

TSRT -- Resident portion of ree MACRD V05.04 Friday 18-Dec-87 07:58
Table of contents

2- 1 RTINT -- Real-time interrupt entry point
4- 1 DIEMT -- EMT executed from interrupt service routine

```

1          .TITLE  TSRT -- Resident portion of real-time code
2          .ENABL  LC
3          .DSABL  GBL
4          ;-----
5          ; This module contains the resident portion of the TSX-Plus real-time
6          ; processing code.
7          ;
8          ; Copyright (c) 1980, 1981, 1982, 1983.
9          ; S&H Computer Systems, Inc.
10         ; Nashville, Tennessee
11         ; All rights reserved.
12         ;
13 000000   .CSECT  TSRT
14 000000   TSRT:
15         ;
16         ; Global definitions
17         ;
18         .GLOBL  RTINT
19         ;
20         ; Global references
21         ;
22         .GLOBL  INTEN, FORK, VF$DIR, VC$FLG, GETRTQ, VC$JOB
23         .GLOBL  CQ$JOB, VC$VEC, CQ$RO, VC$RTN, CQ$RTN
24         .GLOBL  CQ$PA5, VC$PRI, CQ$PRI, QCOMPL, CQ$RNS
25         .GLOBL  CURVC, LCXPAR, KPAR6, UPARO, CUPARO, UPDRO, CUPDRO
26         .GLOBL  FPUUSE, UPMODE, PSW, UMSPSV, CPLEMT, UMODE, UPMODE
27         .GLOBL  LSPND, FORCEX, CFLAG, EMTENT, CQ$R1, MAXPRI, VPRIHI, S$RT
28         .GLOBL  LBPRI, S$TWFN, FP$RT, KPAR5, MAPPAR
29         .GLOBL  CQ$CP, CP$RT, CP$STD
30         ;
31         ; Data areas
32         ;
33 000000   000000   RISPSV: .WORD  0          ;Holds SP during direct interrupt service call
34 000002   000000   VCHOLD: .WORD  0          ;Used to hold address of vec ctrl blk during .inten

```

RTINT -- Real-time interrupt entry point

```

1          .SBTTL  RTINT  -- Real-time interrupt entry point
2
3          ;-----
4          ; RTINT is the entry point for all real-time interrupts.
5          ; The actual interrupt vector sent the interrupt to the vector control
6          ; block for the interrupt which had the following instruction stored
7          ; as its first word: [ JSR  R2,RTINT ]. Thus on entry to RTINT, R2
8          ; points to byte 4 of the vector control block.
9          ;
10         ; Save address of the vector control block.
11         RTINT:  SUB      #4,R2          ;GET ADDRESS OF START OF CONTROL BLOCK
12         MOV     R2,VCHOLD             ;SAVE ADDRESS ACROSS .INTEN
13         MOV     (SP)+,R2             ;RESTORE R2
14
15         ; Now do a .INTEN and .FORK to get out of interrupt mode.
16
17         JSR     R5,INTEN             ;STANDARD INTERRUPT ENTRY
18         .WORD  0                     ;(PRIORITY 7)
19         MOV     VCHOLD,R4            ;PICK UP ADDRESS OF INTERRUPT CONTROL BLOCK
20         JSR     R5,FORK              ;DROP DOWN TO FORK LEVEL
21         .WORD  FP$RT                 ;Fork processing priority
22
23         ; At this point we are running at fork level with the address of the
24         ; vector control block in R4.
25
26         ; Determine if interrupt is directly connected with interrupt
27         ; service routine or if interrupt service routine is to be
28         ; called as a completion routine.
29
30         BITB   #VF$DIR,VC$FLG(R4); Is this a directly connected interrupt?
31         BNE    DIENTR                 ;Br if yes
32
33         ; This interrupt is connected to job through a completion routine.
34         ; Set up a completion routine queue entry for the job connected with
35         ; the interrupt.
36
37         CALL   GETRTQ                 ;GET A COMPLETION QUEUE ENTRY
38         MOVB   VC$JOB(R4),CQ$JOB(R1) ;SET JOB NUMBER
39         CLR    R0
40         BISB   VC$VEC(R4),R0         ;GET VECTOR ADDRESS WITHOUT SIGN EXTENSION
41         ASL   R0                         ;CONVERT TO ABS ADDRESS
42         MOV   R0,CQ$R0(R1)            ;PUT IN R0 WHEN WE ENTER COMPLETION ROUTINE
43         MOV   VC$RTN(R4),CQ$RTN(R1) ;SET ADDRESS OF COMPLETION ROUTINE
44         MOV   @#KPAR5,CQ$PA5(R1); Save current kernel PAR 5 mapping
45         MOVB  VC$PRI(R4),R5           ;GET COMPLETION ROUTINE PRIORITY
46         BNE   3$                      ;Br if priority not zero
47         MOVB  #S$TWFN,CQ$RNS(R1); Set non-real-time execution state
48         MOVB  #CP$STD,CQ$CP(R1); Set standard compl rtn class priority
49         MOVB  CQ$JOB(R1),R0          ;Get job index number
50         MOVB  LBSPRI(R0),CQ$PRI(R1); Set execution priority
51         BR    4$
52         MOVB  #S$RT,CQ$RNS(R1); Set real-time execution state
53         MOVB  #CP$RT,CQ$CP(R1); Set real-time compl rtn class priority
54         ADD   VPRIH1,R5              ;Add base real-time priority
55         CMP   R5,#MAXPRI             ;Did priority overflow?
56         BLOS  2$                      ;Br if not
57         MOV   #MAXPRI,R5             ;Reduce to maximum allowed

