Table of contents

```
                                .TITLE  TSEMT   T/S EMT PROCESSOR
                                .ENABL  LC
                                .ENABL  AMA
                                .DSABL  CBL
                        ;
                        ;  Copyright (C) 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985.
                        ;
                        ;  S&H Computer Systems, Inc.
                        ;  Nashville, Tennessee
                        ;
                        ;  this software is furnished under a license for use only
                        ;  on a single computer system and may be copied only with
                        ;  the inclusion of the above copyright notice.  This
                        ;  software, or any other copies thereof, may not be provided
                        ;  or otherwise made available to any other person except
                        ;  for use on such system and to one who agrees to these
                        ;  license terms.  title to and ownership of the software
                        ;  shall at all times remain with S&H Computer Systems, Inc.
                        ;
000000                          .CSECT  TSEMT
000000                  TSEMT:
        177776          PS      =       177776          ;PROCESSOR STATUS WORD
                        ;
                        ;  System macros.
                        ;
                                .MCALL  .READW
                        ;-----------------------------------------------------
                        ;  MACROS TO ENABLE AND DISABLE INTERRUPTS.
                        ;
                                .MACRO  DISABL          ;DISABLE INTERRUPTS
                                BIS     #340,@#PS
                                .ENDM   DISABL
                        ;
                                .MACRO  ENABL           ;ENABLE INTERRUPTS
                                BIC     INTPRI,@#PS
                                .ENDM   ENABL
                        ;
                        ;  Macro to print an error message when a system crash occurs.
                        ;
                        ;  Arguments:
                        ;   MSG = Name of error message to print.
                        ;   ARG = (Optional) argument value to display with error message.
                        ;
                                .GLOBL  DIEMSG,DIEARG,SYSHLT
                                .MACRO  DIE     MSG,ARG
                                MOV     MSG,@#DIEMSG
                                .IF     NB,ARG
                                MOV     ARG,@#DIEARG
                                .ENDC
                                CALL    @#SYSHLT
                                .ENDM   DIE
                        ;
                        ;  Macro definition for calling 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
;
;  Global definitions
;
        .GLOBL  FREUMR, GETUMR, IOHANQ, CLWTIM, CVTPHY
        .GLOBL  TSEMT, IOSTOP, SYQIO, SYBFAD, IOSTRT
        .GLOBL  UACHKB, UACHKW, GETQ, SETC, ABTIO
        .GLOBL  GETSYQ, QIO, PUTUCH, GETUCH, FMMSG, SPLERR
        .GLOBL  EMTENT, IOFIN, QFREE, IOTIMR, SETHAN
        .GLOBL  IOWAIT, RTLOCK, SWPFRC
        .GLOBL  SETERR, EMTXIT, IOHALT, CANIOT
        .GLOBL  PTBYT, GTBYT, PTWRD, MPPHY, ERRLOG, RTLOCK
        .GLOBL  BADEMT, QCOMPL, FAKCMP, GETRTQ, EXTP1, BLKMV
        .GLOBL  REBOOT, USREMT, FILSPC, SPSIZE, SPFLDV, SPFLNM
        .GLOBL  MIOMRD, MIOMWT, HANXIT, QF$IOT
        .GLOBL  READ, WRITE, RELOC
;
;  Global references
;
        .GLOBL  $GEMAR, LSW11, EMTPLS, VPLAS
        .GLOBL  TSINIT, CUREMT, HANIOC, DOEMT, C.USED, SSEMT
        .GLOBL  VSWPFL, MAPPAR, BASMAP, LOMAP, $MLOCK, $NLOCK
        .GLOBL  S$WFM, LSW, SCPGET, SP$JOB, SP$CMD, SA$LOK
        .GLOBL  $DEAD, EMTLEV, CQ$CP, CP$STD, PRIVCO
        .GLOBL  CQ$LNK, CQ$RTN, CQ$PA5, DOSCHD, LCMPL
        .GLOBL  CQ$JOB, CQ$RO, CQ$R1, SYSXIT, $NDMEM
        .GLOBL  INTPRI, LIOCNT, QHIPRI, NMFREQ, LPRI
        .GLOBL  VPAR6, NUMDEV, CSHALC, CP$RT, QF$MIO, QF$CIO
        .GLOBL  MEMSWP, PO$LOK, CKUSP2, E375MX, EMT376, RT11EX, ODTBAS
        .GLOBL  SYTIMH, SYTIML, SYSDAT, CQ$FLG, Q.FLAG
        .GLOBL  QF$SCR, CQ$HOT, CQ$LOT, CQ$RO, CQ$R1, IT$JOB
        .GLOBL  KPAR5, KPAR6, DS$ABT, MRKTHD
        .GLOBL  FRKGET, FORKQ, FQ$PRI, FQ$RTN, FQ$R1, FQ$R5, FQ$R3
        .GLOBL  EMTBLK, IOQSIZ, CSHIO, DX$NCA, Q.ICSW, Q.UCSW
        .GLOBL  CFLAG, ERRLOC, IOCMPL, LPRI, IT$SEQ
        .GLOBL  LSW4, $INKMN, VALADW, VALADB, PRIVCO
        .GLOBL  CS$ERR, CS$EOF, CS$OPN
        .GLOBL  SERFLG
        .GLOBL  IOABFL, LSW3
        .GLOBL  RPDR
        .GLOBL  LSW6, VPRIHI, S$RT, FP$MOV, FP$IOF, FP$IOS, FP$IOA
        .GLOBL  EMTMAP, EMTSP, CHNNUM, LJSW, EM$SOF, JSTKND
        .GLOBL  CXTEND, LSW4, HANSIZ
        .GLOBL  MI$UBP, MI$SBP, MI$UBO, MI$CWC
        .GLOBL  Q.DEVX
        .GLOBL  INTERR, $INKMN
        .GLOBL  CHNADR, EMTASP, EMTRAD
        .GLOBL  CORUSR, UPMODE, LITIME, S$HICP
        .GLOBL  BOTUNI, BOTCSR
        .GLOBL  UBUSMP, DX$DMA, DVFLAG, UMRBAS, UM$IOQ
        .GLOBL  UM$WDS, UM$$SZ, UMREND, UMRWHD, Q.UMRX
        .GLOBL  Q.UMPB, Q.UMVB, UM$UMR, UMRADR, UM$NMR
```

```
                        .GLOBL  Q.UMPP
                        .GLOBL  ABORT,EMTADR
                        .GLOBL  CXTBAS,DX$MAP,MIOCHK
                        .GLOBL  URO,EMTPS,QSRCH,CHKABT,QNSPND
                        .GLOBL  S$NEDQ,DVSTAT,HANENT,FREIOQ,EMTCXT,EMTCXW
                        .GLOBL  JSWLOC,ENSYS,HANPAR
                        .GLOBL  EMTCXN,UHIMEM
                        .GLOBL  EMTERR,LSTATE,S$IOWT,UMODE
                        .GLOBL  C.CSW,Q.WCNT
                        .GLOBL  C.NUMQ,Q.JOB,Q.LINK,Q.BLKN,Q.FUNC
                        .GLOBL  Q.COMP,Q.CHAN,C.DEVQ,Q.UNIT
                        .GLOBL  Q.BUFF,Q.PAR,Q.PA5,C.SBLK,C.LENG,Q.CSW
                        .GLOBL  LBASE,UIOCNT,Q.PA6
                        .GLOBL  LSTPL
                        .GLOBL  ENQTL,FRKPRI
                        .GLOBL  S$IOFN,CQ$PRI,IT$LNK
                        .GLOBL  IT$RTN
                        .GLOBL  CQ$RNS
                        .GLOBL  LEMTPC
                        .GLOBL  LMXNUM,RSR,TSR
                        .GLOBL  MXDTR,ZCLR,MXCSR,EM$NQE
                        .GLOBL  LCXPAR,CUPARO
                        .GLOBL  READ,WRITE
                        .GLOBL  TTYIN,TTYOUT
                        .GLOBL  PRINT
                        .GLOBL  OVRHC,VPAR5,EXCJOB,SROMMR,SR3MMR,VCSHNB
```

```
                                ;-------------------------------------------------------------------------------
                                ;
                                ;  Ascii characters
                                ;
                 000015         CR      =       15              ;Carriage return
                 000012         LF      =       12              ;Line feed
                 000007         BELL    =       07              ;Bell
                                ;
                                ;  Shared data areas for mapped system regions.
                                ;
000000                          USREMT: .BLKW   6               ;Argument area for internal USR emts
000014                          FILSPC: .BLKW   4               ;Holds dev:file.ext[size] spec for current emt
000024  000000                          .WORD   0               ; [size] for file specificed above
000026  000000                  QWTCNT: .WORD   0               ;Non-zero ==> Some job waiting for Q element
000030  000000  000074          CLWTIM: .WORD   0,60.            ;Wait time for CLWAIT routine
                                ;
                                ;  Data areas used for special device (mag tape) directory operations.
                                ;  Note: SPFLDV, SPFLNM, and SPSIZE must be allocated contiguously for
                                ;  magtape directory operations to work correctly.
                                ;
000034                          SPFLDV: .BLKW   1               ;File spec device name
000036                          SPFLNM: .BLKW   4               ;File spec involved in dir operation
000046  000000                  SPSIZE: .WORD   0               ;File size returned by handler
                                        .EVEN
                                ;
                                ;  Text messages.
                                ;
                                        .NLIST  BEX
                                ;  Note: FMMSG must begin on a word boundary so that XXXXXX is
                                ;  word aligned (MOV instructions replace the form name).
                                        .EVEN
000050  015     012     012     FMMSG:  .ASCIZ  <CR><LF><LF><BELL>/ Mount 'XXXXXX' form on YYN/<CR><LF>
000112  124     123     130     SPLERR: .ASCIZ  /TSX-W-Error on write to spool file/
                                        .EVEN
                                        .LIST   BEX
```

```
                                ;-----------------------------------------------------------
                                ;  TABLE FOR V1 TYPE EMTS WHICH SHOWS HOW MANY
                                ;  ARGUMENTS ARE ON THE STACK.
                                ;
000156      000         V1STK:  .BYTE   0               ; 000  DELETE
000157      000                 .BYTE   0               ; 020  LOOKUP
000160      001                 .BYTE   1               ; 040  ENTER
000161      000                 .BYTE   0               ; 060
000162      000                 .BYTE   0               ; 100  RENAME
000163      000                 .BYTE   0               ; 120  SAVE STATUS
000164      000                 .BYTE   0               ; 140  REOPEN
000165      000                 .BYTE   0               ; 160  CLOSE
000166      003                 .BYTE   3               ; 200  READ
000167      003                 .BYTE   3               ; 220  WRITE
000170      000                 .BYTE   0               ; 240  WAIT

                                ;-----------------------------------------------------------------------
                                ;  Table for group 16 EMT's showing how many arguments are on the stack.
                                ;
000171      000         G16STK: .BYTE   0               ; 340 TTYIN
000172      000                 .BYTE   0               ; 341 TTYOUT
000173      001                 .BYTE   1               ; 342 DSTATUS
000174      001                 .BYTE   1               ; 343 FETCH/RELEASE
000175      375                 .BYTE   -3              ; 344 CSIGEN
000176      375                 .BYTE   -3              ; 345 CSISPC
000177      000                 .BYTE   0               ; 346 LOCK
000200      000                 .BYTE   0               ; 347 UNLOCK
000201      000                 .BYTE   0               ; 350 EXIT
000202      000                 .BYTE   0               ; 351 PRINT
000203      000                 .BYTE   0               ; 352 SRESET
000204      001                 .BYTE   1               ; 353 QSET
000205      000                 .BYTE   0               ; 354 SETTOP
000206      000                 .BYTE   0               ; 355 RCTRLO
000207      000                 .BYTE   0               ; 356 (undefined)
000210      000                 .BYTE   0               ; 357 HRESET
                                .EVEN
```

```
                        .SBTTL  EMT entry and initial processing
                ;-----------------------------------------------------------------------
                ;  EMTENT is entered directly from an EMT trap.
                ;  On entry, the processor will be in kernel mode and the stack pointer
                ;  will be pointing to an internal job stack.
                ;  Our initial EMT processing consists of the following steps:
                ;    1. Save registers.
                ;    2. Save current EMT processing context.
                ;    3. Determine if this is a version 1 or version 2 format EMT.
                ;    4. Move arguments for EMT from either users argument area (v2) or
                ;       stack (v1) to internal argument block (EMTBLK).
                ;    5. Jump off to appropriate EMT processing routine.
                ;
000212  013746  000000G         EMTENT: MOV     EMTPS,-(SP)     ;SAVE PREVIOUS EMT PROCESSOR STATUS
000216  016637  000004  000000G         MOV     4(SP),EMTPS     ;SAVE PS ON ENTRY FROM TRAP
000224  013746  000000G                 MOV     EMTADR,-(SP)    ;SAVE PREVIOUS EMT PC
000230  016637  000004  000000G         MOV     4(SP),EMTADR    ;SAVE PC OF TRAP
000236  013746  000000G                 MOV     EMTASP,-(SP)    ;SAVE OLD ARGUMENT STACK POINTER
000242  010637  000000G                 MOV     SP,EMTASP       ;SET EMTASP TO POINT TO STACK TOP BEFORE EMT
000246  062737  000012  000000G         ADD     #12,EMTASP      ;WAS DONE
000254  010546                         MOV     R5,-(SP)
000256  010446                         MOV     R4,-(SP)
000260  010346                         MOV     R3,-(SP)
000262  010246                         MOV     R2,-(SP)
000264  010146                         MOV     R1,-(SP)
                ;
                ;  If an EMT is already being performed, save EMT processing context.
                ;
000266  005237  000000G                 INC     EMTLEV          ;INCREMENT EMT PROCESSING LEVEL
000272  001406                         BEQ     2$              ;BR IF NO EMT IN PROGRESS
000274  012705  000000G                 MOV     #EMTCXT,R5      ;POINT TO START OF EMT CONTEXT AREA
000300  012704  000000G                 MOV     #EMTCXW,R4      ;GET # WORDS TO SAVE
000304  012546                 1$:     MOV     (R5)+,-(SP)     ;SAVE EMT CONTEXT ON STACK
000306  077402                         SOB     R4,1$
000310  010637  000000G         2$:     MOV     SP,EMTSP        ;SAVE FRAME POINTER
000314  013737  000000G 000000G         MOV     @#KPAR5,EMTMAP  ;Save PAR 5 mapping value on EMT entry
000322  010037  000000G                 MOV     R0,URO          ;URO HOLDS USER'S R0 VALUE
                ;
                ;  Initialize EMT context status.
                ;
000326  042737  000000G 000000G         BIC     #CFLAG,EMTPS    ;CLEAR C-FLAG IN PS
000334  105037  000000G                 CLRB    EMTERR          ;NO I/O ERROR CODE YET
000340  105037  000000G                 CLRB    INTERR          ;NO INTERNAL ERROR CODE
000344  013705  000000G                 MOV     EMTADR,R5       ;GET ADDRESS SAVED BY EMT TRAP
000350  006545                         MFPI    -(R5)           ;GET EMT INSTRUCTION ONTO STACK
000352  010537  000000G                 MOV     R5,EMTADR       ;SAVE ADDRESS OF EMT INSTRUCTION
000356  012604                         MOV     (SP)+,R4        ;GET THE EMT INSTRUCTION
000360  010437  000000G                 MOV     R4,CUREMT       ;SAVE THE EMT INSTRUCTION
000364  113701  000000G                 MOVB    CORUSR,R1       ;GET JOB INDEX #
000370  032737  000000G 000000G         BIT     #UMODE,EMTPS    ;WAS EMT DONE IN USER MODE?
000376  001406                         BEQ     5$              ;BR IF NOT
000400  010561  000000G                 MOV     R5,LEMTPC(R1)   ;SAVE ADDRESS OF LAST USER-MODE EMT
000404  106537  000000G                 MFPD    @#JSWLOC        ;GET USER'S JSW VALUE
000410  012661  000000G                 MOV     (SP)+,LJSW(R1)  ;SAVE IN INTERNAL TABLE
000414  020427  104341         5$:     CMP     R4,#104341      ;IS THIS A TTYOUT EMT?
000420  001002                         BNE     6$              ;BR IF NOT
000422  000137  000000G                 JMP     TTYOUT          ;HANDLE TTYOUT SPECIALLY FOR SPEED
```

EMT entry and initial processing

```
     58 000426  020427  104351           6$:     CMP     R4,#104351      ;IS THIS A .PRINT EMT?
     59 000432  001002                           BNE     7$              ;BR IF NOT
     60 000434  000137  000000G                  JMP     PRINT           ;HANDLE .PRINT SPECIALLY FOR SPEED
     61 000440  020427  104340           7$:     CMP     R4,#104340      ;IS THIS A .TTYIN EMT?
     62 000444  001002                           BNE     8$              ;BR IF NOT
     63 000446  000137  000000G                  JMP     TTYIN           ;GO PROCESS .TTYIN
     64 000452  032737  000000G 000000G 8$:     BIT     #UMODE,EMTPS    ;WAS EMT DONE IN USER MODE?
     65 000460  001407                           BEQ     3$              ;BR IF NOT
     66 000462  004737  000000G                  CALL    CKUSP2          ;CHECK VALIDITY OF USER'S SP
     67 000466  103004                           BCC     3$              ;BR IF USER'S SP IS OK
     68 000470  013704  000000G                  MOV     EMTADR,R4       ;SET ADDRESS FOR ABORT MESSAGE
     69 000474  000137  000000G                  JMP     ABORT           ;ABORT THE JOB
     70                                 ;
     71                                 ;  Determine if this is a version 1 or version 2 format EMT.
     72                                 ;
     73 000500  012701  000000G          3$:     MOV     #EMTBLK,R1      ;GET ADDRESS OF AREA FOR ARGUMENT LIST
     74 000504  120427  000375                   CMPB    R4,#375         ;Version 2 EMT?
     75 000510  001505                           BEQ     EMT375          ;Br if yes
     76 000512  120427  000374                   CMPB    R4,#374         ;VERSION 1 OR VERSION 2 FORMAT EMT?
     77 000516  103070                           BHIS    V2EMT           ;BR IF VERSION 2 EMT
```

