

Table of contents

2-	1	MIOCHK	-- Determine if I/O operation should be mapped
4-	1	MIOBGN	-- Initiate a mapped I/O operation
5-	1	MIOVRT	-- Perform mapped write operation
6-	1	MIOWRC	-- Mapped write completion routine
7-	1	MIOIRD	-- Initiate a mapped read operation
8-	1	MIORDC	-- Mapped read completion routine
9-	1	MIOFIN	-- Completion of a mapped I/O operation
10-	1	MIOGTQ	-- Get I/O queue element for mapped I/O operation
11-	1	MIOADV	-- Advance buffer address and word count
12-	1	MIOGET	-- Allocate a mapped I/O data buffer
13-	1	MIOFRE	-- Free a mapped I/O data buffer.

```

1          .TITLE  TSMIO -- TSX-Plus Mapped I/O Support
2          .ENABL  LC
3          .ENABL  AMA
4          .DSABL  QBL
5 000000   .CSECT  TSMIO
6 000000   051267  TSMIO: .RAD50  /MIO/          ;Overlay region id
7          ;
8          ; TSMIO is the TSX-Plus system overlay that contains the routines
9          ; used to perform mapped I/O operations for devices with 18-bit
10         ; controllers running on 22-bit Q-bus systems.
11         ;
12         ; Copyright (c) 1983, 1984.
13         ; S&H Computer Systems, Inc.
14         ; Nashville, Tennessee USA
15         ; All rights reserved.
16         ;
17         ; Global definitions
18         ;
19         .GLOBL  MIOCHK
20         ;
21         ; Global references
22         ;
23         .GLOBL  Q. FUNC, VPAR6, Q. COMP, Q. CSW, GETSYQ, Q. BUFF
24         .GLOBL  MIOGWT, MIOMRD, Q. WCNT, Q. DEVX, IOQSIZ
25         .GLOBL  DVSTAT, DS$ID, SYQIO, Q. PAR, VMIOSZ, S$QMIO, QNSPNX, CHKABT
26         .GLOBL  CORUSR, UREGO, Q. BLKN, Q. JOB, Q. JNUM, PSW, INTPRI, IOCMPL
27         .GLOBL  DI$DY, DI$MM, DI$MS, MIOBHD, MI$LNK, Q. CHAN, MI$SBP
28         .GLOBL  MI$TRW, MI$DGE, MI$RWF, MI$UBP, MI$UBD, MI$CWC, MI$JOB
29         .GLOBL  C. CSW, CS$ERR, CS$EOF, DI$MT, DI$DX, MIODBG
30         .GLOBL  Q. FLAG, QF$MIO, QF$OWC, EM$MIO
31         .GLOBL  MIOWHD, MIOSYQ, MW$LNK, MW$IIOQ, Q. UCSW
32         ;
33         ; Macro definitions
34         ;
35         ; Macro to disable device interrupts
36         ;
37         .MACRO  DISABL
38         BIS    #340, @#PSW
39         .ENDM  DISABL
40         ;
41         ; Macro to enable interrupts
42         ;
43         .MACRO  ENABL
44         BIC    INTPRI, @#PSW
45         .ENDM  ENABL
46         ;
47         ; Macro to print an error message when a system crash occurs.
48         ;
49         ; Arguments:
50         ; MSG = Name of error message to print.
51         ; ARG = (Optional) argument value to display with error message.
52         ;
53         .GLOBL  DIEMSG, DIEARG, SYSHLT
54         .MACRO  DIE    MSG, ARG
55         MOV    MSG, @#DIEMSG
56         .IF   NB, ARG
57         MOV    ARG, @#DIEARG

```

58
59
60

. ENDC
CALL @#SYSHLT
. ENDM DIE

```

1          .SBTTL  MIOCHK -- Determine if I/O operation should be mapped
2          ;-----
3          ; MIOCHK is called whenever an I/O operation is about to be queued
4          ; for an 18-bit device that may potentially need mapping.
5          ; MIOCHK checks to see if the operation is a Read or Write request that
6          ; needs mapping and checks to see if the user's buffer is already
7          ; in the low 248K memory area.
8          ;
9          ; Inputs:
10         ; R1 = Address of I/O queue element.
11         ;
12         ; Outputs:
13         ; C-flag cleared ==> Don't need to do mapping; queue request normally.
14         ; C-flag set      ==> Must map operation; don't queue current request.
15         ;
16 000002 010146 MIOCHK: MOV     R1,-(SP)
17 000004 010246         MOV     R2,-(SP)
18 000006 010346         MOV     R3,-(SP)
19 000010 010446         MOV     R4,-(SP)
20 000012 010546         MOV     R5,-(SP)
21         ;
22         ; See if we are reentering MIOCHK with a queue element that has
23         ; already been worked on by MIOCHK.
24         ;
25 000014 132761 0000000 0000000 BITB   #QF#MIO,Q.FLAG(R1);Has this I/O operation already been mapped?
26 000022 001123         BNE    NDMAP      ;Br if yes
27         ;
28         ; Get device index number
29         ;
30 000024 116102 0000000 MOVVB  Q.DEVX(R1),R2  ;Get device index number
31 000030 001520         BEQ    NDMAP      ;Br if this operation has been cancelled
32         ;
33         ; Determine if a .SPFUN is being queued
34         ;
35 000032 116103 0000000 MOVVB  Q.FUNC(R1),R3  ;Is this a .SPFUN operation?
36 000036 001443         BEQ    3$        ;Br if not
37         ;
38         ; This is a .SPFUN operation.
39         ; Check to see if this operation must be treated specially.
40         ;
41 000040 016204 0000000 MOV     DVSTAT(R2),R4 ;Get device status flags
42 000044 042704 0000000 BIC     #^C<DS#ID>,R4 ;Clear all but device id
43 000050 042703 177400 BIC     #^C377,R3     ;Clear sign extension of special fun code
44 000054 000303 SWAB   R3             ;Put special function code in high order
45 000056 050304 BIS     R3,R4         ;Combine function code with device id
46 000060 012700 000310' MOV     #DEVTBL,R0    ;Get pointer to device/function table
47 000064 020420 1$: CMP    R4,(R0)+    ;Try to find device-id/function-code combo
48 000066 001001 BNE    2$            ;Br if this is not it
49 000070 000130 JMP     @(R0)+        ;Enter special processing routine
50 000072 005720 2$: TST   (R0)+        ;Point to next table entry
51 000074 020027 000450' CMP    R0,#DEVEND    ;Checked all entries?
52 000100 103771 BLD    1$            ;Loop if not
53         ;
54         ; We don't recognize the .SPFUN code as one that requires special
55         ; device-dependent processing.
56         ; Treat .SPFUN 376 as a write and 377 as read.
57         ;

