;
                                1$:     MOV     LBASE(R1),R4    ;GET BASE PAGE # ASSIGNED TO THIS JOB
                                        MOV     LNBLKS(R1),R5   ;GET # PAGES CURRENTLY ASSIGNED TO THIS JOB
                                        ADD     R5,R4           ;GET # OF PAGE ABOVE TOP OF JOB AREA
                                        ADD     BASMAP,R4       ;POINT INTO MEMMAP TABLE
                                        SUB     R5,R0           ;GET # PAGES TO BE ADDED
                                        MOV     R0,-(SP)
                                5$:     TSTB    (R4)+           ;IS THIS PAGE AVAILABLE?
                                        BNE     4$              ;BR IF NOT
                                        SOB     R0,5$           ;CHECK ALL PAGES WE NEED
                                ;
                                ;  The desired memory space is available.  Claim it for our job.
                                ;
                                        MOV     (SP)+,R0        ;GET # PAGES BEING ADDED
                                        MOV     LBASE(R1),R4    ;GET BASE PAGE # ASSIGNED TO THIS JOB
                                        ADD     LNBLKS(R1),R4   ;GET # OF PAGE ABOVE CURRENT TOP OF JOB
                                        ADD     R0,LNBLKS(R1)   ;INCREASE # PAGES ASSIGNED TO THIS JOB
                                        MOV     LNBLKS(R1),LMEMIN(R1);SET UP LMEMIN FOR SWAPPER
                                        ADD     LNSBLK(R1),LMEMIN(R1);ADD SPACE NEEDED FOR PLAS REGIONS
                                        ADD     BASMAP,R4       ;POINT INTO MEMMAP TABLE
                                        SUB     R0,FREPGS       ;KEEP TRACK OF # FREE PAGES
                                6$:     MOVB    R1,(R4)+        ;CLAIM PAGES FOR OUR JOB
                                        SOB     R0,6$
                                        BR      3$
                                ;
                                ;  The memory space we need is not now available.
                                ;  Force the job to be suspended and outswapped.
                                ;  The subsequent inswap will do the memory expansion for us.
                                ;
                                4$:     MOV     (SP)+,R0        ;GET # PAGES BEING ADDED
                                        ADD     LNBLKS(R1),R0   ;GET NEW TOTAL # PAGES FOR JOB ROOT
                                        MOV     R0,-(SP)        ;SAVE SIZE OF JOB ROOT
                                        ADD     LNSBLK(R1),R0   ;ADD SPACE NEEDED FOR PLAS REGIONS
                                        CALL    MEMXPN          ;DO JOB SWAP TO EXPAND MEMORY SPACE
                                        MOV     (SP)+,R0        ;GET BACK # PAGES NEEDED BY JOB ROOT
                                        BR      10$             ;WE SHOULD NOW HAVE ALL NEEDED
                                ;
                                ;  Memory allocation has been done.
                                ;  Load the memory management registers for the job.
                                ;
                                3$:     CALL    SETMAP          ;LOAD MEMORY MANAGEMENT REGISTERS FOR THE JOB
                                ;
                                ;  Finished
                                ;
                                        MOV     (SP)+,@#KPAR5   ;Restore system PAR 5 mapping
                                        MOV     (SP)+,R5
                                        MOV     (SP)+,R4
                                        MOV     (SP)+,R1
                                        MOV     (SP)+,R0
```

```
    115 004076  000207                          RETURN
```

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

```
                                        .SBTTL  CXBMOV -- Move job context data into buffer
                                ;-----------------------------------------------------------------
                                ;  This routine is called to move data from some area of physical memory
                                ;  into the job context block access buffer (CXTBUF).
                                ;  This routine is placed in the root because it uses PAR 5 to access
                                ;  the physical memory area.
                                ;
                                ;  Inputs:
                                ;   R5 = Base 64-byte block number of start of data in physical memory.
                                ;   R3 = Offset within data area of item being accessed.
                                ;   R0 = Number of bytes of data to move (512 maximum).
                                ;
004144  010346                  CXBMOV: MOV     R3,-(SP)
004146  010446                          MOV     R4,-(SP)
004150  013746  0000000                 MOV     @#KPAR5,-(SP)   ;Save PAR 5 mapping
                                ;
                                ;  Map PAR 5 to area being accessed
                                ;
004154  010537  0000000                 MOV     R5,@#KPAR5      ;Map par 5 to context block
                                ;
                                ;  Set up registers for the move
                                ;
004160  062703  0000000                 ADD     #VPAR5,R3       ;Get mapped address of start of data
004164  013704  000172'                 MOV     CXTBUF,R4       ;Point to buffer where data is to go
004170  006200                          ASR     R0              ;Get # words to move
                                ;
                                ;  Move the data
                                ;
004172  006200                          ASR     R0              ;Get number of double-words to move
004174  001403                          BEQ     2$              ;Br if less than 2 words to move
004176  012324                  1$:     MOV     (R3)+,(R4)+     ;Move a word
004200  012324                          MOV     (R3)+,(R4)+     ;Move second word of pair
004202  077003                          SOB     R0,1$           ;Loop till all moved
004204  103001                  2$:     BCC     9$              ;Br if don't need to move odd word at end
004206  011314                          MOV     (R3),(R4)       ;Move last word
                                ;
                                ;  Finished
                                ;
004210  012637  0000000         9$:     MOV     (SP)+,@#KPAR5   ;Restore par 5 mapping
004214  012604                          MOV     (SP)+,R4
004216  012603                          MOV     (SP)+,R3
004220  000207                          RETURN
```

```
                                        .SBTTL  ENQHD  -- Put user at head of queue
                                ;-----------------------------------------------------------------
                                ;  ENQHD is called to place a user in the run queue at the front
                                ;  of the list of users of equal or lower priority.
                                ;  When called, R1 must contain the user index number and
                                ;  R0 must contain the execution state (S$----).
                                ;  All registers are preserved.
                                ;
004222  010246                  ENQHD:  MOV     R2,-(SP)
004224  010346                          MOV     R3,-(SP)
004226  010446                          MOV     R4,-(SP)
                                ;
                                ;  If job is being placed in an executable state and the priority
                                ;  of the job is one of the fixed priorities (very low or very high),
                                ;  then force job to be placed in the S$LOW or S$RT queue.
                                ;
004230  116104  0000000                 MOVB    LPRI(R1),R4     ;Get current priority for job
004234  020027  0000000                 CMP     R0,#S$$RUN      ;Is job being placed in an executable state?
004240  101013                          BHI     3$              ;Br if not
004242  120437  0000000                 CMPB    R4,VPRIHI       ;Does job have a real-time priority?
004246  103403                          BLO     4$              ;Br if not
004250  012700  0000000                 MOV     #S$RT,R0        ;Force real-time jobs into S$RT state
004254  000405                          BR      3$
004256  120437  0000000         4$:     CMPB    R4,VPRILO       ;Is this a low-priority job?
004262  101002                          BHI     3$              ;Br if not
004264  012700  0000000                 MOV     #S$LOW,R0       ;Force into low-priority queue
                                ;
                                ;  Remove user from queue he is in currently.
                                ;
004270  004737  004534'         3$:     CALL    DEQ             ;REMOVE FROM QUEUE ** DISABLE **
                                ;
                                ;  Search down queue looking for right place to insert user.
                                ;
004274  113702  000000'                 MOVB    RUNQHD,R2       ;POINT TO FIRST USER IN QUEUE
004300  001432                          BEQ     ADQ1            ;BR IF QUEUE IS EMPTY
004302  020062  0000000         1$:     CMP     R0,LSTATE(R2)   ;COMPARE EXECUTION STATE PRIO WITH NEXT JOB
004306  101004                          BHI     2$              ;BR IF OUR EXECUTION STATE IS LOWER PRIO
004310  103412                          BLO     ADQMID          ;BR IF OUR EXECUTION STATE IS HIGHER PRIO
004312  120462  0000000                 CMPB    R4,LPRI(R2)     ;EQUAL EXECUTION STATES, COMPARE PRIORITIES
004316  103007                          BHIS    ADQMID          ;BR IF PRIO IS EQUAL TO OR HIGHER THAN NEXT
004320  010203                  2$:     MOV     R2,R3           ;CHAIN ON TO NEXT USER IN LIST
004322  116302  0000000                 MOVB    LQLINK(R3),R2
004326  001365                          BNE     1$              ;BR IF MORE USERS IN QUEUE
                                ;
                                ;  Add user to tail of queue
                                ;
004330  110137  000001'         ADQTL:  MOVB    R1,RUNQTL       ;SAY WE ARE LAST USER IN LIST
004334  000404                          BR      ADQT
                                ;
                                ;  Link in front of user whose index # is in R2
                                ;
004336  116203  0000010         ADQMID: MOVB    LQLINK+1(R2),R3 ;GET INDEX OF EARLIER USER
004342  110162  0000010                 MOVB    R1,LQLINK+1(R2) ;SAY WE ARE PREDECESSOR TO R2 USER
004346  110261  0000000         ADQT:   MOVB    R2,LQLINK(R1)   ;SAY R2 USER FOLLOWS US
004352  110361  0000010                 MOVB    R3,LQLINK+1(R1) ;SAY R3 USER IS OUR PREDECESSOR
004356  001405                          BEQ     ADQHD           ;BR IF WE ARE AT HEAD OF LIST
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:   MOVB    R1,RUNQTL       ;MAKE QUEUE TAIL POINT TO US
      63 004372  110137  000000'        ADQHD:  MOVB    R1,RUNQHD       ;MAKE QUEUE HEAD POINT TO US
      64 004376  010061  000000        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
```

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