EMT entry and initial processing

```
                                        ;
                                        ;  Do setup for a version 1 format EMT.
                                        ;  Convert emt to version 2 format and move arguments to EMTBLK.
                                        ;
000520  010405                  V1EMT:  MOV     R4,R5           ;GET EMT INSTRUCTION
000522  042704  177760                  BIC     #^C17,R4        ;ISOLATE CHANNEL NUMBER
000526  110421                          MOVB    R4,(R1)+        ;STORE INTO EMTBLK
000530  042705  177417                  BIC     #^C360,R5       ;LEAVE ONLY FUNCTION CODE
000534  072527  177774                  ASH     #-4,R5          ;RIGHT JUSTIFY
000540  110521                          MOVB    R5,(R1)+        ;STORE FUNCTION CODE INTO EMTBLK
000542  010021                          MOV     R0,(R1)+        ;USERS R0 VALUE GOES AS ARGUMENT 1
                                        ;
                                        ;  Move any arguments from user's stack to EMTBLK.
                                        ;
000544  106506                          MFPD    SP              ;GET USER'S SP
000546  012603                          MOV     (SP)+,R3        ;INTO R3
000550  120527  000016                  CMPB    R5,#16          ;IS THIS A GROUP 16 EMT?
000554  001015                          BNE     2$              ;BR IF NOT
000556  113702  000000G                 MOVB    EMTBLK,R2       ;GET SUB-FUNCTION CODE (CHANNEL #)
000562  116202  000171'                 MOVB    G16STK(R2),R2   ;GET # ARGUMENTS ON STACK
000566  002012                          BGE     1$              ;BR IF NOT .CSIxxx
                                        ;  .CSIGEN & .CSISPC have a variable number of arguments (3 or 4).
                                        ;  Determine how many we have this call.
000570  005402                          NEG     R2              ;GET 3 AS # ARGS
000572  106563  000004                  MFPD    4(R3)           ;GET 3RD ARGUMENT ON STACK
000576  032726  000001                  BIT     #1,(SP)+        ;IS THERE A 4TH ARGUMENT?
000602  001405                          BEQ     3$              ;BR IF NOT
000604  005202                          INC     R2              ;SET 4 AS # ARGUMENTS
000606  000403                          BR      3$
000610  116502  000156'          2$:     MOVB    V1STK(R5),R2    ;GET # ARGUMENTS ON STACK
000614  001514                  1$:     BEQ     ARGFIN          ;BR IF NO ARGUMENTS ON STACK
000616  032737  000000G 000000G 3$:     BIT     #UMODE,EMTPS    ;WAS EMT DONE IN USER OR KERNEL MODE?
000624  001017                          BNE     4$              ;BR IF IN USER MODE
                                        ;  EMT was executed in kernel mode.
                                        ;  This means the arguments are under all the saved information we pushed
                                        ;  on entry to EMT processing.  Pop the arguments and move down the saved
                                        ;  information on the stack.
000626  013703  000000G                 MOV     EMTASP,R3       ;GET POINTER TO ARGUMENTS ON STACK
000632  010300                          MOV     R3,R0
000634  012321                  6$:     MOV     (R3)+,(R1)+     ;MOVE AND ARGUMENT FROM STACK TO EMTBLK
000636  077202                          SOB     R2,6$           ;MOVE ALL ARGUMENTS FOR EMT
000640  014043                  5$:     MOV     -(R0),-(R3)     ;MOVE DOWN OTHER INFO ON STACK OVER ARG AREA
000642  020006                          CMP     R0,SP
000644  101375                          BHI     5$
000646  160003                          SUB     R0,R3           ;GET # BYTES STACK WAS REDUCED
000650  060306                          ADD     R3,SP           ;CORRECT SP
000652  060337  000000G                 ADD     R3,EMTSP        ;CORRECT FRAME POINTER
000656  060337  000000G                 ADD     R3,EMTASP       ;CORRECT ARGUMENT POINTER
000662  000471                          BR      ARGFIN
                                        ;  EMT was done in user mode.  This means arguments are on user-mode stack.
000664  106523                  4$:     MFPD    (R3)+           ;MOVE ARGUMENT FROM USER'S STACK TO OUR STACK
000666  012621                          MOV     (SP)+,(R1)+     ;MOVE HIS ARGUMENT TO OUR ARGUMENT AREA
000670  077203                          SOB     R2,4$           ;GET ALL OF HIS ARGUMENTS
000672  010346                          MOV     R3,-(SP)        ;NOW RESET HIS STACK POINTER
000674  106606                          MTPD    SP
000676  000463                          BR      ARGFIN
```

```
                                ;
                                ;  Do setup for a version 2 format EMT.
                                ;
000700  001004                  V2EMT:  BNE     1$              ;BR IF NOT EMT 374
                                ;
                                ;  This is an EMT 374.  (R0 Contains function-code, channel).
                                ;
000702  062700  000000C                 ADD     #E375MX*400,R0  ;BIAS FUNCTION CODE
000706  010021                          MOV     R0,(R1)+        ;R0 HAS FUNCTION CODE AND CHANNEL
000710  000456                          BR      ARGFIN          ;THERE ARE NO OTHER ARGS FOR THIS TYPE EMT
                                ;
                                ;  This is an EMT 375.  (R0 points to user's argument list).
                                ;
000712  120427  000376          1$:     CMPB    R4,#376         ;MAKE SURE THIS REALLY IS AN EMT 375
000716  103402                          BLO     EMT375          ;BR IF IT IS 375
000720  000137  0000006                 JMP     EMT376          ;BR IF 376 OR 377
                                ;
                                ;  Validate the address of the argument block
                                ;
000724  032700  000001          EMT375: BIT     #1,R0           ;Is address even?
000730  001003                          BNE     1$              ;Br if not
000732  020037  0000006                 CMP     R0,UHIMEM       ;Is address in normal job area?
000736  101402                          BLOS    5$              ;Br if yes
000740  004737  005520'          1$:     CALL    VALADW          ;MAKE SURE EMT ARG BLOCK ADDRESS IS VALID
                                ;
                                ;  Get 1st word from EMT argument block
                                ;
000744  113702  0000006         5$:     MOVB    CORUSR,R2       ;Get job index number
000750  052762  0000006 0000006         BIS     #$GEMAR,LSW11(R2);Set flag saying to return 0 on MFPD trap
000756  010003                          MOV     R0,R3           ;GET ADDRESS OF USER'S ARGUMENT BLOCK
000760  106523                          MFPD    (R3)+           ;GET USER'S FUNCTION CODE AND CHANNEL
000762  012605                          MOV     (SP)+,R5
                                ;
                                ;  Determine if this is an RT-11 or TSX system function
                                ;
000764  020527  040000                  CMP     R5,#100*400     ;TSX EMT?
000770  103402                          BLO     2$              ;BR IF NOT
000772  062705  000000C                 ADD     #<<RT11EX-100>*400>,R5 ;BIAS FUNCTION CODE
000776  010521                  2$:     MOV     R5,(R1)+        ;STORE FUNCTION CODE & CHANNEL # IN EMTBLK
                                ;
                                ;  Move arguments from user's argument area to EMTBLK.
                                ;
001000  012700  000006                  MOV     #6,R0           ;Get # words to move
001004  106523                  3$:     MFPD    (R3)+           ;GET AN ARG WORD FROM USER'S AREA
001006  012621                          MOV     (SP)+,(R1)+     ;MOVE INTO EMTBLK
001010  077003                          SOB     R0,3$
001012  042762  0000006 0000006         BIC     #$GEMAR,LSW11(R2);Clear flag that says we are getting args
001020  010201                          MOV     R2,R1           ;Carry job index in R1
                                ;
                                ;  Do fast check for shared run-time mapping
                                ;
001022  123727  0000016 0000436         CMPB    EMTBLK+1,#<143+<RT11EX-100>>;Shared run-time mapping?
001030  001002                          BNE     4$              ;Br if not
001032  000137  0000006                 JMP     SSEMT           ;Go do shared run-time mapping
001036  123727  0000016 000036  4$:     CMPB    EMTBLK+1,#36    ;PLAS EMT?
001044  001402                          BEQ     DOPLAS          ;Br if PLAS EMT
                                ;
```

```
                                ;  Jump into TSEM2 overlay to complete EMT processing
                                ;
001046  000137  000000G        ARGFIN: JMP     DOEMT           ;Enter TSEM2 to complete processing
                                ;
                                ;  PLAS EMT
                                ;
001052  005737  000000G        DOPLAS: TST     VPLAS           ;Was PLAS genned into system?
001056  001402                        BEQ     9$              ;Br if not
001060  000137  000000G                JMP     EMTPLS          ;Enter TSPLAS overlay
001064  012700  000007         9$:     MOV     #7,R0           ;Return error code 0
001070  005037  000000G                CLR     UR0             ;Return 0 in R0
001074  000137  004036'                JMP     SETERR

                                ;-------------------------------------------------------------------------------
                                ;  Invalid EMT
                                ;
001100  012700  177767         BADEMT: MOV     #-11,R0         ;ERROR CODE #
001104  000137  004036'                JMP     SETERR
```

```
                                        .SBTTL  EMTXIT -- Exit from EMT processing
                                ;-------------------------------------------------------------------
                                ;  EMTXIT is jumped to at the end of processing of an EMT.
                                ;  Control is returned to the user.
                                ;
                                ;  Check to make sure the job context block stack did not overflow.
                                ;
001110  023727  000000G 123456  EMTXIT: CMP     JSTKND,#123456  ;CHECK FOR FIXED VALUE AT END OF STACK
001116  001410                         BEQ     6$              ;BR IF OK
001120                               1$: DIE     #EM$SOF,#3      ;STACK OVERFLOW -- HALT SYSTEM
                                ;
                                ;  If the EMT was executed in user mode, make sure
                                ;  previous-mode = user in the current PSW.
                                ;
001140  032737  000000G 000000G 6$:     BIT     #UMODE,EMTPS    ;Was EMT executed in user mode?
001146  001403                         BEQ     7$              ;Br if not
001150  052737  000000G 177776         BIS     #UPMODE,@#PS    ;Make sure previous-mode = user in PSW
                                ;
                                ;  See if an internal error code was specified.
                                ;
001156  105737  000000G         7$:     TSTB    INTERR          ;ANY INTERNAL ERROR CODE?
001162  001403                         BEQ     1$              ;BR IF NOT
001164  113737  000000G 000000G         MOVB    INTERR,EMTERR   ;SET AS REAL ERROR CODE
                                ;
                                ;  Return I/O error code to user.
                                ;
001172  106537  000000G         1$:     MFPD    @#ERRLOC        ;GET CONTENTS OF ERROR CELL
001176  113716  000000G                 MOVB    EMTERR,(SP)     ;PUT EMT ERROR CODE IN LOW-ORDER BYTE
001202  001403                         BEQ     2$              ;BR IF THERE WAS NO ERROR
001204  052737  000000G 000000G         BIS     #CFLAG,EMTPS    ;SET C-FLAG IN PS
001212  106637  000000G         2$:     MTPD    @#ERRLOC        ;RESET ERROR CELL
                                ;
                                ;  Return value to user in R0.
                                ;
001216  013700  000000G                 MOV     UR0,R0          ;GET VALUE TO BE RETURNED IN R0
                                ;
                                ;  Restore mapping for kernel PAR 5 region
                                ;
001222  013737  000000G 000000G         MOV     EMTMAP,@#KPAR5  ;Restore mapping for kernel PAR 5 region
                                ;
                                ;  Restore EMT context.
                                ;
001230  013706  000000G                 MOV     EMTSP,SP        ;RESTORE FRAME POINTER
001234  005337  000000G                 DEC     EMTLEV          ;DID WE PUSH CONTEXT INFORMATION?
001240  002406                         BLT     3$              ;BR IF NOT
001242  012705  000000G                 MOV     #EMTCXN,R5      ;POINT TO END OF CONTEXT AREA
001246  012704  000000G                 MOV     #EMTCXW,R4      ;GET # WORDS TO RESTORE
001252  012645                 4$:     MOV     (SP)+,-(R5)     ;RESTORE EMT CONTEXT
001254  077402                         SOB     R4,4$
                                ;
                                ;  Restore registers
                                ;
001256  012601                 3$:     MOV     (SP)+,R1
001260  012602                         MOV     (SP)+,R2
001262  012603                         MOV     (SP)+,R3
001264  012604                         MOV     (SP)+,R4
001266  012605                         MOV     (SP)+,R5
```

EMTXIT -- Exit from EMT processing

```
                                        ;
                                        ;  Restore stack argument pointer
                                        ;
001270  012637  000000G                 5$:     MOV     (SP)+,EMTASP    ;POINTS TO WHERE ARGUMENTS WERE ON STACK
                                        ;
                                        ;  Set PS for return
                                        ;
001274  013766  000000G 000006                  MOV     EMTPS,6(SP)     ;PUT PS ON STACK
                                        ;
                                        ;  Restore EMTPS & EMTADR.
                                        ;
001302  012637  000000G                         MOV     (SP)+,EMTADR
001306  012637  000000G                         MOV     (SP)+,EMTPS
                                        ;
                                        ;  At this point the PS & PC pushed as a result of the EMT are the only
                                        ;  things on the stack.
                                        ;  See if a .SPCPS EMT was done that wants to alter return from the EMT.
                                        ;
001312  005737  000000G                         TST     EMTRAD          ;DO WE WANT TO ALTER EMT EXIT ADDRESS?
001316  001407                                 BEQ     5$              ;BR IF NOT
                                        ;  Force control away from completion routine.
001320  013716  000000G                         MOV     EMTRAD,(SP)     ;SET NEW PC FOR EXIT
001324  012766  000000G 000002                  MOV     #UPMODE,2(SP)   ;SET NEW PS FOR EXIT
001332  005037  000000G                         CLR     EMTRAD          ;SAY WE HAVE FINISHED THE REQUEST
                                        ;
                                        ;  Exit throuth INTEN code.
                                        ;
001336  000137  000000G                 5$:     JMP     SYSXIT          ;EXIT FROM SYSTEM STATE
```

```
                                        .SBTTL  REBOOT -- Kmon EMT to reboot the system
                                ;-------------------------------------------------------------------------
                                ;  Reboot the system.
                                ;  Word 2 of EMT arg block contains address of start of bootstrap code.
                                ;
001342                          REBOOT: DISABL                  ;** DISABLE **
001350  012777  000002  000100'        MOV     #2,@100         ;DISABLE CLOCK PROCESSING
001356  013737  000100  000004        MOV     @#100,@#4       ;Ignore non-existent traps
                                ;
                                ;  Disable time-sharing lines
                                ;
001364  012705  000000G               MOV     #LSTPL,R5       ;GET # OF LAST PRIMARY LINE
001370  032765  000000G 000000G 7$:    BIT     #$DEAD,LSW3(R5) ;IS THIS LINE INSTALLED?
001376  001015                        BNE     6$              ;BR IF NOT INSTALLED - CONTINUE TO NEXT LINE
001400  016504  000000G               MOV     LMXNUM(R5),R4   ;IS THIS A DL-11 OR DZ-11 LINE?
001404  001005                        BNE     4$              ;BR IF DZ-11 LINE
001406  005075  000000G               CLR     @RSR(R5)        ;STOP RECEIVER
001412  005075  000000G               CLR     @TSR(R5)        ;STOP TRANSMITTER
001416  000405                        BR      6$              ;CONTINUE TO NEXT LINE
001420  105074  000000G         4$:   CLRB    @MXDTR(R4)      ;CLEAR DZ-11 DATA TERMINAL READY
001424  052774  000000G 000000G       BIS     #ZCLR,@MXCSR(R4);CLEAR DZ-11 CONTROL STATUS REGISTER
001432  162705  000002          6$:   SUB     #2,R5           ;GO STOP NEXT LINE
001436  003354                        BGT     7$
                                ;
                                ;  Disable the generalized cache and the scheduler.
                                ;
001440  113737  000000G 000000G       MOVB    CORUSR,EXCJOB   ;Say we are the exclusive job
001446  005037  000000G               CLR     @#VCSHNB        ;Disable caching
001452  013705  000004G               MOV     EMTBLK+4,R5     ;GET THE ADDRESS OF THE BOOTSTRAP CODE
                                ;
                                ;  Reset system and issue read to set unit number in controller.
                                ;
001456  013737  000000G 000014'       MOV     @#SROMMR,FILSPC ;Save MMU 0 status
001464  013737  000000G 000016'       MOV     @#SR3MMR,FILSPC+2;Save MMU 3 status
001472  000005                        RESET                   ;Issue bus reset
001474  013737  000014' 000000G       MOV     FILSPC,@#SROMMR ;Restore MMU 0 status
001502  013737  000016' 000000G       MOV     FILSPC+2,@#SR3MMR;Restore MMU 3 status
001510                                .READW  #USREMT,#1,#SPFLNM,#1,#0;Read from boot device
001546                                DISABL                  ;Disable interrupts
001554  013700  000000G               MOV     BOTUNI,R0       ;Get boot device unit number
001560  013701  000000G               MOV     BOTCSR,R1       ;Get boot CSR address
                                ;
                                ;  Move the bootstrap code from the user's region into kernal mapping.
                                ;
                                ;
001564  012702  000000G               MOV     #TSINIT,R2      ;COPY BOOTSTRAP INTO KERNEL MAPPING
001570  012703  002400                MOV     #256.*5,R3      ;MOVE 5 BLOCKS OF CODE
001574  106525                  8$:   MFPD    (R5)+           ;GET THE NEXT WORD
001576  012622                        MOV     (SP)+,(R2)+     ;STORE IT OVER TSINIT
001600  077303                        SOB     R3,8$           ;CONTINUE UNTIL PRIMARY & SECONDARY CODE MOVED
                                ;
                                ;  Clear memory management registers.
                                ;
001602  005037  000000G               CLR     SROMMR          ;Clear MMU 0 status
001606  005037  000000G               CLR     SR3MMR          ;Clear MMU 1 status
                                ;
                                ;  Set up date and time words.
```

```
                                        ;
001612  013737  0000006 005004          MOV     SYSDAT,@#5004   ;DATE
001620  013737  0000006 005000          MOV     SYTIMH,@#5000   ;TIME - HIGH ORDER
001626  013737  0000006 005002          MOV     SYTIML,@#5002   ;TIME - LOW ORDER
                                        ;
                                        ;  Copy bootstrap to low memory (0 - 4777).
                                        ;
001634  012702  0000006                 MOV     #TSINIT,R2      ;Point to copy of bootstrap
001640  005005                          CLR     R5              ;Move to physical location zero
001642  012703  002400                  MOV     #256.*5,R3      ;Move 5 blocks of code
001646  012225                  10$:    MOV     (R2)+,(R5)+     ;Move to low memory
001650  077302                          SOB     R3,10$          ;Continue until all is moved
                                        ;
                                        ;  Set up for bootstrap.
                                        ;
001652  005037  000000                  CLR     @#0             ;SAY DATE & TIME ARE STORED IN MEMORY
001656  010037  004722                  MOV     R0,@#4722       ;SET THE CORRECT BOOTSTRAP UNIT NUMBER
001662  012706  010000                  MOV     #10000,SP       ;SET STACK POINTER FOR BOOTSTRAP
001666  012703  001000                  MOV     #1000,R3        ;Point to bootstrap code
                                        ;
                                        ;  Jump into bootstrap
                                        ;
001672  000113                          JMP     @R3             ;ENTER BOOTSTRAP CODE
```

```
                        .SBTTL  RTLOCK -- EMT to lock a job in low memory
                ;-----------------------------------------------------------------------
                ;  The RTLOCK emt is used to lock a job in memory.  The job is moved
                ;  to the low end of user memory space before being locked.
                ;
001674  032737  0000000G 0000000G RTLOCK: BIT     #PO$LOK,PRIVCO  ;Are we allowed to lock job in memory?
001702  001003                     BNE     3$              ;Br if yes
001704  005000                     CLR     R0              ;Return error code 0
001706  000137  004036'             JMP     SETERR
001712  105737  0000000G           3$:     TSTB    VSWPFL          ;Is this a no swap system?
001716  001465                     BEQ     8$              ;Br if yes - don't change memory allocation
                ;
                ;  Map PAR 5 to the memory management table
                ;
001720  013737  0000000G 0000000G         MOV     MAPPAR,@#KPAR5  ;Map PAR 5 to memory management table
                ;
                ;  See if job is at base of memory now
                ;
001726  016102  0000000G           1$:     MOV     LBASE(R1),R2    ;Get base page # for job
001732  063702  0000000G                   ADD     BASMAP,R2       ;Get address of base page entry for job
001736  020237  0000000G                   CMP     R2,LOMAP        ;Are we at base of memory
001742  101453                     BLOS    8$              ;Br if yes -- don't need to reposition job
001744  114200                     MOVB    -(R2),R0        ;Get # of job using this page
001746  001405                     BEQ     2$              ;Br if page is free
001750  002772                     BLT     1$              ;Br if page is not available to jobs
001752  032760  0000000G 0000000G         BIT     #$MLOCK,LSW6(R0);Is this job locked in memory?
001760  001366                     BNE     1$              ;Br if yes
                ;
                ;  Job is not at base of memory.
                ;  Use the job swapper to reposition it.
                ;
001762  052761  0000000G 0000000G 2$:      BIS     #$NDMEM,LSW(R1) ;Set flag saying we need to be swapped
001770  005237  0000000G                   INC     MEMSWP          ;Tell swapper to do outswap
001774  052761  0000000G 0000000G         BIS     #$NLOCK,LSW6(R1);Set flag saying job needs to be locked in mem
002002  013702  0000000G                   MOV     LOMAP,R2        ;Get lowest page # in use by user jobs
002006  163702  0000000G                   SUB     BASMAP,R2       ;Convert to a page #
002012  016100  0000000G                   MOV     LBASE(R1),R0    ;Get base page # of our job
002016  160200                     SUB     R2,R0           ;Get # pages to be swept
002020  004737  002104'             CALL    SWPFRC          ;Outswap jobs in the memory area
                ;
                ;  Now generate a swap command packet to cause our job to be
                ;  locked in memory.
                ;
002024  004737  0000000G           CALL    SCPGET          ;Get a swap command packet
002030  110165  0000000G           MOVB    R1,SP$JOB(R5)   ;Set our job number
002034  112765  0000000G 0000000G MOVB    #SA$LOK,SP$CMD(R5) ;Set lock-in-memory command
                ;
                ;  Now suspend execution of our job until it is repositioned and locked
                ;  in memory.
                ;
002042  012700  0000000G           5$:     MOV     #S$WFM,R0       ;State = waiting for memory expansion
002046  004737  0000000G                   CALL    QNSPND          ;Suspend job and do swapping to move job
002052  004737  0000000G                   CALL    CHKABT          ;See if we were aborted while asleep
002056  032761  0000000G 0000000G         BIT     #$MLOCK,LSW6(R1);Is locking finished?
002064  001766                     BEQ     5$              ;Br if not
002066  000137  001110'             JMP     EMTXIT          ;Finished
                ;
```

RTLOCK -- EMT to lock a job in low memory

```
                                        ;  Freeze job in current memory position
                                        ;
002072  052761  0000000 0000000 8$:     BIS     #$MLOCK,LSW6(R1);Lock job in memory
002100  000137  001110'                 JMP     EMTXIT          ;Finished
```

```
                                        .SBTTL  SWPFRC -- Force outswap of jobs in certain memory range
                                ;-----------------------------------------------------------------------
                                ;  SWPFRC is called to force the outswap of all jobs that are occuping
                                ;  a certain range of memory.  The jobs are outswapped and then
                                ;  placed in an executable state.
                                ;
                                ;  Inputs:
                                ;   R2 = Lowest page # of region to be cleaned out.
                                ;   R0 = Number of 512-byte pages in region.
                                ;
002104  010146                  SWPFRC: MOV     R1,-(SP)
002106  010246                          MOV     R2,-(SP)
002110  013746  000000G                 MOV     @#KPAR5,-(SP)
                                ;
                                ;  Map PAR 5 to the memory map table
                                ;
002114  013737  000000G 000000G         MOV     MAPPAR,@#KPAR5  ;Map PAR 5 to memory allocation table
                                ;
                                ;  Sweep through allocation table looking for jobs to outswap
                                ;
002122  063702  000000G                 ADD     BASMAP,R2       ;Point to 1st entry in map table
002126  112201                  1$:     MOVB    (R2)+,R1        ;Get info about who is using this page
002130  003415                          BLE     2$              ;Br if page free or in use by system
002132  032761  000000G 000000G         BIT     #$NDMEM,LSW(R1) ;Is job already flagged for outswap?
002140  001011                          BNE     2$              ;Br if yes
002142  032761  000000G 000000G         BIT     #$MLOCK,LSW6(R1);Is job locked in memory?
002150  001005                          BNE     2$              ;Br if yes
                                ;
                                ;  We found a job that needs to be flagged for outswap
                                ;
002152  052761  000000G 000000G         BIS     #$NDMEM,LSW(R1) ;Flag job for outswapping
002160  005237  000000G                 INC     MEMSWP          ;Remember another job to outswap
                                ;
                                ;  Check rest of memory region
                                ;
002164  077020                  2$:     SOB     R0,1$           ;Loop if more of region to check
                                ;
                                ;  Finished
                                ;
002166  012637  000000G                 MOV     (SP)+,@#KPAR5   ;Restore PAR 5 mapping
002172  012602                          MOV     (SP)+,R2
002174  012601                          MOV     (SP)+,R1
002176  000207                          RETURN
```

```
                                        .SBTTL  QIO     -- Queue an I/O request
                                ;-------------------------------------------------------------------
                                ;  QIO is called to add an I/O request to the list of waiting
                                ;  requests for a device.  If the device is idle it is started.
                                ;
                                ;  Inputs:
                                ;  R1 = Address of I/O queue entry.
                                ;
                                ;
                                ;  Entry point for system I/O requests.
                                ;
002200  013761  000000G 000000G SYQIO:  MOV     @#KPAR5,Q.PA5(R1);Save PAR 5 mapping
                                ;
                                ;  Entry point for normal job I/O requests
                                ;
002206  010246                  QIO:    MOV     R2,-(SP)
002210  013746  000000G                 MOV     @#KPAR5,-(SP)
                                ;
                                ;  Increment I/O counters.
                                ;
002214  016102  000000G                 MOV     Q.UCSW(R1),R2   ;Address of user's channel block
002220  001412                          BEQ     1$              ;Br if no channel block
002222  020227  000000G                 CMP     R2,#CXTBAS      ;Is channel in job context block?
002226  103405                          BLO     3$              ;Br if not
002230  016137  000000G 000000G         MOV     Q.PA6(R1),@#KPAR5 ;Map par 5 to loc where context blk is
002236  062702  000000C                 ADD     #<VPAR5-CXTBAS>,R2;Get address of mapped channel block
002242  105262  000000G         3$:     INCB    C.NUMQ(R2)      ;Increment # I/O requests active on channel
002246  116100  000000G         1$:     MOVB    Q.JOB(R1),R0    ;Get job #
002252  001410                          BEQ     2$              ;Br if system doing I/O
002254  006300                          ASL     R0              ;Convert to job index number
002256  105260  000000G                 INCB    LIOCNT(R0)      ;Inc # I/O requests active for this job
002262  116160  000000G 000001G         MOVB    Q.DEVX(R1),LIOCNT+1(R0) ;Save index # of last dev job accessed
002270  005237  000000G                 INC     UIOCNT          ;Count total # of active user I/O operations
                                ;
                                ;  Call IOSTRT to add I/O queue request to list for handler and
                                ;  start the I/O.
                                ;
002274  004737  002310'         2$:     CALL    IOSTRT          ;Add queue entry to handler and start I/O
                                ;
                                ;  Finished
                                ;
002300  012637  000000G                 MOV     (SP)+,@#KPAR5
002304  012602                          MOV     (SP)+,R2
002306  000207                          RETURN
```

```
                        .SBTTL  IOSTRT -- Try to start I/O transfer
                ;-----------------------------------------------------------------------
                ;  IOSTRT is called to initiate an I/O operation.
                ;  It performs the following operations:
                ;   1. Allocate a Unibus Map Register (UMR) if I/O operation is being
                ;      directed to a DMA device.
                ;   2. Move data from user's buffer to system buffer if this is an
                ;      I/O to a device that requires system buffer mapping.
                ;   3. Put I/O queue entry on list for device handler.
                ;   4. Initiate device operation if it is idle.
                ;
                ;  Inputs:
                ;   R1 = Address of I/O queue entry.
                ;
002310  010246  IOSTRT: MOV     R2,-(SP)
002312  010446          MOV     R4,-(SP)
                ;
                ;  Get index number of device this I/O operation is directed to.
                ;
002314  116102  000000G         MOVB    Q.DEVX(R1),R2   ;GET DEVICE INDEX NUMBER
002320  001004          BNE     2$              ;BR IF I/O TO REAL DEVICE
                ;
                ;  If device index number is zero, call IOCMPL to indicate I/O is finished
                ;
002322  010104          MOV     R1,R4           ;GET I/O QUEUE ELEMENT ADDRESS TO R4
002324  004737  003032'  CALL    IOCMPL          ;Do I/O completion processing
002330  000427          BR      QIOFIN          ;WE ARE FINISHED WITH I/O OPERATION
                ;
                ;  See if we are doing caching for this device
                ;
002332  032762  000000G 000000G 2$:     BIT     #DX$NCA,DVFLAG(R2);Is this device eligible for caching?
002340  001006          BNE     4$              ;Br if not
002342  005737  000000G         TST     CSHALC          ;Is data caching genned into system?
002346  001403          BEQ     4$              ;Br if not
002350  004777  000000G         CALL    @CSHIO          ;See if doing data caching for this device
002354  103415          BCS     QIOFIN          ;Br if caching took over the operation
                ;
                ;  See if we need to do system buffering for this I/O operation
                ;
002356  032762  000000G 000000G 4$:     BIT     #DX$MAP,DVFLAG(R2);Does this device require buffer mapping?
002364  001404          BEQ     3$              ;Br if not
002366                  OCALL   MIOCHK          ;See if this particular operation needs maping
002374  103405          BCS     QIOFIN          ;Br if I/O mapping is being done
                ;
                ;  See if we need to allocate a Unibus map register set for this operation
                ;
002376  004737  004470'  3$:     CALL    GETUMR          ;SEE IF WE NEED TO ALLOCATE A UMR
002402  103402          BCS     QIOFIN          ;BR IF NO FREE UMR -- DEFER STARTING OPERATION
                ;
                ;  Place queue element on list for device handler
                ;
002404  004737  002416'         CALL    IOHANQ          ;Place queue element on list for handler
                ;
                ;  Finished
                ;
002410  012604  QIOFIN: MOV     (SP)+,R4
002412  012602          MOV     (SP)+,R2
```

