

Table of contents

4-	1	EMT entry and initial processing
7-	1	EMTXIT -- Exit from EMT processing
8-	1	REBOOT -- Kmon EMT to reboot the system
9-	1	RTLOCK -- EMT to lock a job in low memory
10-	1	SWPFRG -- Force outswap of jobs in certain memory range
11-	1	QIO -- Queue an I/O request
12-	1	IOSTRT -- Try to start I/O transfer
13-	1	IOHANQ -- Place I/O queue request on handler list
14-	1	IOFIN -- I/O operation completion
15-	1	IOFCPL -- IOFIN processing done at fork level
16-	1	IOCMPL -- Do cleanup on completed I/O queue element
17-	1	QCOMPL -- Queue a completion routine request
18-	1	FAKCOMP -- Generate fake completion routine request
19-	1	SETERR -- Set EMT error code and exit
20-	1	GETQ -- Get a free I/O queue element
21-	1	QFREE -- Return an I/O queue element to the free list
22-	1	GETSYQ -- Get queue element for system I/O
22-	39	GETRTQ -- Get queue element for real-time use
23-	1	GETUMR -- Allocate Unibus map register
24-	1	FREUMR -- Free a Unibus map register
25-	1	GETUCH -- Get character from user's buffer
26-	1	PUTUCH -- Move byte to user's buffer
27-	1	SYBFAD -- Set system buffer address in I/O queue entry
28-	1	UACHKx -- Check validity of user address
28-	81	VALADx -- Validate user address
29-	1	IOWAIT -- Wait for I/O to finish
29-	22	IOSTOP -- Wait for all of job's I/O to finish
29-	42	IOHALT -- Abort all I/O for a job
30-	1	DVSTOP -- Abort I/O for a device & job
31-	1	IOTIMR -- Process handler timeout requests
32-	1	CANIOT -- Cancel all .TIMIO requests for a job
33-	1	SETHAN -- Update running copy of device handler
34-	1	MIDMWT -- Mapped I/O move data to system buffer
35-	1	MIDMRD -- Mapped I/O move data to user's buffer
36-	1	HANXIT -- Exit from a mapped handler
37-	1	- Device handler mapping routines -
37-	6	MPPHY -- Convert mapped address to physical address
38-	1	PTBYT -- Move byte to user's buffer
39-	1	GTBYT -- Get byte from user's buffer
40-	1	PTWRD -- Move word to user's buffer
41-	1	EXTP1 -- External KPAR mapping routine for drivers
42-	1	CVTPHY -- Convert a virtual address to a physical addr
43-	1	RELOC -- Mapping relocation calculation
44-	1	BLKMOV -- Memory block move routine

```

1          .TITLE TSEMT T/S EMT PROCESSOR
2          .ENABL LC
3          .ENABL AMA
4          .DSABL QBL
5          ;
6          ; Copyright (C) 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985.
7          ;
8          ; S&H Computer Systems, Inc.
9          ; Nashville, Tennessee
10         ;
11         ; this software is furnished under a license for use only
12         ; on a single computer system and may be copied only with
13         ; the inclusion of the above copyright notice. This
14         ; software, or any other copies thereof, may not be provided
15         ; or otherwise made available to any other person except
16         ; for use on such system and to one who agrees to these
17         ; license terms. title to and ownership of the software
18         ; shall at all times remain with S&H Computer Systems, Inc.
19         ;
20 000000          .CSECT TSEMT
21 000000 TSEMT:
22          PS = 177776 ; PROCESSOR STATUS WORD
23         ;
24         ; System macros.
25         ;
26         .MCALL .READW
27         -----
28         ; MACROS TO ENABLE AND DISABLE INTERRUPTS.
29         ;
30         .MACRO DISABL ;DISABLE INTERRUPTS
31         BIS #340, @#PS
32         .ENDM DISABL
33         ;
34         .MACRO ENABL ;ENABLE INTERRUPTS
35         BIC INTPRI, @#PS
36         .ENDM ENABL
37         ;
38         ; Macro to print an error message when a system crash occurs.
39         ;
40         ; Arguments:
41         ; MSG = Name of error message to print.
42         ; ARG = (Optional) argument value to display with error message.
43         ;
44         .GLOBL DIEMSG, DIEARG, SYSHLT
45         .MACRO DIE MSG, ARG
46         MOV MSG, @#DIEMSG
47         .IF NB, ARG
48         MOV ARG, @#DIEARG
49         .ENDC
50         CALL @#SYSHLT
51         .ENDM DIE
52         ;
53         ; Macro definition for calling global routines residing in
54         ; mapped system regions.
55         ;
56         .MACRO OCALL ENTADD
57         .IF B, ENTADD

```

```

58          . ERROR ;DCALL SPECIFIED WITH NO ENTRY ADDRESS
59          . MEXIT
60          . ENDC
61          CALL     OVRHC          ;CALL THE OVERLAY HANDLER
62          . WORD   ENTADD        ;SPECIFY THE ENTRY POINT
63          . ENDM
64          ;
65          ; Global definitions
66          ;
67          . GLOBL  FREUMR, GETUMR, IOHANQ, CLWTIM, CVTPHY
68          . GLOBL  TSEMT, IDSTOP, SYQIO, SYBFAD, IDSTRT
69          . GLOBL  UACHKB, UACHKW, GETQ, SETC, ABTIO
70          . GLOBL  GETSYQ, QIO, PUTUCH, GETUCH, FMMSG, SPLERR
71          . GLOBL  EMTENT, IOFIN, QFREE, IOTIMR, SETHAN
72          . GLOBL  IOWAIT, RTLOCK, SWPFRC
73          . GLOBL  SETERR, EMTXIT, IOHALT, CANIOT
74          . GLOBL  PTBYT, GTBYT, PTWRD, MPPHY, ERRLOG, RTLOCK
75          . GLOBL  BADEMT, QCOMPL, FAKCMP, GETRTQ, EXTP1, BLKMV
76          . GLOBL  REBOOT, USREMT, FILSPC, SPSIZE, SPFLDV, SPFLNM
77          . GLOBL  MIOMRD, MIOMWT, HANXIT, QF#IOT
78          . GLOBL  READ, WRITE, RELOC
79          ;
80          ; Global references
81          ;
82          . GLOBL  $GEMAR, LSW11, EMTPLS, VPLAS
83          . GLOBL  TSINIT, CUREMT, HANIOC, DOEMT, C. USED, SSEMT
84          . GLOBL  VSWPFL, MAPPAR, BASMAP, LOMAP, $MLOCK, $NLOCK
85          . GLOBL  S$WFM, LSW, SCPGET, SP$JOB, SP$CMD, SA$LOK
86          . GLOBL  $DEAD, EMTLEV, CQ$CP, CP$STD, PRIVCO
87          . GLOBL  CQ$LNK, CQ$RTN, CQ$PA5, DOSCHD, LCMPPL
88          . GLOBL  CQ$JOB, CQ$RO, CQ$R1, SYSXIT, $NDMEM
89          . GLOBL  INTPRI, LIOCNT, QHIPRI, NMFREQ, LPRI
90          . GLOBL  VPAR6, NUMDEV, CSHALC, CP$RT, QF$MID, QF$CID
91          . GLOBL  MEMSWP, PO$LOK, CKUSP2, E375MX, EMT376, RT11EX, ODTBAS
92          . GLOBL  SYTIMH, SYTIML, SYSDAT, CQ$FLG, Q. FLAG
93          . GLOBL  QF$SCR, CQ$HOT, CQ$LOT, CQ$RO, CQ$R1, IT$JOB
94          . GLOBL  KPAR5, KPAR6, DS$ABT, MRKTHD
95          . GLOBL  FRKGET, FORKQ, FQ$PRI, FQ$RTN, FQ$R1, FQ$R5, FQ$R3
96          . GLOBL  EMTBLK, IOQSIZ, CSHIO, DX$NCA, Q. ICSW, Q. UCSW
97          . GLOBL  CFLAG, ERRLOC, IDCMP, LPRI, IT$SEQ
98          . GLOBL  LSW4, $INKMN, VALADW, VALADB, PRIVCO
99          . GLOBL  CS$ERR, CS$EOF, CS$OPN
100         . GLOBL  SERFLG
101         . GLOBL  IOABFL, LSW3
102         . GLOBL  RPDR
103         . GLOBL  LSW6, VPRIHI, S$RT, FP$MOV, FP$IOF, FP$IOS, FP$IOA
104         . GLOBL  EMTMAP, EMTSP, CHNNUM, LJSW, EM$SOF, JSTKND
105         . GLOBL  CXTEND, LSW4, HANSIZ
106         . GLOBL  MI$UBP, MI$SBP, MI$UBD, MI$CWC
107         . GLOBL  Q. DEVX
108         . GLOBL  INTERR, $INKMN
109         . GLOBL  CHNADR, EMTASP, EMTRAD
110         . GLOBL  CORUSR, UPMODE, LITIME, S$HICP
111         . GLOBL  BOTUNI, BOTCSR
112         . GLOBL  UBUSMP, DX$DMA, DVFLAG, UMRBAS, UM$IOQ
113         . GLOBL  UM$WDS, UM$$SZ, UMREND, UMRWHD, Q. UMRX
114         . GLOBL  Q. UMPB, Q. UMYB, UM$UMR, UMRADR, UM$NMR

```

```
115 . GLOBL Q. UMPP
116 . GLOBL ABORT, EMTADR
117 . GLOBL CXTBAS, DX$MAP, MIOCHK
118 . GLOBL URO, EMTPS, QSRCH, CHKABT, QNSPND
119 . GLOBL S$NEDQ, DVSTAT, HANENT, FREIOQ, EMTCXT, EMTCXW
120 . GLOBL JSWLOC, ENSYS, HANPAR
121 . GLOBL EMTCXN, UHIMEM
122 . GLOBL EMTERR, LSTATE, S$IOWT, UMODE
123 . GLOBL C. CSW, Q. WCNT
124 . GLOBL C. NUMQ, Q. JOB, Q. LINK, Q. BLKN, Q. FUNC
125 . GLOBL Q. COMP, Q. CHAN, C. DEVQ, Q. UNIT
126 . GLOBL Q. BUFF, Q. PAR, Q. PA5, C. SBLK, C. LENG, Q. CSW
127 . GLOBL LBASE, UIDCNT, Q. PA6
128 . GLOBL LSTPL
129 . GLOBL ENQTL, FRKPRI
130 . GLOBL S$I OFN, CQ$PRI, IT$LNK
131 . GLOBL IT$RTN
132 . GLOBL CQ$RNS
133 . GLOBL LEMTPC
134 . GLOBL LMXNUM, RSR, TSR
135 . GLOBL MXDTR, ZCLR, MXCSR, EM$NQE
136 . GLOBL LCXPAR, CUPARO
137 . GLOBL READ, WRITE
138 . GLOBL TTYIN, TTYOUT
139 . GLOBL PRINT
140 . GLOBL OVRHC, VPAR5, EXCJOB, SROMMR, SR3MMR, VCSHNB
```

```

1 ;-----
2 ;
3 ; Ascii characters
4 ;
5 000015 CR = 15 ; Carriage return
6 000012 LF = 12 ; Line feed
7 000007 BELL = 07 ; Bell
8 ;
9 ; Shared data areas for mapped system regions.
10 ;
11 000000 USREMT: .BLKW 6 ; Argument area for internal USR emts
12 000014 FILSPC: .BLKW 4 ; Holds dev:file.ext[size] spec for current emt
13 000024 000000 .WORD 0 ; [size] for file specified above
14 000026 000000 QWTCNT: .WORD 0 ; Non-zero ==> Some job waiting for Q element
15 000030 000000 000074 CLWTIM: .WORD 0,60. ; Wait time for CLWAIT routine
16 ;
17 ; Data areas used for special device (mag tape) directory operations.
18 ; Note: SPFLDV, SPFLNM, and SPSIZE must be allocated contiguously for
19 ; magtape directory operations to work correctly.
20 ;
21 000034 SPFLDV: .BLKW 1 ; File spec device name
22 000036 SPFLNM: .BLKW 4 ; File spec involved in dir operation
23 000046 000000 SPSIZE: .WORD 0 ; File size returned by handler
24 .EVEN
25 ;
26 ; Text messages.
27 ;
28 .NLIST BEX
29 ; Note: FMMSG must begin on a word boundary so that XXXXXX is
30 ; word aligned (MOV instructions replace the form name).
31 .EVEN
32 000050 015 012 012 FMMSG: .ASCIZ <CR><LF><LF><BELL>/ Mount 'XXXXXX' form on YYN/<CR><LF>
33 000112 124 123 130 SPLERR: .ASCIZ /TSX-W-Error on write to spool file/
34 .EVEN
35 .LIST BEX

```

```

1
2 ; -----
3 ; TABLE FOR V1 TYPE EMTS WHICH SHOWS HOW MANY
4 ; ARGUMENTS ARE ON THE STACK.
5 ;
6 V1STK: . BYTE 0 ; 000 DELETE
7 . BYTE 0 ; 020 LOOKUP
8 . BYTE 1 ; 040 ENTER
9 . BYTE 0 ; 060
10 . BYTE 0 ; 100 RENAME
11 . BYTE 0 ; 120 SAVE STATUS
12 . BYTE 0 ; 140 REOPEN
13 . BYTE 0 ; 160 CLOSE
14 . BYTE 3 ; 200 READ
15 . BYTE 3 ; 220 WRITE
16 . BYTE 0 ; 240 WAIT

```

```

17 ; -----
18 ; Table for group 16 EMT's showing how many arguments are on the stack.
19 ;
20 @16STK: . BYTE 0 ; 340 TTYIN
21 . BYTE 0 ; 341 TTYOUT
22 . BYTE 1 ; 342 DSTATUS
23 . BYTE 1 ; 343 FETCH/RELEASE
24 . BYTE -3 ; 344 CSIGEN
25 . BYTE -3 ; 345 CSISPC
26 . BYTE 0 ; 346 LOCK
27 . BYTE 0 ; 347 UNLOCK
28 . BYTE 0 ; 350 EXIT
29 . BYTE 0 ; 351 PRINT
30 . BYTE 0 ; 352 SRESET
31 . BYTE 1 ; 353 QSET
32 . BYTE 0 ; 354 SETTOP
33 . BYTE 0 ; 355 RCTRLD
34 . BYTE 0 ; 356 (undefined)
35 . BYTE 0 ; 357 HRESET
36 . EVEN