```

RTINT -- Real-time interrupt entry point

```
58 000200 110561 0000000 2*:      MOVB      R5,CQ#PRI(R1)  ;Set execution priority
59                                     ;
60                                     ; Queue a completion routine for the job.
61                                     ;
62 000204 010104          4*:      MOV       R1,R4          ;GET ADDRESS OF COMPLETION QUEUE ENTRY
63 000206 004767 0000000          CALL      QCOMPL        ;QUEUE A COMPLETION REQUEST FOR THE JOB
64                                     ;
65                                     ; Finished
66                                     ;
67 000212 000207          1*:      RETURN
```

RTINT --- Real-time interrupt entry point

```

1          ; -----
2          ; Enter an interrupt service routine that is directly connected to
3          ; a real-time interrupt.
4          ;
5          ; Save pointer to interrupt control block
6          ; (CURVC non-zero is also a flag that we are executing a directly
7          ; connected interrupt service routine.)
8          ;
9 000214 010467 000000G DIENTR: MOV      R4,CURVC      ;Save address of interrupt control block
10         ;
11         ; Get index number of job that is to receive the interrupt and map
12         ; KPAR6 to job's context block (note, the FORK saved original KPAR6).
13         ;
14 000220 116401 000000G          MOV     VC#JOB(R4),R1  ;Get job index number
15 000224 016137 000000G 000000G  MOV     LCXPAR(R1),@#KPAR6 ;Map kernel par 6 to job context block
16         ;
17         ; Save current user-mode PAR registers and load for new job
18         ;
19 000232 012701 000000G          MOV     #UPARO,R1      ;Point to user PAR registers
20 000236 012702 000000G          MOV     #CUPARO,R2      ;Point to PAR holding cells in context block
21 000242 012703 000000G          MOV     #UPDRO,R3      ;Point to user PDR registers
22 000246 012704 000000G          MOV     #CUPDRO,R4     ;Point to PDR holding cells in context block
23 000252 012700 000010          MOV     #8,R0          ;There are 8 PAR and PDR register pairs
24 000256 011146          1$:  MOV     (R1),-(SP)      ;Save current user PAR value
25 000260 012221          MOV     (R2)+,(R1)+    ;Set PAR to map to job being called
26 000262 011346          MOV     (R3),-(SP)      ;Save current user PDR value
27 000264 012423          MOV     (R4)+,(R3)+    ;Set PDR to map to job being called
28 000266 077005          SOB     R0,1$          ;Loop if more PAR's to load
29 000270 016704 000000G          MOV     CURVC,R4      ;Recover pointer to vector control block
30         ;
31         ; If Floating Point Unit is in use, save its status
32         ;
33 000274 105767 000000G          TSTB   FPUUSE          ;Is FPU in use?
34 000300 001414          BEQ    2$              ;Br if not
35 000302 170202          STFPS  R2              ;Save FPU status
36 000304 010246          MOV    R2,-(SP)
37 000306 170011          SETD   ;Set to 64-bit mode
38 000310 174046          STD   R0,-(SP)        ;AC0
39 000312 174146          STD   R1,-(SP)        ;AC1
40 000314 174246          STD   R2,-(SP)        ;AC2
41 000316 174346          STD   R3,-(SP)        ;AC3
42 000320 172404          LDD   R4,R0           ;AC4-->AC0
43 000322 174046          STD   R0,-(SP)        ;(AC4)
44 000324 172405          LDD   R5,R0           ;AC5-->AC0
45 000326 174046          STD   R0,-(SP)        ;(AC5)
46 000330 170102          LDFPS R2              ;Leave FPU status same as on entry
47         ;
48         ; Set up user-mode SP
49         ;
50 000332 052737 000000G 000000G 2$:  BIS    #UPMODE,@#PSW   ;Make sure previous-mode = user
51 000340 106506          MFPD  SP              ;Save current user-mode stack pointer on stack
52 000342 016702 000000G          MOV    UMSPSV,R2      ;Should we use SP from context block?
53 000346 001001          BNE   3$              ;Br if yes
54 000350 011602          MOV    (SP),R2        ;No, use current user mode SP
55         ;
56         ; Push address of completion exit EMT on user's stack
57         ;

