

Table of contents

2-	1	RTEMT	-- Real-time emt entry point
3-	1	RTCVAD, RTCVDA	-- Convert virtual to physical address
4-	1	RTVMAP	-- Map virtual address to physical region
5-	1	RTFREZ	-- Lock job in memory without repositioning
5-	11	RTUNLK	-- Unlock job from memory
6-	1	RTPEEK	-- Peek at I/O page
6-	15	RTPOKE	-- Poke into I/O page
6-	27	RTBIS	-- Bit set into I/O page
6-	39	RTBIC	-- Bit clear into I/O page
7-	1	RTIOMP	-- Map par7 to I/O page
7-	11	RTRMMP	-- Map par7 to simulated Rmon
8-	1	.SPND	-- Suspend job's execution
8-	27	.RSUM	-- Resume job's execution
9-	1	RTCMPL	-- Schedule a completion routine
10-	1	RTEXJB	-- Gain exclusive use of system
10-	17	RTALJB	-- Release exclusive access claim to system
11-	1	RTSPL	-- Set processor priority level
12-	1	RTCVEC	-- Connect completion routine to interrupt vector
13-	1	RTRVEC	-- Release interrupt vector
14-	1	RTDEV	-- .DEVICE Emt
15-	1	RTSTOP	-- Real-time cleanup at end of job execution
17-	1	** Subroutines **	
17-	2	PRVRT	-- Determine if job has real-time privilege
17-	13	PRVMEM	-- Check for access authorization to phys memory
18-	1	SRCVEC	-- Search for vector control block

```

1          .TITLE  TSRTX -- Non-resident Real-Time Support Module
2          .ENABL  LC
3          .DSABL  GBL

```

```

4          ;-----
5          ; TSRTX is the non-resident portion of the TSX-Plus real-time support
6          ; code.
7          ;
8          ; Copyright (c) 1980,1981,1982,1983,1984,1985,1986,1987,1988,1989.
9          ; S&H Computer Systems, Inc. Nashville, Tennessee
10         ; All rights reserved.
11         ;

```

```

12         .CSECT  TSRTX
13         000000 071670 TSRTX: .RAD50 /RTX/          ;Overlay id

```

```

14         ;
15         ; Macro definitions
16         ;
17         ; Macro to call a routine in another system overlay region.
18         ;

```

```

19         .MACRO  OCALL  ENTADD
20         .IF     B,ENTADD
21         .ERROR  ;OCALL without entry address
22         .ENDC
23         CALL   OVRHC
24         .WORD  ENTADD
25         .ENDM  OCALL

```

```

26         ;
27         ; Global definitions
28         ;
29         .GLOBL  RTSPND,RTRSUM
30         .GLOBL  RTSTOP,RTDEV,RTEMT

```

```

31         ;
32         ; Global references
33         ;
34         .GLOBL  LSW2, SETERR, EMTBLK, BADEMT, LPARBS, EMTXIT
35         .GLOBL  $IOMAP, LSW6, SETMAP, VC$JOB, VC$VEC, VC$RTN
36         .GLOBL  UEXINT, CQ$JOB, CQ$RO, CQ$RTN, QCOMPL
37         .GLOBL  URO, VC$PRI, EMTPS, INTPRI, RTLOCK
38         .GLOBL  S$TWFN, LBSPRI, $MLOCK, PO$LOK
39         .GLOBL  OVRHC, UEXRTN, VUXIFL
40         .GLOBL  VCBAS, VCBEND, VC$$SZ, PO$MEM, PO$RT, PRIVCO
41         .GLOBL  LSPND, CURCP, S$SPND, CHKABT
42         .GLOBL  QHDSFN, DEVL, DEVL, CQ$PRI, CQ$PA5, CQ$RNS
43         .GLOBL  DOSCHD, EXCJOB, CINFLG
44         .GLOBL  RPAR, RPDR, RDAR, RDDR, VF$DIR, VC$FLG, VF$DET
45         .GLOBL  UPARO, UPDRO, CUPARO, CUPDRO
46         .GLOBL  UPMODE, UPMODE, KPAR5, CQ$CP, CP$STD, CP$RT
47         .GLOBL  CUPARO, CUPDRO, CQ$R1, $UDSPC, LSW11
48         .GLOBL  GETQ, CORUSR, MAXPRI, S$RT, VPRIHI

```

```

49         ;
50         ;-----
51         ; MACROS TO ENABLE AND DISABLE INTERRUPTS.
52         ;

```

```

53         PS      =      177776          ;PROCESSOR STATUS WORD
54         .MACRO  DISABL          ;DISABLE INTERRUPTS
55         BIS     #340, @#PS
56         .ENDM  DISABL
57         ;

```

177776

58
59
60

```
.MACRO ENABL ;ENABLE INTERRUPTS  
BIC INTPRI,@#PS  
.ENDM ENABL
```

RTEMT -- Real-time emt entry point

```

1          .SBTTL RTEMT -- Real-time emt entry point
2          ;-----
3          ; RTEMT is jumped to from TSEMT whenever one of the real-time emt's
4          ; is executed.
5          ; The general form of a real-time emt is
6          ;
7          ; .BYTE sub-function,140
8          ; .WORD Arg1 ;(optional)
9          ; .WORD Arg2 ;(optional)
10         ; .WORD Arg3 ;(optional)
11         ;
12         ; Where sub-function is a code that indicates which real-time
13         ; function is to be performed. (See vector at end of this routine)
14         ;
15         ; See if user is authorized to use real-time emt's.
16         ;
17 000002 RTEMT:
18         ;
19         ; Get sub-function code from arg block and jump to processing routine
20         ;
21 000002 116705 000000G 1$: MOVB EMTBLK,R5 ;GET SUB-FUNCTION CODE
22 000006 006305 ; ASL R5 ;CONVERT TO WORD TABLE INDEX
23 000010 020527 000044 ; CMP R5,#MXRTFN ;IS IT VALID SUB-FUNCTION CODE?
24 000014 101402 ; BLOS 2$ ;BR IF OK
25 000016 000167 000000G ; JMP BADEMT ;INVALID SUB-FUNCTION VALUE
26 000022 000175 000026' 2$: JMP @RTEMTV(R5) ;JUMP TO PROCESSING ROUTINE FOR EMT
27         ;
28         ;-----
29         ; Define sub-function jump vector.
30         ;
31 000026 000124' RTEMTV: .WORD RTCVAD ;00 - Convert virtual address to physical
32 000030 000516' .WORD RTPEEK ;01 - Peek at I/O page
33 000032 000532' .WORD RTPOKE ;02 - Poke into I/O page
34 000034 000546' .WORD RTBIS ;03 - Bit set into I/O page
35 000036 000562' .WORD RTBIC ;04 - Bit clear into I/O page
36 000040 000600' .WORD RTIOMP ;05 - Map virtual PAR6 to I/O page
37 000042 000620' .WORD RTRMMP ;06 - Map virtual PAR6 to simulated RMON
38 000044 000000G .WORD RTLOCK ;07 - Lock job in low memory
39 000046 000504' .WORD RTUNLK ;10 - Unlock job from memory
40 000050 001204' .WORD RTCVEC ;11 - Connect completion routine to interrupt
41 000052 001334' .WORD RTRVEC ;12 - Disconnect completion rtn from interrupt
42 000054 000460' .WORD RTFREQ ;13 - Lock job in memory without repositioning
43 000056 001064' .WORD RTEXJB ;14 - Gain exclusive use of system
44 000060 001100' .WORD RTALJB ;15 - Release exclusive use of system
45 000062 001114' .WORD RTSPL ;16 - Set processor priority level level
46 000064 000232' .WORD RTVMAP ;17 - Map virtual region to physical region
47 000066 001152' .WORD RTDVEC ;20 - Connect direct interrupt service routine
48 000070 000720' .WORD RTCMPL ;21 - Schedule completion routine
49 000072 000074' .WORD RTCVDA ;22 - Convert D-space address to physical
50         ;
51         000044 MXRTFN = .-RTEMTV-2 ;2 * Highest legal subfunction number