```

```

1          .SBTTL  EMT entry and initial processing
2          ;-----
3          ; EMTENT is entered directly from an EMT trap.
4          ; On entry, the processor will be in kernel mode and the stack pointer
5          ; will be pointing to an internal job stack.
6          ; Our initial EMT processing consists of the following steps:
7          ; 1. Save registers.
8          ; 2. Save current EMT processing context.
9          ; 3. Determine if this is a version 1 or version 2 format EMT.
10         ; 4. Move arguments for EMT from either users argument area (v2) or
11         ; stack (v1) to internal argument block (EMTBLK).
12         ; 5. Jump off to appropriate EMT processing routine.
13         ;
14 000212 013746 000000G EMTENT: MOV      EMTPS, -(SP)      ;SAVE PREVIOUS EMT PROCESSOR STATUS
15 000216 016637 000004 000000G      MOV      4(SP),EMTPS      ;SAVE PS ON ENTRY FROM TRAP
16 000224 013746 000000G      MOV      EMTADR, -(SP)      ;SAVE PREVIOUS EMT PC
17 000230 016637 000004 000000G      MOV      4(SP),EMTADR      ;SAVE PC OF TRAP
18 000236 013746 000000G      MOV      EMTASP, -(SP)      ;SAVE OLD ARGUMENT STACK POINTER
19 000242 010637 000000G      MOV      SP,EMTASP      ;SET EMTASP TO POINT TO STACK TOP BEFORE EMT
20 000246 062737 000012 000000G      ADD      #12,EMTASP      ;WAS DONE
21 000254 010546          MOV      R5, -(SP)
22 000256 010446          MOV      R4, -(SP)
23 000260 010346          MOV      R3, -(SP)
24 000262 010246          MOV      R2, -(SP)
25 000264 010146          MOV      R1, -(SP)
26         ;
27         ; If an EMT is already being performed, save EMT processing context.
28         ;
29 000266 005237 000000G          INC      EMTLEV      ;INCREMENT EMT PROCESSING LEVEL
30 000272 001406          BEQ      2$          ;BR IF NO EMT IN PROGRESS
31 000274 012705 000000G      MOV      #EMTCXT,R5      ;POINT TO START OF EMT CONTEXT AREA
32 000300 012704 000000G      MOV      #EMTCXW,R4      ;GET # WORDS TO SAVE
33 000304 012546          1$: MOV      (R5)+, -(SP)      ;SAVE EMT CONTEXT ON STACK
34 000306 077402          SOB      R4, 1$
35 000310 010637 000000G      2$: MOV      SP,EMTSP      ;SAVE FRAME POINTER
36 000314 013737 000000G 000000G      MOV      @#KPAR5,EMTMAP  ;Save PAR 5 mapping value on EMT entry
37 000322 010037 000000G      MOV      R0,URO          ;URO HOLDS USER'S R0 VALUE
38         ;
39         ; Initialize EMT context status.
40         ;
41 000326 042737 000000G 000000G      BIC      #CFLAG,EMTPS    ;CLEAR C-FLAG IN PS
42 000334 105037 000000G      CLRB     EMTErr          ;NO I/O ERROR CODE YET
43 000340 105037 000000G      CLRB     INTERR          ;NO INTERNAL ERROR CODE
44 000344 013705 000000G      MOV      EMTADR,R5      ;GET ADDRESS SAVED BY EMT TRAP
45 000350 006545          MFPI     -(R5)          ;GET EMT INSTRUCTION ONTO STACK
46 000352 010537 000000G      MOV      R5,EMTADR      ;SAVE ADDRESS OF EMT INSTRUCTION
47 000356 012604          MOV      (SP)+,R4        ;GET THE EMT INSTRUCTION
48 000360 010437 000000G      MOV      R4,CUREMT      ;SAVE THE EMT INSTRUCTION
49 000364 113701 000000G      MOVVB    CORUSR,R1      ;GET JOB INDEX #
50 000370 032737 000000G 000000G      BIT      #UMODE,EMTPS    ;WAS EMT DONE IN USER MODE?
51 000376 001406          BEQ      5$          ;BR IF NOT
52 000400 010561 000000G      MOV      R5,LEMTPC(R1)   ;SAVE ADDRESS OF LAST USER-MODE EMT
53 000404 106537 000000G      MFPD     @#JSWLOC        ;GET USER'S JSW VALUE
54 000410 012661 000000G      MOV      (SP)+,LJSW(R1)  ;SAVE IN INTERNAL TABLE
55 000414 020427 104341          5$: CMP      R4,#104341    ;IS THIS A TTYOUT EMT?
56 000420 001002          BNE      6$          ;BR IF NOT
57 000422 000137 000000G      JMP      TTYOUT          ;HANDLE TTYOUT SPECIALLY FOR SPEED
    
```

```
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
```

```

1
2 ; Do setup for a version 1 format EMT.
3 ; Convert emt to version 2 format and move arguments to EMTBLK.
4
5 000520 010405 V1EMT: MOV R4,R5 ;GET EMT INSTRUCTION
6 000522 042704 177760 BIC #^C17,R4 ;ISOLATE CHANNEL NUMBER
7 000526 110421 MOV R4,(R1)+ ;STORE INTO EMTBLK
8 000530 042705 177417 BIC #^C360,R5 ;LEAVE ONLY FUNCTION CODE
9 000534 072527 177774 ASH #-4,R5 ;RIGHT JUSTIFY
10 000540 110521 MOV R5,(R1)+ ;STORE FUNCTION CODE INTO EMTBLK
11 000542 010021 MOV R0,(R1)+ ;USERS R0 VALUE GOES AS ARGUMENT 1
12
13 ; Move any arguments from user's stack to EMTBLK.
14
15 000544 106506 MFPD SP ;GET USER'S SP
16 000546 012603 MOV (SP)+,R3 ;INTO R3
17 000550 120527 000016 CMPB R5,#16 ;IS THIS A GROUP 16 EMT?
18 000554 001015 BNE 2$ ;BR IF NOT
19 000556 113702 000000G MOV EMTBLK,R2 ;GET SUB-FUNCTION CODE (CHANNEL #)
20 000562 116202 000171' MOV @16STK(R2),R2 ;GET # ARGUMENTS ON STACK
21 000566 002012 BGE 1$ ;BR IF NOT .CSIxxx
22 ; .CSIxxx & .CSIxxx have a variable number of arguments (3 or 4).
23 ; Determine how many we have this call.
24 000570 005402 NEG R2 ;GET 3 AS # ARGS
25 000572 106563 000004 MFPD 4(R3) ;GET 3RD ARGUMENT ON STACK
26 000576 032726 000001 BIT #1,(SP)+ ;IS THERE A 4TH ARGUMENT?
27 000602 001405 BEQ 3$ ;BR IF NOT
28 000604 005202 INC R2 ;SET 4 AS # ARGUMENTS
29 000606 000403 BR 3$
30 000610 116502 000156' 2$: MOV V1STK(R5),R2 ;GET # ARGUMENTS ON STACK
31 000614 001514 1$: BEQ ARGFIN ;BR IF NO ARGUMENTS ON STACK
32 000616 032737 000000G 000000G 3$: BIT #UMODE,EMTPS ;WAS EMT DONE IN USER OR KERNEL MODE?
33 000624 001017 BNE 4$ ;BR IF IN USER MODE
34 ; EMT was executed in kernel mode.
35 ; This means the arguments are under all the saved information we pushed
36 ; on entry to EMT processing. Pop the arguments and move down the saved
37 ; information on the stack.
38 000626 013703 000000G MOV EMTASP,R3 ;GET POINTER TO ARGUMENTS ON STACK
39 000632 010300 MOV R3,R0
40 000634 012321 6$: MOV (R3)+,(R1)+ ;MOVE AND ARGUMENT FROM STACK TO EMTBLK
41 000636 077202 SOB R2,6$ ;MOVE ALL ARGUMENTS FOR EMT
42 000640 014043 5$: MOV -(R0),-(R3) ;MOVE DOWN OTHER INFO ON STACK OVER ARG AREA
43 000642 020006 CMP R0,SP
44 000644 101375 BHI 5$
45 000646 160003 SUB R0,R3 ;GET # BYTES STACK WAS REDUCED
46 000650 060306 ADD R3,SP ;CORRECT SP
47 000652 060337 000000G ADD R3,EMTSP ;CORRECT FRAME POINTER
48 000656 060337 000000G ADD R3,EMTASP ;CORRECT ARGUMENT POINTER
49 000662 000471 BR ARGFIN
50 ; EMT was done in user mode. This means arguments are on user-mode stack.
51 000664 106523 4$: MFPD (R3)+ ;MOVE ARGUMENT FROM USER'S STACK TO OUR STACK
52 000666 012621 MOV (SP)+,(R1)+ ;MOVE HIS ARGUMENT TO OUR ARGUMENT AREA
53 000670 077203 SOB R2,4$ ;GET ALL OF HIS ARGUMENTS
54 000672 010346 MOV R3,-(SP) ;NOW RESET HIS STACK POINTER
55 000674 106606 MTPD SP
56 000676 000463 BR ARGFIN
    
```

```

1          ;
2          ; Do setup for a version 2 format EMT.
3          ;
4 000700 001004 V2EMT: BNE 1$          ;BR IF NOT EMT 374
5          ;
6          ; This is an EMT 374. (R0 Contains function-code, channel).
7          ;
8 000702 062700 000000C ADD #E375MX*400,R0 ;BIAS FUNCTION CODE
9 000706 010021 MOV R0,(R1)+ ;R0 HAS FUNCTION CODE AND CHANNEL
10 000710 000456 BR ARGFIN ;THERE ARE NO OTHER ARGS FOR THIS TYPE EMT
11         ;
12         ; This is an EMT 375. (R0 points to user's argument list).
13         ;
14 000712 120427 000376 1$: CMPB R4,#376 ;MAKE SURE THIS REALLY IS AN EMT 375
15 000716 103402 BLO EMT375 ;BR IF IT IS 375
16 000720 000137 000000G JMP EMT376 ;BR IF 376 OR 377
17         ;
18         ; Validate the address of the argument block
19         ;
20 000724 032700 000001 EMT375: BIT #1,R0 ;Is address even?
21 000730 001003 BNE 1$ ;Br if not
22 000732 020037 000000G CMP R0,UHIMEM ;Is address in normal job area?
23 000736 101402 BLOS 5$ ;Br if yes
24 000740 004737 005520' 1$: CALL VALADW ;MAKE SURE EMT ARG BLOCK ADDRESS IS VALID
25         ;
26         ; Get 1st word from EMT argument block
27         ;
28 000744 113702 000000G 5$: MOVB CORUSR,R2 ;Get job index number
29 000750 052762 000000G 000000G BIS ##GEMAR,LSW11(R2);Set flag saying to return 0 on MFPD trap
30 000756 010003 MOV R0,R3 ;GET ADDRESS OF USER'S ARGUMENT BLOCK
31 000760 106523 MFPD (R3)+ ;GET USER'S FUNCTION CODE AND CHANNEL
32 000762 012605 MOV (SP)+,R5
33         ;
34         ; Determine if this is an RT-11 or TSX system function
35         ;
36 000764 020527 040000 CMP R5,#100*400 ;TSX EMT?
37 000770 103402 BLO 2$ ;BR IF NOT
38 000772 062705 000000C ADD #<<RT11EX-100>*400>,R5 ;BIAS FUNCTION CODE
39 000776 010521 2$: MOV R5,(R1)+ ;STORE FUNCTION CODE & CHANNEL # IN EMTBLK
40         ;
41         ; Move arguments from user's argument area to EMTBLK.
42         ;
43 001000 012700 000006 MOV #6,R0 ;Get # words to move
44 001004 106523 3$: MFPD (R3)+ ;GET AN ARG WORD FROM USER'S AREA
45 001006 012621 MOV (SP)+,(R1)+ ;MOVE INTO EMTBLK
46 001010 077003 SOB R0,3$
47 001012 042762 000000G 000000G BIC ##GEMAR,LSW11(R2);Clear flag that says we are getting args
48 001020 010201 MOV R2,R1 ;Carry job index in R1
49         ;
50         ; Do fast check for shared run-time mapping
51         ;
52 001022 123727 000001G 000043G CMPB EMTBLK+1,#<143+<RT11EX-100>>;Shared run-time mapping?
53 001030 001002 BNE 4$ ;Br if not
54 001032 000137 000000G JMP SSEMT ;Go do shared run-time mapping
55 001036 123727 000001G 000036 4$: CMPB EMTBLK+1,#36 ;PLAS EMT?
56 001044 001402 BEQ DOPLAS ;Br if PLAS EMT
57         ;
    
```

```
58 ; Jump into TSEM2 overlay to complete EMT processing
59 ;
60 001046 000137 0000000 ARGFIN: JMP DOEMT ;Enter TSEM2 to complete processing
61 ;
62 ; PLAS EMT
63 ;
64 001052 005737 0000000 DOPLAS: TST VPLAS ;Was PLAS genned into system?
65 001056 001402 BEQ 9# ;Br if not
66 001060 000137 0000000 JMP EMTPLS ;Enter TSPLAS overlay
67 001064 012700 0000007 9#: MOV #7,R0 ;Return error code 0
68 001070 005037 0000000 CLR URO ;Return 0 in R0
69 001074 000137 004036' JMP SETERR
70
71 ;-----
72 ; Invalid EMT
73 ;
74 001100 012700 177767 BADEMT: MOV #-11,R0 ;ERROR CODE #
75 001104 000137 004036' JMP SETERR
```

```

1          .SBTTL  EMTXIT -- Exit from EMT processing
2          ;-----
3          ; EMTXIT is jumped to at the end of processing of an EMT.
4          ; Control is returned to the user.
5          ;
6          ; Check to make sure the job context block stack did not overflow.
7          ;
8 001110 023727 0000000 123456 EMTXIT: CMP      JSTKND,#123456 ;CHECK FOR FIXED VALUE AT END OF STACK
9 001116 001410                BEQ      6$          ;BR IF OK
10 001120                DIE      #EM$SOF,#3      ;STACK OVERFLOW -- HALT SYSTEM
11          ;
12          ; If the EMT was executed in user mode, make sure
13          ; previous-mode = user in the current PSW.
14          ;
15 001140 032737 0000000 0000000 6$:  BIT      #UMODE,EMTPS ;Was EMT executed in user mode?
16 001146 001403                BEQ      7$          ;Br if not
17 001150 052737 0000000 177776     BIS      #UPMODE,@#PS ;Make sure previous-mode = user in PSW
18          ;
19          ; See if an internal error code was specified.
20          ;
21 001156 105737 0000000          7$:  TSTB     INTERR          ;ANY INTERNAL ERROR CODE?
22 001162 001403                BEQ      1$          ;BR IF NOT
23 001164 113737 0000000 0000000     MOVB    INTERR,EMTERR ;SET AS REAL ERROR CODE
24          ;
25          ; Return I/O error code to user.
26          ;
27 001172 106537 0000000          1$:  MFPD     @#ERRLOC      ;GET CONTENTS OF ERROR CELL
28 001176 113716 0000000          MOVB    EMTERR,(SP)    ;PUT EMT ERROR CODE IN LOW-ORDER BYTE
29 001202 001403                BEQ      2$          ;BR IF THERE WAS NO ERROR
30 001204 052737 0000000 0000000     BIS      #CFLAG,EMTPS ;SET C-FLAG IN PS
31 001212 106637 0000000          2$:  MTPD     @#ERRLOC      ;RESET ERROR CELL
32          ;
33          ; Return value to user in R0.
34          ;
35 001216 013700 0000000          MOV     URO,R0        ;GET VALUE TO BE RETURNED IN R0
36          ;
37          ; Restore mapping for kernel PAR 5 region
38          ;
39 001222 013737 0000000 0000000     MOV     EMTMAP,@#KPAR5 ;Restore mapping for kernel PAR 5 region
40          ;
41          ; Restore EMT context.
42          ;
43 001230 013706 0000000          MOV     EMTSP,SP      ;RESTORE FRAME POINTER
44 001234 005337 0000000          DEC     EMTLEV        ;DID WE PUSH CONTEXT INFORMATION?
45 001240 002406                BLT     3$          ;BR IF NOT
46 001242 012705 0000000          MOV     #EMTCXN,R5   ;POINT TO END OF CONTEXT AREA
47 001246 012704 0000000          MOV     #EMTCXW,R4   ;GET # WORDS TO RESTORE
48 001252 012645          4$:  MOV     (SP)+,(R5) ;RESTORE EMT CONTEXT
49 001254 077402                SOB     R4,4$
50          ;
51          ; Restore registers
52          ;
53 001256 012601          3$:  MOV     (SP)+,R1
54 001260 012602                MOV     (SP)+,R2
55 001262 012603                MOV     (SP)+,R3
56 001264 012604                MOV     (SP)+,R4
57 001266 012605                MOV     (SP)+,R5
    
```

```

58 ;
59 ; Restore stack argument pointer
60 ;
61 001270 012637 0000000 MOV (SP)+,EMTASP ;POINTS TO WHERE ARGUMENTS WERE ON STACK
62 ;
63 ; Set PS for return
64 ;
65 001274 013766 0000000 0000006 MOV EMTPS,6(SP) ;PUT PS ON STACK
66 ;
67 ; Restore EMTPS & EMTADR.
68 ;
69 001302 012637 0000000 MOV (SP)+,EMTADR
70 001306 012637 0000000 MOV (SP)+,EMTPS
71 ;
72 ; At this point the PS & PC pushed as a result of the EMT are the only
73 ; things on the stack.
74 ; See if a .SPCPS EMT was done that wants to alter return from the EMT.
75 ;
76 001312 005737 0000000 TST EMTRAD ;DO WE WANT TO ALTER EMT EXIT ADDRESS?
77 001316 001407 BEQ 5# ;BR IF NOT
78 ; Force control away from completion routine.
79 001320 013716 0000000 MOV EMTRAD,(SP) ;SET NEW PC FOR EXIT
80 001324 012766 0000000 0000002 MOV #UPMODE,2(SP) ;SET NEW PS FOR EXIT
81 001332 005037 0000000 CLR EMTRAD ;SAY WE HAVE FINISHED THE REQUEST
82 ;
83 ; Exit throuth INTEN code.
84 ;
85 001336 000137 0000000 5#: JMP SYSXIT ;EXIT FROM SYSTEM STATE
    
```

REBOOT -- Kmon EMT to reboot the system

```

1          .SBTTL REBOOT -- Kmon EMT to reboot the system
2          ;-----
3          ; Reboot the system.
4          ; Word 2 of EMT arg block contains address of start of bootstrap code.
5          ;
6 001342 REBOOT: DISABL ;** DISABLE **
7 001350 012777 000002 000100' MOV #2,@100 ;DISABLE CLOCK PROCESSING
8 001356 013737 000100 000004 MOV @#100,@#4 ;Ignore non-existent traps
9          ;
10         ; Disable time-sharing lines
11         ;
12 001364 012705 000000G MOV #LSTPL,R5 ;GET # OF LAST PRIMARY LINE
13 001370 032765 000000G 000000G 7#: BIT ##DEAD,LSW3(R5) ;IS THIS LINE INSTALLED?
14 001376 001015 BNE 6$ ;BR IF NOT INSTALLED - CONTINUE TO NEXT LINE
15 001400 016504 000000G MOV LMXNUM(R5),R4 ;IS THIS A DL-11 OR DZ-11 LINE?
16 001404 001005 BNE 4$ ;BR IF DZ-11 LINE
17 001406 005075 000000G CLR @RSR(R5) ;STOP RECEIVER
18 001412 005075 000000G CLR @TSR(R5) ;STOP TRANSMITTER
19 001416 000405 BR 6$ ;CONTINUE TO NEXT LINE
20 001420 105074 000000G 4#: CLRB @MXDTR(R4) ;CLEAR DZ-11 DATA TERMINAL READY
21 001424 052774 000000G 000000G BIS #ZCLR,@MXCSR(R4);CLEAR DZ-11 CONTROL STATUS REGISTER
22 001432 162705 000002 6#: SUB #2,R5 ;GO STOP NEXT LINE
23 001436 003354 BGT 7$
24         ;
25         ; Disable the generalized cache and the scheduler.
26         ;
27 001440 113737 000000G 000000G MOVB CORUSR,EXCJOB ;Say we are the exclusive job
28 001446 005037 000000G CLR @#VCSHNB ;Disable caching
29 001452 013705 000004G MOV EMTBLK+4,R5 ;GET THE ADDRESS OF THE BOOTSTRAP CODE
30         ;
31         ; Reset system and issue read to set unit number in controller.
32         ;
33 001456 013737 000000G 000014' MOV @#SROMMR,FILSPC ;Save MMU 0 status
34 001464 013737 000000G 000016' MOV @#SR3MMR,FILSPC+2;Save MMU 3 status
35 001472 000005 RESET ;Issue bus reset
36 001474 013737 000014' 000000G MOV FILSPC,@#SROMMR ;Restore MMU 0 status
37 001502 013737 000016' 000000G MOV FILSPC+2,@#SR3MMR;Restore MMU 3 status
38 001510 . READW #USREMT,#1,#SPFLNM,#1,#0;Read from boot device
39 001546 DISABL ;Disable interrupts
40 001554 013700 000000G MOV BOTUNI,R0 ;Get boot device unit number
41 001560 013701 000000G MOV BOTCSR,R1 ;Get boot CSR address
42         ;
43         ; Move the bootstrap code from the user's region into kernel mapping.
44         ;
45         ;
46 001564 012702 000000G MOV #TSINIT,R2 ;COPY BOOTSTRAP INTO KERNEL MAPPING
47 001570 012703 002400 MOV #256.*5,R3 ;MOVE 5 BLOCKS OF CODE
48 001574 106525 8#: MFPD (R5)+ ;GET THE NEXT WORD
49 001576 012622 MOV (SP)+,(R2)+ ;STORE IT OVER TSINIT
50 001600 077303 SOB R3,8$ ;CONTINUE UNTIL PRIMARY & SECONDARY CODE MOVED
51         ;
52         ; Clear memory management registers.
53         ;
54 001602 005037 000000G CLR SROMMR ;Clear MMU 0 status
55 001606 005037 000000G CLR SR3MMR ;Clear MMU 1 status
56         ;
57         ; Set up date and time words.