IOSTRT -- Try to start I/O transfer

```
     58 002414  000207                          RETURN
```

```
                                        .SBTTL  IOHANQ -- Place I/O queue request on handler list
                                ;-------------------------------------------------------------------
                                ;  Place an I/O queue request on the list of pending requests for a device
                                ;  handler and enter the handler front-end if the handler is currently
                                ;  idle.
                                ;
                                ;  Inputs:
                                ;   R1 = Address of I/O queue entry
                                ;
002416  010146                  IOHANQ: MOV     R1,-(SP)
002420  010246                          MOV     R2,-(SP)
002422  010346                          MOV     R3,-(SP)
002424  010446                          MOV     R4,-(SP)
002426  010546                          MOV     R5,-(SP)
002430  013746  000000G                 MOV     @#KPAR5,-(SP)   ;Preserve system PAR 5 mappint
                                ;
                                ;  Increment count of uncompleted I/O operations for this device
                                ;
002434  116102  000000G                 MOVB    Q.DEVX(R1),R2   ;Get device index number
002440  005262  000000G                 INC     HANIOC(R2)      ;Say another I/O operation started for dev
                                ;
                                ;  Get address of device handler and set up PAR 5 if this is a mapped handler
                                ;
002444  016200  000000G                 MOV     HANPAR(R2),R0   ;Is this a mapped handler?
002450  001402                          BEQ     5$              ;Br if not
002452  010037  000000G                 MOV     R0,@#KPAR5      ;Map kernel PAR 5 to handler
002456  016202  000000G          5$:    MOV     HANENT(R2),R2   ;R2 --> 4th word of handler
                                ;
                                ;  Make R1 point to 3rd word of queue element (as handler wants it).
                                ;
002462  005061  000000G                 CLR     Q.LINK(R1)      ;SAY QUEUE ELEMENT WILL BE AT END OF CHAIN
002466  062701  000000G                 ADD     #Q.BLKN,R1      ;MAKE R1 POINT TO 3RD WORD OF QUEUE ELEMENT
                                ;
                                ;  Add new queue element to list of requests for this device.
                                ;
002472                                  DISABL                  ;* DISABLE
002500  011200                          MOV     (R2),R0         ;* ADDRESS OF LAST Q ELEMENT IN LIST
002502  001407                          BEQ     1$              ;* BR IF HANDLER IS IDLE NOW
                                ;
                                ;  Handler is active.
                                ;  Just add our queue element to its list.
                                ;
002504  010160  000000C                 MOV     R1,Q.LINK-Q.BLKN(R0);* SET FORWARD LINK TO POINT TO US
002510  010112                          MOV     R1,(R2)         ;* STORE OUR Q ELEM ADDR INTO LQE IN HANDLER
002512                                  ENABL
002520  000413                          BR      9$
                                ;
                                ;  Handler is idle.
                                ;  Put queue entry into its list and start it.
                                ;
002522  010122                  1$:     MOV     R1,(R2)+        ;* STORE Q POINTER INTO HANDLER LQE
002524  010122                          MOV     R1,(R2)+        ;* STORE Q POINTER INTO HANDLER CQE
002526                                  ENABL
                                ;
                                ;  Enter system state so that we will be running on interrupt stack.
                                ;  We do this so the handler front-end can remap kernel PAR6 if it wants to.
                                ;
```

```
     58 002534  012700  002550'                MOV     #9$,R0          ;GO HERE ON RETURN FROM SYSTEM STATE
     59 002540  004737  0000000                CALL    ENSYS           ;ENTER SYSTEM STATE
     60 002544  0000000                        .WORD   FP$IOS          ;Fork processing priority
     61                                 ;
     62                                 ;  We are now running in system state on interrupt stack.
     63                                 ;  Enter handler front-end to initiate I/O operation.
     64                                 ;  When the handler does a RETURN, we will exit system state and go to QIOFIN.
     65                                 ;
     66 002546  000112                        JMP     @R2             ;ENTER HANDLER FRONT-END
     67                                 ;
     68                                 ;  Finished
     69                                 ;
     70 002550  012637  0000000         9$:    MOV     (SP)+,@#KPAR5
     71 002554  012605                         MOV     (SP)+,R5
     72 002556  012604                         MOV     (SP)+,R4
     73 002560  012603                         MOV     (SP)+,R3
     74 002562  012602                         MOV     (SP)+,R2
     75 002564  012601                         MOV     (SP)+,R1
     76 002566  000207                         RETURN
```

```
                                        .SBTTL  IOFIN  -- I/O operation completion
                                ;-------------------------------------------------------------
                                ;  IOFIN is called by the device driver when it completes an I/O operation.
                                ;  IOFIN queues a completion routine request for the user if it was specified
                                ;  with the EMT, then tries to initiate any other pending I/O requests for
                                ;  the device.
                                ;
                                ;  Inputs:
                                ;  R4 = Address of cell in handler with pointer to current queue entry.
                                ;
002570  010046                  IOFIN:  MOV     R0,-(SP)
002572  010146                          MOV     R1,-(SP)
002574  010346                          MOV     R3,-(SP)
002576  010446                          MOV     R4,-(SP)
002600  010546                          MOV     R5,-(SP)
                                ;
                                ;  If handler is being held, return immediately without doing any cleanup.
                                ;
002602  006264  177774                  ASR     -4(R4)          ;Is handler being held?
002606  100457                          BMI     9$              ;Br if yes
                                ;
                                ;  Handler is not being held.
                                ;  Remove completed (current) queue entry from list for handler.
                                ;
002610  010405                          MOV     R4,R5           ;Get address of CQE cell in handler
002612                                  DISABL                  ;;;** Disable interrupts **
002620  011501                          MOV     (R5),R1         ;;;Address of 3rd word of current Q element
002622  001004                          BNE     1$              ;;;Br if there is a completed Q element
002624                                  ENABL                   ;Handler has no active queue element
002632  000445                          BR      9$
002634  162701  000000G       1$:     SUB     #Q.BLKN,R1      ;;;Point to 1st word of queue element
002640  016115  000000G               MOV     Q.LINK(R1),(R5) ;;;Delink queue element from handler list
002644  011503                          MOV     (R5),R3         ;;;Remember if we need to restart handler
002646  001002                          BNE     2$              ;;;Br if there are more pending elements
002650  005065  177776                  CLR     -2(R5)          ;;;Clear LQE pointer in handler
                                ;
                                ;  At this point we have removed the completed queue element from the
                                ;  handlers list.
                                ;  We have set the current queue element (CQE) pointer in the handler to
                                ;  point to the next queue element (if any) and have zeroed the last
                                ;  queue element (LQE) pointer if there are no remaining queue entries.
                                ;  R1 = Address of 1st word of completed queue element.
                                ;  R3 = Non-zero ==> Need to restart handler for next pending I/O op.
                                ;  R5 = Pointer to CQE cell in handler.
                                ;
002654                          2$:     ENABL                   ;** Enable interrupts **
                                ;
                                ;  Decrement count of uncompleted I/O operations for this device
                                ;
002662  116100  000000G                 MOVB    Q.DEVX(R1),R0   ;Get device index number
002666  005360  000000G                 DEC     HANIOC(R0)      ;Dec # uncompleted I/O operations for dev
                                ;
                                ;  If the device handler did not do a .FORK before calling IOFIN, we
                                ;  queue a fork request to do the completion processing for this
                                ;  queue element.
                                ;  If the handler did do a .FORK, we do the completion processing now.
                                ;
```

```
     58 002672  105737  000000G                  TSTB    FRKPRI          ;Are we running at fork level now?
     59 002676  001403                           BEQ     3$              ;Br if not
     60 002700  004737  002762'                  CALL    IOFCPL          ;Do completion processing
     61 002704  000420                           BR      9$              ;Finished
     62                                ;
     63                                ;  Handler did not fork.  Queue a fork request to process completion routine
     64                                ;
     65 002706  004737  000000G       3$:        CALL    FRKGET          ;Get a free fork request block (addr in R4)
     66 002712  112764  000000G 000000G          MOVB    #FP$IOF,FQ$PRI(R4);Set fork processing priority
     67 002720  012764  002762' 000000G          MOV     #IOFCPL,FQ$RTN(R4);Set address of fork routine
     68 002726  010164  000000G                  MOV     R1,FQ$R1(R4)    ;Pass address of queue element in R1
     69 002732  010364  000000G                  MOV     R3,FQ$R3(R4)    ;Pass restart flag in R3
     70 002736  010564  000000G                  MOV     R5,FQ$R5(R4)    ;Pass address of CQE cell in R5
     71 002742  004737  000000G                  CALL    FORKQ           ;Queue the fork request
     72                                ;
     73                                ;  Finished
     74                                ;
     75 002746  012605                 9$:       MOV     (SP)+,R5
     76 002750  012604                           MOV     (SP)+,R4
     77 002752  012603                           MOV     (SP)+,R3
     78 002754  012601                           MOV     (SP)+,R1
     79 002756  012600                           MOV     (SP)+,R0
     80 002760  000207                           RETURN
```

```
                                        .SBTTL  IOFCPL -- IOFIN processing done at fork level
                                ;-------------------------------------------------------------------
                                ;  This routine performs the part of the IOFIN operation that must
                                ;  run at fork level.
                                ;  This routine does the completion processing for the completed queue
                                ;  element and then recalls the handler front end if there is another
                                ;  queue element waiting to be started.
                                ;
                                ;  Inputs:
                                ;   R1 = Address of 1st word of completed queue element.
                                ;   R3 = Non-zero ==> Must call handler to start pending I/O operation
                                ;   R5 = Address of CQE cell in handler.
                                ;
002762  010446                  IOFCPL: MOV     R4,-(SP)
                                ;
                                ;  Perform completion operation on queue element
                                ;
                                        MOV     R1,R4           ;Get address of completed Q element
002764  010104
002766  004737  003032'         CALL    IOCMPL          ;Do completion processing
                                ;
                                ;  If there is a pending queue element for the handler, call the handler
                                ;  front end to start the next operation.
                                ;
                                        TST     R3              ;Do we need to recall handler?
002772  005703
002774  001414                  BEQ     9$              ;Br if not
                                ;
                                ;  There is a pending I/O operation.
                                ;  Call handler initiation section.
                                ;
002776  010046                          MOV     R0,-(SP)        ;Handler front end may use any register
003000  010146                          MOV     R1,-(SP)
003002  010246                          MOV     R2,-(SP)
003004  010346                          MOV     R3,-(SP)
003006  010546                          MOV     R5,-(SP)
003010  004765  000002                  CALL    2(R5)           ;Call handler front end to start next op
003014  012605                          MOV     (SP)+,R5
003016  012603                          MOV     (SP)+,R3
003020  012602                          MOV     (SP)+,R2
003022  012601                          MOV     (SP)+,R1
003024  012600                          MOV     (SP)+,R0
                                ;
                                ;  Finished
                                ;
003026  012604                  9$:     MOV     (SP)+,R4
003030  000207                          RETURN
```

```
                                        .SBTTL  IOCMPL -- Do cleanup on completed I/O queue element
                                ;-----------------------------------------------------------------------
                                ;  IOCMPL is called to do the cleanup operation on an I/O queue
                                ;  element that belongs to an I/O operation that has just completed.
                                ;
                                ;  Inputs:
                                ;   R4 = Address of I/O queue element to be cleaned up.
                                ;
                                ;  Outputs:
                                ;   All cleanup is done on the queue element and the queue element is
                                ;   returned to the free list or is used to queue a completion routine.
                                ;
003032  010046                  IOCMPL: MOV     R0,-(SP)
003034  010146                          MOV     R1,-(SP)
003036  010346                          MOV     R3,-(SP)
003040  013746  000000G                 MOV     @#KPAR5,-(SP)   ;Save par 5 mapping
003044  010401                          MOV     R4,R1           ;Get address of I/O queue element to R1
003046  004737  004772'                 CALL    FREUMR          ;Release any unibus map reg used for I/O
003052  105764  000000G                 TSTB    Q.FUNC(R4)      ;Special lookup/enter?
003056  003403                          BLE     1$              ;Br if not
003060  016437  000000G 000046'         MOV     Q.WCNT(R4),SPSIZE ;Save file size
003066  016403  000000G          1$:    MOV     Q.UCSW(R4),R3   ;Get address of user's channel status block
003072  116401  000000G                 MOVB    Q.JOB(R4),R1    ;Get job number associated with request
003076  001405                          BEQ     7$              ;Br if request came from system
003100  006301                          ASL     R1              ;Convert job # to job index number
003102  105361  000000G                 DECB    LIOCNT(R1)      ;Dec # I/O operations active for this user
003106  005337  000000G                 DEC     UIOCNT          ;One more user I/O operation finished
003112  005703                  7$:     TST     R3              ;Is there a channel block?
003114  001427                          BEQ     12$             ;Br if not
003116  020327  000000G                 CMP     R3,#CXTBAS      ;Is channel in mapped job context block?
003122  103405                          BLO     9$              ;Br if not
003124  016437  000000G 000000G         MOV     Q.PA6(R4),@#KPAR5;Map par 5 to job context block
003132  062703  000000C                 ADD     #<VPAR5-CXTBAS>,R3;Get address of mapped channel in par 5
003136  016400  000000G          9$:    MOV     Q.ICSW(R4),R0   ;Get internal CSW value
003142  042700  000000C                 BIC     #^C<CS$EOF!CS$ERR>,R0 ;Clear all but error and eof flags
003146  050063  000000G                 BIS     R0,C.CSW(R3)    ;Transfer error and eof flags to user's CSW
003152  016400  000000C                 MOV     Q.ICSW+C.USED(R4),R0;Get highest block # written
003156  020063  000000G                 CMP     R0,C.USED(R3)   ;Compare with blk # in original chan blk
003162  101402                          BLOS    10$             ;Br if handler didn't increase
003164  010063  000000G                 MOV     R0,C.USED(R3)   ;Save high block # in original channel block
003170  105363  000000G          10$:   DECB    C.NUMQ(R3)      ;Dec # I/O operations on channel
                                ;
                                ;  See if user is waiting for I/O to finish.
                                ;
003174  005701                  12$:    TST     R1              ;IS THIS A SYSTEM I/O REQUEST?
003176  001406                          BEQ     2$              ;BR IF YES
003200  026127  000000G 000000G         CMP     LSTATE(R1),#S$IOWT ;IS USER WAITING FOR I/O TO FINISH?
003206  001002                          BNE     2$              ;BR IF NOT
003210  004737  000000G                 CALL    QHIPRI          ;PLACE USER IN RUN QUEUE
                                ;
                                ;  See if we need to queue a completion routine request for user.
                                ;
003214  026427  000000G 000001  2$:     CMP     Q.COMP(R4),#1   ;WAS A COMPLETION ROUTINE SPECIFIED?
003222  101451                          BLOS    3$              ;BR IF NOT
                                ;
                                ;  Queue a completion routine request for user.
                                ;  Convert I/O queue entry into completion queue entry format.
```

```
                                        ;
003224  016464  000000G 000000G          MOV     Q.CHAN(R4),CQ$R1(R4) ;CHANNEL # GOES IN R1 ON COMPL CALL
003232  016364  000000G 000000G          MOV     C.CSW(R3),CQ$R0(R4)  ;CSW GOES IN R0 ON COMPL CALL
003240  110164  000000G                 MOVB    R1,CQ$JOB(R4)        ;SET JOB #
003244  001415                          BEQ     8$                   ;Br if system completion
003246  005761  000000G                 TST     LITIME(R1)           ;Is job in an interactive state?
003252  001404                          BEQ     5$                   ;Br if not
003254  112764  000000G 000000G          MOVB    #S$HICP,CQ$RNS(R4);Set interactive-computation as state
003262  000403                          BR      6$
003264  112764  000000G 000000G 5$:     MOVB    #S$IOFN,CQ$RNS(R4);SET EXECUTION STATE FOR COMPLETION ROUTINE
003272  116164  000000G 000000G 6$:     MOVB    LPRI(R1),CQ$PRI(R4);Set execution priority value
003300  016464  000000G 000000G 8$:     MOV     Q.PA5(R4),CQ$PA5(R4);SET EMT ENTRY MAPPING FOR PAR 5
003306  116464  000000G 000000G          MOVB    Q.FLAG(R4),CQ$FLG(R4) ;Copy control flags
003314  112764  000000G 000000G          MOVB    #CP$STD,CQ$CP(R4);Set standard completion class priority
003322  132764  000000C 000000G          BITB    #<QF$MIO!QF$CIO>,CQ$FLG(R4);Secondary op for MIO or cache?
003330  001403                          BEQ     11$                  ;Br if not
003332  112764  000000G 000000G          MOVB    #CP$RT,CQ$CP(R4);Run cache completion routines at higher prio
003340  004737  003370'          11$:    CALL    QCOMPL               ;QUEUE A COMPLETION ROUTINE FOR USER
003344  000403                          BR      4$
                                        ;
                                        ;  We don't need to queue a completion routine
                                        ;  so just return I/O queue element to the free list.
                                        ;
003346  010401                  3$:     MOV     R4,R1                ;GET ADDRESS OF Q ENTRY TO R1
003350  004737  004222'                  CALL    QFREE                ;FREE THE I/O QUEUE ELEMENT
                                        ;
                                        ;  Finished with cleanup.
                                        ;
003354  012637  000000G         4$:     MOV     (SP)+,@#KPAR5
003360  012603                          MOV     (SP)+,R3
003362  012601                          MOV     (SP)+,R1
003364  012600                          MOV     (SP)+,R0
003366  000207                          RETURN
```

```
                                        .SBTTL  QCOMPL -- Queue a completion routine request
                                ;-------------------------------------------------------------------
                                ;  QCOMPL is called to queue a completion routine request for a user.
                                ;
                                ;  Inputs:
                                ;   R4 = Address of completion request queue element (must be set up).
                                ;
003370  010146                  QCOMPL: MOV     R1,-(SP)
003372  010246                          MOV     R2,-(SP)
                                ;
                                ;  See if this is a completion request for a system or user operation.
                                ;
003374  116401  0000000                 MOVB    CQ$JOB(R4),R1   ;GET JOB # ASSOCIATED WITH REQUEST
003400  001031                          BNE     1$              ;BR IF THIS IS A USER COMPL REQUEST
                                ;
                                ;  This is a completion request for a system operation.
                                ;  Call the completion routine directly.
                                ;
003402  016400  0000000                 MOV     CQ$R0(R4),R0    ;SET UP REGISTERS FOR COMPL ROUTINE
003406  016401  0000000                 MOV     CQ$R1(R4),R1
003412  010246                          MOV     R2,-(SP)
003414  010346                          MOV     R3,-(SP)
003416  010446                          MOV     R4,-(SP)
003420  010546                          MOV     R5,-(SP)
003422  013746  0000000                 MOV     @#KPAR5,-(SP)   ;Save current PAR 5 mapping
003426  016437  0000000 0000000         MOV     CQ$PA5(R4),@#KPAR5;Set up mapping for PAR 5 region
003434  004774  0000000                 CALL    @CQ$RTN(R4)     ;CALL SYSTEM COMPLETION ROUTINE
003440  012637  0000000                 MOV     (SP)+,@#KPAR5   ;Restore PAR 5 mapping
003444  012605                          MOV     (SP)+,R5
003446  012604                          MOV     (SP)+,R4
003450  012603                          MOV     (SP)+,R3
003452  012602                          MOV     (SP)+,R2
                                ;
                                ;  Finished calling system completion routine.
                                ;  Return queue element to free list.
                                ;
003454  010401                          MOV     R4,R1           ;GET ADDRESS OF COMPLETION QUEUE ENTRY
003456  004737  004222'                 CALL    @FREE           ;RETURN COMPL QUEUE ELEMENT TO FREE LIST
003462  000477                          BR      9$              ;FINISHED
                                ;
                                ;  This is a completion routine request for a user job.
                                ;  Add the completion request to the list of pending routines for this job.
                                ;
003464                          1$:     DISABL                  ;** Disable **
003472  010100                          MOV     R1,R0           ;Get current job number
003474  062700  000000C                 ADD     #LCMPL-CQ$LNK,R0;Get address of LCMPL(R1)
003500  010002                  2$:     MOV     R0,R2           ;Move addr of next compl queue element to R2
003502  016200  0000000                 MOV     CQ$LNK(R2),R0   ;Get addr of following queue element
003506  001416                          BEQ     3$              ;Br if we are at the end of the list
003510  126460  0000000 0000000         CMPB    CQ$CP(R4),CQ$CP(R0);Compare completion class priorities
003516  103770                          BLO     2$              ;Br if new element is of lower priority
003520  101011                          BHI     3$              ;Br if new element is of higher priority
003522  126460  0000000 0000000         CMPB    CQ$RNS(R4),CQ$RNS(R0);Compare state of new one to next
003530  101363                          BHI     2$              ;Br if new element is of lower priority state
003532  103404                          BLO     3$              ;Br if new element is of higher priority state
003534  126460  0000000 0000000         CMPB    CQ$PRI(R4),CQ$PRI(R0) ;States are the same, compare prio value
003542  101756                          BLOS    2$              ;Br if new priority is lower or equal
```