```

RTCVAD, RTCVDA -- Convert virtual to physical address

```

1          .SBTTL  RTCVAD, RTCVDA -- Convert virtual to physical address
2          ;-----
3          ; The RTCVAD and RTCVDA emts are used to convert a 16-bit virtual address
4          ; to a 22-bit physical address.  RTCVAD for I-space and RTCVDA for D-space.
5          ;
6          ; Inputs:
7          ;   Arg1 = Virtual address
8          ;   Arg2 = Address of 2-word block to receive 22-bit address result
9          ;
10         ; Outputs:
11         ;   Result buffer in user's area receives 22-bit physical address
12         ;   low-order 16-bits stored in 1st word and high-order 2-bits positioned
13         ;   in bit positions 4-5 are stored in 2nd word.
14         ;
15         ; Convert specially mapped D-space (22,140)
16         ;
17 000074 032761 000000G 000000G RTCVDA: BIT      #$UDSPC,LSW11(R1) ;Has user enabled D-space?
18 000102 001410          BEQ      RTCVAD      ;If not, use I-space routine
19 000104 004767 000102          CALL     RTCVC1      ;Split virtual address to par and offset
20 000110 005762 000000G          TST      RDDR(R2)      ;Is this region specially mapped?
21 000114 001415          BEQ      RTCVC3      ;Br if not, use default mapping
22 000116 016205 000000G          MOV      RDAR(R2),R5    ;Get physical address for this par
23 000122 000407          BR       RTCVC2      ;Br to special mapping adjustment
24         ;
25         ; Convert specially mapped I-space (0,140)
26         ;
27 000124 004767 000062          RTCVAD: CALL     RTCVC1      ;Split virtual address to par and offset
28 000130 005762 000000G          TST      RPDR(R2)      ;IS THIS REGION SPECIALLY MAPPED?
29 000134 001405          BEQ      RTCVC3      ;BR IF NOT
30 000136 016205 000000G          MOV      RPAR(R2),R5    ;GET PHYSICAL ADDRESS FOR THIS PAR
31 000142 042700 160000          RTCVC2: BIC      #160000,R0    ;Remove PAR # from virtual address
32 000146 000402          BR       RTCVC4      ;Go merge back with offset and return
33         ;
34         ; No special mapping, treat as linear offset from job base
35         ;
36 000150 016105 000000G          RTCVC3: MOV      LPARBS(R1),R5 ;GET BASE 64-BYTE BLOCK NUMBER OF JOB REGION
37         ;
38         ; PAR base in R5, merge back with offset and return to user
39         ;
40 000154 005004          RTCVC4: CLR      R4          ;CLEAR HIGH-ORDER WORD
41 000156 073427 000006          ASHC     #6,R4          ;CONVERT TO 22-BIT PHYSICAL ADDR OF JOB BASE
42 000162 060005          ADD      R0,R5          ;ADD VIRTUAL ADDRESS offset
43 000164 005504          ADC      R4          ;PROPOGATE CARRY
44 000166 072427 000004          ASH      #4,R4          ;PUT 2 HIGH-ORDER BITS IN BIT POSITIONS 4-5
45 000172 016703 000004G          MOV      EMTBLK+4,R3    ;GET ADDRESS OF BUFFER WHERE RESULT GOES
46 000176 010546          MOV      R5,-(SP)        ;STACK LOW-ORDER PART OF RESULT
47 000200 106623          MTPD     (R3)+          ;STORE INTO USER'S BUFFER
48 000202 010446          MOV      R4,-(SP)        ;STACK HIGH-ORDER PART OF RESULT
49 000204 106613          MTPD     (R3)          ;STORE INTO USER'S BUFFER
50 000206 000167 000000G          JMP      EMTXIT        ;FINISHED
51         ;
52         ; Common subroutine for virtual to physical address conversions
53         ; Converts user virtual address (from EMTBLK+2) to PAR index in R2
54         ;
55 000212 016700 000002G          RTCVC1: MOV      EMTBLK+2,R0 ;GET VIRTUAL ADDRESS
56 000216 010003          MOV      R0,R3          ;Copy to low order of R2-R3 pair
57 000220 005002          CLR      R2          ;SET UP FOR SHIFT

```

58	000222	073227	000003	ASHC	#3,R2	;GET 3 HIGH-ORDER BITS OF VIRTUAL ADDRESS
59	000226	006302		ASL	R2	;*2 TO CONVERT TO WORD TABLE INDEX
60	000230	000207		RETURN		

