Guide to ARM Interrupts

Core Concept: What is an Interrupt?

An **interrupt** is a signal to the CPU that requires immediate attention. It suspends the current task, saves its state, and executes a special function called an **Interrupt Service Routine (ISR)**. This is far more efficient than **polling**, which wastes CPU cycles constantly checking for events.

ARM Exception Handling

Interrupts are a type of **exception** in ARM. When an exception occurs, the CPU jumps to a specific address in the **Interrupt Vector Table (IVT)**.

Offset	Exception	Purpose
`$0x18`	**IRQ (Interrupt Request)**	Primary, general-purpose hardware interrupts.
`$0x1C`	**FIQ (Fast Interrupt Request)**	Higher-priority, lower-latency interrupts.

- **IRQ vs. FIQ:**
  - **Priority: FIQ** is higher priority than IRQ and can interrupt an IRQ handler.
  - **Latency:** FIQ is faster due to more **banked registers** (`r8`-`r12`, `SP`, `LR`), which means the hardware automatically swaps them, avoiding the need to save and restore them in software.
  - **Use Case:** IRQ is for general use (keyboard, timers). FIQ is for critical, high-speed tasks (DMA, data streams).

The Link Register (`LR`) Explained

The `LR` is used for returning but behaves differently in function calls versus interrupts.

1. **Normal Function Call (`BL`):**
   - **Trigger:** A software `BL` (Branch with Link) instruction.
   - **Action:** Stores the return address (the next instruction) in the `LR`.
   - **Return:** A `BX LR` instruction copies `LR` back to the Program Counter (`PC`).

2. **Hardware Interrupt:**
    - **Trigger:** An external hardware signal.
    - **Action:** The CPU **hardware automatically** saves the return address of the *interrupted program* into the **banked `LR` of the exception mode** (e.g., `LR_irq`).
    - **Return:** A special return instruction restores the `PC` from `LR_irq` and also restores the processor's original state.

> **Key Takeaway:** The `LR` in an ISR holds the gateway back to the suspended program. If your ISR calls another function, you **must** save this `LR` to the stack first.

Interrupt Controllers: GIC vs. NVIC

These controllers manage and prioritize interrupts before they reach the CPU.

GIC (Generic Interrupt Controller) - For Cortex-A

Used in high-performance application processors (e.g., smartphones, servers).

- **Architecture:**
    1. **Distributor (GICD):** The central hub. Configures, prioritizes, and routes interrupts from all peripherals.
    2. **CPU Interface (GICC):** A private interface for each CPU core. Presents interrupts to the core and allows the ISR to acknowledge them and signal completion.
- **End-to-End Flow:**
    1. Peripheral asserts its IRQ line to the GIC Distributor.
    2. Distributor marks the interrupt as **pending**, prioritizes it, and forwards it to the target CPU Interface.
    3. CPU Interface asserts the CPU's `IRQ` pin.
    4. CPU saves context and jumps to the single IRQ vector (`$0x18`).
    5. **ISR reads the `GICC_IAR` register** to acknowledge the interrupt and get its unique **Interrupt ID**.
    6. ISR uses the ID to run the correct driver, service the device, and clear the device's interrupt flag.
    7. **ISR writes to the `GICC_EOIR` register** to signal End of Interrupt.
    8. CPU restores context and resumes the main program.

NVIC (Nested Vectored Interrupt Controller) - For Cortex-M

Used in real-time microcontrollers. Optimized for speed and low latency.

- **Key Feature:** True **vectored interrupts**. The vector table contains a direct pointer to the ISR for **every single interrupt source**.

- **End-to-End Flow:**
    1. Peripheral asserts its IRQ line to the NVIC.
    2. NVIC prioritizes the interrupt and provides its **vector number** to the CPU.
    3. CPU **hardware automatically** saves the core registers to the stack.
    4. CPU uses the vector number to fetch the ISR address **directly** from the vector table and jumps to it. No software lookup is needed.
    5. ISR services the device and clears the device's interrupt flag.
    6. ISR executes a return instruction.
    7. CPU **hardware automatically** restores registers from the stack and resumes the main program.

Advanced Interrupt Concepts

- **Top-Half vs. Bottom-Half:** A software pattern to keep ISRs fast.
    - **Top-Half (The ISR):** Does the absolute minimum, time-critical work (e.g., clear flag, copy data from a buffer). Then it schedules the "bottom-half."
    - **Bottom-Half (Deferred Task):** A normal software task that runs later to do the heavy processing (e.g., passing a network packet up the TCP/IP stack).
- **Security (TrustZone):** The GIC divides interrupts into **Secure** (Group 0, for trusted code) and **Non-secure** (Group 1, for the normal OS). A Secure interrupt can always preempt Non-secure code. FIQs are often reserved for Secure interrupts.
- **Virtualization:** The GIC can trap physical interrupts and **inject virtual interrupts** into guest operating systems, allowing multiple OSs to run on the same hardware without interfering with each other.
- **Message Signaled Interrupts (MSI/MSI-X):** Used for peripherals like PCIe. Instead of a physical wire, a device triggers an interrupt by writing a "message" to a special memory address, which the GIC then converts into a normal interrupt. This is highly scalable.