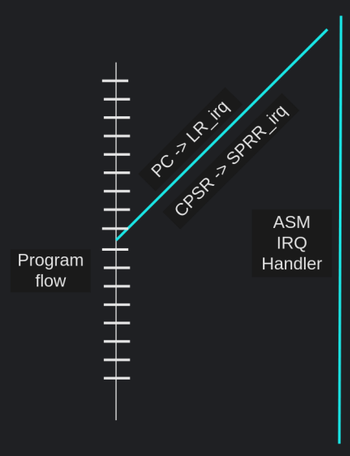
Simplistic interrupt handling

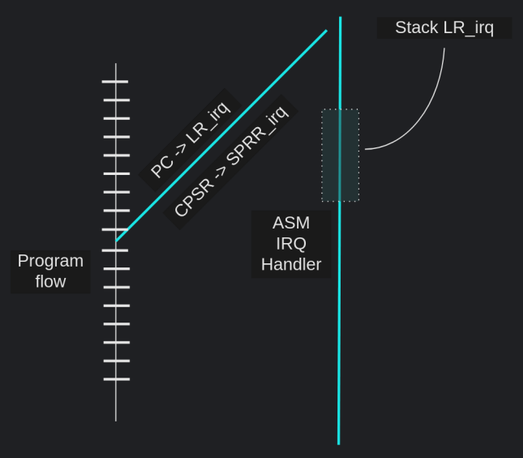
This represents the simplest kind of interrupt handler. On taking an interrupt, additional interrupts of the same kind are disabled until explicitly enabled later. We can only handle additional interrupts at the completion of the first interrupt request and there is no scope for a higher priority or more urgent interrupt to be handled during this time. This is not generally suitable for complex embedded systems, but it is useful to examine before proceeding to a more realistic example, in this case of a non re-entrant interrupt handler.

The steps taken to handle an interrupt are as follows:

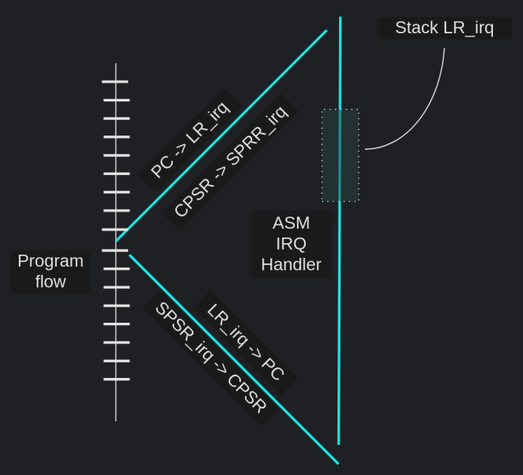
1. An IRQ exception is raised by external hardware. The core performs several steps automatically. The contents of the PC in the current execution mode are stored in LR_IRQ. The CPSR register is copied to SPSR_IRQ. The CPSR content is updated so that the mode bits reflects the IRQ mode, and the I bit is set to mask additional IRQs. The PC is set to the IRQ entry in the vector table.
   Figure 12.1. Save the context of the program



2. The instruction at the IRQ entry in the vector table (a branch to the interrupt handler) is executed.
3. The interrupt handler saves the context of the interrupted program, that is, it pushes onto the stack any registers that will be corrupted by the handler. These registers will be popped from stack when the handler finishes execution.
   Figure 12.2.



4. The interrupt handler determines which interrupt source must be processed and calls the appropriate device driver.
5. Prepare the core to switch to previous execution state by copying the SPSR_IRQ to CPSR, and restoring the context saved earlier, and finally the PC is restored from LR_IRQ.
   Figure 12.3.



The same sequence is also applicable to an FIQ interrupt.

A very simple interrupt handler is shown in Example 12.1.

```
IRQ_Handler
    PUSH    {r0-r3, r12, lr}    @ Store AAPCS registers and LR onto the IRQ mode stack
    BL                          @ identify_and_clear_source
    BL                          @ C-irq_handler
    POP     {r0-r3, r12, lr}    @ Restore registers and
    SUBS    pc, lr, #4          @ return from exception using modified LR
```

AI
Automotive
Embedded and Microcontrollers
Internet of Things
Laptops and Desktops
Mobile, Graphics, and Gaming
Servers and Cloud Computing

Documentation
Learning Paths
On-Demand Videos

Blogs
Communities
Blogs
Developer Events
Discord
Forums
Groups

Open a Support Case
My Support Cases
Contact Support
Product Support Forums
Documentation
Downloads
Licensing
Support and Training Options

Architectures
Compare IP
Compute Subsystems
Corstone
Memory Model Tool
Performance Analysis
Processors
Reference Designs

Leadership
Arm Offices
Careers
Contact Us
Newsroom
Trust Center