3

Interrupt Processing

3.1 Introduction

Different types of processors respond to the occurence of an interrupt in their own unique fashion. In addition, each processor type provides a control mechanism to allow for the proper handling of an interrupt. The processor dependent response to the interrupt modifies the current execution state and results in a change in the execution stream. Most processors require that an interrupt handler utilize some special control mechanisms to return to the normal processing stream. Although **RTEMS** hides many of the processor dependent details of interrupt processing, it is important to understand how the **RTEMS** interrupt manager is mapped onto the processor's unique architecture. Discussed in this chapter are the PA-RISC's interrupt response and control mechanisms as they pertain to **RTEMS**.

3.2 Vectoring of Interrupt Handler

Upon receipt of an interrupt the PA-RISC automatically performs the following actions:

- *The PSW (Program Status Word) is saved in the IPSW (Interrupt Program Status Word).*
- *The current privilege level is set to 0.*
- *The following defined bits in the PSW are set:*
  - E bit is set to the default endian bit

  - M bit is set to 1 if the interrupt is a high-priority machine check and 0 otherwise
  - Q bit is set to zero thuse freezing the IIA (Instruction Address) queues
  - C and D bits are set to zero thus disabling all protection and translation.
  - I bit is set to zero this disabling all external, powerfail, and low-priority machine check interrupts.
  - All others bits are set to zero.

- *General purpose registers r1, r8, r9, r16, r17, r24, and r25 are copied to the shadow registers.*
- *Execution begins at the address given by the formula:* ***Interruption Vector Address + (32 * interrupt vector number)****.*

Once the processor has completed the actions it is is required to perform for each interrupt, the **RTEMS** interrupt management code (the beginning of which is stored in the Interruption Vector Table) gains control and performs the following actions upon each interrupt:

- *returns the processor to "virtual mode" thus reenabling all code and data address translation.*
- *saves all necessary interrupt state information*
- *saves all floating point registers*
- *saves all integer registers*
- *switches the current stack to the interrupt stack*
- *dispatches to the appropriate user provided interrupt service routine (****ISR****). If the* ***ISR*** *was installed with the* ***interrupt_catch*** *directive, then it will be executed at this. Because, the* ***RTEMS*** *interrupt handler saves all registers which are not preserved according to the calling conventions and invokes the application's* ***ISR****, the* ***ISR*** *can easily be written in a high-level language.*

RTEMS refers to the combination of the interrupt state information and registers saved when vectoring an interrupt as the **Interrupt Stack Frame** (**ISF**). A nested interrupt is processed similarly by the PA-RISC and **RTEMS** with the exception that the nested interrupt occurred while executing on the interrupt stack and, thus, the current stack need not be switched.

3.3 Interrupt Stack Frame

The PA-RISC architecture does not alter the stack while processing interrupts. However, **RTEMS** does save information on the stack as part of processing an interrupt. This following shows the format of the **Interrupt Stack Frame** for the PA-RISC as defined by **RTEMS**:

Interrupt Context	0xXXX
Integer Context	0xXXX
Floating Point Context	0xXXX

3.4 External Interrupts and Traps

In addition to the thirty-two unique interrupt sources supported by the PA-RISC architecture, **RTEMS** also supports the installation of handlers for each of the thirty-two external interrupts supported by the PA-RISC architecture. Except for interrupt vector 4, each of the interrupt vectors 0 through 31 may be associated with a user-provided interrupt handler. Interrupt vector 4 is used for external interrupts. When an external interrupt occurs, the **RTEMS** external interrupt handler is invoked and the actual interrupt source is indicated by status bits in the EIR (External Interrupt Request) register. The **RTEMS** external interrupt handler (or interrupt vector four) examines the EIR to determine which interrupt source requires servicing.

RTEMS supports sixty-four interrupt vectors for the PA-RISC. Vectors 0 through 31 map to the normal interrupt sources while **RTEMS** interrupt vectors 32 through 63 are directly associated with the external interrupt sources indicated by bits 0 through 31 in the EIR.

The exact set of interrupt sources which are checked for by the **RTEMS** external interrupt handler and the order in which they are checked are configured by the user in the **CPU Configuration Table**. If an external interrupt occurs which does not have a handler configured, then the spurious interrupt handler will be invoked. The spurious interrupt handler may also be specifiec by the user in the **CPU Configuration Table**.

3.5 Interrupt Levels

Two levels (enabled and disabled) of interrupt priorities are supported by the PA-RISC architecture. Level zero (0) indicates that interrupts are fully enabled (i.e. the I bit of the PSW is 1). Level one (1) indicates that interrupts are disabled (i.e. the I bit of the PSW is 0). Thirty-two independent sources of external

interrupts are supported by the PA-RISC architecture. Each of these interrupts sources may be individually enabled or disabled. When processor interrupts are disabled, all sources of external interrupts are ignored. When processor interrupts are enabled, the EIR (External Interrupt Request) register is used to determine which sources are currently allowed to generate interrupts.

Although **RTEMS** supports 256 interrupt levels, the PA-RISC architecture only supports two. **RTEMS** interrupt level 0 indicates that interrupts are enabled and level 1 indicates that interrupts are disabled. All other **RTEMS** interrupt levels are undefined and their behavior is unpredictable.

3.6 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 external interrupts by setting the I bit in the PSW to 0 before the execution of this section and restores them to the previous level upon completion of the section. **RTEMS** has been optimized to insure that interrupts are disabled for less than **XXX** instructions when compiled with GNU CC at optimization level 4. The exact execution time will vary based on the based on the processor implementation, amount of cache, the number of wait states for primary memory, and processor speed present on the target board.

Non-maskable interrupts (**NMI**) such as high-priority machine checks 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.