```

```

58 000102 000303          SWAB   R3          ;Get back function code
59 000104 120327 000376    CMPB   R3,#376      ;Is this a write?
60 000110 001426          BEQ    WRTFUN       ;Br if yes
61 000112 120327 000377    CMPB   R3,#377      ;Is this a read?
62 000116 001417          BEQ    RDFUN       ;Br if yes
63                          ;
64                          ; Treat .SPFUN 373 as a request to get the device size
65                          ;
66 000120 120327 000373    CMPB   R3,#373      ;Request to get device size?
67 000124 001007          BNE    4$          ;Br if not
68 000126 005003          CLR    R3          ;Say this is a read operation
69 000130 012702 000001    MOV    #1,R2       ;Say 1 word is being read
70 000134 152761 000000G 000000G  BISB   #QF$DWC,Q.FLAG(R1) ;Set flag saying to use original word count
71 000142 000416          BR     CHKBUF      ;Process the operation
72                          ;
73                          ; This special function can be processed without mapping
74                          ;
75 000144 000452          4$:   BR     NOMAP      ;No mapping required
76                          ;
77                          ; This is not a .SPFUN request.
78                          ; Determine if it is a .READ, .WRITE, or (.SEEK)
79                          ;
80 000146 005761 000000G  3$:   TST    Q.WCNT(R1) ;Read/Write/Seek?
81 000152 002405          BLT    WRTFUN       ;Br if write operation
82 000154 001446          BEQ    NOMAP       ;Seek operation
83                          ;
84                          ; This is a read request
85                          ;
86 000156 005003          RDFUN: CLR   R3          ;Set read flag
87 000160 016102 000000G  MOV    Q.WCNT(R1),R2 ;Get word count for read
88 000164 000405          BR     CHKBUF
89                          ;
90                          ; This is a write request
91                          ;
92 000166 012703 000001    WRTFUN: MOV   #1,R3      ;Set write flag
93 000172 016102 000000G  MOV    Q.WCNT(R1),R2 ;Get word count
94 000176 005402          NEG    R2          ;Stored negative to signal write
95                          ;
96                          ; At this point we have determined that data is going to be read
97                          ; or written and have the positive word count in R2.
98                          ; Check to see if the buffer is entirely below 256Kb.
99                          ;
100 000200 016105 000000G  CHKBUF: MOV   Q.BUFF(R1),R5 ;Get user's buffer virtual address
101 000204 162705 000000G  SUB    #VPAR6,R5     ;Remove virtual buffer bias
102 000210 062705 000077    ADD    #63.,R5      ;Bound up to next 64-byte block
103 000214 072527 177772    ASH    #-6.,R5      ;Convert address of 64-byte block #
104 000220 066105 000000G  ADD    Q.PAR(R1),R5  ;Add base 64-byte block #
105 000224 010200          MOV    R2,R0        ;Get word count
106 000226 062700 000037    ADD    #31.,R0      ;Bound up to 64-byte block size
107 000232 072027 177773    ASH    #-5.,R0      ;Convert to # 64-byte blocks
108 000236 060005          ADD    R0,R5        ;Compute 64-byte block # of top of buffer
109 000240 005727 000000G  TST    #MIDDBG      ;Are we to map all operations?
110 000244 001003          BNE    1$          ;Br if yes
111 000246 020527 007600    CMP    R5,#3968     ;Is buffer entirely below 248Kb?
112 000252 101407          BLOS   NOMAP       ;Br if yes -- Mapping not needed
113                          ;
114                          ; We must do buffer mapping for this I/O operation