RTVMAP -- Map virtual address to physical region

```

1          .SBTTL  RTVMAP -- Map virtual address to physical region
2          ;-----
3          ; This EMT is used to map a virtual region of memory to a specified
4          ; physical region.
5          ;
6          ; .BYTE 17,140
7          ; .WORD Base PAR # (0-7)
8          ; .WORD Physical base (64-byte block #)
9          ; .WORD Region length (# 64-byte blocks)
10         ; .BYTE Access control (0==>read only, 1==>read/write)
11         ; .BYTE Cache control (0==>enable caching, 1==>bypass cache)
12         ;
13 000232 004767 001534 RTVMAP: CALL PRVMEM ; Can we access physical memory?
14 000236 016705 000006G MOV EMTBLK+6,R5 ; GET SIZE OF REGION
15 000242 001016 BNE 10$ ; BR IF MAPPING NEW REGION
16         ;
17         ; Reset all mapping for job
18         ;
19 000244 005002 CLR R2
20 000246 005062 000000G 11$: CLR RPAR(R2) ; CLEAR PAR
21 000252 005062 000000G CLR RPDR(R2) ; AND PDR
22 000256 062702 000002 ADD #2,R2
23 000262 020227 000016 CMP R2,#14 ; DONE ALL?
24 000266 101767 BLOS 11$ ; BR IF MORE TO DO
25 000270 004767 000000G CALL SETMAP ; RESET MAPPING FOR JOB
26 000274 000167 000000G JMP EMTXIT ; FINISHED
27         ;
28         ; Map new virtual address to physical region
29         ;
30 000300 016702 000002G 10$: MOV EMTBLK+2,R2 ; GET PAR #
31 000304 020227 000007 CMP R2,#7 ; MUST BE IN RANGE 0-7
32 000310 101404 BLOS 1$ ; BR IF OK
33 000312 012700 177767 MOV #-11,R0 ; ABORT IF TOO LARGE
34 000316 000167 000000G JMP SETERR
35 000322 006302 1$: ASL R2 ; CONVERT TO WORD TABLE INDEX
36 000324 016704 000004G MOV EMTBLK+4,R4 ; GET PHYSICAL ADDRESS BASE
37         ;
38         ; Set up mapping for each PAR that is affected by this request.
39         ; R2 = 2*PAR #
40         ; R4 = Physical address base
41         ; R5 = Size of region being mapped
42         ;
43 000330 010500 2$: MOV R5,R0 ; GET # 64-BYTE BLOCKS LEFT TO MAP
44 000332 001450 BEQ 5$ ; BR IF FINISHED
45 000334 020027 000200 CMP R0,#200 ; EACH PAR CAN ONLY MAP 200 BLOCKS
46 000340 101402 BLOS 3$ ; BR IF OK
47 000342 012700 000200 MOV #200,R0 ; MAP 200 BLOCKS THROUGH THIS PAR
48 000346 010462 000000G 3$: MOV R4,RPAR(R2) ; SET PAR BASE ADDRESS
49 000352 010462 000000G MOV R4,UPARO(R2) ; LOAD HARDWARE MAP REGISTER
50 000356 010462 000000G MOV R4,CUPARO(R2) ; SAVE MAPPING INFO IN CONTEXT BLOCK
51 000362 060004 ADD R0,R4 ; ADVANCE PAR BASE ADDRESS
52 000364 160005 SUB R0,R5 ; DECREASE # BLOCKS REMAINING TO BE MAPPED
53 000366 005300 DEC R0 ; PDR VALUE IS ACTUAL-1
54 000370 000300 SWAB R0 ; POSITION SIZE FOR PLF FIELD IN PDR
55 000372 042700 100377 BIC #^C<77400>,R0 ; CLEAR ALL BUT PLF FIELD IN PDR VALUE
56 000376 052700 000002 BIS #2,R0 ; SET READ-ENABLE FLAG FOR PDR
57 000402 105767 000010G TSTB EMTBLK+10 ; IS WRITE ACCESS ALLOWED?

```

RTVMAP -- Map virtual address to physical region

```

58 000406 001402          BEQ      4$          ; BR IF NOT
59 000410 052700 000004    BIS      #4, R0        ; SET WRITE-ENABLE FLAG FOR PDR
60 000414 105767 000011G   4$:    TSTB    EMTBLK+11 ; BYPASS CACHE?
61 000420 001402          BEQ      6$          ; BR IF NOT
62 000422 052700 100000    BIS      #100000, R0   ; SET CACHE-BYPASS FLAG
63 000426 010062 000000G   6$:    MOV      R0, RPDR(R2) ; SET THE PDR VALUE
64 000432 010062 000000G   MOV      R0, UPDR0(R2) ; LOAD HARDWARE MAP REGISTER
65 000436 010062 000000G   MOV      R0, CUPDR0(R2) ; SAVE MAPPING INFORMATION IN JOB CONTEXT BLOCK
66 000442 062702 000002    ADD      #2, R2        ; ADVANCE PAR INDEX #
67 000446 020227 000016    CMP      R2, #2*7     ; ONLY DO UP TO PAR # 7
68 000452 101726          BLOS    2$          ; LOOP IF MORE PAR'S TO MAP
69
70                          ; Finished
71
72 000454 000167 000000G   5$:    JMP      EMTXIT    ; FINISHED

```

RTFREZ -- Lock job in memory without repositioning

```

1          .SBTTL RTFREZ -- Lock job in memory without repositioning
2          ;-----
3          ; The RTFREZ emt locked a job in memory without repositioning it.
4          ;
5 000460 032767 000000G 000000G RTFREZ: BIT    #PO$LOK,PRIVCO ;Do we have privilege to do this?
6 000466 001001          BNE     20$          ;Br if yes
7 000470 005000          CLR     RO          ;Return error code 0
8 000472 052761 000000G 000000G 20$:  BIS    #MLOCK,LSW6(R1);LOCK JOB IN MEMORY
9 000500 000167 000000G          JMP     EMTXIT          ;FINISHED

```

```

10
11         .SBTTL RTUNLK -- Unlock job from memory
12         ;-----
13         ; The RTUNLK emt unlockes a job from memory.
14         ;
15 000504 042761 000000G 000000G RTUNLK: BIC    #MLOCK,LSW6(R1);UNLOCK FROM MEMORY
16 000512 000167 000000G          JMP     EMTXIT          ;FINISHED

```

RTPEEK -- Peek at I/O page

```

1          .SBTTL  RTPEEK -- Peek at I/O page
2          ;-----
3          ; RTPEEK emt is used to access a single word in the I/O page.
4          ;
5          ; Inputs:
6          ;   Arg1 = Address of cell in I/O page to be accessed.
7          ;
8          ; Outputs:
9          ;   Contents of cell are returned to user in R0.
10         ;
11 000516 004767 001250 RTPEEK: CALL  PRVMEM          ;Can we access physical memory?
12 000522 017767 000002G 000000G MOV   @EMTBLK+2,URO ;PEEK AT I/O PAGE
13 000530 000421        BR     RTXIT

