Exceptions

- In computer systems we may NOT know when
  - –External hardware events will occur.
- Can you think of an example?
  - – Errors will occur
- Exception processing refers to
  - – Handling events whose timing we cannot predict
- 3 questions to answer:
  - – Q: Who detects these events and how? A: The hardware
  - – Q: How do we respond? A: Calling a pre-defined SW function
  - – Q: What is the set of possible events? A: Specific to each processor
- Definition: Any event that causes a break in normal execution
  - – "Exceptions" is a broad term to catch many kinds of events that interrupt normal software execution

Interrupt exceptions

- Two methods for processor and I/O devices to notify each other of events
  - – Polling loop or “busy” loop (responsibility on proc.)
    - Processor has responsibility of checking each I/O device
    - Many I/O events happen infrequently (1-10 ms) with respect to the processors ability to execute instructions (1-100 ns) causing the loop to execute many times
  - – Interrupts (responsibility on I/O device)
    - I/O device notifies processor only when it needs attention

Instruction cycle

- Fetch
- Decode
- Execute
- Check for exceptions

Software Handles Exceptions

1. Save place in current code and disable other interrupts
2. Automatically have the processor call some function/subroutine to handle the issue (a.k.a. Interrupt Service Routine or ISR)
3. Reenable interrupts & resume normal processing back in original code
   - HW detects exceptions.
   - Software handles exceptions.

We must tell the processor in advance which function to associate (i.e. call) with the various exceptions it will check for

Function Calls vs. Interrupts

- Normal function calls
  - Synchronous: Called whenever the program reaches that point in the code
  - Programmer can pass arguments and receive return values
- Interrupts
  - Asynchronous: Called whenever an event occurs (can be anywhere in our program when the ISR needs to be called)
    - – Requires us to know in advance which ISR to call for each possible exception/interrupt
    - – Use ISR(interrupt_type) naming scheme in the Arduino to make this association
  - No arguments or return values
    - – How would we know what to pass if we don't know when it will occur

– Generally interrupts update some global variables

Enabling Interruptions

- Each interrupt source is DISABLED by default and must be ENABLED
- For an interrupt to be handled, two "enablers" need to agree
  - – Enabler 1: A separate "local" interrupt enable bit per source (i.e. ADC, timer, pin change, etc.)
  - – Enabler 2: A single "global" interrupt enable bit (1-bit for entire processor called the I-bit)