Interrupts

How hardware asks the CPU for attention — and how the CPU answers.

youtube.com/ElhosseiniAcademy

Prof. Mostafa Elhosseini — Professor of Smart Systems Engineering

Agenda

01 Device controller — the hardware circuit inside the device

02 Device driver — the operating-system software that drives it

03 Interrupts — what they are, and why polling loses

04 Interrupt vector table (IVT) — the directory of handlers

05 Interrupt service routine (ISR) — the code that answers

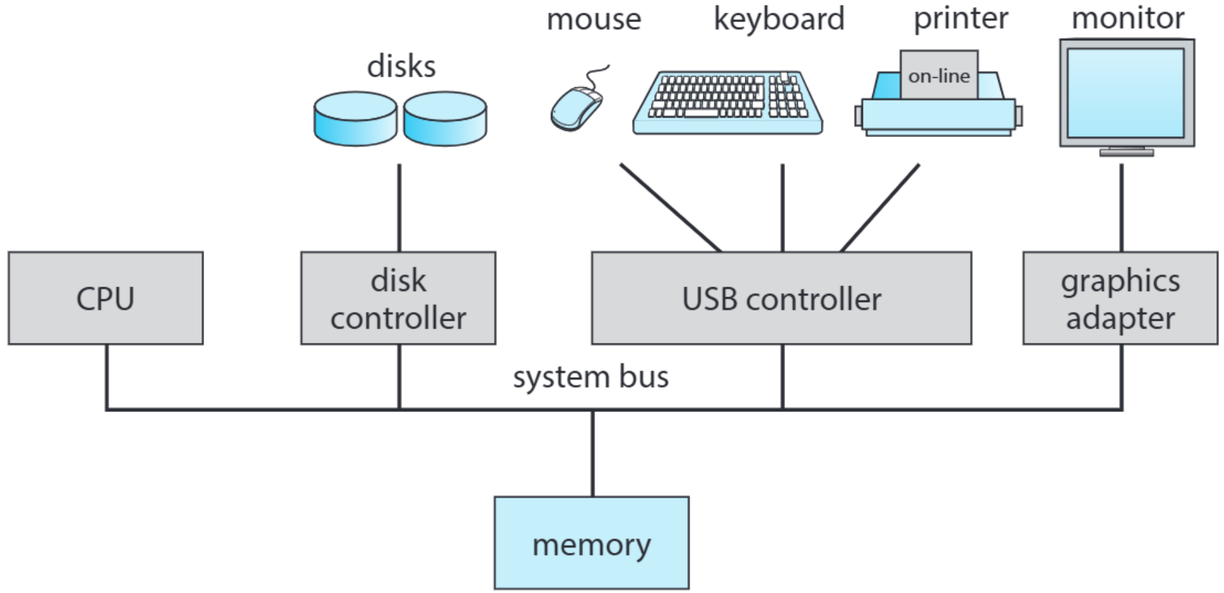
06 Types of interrupts — and the interrupt cycle end to end

PART ONE

The Hardware Path

Controllers, drivers, and the signal that travels up the system bus.

Computer-System Organization



One or more **CPUs** and a number of **device controllers** connect through a common **bus** that reaches shared memory.

Each controller is in charge of one type of device — disk, USB, audio, display — and one port may host many devices through a hub.

CPU and controllers run **in parallel** , competing for memory cycles; a memory controller keeps that access orderly.

Device Controller and Device Driver

The **device** is the actual hardware — keyboard, printer, disk, mouse. The controller sits inside it; the driver lives inside the operating system.

HW

Device controller

A hardware circuit with **local buffer storage** and **special-purpose registers**.

Moves data between the peripheral it controls and that local buffer.

Examples: disk controller, keyboard controller, USB controller.

متحكم الجهاز: دائرة إلكترونية داخل كل جهاز إدخال/إخراج، يدير عمل الجهاز ويتواصل مع وحدة المعالجة.

SW

Device driver

Operating-system code — normally **one driver per controller** .

Gives the rest of the kernel a **uniform interface** to a very non-uniform device.