```

```

14         .SBTTL  RTPOKE -- Poke into I/O page
15         ;-----
16         ; RTPOKE emt is used to store a word into a cell in the I/O page.
17         ;
18         ; Inputs:
19         ;   Arg1 = Address of cell in I/O page to be accessed.
20         ;   Arg2 = Data value to be stored into I/O page cell.
21         ;
22         ;
23 000532 004767 001234 RTPOKE: CALL  PRVMEM          ;Can we access physical memory?
24 000536 016777 000004G 000002G MOV   EMTBLK+4,@EMTBLK+2;STORE INTO CELL IN I/O PAGE
25 000544 000413        BR     RTXIT

```

```

26         .SBTTL  RTBIS  -- Bit set into I/O page
27         ;-----
28         ; RTBIS emt is used to do a bit set into a word in the I/O page.
29         ;
30         ; Inputs:
31         ;   Arg1 = Address of cell in I/O page to be accessed.
32         ;   Arg2 = Data value to be ORed into I/O cell.
33         ;
34         ;
35 000546 004767 001220 RTBIS:  CALL  PRVMEM          ;Can we access physical memory?
36 000552 056777 000004G 000002G BIS   EMTBLK+4,@EMTBLK+2;DO BIS INTO I/O PAGE CELL
37 000560 000405        BR     RTXIT

```

```

38         .SBTTL  RTBIC  -- Bit clear into I/O page
39         ;-----
40         ; RTBIC emt is used to do a bit clear into a word in the I/O page.
41         ;
42         ; Inputs:
43         ;   Arg1 = Address of cell in I/O page to be accessed.
44         ;   Arg2 = Data value to be used as a bit clear mask.
45         ;
46         ;
47 000562 004767 001204 RTBIC:  CALL  PRVMEM          ;Can we access physical memory?
48 000566 046777 000004G 000002G BIC   EMTBLK+4,@EMTBLK+2;DO BIC INTO I/O PAGE CELL
49 000574 000167 000000G RTXIT:  JMP   EMTXIT          ;FINISHED WITH EMT

```

RTIOMP -- Map par7 to I/O page

```

1          .SBTTL RTIOMP -- Map par7 to I/O page
2          ;-----
3          ; RTIOMP is called to map the upper 8Kb (par7) of the job's virtual
4          ; address space to the I/O page.
5          ;
6 000600 004767 001166 RTIOMP: CALL PRVMEM          ;Can we access I/O page?
7 000604 052761 000000G 000000G BIS          #$IOMAP,LSW6(R1);SET FLAG SAYING WE ARE ACCESSING I/O PAGE
8 000612 004767 000000G RTMAP: CALL SETMAP          ;RELOAD MAPPING REGISTERS FOR JOB
9 000616 000766 BR          RTXIT
10
11         .SBTTL RTRMMP -- Map par7 to simulated Rmon
12         ;-----
13         ; RTRMMP emt is used to map the par7 region of the user's job back to
14         ; the simulated Rmon.
15         ;
16 000620 042761 000000G 000000G RTRMMP: BIC          #$IOMAP,LSW6(R1);SAY WE ARE NO LONGER MAPPED TO I/O PAGE
17 000626 000771 BR          RTMAP          ;RELOAD MAPPING REGISTERS FOR JOB

```

.SPND -- Suspend job's execution

```

1                                     .SBTTL .SPND -- Suspend job's execution
2                                     ;-----
3                                     ; The .SPND emt is used to suspend the job's execution until a .RSUM
4                                     ; is done.
5                                     ;
6 000630 005361 000000G RTSPND: DEC      LSPND(R1)      ;DEC SUSPENSION COUNTER FOR JOB
7 000634 002023          BGE      1$              ;BR IF WE SHOULD NOT SUSPEND JOB
8                                     ; Don't suspend job if running in a completion routine.
9 000636 105767 000000G      TSTB     CURCP        ;ARE WE RUNNING IN A COMPLETION ROUTINE NOW?
10 000642 001020          BNE      1$              ;BR IF YES
11                                     ;
12                                     ; Suspend execution of the job until a .RSUM emt is done.
13                                     ;
14 000644          3$:  DISABL          ;;;DISABLE INTERRUPTS
15 000652 005761 000000G      TST      LSPND(R1)      ;;;HAS RESUME BEEN ISSUED?
16 000656 002007          BGE      4$              ;;;BR IF WE SHOULD RESTART JOB
17 000660 012700 000000G      MOV      #S$SPND,RO      ;;;JOB SUSPENDED STATE
18 000664 004767 000000G      CALL     QHDSPN        ;;;SUSPEND THE JOB ** ENABLES INTERRUPTS **
19 000670 004767 000000G      CALL     CHKABT        ;SEE IF WE WERE ABORTED WHILE ASLEEP
20 000674 000763          BR       3$              ;ATTEMPT TO RESUME THE JOB
21                                     ;
22                                     ; Finished doing the .spnd -- resume the job's execution.
23                                     ;
24 000676          4$:  ENABL          ;;;ENABLE INTERRUPTS
25 000704 000167 000000G      1$:  JMP      EMTXIT        ;FINISHED WITH .SPND
26
27                                     .SBTTL .RSUM -- Resume job's execution
28                                     ;-----
29                                     ; The .RSUM emt is used to resume execution of a job that is asleep
30                                     ; because of doing a .SPND.
31                                     ;
32 000710 005261 000000G RTRSUM: INC      LSPND(R1)      ;INC JOB'S .SPND COUNTER
33 000714 000167 000000G      JMP      EMTXIT        ;THAT'S ALL WE HAVE TO DO