QCOMPL -- Queue a completion routine request

```
     58 003544  010064  0000000G          3$:     MOV     R0,CQ$LNK(R4)   ;CHAIN NEW ONE INTO LIST FOR JOB
     59 003550  010462  0000000G                  MOV     R4,CQ$LNK(R2)
     60 003554                                    ENABL                   ;** ENABLE **
     61                                  ;
     62                                  ;  Set default completion routine class priority if none was specified.
     63                                  ;
     64 003562  105764  0000000G                  TSTB    CQ$CP(R4)       ;Was a class priority specified?
     65 003566  001003                            BNE     7$              ;Br if yes
     66 003570  112764  0000000G 000000G         MOVB    #CP$STD,CQ$CP(R4);Set standard class priority
     67                                  ;
     68                                  ;  If the completion priority is at a real-time level, make sure
     69                                  ;  the execution state is S$RT
     70                                  ;
     71 003576  116400  0000000G          7$:     MOVB    CQ$PRI(R4),R0   ;Get execution priority for compl routine
     72 003602  120061  0000000G                  CMPB    R0,LPRI(R1)     ;Is current priority at least this high?
     73 003606  101402                            BLOS    5$              ;Br if yes
     74 003610  110061  0000000G                  MOVB    R0,LPRI(R1)     ;Boost priority till completion routine runs
     75 003614  120037  0000000G          5$:     CMPB    R0,VPRIHI       ;Is this a real-time priority?
     76 003620  103407                            BLO     6$              ;Br if not
     77 003622  126427  0000000G 000000G         CMPB    CQ$RNS(R4),#S$RT;Is real-time execution state specified?
     78 003630  101403                            BLOS    6$              ;Br if yes
     79 003632  112764  0000000G 000000G         MOVB    #S$RT,CQ$RNS(R4);Set real-time execution state
     80                                  ;
     81                                  ;  Place job in appropriate execution state (as specified in CQ$RNS)
     82                                  ;
     83 003640  116400  0000000G          6$:     MOVB    CQ$RNS(R4),R0   ;GET REQUESTED EXECUTION STATE FOR JOB
     84 003644  026100  0000000G                  CMP     LSTATE(R1),R0   ;IS JOB ALREADY IN THAT STATE OR HIGHER PRIO?
     85 003650  101402                            BLOS    4$              ;BR IF YES
     86 003652  004737  0000000G                  CALL    ENQTL           ;CHANGE JOB'S EXECUTION STATE
     87 003656  105237  0000000G          4$:     INCB    DOSCHD          ;REQUEST A JOB SCHEDULER CYCLE
     88                                  ;
     89                                  ;  Finished
     90                                  ;
     91 003662  012602                    9$:     MOV     (SP)+,R2
     92 003664  012601                            MOV     (SP)+,R1
     93 003666  000207                            RETURN
```

```
                        .SBTTL  FAKCMP -- Generate fake completion routine request
                ;-----------------------------------------------------------------------
                ;  FAKCMP is called to generate a completion queue request for I/O operations
                ;  such as TT I/O and spooled I/O that never enter a normal I/O queue request.
                ;
                ;  Inputs:
                ;   EMTBLK = EMT argument block for readc or writc.
                ;   CHNNUM = User's channel number.
                ;
003670  010146          FAKCMP: MOV     R1,-(SP)
003672  010446                  MOV     R4,-(SP)
                ;
                ;  Get an I/O queue entry
                ;
003674  004737  004102'         CALL    GETQ            ;GET AN I/O QUEUE ENTRY FROM FREE LIST
                ;
                ;  Set up queue entry to look like a completion queue request.
                ;  (Address of queue element is now in R1)
                ;
003700  113761  000000G 000000G         MOVB    CORUSR,CQ$JOB(R1);SET JOB # IN QUEUE ENTRY
003706  005061  000000G                 CLR     CQ$RO(R1)       ;SAY CSW IS ZERO (NO ERRORS)
003712  013761  000000G 000000G         MOV     CHNNUM,CQ$R1(R1);RETURN USER'S CHANNEL # IN R1
003720  013700  000010G                 MOV     EMTBLK+10,R0    ;GET ADDRESS OF COMPL ROUTINE
003724  004737  005352'                 CALL    UACHKW          ;MAKE SURE ADDRESS IS LEGAL
003730  103434                          BCS     9$              ;BR IF INVALID
003732  010061  000000G                 MOV     R0,CQ$RTN(R1)   ;SET COMPLETION ROUTINE ADDRESS
003736  013761  000000G 000000G         MOV     EMTMAP,CQ$PA5(R1);SET EMT ENTRY PAR 5 MAPPING
003744  112761  000000G 000000G         MOVB    #CP$STD,CQ$CP(R1);Set standard completion class priority
003752  113700  000000G                 MOVB    CORUSR,R0       ;Get job number
003756  116061  000000G 000000G         MOVB    LPRI(R0),CQ$PRI(R1);Set job execution priority
003764  112761  000000G 000000G         MOVB    #S$IOFN,CQ$RNS(R1);SET EXECUTION STATE FOR COMPL ROUTINE
003772  005761  000000G                 TST     LITIME(R1)      ;Is this an interactive job?
003776  001403                          BEQ     1$              ;Br if not
004000  112761  000000G 000000G         MOVB    #S$HICP,CQ$RNS(R1);Set execution state for interactive job
                ;
                ;  Queue completion request for this job.
                ;
004006  010104          1$:     MOV     R1,R4           ;GET ADDRESS OF COMPL Q ELEMENT TO R4
004010  004737  003370'         CALL    QCOMPL          ;QUEUE COMPL REQUEST FOR JOB
                ;
                ;  Finished
                ;
004014  012604                  MOV     (SP)+,R4
004016  012601                  MOV     (SP)+,R1
004020  000207                  RETURN
                ;
                ;  Error:  Invalid completion routine address.
                ;
004022  004737  004222'  9$:     CALL    QFREE           ;RETURN THE QUEUE ELEMENT
004026  012700  177766          MOV     #-12,R0         ;RETURN INVALID-EMT ERROR CODE
004032  000137  004036'         JMP     SETERR
```

```
                                        .SBTTL  SETERR -- Set EMT error code and exit
                                ;-------------------------------------------------------------
                                ;  SETERR is jumped to to set an EMT error code and exit from the
                                ;  EMT processing.
                                ;
                                ;  Inputs:
                                ;   R0 = EMT error code to be returned.
                                ;
004036  110037  000000G        SETERR: MOVB    R0,EMTERR       ;SAVE ERROR CODE
004042  002010                  BGE     ERROK           ;BR IF POSITIVE VALUE
                                ;
                                ;  We have a negative error code.
                                ;  This implies a fatal error unless a .SERR was done.
                                ;
004044  105737  000000G                 TSTB    SERFLG          ;WAS A .SERR DONE?
004050  001005                  BNE     ERROK           ;BR IF YES
                                ;  Fatal EMT error.
                                ;  Abort the job.
004052  010005                  MOV     R0,R5           ;GET ABORT CODE
004054  013704  000000G        EMTABT: MOV     EMTADR,R4       ;GET ADDRESS OF EMT INSTRUCTION
004060  000137  000000G                 JMP     ABORT           ;ABORT THE JOB
                                ;
                                ;  Non-fatal error.  Return with c-flag set.
                                ;
004064  105037  000000G        ERROK:  CLRB    INTERR          ;IGNORE INTERNAL ERROR CODE
004070  052737  000000G 000000G SETC:   BIS     #CFLAG,EMTPS    ;SET C-FLAG IN PS
004076  000137  001110'                 JMP     EMTXIT
```

```
                                        .SBTTL  GETQ   -- Get a free I/O queue element
                                ;-------------------------------------------------------------------
                                ;  GETQ is called to get a free I/O queue element.
                                ;  If there are no free elements available, the user is suspended until
                                ;  one becomes available.
                                ;
                                ;  Outputs:
                                ;  R1 = Address of I/O queue element obtained.
                                ;
004102                          GETQ:   DISABL                  ;;;DISABLE INTERRUPTS
                                ;
                                ;  Reserve the last few queue elements for system I/O use.
                                ;
004110  005737  000000G                 TST     NMFREQ          ;;;ANY QUEUE ELEMENTS WE CAN USE?
004114  003011                          BGT     1$              ;;;BR IF YES
                                ;
                                ;  There are no free I/O queue elements.
                                ;  Put user in queue waiting for one.
                                ;
004116  005237  000026'                INC     QWTCNT          ;;;Say another job waiting for queue element
004122  012700  000000G                MOV     #S$NEDQ,R0      ;;;QUEUE FOR JOBS WAITING FOR Q ELEMENTS
004126  004737  000000G                CALL    QNSPND          ;;;ADD JOB TO END OF QUEUE LIST (ENABLE INTS)
004132  004737  000000G                CALL    CHKABT          ;See if job was aborted while asleep
004136  000761                          BR      GETQ            ;GO BACK AND TRY TO GET A QUEUE ELEMENT
                                ;
                                ;  We have found a free queue element.
                                ;  Remove it from the free list.
                                ;
004140  005337  000000G         1$:     DEC     NMFREQ          ;;;DEC # FREE QUEUE ELEMENTS
004144  013701  000000G                 MOV     FREIOQ,R1       ;;;GET ADDRESS OF 1ST FREE Q ELEMENT
004150  016137  000000G 000000G         MOV     Q.LINK(R1),FREIOQ;;;REMOVE ELEMENT FROM FREE LIST
004156                                  ENABL                   ;Enable interrupts
                                ;
                                ;  Zero the I/O queue element
                                ;
004164  004737  004200'                 CALL    ZEROQ           ;ZERO THE QUEUE ELEMENT
                                ;
                                ;  Save info about current job context block mapping in queue element
                                ;
004170  013761  000000G 000000G         MOV     @#KPAR6,Q.PA6(R1) ;Save context block mapping
                                ;
                                ;  Finished
                                ;
004176  000207                          RETURN
                                ;
                                ;-------------------------------------------------------------------
                                ;  ZEROQ is called to zero an I/O queue element.
                                ;
                                ;  Inputs:
                                ;   R1 = Address of I/O queue element
                                ;
004200  010146                  ZEROQ:  MOV     R1,-(SP)
004202  010246                          MOV     R2,-(SP)
004204  012702  000000C                 MOV     #IOQSIZ/2,R2    ;GET # WORDS IN I/O QUEUE ELEMENT
004210  005021                  1$:     CLR     (R1)+           ;ZERO ALL WORDS IN QUEUE ELEMENT
004212  077202                          SOB     R2,1$
004214  012602                          MOV     (SP)+,R2
```

GETQ -- Get a free I/O queue element

```
     58 004216  012601                          MOV     (SP)+,R1
     59 004220  000207                          RETURN
```

```
                                        .SBTTL  QFREE  -- Return an I/O queue element to the free list
                                ;-------------------------------------------------------------------
                                ;  QFREE is called to return an I/O queue element to the free list.
                                ;
                                ;  Inputs:
                                ;  R1 = Address of queue element to be freed.
                                ;  All registers are preserve.
                                ;
004222  010046                  QFREE:  MOV     R0,-(SP)
004224                                  DISABL                  ;* DISABLE INTERRUPTS
004232  013700  0000000G                MOV     FREIOQ,R0       ;* GET CURRENT HEAD OF FREE LIST
004236  010061  0000000G                MOV     R0,Q.LINK(R1)   ;* CHAIN NEW ELEMENT INTO FREE LIST
004242  010137  0000000G                MOV     R1,FREIOQ       ;*
004246                                  ENABL                   ;ENABLE INTERRUPTS
004254  005237  0000000G                INC     NMFREQ          ;INC # FREE QUEUE ELEMENTS
004260  003415                        BLE     9$              ;BR IF NOT ENOUGH FOR USER JOBS TO USE
                                ;
                                ;  Restart users waiting for free I/O queue elements.
                                ;
004262  005737  000026'                TST     QWTCNT          ;Are any jobs waiting for a queue element?
004266  001412                        BEQ     9$              ;Br if not
004270  012700  0000000G                MOV     #S$NEDQ,R0      ;Get wait state
004274  004737  0000000G                CALL    QSRCH           ;See if any jobs waiting for queue element
004300  103403                        BCS     2$              ;Br if no jobs waiting
004302  004737  0000000G                CALL    QHIPRI          ;Requeue job in high-priority state
004306  000402                        BR      9$
004310  005037  000026'          2$:   CLR     QWTCNT          ;Restarted all waiting jobs
                                ;
                                ;  Finished
                                ;
004314  012600                  9$:   MOV     (SP)+,R0
004316  000207                        RETURN
```

```
                                        .SBTTL  GETSYQ -- Get queue element for system I/O
                                ;-------------------------------------------------------------------------
                                ;  GETSYQ is called to get a queue element to be used for system
                                ;  (kernel mode) I/O operations.
                                ;  GETSYQ gets a free queue element and sets up information in it.
                                ;
                                ;  Inputs:
                                ;   R1 = Address of system channel block to be used for I/O.
                                ;
                                ;  Outputs:
                                ;   R1 = Address of I/O queue element obtained.
                                ;
004320  010246                  GETSYQ: MOV     R2,-(SP)
004322  010102                          MOV     R1,R2           ;GET ADDRESS OF CHANNEL BLOCK
                                ;
                                ;  Get a free I/O queue element
                                ;
004324  004737  004414'                 CALL    GETRTQ          ;GET A QUEUE ELEMENT (ADDRESS RETURNED IN R1)
                                ;
                                ;  Set up information in queue element about the channel
                                ;
004330  010261  000000G                 MOV     R2,Q.UCSW(R1)   ;ADDRESS OF CSW FOR CHANNEL
004334  010100                          MOV     R1,R0           ;GET ADDRESS OF I/O QUEUE BLOCK
004336  062700  000000G                 ADD     #Q.ICSW,R0      ;POINT TO INTERNAL CSW WORD
004342  010061  000000G                 MOV     R0,Q.CSW(R1)    ;SET POINTER TO INTERNAL CSW WORD
004346  016210  000000G                 MOV     C.CSW(R2),(R0)  ;INITIALIZE CSW WORD
004352  016261  000000G 000000G         MOV     C.SBLK(R2),Q.BLKN(R1);BASE BLOCK # OF FILE
004360  116261  000000G 000000G         MOVB    C.DEVQ(R2),Q.UNIT(R1);DEVICE UNIT #
004366  012761  000000G 000000G         MOV     #VPAR6,Q.BUFF(R1);SET BUFFER ADDRESS TO MAP THROUGH PAR 6
004374  016202  000000G                 MOV     C.CSW(R2),R2    ;GET CHANNEL STATUS WORD
004400  042702  177701                  BIC     #^C76,R2        ;CLEAR ALL BUT DEVICE INDEX NUMBER
004404  110261  000000G                 MOVB    R2,Q.DEVX(R1)   ;SET DEVICE INDEX NUMBER
                                ;
                                ;  Finished
                                ;
004410  012602                          MOV     (SP)+,R2
004412  000207                          RETURN

                                        .SBTTL  GETRTQ -- Get queue element for real-time use
                                ;-------------------------------------------------------------------------
                                ;  GETRTQ is called to get a free I/O queue element for system or
                                ;  real-time use.
                                ;
                                ;  Outputs:
                                ;   R1 = Address of free queue element obtained.
                                ;   A system crash occurs if there are no free queue elements.
                                ;
004414                          GETRTQ: DISABL                  ;** DISABLE **
004422  013701  000000G                 MOV     FREIOQ,R1       ;GET ADDRESS OF A FREE I/O QUEUE ELEMENT
004426  001413                          BEQ     1$              ;SYSTEM CRASH IF NO FREE ELEMENTS AVAILABLE
004430  016137  000000G 000000G         MOV     Q.LINK(R1),FREIOQ;REMOVE QUEUE ELEMENT FROM FREE LIST
004436  005337  000000G                 DEC     NMFREQ          ;DEC # AVAILABLE QUEUE ELEMENTS
004442                                  ENABL                   ;** ENABLE **
004450  004737  004200'                 CALL    ZEROQ           ;ZERO THE QUEUE ELEMENT
004454  000207                          RETURN
                                ;
                                ;  Fatal system error -- No free I/O queue elements available.
```