```
                                        .SBTTL  DEQ     -- Remove user from run queue
                                ;-----------------------------------------------------------------
                                ;  DEQ is called to remove from the run queue the user whose
                                ;  index number is in R1.
                                ;  On return, the user will be left unlinked from the run queue
                                ;  and his state code (LSTATE) will be zeroed.
                                ;  ** The interrupts are left disabled on return **
                                ;  All registers are preserved.
                                ;
004534  010246                  DEQ:    MOV     R2,-(SP)
004536  010346                          MOV     R3,-(SP)
004540                                  DISABL                  ;** DISABLE **
004546  005761  0000000                 TST     LQLINK(R1)      ;IS USER UNLINKED NOW?
004552  001010                          BNE     1$              ;BR IF NOT
004554  120137  000000'                 CMPB    R1,RUNQHD       ;IS USER ONLY ENTRY IN QUEUE?
004560  001030                          BNE     2$              ;BR IF NOT -- MUST NOT BE IN QUEUE AT ALL
                                ;  User is only entry in queue
004562  105037  000000'                 CLRB    RUNQHD          ;REMOVE FROM QUEUE
004566  105037  000001'                 CLRB    RUNQTL
004572  000423                          BR      2$
                                ;  Unlink from queue
004574  116103  0000010         1$:     MOVB    LQLINK+1(R1),R3 ;GET # OF USER IN FRONT OF US IN QUEUE
004600  116102  0000000                 MOVB    LQLINK(R1),R2   ;GET # OF USER WHO FOLLOWS US IN QUEUE
004604  001003                          BNE     4$              ;BR IF NOT AT TAIL OF QUEUE
004606  110337  000001'                 MOVB    R3,RUNQTL       ;MAKE TAIL POINT TO OUR PREDECESSOR
004612  000402                          BR      5$
004614  110362  0000010         4$:     MOVB    R3,LQLINK+1(R2) ;MAKE OUR SUCCESSOR POINT BACK OVER US
004620  005703                  5$:     TST     R3              ;ARE WE AT HEAD OF QUEUE?
004622  001003                          BNE     6$              ;BR IF NOT AT HEAD OF QUEUE
004624  110237  000000'                 MOVB    R2,RUNQHD       ;MAKE QUEUE HEAD POINT TO OUR SUCESSOR
004630  000402                          BR      3$
004632  110263  0000000         6$:     MOVB    R2,LQLINK(R3)   ;MAKE OUR PREDECESSOR POINT TO OUR SUCESSOR
                                ;  Finished unlinking.
                                ;  Say user is not in any queue
004636  005061  0000000         3$:     CLR     LQLINK(R1)      ;CLEAR BOTH OUR FORWARD & BACKWARD LINKS
004642  005061  0000000         2$:     CLR     LSTATE(R1)      ;SAY WE HAVE NO EXECUTION STATE
004646  012603                          MOV     (SP)+,R3
004650  012602                          MOV     (SP)+,R2
004652  000207                          RETURN
```

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

```
                        .SBTTL  QNSPND -- Put job in wait state
                ;-----------------------------------------------------------------------
                ;  There are four routines that can be called to suspend the
                ;  execution of a job: QNSPND, QNSPNX, QHDSPN, QHDSPX.
                ;  All four routines perform the functions of changing the
                ;  job state to a specified wait state and then calling the
                ;  job scheduler to run some other job while the current job is waiting.
                ;  Before placing the job in the specified wait state, these routines
                ;  check to see if there is a pending completion routine for the job.
                ;  If there is a pending completion routine, the job state is changed
                ;  to the state associated with the completion routine to allow the
                ;  completion routine to run before the job is suspended.
                ;  The difference between the four routines is whether the job is
                ;  queued at the head or tail of the wait queue and whether the
                ;  job's time-slice quantum is reset.
                ;
                ;  QNSPND -- Queue at tail of wait list,  reset quantum.
                ;  QNSPNX -- Queue at tail of wait list,  don't reset quantum.
                ;  QHDSPN -- Queue at head of wait list,  reset quantum.
                ;  QHDSPX -- Queue at head of wait list,  don't reset quantum.
                ;
                ;  Inputs:
                ;   R0 = Job state to which job is to be set before calling scheduler.
                ;
                ;  Outputs:
                ;   Interrupts are enabled.
                ;
                ;  Queue at tail of wait list, reset quantum.
                ;
004722  010146  QNSPND: MOV     R1,-(SP)
004724  004737  005014'         CALL    QCKCPL          ;DO COMMON ENTRY SETUP  ** DISABLE **
004730  004737  004424'         CALL    ENQTL           ;PUT JOB IN WAIT QUEUE  ** ENABLE **
004734  000421                 BR      DOSPNR          ;RESET QUANTUM AND SUSPEND JOB
                ;
                ;  Queue at tail of wait list, don't reset quantum.
                ;
004736  010146  QNSPNX: MOV     R1,-(SP)
004740  004737  005014'         CALL    QCKCPL          ;DO COMMON ENTRY SETUP  ** DISABLE **
004744  004737  004424'         CALL    ENQTL           ;PUT JOB IN WAIT QUEUE  ** ENABLE **
004750  000415                 BR      DOSPNX          ;GO SUSPEND JOB
                ;
                ;  Queue at head of wait list, don't reset quantum.
                ;
004752  010146  QHDSPX: MOV     R1,-(SP)
004754  004737  005014'         CALL    QCKCPL          ;DO COMMON ENTRY SETUP  ** DISABLE **
004760  004737  004222'         CALL    ENQHD           ;PUT JOB IN WAIT QUEUE  ** ENABLE **
004764  000407                 BR      DOSPNX          ;GO SUSPEND JOB
                ;
                ;  Queue at head of wait list, reset quantum.
                ;
004766  010146  QHDSPN: MOV     R1,-(SP)
004770  004737  005014'         CALL    QCKCPL          ;DO COMMON ENTRY SETUP  ** DISABLE **
004774  004737  004222'         CALL    ENQHD           ;PUT JOB IN WAIT QUEUE  ** ENABLE **
005000  005061  000000  DOSPNR: CLR     LQUAN(R1)       ;RESET JOB TIME-SLICE QUANTUM
005004  012601          DOSPNX: MOV     (SP)+,R1
005006  004737  000210'         CALL    SCHED           ;CALL JOB SCHEDULER TO RUN ANOTHER JOB
005012  000207                 RETURN                  ;RESUME EXECUTION OF OUR JOB
```

```
                                ;-----------------------------------------------------------------
                                ;  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:
                                ;   R0 = Requested wait state for job to be placed in.
                                ;
                                ;  Outputs:
                                ;   R0 = Job state that job is actually to be put in.
                                ;   R1 = Current job index number.
                                ;   Interrupts are left disabled.
                                ;
005014  010246                  QCKCPL: MOV     R2,-(SP)
005016  113701  0000000                 MOVB    CORUSR,R1       ;Get current job index number
005022                                  DISABL                  ;** Disable **
                                ;
                                ;  Check for pending completion routines
                                ;
005030  016102  0000000                 MOV     LCMPL(R1),R2    ;Are there any pending completion rtns?
005034  001413                          BEQ     9$              ;Br if not
005036  005761  0000000                 TST     LIOHLD(R1)      ;Are we holding completion routines?
005042  001010                          BNE     9$              ;Br if yes
005044  105737  0000000                 TSTB    CURCP           ;Are we in a completion routine now?
005050  001005                          BNE     9$              ;Br if yes
005052  120062  0000000                 CMPB    R0,CQ$RNS(R2)   ;Does compl routine have higher priority?
005056  101402                          BLOS    9$              ;Br if not
005060  116200  0000000                 MOVB    CQ$RNS(R2),R0   ;Change to prio as specified by compl routine
                                ;
                                ;  Finished
                                ;
005064  012602                  9$:     MOV     (SP)+,R2
005066  000207                          RETURN
```

```
                                        .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.
                                ;
005070  010046                 FORCEX: MOV     R0,-(SP)
005072  026127  000000G 000000G         CMP     LSTATE(R1),#S$$HIP ;IS USER RUNNING NOW?
005100  103404                         BLO     1$                 ;BR IF YES -- CAN'T BEAT THAT
                                ;
                                ;  User is not running now.
                                ;  Put user in high priority run state
                                ;
005102  012700  000000G                MOV     #S$IOFN,R0         ;GET HIGH-PRIORITY STATE CODE
005106  004737  004424'                CALL    ENQTL              ;PUT USER AT TAIL OF QUEUE
005112  105237  000004'         1$:    INCB    DOSCHD             ;REQUEST A JOB SCHEDULER CYCLE
005116  012600                         MOV     (SP)+,R0
005120  000207                         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.
                                ;
005122  010146                 CHKABT: MOV     R1,-(SP)
005124  113701  000000G                MOVB    CORUSR,R1          ;GET USER INDEX NUMBER
005130  032761  000000C 000000G        BIT     #<$CTRLC!$DISCN>,LSW(R1);ABORT PENDING?
005136  001411                         BEQ     1$                 ;BRANCH IF NOT
005140  120137  000002'                CMPB    R1,USRJOB          ;ARE WE CURRENTLY DOING A DIRECTORY OPERATION?
005144  001406                         BEQ     1$                 ;IF YES DON'T ABORT NOW
005146  032761  000000G 000000G        BIT     #$NOABT,LSW9(R1);Is the No-abort flag set for job?
005154  001002                         BNE     1$                 ;Br if yes -- Don't abort now
005156  004737  000000G         2$:    CALL    STOP               ;ABORT THE USER
005162  012601                 1$:     MOV     (SP)+,R1
005164  000207                         RETURN
```

```
                                        .SBTTL  UREGO  -- Restart user at head of wait queue
                                ;-----------------------------------------------------------------------
                                ;  UREGO is called to restart the user who
                                ;  is at the head of the wait queue whose state code is in R0.
                                ;  This user is removed from the wait queue and
                                ;  added to the tail of the S$IOFN queue.
                                ;  All registers are preserved.
                                ;  Interrupts are enabled on return.
                                ;
005166  010146                  UREGO:  MOV     R1,-(SP)
005170  004737  004654'                 CALL    QSRCH           ;FIND USER AT HEAD OF QUEUE * DISABLE *
005174  103402                          BCS     1$              ;BR IF CAN'T FIND ANY USERS IN THAT STATE
005176  004737  005206'                 CALL    QHIPRI          ;REQUEUE USER AT TAIL OF S$IOFN QUEUE
005202  012601                  1$:     MOV     (SP)+,R1
005204  000207                          RETURN
```