```

RTCMPL -- Schedule a completion routine

```

1          .SBTTL  RTCMPL -- Schedule a completion routine
2          ;-----
3          ; The RTCMPL EMT is used to schedule a completion routine for the job.
4          ;
5          ; The form of the EMT is
6          ;
7          ;     EMT      375
8          ;
9          ; With RO pointing to the following EMT argument block:
10         ;
11         ;     .BYTE   21,140
12         ;     .WORD   completion_routine_address
13         ;     .WORD   completion_routine_priority
14         ;     .WORD   value_to_pass_in_R1
15         ;     .WORD   0
16         ;
17 000720  RTCMPL:
18         ;
19         ; Get a completion routine queue element
20         ;
21 000720  004767  000000G      CALL      GETQ          ;Get a free completion queue element
22         ;
23         ; Set up the queue element
24         ;
25 000724  010104              MOV      R1,R4          ;Carry queue element pointer in R4
26 000726  116701  000000G     MOVB     CORUSR,R1      ;Get job index number
27 000732  110164  000000G     MOVB     R1,CQ$JOB(R4) ;Set job number in queue element
28 000736  012703  000002G     MOV      #EMTBLK+2,R3 ;Get pointer to block with EMT arguments
29 000742  012364  000000G     MOV      (R3)+,CQ$RTN(R4);Set address of completion routine
30 000746  012302              MOV      (R3)+,R2      ;Get execution priority value
31 000750  001012              BNE     2$           ;Br if priority not = 0
32 000752  112764  000000G  000000G  MOVB     #S$TWFN,CQ$RNS(R4);Set non-real-time execution state
33 000760  116164  000000G  000000G  MOVB     LBSPRI(R1),CQ$PRI(R4);Set execution priority
34 000766  112764  000000G  000000G  MOVB     #CP$STD,CQ$CP(R4);Set standard compl rtn class priority
35 000774  000420              BR      4$
36 000776  112764  000000G  000000G  2$:    MOVB     #CP$RT,CQ$CP(R4);Set real-time completion class priority
37 001004  112764  000000G  000000G  MOVB     #S$RT,CQ$RNS(R4);Set real-time execution state
38 001012  116700  000000G     MOVB     VPRIHI,RO    ;Get base real-time priority
39 001016  060002              ADD     RO,R2         ;Add base real-time priority
40 001020  020227  000000G     CMP     R2,#MAXPRI   ;Make sure we don't overflow
41 001024  101402              BLOS   3$           ;Br if ok
42 001026  012702  000000G     MOV     #MAXPRI,R2   ;Truncate to max allowed
43 001032  110264  000000G     3$:    MOVB     R2,CQ$PRI(R4) ;Set execution priority value
44 001036  012364  000000G     4$:    MOV     (R3)+,CQ$R1(R4) ;Set value to pass to completion routine in R1
45 001042  005064  000000G     CLR     CQ$RO(R4)    ;Pass 0 in RO
46 001046  013764  000000G  000000G  MOV     @#KPAR5,CQ$PA5(R4);Save current kernel PAR 5 mapping
47         ;
48         ; Queue the completion routine request
49         ;
50 001054  004767  000000G      CALL     QCOMPL       ;Queue the completion routine
51         ;
52         ; Finished
53         ;
54 001060  000167  000000G      JMP     EMTXIT       ;Finished with EMT

```

RTEXJB -- Gain exclusive use of system

```

1          .SBTTL  RTEXJB -- Gain exclusive use of system
2          ;-----
3          ; The RTEXJB EMT is used to obtain exclusive use of the system
4          ; by a real-time job. All other jobs are suspended until the RTALJB
5          ; EMT is used to release exclusive access.
6          ;
7 001064 004767 000662 RTEXJB: CALL  PRVRT          ;Do we have Real-time privilege?
8          ;
9          ; Set flag that says this job has exclusive access to the system
10         ;
11 001070 110167 000000G          MOVB  R1,EXCJOB          ;SAY THIS JOB HAS EXCLUSIVE ACCESS
12         ;
13         ; Finished
14         ;
15 001074 000167 000000G          JMP    EMTXIT          ;EXIT FROM EMT
16
17         .SBTTL  RTALJB -- Release exclusive access claim to system
18         ;-----
19         ; The RTALJB emt is used to release exclusive access to the system.
20         ;
21 001100 105067 000000G RTALJB: CLRB  EXCJOB          ;SAY WE NO LONGER HAVE EXCLUSIVE ACCESS
22         ;
23         ; Set flag to cause execution scheduler to be called
24         ;
25 001104 105267 000000G          INCB  DOSCHD          ;CALL THE SCHEDULER LATER
26         ;
27         ; Finished
28         ;
29 001110 000167 000000G          JMP    EMTXIT          ;EXIT FROM EMT

```

RTSPL -- Set processor priority level

```

1          .SBTTL  RTSPL  -- Set processor priority level
2          ;-----
3          ; The RTSPL EMT is used to set the processor priority level that
4          ; will be used while running in user mode.
5          ;
6          ; Inputs:
7          ;   Arg1 = Processor priority level (0 to 7).
8          ;
9 001114 004767 000632      RTSPL:  CALL   PRVRT      ;Do we have real-time privilege?
10 001120 016702 000002G    MOV     EMTBLK+2,R2    ;GET REQUESTED PROCESSOR PRIORITY LEVEL
11 001124 042702 177770    BIC     #177770,R2    ;CLEAR ALL BUT PRIORITY LEVEL FIELD
12 001130 072227 000005    ASH     #5,R2      ;POSITION TO PRIORITY FIELD IN PS
13 001134 042767 000340 000000G  BIC     #340,EMTPS   ;CLEAR OLD PRIORITY FIELD IN PS
14 001142 050267 000000G    BIS     R2,EMTPS     ;STORE NEW PRIORITY LEVEL INTO PS
15 001146 000167 000000G    JMP     EMTXIT      ;DO EMT EXIT WHICH WILL LOAD PS