GETRTQ -- Get queue element for real-time use

```
                                        ;
004456                          1$:     DIE     #EM$NQE         ;NO QUEUE ELEMENTS AVAILABLE
```

```
                                        .SBTTL  GETUMR -- Allocate Unibus map register
                                ;-----------------------------------------------------------------------
                                ;  GETUMR is called to determine if a Unibus map register set needs to be
                                ;  allocated for an I/O operation.  If unibus mapping is needed for the
                                ;  operation, GETUMR attempts to find a free unibus map register set large
                                ;  enough to handle the operation.  If a free UMR can be found, it is
                                ;  allocated to the operation.  If no free UMR can be found, the I/O
                                ;  queue request is placed on a waiting list and an exception condition
                                ;  is signaled on return.
                                ;
                                ;  Inputs:
                                ;   R1 = Address of I/O queue entry.
                                ;   R2 = Device index number of device I/O is being directed to.
                                ;
                                ;  Outputs:
                                ;   C-flag clear ==> A UMR was successfully allocated.
                                ;   C-flag set   ==> No free UMR.  Defer I/O request.
                                ;   Q.UMRX = Address of UMR descriptor block for UMR that was allocated.
                                ;   Q.UMPB = Original value of Q.BUFF
                                ;   Q.UMPP = Original value of Q.PAR
                                ;   Q.UMVB = Number of base UMR being used by operation.
                                ;
                                ;  See if this device needs unibus mapping
                                ;
004470  005761  000000G        GETUMR: TST     Q.UMRX(R1)          ;Have we already allocated a UMR?
004474  001007                        BNE     9$                  ;Br if yes
004476  105737  000000G               TSTB    UBUSMP              ;IS UNIBUS MAPPING NEEDED AT ALL?
004502  001404                        BEQ     9$                  ;BR IF NOT
004504  032762  000000G 000000G       BIT     #DX$DMA,DVFLAG(R2)  ;IS THIS A DMA DEVICE?
004512  001002                        BNE     11$                 ;BR IF YES
004514  000241                  9$:   CLC                         ;SIGNAL SUCCESS ON RETURN
004516  000207                        RETURN
                                ;
                                ;  We must do unibus mapping for this I/O operation.
                                ;
004520  010246                  11$:  MOV     R2,-(SP)
004522  010346                        MOV     R3,-(SP)
004524  010446                        MOV     R4,-(SP)
004526  010546                        MOV     R5,-(SP)
                                ;
                                ;  Determine how many words are being transferred
                                ;
004530  016103  000000G               MOV     Q.WCNT(R1),R3       ;GET SPECIFIED WORD COUNT
004534  002001                        BGE     10$                 ;BR IF VALUE POSITIVE
004536  005403                        NEG     R3                  ;GET POSITIVE WORD COUNT
                                ;
                                ;  Find the smallest free UMR set that is large enough for this transfer
                                ;
004540  012704  000000G         10$:  MOV     #UMRBAS,R4          ;POINT TO BASE OF UMR DESCRIPTOR BLOCKS
004544                                DISABL                      ;** DISABLE **
004552  005764  000000G         1$:   TST     UM$IOQ(R4)          ;IS THIS UMR SET FREE?
004556  001003                        BNE     5$                  ;BR IF NOT
004560  020364  000000G               CMP     R3,UM$WDS(R4)       ;IS THIS UMR SET LARGE ENOUGH?
004564  101425                        BLOS    4$                  ;BR IF YES
004566  062704  000000G         5$:   ADD     #UM$$SZ,R4          ;POINT TO NEXT UMR DESCRIPTOR
004572  020427  000000G               CMP     R4,#UMREND          ;CHECKED ALL?
004576  103765                        BLO     1$                  ;BR IF NOT
```

```
                                        ;
                                        ;  There are no free UMR register sets of adequate size.
                                        ;  Put I/O queue entry on list waiting for a free UMR.
                                        ;
004600  012704  000000C                       MOV     #<UMRWHD-Q.LINK>,R4 ;POINT TO HEAD OF WAIT LIST
004604  016405  0000000         2$:           MOV     Q.LINK(R4),R5       ;IS THIS THE LAST ENTRY IN THE LIST?
004610  001402                                BEQ     3$                  ;BR IF YES
004612  010504                                MOV     R5,R4               ;LINK TO NEXT ENTRY
004614  000773                                BR      2$                  ;GO CHECK IT
004616  010164  0000000         3$:           MOV     R1,Q.LINK(R4)       ;ADD OUR I/O QUEUE ENTRY TO TAIL OF LIST
004622  005061  0000000                       CLR     Q.LINK(R1)          ;SAY WE ARE THE LAST ENTRY IN LIST
004626                                        ENABL                       ;** ENABLE **
004634  000261                                SEC                         ;SIGNAL FAILURE
004636  000450                                BR      8$                  ;GO RETURN
                                        ;
                                        ;  We found a free UMR.  Claim it for our I/O request.
                                        ;
004640  010164  0000000         4$:           MOV     R1,UM$IOQ(R4)       ;SAY UMR SET IS IN USE
004644                                        ENABL                       ;** ENABLE **
004652  010461  0000000                       MOV     R4,Q.UMRX(R1)       ;REMEMBER WHICH UMR SET IS USED BY OPERATION
                                        ;
                                        ;  Set up information in I/O queue entry that will be used by MPPHY
                                        ;  routine to calculate unibus virtual address.
                                        ;   Q.UMPB = Original value of Q.BUFF
                                        ;   Q.UMPP = Original value of Q.PAR
                                        ;   Q.UMVB = Number of base UMR being used by transfer.
                                        ;
004656  016105  0000000                       MOV     Q.BUFF(R1),R5       ;GET CURRENT BUFFER ADDRESS RELATIVE TO PAR6
004662  010561  0000000                       MOV     R5,Q.UMPB(R1)       ;SAVE ORIGINAL BUFFER ADDRESS
004666  016103  0000000                       MOV     Q.PAR(R1),R3        ;GET PAR6 BASE ADDRESS
004672  010361  0000000                       MOV     R3,Q.UMPP(R1)       ;Save original value of Q.PAR
004676  116461  0000000 0000000             MOVB    UM$UMR(R4),Q.UMVB(R1) ;BASE UMR #
                                        ;
                                        ;  Set up mapping base value in unibus map register.
                                        ;
004704  162705  0000000                       SUB     #VPAR6,R5           ;REMOVE PAR6 BIAS FROM BUFFER ADDRESS
004710  005002                                CLR     R2                  ;SET UP HIGH-ORDER REGISTER (R2-R3)
004712  073227  000006                        ASHC    #6,R2               ;SHIFT Q.PAR VALUE 6 PLACES (R2-R3)
004716  060503                                ADD     R5,R3               ;ADD IN BUFFER BASE OFFSET
004720  005502                                ADC     R2                  ;PROPOGATE CARRY
004722  116405  0000000                       MOVB    UM$UMR(R4),R5       ;GET UNIBUS MAP REGISTER #
004726  072527  000002                        ASH     #2,R5               ;*4 BYTES PER REGISTER
004732  062705  0000000                       ADD     #UMRADR,R5          ;GET ADDRESS OF UNIBUS MAP REGISTER
004736  116400  0000000                       MOVB    UM$NMR(R4),R0       ;GET # MAP REGISTERS USED IN THIS SET
004742  010325                    6$:         MOV     R3,(R5)+            ;SET LOW ORDER VALUE
004744  010225                                MOV     R2,(R5)+            ;SET HIGH ORDER VALUE
004746  062703  020000                        ADD     #8192.,R3           ;ADVANCE ADDRESS VALUE
004752  005502                                ADC     R2                  ;PROPOGATE CARRY
004754  077006                                SOB     R0,6$               ;LOOP IF MORE REGISTERS IN SET
                                        ;
                                        ;  Finished
                                        ;
004756  000241                    7$:         CLC                         ;SIGNAL SUCCESS ON RETURN
004760  012605                    8$:         MOV     (SP)+,R5
004762  012604                                MOV     (SP)+,R4
004764  012603                                MOV     (SP)+,R3
004766  012602                                MOV     (SP)+,R2
```

```
    115 004770  000207                         RETURN
```

```
                                        .SBTTL  FREUMR -- Free a Unibus map register
                                ;-------------------------------------------------------------------
                                ;  FREUMR is called to free a unibus map register set associated with an
                                ;  I/O request.
                                ;  If there is an I/O request waiting for a free UMR set and the set being
                                ;  freed is large enough for the waiting request, the request is removed
                                ;  from the wait list and is started.
                                ;
                                ;  Inputs:
                                ;   R1 = Address of I/O queue element that is being freed.
                                ;
                                ;  See if there is a UMR associated with this request.
                                ;
004772  005761  0000000         FREUMR: TST     Q.UMRX(R1)      ;IS THERE A UMR ASSOCIATED WITH THIS REQUEST?
004776  001464                          BEQ     9$              ;BR IF NOT
                                ;
                                ;  This I/O queue does have an associated UMR.
                                ;
005000  010146                          MOV     R1,-(SP)
005002  010246                          MOV     R2,-(SP)
005004  010346                          MOV     R3,-(SP)
005006  010446                          MOV     R4,-(SP)
005010  010546                          MOV     R5,-(SP)
                                ;
                                ;  Free the UMR
                                ;
005012  016104  0000000                 MOV     Q.UMRX(R1),R4   ;GET ADDRESS OF UMR DESCRIPTOR BLOCK
005016  005064  0000000                 CLR     UM$IOQ(R4)      ;SAY THE UMR IS FREE
005022  005061  0000000                 CLR     Q.UMRX(R1)      ;SAY NO UMR ASSOCIATED WITH I/O REQUEST
                                ;
                                ;  See if there is any I/O queue request waiting for a free UMR
                                ;  that could be satisfied with the UMR that is being freed.
                                ;
005026  016405  0000000                 MOV     UM$WDS(R4),R5   ;GET SIZE OF UMR SET THAT IS BEING FREED
005032                                  DISABL                  ;** DISABLE **
005040  012702  000000C                 MOV     #<UMRWHD-Q.LINK>,R2 ;POINT TO HEAD OF WAIT LIST
005044  000406                          BR      3$
005046  016200  0000000         1$:     MOV     Q.WCNT(R2),R0   ;GET # WORDS NEEDED BY THIS I/O REQUEST
005052  002001                          BGE     2$
005054  005400                          NEG     R0              ;GET POSITIVE VALUE
005056  020005                  2$:     CMP     R0,R5           ;IS THE FREED UMR LARGE ENOUGH?
005060  101410                          BLOS    4$              ;BR IF YES
005062  010203                  3$:     MOV     R2,R3           ;SAVE ADDRESS OF THIS I/O REQUEST
005064  016202  0000000                 MOV     Q.LINK(R2),R2   ;GET ADDRESS OF NEXT WAITING I/O REQUEST
005070  001366                          BNE     1$              ;BR IF THERE IS ANOTHER
005072                                  ENABL                   ;** ENABLE **
005100  000416                          BR      8$              ;THERE ARE NO WAITING ENTRIES TO BE STARTED
                                ;
                                ;  We found a waiting I/O queue entry that could be started by
                                ;  the UMR that has just been freed/
                                ;  Remove I/O queue entry from wait list and try to start it.
                                ;
005102  016263  0000000 0000000 4$:     MOV     Q.LINK(R2),Q.LINK(R3) ;REMOVE I/O QUEUE ENTRY FROM WAIT LIST
005110                                  ENABL                   ;** ENABLE **
005116  010201                          MOV     R2,R1           ;GET ADDRESS OF I/O QUEUE ENTRY TO R1
005120  116102  0000000                 MOVB    Q.DEVX(R1),R2   ;Get device index number
005124  004737  004470'                 CALL    GETUMR          ;We should be able to get a free UMR now
```

```
     58 005130  103402                        BCS     8$              ;Br if could not get UMR
     59 005132  004737  002416'                CALL    IOHANQ          ;Queue this request to the device handler
     60                                ;
     61                                ;  Finished
     62                                ;
     63 005136  012605                 8$:    MOV     (SP)+,R5
     64 005140  012604                        MOV     (SP)+,R4
     65 005142  012603                        MOV     (SP)+,R3
     66 005144  012602                        MOV     (SP)+,R2
     67 005146  012601                        MOV     (SP)+,R1
     68 005150  000207                 9$:    RETURN
```

```
                                        .SBTTL  GETUCH -- Get character from user's buffer
                                ;-------------------------------------------------------------------
                                ;  GETUCH is called to get a character from the user's buffer.
                                ;
                                ;  Inputs:
                                ;   R3 = Address of byte to be fetched.
                                ;
                                ;  Outputs:
                                ;   R0 = Byte we got.
                                ;   R3 = Incremented.
                                ;
005152  032703  000001          GETUCH: BIT     #1,R3           ;GETTING ODD OR EVEN BYTE?
005156  001004                          BNE     1$              ;BR IF ODD
                                ;
                                ;  Byte is on a word boundary
                                ;
005160  106513                          MFPD    (R3)            ;GET WORD CONTAINING BYTE
005162  005203                          INC     R3              ;ADVANCE R3
005164  112600                          MOVB    (SP)+,R0        ;RETURN BYTE IN R0
005166  000207                          RETURN
                                ;
                                ;  Byte address is odd
                                ;
005170  005303                  1$:     DEC     R3              ;POINT TO START OF WORD WITH BYTE
005172  106523                          MFPD    (R3)+           ;GET WORD WITH BYTE
005174  012600                          MOV     (SP)+,R0
005176  105000                          CLRB    R0              ;CLEAR LOW-ORDER BYTE
005200  000300                          SWAB    R0              ;MOVE HIGH-ORDER BYTE TO LOW-ORDER
005202  000207                          RETURN
```

```
                                        .SBTTL  PUTUCH -- Move byte to user's buffer
                                ;-------------------------------------------------------------------
                                ;  PUTUCH is called to move a byte into the user's buffer.
                                ;
                                ;  Inputs:
                                ;   R0 = Byte to be stored.
                                ;   R3 = Address where byte is to be stored.
                                ;
                                ;  Outputs:
                                ;   R3 = Incremented.
                                ;
005204  032703  000001          PUTUCH: BIT     #1,R3           ;IS BYTE ADDRESS ODD OR EVEN?
005210  001005                          BNE     1$              ;BR IF ODD
                                ;
                                ;  Byte address is even.
                                ;
005212  106513                          MFPD    (R3)            ;GET WORD WHERE BYTE IS TO BE STORED
005214  110016                          MOVB    R0,(SP)         ;PUT IN OUR BYTE
005216  106613                          MTPD    (R3)            ;REPLACE WORD
005220  005203                          INC     R3              ;INCREMENT R3
005222  000406                          BR      2$
                                ;
                                ;  Byte address is odd.
                                ;
005224  005303                  1$:     DEC     R3              ;POINT TO WORD WITH BYTE
005226  106513                          MFPD    (R3)            ;GET WORD WITH BYTE
005230  000316                          SWAB    (SP)            ;GET BYTE TO LOW-ORDER
005232  110016                          MOVB    R0,(SP)         ;PUT IN NEW BYTE
005234  000316                          SWAB    (SP)            ;REPOSITION
005236  106623                          MTPD    (R3)+           ;STORE BACK INTO USER'S BUFFER
005240  000207                  2$:     RETURN
```

```
                                        .SBTTL  SYBFAD -- Set system buffer address in I/O queue entry
                                ;-------------------------------------------------------------------------
                                ;  SYBFAD is called to convert a buffer address in system space into an
                                ;  address that will be mapped through PAR 6.  This address is stored
                                ;  into a specified I/O queue element.
                                ;  The buffer address may be in low physical memory (PAR 0 through PAR 4),
                                ;  in a mapped system overlay region (PAR 5), or in the user context
                                ;  block (PAR 6).
                                ;
                                ;  Inputs:
                                ;   R0 = Physical address to be converted.
                                ;   R1 = Address of I/O queue entry to be set up.
                                ;
                                ;  Outputs:
                                ;   Q.PAR(R1) & Q.BUFF(R1)  are set up.
                                ;
005242  010046                  SYBFAD: MOV     R0,-(SP)
005244  010246                          MOV     R2,-(SP)
                                ;
                                ;  Determine which region buffer is in.
                                ;
005246  020027  0000000                 CMP     R0,#CXTBAS      ;Is buffer in job context block?
005252  103407                          BLO     2$              ;Br if not in context block
                                ;
                                ;  Buffer is in job context block
                                ;
005254  113702  0000000                 MOVB    CORUSR,R2       ;Get current job index number
005260  016202  0000000                 MOV     LBASE(R2),R2    ;Get base 256-word block number of context blk
005264  072227  000003                  ASH     #3,R2           ;Convert to 32-word block number
005270  000421                          BR      9$              ;(Note: virtual address in R0 already in PAR 6
                                ;
                                ;  See if we are accessing a buffer in a system mapped region
                                ;
005272  020027  0000000         2$:     CMP     R0,#VPAR5       ;Is buffer in a system mapped region?
005276  103405                          BLO     12$             ;Br if not
                                ;
                                ;  Buffer is in a system mapped region
                                ;
005300  013702  0000000                 MOV     @#KPAR5,R2      ;Get current mapping for kernel PAR 5
005304  062700  020000                  ADD     #20000,R0       ;Bias virtual address to PAR 6 region
005310  000411                          BR      9$
                                ;
                                ;  We are accessing a buffer stored in low physical memory (PAR 0-4)
                                ;
005312  010002                  12$:    MOV     R0,R2           ;Get actual address
005314  000241                          CLC                     ;Convert to page number
005316  006002                          ROR     R2
005320  072227  177773                  ASH     #-5,R2
005324  042700  177700                  BIC     #^C77,R0        ;Get byte-in-page to R0
005330  062700  0000000                 ADD     #VPAR6,R0       ;Map through PAR 6
                                ;
                                ;  Store values into I/O queue element
                                ;
005334  010261  0000000         9$:     MOV     R2,Q.PAR(R1)    ;Page relocation bias
005340  010061  0000000                 MOV     R0,Q.BUFF(R1)   ;Buffer address
                                ;
                                ;  Finished
```