```
 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$IOFN high priority execution queue.
 5                                      ;  All registers are preserved.
 6                                      ;
 7 005206  010046                       QHIPRI: MOV     R0,-(SP)
 8                                      ;
 9                                      ;  If this is an interactive job doing I/O, put the job in the S$HICP
10                                      ;  state.
11                                      ;
12 005210  005761  000000G                     TST     LITIME(R1)      ;Is this job interactive or compute bound?
13 005214  001426                              BEQ     3$              ;Br if compute bound
14 005216  005761  000000G                     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  000000G                     DEC     LHIPCT(R1)      ;Decrease remaining number of interactive I/O
17 005230  012700  000000G                     MOV     #S$HICP,R0      ;Get interactive completion state code
18 005234  004737  004424'                     CALL    ENQTL           ;Put job at tail of that queue
19 005240  000433                              BR      2$
20                                      ;
21                                      ;  Interactive job has performed maximum number of I/O operations while
22                                      ;  in interactive state.  Reclasify the job as non-interactive.
23                                      ;
24 005242  005061  000000G             4$:     CLR     LITIME(R1)      ;Say job is no longer interactive
25 005246  004037  010040'                     JSR     R0,QUNSIG       ;Signal that INTIOC has been used up
26 005252  000000G                             .WORD   $SGIIO
27 005254  013700  000000G                     MOV     VHIPCT,R0       ;Get # high-prio boosts allowed for CPU jobs
28 005260  163700  000000G                     SUB     VINTIO,R0       ;Any remaining after number already used?
29 005264  003410                              BLE     5$              ;Br if not
30 005266  010061  000000G                     MOV     R0,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  000000G             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     R0,QUNSIG       ;SIGNAL THAT HIPRCT WAS USED UP
38 005304  000000G                             .WORD   $SGHIO
39 005306  004737  005334'             5$:     CALL    QCPU            ;QUEUE AS CPU-BOUND JOB IF YES
40 005312  000406                              BR      2$
41 005314  005361  000000G             1$:     DEC     LHIPCT(R1)      ;ONE LESS HIGH-PRIORITY HIT REMAINING
42 005320  012700  000000G                     MOV     #S$IOFN,R0      ;GET STATE CODE
43 005324  004737  004424'                     CALL    ENQTL           ;PUT USER AT TAIL OF QUEUE
44 005330  012600                      2$:     MOV     (SP)+,R0
45 005332  000207                              RETURN
```

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

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

```
                                        .SBTTL  TSXTRP -- Catch traps
                                ;-------------------------------------------------------------------------
                                ;  TSXT4 and TSXT10 catch traps to 4 and 10 respectively.
                                ;  If the user did a .TRPSET his routine is entered.
                                ;  Otherwise the job is aborted.
                                ;
                                ;
                                ;  Trap to 250 (Memory management trap)
                                ;
005444  012737  0000000 0000000 TRP250: MOV     #MMENBL,@#SROMMR;RESET ERROR FLAGS
                                ;  Treat trap to 250 like trap to 4.
                                ;
                                ;  Trap to 4.
                                ;
005452  010446                  TRP4:   MOV     R4,-(SP)
005454  113704  0000000                 MOVB    CORUSR,R4       ;Get current job index number
005460  032764  0000000 0000000         BIT     #$GEMAR,LSW11(R4);Are we accessing user's argument block?
005466  001410                          BEQ     1$              ;Br if not
005470  012604                          MOV     (SP)+,R4        ;Pop R4
005472  011646                          MOV     (SP),-(SP)      ;Move down PC
005474  016666  000004  000002          MOV     4(SP),2(SP)     ;Move down PS
005502  005066  000004                  CLR     4(SP)           ;Store 0 value to be returned
005506  000002                          RTI                     ;Return following MFPD with 0 on stack
005510  010546                  1$:     MOV     R5,-(SP)
005512  013737  0000000 000034'        MOV     @#KPAR5,TRPAR5  ;SAVE KPAR5 FROM TRAP FOR ERROR HANDLING
005520  013737  120002  0000000         MOV     @#120002,ABRTOV ;SAVE RAD50 OVERLAY NAME
005526  012705  000001                  MOV     #1,R5           ;ERROR CODE FOR TRAP 4
005532  000137  0000000                 JMP     TSXTX           ;ENTER TRAP HANDLER IN TSX OVERLAY
                                ;
                                ;  Trap to 10
                                ;
005536  010446                  TRP10:  MOV     R4,-(SP)
005540  010546                          MOV     R5,-(SP)
005542  013737  0000000 000034'        MOV     @#KPAR5,TRPAR5  ;SAVE KPAR5 FROM TRAP FOR ERROR HANDLING
005550  013737  120002  0000000         MOV     @#120002,ABRTOV ;SAVE RAD50 OVERLAY NAME
005556  012705  000002                  MOV     #2,R5           ;ERROR CODE FOR TRAP 10
005562  000137  0000000                 JMP     TSXTX           ;ENTER TRAP HANDLER IN TSX OVERLAY
                                ;
                                ;  Trap to 14 (Breakpoint trap)
                                ;
005566  032766  0000000 000002  TRP14:  BIT     #UMODE,2(SP)    ;DID BREAKPOINT OCCUR IN USER OR KERNEL MODE?
005574  001002                          BNE     1$              ;BR IF USER MODE
                                ;  Breakpoint occurred in kernel mode.
                                ;  Give control to system ODT.
005576  000177  172226                  JMP     @ODTTRP         ;ENTER SYSTEM DEBUGGER
                                ;  Breakpoint occurred in user mode.
                                ;  Give control to user's debugger program.
005602  010446                  1$:     MOV     R4,-(SP)        ;
005604  113704  0000000                 MOVB    CORUSR,R4       ;Get current job index number
005610  032764  0000000 0000000         BIT     #$DEBUG,LSW9(R4);Is program being run with TSX debugger?
005616  001403                          BEQ     2$              ;Br if not
                                ;  Enter TSX-Plus debugger
005620  012604                          MOV     (SP)+,R4
005622  000137  0000000                 JMP     BRKENT          ;Enter TSX-Plus debugger
                                ;  Enter user's debugger
005626  000137  0000000         2$:     JMP     TRPBPT          ;ENTER TRAP HANDLING ROUTINE IN TSX OVERLAY
                                ;
```

```
                                ;  Trap to 20 (IOT)
                                ;
005632  010446                TRP20:  MOV     R4,-(SP)
005634  010546                        MOV     R5,-(SP)
005636  012705  000013                MOV     #13,R5          ;GET ERROR CODE
005642  012704  000020                MOV     #20,R4          ;GET TRAP LOCATION
005646  000137  0000000               JMP     TRPCOM          ;ENTER TRAP HANDLING ROUTINE IN TSX OVERLAY
                                ;
                                ;  Trap to 24 (Power fail)
                                ;
005652                          TRP24:  DIE     #EM$PFT         ;SYSTEM HALT IF POWER FAIL TRAP
                                ;
                                ;  Trap to 34 (TRAP instruction)
                                ;
005664  010446                TRP34:  MOV     R4,-(SP)
005666  010546                        MOV     R5,-(SP)
005670  105737  0000000               TSTB    DOTRMP          ;Using TRAP instruction for mapping?
005674  001017                        BNE     TRPMAP          ;Br if yes
005676  012705  000015                MOV     #15,R5          ;GET ERROR CODE
005702  012704  000034                MOV     #34,R4          ;GET TRAP LOCATION
005706  000137  0000000               JMP     TRPCOM          ;ENTER TRAP HANDLING ROUTINE IN TSX OVERLAY
                                ;
                                ;  Trap to 244 (Floating point exception interrupt).
                                ;  Set $FPUEX flag and return through SYSXIT which will do actual
                                ;  FPU exception processing when we are about to return to user mode.
                                ;
005712  004537  006404'       FPTRAP: JSR     R5,INTEN        ;Standard interrupt entry
005716  000000                        .WORD   0               ;Run at priority 7
005720  113704  0000000               MOVB    CORUSR,R4       ;Get current job index number
005724  052764  0000000 0000000       BIS     #$FPUEX,LSW(R4) ;Set flag for job saying FPU interrupt
005732  000207                        RETURN                  ;Return and perform FPU exception code
```

```
                                        .SBTTL  TRPMAP -- High-performance memory mapping service
                                ;-------------------------------------------------------------------
                                ;  This routine is jumped to when a TRAP instruction is executed and we
                                ;  are are doing high-performance memory mapping.
                                ;
                                ;  Inputs:
                                ;   R0 = Mapping region index number.
                                ;
                                ;  Stack:
                                ;         (SP) = R5
                                ;       2(SP) = R4
                                ;       4(SP) = PC
                                ;       6(SP) = PS
                                ;
005734                          TRPMAP:
                                ;
                                ;  Clear C-flag in PS
                                ;
005734  042766  0000000 000006          BIC     #CFLAG,6(SP)    ;Clear C-flag in PS on stack
                                ;
                                ;  Make sure the region index number is valid
                                ;
005742  020027  0000000                 CMP     R0,#MAXSRD      ;Is region index valid?
005746  103404                          BLO     1$              ;Br if ok
005750  052766  0000000 000006          BIS     #CFLAG,6(SP)    ;Set carry flag for return
005756  000423                          BR      9$
                                ;
                                ;  Get the PAR index
                                ;
005760  116005  0000000         1$:     MOVB    SR$PX(R0),R5    ;Get PAR index number
                                ;
                                ;  Load the PAR value
                                ;
005764  006300                          ASL     R0              ;Get word table index
005766  016004  0000000                 MOV     SR$PAR(R0),R4   ;Get value to load into PAR registers
005772  010465  0000000                 MOV     R4,RPAR(R5)     ;Shared run-time mapping for this PAR
005776  010465  0000000                 MOV     R4,CUPAR0(R5)   ;Set PAR value in context block
006002  010465  0000000                 MOV     R4,UPAR0(R5)    ;Set PAR value in hardware register
                                ;
                                ;  Load the PDR value
                                ;
006006  016004  0000000                 MOV     SR$PDR(R0),R4   ;Get value to load into PDR registers
006012  010465  0000000                 MOV     R4,RPDR(R5)     ;Shared run-time mapping for this PDR
006016  010465  0000000                 MOV     R4,CUPDR0(R5)   ;Set PDR value in context block
006022  010465  0000000                 MOV     R4,UPDR0(R5)    ;Set PDR value in hardware register
                                ;
                                ;  Finished
                                ;
006026  012605                  9$:     MOV     (SP)+,R5
006030  012604                          MOV     (SP)+,R4
006032  000002                          RTI                     ;Return from TRAP instruction
```