```

RTINT -- Real-time interrupt entry point

```

58 000352 012746 0000000 3#:   MOV    #CPLEMT,-(SP) ;Address of exit emt in user's job
59 000356 106642          MTPD   -(R2)        ;Push onto user's stack
60 000360 010246          MOV    R2,-(SP)    ;Set user-mode SP for job being entered
61 000362 106606          MTPD   SP
62                                ;
63                                ; Enter user's interrupt service routine
64                                ;
65 000364 010667 177410          MOV    SP,RISPSV   ;Save SP before entry to user's routine
66 000370 012737 000410' 000030    MOV    #DIEMT,@#30 ;Send EMT traps to DIEMT routine
67 000376 012746 0000000          MOV    #UMODE!UPMODE,-(SP) ;Set PS for interrupt routine
68 000402 016446 0000000          MOV    VC#RTN(R4),-(SP);Set PC for interrupt routine
69 000406 000002          RTI                    ;Enter interrupt service routine

```

```

1          .SBTTL  DIEMT  -- EMT executed from interrupt service routine
2          ;-----
3          ; An EMT was executed from within a directly connected real-time interrupt
4          ; service routine.  This can only be one of three things:
5          ;   1. A request to return from the interrupt service routine.
6          ;   2. A request to schedule a completion routine for the job.
7          ;   3. A .RSUM EMT to restart the main-line program routine.
8          ;
9 000410  DIEMT:
10         ;
11         ; Determine if this is a request to exit from the interrupt service routine
12         ;
13 000410 021627 0000026      CMP      (SP),#CPLEMT+2  ;Is this the EMT to exit from int routine?
14 000414 001522              BEQ      DIEXIT      ;Br if yes
15         ;
16         ; This is not a request to exit from the interrupt service routine.
17         ;
18 000416 010046              MOV      R0,-(SP)
19 000420 010146              MOV      R1,-(SP)
20 000422 010246              MOV      R2,-(SP)
21 000424 010446              MOV      R4,-(SP)
22         ;
23         ; See if it is a .RSUM EMT.
24         ;
25 000426 016704 0000006      MOV      CURVC,R4      ;Get address of vector control block
26 000432 016601 0000010      MOV      B,(SP),R1     ;Get address past EMT instruction
27 000436 106541              MFPD     -(R1)        ;Get the EMT instruction
28 000440 021627 104374      CMP      (SP),#104374  ;Could this be a .RSUM EMT?
29 000444 001012              BNE     DICMPL      ;Br if not
30 000446 020027 001000      CMP      R0,#1000     ;Check contents of R0
31 000452 001103              BNE     DIEXIT      ;Br if not .RSUM
32         ;
33         ; Perform a .RSUM from within a directly connected interrupt routine
34         ;
35 000454 116401 0000006  DIRSUM:  MOVB     VC$JOB(R4),R1  ;Get current job number
36 000460 005261 0000006      INC      LSPND(R1)    ;Increment .SPND count for job
37 000464 004767 0000006      CALL     FORCEX       ;Force execution of the job
38 000470 000463              BR       DIEMTX      ;Go exit from EMT processing

