EE458 - Embedded Systems
Exceptions and Interrupts

- Outline
  - Exceptions
  - Interrupts

- References
  - RTC: Chapters 10
  - CUG: Chapters 6, 17, 22
Exceptions and Interrupts

Introduction

- An exception is any event that disrupts the normal execution of the processor and forces the processor to execute special instructions. Exceptions can be synchronous or asynchronous.

- Synchronous exceptions are raised by internal events: division by zero, illegal instructions, memory access errors, etc. The term exception is often used to refer to synchronous exceptions.
Exceptions and Interrupts

Introduction

- Asynchronous exceptions are raised by external events. Examples: reset, timer interrupt. The term interrupt is often used to refer to asynchronous exceptions.

- Many processors have two modes of execution: normal and privileged. Privileged mode is normally used during exception processing. Special instructions are often available in privileged mode.
Exceptions and Interrupts

Prog. Interrupt Controller (PIC)

- Most embedded systems use a programmable interrupt controller (PIC) to prioritize multiple interrupts.
- A PIC has a set of interrupt lines, one line for each interrupt source. The PIC uses a single interrupt line to signal the CPU that an interrupt has occurred. The PIC also passes an interrupt vector (a number) which corresponds to the particular interrupt source to the CPU.
Exceptions and Interrupts
Prog. Interrupt Controller (PIC)

- The interrupt vector identifies a particular entry in the interrupt vector table. The entry contains the address of the interrupt service routine (ISR).

- The PIC allows priorities to be assigned to the interrupts and can allow **nested** interrupts. (A higher priority interrupt can preempt the processing of a low priority interrupt.)
Exceptions and Interrupts
Exception Processing

- When an exception occurs the processor automatically saves the current processor state to the stack, loads the address of the exception handler into the program counter, begins executing the handler.

- On a return from exception the processor restores the processor state from the stack and resumes execution of the original process.
Exceptions and Interrupts
Exception Processing

- An exception handler will typically switch to an exception frame (an interrupt stack), save additional processor state on the new stack, and disable lower priority interrupts.

- A handler should perform a minimum amount of work. A task should be used to complete the main processing.

- The handler should restore the original processor state before returning.
Exceptions and Interrupts

RTEMS Exceptions

- RTEMS uses the term interrupt to refer to asynchronous exceptions (interrupts) and uses the term **fatal error** to refer to synchronous exceptions (exceptions).
Exceptions and Interrupts

RTEMS Interrupt Manager

- ISRs should be installed through the RTEMS interrupt manager. This allows a task switch to occur upon return from an ISR. This is a critical feature of a real-time OS.

- When an interrupt occurs the processor vectors to RTEMS. RTEMS will invoke the user's ISR. The user's ISR is responsible for processing and clearing the interrupt.
Exceptions and Interrupts
RTEMS Interrupt Directives

• Interrupt processing is necessarily processor specific, see the RTEMS BSP supplemental documentation for information.

• There are five RTEMS interrupt related directives. You can establish an ISR, disable and enable interrupts, flash interrupts, and check to see if an ISR is in progress.
Exceptions and Interrupts
RTEMS Interrupt Directives

- The `rtems_interrupt_catch()` directive is used to establish an ISR for a particular interrupt vector number.

- The `rtems_interrupt_disable()` and `rtems_interrupt_enable()` directives are used to disable and enable interrupts. This pair provides one means of protecting (short!!!) critical sections.

- Refer to Chapter 6 of the CUG for details.
Exceptions and Interrupts
RTEMS Fatal Error Processing

• All fatal errors (synchronous exceptions) are processed by the RTEMS fatal error manager. The manager invokes the `rtems_fatal_error_occurred()` directive. (Refer to CUG Chapter 17.)

• The fatal error manager can optionally invoke a user supplied fatal error handler installed via the RTEMS User Extensions Manager (CUG Chapter 22)