Examples: printer driver, graphics driver, network driver.

برنامج تشغيل الجهاز: برنامج داخل نظام التشغيل يترجم الأوامر عالية المستوى إلى أوامر منخفضة المستوى يفهمها متحكم الجهاز.

In short. The driver (software) writes to the controller's registers; the controller (hardware) executes the command.

Analogy. Driver = interpreter, translating the OS. Controller = manager, giving the device its orders.

Starting an I/O Operation

1. The **device driver** loads the appropriate registers in the device controller.
2. The **controller** examines those registers to decide what action to take — `read a character from the keyboard`
3. The controller transfers data from the device into its **local buffer**.
4. The driver hands control back to the OS with data, a pointer, or status — `write completed successfully`

But how does the controller tell the driver it has finished? Via an **interrupt**.

Polling Compared with Interrupts

ASPECT	POLLING	INTERRUPT
Who asks	CPU keeps asking the device	Device signals the CPU
CPU time	Spent on checks that find nothing	Spent on useful work until notified
Nature	Synchronous, program-driven	Asynchronous, event-driven

Analogy. Polling is walking to the mailbox every five minutes. An interrupt is the doorbell.

What an Interrupt Is

Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by way of the system bus.

المقاطعة: إشارة يرسلها جهاز إلى وحدة المعالجة المركزية ليُعلِمها بأنه يحتاج إلى اهتمام أو خدمة.

The CPU **stops what it is doing** and transfers execution to a **fixed location**.

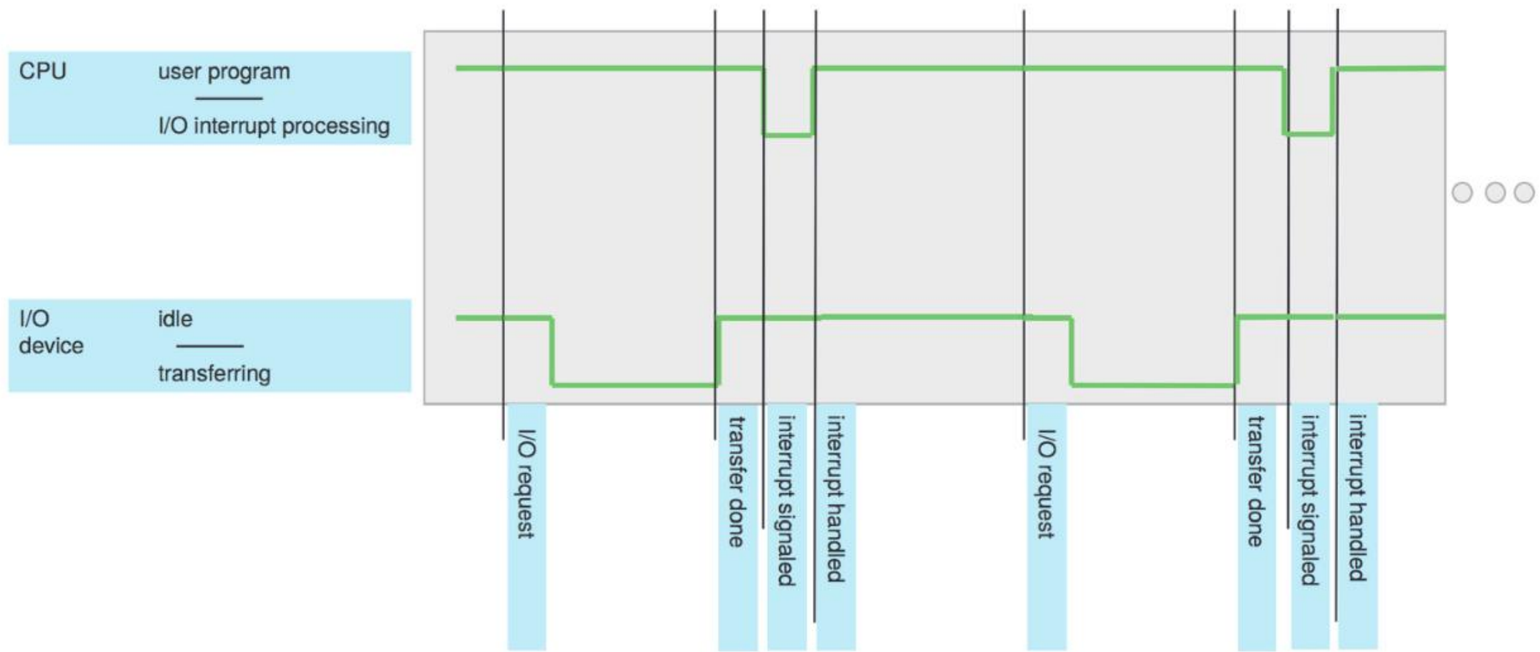
That location holds the starting address of the **interrupt service routine** (ISR).