```

```
58 ;  
59 001612 013737 0000000 005004      MOV    SYSDAT,@#5004    ;DATE  
60 001620 013737 0000000 005000      MOV    SYTIMH,@#5000   ;TIME - HIGH ORDER  
61 001626 013737 0000000 005002      MOV    SYTIML,@#5002   ;TIME - LOW ORDER  
62 ;  
63 ; Copy bootstrap to low memory (0 - 4777).  
64 ;  
65 001634 012702 0000000           MOV    #TSINIT,R2      ;Point to copy of bootstrap  
66 001640 005005           CLR    R5              ;Move to physical location zero  
67 001642 012703 002400           MOV    #256.*5,R3      ;Move 5 blocks of code  
68 001646 012225 10$:          MOV    (R2)+,(R5)+     ;Move to low memory  
69 001650 077302           SOB    R3,10$         ;Continue until all is moved  
70 ;  
71 ; Set up for bootstrap.  
72 ;  
73 001652 005037 000000           CLR    @#0            ;SAY DATE & TIME ARE STORED IN MEMORY  
74 001656 010037 004722           MOV    R0,@#4722      ;SET THE CORRECT BOOTSTRAP UNIT NUMBER  
75 001662 012706 010000           MOV    #10000,SP      ;SET STACK POINTER FOR BOOTSTRAP  
76 001666 012703 001000           MOV    #1000,R3       ;Point to bootstrap code  
77 ;  
78 ; Jump into bootstrap  
79 ;  
80 001672 000113           JMP    @R3            ;ENTER BOOTSTRAP CODE
```

```
1 .SBTTL RTLOCK -- EMT to lock a job in low memory
2 -----
3 ; The RTLOCK emt is used to lock a job in memory. The job is moved
4 ; to the low end of user memory space before being locked.
5 ;
6 001674 032737 0000000 0000000 RTLOCK: BIT #PO$LOK,PRIVCO ;Are we allowed to lock job in memory?
7 001702 001003 BNE 3$ ;Br if yes
8 001704 005000 CLR R0 ;Return error code 0
9 001706 000137 004036' JMP SETERR
10 001712 105737 0000000 3$: TSTB VSWPFL ;Is this a no swap system?
11 001716 001465 BEQ 8$ ;Br if yes - don't change memory allocation
12 ;
13 ; Map PAR 5 to the memory management table
14 ;
15 001720 013737 0000000 0000000 MOV MAPPAR,@#KPAR5 ;Map PAR 5 to memory management table
16 ;
17 ; See if job is at base of memory now
18 ;
19 001726 016102 0000000 MOV LBASE(R1),R2 ;Get base page # for job
20 001732 063702 0000000 ADD BASMAP,R2 ;Get address of base page entry for job
21 001736 020237 0000000 1$: CMP R2,LOMAP ;Are we at base of memory
22 001742 101453 BLOS 8$ ;Br if yes -- don't need to reposition job
23 001744 114200 MOVB -(R2),R0 ;Get # of job using this page
24 001746 001405 BEQ 2$ ;Br if page is free
25 001750 002772 BLT 1$ ;Br if page is not available to jobs
26 001752 032760 0000000 0000000 BIT #MLOCK,LSW6(R0);Is this job locked in memory?
27 001760 001366 BNE 1$ ;Br if yes
28 ;
29 ; Job is not at base of memory.
30 ; Use the job swapper to reposition it.
31 ;
32 001762 052761 0000000 0000000 2$: BIS #NDMEM,LSW(R1) ;Set flag saying we need to be swapped
33 001770 005237 0000000 INC MEMSWP ;Tell swapper to do outswap
34 001774 052761 0000000 0000000 BIS #NLOCK,LSW6(R1);Set flag saying job needs to be locked in mem
35 002002 013702 0000000 MOV LOMAP,R2 ;Get lowest page # in use by user jobs
36 002006 163702 0000000 SUB BASMAP,R2 ;Convert to a page #
37 002012 016100 0000000 MOV LBASE(R1),R0 ;Get base page # of our job
38 002016 160200 SUB R2,R0 ;Get # pages to be swept
39 002020 004737 002104' CALL SWPFRC ;Outswap jobs in the memory area
40 ;
41 ; Now generate a swap command packet to cause our job to be
42 ; locked in memory.
43 ;
44 002024 004737 0000000 CALL SCPGET ;Get a swap command packet
45 002030 110165 0000000 MOV R1,SP$JOB(R5) ;Set our job number
46 002034 112765 0000000 0000000 MOV #SA$LOK,SP$CMD(R5) ;Set lock-in-memory command
47 ;
48 ; Now suspend execution of our job until it is repositioned and locked
49 ; in memory.
50 ;
51 002042 012700 0000000 5$: MOV #S$WFM,R0 ;State = waiting for memory expansion
52 002046 004737 0000000 CALL QNSPND ;Suspend job and do swapping to move job
53 002052 004737 0000000 CALL CHKABT ;See if we were aborted while asleep
54 002056 032761 0000000 0000000 BIT #MLOCK,LSW6(R1);Is locking finished?
55 002064 001766 BEQ 5$ ;Br if not
56 002066 000137 001110' JMP EMTXIT ;Finished
57 ;
```

```
58 ; Freeze job in current memory position
59 ;
60 002072 052761 0000000 0000000 B$: BIS ##MLDCK,LSW6(R1);Lock job in memory
61 002100 000137 001110' JMP EMTXIT ;Finished
```

```

1          .SBTTL  SWPFRC -- Force outswap of jobs in certain memory range
2          ;-----
3          ; SWPFRC is called to force the outswap of all jobs that are occupying
4          ; a certain range of memory.  The jobs are outswapped and then
5          ; placed in an executable state.
6          ;
7          ; Inputs:
8          ; R2 = Lowest page # of region to be cleaned out.
9          ; R0 = Number of 512-byte pages in region.
10         ;
11 002104 010146 SWPFRC: MOV     R1,-(SP)
12 002106 010246      MOV     R2,-(SP)
13 002110 013746 0000000      MOV     @#KPAR5,-(SP)
14         ;
15         ; Map PAR 5 to the memory map table
16         ;
17 002114 013737 0000000 0000000      MOV     MAPPAR,@#KPAR5 ;Map PAR 5 to memory allocation table
18         ;
19         ; Sweep through allocation table looking for jobs to outswap
20         ;
21 002122 063702 0000000      ADD     BASMAP,R2      ;Point to 1st entry in map table
22 002126 112201 1#:      MOVB   (R2)+,R1      ;Get info about who is using this page
23 002130 003415      BLE     2$      ;Br if page free or in use by system
24 002132 032761 0000000 0000000      BIT     ##NDMEM,LSW(R1) ;Is job already flagged for outswap?
25 002140 001011      BNE     2$      ;Br if yes
26 002142 032761 0000000 0000000      BIT     ##MLOCK,LSW6(R1);Is job locked in memory?
27 002150 001005      BNE     2$      ;Br if yes
28         ;
29         ; We found a job that needs to be flagged for outswap
30         ;
31 002152 052761 0000000 0000000      BIS     ##NDMEM,LSW(R1) ;Flag job for outswapping
32 002160 005237 0000000      INC     MEMSWP      ;Remember another job to outswap
33         ;
34         ; Check rest of memory region
35         ;
36 002164 077020 2$:      SOB     R0,1$      ;Loop if more of region to check
37         ;
38         ; Finished
39         ;
40 002166 012637 0000000      MOV     (SP)+,@#KPAR5 ;Restore PAR 5 mapping
41 002172 012602      MOV     (SP)+,R2
42 002174 012601      MOV     (SP)+,R1
43 002176 000207      RETURN

```

```
1          .SBTTL  QIO  -- Queue an I/O request
2          ;-----
3          ; QIO is called to add an I/O request to the list of waiting
4          ; requests for a device.  If the device is idle it is started.
5          ;
6          ; Inputs:
7          ; R1 = Address of I/O queue entry.
8          ;
9          ;
10         ; Entry point for system I/O requests.
11         ;
12 002200 013761 0000000 0000000 SYQIO:  MOV     @#KPAR5,Q.PA5(R1);Save PAR 5 mapping
13         ;
14         ; Entry point for normal job I/O requests
15         ;
16 002206 010246          QIO:    MOV     R2,-(SP)
17 002210 013746 0000000          MOV     @#KPAR5,-(SP)
18         ;
19         ; Increment I/O counters.
20         ;
21 002214 016102 0000000          MOV     Q.UCSW(R1),R2 ;Address of user's channel block
22 002220 001412          BEQ     1$ ;Br if no channel block
23 002222 020227 0000000          CMP     R2,#CXTBAS ;Is channel in job context block?
24 002226 103405          BLO     3$ ;Br if not
25 002230 016137 0000000 0000000          MOV     Q.PA6(R1),@#KPAR5 ;Map par 5 to loc where context blk is
26 002236 062702 0000000          ADD     #<VPAR5-CXTBAS>,R2;Get address of mapped channel block
27 002242 105262 0000000          3$:   INCB   C.NUMQ(R2) ;Increment # I/O requests active on channel
28 002246 116100 0000000          1$:   MOVB   Q.JOB(R1),R0 ;Get job #
29 002252 001410          BEQ     2$ ;Br if system doing I/O
30 002254 006300          ASL     R0 ;Convert to job index number
31 002256 105260 0000000          INCB   LIOCNT(R0) ;Inc # I/O requests active for this job
32 002262 116160 0000000 0000010          MOVB   Q.DEVX(R1),LIOCNT+1(R0) ;Save index # of last dev job accessed
33 002270 005237 0000000          INC     UIOCNT ;Count total # of active user I/O operations
34         ;
35         ; Call IOSTRT to add I/O queue request to list for handler and
36         ; start the I/O.
37         ;
38 002274 004737 002310'          2$:   CALL   IOSTRT ;Add queue entry to handler and start I/O
39         ;
40         ; Finished
41         ;
42 002300 012637 0000000          MOV     (SP)+,@#KPAR5
43 002304 012602          MOV     (SP)+,R2
44 002306 000207          RETURN
```

```

1          .SBTTL  IOSTRT -- Try to start I/O transfer
2          ;-----
3          ; IOSTRT is called to initiate an I/O operation.
4          ; It performs the following operations:
5          ;   1. Allocate a Unibus Map Register (UMR) if I/O operation is being
6          ;      directed to a DMA device.
7          ;   2. Move data from user's buffer to system buffer if this is an
8          ;      I/O to a device that requires system buffer mapping.
9          ;   3. Put I/O queue entry on list for device handler.
10         ;   4. Initiate device operation if it is idle.
11         ;
12         ; Inputs:
13         ;   R1 = Address of I/O queue entry.
14         ;
15 002310 010246 IOSTRT: MOV     R2,-(SP)
16 002312 010446         MOV     R4,-(SP)
17         ;
18         ; Get index number of device this I/O operation is directed to.
19         ;
20 002314 116102 000000G         MOVB   Q.DEVX(R1),R2 ;GET DEVICE INDEX NUMBER
21 002320 001004         BNE     2$ ;BR IF I/O TO REAL DEVICE
22         ;
23         ; If device index number is zero, call IOCMPL to indicate I/O is finished
24         ;
25 002322 010104         MOV     R1,R4 ;GET I/O QUEUE ELEMENT ADDRESS TO R4
26 002324 004737 003032'         CALL   IOCMPL ;Do I/O completion processing
27 002330 000427         BR      QIOFIN ;WE ARE FINISHED WITH I/O OPERATION
28         ;
29         ; See if we are doing caching for this device
30         ;
31 002332 032762 000000G 000000G 2$: BIT     #DX#NCA,DVFLAG(R2);Is this device eligible for caching?
32 002340 001006         BNE     4$ ;Br if not
33 002342 005737 000000G         TST    CSHALC ;Is data caching genned into system?
34 002346 001403         BEQ     4$ ;Br if not
35 002350 004777 000000G         CALL   @CSHID ;See if doing data caching for this device
36 002354 103415         BCS    QIOFIN ;Br if caching took over the operation
37         ;
38         ; See if we need to do system buffering for this I/O operation
39         ;
40 002356 032762 000000G 000000G 4$: BIT     #DX#MAP,DVFLAG(R2);Does this device require buffer mapping?
41 002364 001404         BEQ     3$ ;Br if not
42 002366         DCALL   MIOCHK ;See if this particular operation needs maping
43 002374 103405         BCS    QIOFIN ;Br if I/O mapping is being done
44         ;
45         ; See if we need to allocate a Unibus map register set for this operation
46         ;
47 002376 004737 004470' 3$: CALL   GETUMR ;SEE IF WE NEED TO ALLOCATE A UMR
48 002402 103402         BCS    QIOFIN ;BR IF NO FREE UMR -- DEFER STARTING OPERATION
49         ;
50         ; Place queue element on list for device handler
51         ;
52 002404 004737 002416'         CALL   IOHANG ;Place queue element on list for handler
53         ;
54         ; Finished
55         ;
56 002410 012604 QIOFIN: MOV     (SP)+,R4
57 002412 012602         MOV     (SP)+,R2

```

58 002414 000207

RETURN

```

1          .SBTTL IOHANQ -- Place I/O queue request on handler list
2          ;-----
3          ; Place an I/O queue request on the list of pending requests for a device
4          ; handler and enter the handler front-end if the handler is currently
5          ; idle.
6          ;
7          ; Inputs:
8          ; R1 = Address of I/O queue entry
9          ;
10         IOHANQ: MOV     R1,-(SP)
11         MOV     R2,-(SP)
12         MOV     R3,-(SP)
13         MOV     R4,-(SP)
14         MOV     R5,-(SP)
15         MOV     @#KPAR5,-(SP) ; Preserve system PAR 5 mappint
16         ;
17         ; Increment count of uncompleted I/O operations for this device
18         ;
19         MOVB   Q.DEVX(R1),R2 ; Get device index number
20         INC    HANIOC(R2)    ; Say another I/O operation started for dev
21         ;
22         ; Get address of device handler and set up PAR 5 if this is a mapped handler
23         ;
24         MOV    HANPAR(R2),R0 ; Is this a mapped handler?
25         BEQ   5$             ; Br if not
26         MOV    R0,@#KPAR5    ; Map kernel PAR 5 to handler
27         5$:  MOV    HANENT(R2),R2 ; R2 --> 4th word of handler
28         ;
29         ; Make R1 point to 3rd word of queue element (as handler wants it).
30         ;
31         CLR    Q.LINK(R1)    ; SAY QUEUE ELEMENT WILL BE AT END OF CHAIN
32         ADD    #Q.BLKN,R1    ; MAKE R1 POINT TO 3RD WORD OF QUEUE ELEMENT
33         ;
34         ; Add new queue element to list of requests for this device.
35         ;
36         DISABL ; * DISABLE
37         MOV    (R2),R0       ; * ADDRESS OF LAST Q ELEMENT IN LIST
38         BEQ   1$             ; * BR IF HANDLER IS IDLE NOW
39         ;
40         ; Handler is active.
41         ; Just add our queue element to its list.
42         ;
43         MOV    R1,Q.LINK-Q.BLKN(R0); * SET FORWARD LINK TO POINT TO US
44         MOV    R1,(R2)       ; * STORE OUR Q ELEM ADDR INTO LQE IN HANDLER
45         ENABL
46         BR    9$
47         ;
48         ; Handler is idle.
49         ; Put queue entry into its list and start it.
50         ;
51         1$:  MOV    R1,(R2)+   ; * STORE Q POINTER INTO HANDLER LQE
52         MOV    R1,(R2)+   ; * STORE Q POINTER INTO HANDLER CQE
53         ENABL
54         ;
55         ; Enter system state so that we will be running on interrupt stack.
56         ; We do this so the handler front-end can remap kernel PAR6 if it wants to.
57         ;

```

```
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
```

```

1          .SBTTL IOFIN -- I/O operation completion
2          ;-----
3          ; IOFIN is called by the device driver when it completes an I/O operation.
4          ; IOFIN queues a completion routine request for the user if it was specified
5          ; with the EMT, then tries to initiate any other pending I/O requests for
6          ; the device.
7          ;
8          ; Inputs:
9          ; R4 = Address of cell in handler with pointer to current queue entry.
10         ;
11 002570 010046 IOFIN: MOV     R0,-(SP)
12 002572 010146         MOV     R1,-(SP)
13 002574 010346         MOV     R3,-(SP)
14 002576 010446         MOV     R4,-(SP)
15 002600 010546         MOV     R5,-(SP)
16         ;
17         ; If handler is being held, return immediately without doing any cleanup.
18         ;
19 002602 006264 177774         ASR     -4(R4)      ;Is handler being held?
20 002606 100457         BMI     9$          ;Br if yes
21         ;
22         ; Handler is not being held.
23         ; Remove completed (current) queue entry from list for handler.
24         ;
25 002610 010405         MOV     R4,R5          ;Get address of CQE cell in handler
26 002612         DISABL          ;** Disable interrupts **
27 002620 011501         MOV     (R5),R1      ;Address of 3rd word of current Q element
28 002622 001004         BNE     1$          ;Br if there is a completed Q element
29 002624         ENABL          ;Handler has no active queue element
30 002632 000445         BR     9$
31 002634 162701 000000G 1$: SUB     #Q.BLKN,R1    ;Point to 1st word of queue element
32 002640 016115 000000G     MOV     Q.LINK(R1),(R5) ;Delink queue element from handler list
33 002644 011503         MOV     (R5),R3      ;Remember if we need to restart handler
34 002646 001002         BNE     2$          ;Br if there are more pending elements
35 002650 005065 177776         CLR     -2(R5)      ;Clear LQE pointer in handler
36         ;
37         ; At this point we have removed the completed queue element from the
38         ; handlers list.
39         ; We have set the current queue element (CQE) pointer in the handler to
40         ; point to the next queue element (if any) and have zeroed the last
41         ; queue element (LQE) pointer if there are no remaining queue entries.
42         ; R1 = Address of 1st word of completed queue element.
43         ; R3 = Non-zero ==> Need to restart handler for next pending I/O op.
44         ; R5 = Pointer to CQE cell in handler.
45         ;
46 002654 2$: ENABL          ;** Enable interrupts **
47         ;
48         ; Decrement count of uncompleted I/O operations for this device
49         ;
50 002662 116100 000000G     MOVVB  Q.DEVX(R1),R0   ;Get device index number
51 002666 005360 000000G     DEC     HANIOC(R0)   ;Dec # uncompleted I/O operations for dev
52         ;
53         ; If the device handler did not do a .FORK before calling IOFIN, we
54         ; queue a fork request to do the completion processing for this
55         ; queue element.
56         ; If the handler did do a .FORK, we do the completion processing now.
57         ;