SYBFAD -- Set system buffer address in I/O queue entry

```
     58                                         ;
     59 005344  012602                          MOV     (SP)+,R2
     60 005346  012600                          MOV     (SP)+,R0
     61 005350  000207                          RETURN
```

```
                                        .SBTTL  UACHKx -- Check validity of user address
                                ;-------------------------------------------------------------------------
                                ;  UACHKW and UACHKB are called to determine if an emt address is
                                ;  within the valid region of the current job.
                                ;  UACHKW also checks to see that the address is on a word boundary.
                                ;
                                ;  Inputs:
                                ;   R0 = Address to be checked (preserved)
                                ;   EMTPS = PS saved by EMT
                                ;   UHIMEM = Top address of job region
                                ;
                                ;  Outputs:
                                ;   C-flag set on return if address is invalid.
                                ;
                                ;  Check a word address
                                ;
005352  032700  000001        UACHKW: BIT     #1,R0           ;IS ADDRESS EVEN?
005356  001402                        BEQ     UACHKB          ;BR IF YES
005360  000261                        SEC                     ;SIGNAL ERROR
005362  000207                        RETURN
                                ;
                                ;  Check byte address
                                ;
                                ;  See if address is in normal job region
005364  020037  0000000         UACHKB: CMP     R0,UHIMEM       ;IS ADDRESS IN NORMAL JOB REGION
005370  101404                        BLOS    4$              ;BR IF YES
                                ;
                                ;  Address is not in normal job region.
                                ;  Allow kernel-mode emt's to access all of memory
                                ;
005372  032737  0000000 0000000       BIT     #UMODE,EMTPS    ;IS THIS A KERNEL MODE EMT?
005400  001002                        BNE     5$              ;BR IF NOT
005402  000241                4$:     CLC                     ;SIGNAL SUCCESSFUL RETURN
005404  000207                        RETURN
                                ;
                                ;  This is a user-mode emt accessing a region above the normal top of job
                                ;
005406  010146                5$:     MOV     R1,-(SP)
005410  010246                        MOV     R2,-(SP)
005412  113701  0000000             MOVB    CORUSR,R1       ;GET CURRENT JOB INDEX NUMBER
                                ;
                                ;  Allow KMON to access job context area
                                ;
005416  032761  0000000 0000000       BIT     #$INKMN,LSW4(R1);IS KMON RUNNING?
005424  001403                        BEQ     2$              ;BR IF NOT
005426  020027  0000000             CMP     R0,#CXTEND      ;IS ADDRESS IN JOB CONTEXT REGION?
005432  103426                        BLO     1$              ;BR IF YES
                                ;
                                ;  See if there are any associated shared run-time systems
                                ;
005434  010001                2$:     MOV     R0,R1           ;GET ADDRESS BEING CHECKED
005436  072127  177764              ASH     #-12.,R1        ;CONVERT ADDRESS TO 2 * PAR REGION #
005442  042701  177761              BIC     #^C16,R1
005446  016102  0000000             MOV     RPDR(R1),R2     ;IS THAT REGION MAPPED TO SHARED RUN-TIME?
005452  001414                        BEQ     3$              ;BR IF NOT
005454  000302                        SWAB    R2              ;GET PAGE LENGTH TO LOW-ORDER BYTE
005456  042702  177600              BIC     #^C177,R2       ;GET # 64-BYTE BLOCKS USED IN PAR REGION
```

```
UACHKx -- Check validity of user address

 005462  005202                    INC     R2              ;# IS OFFSET BY 1
 005464  072227  000006            ASH     #6,R2           ;CONVERT TO # BYTES
 005470  005302                    DEC     R2              ;AVOID ADDRESS OVERFLOW AT TOP OF MEMORY
 005472  072127  000014            ASH     #12.,R1         ;GET ADDRESS OF BASE OF PAR REGION
 005476  060201                    ADD     R2,R1           ;GET ADDRESS OF LAST BYTE IN PAR REGION
 005500  020001                    CMP     R0,R1           ;IS OUR ADDRESS IN MAPPED REGION?
 005502  101402                    BLOS    1$              ;BR IF YES
                                ;
                                ;  Address is invalid
                                ;
 005504  000261                 3$:     SEC                     ;SIGNAL ERROR ON RETURN
 005506  000401                         BR      9$
                                ;
                                ;  Address is valid
                                ;
 005510  000241                 1$:     CLC                     ;SIGNAL SUCCESS ON RETURN
                                ;
                                ;  Finished
                                ;
 005512  012602                 9$:     MOV     (SP)+,R2
 005514  012601                         MOV     (SP)+,R1
 005516  000207                         RETURN

                                        .SBTTL  VALADx -- Validate user address
                                ;-----------------------------------------------------------------------
                                ;  VALADW and VALADB are called to validate an address provided as
                                ;  an EMT argument.  If the address is invalid, these routines do not
                                ;  return but rather produce a fatal EMT error.
                                ;  VALADW checks to see that the address is even.
                                ;
                                ;  Inputs:
                                ;   R0 = Address to be checked (preserved)
                                ;   EMTPS = PS saved by EMT
                                ;
                                ;  Outputs:
                                ;   A fata EMT error is produced if the address is invalid.
                                ;
 005520  004737  005352'        VALADW: CALL    UACHKW          ;CHECK WORD ADDRESS
 005524  103405                         BCS     BADAD           ;BR IF INVALID
 005526  000207                         RETURN                  ;ADDRESS IS OK

 005530  004737  005364'        VALADB: CALL    UACHKB          ;CHECK BYTE ADDRESS
 005534  103401                         BCS     BADAD           ;BR IF INVALID
 005536  000207                         RETURN
                                ;
                                ;  Invalid address -- Give fatal emt error
                                ;
 005540  012700  177766         BADAD:  MOV     #-12,R0         ;GIVE FATAL EMT ERROR
 005544  000137  004036'                 JMP     SETERR
```

```
                                        .SBTTL  IOWAIT -- Wait for I/O to finish
                                ;-------------------------------------------------------------------
                                ;  IOWAIT is called to suspend user execution until all I/O on the
                                ;  current channel is finished.
                                ;
                                ;  Inputs:
                                ;   CHNADR = Address of current channel we are waiting on.
                                ;
005550  010346                 IOWAIT: MOV     R3,-(SP)
005552  013703  0000000G              MOV     CHNADR,R3       ;GET ADDRESS OF CHANNEL
005556                          1$:     DISABL                  ;* DISABLE INTERRUPTS
005564  105763  0000000G              TSTB    C.NUMQ(R3)      ;* ANY ACTIVE I/O REQUESTS ON CHANNEL?
005570  001407                       BEQ     2$              ;* BR IF NOT -- ALL I/O IS FINISHED
005572  012700  0000000G              MOV     #S$IOWT,R0      ;* SAY WE ARE WAITING FOR I/O TO FINSIH
005576  004737  0000000G              CALL    QNSPND          ;ENQUEUE FOR I/O WAIT  (ENABLE)
005602  004737  0000000G              CALL    CHKABT          ;WERE WE ABORTED WHILE ASLEEP?
005606  000763                       BR      1$              ;GO TRY AGAIN
005610                          2$:     ENABL
005616  012603                       MOV     (SP)+,R3
005620  000207                       RETURN

                                        .SBTTL  IOSTOP -- Wait for all of job's I/O to finish
                                ;-------------------------------------------------------------------
                                ;  IOSTOP suspends execution of the current job until all of its
                                ;  I/O is completed.
                                ;
005622  010146                 IOSTOP: MOV     R1,-(SP)
005624  113701  0000000G              MOVB    CORUSR,R1       ;GET JOB INDEX #
005630                          1$:     DISABL                  ;*** DISABLE ***
005636  105761  0000000G              TSTB    LIOCNT(R1)      ;IS THERE ANY I/O ACTIVE FOR JOB?
005642  001405                       BEQ     2$              ;BR IF NOT
005644  012700  0000000G              MOV     #S$IOWT,R0      ;SUSPEND JOB TILL I/O FINISHES
005650  004737  0000000G              CALL    QNSPND
005654  000765                       BR      1$              ;NOW GO SEE IF IT HAS COMPLETED
                                ;
                                ;  All I/O is finished
                                ;
005656                          2$:     ENABL                   ;** ENABLE **
005664  012601                       MOV     (SP)+,R1
005666  000207                       RETURN

                                        .SBTTL  IOHALT -- Abort all I/O for a job
                                ;-------------------------------------------------------------------
                                ;  Process .ABTIO EMT
                                ;
005670  005737  0000000G        ABTIO:  TST     IOABFL          ;CHECK TO SEE IF IO ABORT WAS CHOSEN
005674  001752                       BEQ     IOSTOP          ;BR IF IO RUNDOWN WAS CHOSEN
005676  010246                       MOV     R2,-(SP)        ;Save registers
005700  010346                       MOV     R3,-(SP)
005702  013703  0000000G              MOV     CHNADR,R3       ;Get address of channel status word
005706  032763  0000000G 0000000G     BIT     #CS$OPN,C.CSW(R3);See if channel is open
005714  001406                       BEQ     1$              ;Br if channel is not open
005716  016302  0000000G              MOV     C.CSW(R3),R2    ;Get channel status word
005722  042702  177701               BIC     #^C76,R2        ;Mask channel device index number
005726  004737  006006'              CALL    DVSTOP          ;Stop I/O to this device
005732  012603                 1$:     MOV     (SP)+,R3        ;Restore registers
005734  012602                       MOV     (SP)+,R2
```

```
     58 005736  000137  001110'                JMP     EMTXIT          ;Finished
     59
     60                                 ;-----------------------------------------------------------------------
     61                                 ;  IOHALT is called to abort all I/O operations for a particular job.
     62                                 ;
     63                                 ;  Inputs:
     64                                 ;   R1 = Job index number.
     65                                 ;
     66 005742  005737  000000G         IOHALT: TST     IOABFL          ;CHECK TO SEE IF IO ABORT WAS CHOSEN
     67 005746  001725                          BEQ     IOSTOP          ;BR IF IO RUNDOWN WAS CHOSEN
     68 005750  010246                          MOV     R2,-(SP)
     69 005752  010346                          MOV     R3,-(SP)
     70                                 ;
     71                                 ;  Cancel any pending .TIMIO entries for this job
     72                                 ;
     73 005754  004737  006626'                 CALL    CANIOT          ;Cancel any pending .TIMIO entries for job
     74                                 ;
     75                                 ;  Now call handler abort entry points
     76                                 ;
     77 005760  013702  000000G                 MOV     NUMDEV,R2       ;GET INDEX # OF LAST DEVICE
     78 005764  005003                          CLR     R3              ;Clear channel address pointer
     79 005766  004737  006006'         1$:     CALL    DVSTOP          ;STOP I/O ON THAT DEVICE FOR THIS JOB
     80 005772  162702  000002                  SUB     #2,R2           ;GET NEXT DEVICE INDEX NUMBER
     81 005776  002373                          BGE     1$              ;BR IF MORE DEVICES
     82 006000  012603                          MOV     (SP)+,R3
     83 006002  012602                          MOV     (SP)+,R2
     84 006004  000207                          RETURN
```

```
                                        .SBTTL  DVSTOP -- Abort I/O for a device & job
                                ;-------------------------------------------------------------------
                                ;  DVSTOP is called to abort all I/O requests for a particular job
                                ;  that are pending for a particular device.
                                ;  The abort entry point for the device handler is called if the
                                ;  current queue entry belongs to the specified job or if the abort
                                ;  flag is set for the handler.
                                ;
                                ;  Inputs:
                                ;   R1 = Index number of job whose I/O is to be aborted.
                                ;   R2 = Device index number of device to be serviced.
                                ;   R3 = Address of channel status word (for .ABTIO) or zero
                                ;
006006  010346                  DVSTOP: MOV     R3,-(SP)
006010  010446                          MOV     R4,-(SP)
006012  010546                          MOV     R5,-(SP)
006014  013746  0000000                 MOV     @#KPAR5,-(SP)   ;Save PAR 5 in case we map to a handler
                                ;
                                ;  Return immediately if this is a pseudo-device like LD, TT, etc.
                                ;
006020  026227  0000000 000004          CMP     HANENT(R2),#4   ;Is this a real device?
006026  101532                          BLOS    7$              ;Br if not
                                ;
                                ;  Enter system state while we perform I/O abort processing
                                ;
006030  012700  006314'                 MOV     #7$,R0          ;Set exit address for system state
006034  004737  0000000                 CALL    ENSYS           ;Enter system state
006040  0000000                         .WORD   FP$IOA          ;Fork processing priority
006042  010305                          MOV     R3,R5           ;Move channel status word
                                ;
                                ;  Set up pointer to handler.
                                ;
006044  016203  0000000                 MOV     HANPAR(R2),R3   ;Is this a mapped handler?
006050  001402                          BEQ     8$              ;Br if not
006052  010337  0000000                 MOV     R3,@#KPAR5      ;Map PAR 5 to the handler
006056  016203  0000000         8$:     MOV     HANENT(R2),R3   ;Get pointer to LQE cell in handler
                                ;
                                ;  Hold the handler.
                                ;  This prevents new I/O from being started and prevents cleanup on
                                ;  current I/O request for handler.
                                ;
006062  012763  100000  177776          MOV     #100000,-2(R3)  ;Set handler-hold flag
006070  005723                          TST     (R3)+           ;Point to CQE cell in handler
                                ;
                                ;  Call handler abort entry point if abort flag is set for handler
                                ;  or if current queue element belongs to job being aborted.
                                ;
006072  032762  0000000 0000000         BIT     #DS$ABT,DVSTAT(R2);Is abort flag set for handler?
006100  001015                          BNE     1$              ;Br if yes
006102  011304                          MOV     (R3),R4         ;Get pointer to current queue element for hand
006104  001430                          BEQ     2$              ;Br if none
006106  116400  000000C                 MOVB    Q.JOB-Q.BLKN(R4),R0 ;Get job # from queue element
006112  006300                          ASL     R0              ;Convert to job index number
006114  020001                          CMP     R0,R1           ;Does this element belong to job being aborted
006116  001023                          BNE     2$              ;Br if not
006120  016400  000000C                 MOV     Q.CSW-Q.BLKN(R4),R0;Get address of Channel Status Word
006124  001403                          BEQ     1$              ;Br if none
```

```
     58 006126  052760  0000000 0000000          BIS     #CS$ERR,C.CSW(R0);Say an error occurred on this operation
     59                                ;
     60                                ;  Call handler abort entry point.
     61                                ;
     62 006134  010146                 1$:      MOV     R1,-(SP)
     63 006136  010246                          MOV     R2,-(SP)
     64 006140  010346                          MOV     R3,-(SP)
     65 006142  010104                          MOV     R1,R4           ;Get job number
     66 006144  006204                          ASR     R4              ;Get in form handler expects
     67 006146  162703  000006              SUB     #6,R3           ;Point to handler cell with interrupt entry
     68 006152  011300                          MOV     (R3),R0         ;Get offset to interrupt entry pt in handler
     69 006154  060300                          ADD     R3,R0           ;Get abs address of interrupt entry point
     70 006156  004740                          CALL    -(R0)           ;Call handler abort entry point
     71 006160  012603                          MOV     (SP)+,R3
     72 006162  012602                          MOV     (SP)+,R2
     73 006164  012601                          MOV     (SP)+,R1
     74                                ;
     75                                ;  Now remove any queue elements that belong to the job being aborted
     76                                ;  (But ignore the current queue element since it will be cleaned up
     77                                ;   by calling IOFIN)
     78                                ;
     79 006166  011305                 2$:      MOV     (R3),R5         ;Get pointer to current queue element
     80 006170  001440                          BEQ     6$              ;Br if there is none
     81 006172  016504  000000C        3$:      MOV     Q.LINK-Q.BLKN(R5),R4 ;Is there another queue element in list?
     82 006176  001435                          BEQ     6$              ;Br if not
     83 006200  116400  000000C              MOVB    Q.JOB-Q.BLKN(R4),R0 ;Get job # from queue element
     84 006204  006300                          ASL     R0              ;Convert to job index number
     85 006206  020001                          CMP     R0,R1           ;Does this element belong to job being aborted
     86 006210  001402                          BEQ     4$              ;Br if yes
     87 006212  010405                          MOV     R4,R5           ;Link to next queue element
     88 006214  000766                          BR      3$              ;And go check it
     89 006216  016465  000000C 000000C 4$:      MOV     Q.LINK-Q.BLKN(R4),Q.LINK-Q.BLKN(R5) ;Remove element from list
     90 006224  001005                          BNE     5$              ;Br if this is not last element in list
     91 006226  011363  177776              MOV     (R3),-2(R3)     ;CQE-->LQE
     92 006232  001402                          BEQ     5$              ;Br if queue is empty now
     93 006234  010563  177776              MOV     R5,-2(R3)       ;Make LQE point to last queue element
     94 006240  162704  0000000            5$:      SUB     #Q.BLKN,R4      ;Make R4 point to start of queue element
     95 006244  016400  0000000                MOV     Q.CSW(R4),R0    ;Get address of channel block for this element
     96 006250  001403                          BEQ     9$              ;Br if no associated channel
     97 006252  052760  0000000 0000000          BIS     #CS$ERR,C.CSW(R0);Set error flag in channel status
     98 006260  004737  003032'           9$:      CALL    IOCMPL          ;Do I/O completion for this queue element
     99 006264  005362  0000000                DEC     HANIOC(R2)      ;Decrement pending I/O count for device
    100 006270  000740                          BR      3$              ;See if there are more queue elements
    101                               ;
    102                               ;  If handler finished I/O operation while being held, we must call
    103                               ;  cleanup routine for it.
    104                               ;
    105 006272  006363  177774            6$:      ASL     -4(R3)          ;Did handler finish i/o while being held?
    106 006276  100005                          BPL     10$             ;Br if not
    107 006300  010304                          MOV     R3,R4           ;Set up pointer to CQE cell in handler
    108 006302  005063  177774              CLR     -4(R3)          ;Clear handler-hold flag
    109 006306  004737  002570'              CALL    IOFIN           ;Do cleanup for handler
    110                               ;
    111                               ;  Now exit from system state.
    112                               ;  This will transfer control to 7$.
    113                               ;
    114 006312  000207                    10$:     RETURN                  ;Exit from system state -- Go to 7$
```

```
                                        ;
                                        ;  Finished aborting I/O for this device
                                        ;
006314  012637  0000000         7$:     MOV     (SP)+,@#KPAR5   ;Restore PAR 5 mapping
006320  012605                          MOV     (SP)+,R5
006322  012604                          MOV     (SP)+,R4
006324  012603                          MOV     (SP)+,R3
006326  000207                          RETURN
```

```
                                        .SBTTL  IOTIMR -- Process handler timeout requests
                                ;-------------------------------------------------------------------
                                ;  IOTIMR is the routine called from device handlers to process I/O
                                ;  timer requests.  Normally the .TIMIO and .CTIMIO macros are used
                                ;  to call IOTIMR.
                                ;
                                ;  Inputs:
                                ;   R5 = Address of argument list generated by .TIMIO or .  CTIMIO macros
                                ;
006330  010046                 IOTIMR: MOV     R0,-(SP)
006332  010146                         MOV     R1,-(SP)
006334  010246                         MOV     R2,-(SP)
006336  010346                         MOV     R3,-(SP)
                                ;
                                ;  Determine if we are starting or canceling a timer request
                                ;
006340  010502                         MOV     R5,R2           ;GET PTR TO CELL WITH OFFSET TO TIMER BLOCK
006342  062502                         ADD     (R5)+,R2        ;GET ADDRESS OF TIMER BLOCK IN HANDLER
006344  005725                         TST     (R5)+           ;IS THIS A .TIMIO OR .CTIMIO REQUEST?
006346  001062                         BNE     IOTIMC          ;BR IF REQUEST TO CANCEL TIMER
                                ;
                                ;  This is a request to start a timer.
                                ;  Get a completion request queue entry and set up values in it.
                                ;
006350  005762  000000G                TST     IT$RTN(R2)      ;Make sure there is a specified compl routine
006354  001003                         BNE     4$              ;Br if compl routine address specified
006356  022525                         CMP     (R5)+,(R5)+     ;Skip over high & low order time words
006360  000261                         SEC                     ;Signal error on return
006362  000514                         BR      IOTMXT          ;Return
006364  004737  004414'          4$:    CALL    GETRTQ          ;Get a queue element (addr returned in R1)
006370  010162  000000G                MOV     R1,IT$LNK(R2)   ;Save ptr to queue element in handler block
006374  152761  000000C 000000G        BISB    #<QF$IOT!QF$SCR>,CQ$FLG(R1);Set flag saying .TIMIO request
006402  012561  000000G                MOV     (R5)+,CQ$HOT(R1);Set high-order time value
006406  012561  000000G                MOV     (R5)+,CQ$LOT(R1);Set low-order time value
006412  016261  000000G 000000G        MOV     IT$RTN(R2),CQ$RTN(R1);Set address of routine to call
006420  016261  000000G 000000G        MOV     IT$SEQ(R2),CQ$R0(R1);Set sequence # to be passed in R0
006426  010261  000000G                MOV     R2,CQ$R1(R1)    ;Set address of cell to be cleared on call
006432  062761  000000G 000000G        ADD     #IT$RTN,CQ$R1(R1)
006440  013761  000000G 000000G        MOV     @#KPAR5,CQ$PA5(R1);Set PAR 5 mapping for compl routine
006446  016200  000000G                MOV     IT$JOB(R2),R0   ;Get # of job to synch with
006452  042700  177400                 BIC     #^C377,R0       ;Clear all but job number
006456  006300                         ASL     R0              ;Convert to job index number
006460  110061  000000G                MOVB    R0,CQ$JOB(R1)   ;Set job number in compl queue entry
                                ;
                                ;  Link timer element into list of active requests
                                ;
006464                                  DISABL                  ;** DISABLE **
006472  013761  000000G 000000G        MOV     MRKTHD,CQ$LNK(R1);Put new element at head of timer list
006500  010137  000000G                MOV     R1,MRKTHD
006504                                  ENABL                   ;** ENABLE **
006512  000437                         BR      IOTMCC          ;FINISHED
                                ;
                                ;  Cancel a timer request
                                ;
006514  016201  000000G         IOTIMC: MOV     IT$LNK(R2),R1   ;Get address of queue element used for request
006520  001416                         BEQ     3$              ;Invalid if zero
006522                                  DISABL                  ;** DISABLE **
```

