003

Interrupt and Exception Handling

Operating Systems

Lab003 Interrupts and Exception Handling

Building a Bare-Metal ARM Application with Timer Interrupt Handling

Objective

The goal of this lab is to introduce students to **interrupt-driven programming** on ARM architectures. By working with **timer interrupts**, students will:

- Learn how to configure **ARM's timer peripheral** for periodic execution.
- Understand **Interrupt Vector Table (IVT) handling** and how to service interrupts in assembly.
- Implement **interrupt-driven execution** instead of relying on busy loops.
- Gain insight into how **low-level exception handling** works in ARM-based systems.

Overview

In this lab, you will extend the existing **bare-metal framework** by implementing a **hardware timer interrupt** that triggers every **2 seconds**. Instead of relying on manual delay loops, the system will execute code when the timer reaches zero and raises an **IRQ (Interrupt Request).**

Your task is to:

1. **Modify os.c to handle timer interrupts**, clearing them appropriately.
2. **Update main.c to initialize the timer and enable interrupts**.
3. **Expand root.s to ensure the Interrupt Service Routine (ISR) properly handles the timer event**.
4. **Observe the execution using QEMU** to confirm that the interrupt fires at the correct interval.

Project Structure

The existing codebase is structured into **three layers**, which you must continue following:

1. User Level (Application Logic)

- **File:** main.c
- **Purpose:** Implements the main logic of the system.
- **Your Task:**
    - Set up and enable the **timer interrupt**.
    - Print system messages before and after enabling interrupts.
    - Continuously **monitor and print timer values** to confirm it decrements.
    - Ensure the timer **automatically restarts** without manual intervention.

2. **OS Level (Interrupt Handling & Hardware Interface)**
   - **File:** os.c
   - **Purpose:** Provides an interface to hardware and manages **low-level OS-like functionalities**.
   - **Your Task:**
     - Implement **a function to configure the timer peripheral**.
     - Implement **VIC (Vector Interrupt Controller) setup** to enable IRQs.
     - Implement **the actual timer interrupt handler**, ensuring it clears the interrupt flag.
     - Add **support for multiple interrupts** (e.g., handling both software and timer interrupts).

3. **Low-Level Hardware Interface (Exception Handling)**
   - **File:** root.s
   - **Purpose:** Manages the **ARM Exception Vector Table** and routes IRQs to the correct handlers.
   - **Your Task:**
     - Ensure the **IRQ vector points to the correct handler**.
     - Preserve CPU registers inside the ISR before modifying them.
     - Ensure the ISR **acknowledges the interrupt** and resumes normal execution.

Assignment Tasks

1. **Understanding the Codebase**
   - Review how **the current system prints messages via UART**.
   - Identify **how os.c currently interacts with hardware**.
   - Locate **where new interrupt functionality needs to be inserted**.

2. **Implement Timer Initialization (os.c)**
   - Add a function that:
     - Loads a countdown value (e.g., **2 seconds**) into the timer.
     - Configures the timer in **periodic mode**.
     - Enables **interrupt generation**.
     - Ensures the timer **auto-reloads** upon reaching zero.

3. **Configure VIC for Interrupt Handling (os.c)**
   - Implement a function that:
     - Enables **IRQ #4 (Timer IRQ)** in the **VIC (Vector Interrupt Controller)**.
     - Ensures the VIC can **route the interrupt to the CPU**.
     - Provides a mechanism to **manually clear** and acknowledge interrupts.

4. Implement the IRQ Handler (root.s)

- Modify the **vector table** to ensure that IRQs are routed to the correct handler.
- Implement an **IRQ service routine** that:
    - **Reads the VIC to determine the interrupt source**.
    - Calls the appropriate **interrupt handler in os.c**.
    - **Acknowledges the interrupt** and clears flags before returning.

5. Modify main.c to Initialize and Test the Interrupt

- Ensure main.c:
    - Configures and enables the timer.
    - Prints system messages **before and after enabling IRQs**.
    - Continuously prints the timer countdown value.
    - Monitors **whether the interrupt fires correctly every 2 seconds**.

Validation & Debugging

Successful Behavior

- The system **prints an initialization message**.
- The **timer value prints continuously** and decrements over time.
- Every **2 seconds**, a message (e.g., "Timer Interrupt Triggered") appears.
- The timer **resets automatically** after reaching zero.
- The system does **not freeze or crash**.

Common Issues

Issue	Likely Cause	Solution
Interrupt never fires	VIC is not configured correctly	Verify VIC_INTENABLE register
System hangs after enabling IRQs	CPU is not acknowledging interrupts	Ensure VIC_VADDR is written to in the ISR
Timer counts down but does not restart	Timer is not in periodic mode	Ensure **periodic mode** is enabled in TIMER_CONTROL
Multiple interrupt triggers per second	Timer is not properly clearing the interrupt	Ensure TIMER_INTCLR = 1 is written after servicing the IRQ

Deliverables

1. **Updated Source Files**
    - main.c: Implements timer setup and testing.
    - os.c: Configures the timer, VIC, and interrupt handlers.
    - root.s: Handles IRQ vectoring.

2. **Documentation**
    - Explanation of **how the timer interrupt works**.
    - Description of **each modified function**.
    - Explanation of **how the system handles IRQs**.

3. **Execution Logs**
    - Output logs showing the **timer countdown** and **interrupt triggers**.
    -

Expected Outcomes

By the end of this lab, students will:

- Understand **how hardware timers work** in ARM-based systems.
- Implement **interrupt-driven execution** instead of busy loops.
- Successfully **route IRQs from the VIC to the CPU**.
  Develop a deeper understanding of **low-level exception handling** in ARM.

Tips for Success

- **Test Incrementally**:
    - First, get the **timer countdown working**.
    - Then, **verify the IRQ fires**.
    - Finally, ensure **the system acknowledges and clears interrupts correctly**.

- **Use Debugging Prints**:
    - Add print statements **before and after enabling IRQs**.
    - Print **VIC IRQ status before and after clearing the interrupt**.

- **Check Register Values**:
    - If an interrupt is not firing, **print the VIC & Timer registers** to diagnose the issue.

Additional Resources

ARM Interrupt handlers
IRQ Exception
Cortex-A8 (GES)