```

```

1      ;
2      ; Determine if this is an EMT to request that a completion routine
3      ; be scheduled for the job.
4      ;
5      ; The form of the EMT is
6      ;
7      ;     EMT     375
8      ;
9      ; With R0 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 000472 021627 104375  DICMPL: CMP     (SP),#104375 ;Is this an EMT 375 instruction?
18 000476 001071          BNE     DIEXIT ;Br if not
19 000500 106520          MFPD   (RO)+ ;Get 1st word of EMT argument block
20 000502 022627 060021  CMP     (SP)+,#60021 ;Function code = 140 and sub-fun = 21?
21 000506 001065          BNE     DIEXIT ;Br if not
22     ;
23     ; This is a request to schedule a completion routine.
24     ; Get a completion request queue element.
25     ;
26 000510 004767 000000G  CALL    GETRTQ ;Get a completion Q element (address in R1)
27     ;
28     ; Set up the queue element
29     ;
30 000514 116461 000000G 000000G  MOVVB  VC$JOB(R4),CQ$JOB(R1) ;Set job number for completion routine
31 000522 106520          MFPD   (RO)+ ;Get address of completion routine
32 000524 012661 000000G  MOV    (SP)+,CQ$RTN(R1);Set completion routine address in Q element
33 000530 106520          MFPD   (RO)+ ;Get priority value
34 000532 012602          MOV    (SP)+,R2
35 000534 001011          BNE     2$ ;Br if priority value not zero
36 000536 112761 000000G 000000G  MOVVB  #$TWFN,CQ$RNS(R1);Set non-real-time execution state
37 000544 116102 000000G  MOVVB  CQ$JOB(R1),R2 ;Get job number
38 000550 116261 000000G 000000G  MOVVB  LBSPRI(R2),CQ$PRI(R1);Set execution priority
39 000556 000414          BR     3$
40 000560 112761 000000G 000000G 2$: MOVVB  #$RT,CQ$RNS(R1);Set real-time execution state
41 000566 066702 000000G  ADD    VPRIHI,R2 ;Add base priority for real-time jobs
42 000572 020227 000000G  CMP    R2,#MAXPRI ;Did priority value overflow?
43 000576 101402          BLOS   1$ ;Br if not
44 000600 012702 000000G  MOV    #MAXPRI,R2 ;Set maximum allowed priority
45 000604 110261 000000G 1$: MOVVB  R2,CQ$PRI(R1) ;Set execution priority
46 000610 106520         3$: MFPD   (RO)+ ;Get value to be passed in R1
47 000612 012661 000000G  MOV    (SP)+,CQ$R1(R1) ;Set R1 value for completion routine
48 000616 005002          CLR    R2
49 000620 156402 000000G  BISB  VC$VEC(R4),R2 ;Get vector address without sign extension
50 000624 006302          ASL   R2 ;Convert to abs address
51 000626 010261 000000G  MOV    R2,CQ$RO(R1) ;Pass to completion routine in R0
52     ;
53     ; Queue the completion routine for the job
54     ;
55 000632 010104          MOV    R1,R4 ;Get completion Q element pointer to R4
56 000634 004767 000000G  CALL   QCOMPL ;Queue the completion routine request
57     ;

```



```
58          ; Return to interrupt service routine following EMT
59          ;
60 000640 005726          DIEMTX: TST      (SP)+          ;Remove EMT instruction from stack
61 000642 012604          MOV      (SP)+,R4
62 000644 012602          MOV      (SP)+,R2
63 000646 012601          MOV      (SP)+,R1
64 000650 012600          MOV      (SP)+,R0
65 000652 042766 0000000 000002          BIC      #CFLAG,2(SP) ;C-flag will be reset on return from EMT
66 000660 000002          RTI          ;Return from .RSUM EMT
```