IOTIMR -- Process handler timeout requests

```
     58 006530  012703  000000C                    MOV     #MRKTHD-CQ$LNK,R3;FAKE POINTER TO QUEUE HEAD
     59 006534  026301  0000000          1$:       CMP     CQ$LNK(R3),R1    ;IS ELEMENT WE WANT TO REMOVE NEXT IN LIST?
     60 006540  001410                             BEQ     2$               ;BR IF YES
     61 006542  016303  0000000                    MOV     CQ$LNK(R3),R3    ;CHAIN FORWARD TO NEXT ELEMENT
     62 006546  001372                             BNE     1$               ;BR IF MORE TO CHECK
     63 006550                                     ENABL                    ;** ENABLE **
     64 006556  000261                   3$:       SEC                      ;SIGNAL FAILURE TO FIND TIMER ELEMENT IN LIST
     65 006560  000415                             BR      IOTMXT
     66 006562  016163  0000000 0000000  2$:       MOV     CQ$LNK(R1),CQ$LNK(R3);REMOVE TIMER ELEMENT FROM LIST
     67 006570  005062  0000000                    CLR     IT$RTN(R2)       ;SAY TIMER BLOCK IS NOW FREE
     68 006574  005062  0000000                    CLR     IT$LNK(R2)
     69 006600                                     ENABL                    ;** ENABLE **
     70 006606  004737  004222'                    CALL    QFREE            ;Free the timer queue element
     71 006612  000241                   IOTMCC:   CLC                      ;Return with C-flag cleared
     72                                  ;
     73                                  ;  Finished -- Return to handler
     74                                  ;
     75 006614  012603                   IOTMXT:   MOV     (SP)+,R3
     76 006616  012602                             MOV     (SP)+,R2
     77 006620  012601                             MOV     (SP)+,R1
     78 006622  012600                             MOV     (SP)+,R0
     79 006624  000205                             RTS     R5
```

```
                                        .SBTTL  CANIOT -- Cancel all .TIMIO requests for a job
                                ;-----------------------------------------------------------------------
                                ;  CANIOT is called to cancel all pending .TIMIO requests for a job.
                                ;
                                ;  Inputs:
                                ;   R1 = Index number of job.
                                ;
006626  010246                  CANIOT: MOV     R2,-(SP)
006630  010446                          MOV     R4,-(SP)
006632  013746  000000G                 MOV     @#KPAR5,-(SP)   ;Save current PAR 5 mapping
                                ;
                                ;  Search through pending queue looking for requests belonging to this job
                                ;
006636  012704  000000C         1$:     MOV     #MRKTHD-CQ$LNK,R4 ;Get dummy pointer to queue head
006642                                  DISABL                  ;** Disable interrupts **
006650  016402  000000G         2$:     MOV     CQ$LNK(R4),R2   ;;;Get address of next pending entry
006654  001434                          BEQ     4$              ;;;Br if hit end of list
006656  132762  000000G 000000G         BITB    #QF$IOT,CQ$FLG(R2);;;Is this entry for a .TIMIO?
006664  001403                          BEQ     3$              ;;;Br if not
006666  120162  000000G                 CMPB    R1,CQ$JOB(R2)   ;;;Is this entry for our job?
006672  001402                          BEQ     5$              ;;;Br if yes
006674  010204                  3$:     MOV     R2,R4           ;;;Link to next entry
006676  000764                          BR      2$              ;;;Go check it
                                ;
                                ;  We found an entry for this job
                                ;
006700  016264  000000G 000000G 5$:     MOV     CQ$LNK(R2),CQ$LNK(R4);;;Remove entry from linked list
006706                                  ENABL                   ;** Enable interrupts **
006714  016200  000000G                 MOV     CQ$R1(R2),R0    ;Get addr of cell in handler to be cleared
006720  001404                          BEQ     6$              ;Br if nothing to clear
006722  016237  000000G 000000G         MOV     CQ$PA5(R2),@#KPAR5;Set up PAR 5 mapping for the handler
006730  005010                          CLR     (R0)            ;Clear the cell in the handler
006732  010104                  6$:     MOV     R1,R4           ;Save the job index number
006734  010201                          MOV     R2,R1           ;Get address of queue element to R1
006736  004737  004222'                 CALL    QFREE           ;Free the queue element
006742  010401                          MOV     R4,R1           ;Get back job index number
006744  000734                          BR      1$              ;See if there are more entries to free
                                ;
                                ;  Finished
                                ;
006746                          4$:     ENABL                   ;** Enable interrupts **
006754  012637  000000G         9$:     MOV     (SP)+,@#KPAR5   ;Restore PAR 5 mapping
006760  012604                          MOV     (SP)+,R4
006762  012602                          MOV     (SP)+,R2
006764  000207                          RETURN

                                ;-----------------------------------------------------------------------
                                ;  Dummy subroutine called by handlers trying to do error logging.
                                ;  Return immediately.
                                ;
006766  000207                  ERRLOG: RETURN
```

```
                                        .SBTTL  SETHAN -- Update running copy of device handler
                                ;-------------------------------------------------------------------------
                                ;  SETHAN is the EMT used by kmon to update the running copy of a device
                                ;  handler after a SET command has been done.
                                ;  If the handler is active, an error return occurs.
                                ;
006770  013704  000004@  SETHAN: MOV     EMTBLK+4,R4     ;GET DEVICE TABLE INDEX FOR HANDLER
006774  016402  000000@          MOV     HANSIZ(R4),R2   ;GET SIZE OF HANDLER
007000  162702  000022          SUB     #18.,R2         ;DON'T MODIFY LAST 9 WORDS OF HANDLER (.DREND TABLE)
007004  020227  001000          CMP     R2,#1000        ;MAX CODE MODIFIED IS 1000
007010  101402                  BLOS    1$              ;BR IF HANDLER IS SMALLER
007012  012702  001000          MOV     #1000,R2
007016  006202          1$:     ASR     R2              ;CONVERT # BYTES TO # WORDS
007020  016401  000000@          MOV     HANENT(R4),R1   ;GET ADDRESS OF HANDLER ENTRY POINT
007024  016400  000000@          MOV     HANPAR(R4),R0   ;IS THIS A MAPPED HANDLER?
007030  001402                  BEQ     4$              ;BR IF NOT
007032  010037  000000@          MOV     R0,@#KPAR5      ;MAP PAR 5 TO THE HANDLER
007036  013703  000006@  4$:     MOV     EMTBLK+6,R3     ;GET ADDRESS OF BUFFER IN KMON WITH NEW CODE
007042                          DISABL                  ;*** DISABLE ***
007050  005711                  TST     (R1)            ;IS THE HANDLER ACTIVE NOW?
007052  001012                  BNE     3$              ;BR IF YES -- GIVE ERROR RETURN
007054  162701  000006          SUB     #6,R1           ;POINT TO START OF HANDLER IN MEMORY
007060  106523          2$:     MFPD    (R3)+           ;GET WORD FROM KMON BUFFER
007062  012621                  MOV     (SP)+,(R1)+     ;MOVE OVER OLD HANDLER CODE
007064  077203                  SOB     R2,2$           ;MOVE ALL OF NEW CODE
007066                          ENABL                   ;** ENABLE **
007074  000137  001110'          JMP     EMTXIT          ;SUCCESSFUL EXIT
                                ;
                                ;  Error -- Handler is active now.
                                ;
007100                  3$:     ENABL                   ;** ENABLE **
007106  005000                  CLR     R0              ;RETURN ERROR CODE 0
007110  000137  004036'          JMP     SETERR          ;ERROR EXIT
```

```
                                        .SBTTL  MIOMWT -- Mapped I/O move data to system buffer
                                ;-------------------------------------------------------------------
                                ;  MIOMWT is called when doing a write to a device that requires its I/O
                                ;  to be mapped through a system buffer.
                                ;  MIOMWT moves the data from the user's buffer into a system buffer.
                                ;
                                ;  Inputs:
                                ;   R5 = Address of mapped I/O control block.
                                ;
007114  010246                  MIOMWT: MOV     R2,-(SP)
007116  010446                          MOV     R4,-(SP)
007120  010546                          MOV     R5,-(SP)
007122  010502                          MOV     R5,R2           ;Get address of mapped I/O control block
                                ;
                                ;  Enter system state so that we can use PAR5 and PAR6 to map to the system
                                ;  and user buffers.
                                ;
007124  012700  007212'                 MOV     #1$,R0          ;Go to 1$ on exit from system state
007130  004737  0000000G                CALL    ENSYS           ;Enter system state
007134  0000000G                        .WORD   FP$MOV          ;Fork priority
                                ;
                                ;  We are now running in system state on the interrupt stack.
                                ;  Kernel PAR5 and PAR6 are free.
                                ;  Set up buffer pointers.
                                ;
007136  016237  0000000G 0000000G       MOV     MI$UBP(R2),@#KPAR5 ;Set up PAR base for user's buffer
007144  016237  0000000G 0000000G       MOV     MI$SBP(R2),@#KPAR6 ;Set up PAR base for system buffer
007152  012704  0000000G               MOV     #VPAR5,R4       ;Get virtual address base for user's buffer
007156  066204  0000000G               ADD     MI$UBO(R2),R4   ;Add offset within 64-byte block region
007162  012705  0000000G               MOV     #VPAR6,R5       ;Get virtual address base for system buffer
                                ;
                                ;  Get number of words to move and do the transfer
                                ;
007166  016200  0000000G               MOV     MI$CWC(R2),R0   ;Get number of words to move
007172  006200                         ASR     R0              ;Get number of doublewords
007174  001403                         BEQ     3$              ;Br if less than 2 words to move
007176  012425                  2$:    MOV     (R4)+,(R5)+     ;Move a word
007200  012425                         MOV     (R4)+,(R5)+     ;Move a second word
007202  077003                         SOB     R0,2$           ;Loop if more doublewords to move
007204  103001                  3$:    BCC     4$              ;Br if no odd word at end to move
007206  011415                         MOV     (R4),(R5)       ;Move the last word
                                ;
                                ;  Finished moving data.
                                ;  Exit from system state (go to 1$)
                                ;
007210  000207                  4$:    RETURN                  ;Exit from system state
                                ;
                                ;  Finished
                                ;
007212  012605                  1$:    MOV     (SP)+,R5
007214  012604                         MOV     (SP)+,R4
007216  012602                         MOV     (SP)+,R2
007220  000207                         RETURN
```

```
                                        .SBTTL  MIOMRD -- Mapped I/O move data to user's buffer
                                ;-----------------------------------------------------------------------
                                ;  MIOMRD is called when doing a read from a device that requires its I/O
                                ;  to be mapped through a system buffer.
                                ;  MIOMRD moves the data from the system buffer into the user's buffer.
                                ;
                                ;  Inputs:
                                ;   R5 = Address of mapped I/O control block.
                                ;
007222  010246                  MIOMRD: MOV     R2,-(SP)
007224  010446                          MOV     R4,-(SP)
007226  010546                          MOV     R5,-(SP)
007230  010502                          MOV     R5,R2           ;Get address of mapped I/O control block
                                ;
                                ;  Enter system state so that we can use PAR5 and PAR6 to map to the system
                                ;  and user buffers.
                                ;
007232  012700  007320'                 MOV     #1$,R0          ;Go to 1$ on exit from system state
007236  004737  000000G                 CALL    ENSYS           ;Enter system state
007242  000000G                         .WORD   FP$MOV          ;Fork priority
                                ;
                                ;  We are now running in system state on the interrupt stack.
                                ;  Kernel PAR5 and PAR6 are free.
                                ;  Set up buffer pointers.
                                ;
007244  016237  000000G 000000G         MOV     MI$UBP(R2),@#KPAR5 ;Set up PAR base for user's buffer
007252  016237  000000G 000000G         MOV     MI$SBP(R2),@#KPAR6 ;Set up PAR base for system buffer
007260  012704  000000G                 MOV     #VPAR5,R4       ;Get virtual address base for user's buffer
007264  066204  000000G                 ADD     MI$UBO(R2),R4   ;Add offset within 64-byte block region
007270  012705  000000G                 MOV     #VPAR6,R5       ;Get virtual address base for system buffer
                                ;
                                ;  Get number of words to move and do the transfer
                                ;
007274  016200  000000G                 MOV     MI$CWC(R2),R0   ;Get number of words to move
007300  006200                          ASR     R0              ;Get number of doublewords
007302  001403                          BEQ     3$              ;Br if less than 2 words to move
007304  012524                  2$:     MOV     (R5)+,(R4)+     ;Move a word
007306  012524                          MOV     (R5)+,(R4)+     ;Move a second word
007310  077003                          SOB     R0,2$           ;Loop if more doublewords to move
007312  103001                  3$:     BCC     4$              ;Br if no odd word at end to move
007314  011514                          MOV     (R5),(R4)       ;Move the last word
                                ;
                                ;  Finished moving data.
                                ;  Exit from system state (go to 1$)
                                ;
007316  000207                  4$:     RETURN                  ;Exit from system state
                                ;
                                ;  Finished
                                ;
007320  012605                  1$:     MOV     (SP)+,R5
007322  012604                          MOV     (SP)+,R4
007324  012602                          MOV     (SP)+,R2
007326  000207                          RETURN
```

```
                                        .SBTTL  HANXIT -- Exit from a mapped handler
                                ;-------------------------------------------------------------------------------
                                ;  HANXIT is jumped to when a mapped handler performs a RTI to exit
                                ;  from an interrupt.
                                ;
007330                          HANXIT:
                                ;
                                ;  Restore PAR 5 mapping
                                ;
007330  012637  0000000               MOV     (SP)+,@#KPAR5   ;Restore PAR 5 mapping
                                ;
                                ;  Finished
                                ;
007334  000002                        RTI                     ;Do the real interrupt exit
```

```
                                        .SBTTL  - Device handler mapping routines -
                                ;-------------------------------------------------------------------
                                ;  The following routines provide XM support for device handlers.
                                ;  Kernel-mode PAR 6 is used to access the user's data area.
                                ;
                                        .SBTTL  MPPHY  -- Convert mapped address to physical address
                                ;-------------------------------------------------------------------
                                ;  Convert a mapped address to a physical address.
                                ;
                                ;  Inputs:
                                ;   R5 = Address of Q.BUFF cell in I/O queue entry.
                                ;   Q.BUFF = Address relative to Q.PAR
                                ;   Q.PAR  = Address offset.
                                ;   Q.UMRX = 0==>No Unibus mapping needed; non-zero==>Do unibus mapping
                                ;   Q.UMPB = Original value of Q.BUFF when I/O was queued
                                ;   Q.UMPP = Original value of Q.PAR when I/O was queued
                                ;   Q.UMVB = Index number of base Unibus map register for transfer
                                ;
                                ;  Outputs:
                                ;   (SP) = Low-order 16-bits of physical address.
                                ;   2(SP)= High-order 6-bits of physical address (in bits 4-9).
                                ;   R5   = Address of Q.WCNT in queue element.
                                ;
007336  162706  000004          MPPHY:  SUB     #4,SP           ;MAKE ROOM ON STACK FOR RESULT WORDS
007342  016616  000004                  MOV     4(SP),(SP)      ;MOVE UP RETURN ADDRESS ON STACK
007346  010246                          MOV     R2,-(SP)
007350  010346                          MOV     R3,-(SP)
007352  012546                          MOV     (R5)+,-(SP)     ;GET VIRTUAL ADDRESS FROM Q.BUFF
007354  005002                          CLR     R2              ;WE WILL FORM PHYSICAL ADDRESS IN R2-R3
                                ;
                                ;  Convert virtual address in I/O queue element into physical
                                ;  address and place in R2-R3.
                                ;
007356  005765  000000C                 TST     Q.UMRX-Q.WCNT(R5);IS UNIBUS MAPPING NEEDED?
007362  001416                          BEQ     1$              ;BR IF NOT
                                ;
                                ;  We must do Unibus mapping for this I/O request.
                                ;  What we compute is not a physical address but is rather a
                                ;  virtual address within the Unibus address space.
                                ;
007364  166516  000000C                 SUB     Q.UMPB-Q.WCNT(R5),(SP) ;Subtract original Q.BUFF value
007370  016503  000000C                 MOV     Q.PAR-Q.WCNT(R5),R3  ;Get current PAR offset (64-byte units)
007374  166503  000000C                 SUB     Q.UMPP-Q.WCNT(R5),R3 ;Get # 64-byte blocks Q.PAR has been incr
007400  072327  000006                  ASH     #6.,R3          ;Convert to byte offset
007404  060316                          ADD     R3,(SP)         ;Increment buffer address
007406  116503  000000C                 MOVB    Q.UMVB-Q.WCNT(R5),R3 ;Get unibus register #
007412  073227  000015                  ASHC    #13.,R2         ;* 8192.  bytes per register
007416  000406                          BR      2$
                                ;
                                ;  We are not doing Unibus mapping.  Get 22-bit physical address to R2-R3.
                                ;
007420  162716  0000000          1$:    SUB     #VPAR6,(SP)     ;REMOVE PAGE-6 BIAS
007424  016503  000000C                 MOV     Q.PAR-Q.WCNT(R5),R3 ;GET RELOCATION BASE
007430  073227  000006                  ASHC    #6,R2           ;SHIFT R2-R3 LEFT 6 BITS
007434  062603                  2$:     ADD     (SP)+,R3        ;ADD (Q.BUFF) TO LOW ORDER
007436  005502                          ADC     R2              ;PROPOGATE CARRY TO HIGH-ORDER
                                ;
```

```
                                        ;  We have computed the address and it is in R2-R3.
                                        ;  Position correctly and store into stack.
                                        ;
007440  072227  000004                        ASH     #4,R2           ;POSITION HIGH-ORDER TO BITS 4-9
007444  010366  000006                        MOV     R3,6(SP)        ;STORE LOW-ORDER VALUE
007450  010266  000010                        MOV     R2,10(SP)       ;STORE HIGH-ORDER VALUE
007454  012603                              MOV     (SP)+,R3
007456  012602                              MOV     (SP)+,R2
                                        ;
                                        ;  Finished -- Converted values are on stack below return address.
                                        ;
007460  000207                              RETURN
```

```
                                        .SBTTL  PTBYT  -- Move byte to user's buffer
                                ;-------------------------------------------------------------
                                ;  PTBYT is called to transfer a byte to the user's buffer area.
                                ;
                                ;  Inputs:
                                ;   (SP) = Byte to be transfered.
                                ;   R4   = Pointer to Q.BLKN in queue element
                                ;
                                ;  Outputs:
                                ;   Byte is popped from stack.
                                ;   Q.BUFF is incremented.
                                ;
007462                          PTBYT:
                                ;
                                ;  Make R4 point to Q.BUFF rather than Q.BLKN
                                ;
007462  062704  000000C                 ADD     #<Q.BUFF-Q.BLKN>,R4;MAKE R4 POINT TO Q.BUFF
                                ;
                                ;  This request must be mapped through PAR 6.
                                ;
                                ;
                                ;  See if we cross a page boundary.
                                ;
007466  032714  020000                  BIT     #20000,(R4)     ;DID WE CROSS A PAGE BOUNDARY?
007472  001405                          BEQ     1$              ;BR IF NOT
                                ;
                                ;  We crossed a page boundary.
                                ;  Increment Q.PAR and reset Q.BUFF
                                ;
007474  062764  000200  000000C         ADD     #200,Q.PAR-Q.BUFF(R4);ADVANCE PAGE NUMBER
007502  162714  020000                  SUB     #20000,(R4)     ;RESET BUFFER POINTER
                                ;
                                ;  Set up Kernel-mode PAR6 to access user's page.
                                ;
007506                          1$:     DISABL                  ;DISABLE INTERRUPTS
007514  013727  0000000                 MOV     @#KPAR6,(PC)+   ;SAVE CURRENT KPAR6 CONTENTS
007520  000000                  11$:    .WORD   0               ;CONTENTS OF KPAR6
007522  016437  000000C 0000000         MOV     Q.PAR-Q.BUFF(R4),@#KPAR6;MAP TO USER'S PAGE
                                ;
                                ;  Move byte into the user's area
                                ;
007530  116634  000002                  MOVB    2(SP),@(R4)+    ;MOVE BYTE INTO USER'S SPACE
007534  013737  007520' 0000000         MOV     11$,@#KPAR6     ;RESTORE KPAR6 CONTENTS
007542                                  ENABL                   ;ENABLE INTERRUPTS
                                ;
                                ;  Increment buffer pointer
                                ;
007550  005244                          INC     -(R4)           ;INC Q.BUFF
                                ;
                                ;  Fix up R4 and pop byte off stack.
                                ;
007552  162704  000000C                 SUB     #<Q.BUFF-Q.BLKN>,R4;MAKE R4 POINT TO Q.BLKN
007556  012616                          MOV     (SP)+,(SP)      ;MOVE RETURN ADDRESS OVER DATA BYTE
007560  000207                          RETURN
```

```
                                        .SBTTL  GTBYT  -- Get byte from user's buffer
                                ;-------------------------------------------------------------
                                ;  GTBYT is called to acquire a byte from a user buffer.
                                ;
                                ;  Inputs:
                                ;   R4 = Pointer to Q.BLKN in I/O queue element.
                                ;
                                ;  Outputs:
                                ;   (SP) = Byte acquired.
                                ;   Q.BUFF is incremented.
                                ;
007562                          GTBYT:
                                ;
                                ;  Make room for returned byte on stack.
                                ;
007562  011646                          MOV     (SP),-(SP)      ;MOVE DOWN RETURN ADDRESS
                                ;
                                ;  Make R4 point to Q.BUFF
                                ;
007564  062704  000000C                 ADD     #<Q.BUFF-Q.BLKN>,R4;MAKE R4 POINT TO Q.BUFF
                                ;
                                ;  This address needs to be mapped through PAR 6.
                                ;
                                ;
                                ;  See if we crossed a page boundary.
                                ;
007570  031427  020000                  BIT     (R4),#20000     ;DID ADDRESS CROSS A PAGE BOUNDARY?
007574  001405                          BEQ     1$              ;BR IF NOT
                                ;
                                ;  We have crossed a page boundary.
                                ;   Increment Q.PAR and reset Q.BUFF
                                ;
007576  062764  000200  000000C         ADD     #200,Q.PAR-Q.BUFF(R4);INCREMENT PAGE POINTER
007604  162714  020000                  SUB     #20000,(R4)     ;RESET BYTE-WITHIN-PAGE #
                                ;
                                ;  Set up kernel-mode PAR6 to map to user's page
                                ;
007610                          1$:     DISABL                  ;DISABLE INTERRUPTS
007616  013727  0000000                 MOV     @#KPAR6,(PC)+   ;SAVE CURRENT KPAR6 CONTENTS
007622  000000                  11$:    .WORD   0               ;CONTENTS OF KPAR6
007624  016437  000000C 0000000         MOV     Q.PAR-Q.BUFF(R4),@#KPAR6;MAP TO USER'S PAGE
                                ;
                                ;  Move byte from user's buffer
                                ;
007632  005066  000002                  CLR     2(SP)           ;CLEAR BYTE AREA
007636  113466  000002                  MOVB    @(R4)+,2(SP)    ;GET BYTE FROM USER'S AREA
007642  013737  007622' 0000000         MOV     11$,@#KPAR6     ;RESTORE KPAR6 CONTENTS
007650                                  ENABL                   ;ENABLE INTERRUPTS
                                ;
                                ;  Increment buffer pointer
                                ;
007656  005244                          INC     -(R4)           ;INCREMENT Q.BUFF
                                ;
                                ;  Fix R4 and return with data byte on top of stack
                                ;
007660  162704  000000C                 SUB     #<Q.BUFF-Q.BLKN>,R4;MAKE R4 POINT TO Q.BLKN
007664  000207                          RETURN
```

```
                                        .SBTTL  PTWRD  -- Move word to user's buffer
                                ;-------------------------------------------------------------------------
                                ;  PTWRD is called to move a word into a user buffer area.
                                ;
                                ;  Inputs:
                                ;   (SP) = Word to be moved to user's buffer.
                                ;   R4   = Pointer to Q.BLKN in queue element.
                                ;
                                ;  Outputs:
                                ;   Data word is popped from stack.
                                ;   Q.BUFF is incremented.
                                ;
007666                          PTWRD:
                                ;
                                ;  Make R4 point to Q.BUFF cell in queue element.
                                ;
007666  062704  000000C                 ADD     #<Q.BUFF-Q.BLKN>,R4;POINT TO Q.BUFF
                                ;
                                ;  This address must be mapped through PAR 6.
                                ;
                                ;
                                ;  See if we crossed a page boundary
                                ;
007672  031427  020000                  BIT     (R4),#20000     ;DID WE CROSS A PAGE BOUNDARY?
007676  001405                          BEQ     1$              ;BR IF NOT
                                ;
                                ;  We crossed a page boundary
                                ;  Increment Q.PAR and reset Q.BUFF
                                ;
007700  062764  000200  000000C         ADD     #200,Q.PAR-Q.BUFF(R4);ADVANCE PAGE POINTER
007706  162714  020000                  SUB     #20000,(R4)     ;RESET BYTE-WITHIN-PAGE ADDRESS
                                ;
                                ;  Set up kernel-mode PAR6 to map to user's page.
                                ;
007712                          1$:     DISABL                  ;DISABLE INTERRUPTS
007720  013727  0000000         11$:    MOV     @#KPAR6,(PC)+   ;SAVE CURRENT KPAR6 CONTENTS
007724  000000                  11$:    .WORD   0               ;CONTENTS OF KPAR6
007726  016437  000000C 0000000         MOV     Q.PAR-Q.BUFF(R4),@#KPAR6;MAP TO USER'S PAGE
                                ;
                                ;  Move word into user's buffer.
                                ;
007734  016634  000002                  MOV     2(SP),@(R4)+    ;MOVE WORD INTO USER'S BUFFER
007740  013737  007724' 0000000         MOV     11$,@#KPAR6     ;RESTORE KPAR6 CONTENTS
007746                                  ENABL                   ;ENABLE INTERRUPTS
                                ;
                                ;  Advance buffer pointer
                                ;
007754  062744  000002                  ADD     #2,-(R4)        ;ADVANCE Q.BUFF
                                ;
                                ;  Fix R4 and pop data word from stack.
                                ;
007760  162704  000000C                 SUB     #<Q.BUFF-Q.BLKN>,R4;MAKE R4 POINT TO Q.BLKN
007764  012616                          MOV     (SP)+,(SP)      ;POP DATA WORD -- MOVE UP RETURN ADDRESS
007766  000207                          RETURN
```

```
                                        .SBTTL  EXTP1  -- External KPAR mapping routine for drivers
                                ;-------------------------------------------------------------------
                                ;
                                ;  EXTP1 is called from RMON offset 432 by device drivers wishing
                                ;  to map directly to the user's I/O buffer.  It is initiated in the
                                ;  device driver by the following call:
                                ;       JSR     R0,@$P1EXT
                                ;       .WORD   <<instruction length> + 2 (for KPAR6)>
                                ;       <instructions>
                                ;       .WORD   <new KPAR 6 mapping value>
                                ;  The drivers which use this routine are DD and DM.
                                ;

007770                          EXTP1:
007770  013746  0000000                 MOV     @#KPAR6,-(SP)   ;Save kernal par 6 contents
007774  010146                          MOV     R1,-(SP)        ;Save r1
007776  010246                          MOV     R2,-(SP)        ;Save r2
010000  012002                          MOV     (R0)+,R2        ;Get the length of code (# bytes +2)
010002  060200                          ADD     R2,R0           ;Point to the code beyond par value
010004  010001                          MOV     R0,R1           ;Copy return address
010006  005741                          TST     -(R1)           ;Skip KPAR6 mapping value
                                ;
                                ;  Put instructions and a return on the stack.
                                ;
010010  012746                          MOV     (PC)+,-(SP)     ;Save return instruction on stack
010012  000207                          RETURN                  ;Return instruction pushed to stack
010014  006202                          ASR     R2              ;Convert # bytes to # words
010016  014146                  1$:     MOV     -(R1),-(SP)     ;Push instructions and # bytes on stack
010020  077202                          SOB     R2,1$           ;Continue until all saved
                                ;
                                ;  Calculate a stack frame pointer and put it on the stack.
                                ;
010022  012602                          MOV     (SP)+,R2        ;Get length of code (# bytes +2)
010024  060602                          ADD     SP,R2           ;Calculate stack pointer to saved R2
010026  005722                          TST     (R2)+           ;Calculate stack frame pointer to saved R1
010030  010246                          MOV     R2,-(SP)        ;Save stack frame pointer on top of stack
010032  011201                          MOV     (R2),R1         ;Restore r1 prior to JSR
010034  014202                          MOV     -(R2),R2        ;Restore r2 prior to JSR
010036  016037  177776  0000006         MOV     -2(R0),@#KPAR6  ;Initialize KPAR6 mapping
                                ;
                                ;  The stack is in the following configuration:
                                ;       SP -->  stack frame pointer to saved R1
                                ;               instructions
                                ;               RETURN
                                ;               saved R2
                                ;               saved R1 (pointed to by stack frame pointer on top of stack)
                                ;               KPAR6
                                ;               saved R0 (from JSR call)
                                ;
010044  004766  000002                  CALL    2(SP)           ;Call instructions saved on stack
010050  011606                          MOV     @SP,SP          ;Restore stack frame pointer
010052  005726                          TST     (SP)+           ;Skip over r1 value
010054  012637  0000006                 MOV     (SP)+,@#KPAR6   ;Restore kernal par 6 contents
010060  000200                          RTS     R0              ;Return to handler
```

```
                                        .SBTTL  CVTPHY -- Convert a virtual address to a physical addr
                                ;-------------------------------------------------------------------------
                                ;  Convert a 16-bit virtual address within a job region into a full
                                ;  22-bit physical address.
                                ;
                                ;  Inputs:
                                ;   R0 = 16-bit virtual address within job region
                                ;   R5 = Pointer to 3rd word of I/O queue element
                                ;
                                ;  Outputs:
                                ;   R1 = High-order 6 bits of physical address (starting at bit 4)
                                ;   R2 = Low-order 16 bits of physical address
                                ;   C-flag is set on return if physical address is > 56Kb
                                ;
010062  010046                  CVTPHY: MOV     R0,-(SP)
010064  010446                          MOV     R4,-(SP)
010066  010504                          MOV     R5,R4           ;Put I/O queue element pointer in R5
                                ;
                                ;  Call RELOC to convert the virtual address into an address within the
                                ;  PAR6 region and get a base offset for PAR6.
                                ;
010070  004737  010126'                 CALL    RELOC           ;Get user's par mapping value
                                ;
                                ;  At this point:
                                ;    R1 = Mapping value to use to map Par6 to the area.
                                ;    R2 = Address within Par6 region that specified mapped start of area.
                                ;
010074  162702  0000000                 SUB     #VPAR6,R2       ;Strip par6 relocation bias
010100  005000                          CLR     R0              ;Set up for shift
010102  073027  000006                  ASHC    #6,R0           ;Shift R0-R1 left 6 bits
010106  060102                          ADD     R1,R2           ;Get low-order physical address
010110  005500                          ADC     R0              ;Propogate carry to high-order
010112  010001                          MOV     R0,R1           ;Return high-order part in R1
010114  072127  000004                  ASH     #4,R1           ;Align high bits starting at bit 4
                                ;
                                ;  Finished
                                ;
010120  012604                          MOV     (SP)+,R4
010122  012600                          MOV     (SP)+,R0
010124  000207                          RETURN
```

```
                                        .SBTTL  RELOC  -- Mapping relocation calculation
                                ;-------------------------------------------------------------------
                                ;  RELOC is a subroutine which converts a 16-bit virtual address contained
                                ;  in R0 into a PAR6 relocation value returned in R1 and PAR6 offset address
                                ;  returned in R2.
                                ;
                                ;  Inputs:
                                ;   R0 = Virtual address to be converted.
                                ;   R4 = Pointer to 3rd word of Queue element.
                                ;
                                ;  Outputs:
                                ;   R1 = PAR6 mapping relocation value.
                                ;   R2 = PAR6 offset (i.e., address within PAR6 region)
                                ;
010126                          RELOC:
                                ;
                                ;  Save registers.
                                ;
010126  010346                          MOV     R3,-(SP)
010130  116401  000000C                 MOVB    Q.JOB-Q.BLKN(R4),R1     ;Get job number from queue element
010134  006301                          ASL     R1                      ;Convert to job index number
                                ;
                                ;  R0 contains the virtual address.  Separate the par register number
                                ;  from the offset value to obtain the mapped address
                                ;
010136  010003                          MOV     R0,R3                   ;Save conversion address
010140  005002                          CLR     R2                      ;Clear high order bits
010142  071227  020000                  DIV     #20000,R2               ;Separate page and offset pointer
010146  006302                          ASL     R2                      ;Create word offset pointer
                                ;
                                ;  User's mapping in job context is valid - map to job context.
                                ;
010150                                  DISABL                          ;;;Disable interrupts
010156  013727  0000000                 MOV     @#KPAR6,(PC)+           ;;;Save par6 contents
010162  000000                  11$:    .WORD   0                       ;;;Contains par6 contents
010164  016137  0000000 0000000         MOV     LCXPAR(R1),@#KPAR6      ;;;Map through par6
                                ;
                                ;  Obtain par mapping contents from job context region.
                                ;
010172  016201  0000000                 MOV     CUPAR0(R2),R1           ;;;Get the PAR value for the page
010176  013737  010162' 0000000         MOV     11$,@#KPAR6             ;;;Restore par6 original contents
010204                                  ENABL                           ;;;Enable interrupts
                                ;
                                ;  Check for valid offset.
                                ;
010212  005002                          CLR     R2                      ;Clear high order bits
010214  071227  000100                  DIV     #100,R2                 ;Calc lowest 64-byte mapping offset
010220  060201                          ADD     R2,R1                   ;Adjust the PAR value accordingly
010222  010302                          MOV     R3,R2                   ;Move par offset into R2
010224  062702  0000000                 ADD     #VPAR6,R2               ;Adjust to PAR6 virtual address
                                ;
                                ;  Restore registers and return.
                                ;
010230  012603                          MOV     (SP)+,R3
010232  000207                          RETURN
```

```
                                        .SBTTL  BLKMV  -- Memory block move routine
                                ;-------------------------------------------------------------------
                                ;  BLKMV is used by the VM handler to transfer data from one memory area
                                ;  to another.
                                ;
                                ;  Inputs:
                                ;    R1 = input PAR value
                                ;    R2 = input buffer address
                                ;    R3 = output PAR value
                                ;    R4 = output buffer address
                                ;    R5 = word count
                                ;

010234                          BLKMV:
010234  010046                          MOV     R0,-(SP)                ;Save registers
010236  013746  0000000                 MOV     @#KPAR5,-(SP)           ;Save kernel PAR 5 & 6 mapping
010242  013746  0000000                 MOV     @#KPAR6,-(SP)
010246  042702  160000                  BIC     #160000,R2              ;Strip input buffer bias
010252  052702  120000                  BIS     #120000,R2              ;Convert input buffer to PAR5 base
010256  042704  160000                  BIC     #160000,R4              ;Strip output buffer bias
010262  052704  140000                  BIS     #140000,R4              ;Convert output buffer to PAR6 base
                                ;
                                ;  Convert base buffer offsets and PAR to allow block move.
                                ;
010266  020227  130000          1$:     CMP     R2,#130000              ;Check input buffer address
010272  103404                          BLO     2$                      ;Br if below boundary
010274  162702  010000                  SUB     #10000,R2               ;Subtract 4K byte offset
010300  062701  000100                  ADD     #100,R1                 ;Adjust PAR value
010304  020427  150000          2$:     CMP     R4,#150000              ;Check input buffer address
010310  103404                          BLO     3$                      ;Br if below boundary
010312  162704  010000                  SUB     #10000,R4               ;Subtract 4K byte offset
010316  062703  000100                  ADD     #100,R3                 ;Adjust PAR value
                                ;
                                ;  Move up to one block at a time.
                                ;
010322  012700  000400          3$:     MOV     #256.,R0                ;Move 256. words
010326  020005                          CMP     R0,R5                   ;Check with actual word count
010330  103401                          BLO     5$                      ;Br if word count > 256.
010332  010500                          MOV     R5,R0                   ;Otherwise use actual word count
                                ;
                                ;  Initialize the PAR registers and perform the move.
                                ;
010334  160005                  5$:     SUB     R0,R5                   ;Adjust the number of words moved
010336  010137  0000000                 MOV     R1,@#KPAR5              ;Set PAR5 value
010342  010337  0000000                 MOV     R3,@#KPAR6              ;Set PAR6 value
010346  005200                          INC     R0                      ;Adjust word count for double move
010350  006200                          ASR     R0                      ;Divide for double word move
010352  103001                          BCC     20$                     ;Br if single word move necessary
010354  012224                  10$:    MOV     (R2)+,(R4)+             ;Move input buffer to output buffer
010356  012224                  20$:    MOV     (R2)+,(R4)+
010360  077003                          SOB     R0,10$                  ;Continue until all has been moved
010362  005705                          TST     R5                      ;Check for remaining words
010364  101340                          BHI     1$                      ;Br if more to move
010366  012637  0000000                 MOV     (SP)+,@#KPAR6           ;Restore PAR 5 & 6 mapping
010372  012637  0000000                 MOV     (SP)+,@#KPAR5
010376  012600                          MOV     (SP)+,R0                ;Restore registers
010400  000207                          RETURN
```

```
     58
     59          000001                              .END
Errors detected:  0

*** Assembler statistics


Work  file  reads:  0
Work  file writes:  0
Size of work file:  9624 Words  ( 38 Pages)
Size of core pool:  17920 Words  ( 70 Pages)
Operating  system:  RT-11

Elapsed time:  00:01:07.53
DK:TSEMT,LP:TSEMT=DK:TSEMT.MAC/C/N:SYM
```

