5

Interrupt Manager

5.1 Introduction

Any real-time executive must provide a mechanism for quick response to externally generated interrupts to satisfy the critical time constraints of the application. The **interrupt manager** provides this mechanism for **RTEMS**. This manager permits quick interrupt response times by providing the critical ability to alter task execution which allows a task to be preempted upon exit from an ISR. The interrupt manager includes the following directive:

Name	Directive Description
interrupt_catch	Establish an ISR

5.2 Background

5.2.1 Processing an Interrupt

The interrupt manager allows the application to connect a subprogram to a hardware interrupt vector. When an interrupt occurs, the processor will automatically vector to **RTEMS**. **RTEMS** saves and restores all registers which are not preserved by the normal Ada calling convention for the target processor and invokes the user's ISR. The user's ISR is responsible for processing the interrupt, clearing the interrupt if necessary, and device specific manipulation.

The **interrupt_catch** directive connects a procedure to an interrupt vector. The interrupt service routine is assumed to abide by these conventions and have a specification similar to the following:

```
procedure user_isr(
  vector : in     rtems.vector_number
);
```

The **vector** number argument is provided by **RTEMS** to allow the application to identify the interrupt source. This could be used to allow a single routine to service interrupts from multiple instances of the same device. For example, a single routine could service interrupts from multiple serial ports and use the vector number to identify which port requires servicing.

To minimize the masking of lower or equal priority level interrupts, the ISR should perform the minimum actions required to service the interrupt. Other non-essential actions should be handled by application tasks. Once the user's ISR has completed, it returns control to the **RTEMS** interrupt manager which will perform task dispatching and restore the registers saved before the ISR was invoked.

The **RTEMS** interrupt manager guarantees that proper task scheduling and dispatching are performed at the conclusion of an ISR. A system call made by the ISR may have readied a task of higher priority than the interrupted task. Therefore, when the ISR completes, the postponed dispatch processing must be performed. No dispatch processing is performed as part of directives which have been invoked by an ISR.

Applications must adhere to the following rule if proper task scheduling and dispatching is to be performed:

---

*The interrupt manager must be used for all ISRs which may be interrupted by the highest priority ISR which invokes an **RTEMS** directive.*

Consider a processor which allows a numerically low interrupt level to interrupt a numerically greater interrupt level. In this example, if an **RTEMS** directive is used in a level 4 ISR, then all ISRs which execute at levels 0 through 4 must use the interrupt manager.

Interrupts are nested whenever an interrupt occurs during the execution of another ISR. **RTEMS** supports efficient interrupt nesting by allowing the nested ISRs to terminate without performing any dispatch processing. Only when the outermost ISR terminates will the postponed dispatching occur.

5.2.2 RTEMS Interrupt Levels

Many processors support multiple interrupt levels or priorities. The exact number of interrupt levels is processor dependent. **RTEMS** internally supports 256 interrupt levels which are mapped to the processor's interrupt levels. For specific information on the mapping between **RTEMS** and the target processor's interrupt levels, refer to the **Interrupt Processing** chapter of the Ada Applications Supplement document for a specific target processor.

5.2.3 Disabling of Interrupts by RTEMS

During the execution of directive calls, critical sections of code may be executed. When these sections are encountered, **RTEMS** disables all maskable interrupts before the execution of the section and restores them to the previous level upon completion of the section. **RTEMS** has been optimized to insure that interrupts are disabled for a minimum length of time. The maximum length of time interrupts are disabled by **RTEMS** is processor dependent and is detailed in the **Timing Specification** chapter of the **Ada Applications Supplement** document for a specific target processor.

Non-maskable interrupts (NMI) cannot be disabled, and ISRs which execute at this level **MUST NEVER** issue **RTEMS** system calls. If a directive is invoked, unpredictable results may occur due to the inability of **RTEMS** to protect its critical sections. However, ISRs that make no system calls may safely execute as non-maskable interrupts.

5.3 Operations

5.3.1 Establishing an ISR

The **interrupt_catch** directive establishes an ISR for the system. The address of the ISR and its associated CPU vector number are specified to this directive. This directive installs the **RTEMS** interrupt wrapper in the processor's Interrupt Vector Table and the address of the user's ISR in the **RTEMS**' Vector Table. This directive returns the previous contents of the specified vector in the **RTEMS**' Vector Table.

5.3.2 Directives Allowed from an ISR

Using the interrupt manager insures that **RTEMS** knows when a directive is being called from an ISR. The ISR may then use system calls to synchronize itself with an application task. The synchronization may involve messages, events or signals

being passed by the ISR to the desired task. Directives invoked by an ISR must operate only on objects which reside on the local node. The following is a list of **RTEMS** system calls that may be made from an ISR:

- *Task Management*

  task_get_note, task_set_note, task_suspend, task_resume

- *Clock Management*

  clock_get, clock_tick

- *Message, Event, and Signal Management*

  message_queue_send, message_queue_urgent

  event_send

  signal_send

- *Semaphore Management*

  semaphore_release

- *Dual-Ported Memory Management*

  port_external_to_internal, port_internal_to_external

- *IO Management*

  io_initialize, io_open, io_close, io_read, io_write, io_control

- *Fatal Error Management*

  fatal_error_occurred

- *Multiprocessing*

  multiprocessing_announce

5.4 Directives

This section details the interrupt manager's directives. A subsection is dedicated to each of this manager's directives and describes the calling sequence, related constants, usage, and status codes.

5.4.1 INTERRUPT_CATCH - Establish an ISR

CALLING SEQUENCE:

```
procedure interrupt_catch (
   new_isr_handler : in     rtems.address;
   vector          : in     rtems.vector_number;
   old_isr_handler :    out rtems.address;
   result          :    out rtems.status_codes
);
```

DIRECTIVE STATUS CODES:

SUCCESSFUL	ISR established successfully
INVALID_NUMBER	illegal vector number
INVALID_ADDRESS	illegal ISR entry point

DESCRIPTION:

This directive establishes an interrupt service routine (ISR) for the specified interrupt **vector** number. The **new_isr_handler** parameter specifies the entry point of the ISR. The entry point of the previous ISR for the specified vector is returned in **old_isr_handler**.

NOTES:

This directive will not cause the calling task to be preempted.