```
                                        .SBTTL  UEXINT -- Unexpected interrupt
                                ;-------------------------------------------------------------
                                ;  An interrupt occured at an unexpected location.
                                ;  On entry to UEXINT the interrupt vector address is encoded
                                ;  in the PS that was set by the interrupt vector.
                                ;  The address has the two low-order bits removed (they are assumed
                                ;  to be zero) and the remainder of the address stored in the
                                ;  PS fields priority and n-z-v-c (note the T field is not used).
                                ;
006034  013703  0000000         UEXINT: MOV     @#PSW,R3        ;GET CURRENT PROCESSOR STATUS VALUE
006040  006303                          ASL     R3              ;ADD ONE LOW-ORDER ZERO BIT
006042  010302                          MOV     R3,R2           ;COPY VALUE
006044  042703  177077                  BIC     #^C700,R3       ;MASK OUT ALL BUT PRIO FIELD (SHIFTED)
006050  042702  177741                  BIC     #^C36,R2        ;MASK OUT ALL BUT N-Z-V-C FIELDS (SHIFTED)
006054  006302                          ASL     R2              ;ALIGN LOW-ORDER FIELD WITH HIGH-ORDER
006056  050203                          BIS     R2,R3           ;COMBINE LOW- AND HIGH-ORDER FIELDS
006060                                  DIE     #EM$UEI,R3      ;SYSTEM CRASH -- ARG VALUE = INT LOCATION
                                ;
                                ;  Enter at UEXRTN if we should ignore unexpected interrupts
                                ;
006076  000002                  UEXRTN: RTI                     ;Return from interrupt -- Ignore it
                                ;
                                ;  Memory parity error
                                ;
006100                          MEMPAR: DIE     #EM$MPR         ;MEMORY PARITY ERROR
                                ;
                                ;  Jump occured to location 0
                                ;
006112                          JMP0:   DIE     #EM$JM0         ;FATAL SYSTEM HALT
```

```
                                ;-----------------------------------------------------------------------
                                ;  CHKUSP is called to determine if the current user-mode stack pointer (SP)
                                ;  is valid.  The SP is checked to make sure it is even and >400.
                                ;  CKUSP2 is an alternate entry point that is a little faster if it is
                                ;  already known that the previous mode was user.
                                ;
                                ;  Outputs:
                                ;   C-flag cleared if SP is valid, Set if invalid.
                                ;   R5 = abort code for invalid stack if error detected, otherwise unaltered.
                                ;
006124                          CHKUSP:
                                ;
                                ;  Return with carry cleared if previous mode is not user
                                ;
006124  032737  0000000 0000000         BIT     #UPMODE,@#PSW   ;PREVIOUS MODE = USER?
006132  001002                          BNE     CKUSP2          ;BR IF YES
006134  000241                          CLC                     ;CLEAR CARRY FOR RETURN
006136  000207                          RETURN
                                ;
                                ;  Get the user mode SP
                                ;
006140  106506                  CKUSP2: MFPD    SP              ;GET USER-MODE SP
                                ;
                                ;  Make sure the stack is in the right range
                                ;
006142  021627  000400                  CMP     (SP),#400       ;DID A STACK OVERFLOW OCCUR?
006146  103405                          BLO     2$              ;BR IF YES
006150  021637  0000000                 CMP     (SP),UHIMEM     ;IS STACK ADDRESS TOO HIGH?
006154  103002                          BHIS    2$              ;BR IF TOO HIGH
                                ;
                                ;  Make sure the stack address is even
                                ;
006156  006016                          ROR     (SP)            ;IS THE STACK ADDRESS EVEN?
006160  103003                          BCC     3$              ;BR IF EVEN -- OK
                                ;
                                ;  User's stack pointer is invalid
                                ;
006162  012705  000011          2$:     MOV     #11,R5          ;LOAD ABORT ERROR CODE VALUE
006166  000261                          SEC                     ;SIGNAL ERROR ON RETURN
006170  005226                  3$:     INC     (SP)+           ;CLEAN OFF STACK (DON'T ALTER C-FLAG)
                                ;
                                ;  Finished
                                ;
006172  000207                  9$:     RETURN
```

```
                                        .SBTTL  CLKINT -- Clock interrupt routine
                                ;-------------------------------------------------------------------------
                                ;  CLKINT is the interrupt service routine for clock interrupts.
                                ;  It is entered directly from the interrupt (priority = 7).
                                ;
006174                          CLKINT:
                                ;
                                ;  If this is a PRO-350, access the CSR2 clock register to reenable
                                ;  the interrupt.
                                ;
006174  105737  000000                  TSTB    PROFLG          ;Is this a PRO-350?
006200  001411                          BEQ     1$              ;Br if not
006202  005737  000000                  TST     @#PCCCR2        ;Acknowledge the interrupt
                                ;
                                ;  Ignore every 16'th clock tick on a PRO-350 so that the effective
                                ;  clock rate will be 60 Hz.
                                ;
006206  105337  000014'                DECB    PROSKP          ;Is this the 16'th tick?
006212  001004                          BNE     1$              ;Br if not
006214  112737  000020  000014'        MOVB    #16.,PROSKP     ;Reset the counter
006222  000420                          BR      CLKRTI          ;Ignore this clock tick
                                ;
                                ;  Count another clock tick
                                ;
006224  005237  000100'         1$:     INC     TIKCNT          ;ANOTHER TICK HAS OCCURED
006230  003015                          BGT     CLKRTI          ;BR IF STILL PROCESSING LAST TICK
                                ;
                                ;  We are not reentering the clock processing routine.
                                ;  Save interrupted PC & PS for performance monitor to use.
                                ;
006232  011637  000102'                 MOV     (SP),CLKPC      ;INTERRUPTED PC
006236  016637  000002  000104'         MOV     2(SP),CLKPS     ;INTERRUPTED PS
                                ;
                                ;  Drop priority to 6 then fork.
                                ;
006244  004537  006404'                 JSR     R5,INTEN        ;DROP RUNNING PRIORITY TO 6
006250  000040                          .WORD   40              ;MASK TO SET PRIO TO 6
006252  004537  007332'                 JSR     R5,FORK         ;NOW FORK TO GET TO PRIORITY 0
006256  000000                          .WORD   FP$CKT          ;Specify fork priority
                                ;
                                ;  CLKRUN is entered to perform clock servicing in the system mapped region.
                                ;
006260  000137  000000                  JMP     CLKRUN          ;ENTER THE SYSTEM MAPPED REGION
                                ;
                                ;  The clock processing routine is still running from the last tick.
                                ;  Don't reenter it.
                                ;
006264  000002                  CLKRTI: RTI                     ;RETURN FROM INTERRUPT QUICKLY
```

```
                                        .SBTTL  ENSYS  -- Enter system state
                                ;-----------------------------------------------------------------------
                                ;  ENSYS is called to enter system state.  What this consists of is switching
                                ;  to the interrupt stack and saving the kernel PAR6 value.
                                ;  Basically, what we do is fake an interrupt and then do a .INTEN and .FORK.
                                ;  On return from ENSYS we are running in system state at fork level using
                                ;  the interrupt stack.
                                ;  To exit from system state, do a RETURN.
                                ;
                                ;  The form of the call to ENSYS is:
                                ;
                                ;       MOV     #return_address,R0      ;Get return address
                                ;       CALL    ENSYS                   ;Enter system state
                                ;       .WORD   fork_priority           ;Fork priority level to run at
                                ;
                                ;  Inputs:
                                ;   R0 = Address of routine to be jumped to when a RETURN is done to exit
                                ;        from system state.
                                ;
                                ;  Outputs:
                                ;   R0, R4 and R5 are destroyed.
                                ;   On return we are in system state running on the interrupt stack.
                                ;   All registers except R0 are preserved across the ENSYS.
                                ;
006266  012604                  ENSYS:  MOV     (SP)+,R4        ;Get return address to R4
                                ;
                                ;  Check to see if we are already running in system state.
                                ;
006270  105737  000010'                 TSTB    STKLVL          ;Are we already in system state?
006274  002017                          BGE     10$             ;Br if already in system state
                                ;
                                ;  We are not currently in system state.
                                ;  Put PC & PS on the stack to make it look like an interrupt occured.
                                ;
006276  013746  000000                  MOV     @#PSW,-(SP)     ;PS
006302  010046                          MOV     R0,-(SP)        ;PC -- Return here when we exit system state
                                ;
                                ;  Do .INTEN to enter system state.
                                ;
006304                                  DISABL                  ;** Disable interrupts **
006312  004537  006404'                 JSR     R5,INTEN        ;Enter system state
006316  000000                          .WORD   0               ;Priority = 7
                                ;
                                ;  We are now running in system state on the interrupt stack.
                                ;  The processor priority level is 7.
                                ;  Now do a .FORK so that we will not hold out interrupts.
                                ;
006320  012437  006330'                 MOV     (R4)+,1$        ;Set fork priority
006324  004537  007332'                 JSR     R5,FORK         ;Do a fork
006330  000000                  1$:     .WORD   0               ;Fork priority is stored here
                                ;
                                ;  We are now running in system state, fork level.
                                ;  Return to caller in system state.
                                ;  Caller should do a RETURN to exit from system state.
                                ;
006332  000114                  3$:     JMP     (R4)            ;Call calling routine in system state
                                ;
```

```
                                        ;  ENSYS was called while already running in system state.
                                        ;  Save context and set up stack so we will restore it on return.
                                        ;
006334  010046                  10$:     MOV     R0,-(SP)          ;Set ultimate return address
006336  010146                           MOV     R1,-(SP)
006340  010246                           MOV     R2,-(SP)
006342  010346                           MOV     R3,-(SP)
006344  013746  0000000                  MOV     @#KPAR6,-(SP)     ;Save kernel PAR 6
006350  013746  0000000                  MOV     @#KPAR5,-(SP)     ;Save kernel PAR 5
006354  012746  006364'                  MOV     #11$,-(SP)        ;Set address of routine for ENSYS exit
006360  000164  000002                   JMP     2(R4)             ;Enter user's routine
                                        ;
                                        ;  Finished with routine in system state.
                                        ;  Drop down a level.
                                        ;
006364  012637  0000000           11$:     MOV     (SP)+,@#KPAR5     ;Restore kernel PAR 5
006370  012637  0000000                  MOV     (SP)+,@#KPAR6     ;Restore kernel PAR 6
006374  012603                           MOV     (SP)+,R3
006376  012602                           MOV     (SP)+,R2
006400  012601                           MOV     (SP)+,R1
006402  000207                           RETURN                    ;Return
```