```

```
115 ; Get a mapped I/O buffer and control block.
116 ;
117 000254 004737 001434' 1$: CALL MIOGET ;Get mapping buffer and control block
118 000260 103405 BCS MIOXIT ;Br if mapping deferred till buffer freed
119 ;
120 ; Initiate a mapped I/O operation
121 ;
122 000262 004737 000570' CALL MIOBGN ;Initiate the mapping
123 000266 000261 SEC ;Indicate that mapping was needed
124 000270 000401 BR MIOXIT ;Finished
125 ;
126 ; We do not have to do buffer mapping
127 ;
128 000272 000241 NDMAP: CLC ;Indicate mapping not needed
129 ;
130 ; Finished
131 ;
132 000274 012605 MIOXIT: MOV (SP)+, R5
133 000276 012604 MOV (SP)+, R4
134 000300 012603 MOV (SP)+, R3
135 000302 012602 MOV (SP)+, R2
136 000304 012601 MOV (SP)+, R1
137 000306 000207 RETURN
```

```

1          ; -----
2          ; Table of special function codes for certain devices.
3          ;
4          .MACRO  SPFUN  DEV,FUN,RTN
5          .BYTE   DEV,FUN
6          .WORD   RTN
7          .ENDM   SPFUN
8          ;
9          ; Define those devices which have specific .SPFUN functions that
10         ; require special processing.
11         ;
12         DEVTBL:
13         SPFUN  DI$DY, 377, DYREAD      ; DY 377 -- Read 129 word sector
14         SPFUN  DI$DY, 376, DYWRIT     ; DY 376 -- Write 129 word sector
15         SPFUN  DI$DY, 375, DYWRIT     ; DY 375 -- Write 129 word sector & del
16
17         SPFUN  DI$MT, 377, CLRADR     ; MT 377 -- Write EOF
18         SPFUN  DI$MT, 376, CLRADR     ; MT 376 -- Forward one block
19         SPFUN  DI$MT, 374, WRTFUN     ; MT 374 -- Write with extended gap
20         SPFUN  DI$MT, 373, CLRADR     ; MT 373 -- Rewind
21         SPFUN  DI$MT, 371, WRTFUN     ; MT 371 -- Write
22         SPFUN  DI$MT, 370, RDFUN      ; MT 370 -- Read
23
24         SPFUN  DI$MM, 377, CLRADR     ; MM 377 -- Write EOF
25         SPFUN  DI$MM, 376, CLRADR     ; MM 376 -- Forward one block
26         SPFUN  DI$MM, 374, WRTFUN     ; MM 374 -- Write with extended gap
27         SPFUN  DI$MM, 373, CLRADR     ; MM 373 -- Rewind
28         SPFUN  DI$MM, 371, WRTFUN     ; MM 371 -- Write
29         SPFUN  DI$MM, 370, RDFUN      ; MM 370 -- Read
30
31         SPFUN  DI$MS, 377, CLRADR     ; MS 377 -- Write EOF
32         SPFUN  DI$MS, 376, CLRADR     ; MS 376 -- Forward one block
33         SPFUN  DI$MS, 374, WRTFUN     ; MS 374 -- Write with extended gap
34         SPFUN  DI$MS, 373, CLRADR     ; MS 373 -- Rewind
35         SPFUN  DI$MS, 371, WRTFUN     ; MS 371 -- Write
36         SPFUN  DI$MS, 370, RDFUN      ; MS 370 -- Read
37
38         SPFUN  DI$DX, 377, DXREAD     ; DX 377 -- Read 65 word sector
39         SPFUN  DI$DX, 376, DXWRIT     ; DX 376 -- Write 65 word sector
40         SPFUN  DI$DX, 375, DXWRIT     ; DX 375 -- Write 65 word sector & del
41         DEVEND:
42         ;
43         ; DY 377 -- Read 129 words
44         ;
45         000450 012702 000201  DYREAD: MOV    #129, R2      ; Set word count
46         000454 005003                CLR    R3          ; Set read flag
47         000456 152761 000000G 000000G  BISB   #GF$OWC, Q, FLAG(R1) ; Set flag saying to use original word count
48         000464 000137 000200'        JMP    CHKBUFF     ; Process the operation
49         ;
50         ; DY 375/376 -- Write 129 words
51         ;
52         000470 012702 000201  DYWRIT: MOV    #129, R2      ; Set word count
53         000474 012703 000001        MOV    #1, R3      ; Set write flag
54         000500 152761 000000G 000000G  BISB   #GF$OWC, Q, FLAG(R1) ; Set flag saying to use original word count
55         000506 000137 000200'        JMP    CHKBUFF     ; Process the operation
56         ;
57         ; DX 377 -- Read 65 words

```

```
58 ;  
59 000512 012702 000101 DXREAD: MOV #65,R2 ;Get word count  
60 000516 005003 CLR R3 ;Set read flag  
61 000520 152761 0000000 0000000 BISB #QF$QWC,Q.FLAG(R1) ;Set flag saying to use original word count  
62 000526 000137 000200' JMP CHKBUF ;Process the operation  
63 ;  
64 ; DX 375/376 -- Write 65 words  
65 ;  
66 000532 012702 000101 DXWRIT: MOV #65,R2 ;Get word count  
67 000536 012703 000001 MOV #1,R3 ;Set write flag  
68 000542 152761 0000000 0000000 BISB #QF$QWC,Q.FLAG(R1) ;Set flag saying to use original word count  
69 000550 000137 000200' JMP CHKBUF ;Process the operation  
70 ;  
71 ; Clear buffer address for operations which do not require actual I/O.  
72 ;  
73 000554 005061 0000000 CLRADR: CLR Q.BUFF(R1) ;Clear unused buffer address  
74 000560 005061 0000000 CLR Q.PAR(R1) ; and PAR value  
75 000564 000137 000272' JMP NOMAP ;Exit indicating no mapping needed
```