When the ISR completes, the CPU **resumes the interrupted computation** .

Interrupts serve many other purposes too — they are a key part of how operating systems and hardware interact.

In one line. The device raises a signal, the CPU responds, the ISR handles it, the CPU goes back to work.

Interrupt Timeline for a Single Program



I/O request — the driver starts the device; the CPU returns to the user program.

Interrupt signaled — transfer done, so the controller raises the signal.

Interrupt handled — a short dip into I/O processing, then back to user code.

PART TWO

Dispatch: Vectors and Service Routines

Finding the right handler, quickly, without losing the interrupted work.

Functions Common to Every Interrupt Mechanism

1

Transfer control

Control must reach the **appropriate** interrupt service routine — and reach it fast, because interrupts occur very frequently.

2

Save state

The architecture must preserve the state of whatever was interrupted, then **restore** it after the interrupt is serviced.

Rejected design. A single generic routine that inspects the interrupt and then calls the specific handler adds a hop on every single interrupt. Too slow.

The Interrupt Vector Table (IVT)

A **table of pointers** replaces the generic routine. The handler is called **indirectly through the table** , with no intermediate routine.

The table sits in **low memory** — the first hundred or so locations — and holds the addresses of the service routines.

The array is **indexed by a unique number** supplied with the interrupt request.

```
INTERRUPT VECTOR (LOW MEMORY)
[ 0 ]              0x0004A2 → divide-error ISR
[ 1 ]                       0x0004C8 → debug ISR
...                                          ...
[ 9 ]                    0x001F30 → keyboard ISR
[14]                   0x002B14 → page-fault ISR

index = interrupt number given with the request
```

Operating systems as different as Windows and UNIX dispatch interrupts in exactly this manner. Addresses above are illustrative.

The Interrupt Service Routine (ISR)

A **small program** that handles one kind of interrupt — the code the vector entry points at.

It **saves CPU state**, determines the **cause** of the interrupt, does the processing, **restores state**, and returns.

Example. The keyboard ISR reads the keystroke from the controller and passes it to the operating system.

```
KEYBOARD_ISR
save registers
read character from controller
deliver it to the OS
clear the interrupt
restore registers
iret
```

In one line. The IVT says *where* to go; the ISR is *what runs* when you get there.

Vector Dispatch: A Keystroke, Step by Step

1. You press a key. The keyboard controller raises an interrupt carrying number `9`
2. The CPU uses that number as an **index** into the interrupt vector.
3. Entry 9 holds the address of the **keyboard ISR**; execution starts there.
4. The ISR reads the character from the controller and passes it to the OS.
5. State is restored and the CPU resumes the interrupted program.

Analogy. A hotel switchboard: the extension number is the interrupt number, and it rings exactly one room — no searching floor by floor.

In one line. IVT = directory of ISRs. ISR = the routine that handles the interrupt.

Saving and Restoring Processor State

The interrupt architecture saves the state of whatever was interrupted, so it can be restored after servicing.

If the routine will **modify register values** , it must explicitly save the current state and restore it before returning.

The saved **return address** is then loaded into the **program counter**.

QUICK RECAP

The interrupted computation resumes as though the interrupt had never occurred.