DIEMT -- EMT executed from interrupt service routine

```

1 ;-----
2 ; Exit from directly connected interrupt service routine
3 ;
4 000662 016706 177112 DIEXIT: MOV RISPSV, SP ; Restore SP to state before entering int rtn
5 000666 106606 MTPD SP ; Restore user-mode SP
6 ;
7 ; Restore Floating Point Unit status if it is in use
8 ;
9 000670 105767 000000G TSTB FPUUSE ; Is FPU in use?
10 000674 001412 BEQ 1$ ; Br if not
11 000676 170011 SETD ; Set 64-bit mode
12 000700 172426 LDD (SP)+, R0 ; (AC5)
13 000702 174005 STD R0, R5 ; AC0-->AC5
14 000704 172426 LDD (SP)+, R0 ; (AC4)
15 000706 174004 STD R0, R4 ; AC0-->AC4
16 000710 172726 LDD (SP)+, R3 ; AC3
17 000712 172626 LDD (SP)+, R2 ; AC2
18 000714 172526 LDD (SP)+, R1 ; AC1
19 000716 172426 LDD (SP)+, R0 ; AC0
20 000720 170126 LDFPS (SP)+ ; FPU status register
21 ;
22 ; Restore user-mode memory mapping
23 ;
24 000722 012701 000020G 1$: MOV #UPARO+16., R1 ; Point past last PAR register
25 000726 012702 000020G MOV #UPDRO+16., R2 ; Point past last PDR register
26 000732 012700 000010 MOV #8., R0 ; Load 8 PAR/PDR register pairs
27 000736 012642 2$: MOV (SP)+, -(R2) ; Restore PDR register
28 000740 012641 MOV (SP)+, -(R1) ; Restore PAR register
29 000742 077003 SOB R0, 2$ ; Loop if more to restore
30 ;
31 ; Say we are exiting from interrupt service routine
32 ;
33 000744 005067 000000G CLR CURVC ; Say not in interrupt service routine
34 000750 012737 000000G 000030 MOV #EMTENT, @#30 ; Restore EMT trap vector
35 ;
36 ; Return to FORK which will return from interrupt
37 ;
38 000756 000207 RETURN ; Return to FORK
39 000001 .END

```

Errors detected: 0

*** Assembler statistics

Work file reads: 0
 Work file writes: 0
 Size of work file: 64 Words (1 Pages)
 Size of core pool: 17920 Words (70 Pages)
 Operating system: RT-11

Elapsed time: 00:00:05.23
 DK: TSRT, LP: TSRT=DK: TSRT. MAC/C/N: SYM

CFLAG	1-27	5-65			
CP#RT	1-29	2-53			
CP#STD	1-29	2-48			
CPLEMT	1-26	3-58	4-13		
CQ#CP	1-29	2-48*	2-53*		
CQ#JOB	1-23	2-38*	2-49	5-30*	5-37
CQ#PA5	1-24	2-44*			
CQ#PRI	1-24	2-50*	2-58*	5-38*	5-45*
CQ#R0	1-23	2-42*	5-51*		
CQ#R1	1-27	5-47*			
CQ#RNS	1-24	2-47*	2-52*	5-36*	5-40*
CQ#RTN	1-23	2-43*	5-32*		
CUPARO	1-25	3-20			
CUPDRO	1-25	3-22			
CURVC	1-25	3-9*	3-29	4-25	6-33*
DICMPL	4-29	5-17#			
DIEMT	3-66	4-9#			
DIEMTX	4-38	5-60#			
DIENTR	2-31	3-9#			
DIEXIT	4-14	4-31	5-18	5-21	6-4#
DIRSUM	4-35#				
EMTENT	1-27	6-34			
FORCEX	1-27	4-37			
FORK	1-22	2-20			
FP#RT	1-28	2-21			
FPUUSE	1-26	3-33	6-9		
GETRTQ	1-22	2-37	5-26		
INTEN	1-22	2-17			
KPAR5	1-28	2-44			
KPAR6	1-25	3-15*			
LBSPRI	1-28	2-50	5-38		
LCXPAR	1-25	3-15			
LSPND	1-27	4-36*			
MAPPAR	1-28				
MAXPRI	1-27	2-55	2-57	5-42	5-44
PSW	1-26	3-50*			
QCOMPL	1-24	2-63	5-56		
RISPSV	1-33#	3-65*	6-4		
RTINT	1-18	2-11#			
S#RT	1-27	2-52	5-40		
S#TWFN	1-28	2-47	5-36		
TSRT	1-14#				
UMODE	1-26	3-67			
UMSPSV	1-26	3-52			
UPARO	1-25	3-19	6-24		
UPDRO	1-25	3-21	6-25		
UPMODE	1-26	1-26	3-50	3-67	
VC#FLG	1-22	2-30			
VC#JOB	1-22	2-38	3-14	4-35	5-30
VC#PRI	1-24	2-45			
VC#RTN	1-23	2-43	3-68		
VC#VEC	1-23	2-40	5-49		
VCHOLD	1-34#	2-12*	2-19		
VF#DIR	1-22	2-30			
VPRIHI	1-27	2-54	5-41		