```

```
58 002672 105737 0000000      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 0000000      3$:   CALL   FRKGET      ;Get a free fork request block (addr in R4)
66 002712 112764 0000000 0000000  MOVB   #FP$IOF,FQ$PRI(R4);Set fork processing priority
67 002720 012764 002762' 0000000  MOV    #IOFCPL,FQ$RTN(R4);Set address of fork routine
68 002726 010164 0000000      MOV    R1,FQ$R1(R4)  ;Pass address of queue element in R1
69 002732 010364 0000000      MOV    R3,FQ$R3(R4)  ;Pass restart flag in R3
70 002736 010564 0000000      MOV    R5,FQ$R5(R4)  ;Pass address of CQE cell in R5
71 002742 004737 0000000      CALL   FORKQ       ;Queue the fork request
72                               ;
73                               ; Finished
74                               ;
75 002746 012605              9$:   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
```

```

1          .SBTTL IOFCPL -- IOFIN processing done at fork level
2          ;-----
3          ; This routine performs the part of the IOFIN operation that must
4          ; run at fork level.
5          ; This routine does the completion processing for the completed queue
6          ; element and then recalls the handler front end if there is another
7          ; queue element waiting to be started.
8          ;
9          ; Inputs:
10         ; R1 = Address of 1st word of completed queue element.
11         ; R3 = Non-zero ==> Must call handler to start pending I/O operation
12         ; R5 = Address of CQE cell in handler.
13         ;
14 002762 010446 IOFCPL: MOV     R4,-(SP)
15         ;
16         ; Perform completion operation on queue element
17         ;
18 002764 010104         MOV     R1,R4           ;Get address of completed Q element
19 002766 004737 003032' CALL    IOCMPL          ;Do completion processing
20         ;
21         ; If there is a pending queue element for the handler, call the handler
22         ; front end to start the next operation.
23         ;
24 002772 005703         TST     R3           ;Do we need to recall handler?
25 002774 001414         BEQ     9%           ;Br if not
26         ;
27         ; There is a pending I/O operation.
28         ; Call handler initiation section.
29         ;
30 002776 010046         MOV     R0,-(SP)           ;Handler front end may use any register
31 003000 010146         MOV     R1,-(SP)
32 003002 010246         MOV     R2,-(SP)
33 003004 010346         MOV     R3,-(SP)
34 003006 010546         MOV     R5,-(SP)
35 003010 004765 000002 CALL    2(R5)           ;Call handler front end to start next op
36 003014 012605         MOV     (SP)+,R5
37 003016 012603         MOV     (SP)+,R3
38 003020 012602         MOV     (SP)+,R2
39 003022 012601         MOV     (SP)+,R1
40 003024 012600         MOV     (SP)+,R0
41         ;
42         ; Finished
43         ;
44 003026 012604 9%:    MOV     (SP)+,R4
45 003030 000207         RETURN
  
```

```

1          .SBTTL IOCMPL -- Do cleanup on completed I/O queue element
2          ;-----
3          ; IOCMPL is called to do the cleanup operation on an I/O queue
4          ; element that belongs to an I/O operation that has just completed.
5          ;
6          ; Inputs:
7          ; R4 = Address of I/O queue element to be cleaned up.
8          ;
9          ; Outputs:
10         ; All cleanup is done on the queue element and the queue element is
11         ; returned to the free list or is used to queue a completion routine.
12         ;
13 003032 010046 IOCMPL: MOV     RO,-(SP)
14 003034 010146         MOV     R1,-(SP)
15 003036 010346         MOV     R3,-(SP)
16 003040 013746 0000000 MOV     @#KPAR5,-(SP) ; Save par 5 mapping
17 003044 010401         MOV     R4,R1 ; Get address of I/O queue element to R1
18 003046 004737 004772' CALL    FREUMR ; Release any unibus map reg used for I/O
19 003052 105764 0000000 TSTB   Q.FUNC(R4) ; Special lookup/enter?
20 003056 003403         BLE     1$ ; Br if not
21 003060 016437 0000000 000046' MOV     Q.WCNT(R4),SPSIZE ; Save file size
22 003066 016403 0000000 1$: MOV     Q.UCSW(R4),R3 ; Get address of user's channel status block
23 003072 116401 0000000 MOVVB  Q.JOB(R4),R1 ; Get job number associated with request
24 003076 001405         BEQ     7$ ; Br if request came from system
25 003100 006301         ASL     R1 ; Convert job # to job index number
26 003102 105361 0000000 DECB   LIOCNT(R1) ; Dec # I/O operations active for this user
27 003106 005337 0000000 DEC     UIOCNT ; One more user I/O operation finished
28 003112 005703 7$: TST     R3 ; Is there a channel block?
29 003114 001427         BEQ     12$ ; Br if not
30 003116 020327 0000000 CMP     R3,#CXTBAS ; Is channel in mapped job context block?
31 003122 103405         BLO     9$ ; Br if not
32 003124 016437 0000000 0000000 MOV     Q.PA6(R4),@#KPAR5; Map par 5 to job context block
33 003132 062703 0000000 ADD     #<VPAR5-CXTBAS>,R3; Get address of mapped channel in par 5
34 003136 016400 0000000 9$: MOV     Q.ICSW(R4),RO ; Get internal CSW value
35 003142 042700 0000000 BIC     #^C<CS$EOF!CS$ERR>,RO ; Clear all but error and eof flags
36 003146 050063 0000000 BIS     RO,C.CSW(R3) ; Transfer error and eof flags to user's CSW
37 003152 016400 0000000 MOV     Q.ICSW+C.USED(R4),RO; Get highest block # written
38 003156 020063 0000000 CMP     RO,C.USED(R3) ; Compare with blk # in original chan blk
39 003162 101402         BLOS   10$ ; Br if handler didn't increase
40 003164 010063 0000000 MOV     RO,C.USED(R3) ; Save high block # in original channel block
41 003170 105363 0000000 10$: DECB   C.NUMQ(R3) ; Dec # I/O operations on channel
42         ;
43         ; See if user is waiting for I/O to finish.
44         ;
45 003174 005701 12$: TST     R1 ; IS THIS A SYSTEM I/O REQUEST?
46 003176 001406         BEQ     2$ ; BR IF YES
47 003200 026127 0000000 0000000 CMP     LSTATE(R1),#S$IDWT ; IS USER WAITING FOR I/O TO FINISH?
48 003206 001002         BNE     2$ ; BR IF NOT
49 003210 004737 0000000 CALL    QHIPRI ; PLACE USER IN RUN QUEUE
50         ;
51         ; See if we need to queue a completion routine request for user.
52         ;
53 003214 026427 0000000 0000001 2$: CMP     Q.COMP(R4),#1 ; WAS A COMPLETION ROUTINE SPECIFIED?
54 003222 101451         BLOS   3$ ; BR IF NOT
55         ;
56         ; Queue a completion routine request for user.
57         ; Convert I/O queue entry into completion queue entry format.

```

```

58 ;
59 003224 016464 0000000 0000000 MOV Q.CHAN(R4),CQ#R1(R4) ;CHANNEL # GOES IN R1 ON COMPL CALL
60 003232 016364 0000000 0000000 MOV C.CSW(R3),CQ#R0(R4) ;CSW GOES IN R0 ON COMPL CALL
61 003240 110164 0000000 MOV R1,CQ#JOB(R4) ;SET JOB #
62 003244 001415 BEQ 8$ ;Br if system completion
63 003246 005761 0000000 TST LITIME(R1) ;Is job in an interactive state?
64 003252 001404 BEQ 5$ ;Br if not
65 003254 112764 0000000 0000000 MOV #S#HICP,CQ#RNS(R4);Set interactive-computation as state
66 003262 000403 BR 6$
67 003264 112764 0000000 0000000 5$: MOV #S#IOFN,CQ#RNS(R4);SET EXECUTION STATE FOR COMPLETION ROUTINE
68 003272 116164 0000000 0000000 6$: MOV LPRI(R1),CQ#PRI(R4);Set execution priority value
69 003300 016464 0000000 0000000 8$: MOV Q.PA5(R4),CQ#PA5(R4);SET EMT ENTRY MAPPING FOR PAR 5
70 003306 116464 0000000 0000000 MOV Q.FLAG(R4),CQ#FLG(R4) ;Copy control flags
71 003314 112764 0000000 0000000 MOV #CP#STD,CQ#CP(R4);Set standard completion class priority
72 003322 132764 0000000 0000000 BITB #<QF#MID!QF#CID>,CQ#FLG(R4);Secondary op for MIO or cache?
73 003330 001403 BEQ 11$ ;Br if not
74 003332 112764 0000000 0000000 MOV #CP#RT,CQ#CP(R4);Run cache completion routines at higher prio
75 003340 004737 003370' 11$: CALL QCOMPL ;QUEUE A COMPLETION ROUTINE FOR USER
76 003344 000403 BR 4$
77 ;
78 ; We don't need to queue a completion routine
79 ; so just return I/O queue element to the free list.
80 ;
81 003346 010401 3$: MOV R4,R1 ;GET ADDRESS OF Q ENTRY TO R1
82 003350 004737 004222' CALL QFREE ;FREE THE I/O QUEUE ELEMENT
83 ;
84 ; Finished with cleanup.
85 ;
86 003354 012637 0000000 4$: MOV (SP)+,@#KPAR5
87 003360 012603 MOV (SP)+,R3
88 003362 012601 MOV (SP)+,R1
89 003364 012600 MOV (SP)+,R0
90 003366 000207 RETURN
  
```

QCOMPL -- Queue a completion routine request

```

1          .SBTTL  QCOMPL -- Queue a completion routine request
2          ;-----
3          ; QCOMPL is called to queue a completion routine request for a user.
4          ;
5          ; Inputs:
6          ; R4 = Address of completion request queue element (must be set up).
7          ;
8 003370 010146 QCOMPL: MOV     R1,-(SP)
9 003372 010246         MOV     R2,-(SP)
10         ;
11         ; See if this is a completion request for a system or user operation.
12         ;
13 003374 116401 0000000     MOVB   CQ$JOB(R4),R1  ;GET JOB # ASSOCIATED WITH REQUEST
14 003400 001031         BNE    1$          ;BR IF THIS IS A USER COMPL REQUEST
15         ;
16         ; This is a completion request for a system operation.
17         ; Call the completion routine directly.
18         ;
19 003402 016400 0000000     MOV     CQ$RO(R4),R0  ;SET UP REGISTERS FOR COMPL ROUTINE
20 003406 016401 0000000     MOV     CQ$R1(R4),R1
21 003412 010246         MOV     R2,-(SP)
22 003414 010346         MOV     R3,-(SP)
23 003416 010446         MOV     R4,-(SP)
24 003420 010546         MOV     R5,-(SP)
25 003422 013746 0000000     MOV     @#KPAR5,-(SP)  ;Save current PAR 5 mapping
26 003426 016437 0000000 0000000     MOV     CQ$PA5(R4),@#KPAR5;Set up mapping for PAR 5 region
27 003434 004774 0000000     CALL    @CQ$RTN(R4)  ;CALL SYSTEM COMPLETION ROUTINE
28 003440 012637 0000000     MOV     (SP)+,@#KPAR5  ;Restore PAR 5 mapping
29 003444 012605         MOV     (SP)+,R5
30 003446 012604         MOV     (SP)+,R4
31 003450 012603         MOV     (SP)+,R3
32 003452 012602         MOV     (SP)+,R2
33         ;
34         ; Finished calling system completion routine.
35         ; Return queue element to free list.
36         ;
37 003454 010401         MOV     R4,R1          ;GET ADDRESS OF COMPLETION QUEUE ENTRY
38 003456 004737 004222'     CALL    QFREE        ;RETURN COMPL QUEUE ELEMENT TO FREE LIST
39 003462 000477         BR     9$          ;FINISHED
40         ;
41         ; This is a completion routine request for a user job.
42         ; Add the completion request to the list of pending routines for this job.
43         ;
44 003464         1$:  DISABL          ;** Disable **
45 003472 010100         MOV     R1,R0          ;Get current job number
46 003474 062700 0000000     ADD     #LCMPL-CQ$LNK,R0;Get address of LCMPL(R1)
47 003500 010002         2$:  MOV     R0,R2          ;Move addr of next compl queue element to R2
48 003502 016200 0000000     MOV     CQ$LNK(R2),R0  ;Get addr of following queue element
49 003506 001416         BEQ    3$          ;Br if we are at the end of the list
50 003510 126460 0000000 0000000     CMPB   CQ$CP(R4),CQ$CP(R0);Compare completion class priorities
51 003516 103770         BLO    2$          ;Br if new element is of lower priority
52 003520 101011         BHI    3$          ;Br if new element is of higher priority
53 003522 126460 0000000 0000000     CMPB   CQ$RNS(R4),CQ$RNS(R0);Compare state of new one to next
54 003530 101363         BHI    2$          ;Br if new element is of lower priority state
55 003532 103404         BLO    3$          ;Br if new element is of higher priority state
56 003534 126460 0000000 0000000     CMPB   CQ$PRI(R4),CQ$PRI(R0);States are the same, compare prio value
57 003542 101756         BLOS   2$          ;Br if new priority is lower or equal

```

```

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

```

```

1          .SBTTL  FAKCMP -- Generate fake completion routine request
2          ;-----
3          ; FAKCMP is called to generate a completion queue request for I/O operations
4          ; such as TT I/O and spooled I/O that never enter a normal I/O queue request.
5          ;
6          ; Inputs:
7          ;   EMTBLK = EMT argument block for readc or writc.
8          ;   CHNNUM = User's channel number.
9          ;
10         003670 010146 FAKCMP: MOV     R1,-(SP)
11         003672 010446         MOV     R4,-(SP)
12         ;
13         ; Get an I/O queue entry
14         ;
15         003674 004737 004102'         CALL    GETQ          ;GET AN I/O QUEUE ENTRY FROM FREE LIST
16         ;
17         ; Set up queue entry to look like a completion queue request.
18         ; (Address of queue element is now in R1)
19         ;
20         003700 113761 0000000 0000000         MOVB   CQRUSR,CQ#JOB(R1);SET JOB # IN QUEUE ENTRY
21         003706 005061 0000000             CLR    CQ#RO(R1)      ;SAY CSW IS ZERO (NO ERRORS)
22         003712 013761 0000000 0000000         MOV    CHNNUM,CQ#R1(R1);RETURN USER'S CHANNEL # IN R1
23         003720 013700 0000100             MOV    EMTBLK+10,RO   ;GET ADDRESS OF COMPL ROUTINE
24         003724 004737 005352'             CALL    UACHKW       ;MAKE SURE ADDRESS IS LEGAL
25         003730 103434                     BCS    9#           ;BR IF INVALID
26         003732 010061 0000000             MOV    RO,CQ#RTN(R1) ;SET COMPLETION ROUTINE ADDRESS
27         003736 013761 0000000 0000000         MOV    EMTMAP,CQ#PA5(R1);SET EMT ENTRY PAR 5 MAPPING
28         003744 112761 0000000 0000000         MOVB   #CP#STD,CQ#CP(R1);Set standard completion class priority
29         003752 113700 0000000             MOVB   CQRUSR,RO     ;Get job number
30         003756 116061 0000000 0000000         MOVB   LPRI(RO),CQ#PRI(R1);Set job execution priority
31         003764 112761 0000000 0000000         MOVB   #S#IDFN,CQ#RNS(R1);SET EXECUTION STATE FOR COMPL ROUTINE
32         003772 005761 0000000             TST    LITIME(R1)   ;Is this an interactive job?
33         003776 001403                     BEQ    1#           ;Br if not
34         004000 112761 0000000 0000000         MOVB   #S#HICP,CQ#RNS(R1);Set execution state for interactive job
35         ;
36         ; Queue completion request for this job.
37         ;
38         004006 010104 1#: MOV     R1,R4          ;GET ADDRESS OF COMPL Q ELEMENT TO R4
39         004010 004737 003370'           CALL    QCOMPL       ;QUEUE COMPL REQUEST FOR JOB
40         ;
41         ; Finished
42         ;
43         004014 012604                     MOV    (SP)+,R4
44         004016 012601                     MOV    (SP)+,R1
45         004020 000207                     RETURN
46         ;
47         ; Error: Invalid completion routine address.
48         ;
49         004022 004737 004222'           9#: CALL    QFREE     ;RETURN THE QUEUE ELEMENT
50         004026 012700 177766           MOV    #-12,RO      ;RETURN INVALID-EMT ERROR CODE
51         004032 000137 004036'           JMP    SETERR

```

SETERR -- Set EMT error code and exit

```

1          .SBTTL SETERR -- Set EMT error code and exit
2          ;-----
3          ; SETERR is jumped to to set an EMT error code and exit from the
4          ; EMT processing.
5          ;
6          ; Inputs:
7          ; RO = EMT error code to be returned.
8          ;
9 004036 110037 0000000 SETERR: MOVB RO,EMTERR ;SAVE ERROR CODE
10 004042 002010          BGE ERROK ;BR IF POSITIVE VALUE
11          ;
12          ; We have a negative error code.
13          ; This implies a fatal error unless a .SERR was done.
14          ;
15 004044 105737 0000000          TSTB SERFLG ;WAS A .SERR DONE?
16 004050 001005          BNE ERROK ;BR IF YES
17          ; Fatal EMT error.
18          ; Abort the job.
19 004052 010005          MOV RO,R5 ;GET ABORT CODE
20 004054 013704 0000000 EMTABT: MOV EMTADR,R4 ;GET ADDRESS OF EMT INSTRUCTION
21 004060 000137 0000000          JMP ABORT ;ABORT THE JOB
22          ;
23          ; Non-fatal error. Return with c-flag set.
24          ;
25 004064 105037 0000000 ERROK: CLRB INTERR ;IGNORE INTERNAL ERROR CODE
26 004070 052737 0000000 0000000 SETC: BIS #CFLAG,EMTPS ;SET C-FLAG IN PS
27 004076 000137 001110'          JMP EMTXIT

```

GETQ -- Get a free I/O queue element

```

1          .SBTTL  GETQ  -- Get a free I/O queue element
2          ;-----
3          ; GETQ is called to get a free I/O queue element.
4          ; If there are no free elements available, the user is suspended until
5          ; one becomes available.
6          ;
7          ; Outputs:
8          ; R1 = Address of I/O queue element obtained.
9          ;
10         004102  GETQ:  DISABL          ;;;DISABLE INTERRUPTS
11         ;
12         ; Reserve the last few queue elements for system I/O use.
13         ;
14         004110  005737  0000000  TST      NMFREQ          ;;;ANY QUEUE ELEMENTS WE CAN USE?
15         004114  003011          BGT      1$              ;;;BR IF YES
16         ;
17         ; There are no free I/O queue elements.
18         ; Put user in queue waiting for one.
19         ;
20         004116  005237  000026'  INC      QWTCNT          ;;;Say another job waiting for queue element
21         004122  012700  0000000  MOV      #S$NEDQ,RO      ;;;QUEUE FOR JOBS WAITING FOR Q ELEMENTS
22         004126  004737  0000000  CALL    QNSPND          ;;;ADD JOB TO END OF QUEUE LIST (ENABLE INTS)
23         004132  004737  0000000  CALL    CHKABT          ;See if job was aborted while asleep
24         004136  000761          BR      GETQ            ;GO BACK AND TRY TO GET A QUEUE ELEMENT
25         ;
26         ; We have found a free queue element.
27         ; Remove it from the free list.
28         ;
29         004140  005337  0000000  1$:    DEC      NMFREQ          ;;;DEC # FREE QUEUE ELEMENTS
30         004144  013701  0000000  MOV      FREIDQ,R1       ;;;GET ADDRESS OF 1ST FREE Q ELEMENT
31         004150  016137  0000000  0000000  MOV      Q.LINK(R1),FREIDQ; ;REMOVE ELEMENT FROM FREE LIST
32         004156          ENABL          ;Enable interrupts
33         ;
34         ; Zero the I/O queue element
35         ;
36         004164  004737  004200'  CALL    ZEROQ            ;ZERO THE QUEUE ELEMENT
37         ;
38         ; Save info about current job context block mapping in queue element
39         ;
40         004170  013761  0000000  0000000  MOV      @#KPAR6,Q.PA6(R1); Save context block mapping
41         ;
42         ; Finished
43         ;
44         004176  000207          RETURN
45         ;
46         ;-----
47         ; ZEROQ is called to zero an I/O queue element.
48         ;
49         ; Inputs:
50         ; R1 = Address of I/O queue element
51         ;
52         004200  010146  ZEROQ:  MOV      R1,-(SP)
53         004202  010246          MOV      R2,-(SP)
54         004204  012702  0000000  MOV      #IQSIZ/2,R2     ;GET # WORDS IN I/O QUEUE ELEMENT
55         004210  005021  1$:    CLR      (R1)+        ;ZERO ALL WORDS IN QUEUE ELEMENT
56         004212  077202          SOB      R2,1$
57         004214  012602          MOV      (SP)+,R2