MIOBGN -- Initiate a mapped I/O operation

```

1          .SBTTL  MIOBGN -- Initiate a mapped I/O operation
2          ;-----
3          ; MIOBGN is called to initiate a mapped I/O operation.
4          ;
5          ; Inputs:
6          ; R1 = Address of original I/O queue element.
7          ; R2 = Actual word count.
8          ; R3 = 0==>Read; 1==>Write
9          ; R5 = Address of mapped I/O control block
10         ;
11 000570 010246 MIOBGN: MOV      R2,-(SP)
12 000572 010346      MOV      R3,-(SP)
13         ;
14         ; Set up information in the control block about this transfer.
15         ;
16 000574 010165 0000000      MOV      R1,MI$OQE(R5) ; Save address of original queue element
17 000600 010265 0000000      MOV      R2,MI$TRW(R5) ; Total number of words to transfer
18 000604 110365 0000000      MOVB     R3,MI$RWF(R5) ; Set read/write flag (0=read,1=write)
19 000610 016102 0000000      MOV      Q.BUFF(R1),R2 ; Get virtual address of user's buffer
20 000614 162702 0000000      SUB      #VPAR6,R2 ; Remove virtual buffer bias
21 000620 005003      CLR      R3 ; Clear for shift
22 000622 073227 177772      ASHC     #-6.,R2 ; Convert address to 64-byte blk # & offset
23 000626 066102 0000000      ADD      Q.PAR(R1),R2 ; Add base 64-byte block number
24 000632 010265 0000000      MOV      R2,MI$UBP(R5) ; This is the 64-byte block # of buffer base
25 000636 000303      SWAB     R3 ; Get byte within block to low-order of R5
26 000640 072327 177776      ASH      #-2,R3
27 000644 010365 0000000      MOV      R3,MI$UBO(R5) ; Save offset within 64-byte block
28         ;
29         ; Initiate the mapped I/O transfer
30         ;
31 000650 105765 0000000      TSTB     MI$RWF(R5) ; Read or write operation?
32 000654 001403      BEQ      1$ ; Br if read
33 000656 004737 000676'      CALL     MIDWRT ; Start a mapped write
34 000662 000402      BR      9$
35 000664 004737 001026' 1$: CALL     MIORD ; Start a mapped read
36         ;
37         ; Finished
38         ;
39 000670 012603 9$: MOV      (SP)+,R3
40 000672 012602      MOV      (SP)+,R2
41 000674 000207      RETURN

```

MIOVRT -- Perform mapped write operation

```

1          .SBTTL  MIOVRT -- Perform mapped write operation
2          ;-----
3          ; MIOVRT is called to perform a mapped write operation.
4          ; It writes as much data as will fit in the mapped I/O buffer and then
5          ; allows a completion routine call to continue the write.
6          ;
7          ; Inputs:
8          ; R5 = Pointer to mapped I/O control block.
9          ;
10         MIOVRT: MOV      R1, -(SP)
11             MOV      R2, -(SP)
12         ;
13         ; Determine how much data to write this time
14         ;
15         MOV      MI$TRW(R5), R2 ; Get total number of words left to write
16         MOVB    VMIOSZ, R0      ; Get # blocks allocated for buffer
17         SWAB    R0              ; Convert to # words
18         CMP     R2, R0          ; Is request larger than buffer?
19         BLOS   1$              ; Br if not
20         MOV     R0, R2          ; Write only as much as buffer can hold
21         1$:    MOV     R2, MI$CWC(R5) ; Set current word count in control block
22         ;
23         ; Move data from user's buffer to the map buffer
24         ;
25         CALL    MIOVRT          ; Move data to map buffer
26         ;
27         ; Set up an I/O queue element for the mapped write
28         ;
29         CALL    MIOGTQ         ; Get a queue element
30         MOV     #MIOVRT, Q.COMP(R1) ; Set address of completion routine
31         ;
32         ; Update the user's buffer address and remaining word count
33         ;
34         CALL    MIOADV         ; Advance buffer address
35         ;
36         ; Initiate the mapped write
37         ;
38         CALL    SYQID         ; Start the write from the mapped buffer
39         ;
40         ; Finished
41         ;
42         9$:    MOV     (SP)+, R2
43             MOV     (SP)+, R1
44             RETURN

```

MIDWRC -- Mapped write completion routine

```

1          .SBTTL  MIDWRC -- Mapped write completion routine
2          ;-----
3          ; MIDWRC is called as a completion routine to a mapped write operation.
4          ;
5          ; Inputs:
6          ; R1 = Address of mapped I/O control block.
7          ;
8 000762 010105  MIDWRC: MOV      R1,R5          ;Get address of mapped I/O control block
9          ;
10         ; If end of file or hardware error was reported by handler,
11         ; terminate the I/O operation.
12         ;
13 000764 016500 000000G  MOV      MI$QOE(R5),R0  ;Get address of original queue element
14 000770 016000 000000G  MOV      Q.CSW(R0),R0  ;Get address of channel block
15 000774 032760 000000C 000000G  BIT      #<CS$EOF!CS$ERR>,C.CSW(R0) ;End of file or hardware error?
16 001002 001006          BNE      1$          ;Br if yes -- Terminate the transfer
17         ;
18         ; See if there is any more data remaining to be written
19         ;
20 001004 005765 000000G  TST      MI$TRW(R5)    ;Any data left to write?
21 001010 001403          BEQ      1$          ;Br if not
22         ;
23         ; There is data left to be written.
24         ; Initiate another write.
25         ;
26 001012 004737 000676'  CALL     MIDWRT        ;Start another mapped write
27 001016 000402          BR      9$
28         ;
29         ; There is no data left.
30         ; Terminate the write.
31         ;
32 001020 004737 001156'  1$:     CALL     MIOFIN    ;Entire I/O operation is finished
33         ;
34         ; Finished
35         ;
36 001024 000207          9$:     RETURN