```
                                        .SBTTL  INTEN  -- Interrupt entry processing
                                ;-----------------------------------------------------------------------
                                ;  INTEN performs the RT-11 .INTEN function which is used to begin
                                ;  interrupt processing.  The form of the call to INTEN is:
                                ;
                                ;       JSR     R5,INTEN
                                ;       .WORD   <^CPriority to run at>&340
                                ;
                                ;  INTEN switches to the TSX interrupt stack and then calls the calling
                                ;  routine back as a coroutine.
                                ;  When the interrupt processing task completes, it exits back to
                                ;  INTEN by doing an  RTS PC.
                                ;  Before returning from interrupt processing INTEN calls any routines
                                ;  queued as a result of .FORK requests and also may call the job scheduler
                                ;  if any job scheduling event occured during interrupt processing.
                                ;
                                ;  There are three "level indicators" that indicate the processing state.
                                ;  INTLVL indicates the hardware interrupt level.
                                ;  STKLVL indicates if we are running on the interrupt stack.
                                ;  FRKPRI indicates the current fork processing priority.
                                ;  The initial (non-interrupt, non-fork) value for all three is -1.
                                ;    If INTLVL >= 0 we are in an interrupt routine.
                                ;    If INTLVL < 0 we are not at interrupt level but may be at fork level.
                                ;    If FRKPRI > 0 we are at fork level.
                                ;
006404  010446                  INTEN:  MOV     R4,-(SP)        ;R5 is already on stack, save R4 too.
                                ;
                                ;  We should already be running at processor priority level 7.
                                ;  However, make sure we are at level 7.
                                ;
006406                                  DISABL                  ;;;** Disable interrupts **
                                ;
                                ;  Increment interrupt level counter.
                                ;
006414  105237  000007'                 INCB    INTLVL          ;;;Increment interrupt level counter
006420  105237  000010'                 INCB    STKLVL          ;;;Are we already running on interrupt stack?
006424  003004                          BGT     1$              ;;;Br if yes - Interrupting another int/fork
                                ;
                                ;  We were at level 0 when the interrupt occured.
                                ;  Save user's stack pointer and switch to TSX interrupt stack.
                                ;
006426  010637  000120'                 MOV     SP,USP          ;;;Save user's stack pointer
006432  013706  000022'                 MOV     INTSTK,SP       ;;;Switch to interrupt stack
                                ;
                                ;  Drop running priority to that requested by caller.
                                ;
006436  013746  000122'         1$:     MOV     INTPRI,-(SP)    ;;;Save current running priority
006442  113746  000011'                 MOVB    FRKPRI,-(SP)    ;;;Save current fork level priority
006446  112737  000000  000011'        MOVB    #0,FRKPRI       ;;;Not at fork level (preserve C-bit)
006454  013746  0000000                 MOV     @#KPAR6,-(SP)   ;;;Save kernel-mode PAR6 register
006460  013746  0000000                 MOV     @#KPAR5,-(SP)   ;;;Save kernel-mode PAR5 register
006464  011537  000122'                 MOV     (R5),INTPRI     ;;;Set new priority
006470  042537  0000000                 BIC     (R5)+,@#PSW     ;Drop running priority
                                ;
                                ;  We are now running at the requested priority.
                                ;  Call our caller back as a coroutine.
                                ;
```

```
     58 006474  004710                        CALL    @R5             ;Call caller as a coroutine
     59 006476                          INTENX:                       ;Must immediately follow CALL @R5
```

```
                                        ;
                                        ;  Interrupt processing routine is finished.
                                        ;
                                        ;  See if we are returning to level 0 or to a lower level interrupt routine.
                                        ;
006476                          INTXIT: DISABL                  ;** Disable interrupts **
006504  012637  0000000                 MOV     (SP)+,@#KPAR5   ;Restore kernel-mode PAR5 register
006510  012637  0000000                 MOV     (SP)+,@#KPAR6   ;Restore kernel-mode PAR6 register
006514  112637  000011'                MOVB    (SP)+,FRKPRI    ;Restore fork processing priority
006520  012637  000122'                 MOV     (SP)+,INTPRI    ;Reset interrupt priority
006524  105337  000007'                 DECB    INTLVL          ;Are we returning to level 0?
006530  002405                         BLT     1$              ;Br if yes
                                        ;
                                        ;  We are about to return to a lower-level interrupt.
                                        ;  We go back to lower level interrupt routines before we check
                                        ;  for pending fork requests.  This is done to give all interrupt routines
                                        ;  priority over all fork routines.
                                        ;
006532  105337  000010'                 DECB    STKLVL          ;We are going down one level on the stack
006536  012604                         MOV     (SP)+,R4
006540  012605                         MOV     (SP)+,R5
006542  000002                         RTI                     ;Continue processing lower-priority interrupt
                                        ;
                                        ;  We are returning to level 0.
                                        ;
                                        ;  See if there are any pending fork queue requests.
                                        ;
006544  010346                  1$:    MOV     R3,-(SP)
006546                           2$:    DISABL                  ;** Disable interrupts **
006554  013703  000124'                 MOV     FRKCQE,R3       ;Are there any pending fork requests?
006560  001515                         BEQ     6$              ;Br if not
006562  126337  0000000 000011'        CMPB    FQ$PRI(R3),FRKPRI;Is pending request higher prio than current?
006570  101511                         BLOS    6$              ;Br if not
                                        ;
                                        ;   There is a fork request that needs to be processed
                                        ;
006572  113746  000011'                MOVB    FRKPRI,-(SP)    ;Save current fork priority
006576  116337  0000000 000011'        MOVB    FQ$PRI(R3),FRKPRI ;Set current fork priority
006604  016337  0000000 000124'        MOV     FQ$LNK(R3),FRKCQE ;Remove fork block from pending list
006612                                  ENABL                   ;** Enable interrupts **
                                        ;
                                        ;  Save current context before entering the fork routine
                                        ;
006620  013746  000132'                 MOV     CURFRK,-(SP)    ;Address of currently running fork routine
006624  010046                         MOV     R0,-(SP)
006626  010146                         MOV     R1,-(SP)
006630  010246                         MOV     R2,-(SP)
006632  013746  0000000                 MOV     @#KPAR6,-(SP)   ;Save kernel-mode PAR6 register
006636  013746  0000000                 MOV     @#KPAR5,-(SP)   ;Save kernel-mode PAR5 register
                                        ;
                                        ;  Return fork request block to the free list
                                        ;
006642                                  DISABL                  ;** Disable interrupts **
006650  013763  0000000 0000000         MOV     FREFRK,FQ$LNK(R3);Put fork block back on free list
006656  010337  0000000                 MOV     R3,FREFRK
                                        ;
                                        ;  See if fork request has been cancelled
```

```
                                        ;
006662  016337  0000000 0000006         MOV     FQ$PA5(R3),@#KPAR5 ;Set mapping for kernel PAR 5
006670  016337  0000000 0000006         MOV     FQ$PA6(R3),@#KPAR6 ;Set mapping for kernel PAR 6
006676  016304  0000000               MOV     FQ$UFB(R3),R4   ;Get address of FQ$LNK in user's fork block
006702  001403                        BEQ     7$              ;Br if user did not specify a fork block
006704  005024                        CLR     (R4)+           ;Say fork request has been processed
006706  005714                        TST     (R4)            ;Has fork request been cancelled? (FQ$RTN)
006710  001422                        BEQ     8$              ;Br if cancelled
                                ;
                                ;  Set up context based on information in fork block
                                ;
006712  016304  0000000         7$:     MOV     FQ$R4(R3),R4    ;Restore R4 & R5 for fork routine
006716  016305  0000000               MOV     FQ$R5(R3),R5
006722  016302  0000000               MOV     FQ$R2(R3),R2
006726  016301  0000000               MOV     FQ$R1(R3),R1
006732  016300  0000000               MOV     FQ$RTN(R3),R0   ;Get address of fork routine to be called
006736  010037  000132'              MOV     R0,CURFRK       ;Remember address of current fork routine
006742  016303  0000000               MOV     FQ$R3(R3),R3
006746                                ENABL                   ;** Enable interrupts **
                                ;
                                ;  Call the fork routine
                                ;
006754  004710                        CALL    @R0             ;Call routine at fork level
                                ;
                                ;  Restore context
                                ;
006756                          8$:     DISABL                  ;Make sure interrupts are disabled
006764  012637  0000000               MOV     (SP)+,@#KPAR5   ;Restore kernel-mode PAR5 register
006770  012637  0000006               MOV     (SP)+,@#KPAR6   ;Restore kernel-mode PAR6 register
006774  012602                        MOV     (SP)+,R2
006776  012601                        MOV     (SP)+,R1
007000  012600                        MOV     (SP)+,R0
007002  012637  000132'              MOV     (SP)+,CURFRK    ;Address of currently running fork routine
007006  112637  000011'              MOVB    (SP)+,FRKPRI    ;Restore fork priority
                                ;
                                ;  See if there are more pending fork requests
                                ;
007012  000655                        BR      2$              ;See if there are more fork requests to do
                                ;
                                ;  We have processed all fork queue requests.
                                ;
007014  012603                  6$:     MOV     (SP)+,R3
                                ;
                                ;  Switch back to user's stack.
                                ;
007016  105337  000010'               DECB    STKLVL          ;Going down one level on interrupt stack
007022  002002                        BGE     9$              ;Br if still more levels on int stack
007024  013706  000120'               MOV     USP,SP          ;Switch back to user's stack
                                ;
                                ;  Completed interrupt processing
                                ;
007030  012604                  9$:     MOV     (SP)+,R4
007032  012605                        MOV     (SP)+,R5        ;R5 was saved by  JSR R5,INTEN
```

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