```

58 004216 012601
59 004220 000207

MOV (SP)+, R1
RETURN

QFREE -- Return an I/O queue element to the free list

```

1          .SBTTL  QFREE  -- Return an I/O queue element to the free list
2          ;-----
3          ; QFREE is called to return an I/O queue element to the free list.
4          ;
5          ; Inputs:
6          ; R1 = Address of queue element to be freed.
7          ; All registers are preserve.
8          ;
9 004222 010046 QFREE:  MOV     RO,-(SP)
10 004224          DISABL          ;* DISABLE INTERRUPTS
11 004232 013700 0000000 MOV     FREIQ,RO          ;* GET CURRENT HEAD OF FREE LIST
12 004236 010061 0000000 MOV     RO,Q.LINK(R1)    ;* CHAIN NEW ELEMENT INTO FREE LIST
13 004242 010137 0000000 MOV     R1,FREIQ        ;*
14 004246          ENABL          ;ENABLE INTERRUPTS
15 004254 005237 0000000 INC     NMFREQ          ;INC # FREE QUEUE ELEMENTS
16 004260 003415          BLE     9$          ;BR IF NOT ENOUGH FOR USER JOBS TO USE
17          ;
18          ; Restart users waiting for free I/O queue elements.
19          ;
20 004262 005737 000026' TST     QWTCNT          ;Are any jobs waiting for a queue element?
21 004266 001412          BEQ     9$          ;Br if not
22 004270 012700 0000000 MOV     #S$NEDQ,RO      ;Get wait state
23 004274 004737 0000000 CALL    QSRCH          ;See if any jobs waiting for queue element
24 004300 103403          BCS     2$          ;Br if no jobs waiting
25 004302 004737 0000000 CALL    QHIPRI         ;Requeue job in high-priority state
26 004306 000402          BR     9$          ;
27 004310 005037 000026' 2$:  CLR     QWTCNT          ;Restarted all waiting jobs
28          ;
29          ; Finished
30          ;
31 004314 012600 9$:  MOV     (SP)+,RO
32 004316 000207          RETURN

```

```

1          .SBTTL  GETSYQ -- Get queue element for system I/O
2          ;-----
3          ; GETSYQ is called to get a queue element to be used for system
4          ; (kernel mode) I/O operations.
5          ; GETSYQ gets a free queue element and sets up information in it.
6          ;
7          ; Inputs:
8          ; R1 = Address of system channel block to be used for I/O.
9          ;
10         ; Outputs:
11         ; R1 = Address of I/O queue element obtained.
12         ;
13 004320 010246 GETSYQ: MOV     R2, -(SP)
14 004322 010102         MOV     R1, R2           ; GET ADDRESS OF CHANNEL BLOCK
15         ;
16         ; Get a free I/O queue element
17         ;
18 004324 004737 004414'         CALL    GETRTQ           ; GET A QUEUE ELEMENT (ADDRESS RETURNED IN R1)
19         ;
20         ; Set up information in queue element about the channel
21         ;
22 004330 010261 000000G         MOV     R2, Q.UCSW(R1)   ; ADDRESS OF CSW FOR CHANNEL
23 004334 010100         MOV     R1, R0           ; GET ADDRESS OF I/O QUEUE BLOCK
24 004336 062700 000000G         ADD     #Q.ICSW, R0      ; POINT TO INTERNAL CSW WORD
25 004342 010061 000000G         MOV     R0, Q.CSW(R1)    ; SET POINTER TO INTERNAL CSW WORD
26 004346 016210 000000G         MOV     C.CSW(R2), (R0)  ; INITIALIZE CSW WORD
27 004352 016261 000000G 000000G  MOV     C.SBLK(R2), Q.BLKN(R1); BASE BLOCK # OF FILE
28 004360 116261 000000G 000000G  MOVVB  C.DEVQ(R2), Q.UNIT(R1); DEVICE UNIT #
29 004366 012761 000000G 000000G  MOV     #VPAR6, Q.BUFF(R1); SET BUFFER ADDRESS TO MAP THROUGH PAR 6
30 004374 016202 000000G         MOV     C.CSW(R2), R2     ; GET CHANNEL STATUS WORD
31 004400 042702 177701         BIC     #^C76, R2       ; CLEAR ALL BUT DEVICE INDEX NUMBER
32 004404 110261 000000G         MOVVB  R2, Q.DEVX(R1)    ; SET DEVICE INDEX NUMBER
33         ;
34         ; Finished
35         ;
36 004410 012602         MOV     (SP)+, R2
37 004412 000207         RETURN
38         ;
39         .SBTTL  GETRTQ -- Get queue element for real-time use
40         ;-----
41         ; GETRTQ is called to get a free I/O queue element for system or
42         ; real-time use.
43         ;
44         ; Outputs:
45         ; R1 = Address of free queue element obtained.
46         ; A system crash occurs if there are no free queue elements.
47         ;
48 004414 GETRTQ: DISABL           ; ** DISABLE **
49 004422 013701 000000G         MOV     FREIQ, R1       ; GET ADDRESS OF A FREE I/O QUEUE ELEMENT
50 004426 001413         BEQ     1$             ; SYSTEM CRASH IF NO FREE ELEMENTS AVAILABLE
51 004430 016137 000000G 000000G  MOV     Q.LINK(R1), FREIQ; REMOVE QUEUE ELEMENT FROM FREE LIST
52 004436 005337 000000G         DEC     NMFREQ         ; DEC # AVAILABLE QUEUE ELEMENTS
53 004442         ENABL           ; ** ENABLE **
54 004450 004737 004200'         CALL    ZEROQ         ; ZERO THE QUEUE ELEMENT
55 004454 000207         RETURN
56         ;
57         ; Fatal system error -- No free I/O queue elements available.

```

58
59 004456

; 1#: DIE #EM#NQE ; NO QUEUE ELEMENTS AVAILABLE

GETUMR -- Allocate Unibus map register

```

1          .SBTTL  GETUMR -- Allocate Unibus map register
2          ;-----
3          ; GETUMR is called to determine if a Unibus map register set needs to be
4          ; allocated for an I/O operation.  If unibus mapping is needed for the
5          ; operation, GETUMR attempts to find a free unibus map register set large
6          ; enough to handle the operation.  If a free UMR can be found, it is
7          ; allocated to the operation.  If no free UMR can be found, the I/O
8          ; queue request is placed on a waiting list and an exception condition
9          ; is signaled on return.
10         ;
11         ; Inputs:
12         ; R1 = Address of I/O queue entry.
13         ; R2 = Device index number of device I/O is being directed to.
14         ;
15         ; Outputs:
16         ; C-flag clear ==> A UMR was successfully allocated.
17         ; C-flag set  ==> No free UMR.  Defer I/O request.
18         ; Q.UMRX = Address of UMR descriptor block for UMR that was allocated.
19         ; Q.UMPB = Original value of Q.BUFF
20         ; Q.UMPP = Original value of Q.PAR
21         ; Q.UMVB = Number of base UMR being used by operation.
22         ;
23         ; See if this device needs unibus mapping
24         ;
25 004470 005761 0000000 GETUMR: TST      Q.UMRX(R1)      ; Have we already allocated a UMR?
26 004474 001007          BNE      9$              ; Br if yes
27 004476 105737 0000000 TSTB   UBUSMP      ; IS UNIBUS MAPPING NEEDED AT ALL?
28 004502 001404          BEQ      9$              ; BR IF NOT
29 004504 032762 0000000 0000000 BIT     #DX$DMA,DVFLAG(R2) ; IS THIS A DMA DEVICE?
30 004512 001002          BNE      11$             ; BR IF YES
31 004514 000241          9$:    CLC              ; SIGNAL SUCCESS ON RETURN
32 004516 000207          RETURN
33         ;
34         ; We must do unibus mapping for this I/O operation.
35         ;
36 004520 010246          11$:   MOV      R2,-(SP)
37 004522 010346          MOV      R3,-(SP)
38 004524 010446          MOV      R4,-(SP)
39 004526 010546          MOV      R5,-(SP)
40         ;
41         ; Determine how many words are being transferred
42         ;
43 004530 016103 0000000 MOV     Q.WCNT(R1),R3 ; GET SPECIFIED WORD COUNT
44 004534 002001          BGE     10$              ; BR IF VALUE POSITIVE
45 004536 005403          NEG     R3              ; GET POSITIVE WORD COUNT
46         ;
47         ; Find the smallest free UMR set that is large enough for this transfer
48         ;
49 004540 012704 0000000 10$:   MOV     #UMRBAS,R4 ; POINT TO BASE OF UMR DESCRIPTOR BLOCKS
50 004544          DISABL          ; ** DISABLE **
51 004552 005764 0000000 1$:    TST     UM$IOQ(R4) ; IS THIS UMR SET FREE?
52 004556 001003          BNE     5$              ; BR IF NOT
53 004560 020364 0000000 CMP     R3,UM$WDS(R4) ; IS THIS UMR SET LARGE ENOUGH?
54 004564 101425          BLOS   4$              ; BR IF YES
55 004566 062704 0000000 5$:    ADD     #UM$$SZ,R4 ; POINT TO NEXT UMR DESCRIPTOR
56 004572 020427 0000000 CMP     R4,#UMREND ; CHECKED ALL?
57 004576 103765          BLO    1$              ; BR IF NOT

```

```

58 ;
59 ; There are no free UMR register sets of adequate size.
60 ; Put I/O queue entry on list waiting for a free UMR.
61 ;
62 004600 012704 0000000 MOV #<UMRWHD-Q.LINK>,R4 ;POINT TO HEAD OF WAIT LIST
63 004604 016405 0000000 2$: MOV Q.LINK(R4),R5 ; IS THIS THE LAST ENTRY IN THE LIST?
64 004610 001402 BEQ 3$ ;BR IF YES
65 004612 010504 MOV R5,R4 ;LINK TO NEXT ENTRY
66 004614 000773 BR 2$ ;GO CHECK IT
67 004616 010164 0000000 3$: MOV R1,Q.LINK(R4) ;ADD OUR I/O QUEUE ENTRY TO TAIL OF LIST
68 004622 005061 0000000 CLR Q.LINK(R1) ;SAY WE ARE THE LAST ENTRY IN LIST
69 004626 ENABL ;** ENABLE **
70 004634 000261 SEC ;SIGNAL FAILURE
71 004636 000450 BR 8$ ;GO RETURN
72 ;
73 ; We found a free UMR. Claim it for our I/O request.
74 ;
75 004640 010164 0000000 4$: MOV R1,UM$IOQ(R4) ;SAY UMR SET IS IN USE
76 004644 ENABL ;** ENABLE **
77 004652 010461 0000000 MOV R4,Q.UMRX(R1) ;REMEMBER WHICH UMR SET IS USED BY OPERATION
78 ;
79 ; Set up information in I/O queue entry that will be used by MPPHY
80 ; routine to calculate unibus virtual address.
81 ; Q.UMPB = Original value of Q.BUFF
82 ; Q.UMPP = Original value of Q.PAR
83 ; Q.UMVB = Number of base UMR being used by transfer.
84 ;
85 004656 016105 0000000 MOV Q.BUFF(R1),R5 ;GET CURRENT BUFFER ADDRESS RELATIVE TO PAR6
86 004662 010561 0000000 MOV R5,Q.UMPB(R1) ;SAVE ORIGINAL BUFFER ADDRESS
87 004666 016103 0000000 MOV Q.PAR(R1),R3 ;GET PAR6 BASE ADDRESS
88 004672 010361 0000000 MOV R3,Q.UMPP(R1) ;Save original value of Q.PAR
89 004676 116461 0000000 0000000 MOVB UM$UMR(R4),Q.UMVB(R1) ;BASE UMR #
90 ;
91 ; Set up mapping base value in unibus map register.
92 ;
93 004704 162705 0000000 SUB #VPAR6,R5 ;REMOVE PAR6 BIAS FROM BUFFER ADDRESS
94 004710 005002 CLR R2 ;SET UP HIGH-ORDER REGISTER (R2-R3)
95 004712 073227 0000006 ASHC #6,R2 ;SHIFT Q.PAR VALUE 6 PLACES (R2-R3)
96 004716 060503 ADD R5,R3 ;ADD IN BUFFER BASE OFFSET
97 004720 005502 ADC R2 ;PROPOGATE CARRY
98 004722 116405 0000000 MOVB UM$UMR(R4),R5 ;GET UNIBUS MAP REGISTER #
99 004726 072527 0000002 ASH #2,R5 ;*4 BYTES PER REGISTER
100 004732 062705 0000000 ADD #UMRADR,R5 ;GET ADDRESS OF UNIBUS MAP REGISTER
101 004736 116400 0000000 MOVB UM$NMR(R4),R0 ;GET # MAP REGISTERS USED IN THIS SET
102 004742 010325 6$: MOV R3,(R5)+ ;SET LOW ORDER VALUE
103 004744 010225 MOV R2,(R5)+ ;SET HIGH ORDER VALUE
104 004746 062703 020000 ADD #8192.,R3 ;ADVANCE ADDRESS VALUE
105 004752 005502 ADC R2 ;PROPOGATE CARRY
106 004754 077006 SOB R0,6$ ;LOOP IF MORE REGISTERS IN SET
107 ;
108 ; Finished
109 ;
110 004756 000241 7$: CLC ;SIGNAL SUCCESS ON RETURN
111 004760 012605 8$: MOV (SP)+,R5
112 004762 012604 MOV (SP)+,R4
113 004764 012603 MOV (SP)+,R3
114 004766 012602 MOV (SP)+,R2

```

115 004770 000207

RETURN

```

1          .SBTTL  FREUMR -- Free a Unibus map register
2          ;-----
3          ; FREUMR is called to free a unibus map register set associated with an
4          ; I/O request.
5          ; If there is an I/O request waiting for a free UMR set and the set being
6          ; freed is large enough for the waiting request, the request is removed
7          ; from the wait list and is started.
8          ;
9          ; Inputs:
10         ; R1 = Address of I/O queue element that is being freed.
11         ;
12         ; See if there is a UMR associated with this request.
13         ;
14 004772 005761 0000000 FREUMR: TST      Q.UMRX(R1)      ; IS THERE A UMR ASSOCIATED WITH THIS REQUEST?
15 004776 001464          BEQ      9$              ; BR IF NOT
16         ;
17         ; This I/O queue does have an associated UMR.
18         ;
19         MOV      R1,-(SP)
20         MOV      R2,-(SP)
21         MOV      R3,-(SP)
22         MOV      R4,-(SP)
23 005010 010546          MOV      R5,-(SP)
24         ;
25         ; Free the UMR
26         ;
27 005012 016104 0000000          MOV      Q.UMRX(R1),R4      ; GET ADDRESS OF UMR DESCRIPTOR BLOCK
28 005016 005064 0000000          CLR      UM$IOQ(R4)      ; SAY THE UMR IS FREE
29 005022 005061 0000000          CLR      Q.UMRX(R1)      ; SAY NO UMR ASSOCIATED WITH I/O REQUEST
30         ;
31         ; See if there is any I/O queue request waiting for a free UMR
32         ; that could be satisfied with the UMR that is being freed.
33         ;
34 005026 016405 0000000          MOV      UM$WDS(R4),R5      ; GET SIZE OF UMR SET THAT IS BEING FREED
35 005032          DISABL          ; ** DISABLE **
36 005040 012702 0000000          MOV      #<UMRWHD-Q.LINK>,R2 ; POINT TO HEAD OF WAIT LIST
37 005044 000406          BR      3$
38 005046 016200 0000000 1$: MOV      Q.WCNT(R2),R0      ; GET # WORDS NEEDED BY THIS I/O REQUEST
39 005052 002001          BGE      2$
40 005054 005400          NEG      R0              ; GET POSITIVE VALUE
41 005056 020005 2$: CMP      R0,R5          ; IS THE FREED UMR LARGE ENOUGH?
42 005060 101410          BLOS     4$              ; BR IF YES
43 005062 010203 3$: MOV      R2,R3          ; SAVE ADDRESS OF THIS I/O REQUEST
44 005064 016202 0000000          MOV      Q.LINK(R2),R2      ; GET ADDRESS OF NEXT WAITING I/O REQUEST
45 005070 001366          BNE      1$              ; BR IF THERE IS ANOTHER
46 005072          ENABL          ; ** ENABLE **
47 005100 000416          BR      8$              ; THERE ARE NO WAITING ENTRIES TO BE STARTED
48         ;
49         ; We found a waiting I/O queue entry that could be started by
50         ; the UMR that has just been freed/
51         ; Remove I/O queue entry from wait list and try to start it.
52         ;
53 005102 016263 0000000 0000000 4$: MOV      Q.LINK(R2),Q.LINK(R3) ; REMOVE I/O QUEUE ENTRY FROM WAIT LIST
54 005110          ENABL          ; ** ENABLE **
55 005116 010201          MOV      R2,R1          ; GET ADDRESS OF I/O QUEUE ENTRY TO R1
56 005120 116102 0000000          MOVB     Q.DEVX(R1),R2      ; Get device index number
57 005124 004737 004470'          CALL     GETUMR          ; We should be able to get a free UMR now

