**1. Differences between running code with and without interrupts**

**With Interrupts:**

* The microcontroller processes sensor readings in the background using a timer interrupt, independent of the main loop.
* Even if the main loop is delayed (for example, using time.sleep()), the sensor data updates consistently according to the timer interval.
* **Observation:** During the test, with interrupts set to read data every 2 seconds and the main loop set to time.sleep(10), the console showed updated readings every 2 seconds, proving the sensor was actively measuring in the background. However, the OLED only refreshed after the 10-second delay in the main loop, causing a visible lag in display updates.

**Without Interrupts:**

* The main loop directly controls when the sensor is read and the display is updated, with no background processing.
* Any delay (like time.sleep()) completely halts all operations, including sensor readings.
* **Observation:** With a 10-second delay in the main loop, both the console and OLED updated only every 10 seconds, showing no background activity.

**Key Difference:**

Interrupts allow the microcontroller to handle tasks simultaneously — the sensor can update independently of the main loop — whereas without interrupts, all tasks are executed sequentially.

**2. Explanation of the debounce issue and its impact**

**What is Debounce?**

Debouncing is the process of eliminating false or repeated signals caused by mechanical switches or sensors. When a button is pressed or released, the physical contacts may bounce, generating multiple signals instead of a single one.

**Why does it happen?**

* Mechanical vibrations when the contacts touch or separate.
* Poor hardware quality or fast signal reading.

**Impact on Applications:**

* Without debouncing, the microcontroller may register multiple events for a single button press, causing unwanted behavior.
* For example, a single press to turn on an LED might be read as multiple presses, causing the LED to flicker or switch states unpredictably.

**Solution:**

* **Software debounce:** Adding small delays (like time.sleep(0.01)) to wait for the bouncing to stop.
* **Hardware debounce:** Using capacitors or Schmitt triggers to smooth out signals.

**3. Why interrupts are used and how they reduce processing costs**

**Why use interrupts?**

Interrupts allow the microcontroller to respond immediately to critical events (like sensor updates or button presses) without constantly checking for them in the main loop.

**How do interrupts reduce processing costs?**

* **Efficiency:** The microcontroller can perform other tasks or enter low-power mode until an interrupt occurs.
* **Speed:** The CPU doesn’t waste time repeatedly polling sensors; it only acts when something happens.
* **Multitasking:** Tasks like refreshing a display, reading a sensor, and responding to user input can happen simultaneously.

**Example from Lab:** In our lab task, interrupts allowed the DHT11 sensor to update every 2 seconds regardless of the main loop's delay. This means the microcontroller didn't have to wait — it continued reading data without blocking other operations.