```
     58 007212  052737  0000000 0000000          BIS     #UPMODE,@#PSW   ;MAKE SURE PREVIOUS MODE = USER
     59 007220  062701  000004                  ADD     #4,R1           ;POINT TO CELL WHERE OLD PS IS TO BE STORED
     60 007224  016646  000004                  MOV     4(SP),-(SP)     ;GET OLD PS VALUE
     61 007230  106611                          MTPD    (R1)            ;STORE OLD PS IN USER'S INFO BLOCK
     62 007232  016646  000002                  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  000002                  MOV     (SP)+,2(SP)     ;SET NEW PC FOR RETURN
     66 007246  012601                          MOV     (SP)+,R1
     67                                ;
     68                                ;  See if a system stack overflow occured
     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$SOF,#2      ;INTERRUPT STACK (INTSTK) OVERFLOW
     76                                ;
     77                                ;  Return to the user.
     78                                ;
     79 007330  000002                 2$:     RTI                     ;RETURN FROM INTERRUPT OR EMT PROCESSING
```

```
                                        .SBTTL  FORK    -- Queue a fork request
                                ;-----------------------------------------------------------------------
                                ;  FORK is called to queue a fork request.
                                ;  Note that INTEN must have been called before FORK is called and that
                                ;  nothing may be pushed on the stack between the INTEN call and the FORK.
                                ;  A fork request is held until the last active interrupt routine is ready
                                ;  to return to the job that was originally interrupted then the fork
                                ;  requests are processed in order by priority and, within the same priority,
                                ;  by the order in which they were queued.
                                ;  The form of the call to FORK is:
                                ;
                                ;       JSR     R5,FORK
                                ;       .WORD   <forkblock-.> or <priority>
                                ;
                                ;  The TSX fork routine differs from the RT-11 fork routine in that TSX
                                ;  uses an internal set of fork request blocks rather than using blocks
                                ;  provided by the caller of FORK.
                                ;  The word following the JSR R5,FORK may contain the address of a user
                                ;  fork block, or it may contain a priority value in the range 1 to 127
                                ;  which becomes the fork processing priority, or it may be zero (0) in which
                                ;  case a default fork priority (FP$DEF) is used.
                                ;
                                ;
007332  010446                  FORK:   MOV     R4,-(SP)
                                ;
                                ;  Get a free fork block from the free list.
                                ;
007334  004737  007442'                 CALL    FRKGET          ;Get a free fork block
                                ;
                                ;  We got a free fork block.  R4 = Address of block.
                                ;  Set up information in fork request block.
                                ;
007340  012664  0000000                 MOV     (SP)+,FQ$R4(R4) ;Save R4 in fork block
007344  012664  0000000                 MOV     (SP)+,FQ$R5(R4) ;And R5
007350  010164  0000000                 MOV     R1,FQ$R1(R4)    ;Save other registers
007354  010264  0000000                 MOV     R2,FQ$R2(R4)
007360  010364  0000000                 MOV     R3,FQ$R3(R4)
007364  012502                        MOV     (R5)+,R2        ;Get addr of user's fork block or fork prio.
007366  001416                        BEQ     4$              ;Br if no user fork block or specified prio.
007370  003006                        BGT     5$              ;Br if offset to a fork block
007372  020227  0000000               CMP     R2,#FP$MAX      ;Is this a priority or an address?
007376  101003                        BHI     5$              ;Br if it is the address of a user fork block
007400  110264  0000000               MOVB    R2,FQ$PRI(R4)   ;Set fork priority
007404  000407                        BR      4$
007406  060502                  5$:   ADD     R5,R2           ;Get address of FQ$RTN in user's fork block
007410  162702  000002                SUB     #2,R2           ;Get pointer to start of user's fork block
007414  010264  0000000               MOV     R2,FQ$UFB(R4)   ;Save pointer to user's fork block
007420  010522                        MOV     R5,(R2)+        ;Make FQ$LNK in user's fork block non-zero
007422  010512                        MOV     R5,(R2)         ;Make FQ$RTN in user's fork block non-zero
007424  016402  0000000           4$:   MOV     FQ$R2(R4),R2    ;Recover R2
007430  010564  0000000               MOV     R5,FQ$RTN(R4)   ;Save address of routine to call
                                ;
                                ;  Add fork block to queue of waiting fork blocks
                                ;
007434  004737  007534'                 CALL    FORKQ           ;Queue the fork request
                                ;
                                ;  Finished -- Return to INTEN routine which will check for fork requests
```

```
      58                                        ;
      59 007440  000207                           RETURN                    ;Return to INTEN routine
```

```
                                        .SBTTL  FRKGET -- Get a free Fork block
                                ;-----------------------------------------------------------------------
                                ;  FRKGET is called to get a free fork block.
                                ;  If no free fork blocks are available, a system crash occurs.
                                ;
                                ;  Outputs:
                                ;   R4 = Address of fork block.
                                ;   The following fields are initialized in the fork block:
                                ;     FQ$PRI = Default fork priority (FP$DEF)
                                ;     FQ$PA5 = Current KPAR5 mapping
                                ;     FQ$PA6 = Current KPAR6 mapping
                                ;     FQ$UFB = 0
                                ;
007442                          FRKGET:
                                ;
                                ;  Get a fork block from the free list
                                ;
007442                                  DISABL                  ;** Disable interrupts **
007450  013704  0000000                 MOV     FREFRK,R4       ;Get address of a free fork block
007454  001005                          BNE     3$              ;Br if fork block is available
007456                                  DIE     #EM$FRK         ;System halt if no free fork blocks
007470  016437  0000000 0000000 3$:     MOV     FQ$LNK(R4),FREFRK;Remove fork block from the free list
007476                                  ENABL                   ;** Enable interrupts **
                                ;
                                ;  Set default values in the fork block
                                ;
007504  112764  0000000 0000000        MOVB    #FP$DEF,FQ$PRI(R4);Set default priority
007512  013764  0000000 0000000        MOV     @#KPAR5,FQ$PA5(R4);Save KPAR5 mapping
007520  013764  0000000 0000000        MOV     @#KPAR6,FQ$PA6(R4);And PAR6
007526  005064  0000000                CLR     FQ$UFB(R4)      ;No user fork block
                                ;
                                ;  Finished
                                ;
007532  000207                          RETURN
```

```
                                        .SBTTL  FORKQ  -- Queue a fork request
                                ;-----------------------------------------------------------------------
                                ;  FORKQ is called to place a fork request block on the fork-pending
                                ;  list. The queue entry is entered in the request queue based on its
                                ;  priority as stored in the FQ$PRI field of the fork block.
                                ;
                                ;  Inputs:
                                ;   R4 = Address of fork request block to be queued.
                                ;
007534  010246                  FORKQ:  MOV     R2,-(SP)
007536  010346                          MOV     R3,-(SP)
                                ;
                                ;  Do a linear search down the list of current fork entries and look
                                ;  for the correct position to insert our new entry based on its priority.
                                ;
007540  012703  000000C                 MOV     #FRKCQE-FQ$LNK,R3;Get pointer to dummy fork block at head
007544                                  DISABL                  ;** Disable interrupts **
007552  016302  0000000         1$:     MOV     FQ$LNK(R3),R2   ;Get address of following fork block
007556  001406                          BEQ     2$              ;Br if we should insert at end of chain
007560  126462  0000000 0000000         CMPB    FQ$PRI(R4),FQ$PRI(R2);Is next entry of lower priority?
007566  101002                          BHI     2$              ;Br if yes
007570  010203                          MOV     R2,R3           ;Link forward to next entry
007572  000767                          BR      1$              ;And continue searching for insert point
                                ;
                                ;  Insert following the entry pointed to by R3 and before the entry
                                ;  pointed to by R2
                                ;
007574  010463  0000000         2$:     MOV     R4,FQ$LNK(R3)   ;Make previous entry point to us
007600  010264  0000000                 MOV     R2,FQ$LNK(R4)   ;Make new entry point to following one
                                ;
                                ;  Finished
                                ;
007604                                  ENABL                   ;** Enable interrupts **
007612  012603                          MOV     (SP)+,R3
007614  012602                          MOV     (SP)+,R2
007616  000207                          RETURN
```

```
                                        .SBTTL  SYNCH  -- Queue a synch request
                                ;-----------------------------------------------------------------------
                                ;  SYNCH is called to queue a synch request.
                                ;  A synch request can be made by a handler when it reaches a point where
                                ;  it must run in user state.  The call to SYNCH simply queues a synch
                                ;  request for the job.
                                ;  The synch routine is called from the job scheduler at the point where
                                ;  the job is selected and set up ready to run in user state.
                                ;
                                ;  Inputs:
                                ;   R4 = Address of 7-word synch control block.
                                ;   R5 = Address following synch call  (JSR  R5,SYNCH)
                                ;
007620  010046                  SYNCH:  MOV     R0,-(SP)
007622  010146                          MOV     R1,-(SP)
007624  010246                          MOV     R2,-(SP)
007626  010446                          MOV     R4,-(SP)
007630  005764  0000000                 TST     SN$RTN(R4)      ;IS THIS SYNCH BLOCK FREE?
007634  001074                          BNE     9$              ;BR IF NOT
007636  016402  0000000                 MOV     SN$JOB(R4),R2   ;GET JOB NUMBER FROM SYNCH BLOCK
007642  042702  177400                  BIC     #^C<377>,R2     ;KILL SIGN EXTENSION FROM HANDLER MOVB
007646  001467                          BEQ     9$              ;ZERO IS INVALID
007650  006302                          ASL     R2              ;CONVERT TO WORD TABLE INDEX #
007652  020227  0000000                 CMP     R2,#LSTSL       ;IS IT VALID LINE #?
007656  101063                          BHI     9$              ;BR IF NOT
007660  032762  0000000 0000000         BIT     #$DILUP,LSW(R2) ;IS JOB LOGGED ON?
007666  001457                          BEQ     9$              ;BR IF NOT
                                ;
                                ;  Synch block looks good.
                                ;  Queue a completion request for the job.
                                ;
007670  004737  0000000                     CALL    GETRTQ          ;Get a free queue element (address in R1)
007674  152761  000000C 0000000           BISB    #<QF$SYN!QF$SCR>,CQ$FLG(R1);Synch routine, call in kernel mode
007702  110261  0000000                     MOVB    R2,CQ$JOB(R1)   ;Set job index number for compl routine
007706  012700  0000000                     MOV     #S$TWFN,R0      ;Get compl priority for non-interactive jobs
007712  005762  0000000                     TST     LITIME(R2)      ;Is this job interactive?
007716  001402                            BEQ     1$              ;Br if not
007720  012700  0000000                     MOV     #S$HICP,R0      ;Get compl prio for interactive jobs
007724  110061  0000000             1$:     MOVB    R0,CQ$RNS(R1)   ;Set execution state for compl routine
007730  116261  0000000 0000000           MOVB    LPRI(R2),CQ$PRI(R1);Set execution priority
007736  112761  0000000 0000000           MOVB    #CP$SYN,CQ$CP(R1);Set compl routine class priority
007744  005725                            TST     (R5)+           ;Point to successful return point for synch
007746  010561  0000000                     MOV     R5,CQ$RTN(R1)   ;Set address of routine to call
007752  010564  0000000                     MOV     R5,SN$RTN(R4)   ;Set flag saying synch block is busy
007756  016461  0000000 0000000           MOV     SN$ID(R4),CQ$R0(R1);Set synch ID value to be passed in R0
007764  010461  0000000                     MOV     R4,CQ$R1(R1)    ;Set address of cell to be cleared by call
007770  062761  0000000 0000000           ADD     #CQ$RTN,CQ$R1(R1)
007776  013761  0000000 0000000           MOV     @#KPAR5,CQ$PA5(R1);Set PAR 5 mapping to use for synch routine
010004  010104                            MOV     R1,R4           ;Get address of completion request block to R4
010006  004737  0000000                     CALL    QCOMPL          ;Queue a completion request for the job
                                ;
                                ;  Successful completion of synch request.
                                ;  Do a RTS PC to return from handler interrupt.
                                ;  Synch routine will be called from job scheduler.
                                ;
010012  012604                          MOV     (SP)+,R4
010014  012602                          MOV     (SP)+,R2
```