```

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

```

1          .SBTTL  GETUCH -- Get character from user's buffer
2          ;-----
3          ; GETUCH is called to get a character from the user's buffer.
4          ;
5          ; Inputs:
6          ;   R3 = Address of byte to be fetched.
7          ;
8          ; Outputs:
9          ;   R0 = Byte we got.
10         ;   R3 = Incremented.
11         ;
12 005152 032703 000001  GETUCH: BIT    #1,R3      ;GETTING ODD OR EVEN BYTE?
13 005156 001004          BNE    1$          ;BR IF ODD
14         ;
15         ; Byte is on a word boundary
16         ;
17 005160 106513          MFPD   (R3)        ;GET WORD CONTAINING BYTE
18 005162 005203          INC    R3          ;ADVANCE R3
19 005164 112600          MOVB  (SP)+,R0     ;RETURN BYTE IN R0
20 005166 000207          RETURN
21         ;
22         ; Byte address is odd
23         ;
24 005170 005303 1$:     DEC    R3          ;POINT TO START OF WORD WITH BYTE
25 005172 106523          MFPD   (R3)+      ;GET WORD WITH BYTE
26 005174 012600          MOV   (SP)+,R0
27 005176 105000          CLRB  R0          ;CLEAR LOW-ORDER BYTE
28 005200 000300          SWAB  R0          ;MOVE HIGH-ORDER BYTE TO LOW-ORDER
29 005202 000207          RETURN
  
```

```

1          .SBTTL  PUTUCH -- Move byte to user's buffer
2          ;-----
3          ; PUTUCH is called to move a byte into the user's buffer.
4          ;
5          ; Inputs:
6          ;   RO = Byte to be stored.
7          ;   R3 = Address where byte is to be stored.
8          ;
9          ; Outputs:
10         ;   R3 = Incremented.
11         ;
12 005204 032703 000001  PUTUCH: BIT    #1,R3      ; IS BYTE ADDRESS ODD OR EVEN?
13 005210 001005          BNE    1$          ; BR IF ODD
14         ;
15         ;   Byte address is even.
16         ;
17 005212 106513          MFPD   (R3)        ; GET WORD WHERE BYTE IS TO BE STORED
18 005214 110016          MOVB   RO,(SP)    ; PUT IN OUR BYTE
19 005216 106613          MTPD   (R3)        ; REPLACE WORD
20 005220 005203          INC    R3         ; INCREMENT R3
21 005222 000406          BR     2$
22         ;
23         ;   Byte address is odd.
24         ;
25 005224 005303 1$:    DEC    R3          ; POINT TO WORD WITH BYTE
26 005226 106513          MFPD   (R3)        ; GET WORD WITH BYTE
27 005230 000316          SWAB   (SP)        ; GET BYTE TO LOW-ORDER
28 005232 110016          MOVB   RO,(SP)    ; PUT IN NEW BYTE
29 005234 000316          SWAB   (SP)        ; REPOSITION
30 005236 106623          MTPD   (R3)+     ; STORE BACK INTO USER'S BUFFER
31 005240 000207 2$:    RETURN

```

```

1          .SBTTL  SYBFAD -- Set system buffer address in I/O queue entry
2          ;-----
3          ; SYBFAD is called to convert a buffer address in system space into an
4          ; address that will be mapped through PAR 6.  This address is stored
5          ; into a specified I/O queue element.
6          ; The buffer address may be in low physical memory (PAR 0 through PAR 4),
7          ; in a mapped system overlay region (PAR 5), or in the user context
8          ; block (PAR 6).
9          ;
10         ; Inputs:
11         ;   RO = Physical address to be converted.
12         ;   R1 = Address of I/O queue entry to be set up.
13         ;
14         ; Outputs:
15         ;   Q.PAR(R1) & Q.BUFF(R1) are set up.
16         ;
17 005242 010046 SYBFAD: MOV     RO,-(SP)
18 005244 010246      MOV     R2,-(SP)
19         ;
20         ; Determine which region buffer is in.
21         ;
22 005246 020027 0000000  CMP     RO,#CXTBAS      ;Is buffer in job context block?
23 005252 103407      BLD     2$              ;Br if not in context block
24         ;
25         ; Buffer is in job context block
26         ;
27 005254 113702 0000000  MOVB   CORUSR,R2      ;Get current job index number
28 005260 016202 0000000  MOV    LBASE(R2),R2   ;Get base 256-word block number of context blk
29 005264 072227 0000003  ASH   #3,R2          ;Convert to 32-word block number
30 005270 000421      BR     9$              ;(Note: virtual address in RO already in PAR 6
31         ;
32         ; See if we are accessing a buffer in a system mapped region
33         ;
34 005272 020027 0000000 2$:    CMP     RO,#VPAR5      ;Is buffer in a system mapped region?
35 005276 103405      BLD     12$             ;Br if not
36         ;
37         ; Buffer is in a system mapped region
38         ;
39 005300 013702 0000000  MOV    @#KPAR5,R2     ;Get current mapping for kernel PAR 5
40 005304 062700 0200000  ADD    #20000,R0      ;Bias virtual address to PAR 6 region
41 005310 000411      BR     9$
42         ;
43         ; We are accessing a buffer stored in low physical memory (PAR 0-4)
44         ;
45 005312 010002 12$:    MOV    RO,R2          ;Get actual address
46 005314 000241      CLC                    ;Convert to page number
47 005316 006002      ROR     R2
48 005320 072227 177773  ASH   #-5,R2
49 005324 042700 177700  BIC   #^C77,R0       ;Get byte-in-page to RO
50 005330 062700 0000000  ADD    #VPAR6,R0     ;Map through PAR 6
51         ;
52         ; Store values into I/O queue element
53         ;
54 005334 010261 0000000 9$:    MOV    R2,Q.PAR(R1) ;Page relocation bias
55 005340 010061 0000000      MOV    RO,Q.BUFF(R1)  ;Buffer address
56         ;
57         ; Finished

```

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

```

1          .SBTTL  UACHKx -- Check validity of user address
2          ;-----
3          ; UACHKW and UACHKB are called to determine if an emt address is
4          ; within the valid region of the current job.
5          ; UACHKW also checks to see that the address is on a word boundary.
6          ;
7          ; Inputs:
8          ;   RO = Address to be checked (preserved)
9          ;   EMTPS = PS saved by EMT
10         ;   UHIMEM = Top address of job region
11         ;
12         ; Outputs:
13         ;   C-flag set on return if address is invalid.
14         ;
15         ; Check a word address
16         ;
17 005352 032700 000001 UACHKW: BIT    #1,RO      ; IS ADDRESS EVEN?
18 005356 001402      BEQ    UACHKB      ; BR IF YES
19 005360 000261      SEC                ; SIGNAL ERROR
20 005362 000207      RETURN
21         ;
22         ; Check byte address
23         ;
24         ; See if address is in normal job region
25 005364 020037 0000000 UACHKB: CMP    RO,UHIMEM  ; IS ADDRESS IN NORMAL JOB REGION
26 005370 101404      BLOS   4$         ; BR IF YES
27         ;
28         ; Address is not in normal job region.
29         ; Allow kernel-mode emt's to access all of memory
30         ;
31 005372 032737 0000000 0000000 BIT    #UMODE,EMTPS  ; IS THIS A KERNEL MODE EMT?
32 005400 001002      BNE    5$         ; BR IF NOT
33 005402 000241      4$: CLC                ; SIGNAL SUCCESSFUL RETURN
34 005404 000207      RETURN
35         ;
36         ; This is a user-mode emt accessing a region above the normal top of job
37         ;
38 005406 010146      5$: MOV    R1,-(SP)
39 005410 010246      MOV    R2,-(SP)
40 005412 113701 0000000 MOVB  CORUSR,R1      ; GET CURRENT JOB INDEX NUMBER
41         ;
42         ; Allow KMON to access job context area
43         ;
44 005416 032761 0000000 0000000 BIT    #$INKMN,LSW4(R1); IS KMON RUNNING?
45 005424 001403      BEQ    2$         ; BR IF NOT
46 005426 020027 0000000 CMP    RO,#CXTEND    ; IS ADDRESS IN JOB CONTEXT REGION?
47 005432 103426      BLO    1$         ; BR IF YES
48         ;
49         ; See if there are any associated shared run-time systems
50         ;
51 005434 010001      2$: MOV    RO,R1      ; GET ADDRESS BEING CHECKED
52 005436 072127 177764  ASH    #-12.,R1    ; CONVERT ADDRESS TO 2 * PAR REGION #
53 005442 042701 177761  BIC    #^C16.,R1
54 005446 016102 0000000 MOV    RPDR(R1),R2  ; IS THAT REGION MAPPED TO SHARED RUN-TIME?
55 005452 001414      BEQ    3$         ; BR IF NOT
56 005454 000302      SWAB   R2          ; GET PAGE LENGTH TO LOW-ORDER BYTE
57 005456 042702 177600  BIC    #^C177.,R2  ; GET # 64-BYTE BLOCKS USED IN PAR REGION

```

UACHKx -- Check validity of user address

```

58 005462 005202          INC      R2          ;# IS OFFSET BY 1
59 005464 072227 000006  ASH      #6,R2       ;CONVERT TO # BYTES
60 005470 005302          DEC      R2          ;AVOID ADDRESS OVERFLOW AT TOP OF MEMORY
61 005472 072127 000014  ASH      #12,R1      ;GET ADDRESS OF BASE OF PAR REGION
62 005476 060201          ADD      R2,R1       ;GET ADDRESS OF LAST BYTE IN PAR REGION
63 005500 020001          CMP      R0,R1       ;IS OUR ADDRESS IN MAPPED REGION?
64 005502 101402          BLDS    1#          ;BR IF YES
65                          ;
66                          ; Address is invalid
67                          ;
68 005504 000261          3#:     SEC          ;SIGNAL ERROR ON RETURN
69 005506 000401          BR      9#
70                          ;
71                          ; Address is valid
72                          ;
73 005510 000241          1#:     CLC          ;SIGNAL SUCCESS ON RETURN
74                          ;
75                          ; Finished
76                          ;
77 005512 012602          9#:     MOV      (SP)+,R2
78 005514 012601          MOV      (SP)+,R1
79 005516 000207          RETURN
80
81                          .SBTTL  VALADx -- Validate user address
82                          -----
83                          ; VALADW and VALADB are called to validate an address provided as
84                          ; an EMT argument.  If the address is invalid, these routines do not
85                          ; return but rather produce a fatal EMT error.
86                          ; VALADW checks to see that the address is even.
87                          ;
88                          ; Inputs:
89                          ; R0 = Address to be checked (preserved)
90                          ; EMTPS = PS saved by EMT
91                          ;
92                          ; Outputs:
93                          ; A fata EMT error is produced if the address is invalid.
94                          ;
95 005520 004737 005352'  VALADW: CALL    UACHKW          ;CHECK WORD ADDRESS
96 005524 103405          BCS     BADAD          ;BR IF INVALID
97 005526 000207          RETURN          ;ADDRESS IS OK
98
99 005530 004737 005364'  VALADB: CALL    UACHKB          ;CHECK BYTE ADDRESS
100 005534 103401          BCS     BADAD          ;BR IF INVALID
101 005536 000207          RETURN
102
103                          ; Invalid address -- Give fatal emt error
104                          ;
105 005540 012700 177766  BADAD:  MOV      #-12,R0       ;GIVE FATAL EMT ERROR
106 005544 000137 004036'  JMP     SETERR

```

```

1          .SBTTL IOWAIT -- Wait for I/O to finish
2          ;-----
3          ; IOWAIT is called to suspend user execution until all I/O on the
4          ; current channel is finished.
5          ;
6          ; Inputs:
7          ; CHNADR = Address of current channel we are waiting on.
8          ;
9 005550 010346 IOWAIT: MOV      R3,-(SP)
10 005552 013703 0000000 MOV      CHNADR,R3      ;GET ADDRESS OF CHANNEL
11 005556          1$:  DISABL          ;* DISABLE INTERRUPTS
12 005564 105763 0000000 TSTB    C.NUMQ(R3)      ;* ANY ACTIVE I/O REQUESTS ON CHANNEL?
13 005570 001407          BEQ      2$          ;* BR IF NOT -- ALL I/O IS FINISHED
14 005572 012700 0000000 MOV      #S$IOWT,R0     ;* SAY WE ARE WAITING FOR I/O TO FINISH
15 005576 004737 0000000 CALL    QNSPND          ;ENQUEUE FOR I/O WAIT (ENABLE)
16 005602 004737 0000000 CALL    CHKABT          ;WERE WE ABORTED WHILE ASLEEP?
17 005606 000763          BR      1$          ;GO TRY AGAIN
18 005610          2$:  ENABL
19 005616 012603          MOV      (SP)+,R3
20 005620 000207          RETURN
21
22          .SBTTL IOSTOP -- Wait for all of job's I/O to finish
23          ;-----
24          ; IOSTOP suspends execution of the current job until all of its
25          ; I/O is completed.
26          ;
27 005622 010146 IOSTOP: MOV      R1,-(SP)
28 005624 113701 0000000 MOVB   CORUSR,R1      ;GET JOB INDEX #
29 005630          1$:  DISABL          ;*** DISABLE ***
30 005636 105761 0000000 TSTB    LIOCNT(R1)     ;IS THERE ANY I/O ACTIVE FOR JOB?
31 005642 001405          BEQ      2$          ;BR IF NOT
32 005644 012700 0000000 MOV      #S$IOWT,R0     ;SUSPEND JOB TILL I/O FINISHES
33 005650 004737 0000000 CALL    QNSPND
34 005654 000765          BR      1$          ;NOW GO SEE IF IT HAS COMPLETED
35          ;
36          ; All I/O is finished
37          ;
38 005656          2$:  ENABL          ;** ENABLE **
39 005664 012601          MOV      (SP)+,R1
40 005666 000207          RETURN
41
42          .SBTTL IOHALT -- Abort all I/O for a job
43          ;-----
44          ; Process .ABTIO EMT
45          ;
46 005670 005737 0000000 ABTIO: TST      IOABFL          ;CHECK TO SEE IF IO ABORT WAS CHOSEN
47 005674 001752          BEQ      IOSTOP          ;BR IF IO RUNDOWN WAS CHOSEN
48 005676 010246          MOV      R2,-(SP)          ;Save registers
49 005700 010346          MOV      R3,-(SP)
50 005702 013703 0000000 MOV      CHNADR,R3      ;Get address of channel status word
51 005706 032763 0000000 0000000 BIT      #CS#OPN,C.CSW(R3);See if channel is open
52 005714 001406          BEQ      1$          ;Br if channel is not open
53 005716 016302 0000000 MOV      C.CSW(R3),R2   ;Get channel status word
54 005722 042702 177701 BIC     #^C76,R2        ;Mask channel device index number
55 005726 004737 006006' CALL    DVSTOP          ;Stop I/O to this device
56 005732 012603          1$:  MOV      (SP)+,R3          ;Restore registers
57 005734 012602          MOV      (SP)+,R2

```

IOHALT -- Abort all I/O for a job

```

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

```

```

1          .SBTTL  DVSTOP -- Abort I/O for a device & job
2
3          ;-----
4          ; DVSTOP is called to abort all I/O requests for a particular job
5          ; that are pending for a particular device.
6          ; The abort entry point for the device handler is called if the
7          ; current queue entry belongs to the specified job or if the abort
8          ; flag is set for the handler.
9          ;
10         ; Inputs:
11         ; R1 = Index number of job whose I/O is to be aborted.
12         ; R2 = Device index number of device to be serviced.
13         ; R3 = Address of channel status word (for .ABTIO) or zero
14         ;
15         ;-----
16         ;
17         ;
18         ;
19         ;
20         ;
21         ;
22         ;
23         ;
24         ;
25         ;
26         ;
27         ;
28         ;
29         ;
30         ;
31         ;
32         ;
33         ;
34         ;
35         ;
36         ;
37         ;
38         ;
39         ;
40         ;
41         ;
42         ;
43         ;
44         ;
45         ;
46         ;
47         ;
48         ;
49         ;
50         ;
51         ;
52         ;
53         ;
54         ;
55         ;
56         ;
57         ;

```

14	006006	010346			DVSTOP: MOV	R3,-(SP)	
15	006010	010446				MOV	R4,-(SP)
16	006012	010546				MOV	R5,-(SP)
17	006014	013746	0000000			MOV	@#KPAR5,-(SP) ; Save PAR 5 in case we map to a handler
21	006020	026227	0000000	0000004		CMP	HANENT(R2),#4 ; Is this a real device?
22	006026	101532				BLOS	7\$; Br if not
26	006030	012700	006314'			MOV	#7\$,R0 ; Set exit address for system state
27	006034	004737	0000000			CALL	ENSYS ; Enter system state
28	006040	0000000				.WORD	FP\$IDA ; Fork processing priority
29	006042	010305				MOV	R3,R5 ; Move channel status word
33	006044	016203	0000000			MOV	HANPAR(R2),R3 ; Is this a mapped handler?
34	006050	001402				BEQ	8\$; Br if not
35	006052	010337	0000000			MOV	R3,@#KPAR5 ; Map PAR 5 to the handler
36	006056	016203	0000000		B\$:	MOV	HANENT(R2),R3 ; Get pointer to LQE cell in handler
42	006062	012763	100000	177776		MOV	#100000,-2(R3) ; Set handler-hold flag
43	006070	005723				TST	(R3)+ ; Point to CQE cell in handler
48	006072	032762	0000000	0000000		BIT	#DS\$ABT,DVSTAT(R2); Is abort flag set for handler?
49	006100	001015				BNE	1\$; Br if yes
50	006102	011304				MOV	(R3),R4 ; Get pointer to current queue element for hand
51	006104	001430				BEQ	2\$; Br if none
52	006106	116400	0000000			MOVB	Q.JOB-Q.BLKN(R4),R0 ; Get job # from queue element
53	006112	006300				ASL	R0 ; Convert to job index number
54	006114	020001				CMP	R0,R1 ; Does this element belong to job being aborted
55	006116	001023				BNE	2\$; Br if not
56	006120	016400	0000000			MOV	Q.CSW-Q.BLKN(R4),R0; Get address of Channel Status Word
57	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 0000006 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 0000000 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 0000000 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 0000000 0000000 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$

```

```
115 ;  
116 ; Finished aborting I/O for this device  
117 ;  
118 006314 012637 0000000 7#: MOV (SP)+, @#KPAR5 ; Restore PAR 5 mapping  
119 006320 012605 MOV (SP)+, R5  
120 006322 012604 MOV (SP)+, R4  
121 006324 012603 MOV (SP)+, R3  
122 006326 000207 RETURN
```