```

MIORD -- Initiate a mapped read operation

```

1          .SBTTL  MIORD  -- Initiate a mapped read operation
2          ;-----
3          ; MIORD is called to initiate a mapped read operation.
4          ; It reads as many words as will fit in the mapping buffer and uses
5          ; a completion routine to finish the operation.
6          ;
7          ; Inputs:
8          ; R5 = Address of mapped I/O control block.
9          ;
10         MIORD:  MOV      R1,-(SP)
11         001026 010146      MOV      R2,-(SP)
12         ;
13         ; Determine how much data to read this time
14         ;
15         001032 016502 000000G      MOV      MI$TRW(R5),R2      ;Get number of words remaining to be read
16         001036 113700 000000G      MOVB     VMIOSZ,R0        ;Get # blocks allocated for buffer
17         001042 000300              SWAB     R0                ;Convert to # words
18         001044 020200              CMP      R2,R0            ;Can we read all that is left?
19         001046 101401              BLOS    1$              ;Br if yes
20         001050 010002              MOV      R0,R2            ;No -- Truncate to mapping buffer size
21         001052 010265 000000G      1$:     MOV      R2,MI$CWC(R5) ;Set current word count in control block
22         ;
23         ; Get an I/O queue element
24         ;
25         001056 004737 001200'      CALL     MIDGTQ          ;Get an I/O queue element
26         001062 012761 001102' 000000G      MOV      #MIORDC,Q.COMP(R1) ;Set address of read completion routine
27         ;
28         ; Initiate the read
29         ;
30         001070 004737 000000G      CALL     SYQIO          ;Start the read operation
31         ;
32         ; Finished
33         ;
34         001074 012602              MOV      (SP)+,R2
35         001076 012601              MOV      (SP)+,R1
36         001100 000207              RETURN

```

```

1          .SBTTL  MIORDC -- Mapped read completion routine
2          ;-----
3          ; MIORDC is called as a completion routine when a mapped read operation
4          ; finishes.  It moves the data that was read into the mapped buffer to the
5          ; user's buffer and then checks to see if more data needs to be read.
6          ;
7          ; Inputs:
8          ; R1 = Address of mapped I/O control block
9          ;
10         MIORDC: MOV      R1,R5          ;Get address of mapped I/O control block
11         ;
12         ; Move data from system buffer to user's buffer
13         ;
14         001102  010105          CALL      MIOMRD          ;Move data to user's buffer
15         ;
16         ; If end of file or hardware error was reported by handler,
17         ; terminate the I/O operation.
18         ;
19         001110  016500  0000000  MOV      MI$OQE(R5),R0      ;Get address of original queue element
20         001114  016000  0000000  MOV      Q.CSW(R0),R0      ;Get address of channel block
21         001120  032760  0000000  0000000  BIT      #<CS$EOF!CS$ERR>,C.CSW(R0) ;End of file or hardware error?
22         001126  001010          BNE      1$              ;Br if yes -- Terminate the transfer
23         ;
24         ; Advance user's buffer pointer
25         ;
26         001130  004737  001334'  CALL      MIOADV          ;Advance user's buffer pointer
27         ;
28         ; See if there is more data remaining to be read
29         ;
30         001134  005765  0000000  TST      MI$TRW(R5)      ;Is there more data to be read?
31         001140  001403          BEQ      1$              ;Br if not
32         ;
33         ; Initiate another mapped read operation
34         ;
35         001142  004737  001026'  CALL      MIORD          ;Start another mapped read
36         001146  000402          BR       9$
37         ;
38         ; The I/O operation is complete
39         ;
40         001150  004737  001156'  1$:     CALL      MIOFIN          ;Say I/O operation is finished
41         ;
42         ; Finished
43         ;
44         001154  000207          9$:     RETURN

```

MIOFIN -- Completion of a mapped I/O operation

```

1          .SBTTL  MIOFIN -- Completion of a mapped I/O operation
2          ;-----
3          ; MIOFIN is called when a mapped I/O operation is completed.
4          ;
5          ; Inputs:
6          ; R5 = Pointer to mapped I/O control block.
7          ;
8 001156 010446 MIOFIN: MOV      R4, -(SP)
9          ;
10         ; Get address of original I/O queue element.
11         ;
12 001160 016504 00000000      MOV      MI$QOE(R5), R4 ;Get address of original I/O queue element
13         ;
14         ; Free the mapped I/O data buffer and control block
15         ;
16 001164 004737 001606'      CALL     MIOFRE          ;Say mapped I/O is finished
17         ;
18         ; Do completion processing for original queue element
19         ;
20 001170 004737 00000000      CALL     IOCMPL          ;Do I/O completion
21         ;
22         ; Finished
23         ;
24 001174 012604      MOV      (SP)+, R4
25 001176 000207      RETURN