```

RTCVEC -- Connect completion routine to interrupt vector

```

1          .SBTTL  RTCVEC -- Connect completion routine to interrupt vector
2          ;-----
3          ; RTCVEC emt is used to connect an interrupt vector to a user completion
4          ; routine.
5          ;
6          ; Inputs:
7          ; Arg1 = Address of interrupt vector.
8          ; Arg2 = Address of completion routine.
9          ; Arg3 = Priority of completion routine (0 to 7).
10         ;
11         ; Entry point for directly connected interrupt service routine
12         ;
13 001152 004767 000574 RTDVEC: CALL  PRVRT          ; Do we have real-time privilege?
14 001156 032761 000000G 000000G BIT    %%MLOCK,LSW6(R1); IS JOB LOCKED IN MEMORY?
15 001164 001004          BNE    1$          ; BR IF YES
16 001166 012700 000003      MOV    #3,R0          ; RETURN EMT ERROR CODE 3 IF NOT
17 001172 000167 000000G      JMP    SETERR
18 001176 012705 000000G 1$:    MOV    #VF$DIR,R5      ; SET DIRECTLY-CONNECTED INTERRUPT FLAG
19 001202 000401          BR     RTXVEC          ; ENTER COMMON CODE
20         ;
21         ; Entry point for completion routine type service routine
22         ;
23 001204 005005 RTDVEC: CLR    R5          ; CLEAR DIRECTLY-CONNECTED INTERRUPT FLAG
24         ;
25         ; See if interrupt vector is already connected to a service routine
26         ;
27 001206 016700 000002G RTXVEC: MOV    EMTBLK+2,R0      ; GET ADDRESS OF VECTOR WE ARE CONNECTING TO
28 001212 006200          ASR    R0          ; DROP LOW-ORDER BIT
29 001214 010003          MOV    R0,R3          ; SAVE VECTOR VALUE
30 001216 004767 000570          CALL  SRCVEC          ; SEE IF THERE IS A VECTOR CONTROL BLOCK ALREADY
31 001222 103407          BCS   1$          ; BR IF NO CONTROL BLOCK FOR THIS VECTOR
32 001224 126201 000000G      CMPB  VC$JOB(R2),R1      ; IS OUR JOB THE ONE THAT OWNS THE VECTOR?
33 001230 001414          BEQ   2$          ; BR IF YES
34 001232 012700 000002      MOV    #2,R0          ; ERROR CODE 2 MEANS SOMEONE ELSE OWNS VECTOR
35 001236 000167 000000G      JMP    SETERR
36         ;
37         ; Vector is free. Get a vector control block and set it up.
38         ;
39 001242 005000 1$:    CLR    R0          ; GET A FREE VECTOR CONTROL BLOCK
40 001244 004767 000542      CALL  SRCVEC
41 001250 103004          BCC   2$          ; BR IF GOT ONE
42 001252 012700 000001      MOV    #1,R0          ; NO FREE VECTOR CONTROL BLOCKS
43 001256 000167 000000G      JMP    SETERR
44         ;
45         ; Found a free vector control block (Address is in R2).
46         ; Set it up.
47         ;
48 001262 110162 000000G 2$:    MOVB  R1,VC$JOB(R2)      ; SET JOB NUMBER
49 001266 110362 000000G      MOVB  R3,VC$VEC(R2)      ; SET ADDRESS OF VECTOR
50 001272 110562 000000G      MOVB  R5,VC$FLG(R2)     ; SET CONTROL FLAGS (VF$xxx)
51 001276 016762 000004G 000000G      MOV   EMTBLK+4,VC$RTN(R2); SET ADDRESS OF COMPLETION ROUTINE
52 001304 016700 000006G      MOV   EMTBLK+6,R0      ; GET COMPLETION ROUTINE PRIORITY
53 001310 042700 177770      BIC   #^C7,R0         ; FORCE TO RANGE 0-7
54 001314 110062 000000G      MOVB  R0,VC$PRI(R2)     ; SAVE PRIORITY IN VECTOR CONTROL BLOCK
55         ;
56         ; Make vector point to vector control block.
57         ;

```

RTCVEC -- Connect completion routine to interrupt vector

```
58 001320 006303          ASL   R3           ;GET ADDRESS OF VECTOR
59 001322 010223          MOV   R2,(R3)+      ;SET VECTOR PC -- SEND INT TO INT CONTROL BLOCK
60 001324 012713 000340   MOV   #340,(R3)     ;SET VECTOR PS
61                               ;
62                               ; Finished
63                               ;
64 001330 000167 000000G   JMP   EMTXIT
```

RTRVEC -- Release interrupt vector

```

1          .SBTTL  RTRVEC -- Release interrupt vector
2          ;-----
3          ; RTRVEC emt is used to release an interrupt vector connection.
4          ;
5          ; Inputs:
6          ; Arg1 = Address of vector.
7          ;
8 001334 004767 000412 RTRVEC: CALL PRVRT          ;Do we have real-time privilege?
9 001340 016700 000002G MOV EMTBLK+2,R0      ;GET ADDRESS OF VECTOR
10 001344 006200 ASR R0          ;DROP LOW-ORDER BIT
11 001346 004767 000440 CALL SRCVEC        ;SEARCH FOR VECTOR CONTROL BLOCK
12 001352 103405 BCS 1$          ;BR IF NONE FOR THIS VECTOR
13 001354 120162 000000G CMPB R1,VC$JOB(R2) ;DO WE OWN THE VECTOR?
14 001360 001002 BNE 1$          ;BR IF NOT
15 001362 004767 000004 CALL RELVEC        ;RELEASE THE VECTOR
16 001366 000167 000000G 1$: JMP EMTXIT
17          ;-----
18          ; RELVEC is called to release a user's link with an interrupt vector.
19          ; The vector is set to point to the unexpected interrupt routine and
20          ; the vector control block is freed.
21          ;
22          ; Inputs:
23          ; R2 = Address of vector control block.
24          ;
25          ;
26 001372 010346 RELVEC: MOV R3,-(SP)
27          ;
28          ; If .DEVICE list already specified a reset address for this vector,
29          ; don't make it point to normal unexpected-interrupt location.
30          ;
31 001374 132762 000000G 000000G BITB #VF$DET,VC$FLG(R2);Did .DEVICE disconnect the interrupt?
32 001402 001031 BNE 2$          ;Br if yes
33          ;
34          ; Set interrupt vector to point to unexpected interrupt routine.
35          ;
36 001404 005003 CLR R3          ;GET ADDRESS OF INT VECTOR FROM VECTOR CONTROL
37 001406 156203 000000G BISB VC$VEC(R2),R3
38 001412 006303 ASL R3          ;CONVERT TO REAL ADDRESS
39 001414 010300 MOV R3,R0
40 001416 105767 000000G TSTB VUXIFL        ;SHOULD WE CRASH OR IGNORE UNEXPECTED INTS?
41 001422 001005 BNE 1$          ;BR IF WE SHOULD CRASH
42 001424 012720 000000G MOV #UEXRTN,(R0)+ ;SET PC TO ROUTINE TO IGNORE INTS
43 001430 012710 000340 MOV #340,(R0)     ;SET PS PRIO=7 FOR INTERRUPT
44 001434 000414 BR 2$
45 001436 012720 000000G 1$: MOV #UEXINT,(R0)+ ;SET VECTOR TO GO TO UNEXPECTED INT ROUTINE
46          ;
47          ; Set PS in interrupt vector to encode interrupt address.
48          ;
49 001442 072327 177776 ASH #-2,R3      ;DROP LOW-ORDER 2 BITS OF VECTOR ADDRESS
50 001446 010346 MOV R3,-(SP)
51 001450 042716 177760 BIC #^C17,(SP) ;CLEAR ALL BUT N-Z-V-C FIELDS
52 001454 006303 ASL R3          ;SHIFT OVER ADDRESS TO AVOID T FLAG
53 001456 042703 177437 BIC #^C340,R3   ;CLEAR ALL BUT PRIORITY FIELD
54 001462 052603 BIS (SP)+,R3     ;COMBINE N-Z-V-C WITH PRIO FIELD
55 001464 010310 MOV R3,(R0)     ;STORE INTO VECTOR PS WORD
56          ;
57          ; Free the vector control block

```

```
58 ;  
59 001466 105062 0000000 2$: CLRB VC$JOB(R2)  
60 001472 105062 0000000 CLRB VC$VEC(R2)  
61 ;  
62 ; Finished  
63 ;  
64 001476 012603 MOV (SP)+,R3  
65 001500 000207 RETURN
```