```

1          .SBTTL  IOTIMR -- Process handler timeout requests
2          ;-----
3          ; IOTIMR is the routine called from device handlers to process I/O
4          ; timer requests.  Normally the .TIMIO and .CTIMIO macros are used
5          ; to call IOTIMR.
6          ;
7          ; Inputs:
8          ; R5 = Address of argument list generated by .TIMIO or .CTIMIO macros
9          ;
10         IOTIMR: MOV     R0,-(SP)
11             MOV     R1,-(SP)
12             MOV     R2,-(SP)
13             MOV     R3,-(SP)
14         ;
15         ; Determine if we are starting or canceling a timer request
16         ;
17         MOV     R5,R2          ;GET PTR TO CELL WITH OFFSET TO TIMER BLOCK
18         ADD     (R5)+,R2      ;GET ADDRESS OF TIMER BLOCK IN HANDLER
19         TST     (R5)+        ;IS THIS A .TIMIO OR .CTIMIO REQUEST?
20         BNE     IOTIMC       ;BR IF REQUEST TO CANCEL TIMER
21         ;
22         ; This is a request to start a timer.
23         ; Get a completion request queue entry and set up values in it.
24         ;
25         TST     IT$RTN(R2)    ;Make sure there is a specified compl routine
26         BNE     4$           ;Br if compl routine address specified
27         CMP     (R5)+,(R5)+  ;Skip over high & low order time words
28         SEC                     ;Signal error on return
29         BR      IOTMXT       ;Return
30         4$: CALL    GETRTQ    ;Get a queue element (addr returned in R1)
31         MOV     R1,IT$LNK(R2) ;Save ptr to queue element in handler block
32         BISB   #<QF$IOT!QF$SCR>,CQ$FLG(R1);Set flag saying .TIMIO request
33         MOV     (R5)+,CQ$HDT(R1);Set high-order time value
34         MOV     (R5)+,CQ$LOT(R1);Set low-order time value
35         MOV     IT$RTN(R2),CQ$RTN(R1);Set address of routine to call
36         MOV     IT$SEQ(R2),CQ$RO(R1);Set sequence # to be passed in R0
37         MOV     R2,CQ$R1(R1)  ;Set address of cell to be cleared on call
38         ADD     #IT$RTN,CQ$R1(R1)
39         MOV     @#KPAR5,CQ$PA5(R1);Set PAR 5 mapping for compl routine
40         MOV     IT$JOB(R2),R0  ;Get # of job to synch with
41         BIC     #^C377,R0     ;Clear all but job number
42         ASL     R0             ;Convert to job index number
43         MOVB   R0,CQ$JOB(R1)  ;Set job number in compl queue entry
44         ;
45         ; Link timer element into list of active requests
46         ;
47         DISABL                    ;** DISABLE **
48         MOV     MRKTHD,CQ$LNK(R1);Put new element at head of timer list
49         MOV     R1,MRKTHD
50         ENABL                    ;** ENABLE **
51         BR      IOTMCC         ;FINISHED
52         ;
53         ; Cancel a timer request
54         ;
55         IOTIMC: MOV     IT$LNK(R2),R1 ;Get address of queue element used for request
56             BEQ     3$         ;Invalid if zero
57             DISABL                    ;** DISABLE **

```

```
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
```

```

1          .SBTTL  CANIOT -- Cancel all .TIMIO requests for a job
2          ;-----
3          ; CANIOT is called to cancel all pending .TIMIO requests for a job.
4          ;
5          ; Inputs:
6          ; R1 = Index number of job.
7          ;
8 006626 010246 CANIOT: MOV     R2,-(SP)
9 006630 010446          MOV     R4,-(SP)
10 006632 013746 000000G          MOV     @#KPAR5,-(SP) ; Save current PAR 5 mapping
11          ;
12          ; Search through pending queue looking for requests belonging to this job
13          ;
14 006636 012704 000000G 1$:     MOV     #MRKTHD-CQ$LNK,R4 ; Get dummy pointer to queue head
15 006642          DISABL          ; ** Disable interrupts **
16 006650 016402 000000G 2$:     MOV     CQ$LNK(R4),R2 ;;; Get address of next pending entry
17 006654 001434          BEQ     4$ ;;; Br if hit end of list
18 006656 132762 000000G 000000G BITB   #QF$IOT,CQ$FLG(R2);;; Is this entry for a .TIMIO?
19 006664 001403          BEQ     3$ ;;; Br if not
20 006666 120162 000000G          CMPB   R1,CQ$JOB(R2) ;;; Is this entry for our job?
21 006672 001402          BEQ     5$ ;;; Br if yes
22 006674 010204 3$:     MOV     R2,R4 ;;; Link to next entry
23 006676 000764          BR      2$ ;;; Go check it
24          ;
25          ; We found an entry for this job
26          ;
27 006700 016264 000000G 000000G 5$:     MOV     CQ$LNK(R2),CQ$LNK(R4);;; Remove entry from linked list
28 006706          ENABL          ; ** Enable interrupts **
29 006714 016200 000000G          MOV     CQ$R1(R2),R0 ; Get addr of cell in handler to be cleared
30 006720 001404          BEQ     6$ ; Br if nothing to clear
31 006722 016237 000000G 000000G          MOV     CQ$PA5(R2),@#KPAR5; Set up PAR 5 mapping for the handler
32 006730 005010          CLR     (R0) ; Clear the cell in the handler
33 006732 010104 6$:     MOV     R1,R4 ; Save the job index number
34 006734 010201          MOV     R2,R1 ; Get address of queue element to R1
35 006736 004737 004222'          CALL   QFREE ; Free the queue element
36 006742 010401          MOV     R4,R1 ; Get back job index number
37 006744 000734          BR      1$ ; See if there are more entries to free
38          ;
39          ; Finished
40          ;
41 006746 4$:     ENABL          ; ** Enable interrupts **
42 006754 012637 000000G 9$:     MOV     (SP)+,@#KPAR5 ; Restore PAR 5 mapping
43 006760 012604          MOV     (SP)+,R4
44 006762 012602          MOV     (SP)+,R2
45 006764 000207          RETURN
46          ;-----
47          ;
48          ; Dummy subroutine called by handlers trying to do error logging.
49          ; Return immediately.
50          ;
51 006766 000207 ERRLOG: RETURN

```

```

1          .SBTTL  SETHAN -- Update running copy of device handler
2          ;-----
3          ; SETHAN is the EMT used by kmon to update the running copy of a device
4          ; handler after a SET command has been done.
5          ; If the handler is active, an error return occurs.
6          ;
7 006770 013704 0000040 SETHAN: MOV     EMTBLK+4,R4      ;GET DEVICE TABLE INDEX FOR HANDLER
8 006774 016402 0000000      MOV     HANSIZ(R4),R2      ;GET SIZE OF HANDLER
9 007000 162702 000022      SUB     #18.,R2          ;DON'T MODIFY LAST 9 WORDS OF HANDLER (.DREND TABLE)
10 007004 020227 001000      CMP     R2,#1000        ;MAX CODE MODIFIED IS 1000
11 007010 101402          BLOS   1$              ;BR IF HANDLER IS SMALLER
12 007012 012702 001000      MOV     #1000,R2
13 007016 006202          1$:   ASR     R2          ; CONVERT # BYTES TO # WORDS
14 007020 016401 0000000      MOV     HANENT(R4),R1   ; GET ADDRESS OF HANDLER ENTRY POINT
15 007024 016400 0000000      MOV     HANPAR(R4),R0   ; IS THIS A MAPPED HANDLER?
16 007030 001402          BEQ     4$              ; BR IF NOT
17 007032 010037 0000000      MOV     R0,@#KPAR5     ; MAP PAR 5 TO THE HANDLER
18 007036 013703 0000060          4$:   MOV     EMTBLK+6,R3   ; GET ADDRESS OF BUFFER IN KMON WITH NEW CODE
19 007042          DISABL          ; *** DISABLE ***
20 007050 005711          TST     (R1)           ; IS THE HANDLER ACTIVE NOW?
21 007052 001012          BNE     3$              ; BR IF YES -- GIVE ERROR RETURN
22 007054 162701 000006      SUB     #6,R1          ; POINT TO START OF HANDLER IN MEMORY
23 007060 106523          2$:   MFPD   (R3)+       ; GET WORD FROM KMON BUFFER
24 007062 012621          MOV     (SP)+,(R1)+    ; MOVE OVER OLD HANDLER CODE
25 007064 077203          SOB    R2,2$          ; MOVE ALL OF NEW CODE
26 007066          ENABL          ; ** ENABLE **
27 007074 000137 001110'      JMP     EMTXIT         ; SUCCESSFUL EXIT
28          ;
29          ; Error -- Handler is active now.
30          ;
31 007100          3$:   ENABL          ; ** ENABLE **
32 007106 005000          CLR     R0            ; RETURN ERROR CODE 0
33 007110 000137 004036'      JMP     SETERR        ; ERROR EXIT
    
```

```

1          .SBTTL  MIOMWT -- Mapped I/O move data to system buffer
2          ;-----
3          ; MIOMWT is called when doing a write to a device that requires its I/O
4          ; to be mapped through a system buffer.
5          ; MIOMWT moves the data from the user's buffer into a system buffer.
6          ;
7          ; Inputs:
8          ; R5 = Address of mapped I/O control block.
9          ;
10         MIOMWT: MOV     R2,-(SP)
11         MOV     R4,-(SP)
12         MOV     R5,-(SP)
13         MOV     R5,R2          ;Get address of mapped I/O control block
14         ;
15         ; Enter system state so that we can use PAR5 and PAR6 to map to the system
16         ; and user buffers.
17         ;
18         MOV     #1$,R0        ;Go to 1$ on exit from system state
19         CALL    ENSYS        ;Enter system state
20         .WORD   FP$MOV       ;Fork priority
21         ;
22         ; We are now running in system state on the interrupt stack.
23         ; Kernel PAR5 and PAR6 are free.
24         ; Set up buffer pointers.
25         ;
26         MOV     MI$UBP(R2),@#KPAR5 ;Set up PAR base for user's buffer
27         MOV     MI$SBP(R2),@#KPAR6 ;Set up PAR base for system buffer
28         MOV     #VPAR5,R4      ;Get virtual address base for user's buffer
29         ADD     MI$UBO(R2),R4   ;Add offset within 64-byte block region
30         MOV     #VPAR6,R5      ;Get virtual address base for system buffer
31         ;
32         ; Get number of words to move and do the transfer
33         ;
34         MOV     MI$CWC(R2),R0   ;Get number of words to move
35         ASR     R0              ;Get number of doublewords
36         BEQ     3$             ;Br if less than 2 words to move
37         MOV     (R4)+,(R5)+    ;Move a word
38         MOV     (R4)+,(R5)+    ;Move a second word
39         SOB     R0,2$         ;Loop if more doublewords to move
40         BCC     4$             ;Br if no odd word at end to move
41         MOV     (R4),(R5)      ;Move the last word
42         ;
43         ; Finished moving data.
44         ; Exit from system state (go to 1$)
45         ;
46         4$:  RETURN           ;Exit from system state
47         ;
48         ; Finished
49         ;
50         1$:  MOV     (SP)+,R5
51         MOV     (SP)+,R4
52         MOV     (SP)+,R2
53         RETURN

```

```

1          .SBTTL MIOMRD -- Mapped I/O move data to user's buffer
2          ;-----
3          ; MIOMRD is called when doing a read from a device that requires its I/O
4          ; to be mapped through a system buffer.
5          ; MIOMRD moves the data from the system buffer into the user's buffer.
6          ;
7          ; Inputs:
8          ; R5 = Address of mapped I/O control block.
9          ;
10         MIOMRD: MOV     R2, -(SP)
11         MOV     R4, -(SP)
12         MOV     R5, -(SP)
13         MOV     R5, R2          ;Get address of mapped I/O control block
14         ;
15         ; Enter system state so that we can use PAR5 and PAR6 to map to the system
16         ; and user buffers.
17         ;
18         MOV     #1$, R0        ;Go to 1$ on exit from system state
19         CALL    ENSYS         ;Enter system state
20         .WORD   FP$MOV        ;Fork priority
21         ;
22         ; We are now running in system state on the interrupt stack.
23         ; Kernel PAR5 and PAR6 are free.
24         ; Set up buffer pointers.
25         ;
26         MOV     MI$UBP(R2), @KPAR5 ;Set up PAR base for user's buffer
27         MOV     MI$SBP(R2), @KPAR6 ;Set up PAR base for system buffer
28         MOV     #VPAR5, R4      ;Get virtual address base for user's buffer
29         ADD     MI$UBO(R2), R4   ;Add offset within 64-byte block region
30         MOV     #VPAR6, R5      ;Get virtual address base for system buffer
31         ;
32         ; Get number of words to move and do the transfer
33         ;
34         MOV     MI$CWC(R2), R0   ;Get number of words to move
35         ASR     R0              ;Get number of doublewords
36         BEQ     3$             ;Br if less than 2 words to move
37         MOV     (R5)+, (R4)+    ;Move a word
38         MOV     (R5)+, (R4)+    ;Move a second word
39         SOB     R0, 2$         ;Loop if more doublewords to move
40         BCC     4$             ;Br if no odd word at end to move
41         MOV     (R5), (R4)      ;Move the last word
42         ;
43         ; Finished moving data.
44         ; Exit from system state (go to 1$)
45         ;
46         4$: RETURN             ;Exit from system state
47         ;
48         ; Finished
49         ;
50         1$: MOV     (SP)+, R5
51         MOV     (SP)+, R4
52         MOV     (SP)+, R2
53         RETURN

```

```
1 .SBTTL HANXIT -- Exit from a mapped handler
2 ;-----
3 ; HANXIT is jumped to when a mapped handler performs a RTI to exit
4 ; from an interrupt.
5 ;
6 007330 HANXIT:
7 ;
8 ; Restore PAR 5 mapping
9 ;
10 007330 012637 0000000 MOV (SP)+, @#KPAR5 ; Restore PAR 5 mapping
11 ;
12 ; Finished
13 ;
14 007334 000002 RTI ; Do the real interrupt exit
```

- Device handler mapping routines -

```

1          .SBTTL - Device handler mapping routines -
2          ;-----
3          ; The following routines provide XM support for device handlers.
4          ; Kernel-mode PAR 6 is used to access the user's data area.
5          ;
6          .SBTTL MPPHY -- Convert mapped address to physical address
7          ;-----
8          ; Convert a mapped address to a physical address.
9          ;
10         ; Inputs:
11         ; R5 = Address of Q.BUFF cell in I/O queue entry.
12         ; Q.BUFF = Address relative to Q.PAR
13         ; Q.PAR = Address offset.
14         ; Q.UMRX = 0==>No Unibus mapping needed; non-zero==>Do unibus mapping
15         ; Q.UMPB = Original value of Q.BUFF when I/O was queued
16         ; Q.UMPP = Original value of Q.PAR when I/O was queued
17         ; Q.UMVB = Index number of base Unibus map register for transfer
18         ;
19         ; Outputs:
20         ; (SP) = Low-order 16-bits of physical address.
21         ; 2(SP)= High-order 6-bits of physical address (in bits 4-9).
22         ; R5 = Address of Q.WCNT in queue element.
23         ;
24 007336 162706 000004 MPPHY: SUB #4, SP ; MAKE ROOM ON STACK FOR RESULT WORDS
25 007342 016616 000004 MOV 4(SP), (SP) ; MOVE UP RETURN ADDRESS ON STACK
26 007346 010246 MOV R2, -(SP)
27 007350 010346 MOV R3, -(SP)
28 007352 012546 MOV (R5)+, -(SP) ; GET VIRTUAL ADDRESS FROM Q.BUFF
29 007354 005002 CLR R2 ; WE WILL FORM PHYSICAL ADDRESS IN R2-R3
30 ;
31 ; Convert virtual address in I/O queue element into physical
32 ; address and place in R2-R3.
33 ;
34 007356 005765 000000C TST Q.UMRX-Q.WCNT(R5); IS UNIBUS MAPPING NEEDED?
35 007362 001416 BEQ 1$ ; BR IF NOT
36 ;
37 ; We must do Unibus mapping for this I/O request.
38 ; What we compute is not a physical address but is rather a
39 ; virtual address within the Unibus address space.
40 ;
41 007364 166516 000000C SUB Q.UMPB-Q.WCNT(R5), (SP) ; Subtract original Q.BUFF value
42 007370 016503 000000C MOV Q.PAR-Q.WCNT(R5), R3 ; Get current PAR offset (64-byte units)
43 007374 166503 000000C SUB Q.UMPP-Q.WCNT(R5), R3 ; Get # 64-byte blocks Q.PAR has been incr
44 007400 072327 000006 ASH #6, R3 ; Convert to byte offset
45 007404 060316 ADD R3, (SP) ; Increment buffer address
46 007406 116503 000000C MOVQ Q.UMVB-Q.WCNT(R5), R3 ; Get unibus register #
47 007412 073227 000015 ASHC #13, R2 ; * 8192. bytes per register
48 007416 000406 BR 2$
49 ;
50 ; We are not doing Unibus mapping. Get 22-bit physical address to R2-R3.
51 ;
52 007420 162716 000000G 1$: SUB #VPAR6, (SP) ; REMOVE PAGE-6 BIAS
53 007424 016503 000000C MOV Q.PAR-Q.WCNT(R5), R3 ; GET RELOCATION BASE
54 007430 073227 000006 ASHC #6, R2 ; SHIFT R2-R3 LEFT 6 BITS
55 007434 062603 2$: ADD (SP)+, R3 ; ADD (Q.BUFF) TO LOW ORDER
56 007436 005502 ADC R2 ; PROPOGATE CARRY TO HIGH-ORDER
57 ;

```

```
58 ; We have computed the address and it is in R2-R3.  
59 ; Position correctly and store into stack.  
60 ;  
61 007440 072227 000004 ASH #4,R2 ;POSITION HIGH-ORDER TO BITS 4-9  
62 007444 010366 000006 MOV R3,6(SP) ;STORE LOW-ORDER VALUE  
63 007450 010266 000010 MOV R2,10(SP) ;STORE HIGH-ORDER VALUE  
64 007454 012603 MOV (SP)+,R3  
65 007456 012602 MOV (SP)+,R2  
66 ;  
67 ; Finished -- Converted values are on stack below return address.  
68 ;  
69 007460 000207 RETURN
```

PTBYT -- Move byte to user's buffer