```

MIDGTQ -- Get I/O queue element for mapped I/O operation

```

1          .SBTTL  MIDGTQ -- Get I/O queue element for mapped I/O operation
2          ;-----
3          ; MIDGTQ is called to get an I/O queue element to use for a mapped
4          ; I/O operation.  In addition to getting a free queue element, this
5          ; routine also sets up the word count and copies other information
6          ; from the original queue element.
7          ;
8          ; Inputs:
9          ;   R5 = Pointer to mapped I/O control block.
10         ;
11         ; Outputs:
12         ;   R1 = Address of queue element.
13         ;
14 001200  010346  MIDGTQ:  MOV     R3, -(SP)
15 001202  010446          MOV     R4, -(SP)
16         ;
17         ; Get a free I/O queue element
18         ;
19 001204  016504  0000000  MOV     MI$OQE(R5), R4 ;Get address of original queue element
20 001210  016401  0000000  MOV     Q.UCSW(R4), R1 ;Get address of channel block
21 001214  004737  0000000  CALL    GETSYQ        ;Get a free I/O queue element for system use
22         ;
23         ; Copy information from original queue element to new queue element
24         ;
25 001220  010103          MOV     R1, R3          ;Get address of new queue element
26 001222  012700  0000000  MOV     #IQ$SIZ/2, R0 ;Get # words to copy
27 001226  012423  1$:    MOV     (R4)+, (R3)+ ;Copy contents of old queue element to new
28 001230  077002          SOB     R0, 1$
29         ;
30         ; Clear job number field of I/O queue so completion routine will
31         ; run in system state.
32         ;
33 001232  105061  0000000  CLRB   Q.JOB(R1)      ;Set job number = 0
34 001236  142761  000370  0000000  BICB   #370, Q.JNUM(R1) ;Set job number = 0
35         ;
36         ; Set control flag in I/O queue element indicating that this I/O
37         ; operation is a secondary operation for mapped I/O.
38         ;
39 001244  152761  0000000  0000000  BISB   #QF$MID, Q.FLAG(R1) ;Secondary op for mapped I/O
40         ;
41         ; Store the address of the mapped I/O control block into the cell of
42         ; the I/O queue element normally used for the channel number.
43         ; This will cause the address of the mapped I/O control block to be
44         ; passed to the completion routine in R1.
45         ;
46 001252  010561  0000000          MOV     R5, Q.CHAN(R1) ;Set address of control block as "channel #"
47         ;
48         ; Store actual word count
49         ;
50 001256  016504  0000000  MOV     MI$OQE(R5), R4 ;Get pointer to original Q element
51 001262  132764  0000000  0000000  BITB   #QF$OWC, Q.FLAG(R4) ;Should we always use original word count?
52 001270  001010          BNE    3$             ;Br if yes
53 001272  016500  0000000  MOV     MI$CWC(R5), R0 ;Get current word count for this transfer
54 001276  005764  0000000  TST    Q.WCNT(R4)     ;Was original count positive or negative?
55 001302  002001          BGE    2$             ;Br if positive
56 001304  005400          NEG    R0             ;Make count negative
57 001306  010061  0000000  2$:    MOV     R0, Q.WCNT(R1) ;Set word count in queue element

```

MIOGTQ -- Get I/O queue element for mapped I/O operation

```
58 ;  
59 ; Set the buffer address  
60 ;  
61 001312 012761 0000000 0000000 3#: MOV #VPAR6,Q.BUFF(R1);Set buffer virtual address  
62 001320 016561 0000000 0000000 MOV MI$SBP(R5),Q.PAR(R1) ;Set 64-byte block number of buffer base  
63 ;  
64 ; Finished  
65 ;  
66 001326 012604 MOV (SP)+,R4  
67 001330 012603 MOV (SP)+,R3  
68 001332 000207 RETURN
```

```

1          .SBTTL  MIOADV -- Advance buffer address and word count
2          ;-----
3          ; MIOADV is called to update the buffer address and word count following
4          ; a mapped I/O operation.
5          ;
6          ; Inputs:
7          ; R5 = Pointer to mapped I/O control block.
8          ;
9 001334 010246 MIOADV: MOV      R2,-(SP)
10 001336 010346      MOV      R3,-(SP)
11          ;
12          ; Advance buffer address
13          ;
14 001340 016502 0000000      MOV      MI$CWC(R5),R2      ;Get # words transferred this time
15 001344 006302              ASL      R2                      ;Convert to # bytes
16 001346 066502 0000000      ADD      MI$UBD(R5),R2      ;Compute new buffer offset
17 001352 005003              CLR      R3                      ;Clear for shift
18 001354 073227 177772      ASHC     #-6.,R2           ;Compute # 64-byte blocks & offset
19 001360 060265 0000000      ADD      R2,MI$UBP(R5)    ;Update 64-byte block base
20 001364 000303              SWAB     R3                      ;Get offset to low-order of R5
21 001366 072327 177776      ASH      #-2,R3           ;
22 001372 010365 0000000      MOV      R3,MI$UBD(R5)    ;New offset within 64-byte block
23          ;
24          ; Decrease number of words left to transfer
25          ;
26 001376 016502 0000000      MOV      MI$CWC(R5),R2    ;Get number of words transferred
27 001402 160265 0000000      SUB      R2,MI$TRW(R5)    ;Decrease number of words left
28          ;
29          ; Advance file block number
30          ;
31 001406 062702 000377      ADD      #255.,R2         ;Convert # words to # blocks
32 001412 072227 177770      ASH      #-8.,R2         ;
33 001416 016500 0000000      MOV      MI$OQE(R5),R0    ;Get pointer to original queue element
34 001422 060260 0000000      ADD      R2,Q.BLKN(R0)   ;Advance block number
35          ;
36          ; Finished
37          ;
38 001426 012603              MOV      (SP)+,R3
39 001430 012602              MOV      (SP)+,R2
40 001432 000207              RETURN