RTDEV -- .DEVICE Emt

```

1          .SBTTL  RTDEV  -- .DEVICE Emt
2          ;-----
3          ; The .DEVICE emt is used to declare a list of addresses and values to
4          ; be stored when a job exits.
5          ;
6 001502  004767  000244  RTDEV:  CALL    PRVRT          ;Do we have real-time privilege?
7          ;
8          ; Determine if this is a simple or linked type list
9          ;
10 001506  105767  000000G  1$:      TSTB    EMTBLK          ;SIMPLE OR LINKED LIST?
11 001512  001004          BNE      2$          ;BR IF LINKED LIST
12          ;
13          ; Simple (non-linked) list
14          ;
15 001514  016767  000002G  000000G  MOV      EMTBLK+2,DEVLS ;SET ADDRESS OF .DEVICE LIST
16 001522  000407          BR       9$          ;FINISHED
17          ;
18          ; Linked list
19          ;
20 001524  016746  000000G  2$:      MOV      DEVL, -(SP)      ;PUSH ADDRESS OF CURRENT LIST HEAD
21 001530  106677  000002G  MTPD    @EMTBLK+2      ;STORE AS FORWARD LINK FROM NEW LIST
22 001534  016767  000002G  000000G  MOV      EMTBLK+2,DEVL  ;SAVE ADDRESS OF NEW LIST
23          ;
24          ; Finished
25          ;
26 001542  000167  000000G  9$:      JMP      EMTXIT

```

```

1          .SBTTL  RTSTOP -- Real-time cleanup at end of job execution
2          ;-----
3          ; RTSTOP is called when a job exits for any reason.
4          ; It does any necessary real-time cleanup for the job.
5          ; This consists of the following things:
6          ;   1. Process any specified .DEVICE list.
7          ;   2. Disconnect any interrupt vectors attached to the job.
8          ;   3. Release exclusive access of system if job has gotten it.
9          ;
10         ; Inputs:
11         ;   R1 = Current job number
12         ;
13 001546  010246  RTSTOP: MOV      R2, -(SP)
14         ;
15         ; If job did a .DEVICE emt, process the device reset list.
16         ;
17         ; Process linked type lists first
18 001550  016702  000000G  1$:  MOV      DEVL, R2      ; IS THERE A LINKED TYPE LIST TO PROCESS?
19 001554  001415          BEQ      3$              ; BR IF NOT
20 001556  005067  000000G  CLR      DEVL          ; CLEAR IN CASE WE TRAP
21 001562  106522          MFPD     (R2)+        ; GET ADDRESS OF FORWARD LINK FROM LIST HEAD
22 001564  012667  000000G  MOV      (SP)+, DEVL   ; SAVE THIS ADDRESS AS NEW HEAD OF LINKED LIST
23 001570  106522          2$:  MFPD     (R2)+        ; GET ADDRESS TO STORE INTO
24 001572  012600          MOV      (SP)+, R0
25 001574  001765          BEQ      1$              ; BR IF REACHED END OF LIST
26 001576  106522          MFPD     (R2)+        ; GET VALUE TO STORE INTO CELL
27 001600  012610          MOV      (SP)+, (R0)   ; DO THE STORE
28 001602  004767  000112  CALL     CKVREL        ; See if we just released an interrupt conctn.
29 001606  000770          BR       2$              ; GO PROCESS NEXT ITEM IN LIST
30         ; Now process non-linked type lists
31 001610  016702  000000G  3$:  MOV      DEVLS, R2   ; ANY NON-LINKED .DEVICE LIST?
32 001614  001412          BEQ      INTSTP       ; BR IF NOT
33 001616  005067  000000G  CLR      DEVLS        ; CLEAR IN CASE WE TRAP
34 001622  106522          4$:  MFPD     (R2)+        ; GET ADDRESS TO STORE INTO
35 001624  012600          MOV      (SP)+, R0
36 001626  001405          BEQ      INTSTP       ; BR IF REACHED END OF LIST
37 001630  106522          MFPD     (R2)+        ; GET VALUE TO STORE
38 001632  012610          MOV      (SP)+, (R0)   ; DO THE STORE
39 001634  004767  000060  CALL     CKVREL        ; See if we released an interrupt connection
40 001640  000770          BR       4$              ; GO PROCESS NEXT ITEM IN LIST
41         ;
42         ; Disconnect all interrupt vectors associated with this job
43         ;
44 001642  012702  000000G  INTSTP: MOV     #VCBBAS, R2 ; POINT TO 1ST VECTOR CONTROL BLOCK
45 001646  020227  000000G  1$:  CMP      R2, #VCBEND  ; CHECKED ALL VECTOR CONTROL BLOCKS?
46 001652  103010          BHS     4$              ; BR IF YES
47 001654  120162  000000G  CMPB    R1, VC$JOB(R2) ; IS THIS VCB IN USE BY THIS JOB?
48 001660  001002          BNE     2$              ; BR IF NOT
49 001662  004767  177504  CALL     RELVEC       ; RELEASE THE VECTOR AND THE VCB
50 001666  062702  000000G  2$:  ADD     #VC$$SZ, R2  ; POINT TO NEXT VCB
51 001672  000765          BR       1$
52         ;
53         ; If job has exclusive access to the system, release it
54         ;
55 001674  120167  000000G  4$:  CMPB    R1, EXCJOB   ; DOES THIS JOB HAVE EXCLUSIVE ACCESS TO SYS?
56 001700  001005          BNE     3$              ; BR IF NOT
57 001702  105767  000000G  TSTB   CINFLG        ; IS THIS JOB DOING A CHAIN NOW?