Save · Service · Restore · Return

The Interrupt-Request Line

The CPU has a wire — the **interrupt-request line** — that it **senses after executing every instruction**.

When a controller has asserted a signal there, the CPU reads the **interrupt number** and jumps through the vector.

The handler then saves state, determines the cause, does the work, restores state, and executes **return from interrupt**.

```
CPU INSTRUCTION LOOP
fetch instruction
execute instruction
if (IRQ line asserted)
  read interrupt number
  jump via interrupt vector
  run handler, iret
repeat
```

Raise, Catch, Dispatch, Clear

R

Raise

The device controller asserts a signal on the interrupt-request line.

C

Catch

The CPU senses the line between instructions and accepts the request.

D

Dispatch

The vector sends control to the matching interrupt handler.

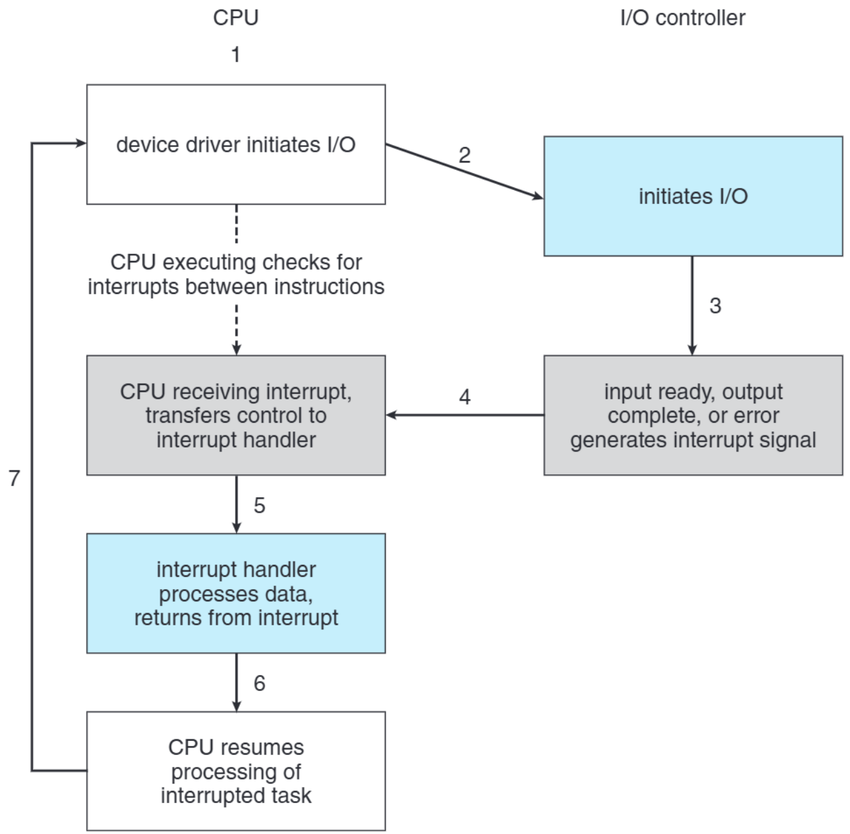
X

Clear

The handler services the device, which drops the signal.

Phone-call mapping. Ring = raise · pick up = catch · transfer to the right person = dispatch · hang up after solving it = clear.

The Interrupt-Driven I/O Cycle



1–2 The device driver initiates I/O; the controller starts the operation.

3 Input ready, output complete, or error — the controller generates the interrupt signal.

4 The CPU, which checks for interrupts between instructions, receives it and transfers control.

5 The interrupt handler processes the data and returns from the interrupt.

6–7 The CPU resumes the interrupted task, ready for the next request.

Meanwhile the CPU never idles waiting on the device — that is the whole point of the cycle.

PART THREE

Features of a Modern Interrupt System

Masking, chaining, and priority — provided by the CPU and the interrupt-controller hardware.

Types of Interrupts

NMI

Nonmaskable

Cannot be disabled. Hardware failure, unrecoverable memory error — handled immediately.