```

MIDGET -- Allocate a mapped I/O data buffer

```

1          .SBTTL  MIDGET -- Allocate a mapped I/O data buffer
2          ;-----
3          ; MIDGET is called to gain exclusive access to the I/O mapping data base.
4          ;
5          ; Inputs:
6          ; R1 = Address of I/O queue element
7          ;
8          ; Outputs:
9          ; R5 = Address of mapped I/O control block
10         ; C-flag set ==> Could not get a mapping buffer.
11         ; Mapping request has been queued for later.
12         ;
13 001434 010346 MIDGET: MOV      R3,-(SP)
14         ;
15         ; Get number of job that wants to do mapped I/O
16         ;
17 001436 116103 000000G      MOVB   Q.JOB(R1),R3      ;Get job number
18 001442 006303              ASL     R3                ;Convert to job index number
19         ;
20         ; See if there is a mapped I/O control block and data buffer available
21         ;
22 001444 1#:  DISABL              ;;;Disable interrupts
23 001452 013705 000000G      MOV     MIOBHD,R5          ;;;Is there a free buffer available?
24 001456 001040              BNE     2$                ;;;Br if yes
25         ;
26         ; There are no free mapping buffers.
27         ; Wait until one becomes free.
28         ;
29 001460 005703              TST     R3                ;;;Is system or user job doing I/O?
30 001462 001027              BNE     3$                ;;;Br if user job
31         ;
32         ; System is trying to do mapped I/O.
33         ; Queue a request for a mapping buffer and then return with
34         ; the C-flag set indicating that the operation has been deferred.
35         ;
36 001464 013700 000000G      MOV     MIOWHD,R0          ;;;Get address of free queue block
37 001470 001005              BNE     4$                ;;;We better get one
38 001472              DIE     #EM$MIO              ;;;Die if we cannot queue request
39 001504 016037 000000G 000000G 4$:  MOV     MW$LNK(R0),MIOWHD ;;;Remove block from free list
40 001512 010160 000000G      MOV     R1,MW$IOQ(R0) ;;;Save address of I/O queue element
41 001516 013760 000000G 000000G      MOV     MIOSYQ,MW$LNK(R0) ;;;Put wait block on active list
42 001524 010037 000000G      MOV     R0,MIOSYQ ;;;
43 001530              ENABL              ;Enable interrupts
44 001536 000261              SEC                ;Signal that operation is to be deferred
45 001540 000420              BR      9$
46         ;
47         ; Suspend a job until mapping data becomes free
48         ;
49 001542 012700 000000G 3$:  MOV     #$QMIO,R0          ;;;Waiting for mapped I/O
50 001546 004737 000000G      CALL   QNSPNX          ;;;Suspend job till mapping is free (Enable)
51 001552 004737 000000G      CALL   CHKABT          ;Was job aborted while it was asleep?
52 001556 000732              BR      1$                ;Go try to get the data base now
53         ;
54         ; There is a free mapping buffer.
55         ; Claim it for our job.
56         ;
57 001560 016537 000000G 000000G 2$:  MOV     MI$LNK(R5),MIOBHD; ;Remove control block from free list

```

```
58 001566          ENABL          ;Enable interrupts
59 001574  110365  0000000      MOVB      R3,MI#JOB(R5)  ;Save associated job number
60 001600  000241          CLC          ;Signal success on return
61          ;
62          ; Finished
63          ;
64 001602  012603      9#:  MOV      (SP)+,R3
65 001604  000207          RETURN
```

```

1          .SBTTL  MIOFRE -- Free a mapped I/O data buffer.
2          ;-----
3          ; MIOFRE is called to free a mapped I/O data buffer.
4          ;
5          ; Inputs:
6          ; R5 = Pointer to mapped I/O control block.
7          ;
8 001606 010146 MIOFRE: MOV     R1,-(SP)
9 001610          DISABL          ;;Disable interrupts
10         ;
11         ; Put control block back on free list
12         ;
13 001616 013765 0000000 0000000 MOV     MIOBHD,MI$LNK(R5);;Put control block back on free list
14 001624 010537 0000000          MOV     R5,MIOBHD          ;;
15         ;
16         ; See if there are any requests from the system for a mapping I/O buffer
17         ;
18 001630 013700 0000000          MOV     MIOSYQ,R0          ;;Is there a pending system request?
19 001634 001420          BEQ     1$          ;;Br if not
20         ;
21         ; There is a pending request from the system for a mapping buffer.
22         ; Start this request now.
23         ;
24 001636 016037 0000000 0000000 MOV     MW$LNK(R0),MIOSYQ;Remove wait queue element from active list
25 001644 016001 0000000          MOV     MW$IOQ(R0),R1          ;;Get address of original I/O queue element
26 001650 013760 0000000 0000000 MOV     MIOWHD,MW$LNK(R0);;Put wait queue element back on free list
27 001656 010037 0000000          MOV     R0,MIOWHD          ;;
28 001662          ENABL          ;Enable interrupts
29 001670 004737 000002'          CALL    MIOCHK          ;Initiate the mapped I/O operation
30 001674 000407          BR     9$
31         ;
32         ; Restart any jobs that are waiting for mapped I/O
33         ;
34 001676          1$: ENABL          ;;Enable interrupts
35 001704 012700 0000000          MOV     #$QMIO,R0          ;Get wait state code
36 001710 004737 0000000          CALL    UREGO          ;Restart any waiting jobs
37         ;
38         ; Finished
39         ;
40 001714 012601          9$: MOV     (SP)+,R1
41 001716 000207          RETURN
42         ;
43         .END

