EEL 4744C: Microprocessor Applications

Lecture 7

Part 1

Interrupt

Reading Assignment

• M&M: Chapter 8
  Or
  • Software and Hardware Engineering (new version): Chapter 12

An interrupt is an important asynchronous event that needs to be recognized and dealt with by the CPU that is executing instructions in a program.

You can keep polling the external devices, e.g. sensors, detectors, etc. But doing this periodically takes time away from your system and reduce overall performance.

Use interrupts! Your external sensor, when it detects something peculiar, generates a signal called interrupt request, i.e. IRQ.
Interrupt

- The IRQ is asynchronous, i.e. the time it happens has no relation to the instructions being executed by the CPU
- The IRQ requires an ISR, or interrupt service routine, to be executed immediately. The ISR should be part of the running program
- Interrupts can also be used to synchronize a real-time system to control exactly when things are performed by the system, e.g. A/D IRQ to CPU signaling completion of A-to-D conversion

Interrupt System Specifications

- Allow for asynchronous events to occur and be recognized
- Wait for the current instruction to finish before taking care of any interrupt
- Branch to the correct ISR to service the external device that issued the IRQ
- Return to the interrupted program where it was interrupted
- Allow for a variety of interrupting signals, including signal levels and edges (rising/falling)
- Signal the interrupting device with ACK signal when its IRQ is recognized

Interrupt System Specifications

- Allow the programmer to selectively enable and disable all interrupts
- Allow the programmer to selectively enable and disable selected interrupts
- Disable further interrupts while the first is being serviced
- Deal with multiple sources of interrupts
- Deal with multiple simultaneous interrupts

Process Control with Interrupts

- ISR is executed when an IRQ occurs

Dr. Tao Li
**Interrupts and Sequence Control**

- The sequence controller can be modified to check for an IRQ before fetching the next instruction.
- More states can be added to the sequential state machine to cover IRQ sampling, IRQ ACK, etc.

**Interrupt Hardware**

- IRQ is usually active low, and many IRQ's are often wire-ORed together.
- An IRQ flip-flop (IRQ-FF) is used to “catch” and “remember” an IRQ until the CPU gets a chance to acknowledge and service it.
- When IRQ-FF is set, it generates a signal for the pending interrupt (a.k.a. the IRQ that has not been acted upon). This signal is sent toward the sequence controller.

**Interrupt Hardware**

- The IRQ-FF is reset when it receives the INTA (a.k.a. interrupt acknowledge) signal from CPU.
- The interrupt enable flip-flop (INTE-FF) allows the programmer to enable or disable interrupts with program instructions.
- Setting the INTE-FF enables interrupts through the AND gate. When the INTE-FF is reset, all interrupts are blocked. Input signals of the 2 OR gates give the conditions to reset the INTE-FF.

**Interrupt Hardware**

- The CPU generates an interrupt acknowledge (INTA) signal when: (1) the current instruction has finished execution and (2) CPU has detected the IRQ.
- INTA tells the interrupting device that the CPU is ready to execute the ISR.
- Upon completion the ISR, the CPU executes the return-from-interrupt instruction.
- Can you have interrupts interrupting interrupts?
Interrupt Recognition and Acknowledgement Hardware

- Determine which device generated the IRQ in order to execute the correct ISR
- Prioritize simultaneous IRQ's

Interrupt Polling

- A software process where CPU reads each of the potential interrupting device’s status register to find the device that generated the IRQ

Vectored Interrupts

- A vector, in this case, is an address, the address of the ISR to be executed
- In response to the CPU’s INTA signal, the device places a “vector” onto the data bus for the CPU to read. This vector identifies the device itself
- Can also have multiple IRQ input pins on CPU, each pin with its own memory locations for a vector dedicated to each IRQ signal
- Faster than polling, but needs more hardware
**Vectored Interrupts**

- Use a mask register to enable a specific IRQ

**Multiple Interrupt Masking**

- Use a mask register to enable a specific IRQ

**Sequential vs. Nested Interrupts**

- **Sequential**: A pending IRQ prevents further interrupts from being recognized by CPU since INTA resets the INTE-FF
- **Nested interrupts**: Programmers can enable this by including the “enable interrupt instruction” within the ISR
- **The program has to keep track of the status of INTE-FF, re-enabling the FF if necessary at the end of the current ISR**

**Prioritizing Simultaneous IRQs**

- **Software priority resolution**: Polling order fixes the priority for each device. So a lower-priority device must keep asserting its IRQ until it is recognized and serviced
- **Hardware priority resolution**: Required for vectored interrupts, several methods possible
- **Daisy chain**: Similar to that used for parallel I/O
- **Separate IRQ lines**: IRQ0 > IRQ1 > IRQ2 > ...
**Prioritizing Simultaneous IRQs**

- Hierarchical prioritization: Based on a hierarchy of high-to low-priority IRQs.

- Nonmaskable interrupts: These are reserved for critical events, e.g. power failure, that must not be masked (i.e. disabled). Usually this requires an IRQ line that is not accessible by programmers.

- Programmable interrupt controllers: Can resolve IRQ priorities and be programmed to generate a vector for several interrupting sources. For example, the Motorola MC6828.

**Transfer of Control to ISR**

- Context saving: Before executing the ISR, the CPU pushes all the register and flag values onto the stack.

- Interrupt latency: The delay from when the IRQ is generated to when the ISR begins to execute.

- This depends on many factors, e.g. length of the current instruction, amount of data pushed onto the stack, existence of a higher-priority IRQ and its corresponding ISR that must be finished first.

**Tips on ISR Design**

- Minimize the amount of data pushed onto the stack, and don't modify any registers or flags in the ISR.

- Re-enable or unmask interrupts at the end of the ISR if this is not automatically done by the CPU.

- Restore registers if this is not automatically done.

- Use the proper ISR return instruction.

- Keep the ISR short!

**ISR Return**

- An ISR return is different from a normal subroutine return, because this return instruction needs to perform other interrupt-related tasks. For example, we have RTS and RTI.
• Nonmaskable interrupts (NMI): For example, a power failure NMI is asserted before the power supply voltage goes down to 0. Then, using the very little time left, the CPU runs a small ISR saving critical data to nonvolatile memory!

• Software interrupts (SWI): An instruction that activates interrupt processing without an IRQ

• SWI is frequently used for breakpoint instructions in debuggers

• Exceptions: Occurs during run time, e.g. internal bus errors, division by zero, etc. The programmer can supply an ISR dealing with an exception

• Reset: CPU reset, nonmaskable, done through a signal asserted on a separate pin