IRQ

Maskable

Can be masked by the CPU. Normal device requests: keyboard, mouse, disk, printer.

LVL

Multilevel

Priority levels, so urgent work is served first and low-priority work can wait.

CHN

Chained

More devices than vector entries, so one entry holds a list of handlers.

In one line. NMI = urgent · maskable IRQ = normal · multilevel = priorities · chaining = linked ISRs.

Three Needs Beyond the Basic Mechanism

1

Defer handling

Interrupt handling must be postponable during critical processing.

→ maskable line

2

Dispatch efficiently

Control must reach the proper handler for a device without a search.

→ vector, chaining

3

Multilevel interrupts

High- and low-priority interrupts must be told apart and answered accordingly.

→ priority levels

In modern hardware all three are provided by the CPU together with the interrupt-controller hardware.

Two Interrupt-Request Lines

LINE 1

Nonmaskable interrupt

Reserved for events such as **unrecoverable memory errors** .

Cannot be turned off — the CPU must respond.

LINE 2

Maskable interrupt

Can be **turned off by the CPU** before critical instruction sequences that must not be interrupted.

This is the line **device controllers** use to request service.

Memory hook. Nonmaskable is the fire alarm; maskable is the desk phone you can silence for ten minutes.

Interrupt Chaining

Computers have **more devices — and handlers — than vector slots** .

So each element of the vector points to the **head of a list** of interrupt handlers.

When an interrupt is raised, handlers on that list are called **one by one** until one can service the request.

ONE VECTOR ENTRY, MANY HANDLERS

A compromise between the overhead of a huge interrupt table and the inefficiency of one handler searching every source.

سلسلة المقاطعات: عندما يكون عدد الأجهزة أكبر من عدد المداخل في الجدول، يشير كل مدخل إلى قائمة من برامج الخدمة تُنفَّذ واحدًا تلو الآخر حتى يوجد البرنامج القادر على خدمة الجهاز.

Analogy. One shared extension for a whole department: reception rings it, and colleagues pass the call along until the right person answers.

Intel Processor Event-Vector Table

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

`0 – 31`

Nonmaskable. Error conditions raised by the processor itself — divide error, page fault, double fault, machine check.

`32 – 255`

Maskable. Used for purposes such as device-generated interrupts.

Some slots are reserved by Intel and must not be used.

Interrupt Priority Levels

Levels let the CPU **defer low-priority interrupts** without masking all interrupts.

They also let a high-priority interrupt **preempt** the execution of a low-priority one.

Because interrupts are used so heavily for time-sensitive processing, efficient handling is required for good system performance.

HIGH	Unrecoverable memory error, disk failure
MID	Network packet arrival, disk transfer complete
LOW	Mouse movement, keyboard input

urgent work first — the rest waits its turn

Terminology Recap

interrupt	Hardware signal to the CPU that an event needs attention
ISR	Interrupt service routine — the code that services the device
IRQ line	Wire the CPU senses after executing every instruction
nonmaskable	Line that cannot be disabled; unrecoverable errors
chaining	Vector entry pointing to a list of handlers tried in turn
system bus	Main communications path between the major components
interrupt vector	Array of ISR addresses in low memory, indexed by interrupt number
iret	Return-from-interrupt instruction that restores the prior state
maskable	Line the CPU can switch off during critical sequences
priority level	Rank that allows deferral and preemption of handlers

Section Summary

1. Interrupts are used throughout modern operating systems to handle **asynchronous events**.
2. **Device controllers and hardware faults** are what raise them.
3. A system of **interrupt priorities** lets the most urgent work be done first.
4. Because interrupts carry **time-sensitive** work, efficient handling is required for good performance.

Check Your Understanding

Q1 Why is a table of pointers preferred over one generic routine that inspects each interrupt?

Q2 Which line would a disk controller use to request service, and why not the other one?

Q3 A machine has 300 devices and 256 vector slots. What mechanism resolves this, and at what cost?

Q4 In Figure 1.3, what is the CPU doing between "I/O request" and "interrupt signaled"?