```

Errors detected: 0

*** Assembler statistics

Work file reads: 0
Work file writes: 0
Size of work file: 215 Words (1 Pages)
Size of core pool: 17920 Words (70 Pages)
Operating system: RT-11

Elapsed time: 00:00:16.30
DK: TSMIO, LP: TSMIO=DK: TSMIO. MAC/C/N: SYM

C.CSW	1-29	6-15	8-21							
CHKABT	1-25	12-51								
CHKBUF	2-71	2-88	2-100#	3-48	3-55	3-62	3-69			
CLRADR	3-17	3-18	3-20	3-24	3-25	3-27	3-31	3-32	3-34	3-73#
CDRUSR	1-26									
CS*EOF	1-29	6-15	8-21							
CS*ERR	1-29	6-15	8-21							
DEVEND	2-51	3-41#								
DEVTBL	2-46	3-12#								
DI#DX	1-29	3-38	3-39	3-40						
DI#DY	1-27	3-13	3-14	3-15						
DI#MM	1-27	3-24	3-25	3-26	3-27	3-28	3-29			
DI#MS	1-27	3-31	3-32	3-33	3-34	3-35	3-36			
DI#MT	1-29	3-17	3-18	3-19	3-20	3-21	3-22			
DIEARG	1-53									
DIEMSG	1-53	12-38#								
DS*ID	1-25	2-42								
DVSTAT	1-25	2-41								
DXREAD	3-38	3-59#								
DXWRIT	3-39	3-40	3-66#							
DYREAD	3-13	3-45#								
DYWRIT	3-14	3-15	3-52#							
EM*MIO	1-30	12-38								
GETSYQ	1-23	10-21								
INTPRI	1-26	12-43	12-58	13-28	13-34					
IDCMPL	1-26	9-20								
IDQSIZ	1-24	10-26								
MI#CWC	1-28	5-21*	7-21*	10-53	11-14	11-26				
MI#JOB	1-28	12-59*								
MI#LNK	1-27	12-57	13-13*							
MI#OQE	1-28	4-16*	6-13	8-19	9-12	10-19	10-50	11-33		
MI#RWF	1-28	4-18*	4-31							
MI#SBP	1-27	10-62								
MI#TRW	1-28	4-17*	5-15	6-20	7-15	8-30	11-27*			
MI#UBD	1-28	4-27*	11-16	11-22*						
MI#UBP	1-28	4-24*	11-19*							
MICADV	5-34	8-26	11-9#							
MIOBGN	2-122	4-11#								
MIOBHD	1-27	12-23	12-57*	13-13	13-14*					
MIOCHK	1-19	2-16#	13-29							
MIODBG	1-29	2-109								
MIOFIN	6-32	8-40	9-8#							
MIOFRE	9-16	13-8#								
MIOGET	2-117	12-13#								
MIOGTQ	5-29	7-25	10-14#							
MIOMRD	1-24	8-14								
MIOMWT	1-24	5-25								
MIORO	4-35	7-10#	8-35							
MIORDC	7-26	8-10#								
MIOSYQ	1-31	12-41	12-42*	13-18	13-24*					
MIOWHD	1-31	12-36	12-39*	13-26	13-27*					
MIOWRC	5-30	6-8#								
MIOWRT	4-33	5-10#	6-26							
MIOXIT	2-118	2-124	2-132#							
MW#IOQ	1-31	12-40*	13-25							
MW#LNK	1-31	12-39	12-41*	13-24	13-26*					

NOMAP	2-26	2-31	2-75	2-82	2-112	2-128#	3-75		
PSW	1-26	12-22*	12-43*	12-58*	13-9*	13-28*	13-34*		
Q. BLKN	1-26	11-34*							
Q. BUFF	1-23	2-100	3-73*	4-19	10-61*				
Q. CHAN	1-27	10-46*							
Q. COMP	1-23	5-30*	7-26*						
Q. CSW	1-23	6-14	8-20						
Q. DEVX	1-24	2-30							
Q. FLAG	1-30	2-25	2-70*	3-47*	3-54*	3-61*	3-68*	10-39*	10-51
Q. FUNC	1-23	2-35							
Q. JNUM	1-26	10-34*							
Q. JOB	1-26	10-33*	12-17						
Q. PAR	1-25	2-104	3-74*	4-23	10-62*				
Q. UCSW	1-31	10-20							
Q. WCNT	1-24	2-80	2-87	2-93	10-54	10-57*			
QF\$MID	1-30	2-25	10-39						
QF\$DWC	1-30	2-70	3-47	3-54	3-61	3-68	10-51		
QNSPNX	1-25	12-50							
RDFUN	2-62	2-86#	3-22	3-29	3-36				
S\$QMID	1-25	12-49	13-35						
SYQID	1-25	5-38	7-30						
SYSHLT	1-53	12-38							
TSMIO	1-6#								
UREGO	1-26	13-36							
VMIOSZ	1-25	5-16	7-16						
VPAR6	1-23	2-101	4-20	10-61					
WRTFUN	2-60	2-81	2-92#	3-19	3-21	3-26	3-28	3-33	3-35