```

```
58 001706 001002          BNE      3$          ; IF YES THEN RETAIN EXCLUSIVE ACCESS
59 001710 105067 000000G   CLRB    EXCJOB       ; RELEASE EXCLUSIVE ACCESS
60                          ;
61                          ; Finished
62                          ;
63 001714 012602          3$:    MOV     (SP)+, R2
64 001716 000207          RETURN
```

RTSTOP -- Real-time cleanup at end of job execution

```

1          ; -----
2          ; See if the cell being stored into by the .DEVICE list is an interrupt
3          ; vector.  If so, mark the vector control block so that we don't alter
4          ; the interrupt location later.
5          ;
6          ; Inputs:
7          ; RO = Address of cell being modified by .DEVICE list processing.
8          ;
9 001720 010246 CKVREL: MOV      R2,-(SP)
10         ;
11         ; See if location being modified is an interrupt vector
12         ;
13 001722 020027 000500          CMP      RO,#500          ;Could this be an interrupt vector?
14 001726 103007          BHS      9$              ;Br if not
15 001730 006200          ASR      RO              ;Get vector address / 2
16 001732 004767 000054          CALL     SRCVEC          ;Try to find vector control block
17 001736 103403          BCS      9$              ;Br if no vector attachment
18         ;
19         ; Set flag in vector control block which prevents us from connecting
20         ; this vector to unexpected-interrupt location later.
21         ;
22 001740 152762 000000G 000000G  BISB     #VF$DET,VC$FLG(R2) ;Say vector has been disconnected
23         ;
24         ; Finished
25         ;
26 001746 012602 9$:      MOV      (SP)+,R2
27 001750 000207          RETURN

```

** Subroutines **

```

1          .SBTTL  ** Subroutines **
2          .SBTTL  PRVRT  -- Determine if job has real-time privilege
3          ;-----
4          ; Determine if the job has real-time privilege.
5          ; If not, error code 0 is returned for the EMT.
6          ;
7 001752  032767  000000G 000000G PRVRT:  BIT    #PO$RT,PRIVCO  ;Does job have real-time privilege?
8 001760  001003                BNE    9$          ;Br if yes
9 001762  005000                CLR    RO          ;Return error code 0
10 001764  000167  000000G      JMP    SETERR
11 001770  000207                9$:   RETURN
12
13          .SBTTL  PRVMEM -- Check for access authorization to phys memory
14          ;-----
15          ; Determine if the job can access physical memory.
16          ; If not, error code 0 is returned for the EMT.
17          ;
18 001772  032767  000000G 000000G PRVMEM: BIT    #PO$MEM,PRIVCO  ;Can we access physical memory?
19 002000  001003                BNE    9$          ;Br if yes
20 002002  005000                CLR    RO          ;Return error code 0
21 002004  000167  000000G      JMP    SETERR
22 002010  000207                9$:   RETURN

```

SRCVEC -- Search for vector control block

```

1          .SBTTL  SRCVEC -- Search for vector control block
2          ;-----
3          ; SRCVEC is called to search for a vector control block associated
4          ; with a particular interrupt vector.
5          ;
6          ; Inputs:
7          ; R0 = Address of vector / 2
8          ;
9          ; Outputs:
10         ; C-flag set on return if no vector control block found for vector.
11         ; R2 = Address of vector control block if one found.
12         ;
13 002012 012702 000000G SRCVEC: MOV    #VCBBAS,R2    ;GET ADDRESS OF BASE OF CONTROL BLOCK AREA
14 002016 020227 000000G 1$:  CMP    R2,#VCBEND    ;CHECKED ALL BLOCKS?
15 002022 103006          BHS    4$          ;BR IF YES
16 002024 120062 000000G    CMPB   R0,VC$VEC(R2) ;IS THIS CONTROL BLOCK FOR THE VECTOR?
17 002030 001405          BEQ    2$          ;BR IF YES
18 002032 062702 000000G    ADD    #VC$$SZ,R2    ;POINT TO NEXT VECTOR CONTROL BLOCK
19 002036 000767          BR     1$
20 002040 000261          4$:  SEC          ;SIGNAL FAILURE ON RETURN
21 002042 000401          BR     3$
22 002044 000241          2$:  CLC          ;SIGNAL SUCCESS ON RETURN
23 002046 000207          3$:  RETURN
24          000001          .END

```

Errors detected: 0

*** Assembler statistics

Work file reads: 0
 Work file writes: 0
 Size of work file: 153 Words (1 Pages)
 Size of core pool: 18176 Words (71 Pages)
 Operating system: RT-11

Elapsed time: 00:00:20.11
 ,LP: TSRTX=DK: TSRTX/C/N: SYM

RPAR	1-44	3-30	4-20*	4-48*			
RPDR	1-44	3-28	4-21*	4-63*			
RTALJB	2-44	10-21#					
RTBIC	2-35	6-47#					
RTBIS	2-34	6-35#					
RTCMPL	2-48	9-17#					
RTCVAD	2-31	3-18	3-27#				
RTCVC1	3-19	3-27	3-55#				
RTCVC2	3-23	3-31#					
RTCVC3	3-21	3-29	3-36#				
RTCVC4	3-32	3-40#					
RTCVDA	2-49	3-17#					
RTCVEC	2-40	12-23#					
RTDEV	1-30	14-6#					
RTDVEC	2-47	12-13#					
RTEMT	1-30	2-17#					
RTEMTV	2-26	2-31#	2-51				
RTEXJB	2-43	10-7#					
RTFREZ	2-42	5-5#					
RTIOMP	2-36	7-6#					
RTLCK	1-37	2-38					
RTMAP	7-8#	7-17					
RTPEEK	2-32	6-11#					
RTPOKE	2-33	6-23#					
RTRMMP	2-37	7-16#					
RTRSUM	1-29	8-32#					
RTRVEC	2-41	13-8#					
RTSPL	2-45	11-9#					
RTSPND	1-29	8-6#					
RTSTOP	1-30	15-13#					
RTUNLK	2-39	5-15#					
RTVMAP	2-46	4-13#					
RTXIT	6-13	6-25	6-37	6-49#	7-9		
RTXVEC	12-19	12-27#					
S*RT	1-48	9-37					
S*SPND	1-41	8-17					
S*TWFN	1-38	9-32					
SETERR	1-34	4-34	12-17	12-35	12-43	17-10	17-21
SEMAP	1-35	4-25	7-8				
SRCVEC	12-30	12-40	13-11	16-16	18-13#		
TSRTX	1-13#						
UEXINT	1-36	13-45					
UEXRTN	1-39	13-42					
UPARO	1-45	4-49*					
UPDRO	1-45	4-64*					
UPMODE	1-46	1-46					
URO	1-37	6-12*					
VC*SZ	1-40	15-50	18-18				
VC*FLG	1-44	12-50*	13-31	16-22*			
VC*JOB	1-35	12-32	12-48*	13-13	13-59*	15-47	
VC*PRI	1-37	12-54*					
VC*RTN	1-35	12-51*					
VC*VEC	1-35	12-49*	13-37	13-60*	18-16		
VCBBAS	1-40	15-44	18-13				
VCBEND	1-40	15-45	18-14				
VF*DET	1-44	13-31	16-22				

VF\$DIR	1-44	12-18
VPRIHI	1-48	9-38
VUXIFL	1-39	13-40

DISABL	1-54#	8-14
ENABL	1-58#	8-24
OCALL	1-19#	