Cross reference table (CREF V05.04)

```
$DEAD     1-86      8-13
$GEMAR    1-82      6-29      6-47
$INKMN    1-98      1-108     28-44
$MLOCK    1-84      9-26      9-54      9-60      10-26
$NDMEM    1-88      9-32      10-24     10-31
$NLOCK    1-84      9-34
...V1     8-38      8-38
...V2     8-38      8-38      8-38#     8-38#
ABORT     1-116     4-69      19-21
ABTIO     1-69      29-46#
ARGFIN    5-31      5-49      5-56      6-10      6-60#
BADAD     28-96     28-100    28-105#
BADEMT    1-75      6-74#
BASMAP    1-84      9-20      9-36      10-21
BELL      2-7#      2-32
BLKMV     1-75      44-14#
BOTCSR    1-111     8-41
BOTUNI    1-111     8-40
C.CSW     1-123     16-36*    16-60     22-26     22-30     29-51     29-53     30-58*    30-97*
C.DEVQ    1-125     22-28
C.LENG    1-126
C.NUMQ    1-124     11-27*    16-41*    29-12
C.SBLK    1-126     22-27
C.USED    1-83      16-37     16-38     16-40*
CANIOT    1-73      29-73     32-8#
CFLAG     1-97      4-41      7-30      19-26
CHKABT    1-118     9-53      20-23     29-16
CHNADR    1-109     29-10     29-50
CHNNUM    1-104     18-22
CKUSP2    1-91      4-66
CLWTIM    1-67      2-15#
CORUSR    1-110     4-49      6-28      8-27      18-20     18-29     27-27     28-40     29-28
CP$RT     1-90      16-74
CP$STD    1-86      16-71     17-66     18-28
CQ$CP     1-86      16-71*    16-74*    17-50     17-50     17-64     17-66*    18-28*
CQ$FLG    1-92      16-70*    16-72     31-32*    32-18
CQ$HOT    1-93      31-33*
CQ$JOB    1-88      16-61*    17-13     18-20*    31-43*    32-20
CQ$LNK    1-87      17-46     17-48     17-58*    17-59*    31-48*    31-58     31-59     31-61     31-66     31-66*    32-14
          32-16     32-27     32-27*
CQ$LOT    1-93      31-34*
CQ$PA5    1-87      16-69*    17-26     18-27*    31-39*    32-31
CQ$PRI    1-130     16-68*    17-56     17-56     17-71     18-30*
CQ$RO     1-88      1-93      16-60*    17-19     18-21*    31-36*
CQ$R1     1-88      1-93      16-59*    17-20     18-22*    31-37*    31-38*    32-29
CQ$RNS    1-132     16-65*    16-67*    17-53     17-53     17-77     17-79*    17-83     18-31*    18-34*
CQ$RTN    1-87      17-27     18-26*    31-35*
CR        2-5#      2-32      2-32
CS$EOF    1-99      16-35
CS$ERR    1-99      16-35     30-58     30-97
CS$OPN    1-99      29-51
CSHALC    1-90      12-33
CSHIO     1-96      12-35
CUPARO    1-136     43-40
CUREMT    1-83      4-48*
CVTPHY    1-67      42-15#
```

Cross reference table (CREF V05.04)

```
CXTBAS   1-117    11-23    11-26    16-30    16-33    27-22
CXTEND   1-105    28-46
DIEARG   1-44      7-10*
DIEMSG   1-44      7-10*   22-59*
DOEMT    1-83      6-60
DOPLAS   6-56      6-64#
DOSCHD   1-87     17-87*
DS$ABT   1-94     30-48
DVFLAG   1-112    12-31    12-40    23-29
DVSTAT   1-119    30-48
DVSTOP  29-55     29-79    30-14#
DX$DMA   1-112    23-29
DX$MAP   1-117    12-40
DX$NCA   1-96     12-31
E375MX   1-91      6-8
EM$NQE   1-135    22-59
EM$SOF   1-104     7-10
EMT375   4-75      6-15     6-20#
EMT376   1-91      6-16
EMTABT  19-20#
EMTADR   1-116     4-16     4-17*    4-44     4-46*    4-68     7-69*   19-20
EMTASP   1-109     4-18     4-19*    4-20*    5-38     5-48*    7-61*
EMTBLK   1-96      4-73     5-19     6-52     6-55     8-29    18-23    33-7    33-18
EMTCXN   1-121     7-46
EMTCXT   1-119     4-31
EMTCXW   1-119     4-32     7-47
EMTENT   1-71      4-14#
EMTERR   1-122     4-42*    7-23*    7-28     19-9*
EMTLEV   1-86      4-29*    7-44*
EMTMAP   1-104     4-36*    7-39    18-27
EMTPLS   1-82      6-66
EMTPS    1-118     4-14     4-15*    4-41*    4-50     4-64     5-32    7-15    7-30*   7-65    7-70*   19-26*
        28-31
EMTRAD   1-109     7-76     7-79     7-81*
EMTSP    1-104     4-35*    5-47*    7-43
EMTXIT   1-73      7-8#     9-56     9-61     19-27    29-58    33-27
ENQTL    1-129    17-86
ENSYS    1-120    13-59    30-27    34-19    35-19
ERRLOC   1-97      7-27     7-31*
ERRLOG   1-74     32-51#
ERROK   19-10     19-16    19-25#
EXCJOB   1-140     8-27*
EXTP1    1-75     41-14#
FAKCMP   1-75     18-10#
FILSPC   1-76      2-12#    8-33*    8-34*    8-36     8-37
FMMSG    1-70      2-32#
FORKQ    1-95     14-71
FP$IOA   1-103    30-28
FP$IOF   1-103    14-66
FP$IOS   1-103    13-60
FP$MOV   1-103    34-20    35-20
FQ$PRI   1-95     14-66*
FQ$R1    1-95     14-68*
FQ$R3    1-95     14-69*
FQ$R5    1-95     14-70*
FQ$RTN   1-95     14-67*
```

Cross reference table (CREF V05.04)

```
FREIOQ   1-119    20-30    20-31*   21-11    21-13*   22-49    22-51*
FREUMR   1-67     16-18    24-14#
FRKGET   1-95     14-65
FRKPRI   1-129    14-58
G16STK   3-20#     5-20
GETQ     1-69     18-15    20-10#   20-24
GETRTQ   1-75     22-18    22-48#   31-30
GETSYQ   1-70     22-13#
GETUCH   1-70     25-12#
GETUMR   1-67     12-47    23-25#   24-57
GTBYT    1-74     39-12#
HANENT   1-119    13-27    30-21    30-36    33-14
HANIOC   1-83     13-20*   14-51*   30-99*
HANPAR   1-120    13-24    30-33    33-15
HANSIZ   1-105    33-8
HANXIT   1-77     36-6#
INTERR   1-108     4-43*    7-21     7-23    19-25*
INTPRI   1-89     13-45    13-53    14-29    14-46    17-60    20-32    21-14    22-53    23-69    23-76    24-46
        24-54     29-18    29-38    31-50    31-63    31-69    32-28    32-41    33-26    33-31    38-44    39-48
        40-44     43-42
IOABFL   1-101    29-46    29-66
IOCMPL   1-97     12-26    15-19    16-13#   30-98
IOFCPL  14-60     14-67    15-14#
IOFIN    1-71     14-11#   30-109
IOHALT   1-73     29-66#
IOHANQ   1-67     12-52    13-10#   24-59
IOQSIZ   1-96     20-54
IOSTOP   1-68     29-27#   29-47    29-67
IOSTRT   1-68     11-38    12-15#
IOTIMC  31-20     31-55#
IOTIMR   1-71     31-10#
IOTMCC  31-51     31-71#
IOTMXT  31-29     31-65    31-75#
IOWAIT   1-72     29-9#
IT$JOB   1-93     31-40
IT$LNK   1-130    31-31*   31-55    31-68*
IT$RTN   1-131    31-25    31-35    31-38    31-67*
IT$SEQ   1-97     31-36
JSTKND   1-104     7-8
JSWLOC   1-120     4-53
KPAR5    1-94      4-36     7-39*    9-15*   10-13    10-17*   10-40*   11-12    11-17    11-25*   11-42*   13-15
        13-26*    13-70*   16-16    16-32*   16-86*   17-25    17-26*   17-28*   27-39    30-17    30-35*   30-118*
        31-39     32-10    32-31*   32-42*   33-17*   34-26*   35-26*   36-10*   44-16    44-44*   44-55*
KPAR6    1-94     20-40    34-27*   35-27*   38-36    38-38*   38-43*   39-39    39-41*   39-47*   40-36    40-38*
        40-43*    41-15    41-39*   41-53*   43-34    43-36*   43-41*   44-17    44-45*   44-54*
LBASE    1-127     9-19     9-37    27-28
LCMPL    1-87     17-46
LCXPAR   1-136    43-36
LEMTPC   1-133     4-52*
LF       2-6#      2-32     2-32     2-32
LIOCNT   1-89     11-31*   11-32*   16-26*   29-30
LITIME   1-110    16-63    18-32
LJSW     1-104     4-54*
LMXNUM   1-134     8-15
LOMAP    1-84      9-21     9-35
LPRI     1-89      1-97    16-68    17-72    17-74*   18-30
```

Cross reference table (CREF V05.04)

```
LSTATE    1-122    16-47    17-84
LSTPL     1-128     8-12
LSW       1-85      9-32*    10-24     10-31*
LSW11     1-82      6-29*     6-47*
LSW3      1-101     8-13
LSW4      1-98      1-105    28-44
LSW6      1-103     9-26      9-34*     9-54      9-60*     10-26
MAPPAR    1-84      9-15     10-17
MEMSWP    1-91      9-33*    10-32*
MI$CWC    1-106    34-34     35-34
MI$SBP    1-106    34-27     35-27
MI$UBO    1-106    34-29     35-29
MI$UBP    1-106    34-26     35-26
MIOCHK    1-117    12-42
MIOMRD    1-77     35-10#
MIOMWT    1-77     34-10#
MPPHY     1-74     37-24#
MRKTHD    1-94     31-48     31-49*    31-58     32-14
MXCSR     1-135     8-21*
MXDTR     1-135     8-20*
NMFREQ    1-89     20-14     20-29*    21-15*    22-52*
NUMDEV    1-90     29-77
ODTBAS    1-91
OVRHC     1-140    12-42
PO$LOK    1-91      9-6
PRINT     1-139     4-60
PRIVCO    1-86      1-98      9-6
PS        1-22#     7-17*     8-6*      8-39*     13-36*    13-45*    13-53*    14-26*    14-29*    14-46*    17-44*    17-60*
         20-10*    20-32*    21-10*    21-14*    22-48*    22-53*    23-50*    23-69*    23-76*    24-35*    24-46*    24-54*
         29-11*    29-18*    29-29*    29-38*    31-47*    31-50*    31-57*    31-63*    31-69*    32-15*    32-28*    32-41*
         33-19*    33-26*    33-31*    38-35*    38-44*    39-38*    39-48*    40-35*    40-44*    43-33*    43-42*
PTBYT     1-74     38-13#
PTWRD     1-74     40-13#
PUTUCH    1-70     26-12#
Q.BLKN    1-124    13-32     13-43*    14-31     22-27*    30-52     30-56     30-81     30-83     30-89     30-89*    30-94
         38-17     38-52     39-20     39-56     40-17     40-52     43-20
Q.BUFF    1-126    22-29*    23-85     27-55*    38-17     38-30*    38-38     38-52     39-20     39-33*    39-41     39-56
         40-17     40-30*    40-38     40-52
Q.CHAN    1-125    16-59
Q.COMP    1-125    16-53
Q.CSW     1-126    22-25*    30-56     30-95
Q.DEVX    1-107    11-32     12-20     13-19     14-50     22-32*    24-56
Q.FLAG    1-92     16-70
Q.FUNC    1-124    16-19
Q.ICSW    1-96     16-34     16-37     22-24
Q.JOB     1-124    11-28     16-23     30-52     30-83     43-20
Q.LINK    1-124    13-31*    13-43*    14-32     20-31     21-12*    22-51     23-62     23-63     23-67*    23-68*    24-36
         24-44     24-53     24-53*    30-81     30-89     30-89*
Q.PA5     1-126    11-12*    16-69
Q.PA6     1-127    11-25     16-32     20-40*
Q.PAR     1-126    23-87     27-54*    37-42     37-53     38-30*    38-38     39-33*    39-41     40-30*    40-38
Q.UCSW    1-96     11-21     16-22     22-22*
Q.UMPB    1-114    23-86*    37-41
Q.UMPP    1-115    23-88*    37-43
Q.UMRX    1-113    23-25     23-77*    24-14     24-27     24-29*    37-34
Q.UMVB    1-114    23-89*    37-46
```

Cross reference table (CREF V05.04)

```
Q.UNIT    1-125    22-28*
Q.WCNT    1-123    16-21     23-43     24-38     37-34     37-41     37-42     37-43     37-46     37-53
QCOMPL    1-75     16-75     17-8#     18-39
QF$CIO    1-90     16-72
QF$IOT    1-77     31-32     32-18
QF$MIO    1-90     16-72
QF$SCR    1-93     31-32
QFREE     1-71     16-82     17-38     18-49     21-9#     31-70     32-35
QHIPRI    1-89     16-49     21-25
QIO       1-70     11-16#
QIOFIN   12-27     12-36     12-43     12-48     12-56#
QNSPND    1-118     9-52     20-22     29-15     29-33
QSRCH     1-118    21-23
QWTCNT    2-14#    20-20*    21-20     21-27*
READ      1-78      1-137
REBOOT    1-76      8-6#
RELOC     1-78     42-22     43-15#
RPDR      1-102    28-54
RSR       1-134     8-17*
RT11EX    1-91      6-38      6-52
RTLOCK    1-72      1-74      9-6#
S$HICP    1-110    16-65     18-34
S$IOFN    1-130    16-67     18-31
S$IOWT    1-122    16-47     29-14     29-32
S$NEDQ    1-119    20-21     21-22
S$RT      1-103    17-77     17-79
S$WFM     1-85      9-51
SA$LOK    1-85      9-46
SCPGET    1-85      9-44
SERFLG    1-100    19-15
SETC      1-69     19-26#
SETERR    1-73      6-69      6-75      9-9       18-51     19-9#     28-106    33-33
SETHAN    1-71     33-7#
SP$CMD    1-85      9-46*
SP$JOB    1-85      9-45*
SPFLDV    1-76      2-21#
SPFLNM    1-76      2-22#     8-38
SPLERR    1-70      2-33#
SPSIZE    1-76      2-23#     16-21*
SROMMR    1-140     8-33      8-36*     8-54*
SR3MMR    1-140     8-34      8-37*     8-55*
SSEMT     1-83      6-54
SWPFRC    1-72      9-39     10-11#
SYBFAD    1-68     27-17#
SYQIO     1-68     11-12#
SYSDAT    1-92      8-59
SYSHLT    1-44      7-10     22-59
SYSXIT    1-88      7-85
SYTIMH    1-92      8-60
SYTIML    1-92      8-61
TSEMT     1-21#     1-68
TSINIT    1-83      8-46      8-65
TSR       1-134     8-18*
TTYIN     1-138     4-63
TTYOUT    1-138     4-57
UACHKB    1-69     28-18     28-25#    28-99
```

Cross reference table (CREF V05.04)

```
UACHKW   1-69    18-24    28-17#   28-95
UBUSMP   1-112   23-27
UHIMEM   1-121    6-22    28-25
UIOCNT   1-127   11-33*   16-27*
UM$$SZ   1-113   23-55
UM$IOQ   1-112   23-51    23-75*   24-28*
UM$NMR   1-114   23-101
UM$UMR   1-114   23-89    23-98
UM$WDS   1-113   23-53    24-34
UMODE    1-122    4-50     4-64     5-32     7-15     28-31
UMRADR   1-114   23-100
UMRBAS   1-112   23-49
UMREND   1-113   23-56
UMRWHD   1-113   23-62    24-36
UPMODE   1-110    7-17     7-80
URO      1-118    4-37*    6-68*    7-35
USREMT   1-76     2-11#    8-38
V1EMT    5-5#
V1STK    3-5#     5-30
V2EMT    4-77     6-4#
VALADB   1-98    28-99#
VALADW   1-98     6-24    28-95#
VCSHNB   1-140    8-28*
VPAR5    1-140   11-26    16-33    27-34    34-28    35-28
VPAR6    1-90    22-29    23-93    27-50    34-30    35-30    37-52    42-28    43-50
VPLAS    1-82     6-64
VPRIHI   1-103   17-75
VSWPFL   1-84     9-10
WRITE    1-78     1-137
ZCLR     1-135    8-21
ZEROQ    20-36    20-52#   22-54
```