```
      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
```

```
                                        .SBTTL  QUNSIG -- Signal quantum expiration
                                ;-----------------------------------------------------------------------
                                ;  QUNSIG is called when a time-slice quentum expires to see if the
                                ;  user wants to have the terminal bell rung to signal that the time
                                ;  slice has expired.
                                ;  The SET SIGNAL command is used to control this feature.
                                ;
                                ;  Form of the call:
                                ;       JSR     R0,QUNSIG
                                ;       .WORD   $SGxxx
                                ;
                                ;  Where $SGxxx is the flag in LSW8 that controls which quantum expired.
                                ;
                                ;  Inputs:
                                ;   R1 = Job index number.
                                ;
010040                          QUNSIG:
                                ;
                                ;  See if used wants to be notified about this quantum running out
                                ;
010040  032061  0000008                 BIT     (R0)+,LSW8(R1)  ;Is notification wanted?
010044  001407                          BEQ     9$              ;Br if not
                                ;
                                ;  Ring terminal bell
                                ;
010046  010046                          MOV     R0,-(SP)
010050  012700  000007                  MOV     #7,R0           ;Get bell character
010054                                  QCALL   QUECHR          ;Send bell to terminal
010062  012600                          MOV     (SP)+,R0
                                ;
                                ;  Finished
                                ;
010064  000200                  9$:     RTS     R0              ;Return
```

```
                                        .SBTTL  SYSHLT -- Fatal system halt
                                ;-----------------------------------------------------------------------
                                ;  SYSHLT is entered when a fatal system error is detected.
                                ;  An error message is printed and the system is halted.
                                ;
                                ;  Inputs:
                                ;   (SP) = Address of call to SYSHLT.
                                ;   DIEMSG = Address of error message to print.
                                ;   DIEARG = Argument value to print with error message.
                                ;
010066  013737  0000000 000034' SYSHLT: MOV     @#KPAR5,TRPAR5  ;Save current kernel par 5 mapping
010074                          SYSHL1: DISABL                  ;DISABLE ALL INTERRUPTS
010102  012637  000042'                 MOV     (SP)+,DIEPC     ;GET ADDRESS OF CALL TO SYHLT
010106  010637  000044'                 MOV     SP,DIESP        ;Save stack pointer at time of crash
010112  020627  0000000                 CMP     SP,#VPAR5       ;Are we running on context blk stack?
010116  103402                         BLO     2$              ;Br if not
010120  012706  001000                  MOV     #SS,SP          ;RUN ON SYSTEM STACK NOW
010124  000005                  2$:     RESET                   ;RESET ALL DEVICES
010126  052737  0000000 0000000         BIS     #MMENBL,@#SROMMR;MAKE SURE MEMORY MANAGEMENT ENABLED
010134  105737  0000000                 TSTB    MEM256          ;DOES MACHINE HAVE AT LEAST 256 KB?
010140  001403                         BEQ     1$              ;BR IF NOT
010142  052737  000000C 0000000         BIS     #EMMAP!IOMAP,@#SR3MMR ;TURN ON 22-BIT MEMORY MANAGEMENT
010150  000137  0000000          1$:    JMP     SYSDIE          ;ENTER OVERLAY TO PRINT ERROR MESSAGE
```

```
      1                                          .SBTTL  INIJMP -- Final system initialization
      2                                ;-----------------------------------------------------------------------
      3                                ;  Turn on kernel mode memory management and perform final system
      4                                ;  initialization.
      5                                ;
      6 010154                         INIJMP:
      7                                ;
      8                                ;  Now jump to EXCINI in TSMISC overlay to complete initialization
      9                                ;
     10 010154  000137  0000000                 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
```

Cross reference table (CREF V05.04)

```
$CTRLC    1-69      28-35
$DBGBK    1-53      41-25     41-29
$DEBUG    1-56      33-50
$DILUP    1-48      45-26
$DISCN    1-48      28-35
$FPUEX    1-51      33-87     41-19
$GEMAR    1-46      33-17
$INCOR    1-75       6-29
$INIT     1-48       7-10
$INKMN    1-50      14-23     14-120
$IOMAP    1-82      14-56     14-94
$KINIT    1-89      14-144
$MAPOK    1-92       7-43     14-146
$MLOCK    1-79      12-35     13-42     13-59
$NDMEM    1-77      21-14     21-16
$NLOCK    1-79      12-35
$NOABT    1-70      19-31     28-39
$NOUCR    1-56      19-129
$RDSAV    1-50      17-28     18-44
$SETCC    1-50       7-53      7-61
$SGHIO    1-58      30-38
$SGIIO    1-58      30-26
$SUSPN    1-52       7-17
$VIRJB    1-69      14-107
$VNOTT    1-49      19-172
$WDISP    1-49       7-78
ABRTOV    1-24      33-26*    33-35*
ADQ1      23-35     23-62#    24-36
ADQHD     23-56     23-63#
ADQMID    23-38     23-40     23-52#    24-39     24-41
ADQT      23-48     23-54#
ADQTL     23-47#    24-45
ADQXIT    23-58     23-64#
BASMAP     1-92     10-69     11-28     13-83     20-52     20-70     20-85
BELL       2-11#
BRKENT    1-56      33-54
BRKPT     1-36       4-10#
CCFLG     1-62       7-59
CFLAG     1-47      34-19     34-25
CHKABT    1-24      21-20     28-33#    41-42     41-44
CHKUSP    1-33      36-11#
CKUSP2    1-32      36-16     36-22#
CLKCNT    1-34       2-50#
CLKINT    1-21      37-6#
CLKPC     1-34       2-52#    37-31*
CLKPS     1-34       2-53#    37-32*
CLKRTI    1-21      37-21     37-26     37-48#
CLKRUN    1-102     37-43
CONFIG    1-73       1-77     17-43     18-23
CORUSR    1-71       5-34*     7-5*     15-29     17-16     19-160    20-19     27-77     28-34     33-16     33-49     33-86
          41-15
CP$SYN    1-60      45-41
CPLEMT    1-52      19-140
CQ$CP     1-59      19-25     19-72     19-89     45-41*
CQ$FLG    1-58      19-74     19-103    45-33*
CQ$JOB    1-54      45-34*
```

```
CQ$LNK    1-61     19-68    19-70    19-83    19-83*
CQ$PA5    1-61     19-104   45-48*
CQ$PRI    1-94     19-90    45-40*
CQ$RO     1-61     19-101   45-45*
CQ$R1     1-61     19-102   45-46*   45-47*
CQ$RNS    1-94     27-88    27-90    45-39*
CQ$RTN    1-61     19-100   45-43*   45-47
CR        2-9#
CS$EOF    1-96
CS$ERR    1-96
CUPAR0    1-105    14-35*   14-76*   14-136*  15-16    34-37*
CUPAR1    1-105    14-39*
CUPAR6    1-105    14-124*
CUPAR7    1-105    14-100*  14-114*
CUPDR0    1-106    14-37*   14-74*   14-88*   14-134*  15-18    34-44*
CUPDR1    1-106    14-41*
CUPDR6    1-106    14-126*
CUPDR7    1-106    14-102*  14-116*
CURCP     1-57     19-25    19-72    19-88    19-89*   19-161*  19-169   27-86    41-50
CURFRK    1-40      2-64#   40-44    40-74*   40-90*
CURVC     1-38      2-63#
CW$FPU    1-73      1-86    17-43    18-23
CXBBAS    1-42      2-81#
CXBJOB    1-42      2-27#
CXBMOV    1-42     22-13#
CXBOWN    1-42      2-26#
CXBSIZ    1-42      2-82#
CXTBAS    1-70     14-110
CXTBUF    1-42      2-80#   22-24
CXTPAG    1-41      2-77#    9-39    14-60    20-38
CXTPDR    1-41      2-78#   14-125   14-126
CXTRMN    1-41      2-79#   14-109   20-24
CXTWDS    1-41      2-76#
D$DMON    1-64     19-149
D.FLAG    1-64     19-149
DATIMH    1-29      2-55#
DATIML    1-29      2-54#
DBGBRK    1-53     41-32
DEQ       1-23     23-30    24-31    25-10#
DIEARG    1-21      2-39#    5-26*   35-17*   41-72*   41-75*
DIEMSG    1-21      2-38#    5-26*   33-68*   35-17*   35-25*   35-29*   41-72*   41-75*   43-21*
DIEPC     1-22      2-40#   47-13*
DIESP     1-22      2-41#   47-14*
DOCMPL    7-74     19-12#
DORTI    41-7      41-9     41-49    41-51    41-70#
DOSCHD    1-25      2-17#    6-24*   19-94*   19-187*  20-60*   23-68*   28-19*   41-37
DOSPNR   27-33     27-54#
DOSPNX   27-40     27-47    27-55#
DOTRMP    1-47     33-74
DTLX      1-32      2-34#
EM$FRK    1-67     43-21
EM$JMO    1-67     35-29
EM$MPR    1-67     35-25
EM$PFT    1-66     33-68
EM$SOF    1-33      5-26    41-72    41-75
EM$UEI    1-67     35-17
```

Cross reference table (CREF V05.04)