```

1          .SBTTL PTBYT -- Move byte to user's buffer
2          ;-----
3          ; PTBYT is called to transfer a byte to the user's buffer area.
4          ;
5          ; Inputs:
6          ;   (SP) = Byte to be transferred.
7          ;   R4  = Pointer to Q.BLKN in queue element
8          ;
9          ; Outputs:
10         ;   Byte is popped from stack.
11         ;   Q.BUFF is incremented.
12         ;
13 007462 PTBYT:
14         ;
15         ; Make R4 point to Q.BUFF rather than Q.BLKN
16         ;
17 007462 062704 000000C          ADD    #<Q.BUFF-Q.BLKN>,R4;MAKE R4 POINT TO Q.BUFF
18         ;
19         ; This request must be mapped through PAR 6.
20         ;
21         ;
22         ; See if we cross a page boundary.
23         ;
24 007466 032714 020000          BIT    #20000,(R4)      ;DID WE CROSS A PAGE BOUNDARY?
25 007472 001405                    BEQ    1$                ;BR IF NOT
26         ;
27         ; We crossed a page boundary.
28         ; Increment Q.PAR and reset Q.BUFF
29         ;
30 007474 062764 000200 000000C    ADD    #200,Q.PAR-Q.BUFF(R4);ADVANCE PAGE NUMBER
31 007502 162714 020000          SUB    #20000,(R4)      ;RESET BUFFER POINTER
32         ;
33         ; Set up Kernel-mode PAR6 to access user's page.
34         ;
35 007506 1$:  DISABL                    ;DISABLE INTERRUPTS
36 007514 013727 000000C    MOV    @#KPAR6,(PC)+   ;SAVE CURRENT KPAR6 CONTENTS
37 007520 000000          11$:  .WORD  0                ;CONTENTS OF KPAR6
38 007522 016437 000000C 000000C    MOV    Q.PAR-Q.BUFF(R4),@#KPAR6;MAP TO USER'S PAGE
39         ;
40         ; Move byte into the user's area
41         ;
42 007530 116634 000002          MOVB   2(SP),@(R4)+    ;MOVE BYTE INTO USER'S SPACE
43 007534 013737 007520' 000000C    MOV    11@,@#KPAR6    ;RESTORE KPAR6 CONTENTS
44 007542          ENABL                    ;ENABLE INTERRUPTS
45         ;
46         ; Increment buffer pointer
47         ;
48 007550 005244          INC    -(R4)          ;INC Q.BUFF
49         ;
50         ; Fix up R4 and pop byte off stack.
51         ;
52 007552 162704 000000C    SUB    #<Q.BUFF-Q.BLKN>,R4;MAKE R4 POINT TO Q.BLKN
53 007556 012616          MOV    (SP)+,(SP)     ;MOVE RETURN ADDRESS OVER DATA BYTE
54 007560 000207          RETURN

```

GTBYT -- Get byte from user's buffer

```

1          .SBTTL  GTBYT  -- Get byte from user's buffer
2          ;-----
3          ; GTBYT is called to acquire a byte from a user buffer.
4          ;
5          ; Inputs:
6          ; R4 = Pointer to Q.BLKN in I/O queue element.
7          ;
8          ; Outputs:
9          ; (SP) = Byte acquired.
10         ; Q.BUFF is incremented.
11         ;
12 007562  GTBYT:
13         ;
14         ; Make room for returned byte on stack.
15         ;
16 007562  011646      MOV      (SP), -(SP)      ; MOVE DOWN RETURN ADDRESS
17         ;
18         ; Make R4 point to Q.BUFF
19         ;
20 007564  062704  000000C      ADD      #<Q.BUFF-Q.BLKN>, R4; MAKE R4 POINT TO Q.BUFF
21         ;
22         ; This address needs to be mapped through PAR 6.
23         ;
24         ;
25         ; See if we crossed a page boundary.
26         ;
27 007570  031427  020000      BIT      (R4), #20000      ; DID ADDRESS CROSS A PAGE BOUNDARY?
28 007574  001405          BEQ      1$              ; BR IF NOT
29         ;
30         ; We have crossed a page boundary.
31         ; Increment Q.PAR and reset Q.BUFF
32         ;
33 007576  062764  000200  000000C      ADD      #200, Q.PAR-Q.BUFF(R4); INCREMENT PAGE POINTER
34 007604  162714  020000      SUB      #20000, (R4)      ; RESET BYTE-WITHIN-PAGE #
35         ;
36         ; Set up kernel-mode PAR6 to map to user's page
37         ;
38 007610          1$:  DISABL          ; DISABLE INTERRUPTS
39 007616  013727  000000G      MOV      @#KPAR6, (PC)+    ; SAVE CURRENT KPAR6 CONTENTS
40 007622  000000          11$:  .WORD  0-          ; CONTENTS OF KPAR6
41 007624  016437  000000C  000000G      MOV      Q.PAR-Q.BUFF(R4), @#KPAR6; MAP TO USER'S PAGE
42         ;
43         ; Move byte from user's buffer
44         ;
45 007632  005066  000002          CLR      2(SP)          ; CLEAR BYTE AREA
46 007636  113466  000002          MOVVB   @(R4)+, 2(SP)     ; GET BYTE FROM USER'S AREA
47 007642  013737  007622'  000000G      MOV      11$, @#KPAR6    ; RESTORE KPAR6 CONTENTS
48 007650          - ENABL          ; ENABLE INTERRUPTS
49         ;
50         ; Increment buffer pointer
51         ;
52 007656  005244          INC      -(R4)          ; INCREMENT Q.BUFF
53         ;
54         ; Fix R4 and return with data byte on top of stack
55         ;
56 007660  162704  000000C      SUB      #<Q.BUFF-Q.BLKN>, R4; MAKE R4 POINT TO Q.BLKN
57 007664  000207          RETURN

```

PTWRD -- Move word to user's buffer

```

1          .SBTTL  PTWRD  -- Move word to user's buffer
2          ;-----
3          ; PTWRD is called to move a word into a user buffer area.
4          ;
5          ; Inputs:
6          ;   (SP) = Word to be moved to user's buffer.
7          ;   R4   = Pointer to Q.BLKN in queue element.
8          ;
9          ; Outputs:
10         ;   Data word is popped from stack.
11         ;   Q.BUFF is incremented.
12         ;
13 007666 PTWRD:
14         ;
15         ; Make R4 point to Q.BUFF cell in queue element.
16         ;
17 007666 062704 000000C          ADD    #<Q.BUFF-Q.BLKN>,R4;POINT TO Q.BUFF
18         ;
19         ; This address must be mapped through PAR 6.
20         ;
21         ;
22         ; See if we crossed a page boundary
23         ;
24 007672 031427 020000          BIT    (R4),#20000    ;DID WE CROSS A PAGE BOUNDARY?
25 007676 001405                    BEQ    1$                ;BR IF NOT
26         ;
27         ; We crossed a page boundary
28         ; Increment Q.PAR and reset Q.BUFF
29         ;
30 007700 062764 000200 000000C    ADD    #200,Q.PAR-Q.BUFF(R4);ADVANCE PAGE POINTER
31 007706 162714 020000          SUB    #20000,(R4)    ;RESET BYTE-WITHIN-PAGE ADDRESS
32         ;
33         ; Set up kernel-mode PAR6 to map to user's page.
34         ;
35 007712 1$:  DISABL                ;DISABLE INTERRUPTS
36 007720 013727 000000C    MOV    @#KPAR6,(PC)+    ;SAVE CURRENT KPAR6 CONTENTS
37 007724 000000          11$: .WORD 0                ;CONTENTS OF KPAR6
38 007726 016437 000000C 000000C    MOV    Q.PAR-Q.BUFF(R4),@#KPAR6;MAP TO USER'S PAGE
39         ;
40         ; Move word into user's buffer.
41         ;
42 007734 016634 000002          MOV    2(SP),@(R4)+    ;MOVE WORD INTO USER'S BUFFER
43 007740 013737 007724' 000000C    MOV    11@,@#KPAR6    ;RESTORE KPAR6 CONTENTS
44 007746                    ENABL                ;ENABLE INTERRUPTS
45         ;
46         ; Advance buffer pointer
47         ;
48 007754 062744 000002          ADD    #2,-(R4)        ;ADVANCE Q.BUFF
49         ;
50         ; Fix R4 and pop data word from stack.
51         ;
52 007760 162704 000000C    SUB    #<Q.BUFF-Q.BLKN>,R4;MAKE R4 POINT TO Q.BLKN
53 007764 012616                    MOV    (SP)+,(SP)      ;POP DATA WORD -- MOVE UP RETURN ADDRESS
54 007766 000207                    RETURN

```

```

1          .SBTTL  EXTP1  -- External KPAR mapping routine for drivers
2          ;-----
3          ;
4          ; EXTP1 is called from RMON offset 432 by device drivers wishing
5          ; to map directly to the user's I/O buffer.  It is initiated in the
6          ; device driver by the following call:
7          ;     JSR     RO,@#P1EXT
8          ;     .WORD  <<instruction length> + 2 (for KPAR6)>
9          ;     <instructions>
10         ;     .WORD  <new KPAR 6 mapping value>
11         ; The drivers which use this routine are DD and DM.
12         ;
13         ;
14 007770  EXTP1:
15 007770  013746  0000000  MOV     @#KPAR6,-(SP)  ;Save kernal par 6 contents
16 007774  010146          MOV     R1,-(SP)      ;Save r1
17 007776  010246          MOV     R2,-(SP)      ;Save r2
18 010000  012002          MOV     (R0)+,R2      ;Get the length of code (# bytes +2)
19 010002  060200          ADD     R2,R0         ;Point to the code beyond par value
20 010004  010001          MOV     R0,R1        ;Copy return address
21 010006  005741          TST     -(R1)        ;Skip KPAR6 mapping value
22         ;
23         ; Put instructions and a return on the stack.
24         ;
25 010010  012746          MOV     (PC)+,-(SP)  ;Save return instruction on stack
26 010012  000207          RETURN          ;Return instruction pushed to stack
27 010014  006202          ASR     R2          ;Convert # bytes to # words
28 010016  014146  1#:  MOV     -(R1),-(SP)  ;Push instructions and # bytes on stack
29 010020  077202          SOB    R2,1#       ;Continue until all saved
30         ;
31         ; Calculate a stack frame pointer and put it on the stack.
32         ;
33 010022  012602          MOV     (SP)+,R2     ;Get length of code (# bytes +2)
34 010024  060602          ADD     SP,R2       ;Calculate stack pointer to saved R2
35 010026  005722          TST     (R2)+       ;Calculate stack frame pointer to saved R1
36 010030  010246          MOV     R2,-(SP)    ;Save stack frame pointer on top of stack
37 010032  011201          MOV     (R2),R1     ;Restore r1 prior to JSR
38 010034  014202          MOV     -(R2),R2    ;Restore r2 prior to JSR
39 010036  016037  177776  000000G  MOV     -2(R0),@#KPAR6 ;Initialize KPAR6 mapping
40         ;
41         ; The stack is in the following configuration:
42         ;     SP --> stack frame pointer to saved R1
43         ;     instructions
44         ;     RETURN
45         ;     saved R2
46         ;     saved R1 (pointed to by stack frame pointer on top of stack)
47         ;     KPAR6
48         ;     saved R0 (from JSR call)
49         ;
50 010044  004766  000002          CALL    2(SP)       ;Call instructions saved on stack
51 010050  011606          MOV     @SP,SP      ;Restore stack frame pointer
52 010052  005726          TST     (SP)+       ;Skip over r1 value
53 010054  012637  000000G  MOV     (SP)+,@#KPAR6 ;Restore kernal par 6 contents
54 010060  000200          RTS     R0          ;Return to handler

```

```

1          .SBTTL  CVTPHY -- Convert a virtual address to a physical addr
2          ;-----
3          ; Convert a 16-bit virtual address within a job region into a full
4          ; 22-bit physical address.
5          ;
6          ; Inputs:
7          ; R0 = 16-bit virtual address within job region
8          ; R5 = Pointer to 3rd word of I/O queue element
9          ;
10         ; Outputs:
11         ; R1 = High-order 6 bits of physical address (starting at bit 4)
12         ; R2 = Low-order 16 bits of physical address
13         ; C-flag is set on return if physical address is > 56Kb
14         ;
15 010062 010046 CVTPHY: MOV     R0, -(SP)
16 010064 010446      MOV     R4, -(SP)
17 010066 010504      MOV     R5, R4          ; Put I/O queue element pointer in R5
18         ;
19         ; Call RELOC to convert the virtual address into an address within the
20         ; PAR6 region and get a base offset for PAR6.
21         ;
22 010070 004737 010126' CALL   RELOC          ; Get user's par mapping value
23         ;
24         ; At this point:
25         ; R1 = Mapping value to use to map Par6 to the area.
26         ; R2 = Address within Par6 region that specified mapped start of area.
27         ;
28 010074 162702 0000000 SUB    #VPAR6, R2    ; Strip par6 relocation bias
29 010100 005000      CLR    R0            ; Set up for shift
30 010102 073027 000006 ASHC   #6, R0        ; Shift R0-R1 left 6 bits
31 010106 060102      ADD    R1, R2        ; Get low-order physical address
32 010110 005500      ADC    R0            ; Propagate carry to high-order
33 010112 010001      MOV    R0, R1        ; Return high-order part in R1
34 010114 072127 000004 ASH    #4, R1        ; Align high bits starting at bit 4
35         ;
36         ; Finished
37         ;
38 010120 012604      MOV    (SP)+, R4
39 010122 012600      MOV    (SP)+, R0
40 010124 000207      RETURN

```

```

1          ; SBTTL RELOC -- Mapping relocation calculation
2          ; -----
3          ; RELOC is a subroutine which converts a 16-bit virtual address contained
4          ; in R0 into a PAR6 relocation value returned in R1 and PAR6 offset address
5          ; returned in R2.
6          ;
7          ; Inputs:
8          ;   R0 = Virtual address to be converted.
9          ;   R4 = Pointer to 3rd word of Queue element.
10         ;
11         ; Outputs:
12         ;   R1 = PAR6 mapping relocation value.
13         ;   R2 = PAR6 offset (i.e., address within PAR6 region)
14         ;
15 010126 RELOC:
16         ;
17         ; Save registers.
18         ;
19 010126 010346      MOV     R3, -(SP)
20 010130 116401 000000C  MOVB   Q.JOB-Q.BLKN(R4), R1      ; Get job number from queue element
21 010134 006301      ASL     R1              ; Convert to job index number
22         ;
23         ; R0 contains the virtual address. Separate the par register number
24         ; from the offset value to obtain the mapped address
25         ;
26 010136 010003      MOV     R0, R3              ; Save conversion address
27 010140 005002      CLR     R2              ; Clear high order bits
28 010142 071227 020000  DIV     #20000, R2      ; Separate page and offset pointer
29 010146 006302      ASL     R2              ; Create word offset pointer
30         ;
31         ; User's mapping in job context is valid - map to job context.
32         ;
33 010150      DISABL                    ;;; Disable interrupts
34 010156 013727 000000G  MOV     @#KPAR6, (PC)+      ;;; Save par6 contents
35 010162 000000      .WORD 0          ;;; Contains par6 contents
36 010164 016137 000000G 000000G  MOV     LCXPAR(R1), @#KPAR6 ;;; Map through par6
37         ;
38         ; Obtain par mapping contents from job context region.
39         ;
40 010172 016201 000000G  MOV     CUPARO(R2), R1      ;;; Get the PAR value for the page
41 010176 013737 010162' 000000G  MOV     11$, @#KPAR6      ;;; Restore par6 original contents
42 010204      ENABL                    ;;; Enable interrupts
43         ;
44         ; Check for valid offset.
45         ;
46 010212 005002      CLR     R2              ; Clear high order bits
47 010214 071227 000100  DIV     #100, R2          ; Calc lowest 64-byte mapping offset
48 010220 060201      ADD     R2, R1          ; Adjust the PAR value accordingly
49 010222 010302      MOV     R3, R2          ; Move par offset into R2
50 010224 062702 000000G  ADD     #VPAR6, R2        ; Adjust to PAR6 virtual address
51         ;
52         ; Restore registers and return.
53         ;
54 010230 012603      MOV     (SP)+, R3
55 010232 000207      RETURN

```

```

1          .SBTTL  BLKMOV  -- Memory block move routine
2          ;-----
3          ; BLKMOV is used by the VM handler to transfer data from one memory area
4          ; to another.
5          ;
6          ; Inputs:
7          ;   R1 = input PAR value
8          ;   R2 = input buffer address
9          ;   R3 = output PAR value
10         ;   R4 = output buffer address
11         ;   R5 = word count
12         ;
13
14 010234  BLKMOV:
15 010234  010046      MOV      R0, -(SP)          ; Save registers
16 010236  013746  0000000  MOV      @#KPAR5, -(SP)      ; Save kernel PAR 5 & 6 mapping
17 010242  013746  0000000  MOV      @#KPAR6, -(SP)
18 010246  042702  160000    BIC      #160000, R2        ; Strip input buffer bias
19 010252  052702  120000    BIS      #120000, R2        ; Convert input buffer to PAR5 base
20 010256  042704  160000    BIC      #160000, R4        ; Strip output buffer bias
21 010262  052704  140000    BIS      #140000, R4        ; Convert output buffer to PAR6 base
22         ;
23         ; Convert base buffer offsets and PAR to allow block move.
24         ;
25 010266  020227  130000    1$:     CMP      R2, #130000      ; Check input buffer address
26 010272  103404          BLO      2$                 ; Br if below boundary
27 010274  162702  010000    SUB      #10000, R2         ; Subtract 4K byte offset
28 010300  062701  000100    ADD      #100, R1          ; Adjust PAR value
29 010304  020427  150000    2$:     CMP      R4, #150000      ; Check input buffer address
30 010310  103404          BLO      3$                 ; Br if below boundary
31 010312  162704  010000    SUB      #10000, R4        ; Subtract 4K byte offset
32 010316  062703  000100    ADD      #100, R3          ; Adjust PAR value
33         ;
34         ; Move up to one block at a time.
35         ;
36 010322  012700  000400    3$:     MOV      #256, R0         ; Move 256. words
37 010326  020005          CMP      R0, R5            ; Check with actual word count
38 010330  103401          BLO      5$                 ; Br if word count > 256.
39 010332  010500          MOV      R5, R0           ; Otherwise use actual word count
40         ;
41         ; Initialize the PAR registers and perform the move.
42         ;
43 010334  160005    5$:     SUB      R0, R5            ; Adjust the number of words moved
44 010336  010137  0000000  MOV      R1, @#KPAR5       ; Set PAR5 value
45 010342  010337  0000000  MOV      R3, @#KPAR6       ; Set PAR6 value
46 010346  005200    INC      R0                ; Adjust word count for double move
47 010350  006200    ASR      R0                ; Divide for double word move
48 010352  103001    BCC      20$              ; Br if single word move necessary
49 010354  012224    10$:    MOV      (R2)+, (R4)+    ; Move input buffer to output buffer
50 010356  012224    20$:    MOV      (R2)+, (R4)+
51 010360  077003    SOB      R0, 10$          ; Continue until all has been moved
52 010362  005705    TST      R5                ; Check for remaining words
53 010364  101340    BHI      1$                ; Br if more to move
54 010366  012637  0000000  MOV      (SP)+, @#KPAR6    ; Restore PAR 5 & 6 mapping
55 010372  012637  0000000  MOV      (SP)+, @#KPAR5
56 010376  012600    MOV      (SP)+, R0         ; Restore registers
57 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

FREIDQ	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											
Q16STK	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#											
IOHANG	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*											
LNXNUM	1-134	8-15											
LOMAP	1-84	9-21	9-35										
LPRI	1-89	1-97	16-68	17-72	17-74*	18-30							

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								
SWPFR	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						

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							
UMRWH	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							
ZERQ	20-36	20-52#	22-54						