```
EMMAP     1-55      47-22
EMTCAD    1-52      19-135    19-137*
ENQHD     1-23      23-9#     27-46     27-53
ENQTL     1-23       7-24     24-10#    27-32     27-39     28-18     30-18     30-43     31-11
ENSYS     1-29      38-25#
ERRLOC    1-65      17-30     18-49*
EXCBUF    1-22       2-88#
EXCINI    1-28      48-10
EXCJOB    1-33       2-19#     6-13
EXEC      1-21       6-18#     6-38      7-25
FORCEX    1-23      28-10#
FORK      1-26      37-38     38-49     42-24#
FORKQ     1-40      42-55     44-10#
FP$CKT    1-72      37-39
FP$DEF    1-60      43-27
FP$MAX    1-60      42-41
FPTRAP    1-24      33-84#
FPTRPX    1-75      41-21
FPUUSE    1-38       2-24#    17-59*    18-26*    19-53     19-55*
FQ$LNK    1-80      40-39     40-54*    43-22     44-16     44-18     44-28*    44-29*
FQ$PA5    1-82      40-59     43-28*
FQ$PA6    1-80      40-60     43-29*
FQ$PRI    1-60      40-32     40-38     42-43*    43-27*    44-20     44-20
FQ$R1     1-84      40-72     42-35*
FQ$R2     1-84      40-71     42-36*    42-50
FQ$R3     1-84      40-75     42-37*
FQ$R4     1-80      40-69     42-33*
FQ$R5     1-80      40-70     42-34*
FQ$RTN    1-80      40-73     42-51*
FQ$UFB    1-83      40-61     42-47*    43-30*
FREFRK    1-62      40-54     40-55*    43-19     43-22*
FREIOQ    1-59       1-83
FREMEM    1-38      11-13#
FREPGS    1-92       9-22     10-66*    11-24*    13-93*    20-57*    20-86*
FRKCQE    1-37       2-61#    40-30     40-39*    44-16
FRKGET    1-40      42-28     43-14#
FRKPRI    1-40       2-22#    39-48     39-49*    40-9*     40-32     40-37     40-38*    40-91*
GETMEM    1-36       9-17#
GETRTQ    1-49      45-32
GTRDY     6-14       6-30      7-5#
GTSYMB    1-28      32-9#
HIMAP     1-92      10-30     12-28     13-31
INBSY     1-22       2-28#     8-11
INIJMP    1-25      48-6#
INITFL    1-28       2-23#
INITQO    1-63       4-6
INTEN     1-25      33-84     37-36     38-41     39-26#
INTENX    1-30      39-59#
INTLVL    1-21       2-20#    39-35*    40-11*
INTPRI    1-27       1-37      2-60#     6-37      7-6       19-84     19-167    23-69     26-23     32-17     32-29     39-47
         39-52*    40-10*    40-40     40-76     41-13     43-23     44-33
INTSND    1-39       2-33#    41-73
INTSTK    1-39       2-32#    39-43
INTXIT   40-6#
IOHLTM    1-48       7-21
IOMAP     1-55      47-22
```

Cross reference table (CREF V05.04)

```
IOPAGE   1-102    14-99    14-100
JMPO     1-32     35-29#
JSTKND   1-86      5-24
JSWLOC   1-65     17-32    18-47*
KMNCHN   1-27      2-45#
KMNPGS   1-26      2-44#
KMNSTK   1-26      2-42#
KMNSTR   1-26      2-43#
KPAR5    1-101    10-19    10-24*   10-80*   11-14    11-20*   11-35*   12-15    12-19*   12-44*   13-25    13-30*
        13-108*  16-29*   17-37    18-40*   19-46    19-104*  19-203*  20-17    20-18*   20-110*  22-15    22-19*
        22-39*   33-25    33-34    38-66    38-73*   39-51    40-7*    40-49    40-59*   40-85*   43-28    45-48
        47-11
KPAR6    1-78      7-29*   38-65    38-74*   39-50    40-8*    40-48    40-60*   40-86*   43-29
LBASE    1-76      9-36*   20-49    20-67    20-80
LBSPRI   1-57     19-171
LCMPL    1-59     19-16    19-68    27-82
LCXPAR   1-82      7-29     9-38*   14-112   14-122
LF       2-10#
LHIPCT   1-104    30-14    30-16*   30-30*   30-35    30-41*   31-13*
LIOCNT   1-48      7-19
LIOHLD   1-73      7-21*   19-76    27-84
LITIME   1-84     30-12    30-24*   45-36
LJSW     1-57     17-33*   18-46
LMEMIN   1-76      9-21    20-54*   20-55*   20-83*   20-84*   21-13*
LNBLKS   1-79     12-30    12-37    14-59    20-42    20-50    20-53*   20-68    20-81    20-82*   20-83    20-96
LNSBLK   1-103    12-31    12-38    20-55    20-84    20-98
LODMAP   7-47     15-8#
LOKSWP   1-32      2-48#
LOMAP    1-92     10-25    12-23    13-37    13-54
LPARBS   1-79      9-42*   14-62
LPRI     1-68     19-63    19-91    19-93*   19-184   19-186*  23-17    23-39    24-18    24-40    45-40
LQLINK   1-70      6-31    23-42    23-52    23-53*   23-54*   23-55*   23-57*   24-43    25-13    25-22    25-23
        25-27*   25-32*   25-35*   26-20
LQUAN    1-91     27-54*   31-12*
LSCCA    1-62      7-55
LSTATE   1-60      6-27    19-188   23-36    23-64*   24-37    25-36*   26-18    28-11
LSTSL    1-77     12-32    45-24
LSW      1-57      6-29     7-10     7-17    14-144   19-172   21-14    21-16*   28-35    33-87*   41-19    45-26
LSW11    1-50     17-28    18-44    33-17
LSW4     1-57      7-53     7-61*   14-23    14-120
LSW6     1-99      7-78    12-35    13-42    13-59    14-56    14-94
LSW7     1-92      7-43    14-146*
LSW8     1-58     46-21
LSW9     1-69     14-107   19-31    19-129   28-39    33-50    41-25    41-29*
MA$RGN   1-71     13-94
MAPPAR   1-52     10-24    11-20    12-19    13-30    20-18
MAPSYS   1-33     16-23#
MAPUSR   1-95      7-35    14-143*  15-29*
MAXSRD   1-47     34-23
MBFFLG   1-28      2-49#
MEM256   1-55     47-20
MEMPAR   1-31     35-25#
MEMSWP   1-32      2-47#   21-17*
MEMXPN   1-37     20-99    21-13#
MINCTR   1-34      2-58#
MINTIM   1-72      1-87
```

Cross reference table (CREF V05.04)

```
MMENBL    1-55      33-10     47-19
MRKTHD    1-25       2-46#
NEWUSR    1-50       7-12
NMUMB     1-90      32-16*
O.PAR     1-101     16-29
ODTTRP    1-29       2-35#    33-45
OUTBSY    1-22       2-29#     8-10
OVRADD    1-101     16-28
OVRHC     1-101      7-80     46-28
PO$DBG    1-51      41-27
PCCCR2    1-76      37-13
PKSTAT    5-32      17-15#    19-54
PMBASE    1-30       2-70#
PMCELS    1-31       2-74#
PMFLGS    1-30       2-69#
PMNBPC    1-30       2-72#
PMPAR     1-31       2-73#
PMRUN     1-31       2-75#
PMSIZE    1-97       2-74
PMTOP     1-30       2-71#
PMUSER    1-30       2-68#
PRIVCO    1-51      41-27
PROFLG    1-76      37-11
PROSKP    2-25#     37-18*    37-20*
PSW       1-78       6-23*     6-37*     7-6*      7-57*     16-25     16-26*    17-20     17-21*    18-17*    18-58*    18-59*
         19-67*    19-84*    19-134*   19-167*   23-69*    25-12*    26-15*    26-23*    27-78*    32-9*     32-17*    32-29*
         35-10     36-15     38-35     38-40*    39-31*    39-53*    40-6*     40-29*    40-40*    40-53*    40-76*    40-84*
         41-13*    41-58*    43-18*    43-23*    44-17*    44-33*    47-12*
QCKCPL    27-31     27-38     27-45     27-52     27-76#
QCOMPL     1-54     45-50
QCPU       1-35     30-39     31-10#
QF$IOT     1-54     19-119
QF$SCR     1-58     19-74     19-114    45-33
QF$SYN     1-54     19-119    45-33
QFREE      1-77     19-106
QHDSPN     1-28     27-51#
QHDSPX     1-35     27-44#
QHIPRI     1-24     19-192    29-13     30-7#
QNSPND     1-27     27-30#
QNSPNX     1-35     21-19     27-37#
QSRCH      1-23     26-15#    29-11
QUECHR     1-53     46-28
QUNSIG     1-39     30-25     30-37     46-17#
R$UBAS     1-98     20-25*
RMNPDR     1-81      2-83#    14-115    14-116
RPAR       1-64     14-135    14-136    34-36*
RPDR       1-64     14-131    34-43*
RUNQHD     1-35      1-37      2-13#     6-25     23-34     23-63*    24-35     25-15     25-18*    25-30*    26-16
RUNQTL     1-35      2-14#    23-47*    23-62*    25-19*    25-25*
S$$HIP     1-74     28-11
S$$RT      1-82     19-188
S$$RUN     1-53      6-27     23-18     24-19
S$CPU      1-51     31-10
S$HICP     1-85     30-17     45-38
S$INWT     1-56      1-93
S$IOFN     1-51     28-17     30-42
```

Cross reference table (CREF V05.04)

```
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#
```

Cross reference table (CREF V05.04)

```
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
UIOCNT    1-29       2-57#
UMODE     1-82      19-148    33-41     41-6
UMSPSV    1-38      17-24*    18-55*    19-57*
UPAR0     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*
UPDR0     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#    19-198
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
```

Cross reference table (CREF V05.04)