2ème Ingénieur Informatique

Chapter 4

The Interrupt Systems

1

BY
PR. CH. BENCHERIET

Plan

2

1. Introduction

3

- The interrupt mechanism forms the basis of the input/output module in operating systems.
- Interrupting a running program to handle a more urgent task.
- Taking asynchronous events into account.

Objectives

- Detect an unexpected event: alarm, power outage, etc.
- Avoid constant polling: e.g., analogy with a telephone ringing.
- To execute a subroutine called an interrupt service routine.

2. What is an Interrupt?

4

An interrupt is a mechanism that allows the execution of a process to be interrupted following an external or internal event, transferring control to a routine called an "**interrupt routine**" (handling the interrupt).

3. The Different Causes of Interrupts

5

Interrupts can originate from various sources and are classified into three types:

External (Hardware) Interrupts

- These are independent of the process and are triggered by a peripheral device (e.g., keyboard, I/O port, printer, etc.).
- They help manage access conflicts to the processor.

Internal Interrupts (Trap or Exception)

- These occur due to internal processor errors such as overflow, division by zero, page faults, etc.

System Calls

- These happen when a running program in an operating system requests kernel services (e.g., I/O requests, reading or writing files, memory allocation, etc.).
- This is also referred to as an external software interrupt, generated by a program, such as through the assembly instruction INT.

4. Interrupt Recognition

6

Different Physical Methods for Determining the Source of an Interrupt (IT)

Multi-level Interrupts

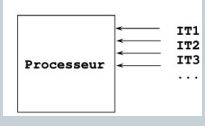
- Each device is connected to a specific interrupt input on the microprocessor.

Advantages

- Technically straightforward solution.

Disadvantages

- Expensive in terms of processor input pins.



4. Interrupt Recognition

7

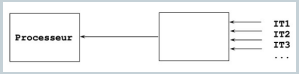
Single-Line Interrupt

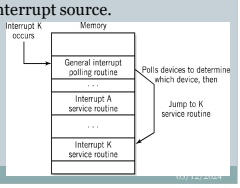
Advantage

- Only one interrupt line is required on the processor.

Disadvantage

- Requires polling of peripherals to identify the interrupt source.





5. Vectorized and Non-Vectorized Interrupts

8

There are two main types of interrupts, based on how the processor handles their processing:

Non-Vectorized Interrupts: In this case, the processor does not have a specific address for each type of interrupt. A single interrupt routine is used (called the general or common routine). This routine identifies the source of the interrupt, often by checking a status register or polling the relevant peripherals.

Vectorized Interrupts: Each type of interrupt has a specific address (vector) that points to its own processing routine. This makes the processing faster, as the processor immediately knows which routine to execute.

5. Vectorized and Non-Vectorized Interrupts

9

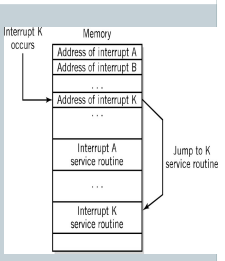
Why isn't everything vectorized?

- In some simple or older systems, non-vectorized interrupts were sufficient and easier to implement.
- Vectorized interrupts require additional mechanisms (such as a vector table), which can be costly or unnecessary for some basic devices.
- Modern systems often favor vectorized interrupts for their speed and efficiency.

5.1 Vectorized Interrupt

10

- A vectorized interrupt, also known as a vectored interrupt, refers to a mechanism in which an interrupt is associated with a specific interrupt vector.
- Each type of interrupt has a unique interrupt number that is used as an index in a table called the interrupt vector table.
- This interrupt vector points to the appropriate interrupt handler.

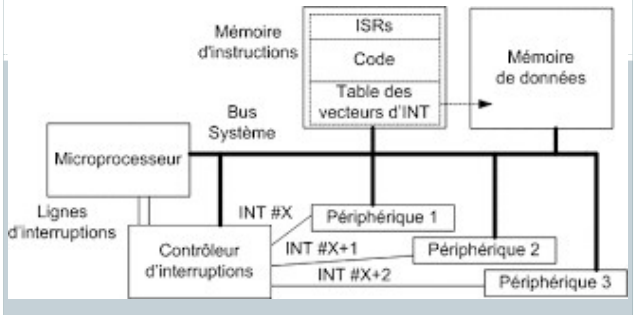


5.1 Vectorized Interrupt

11

The process of handling vectorized interrupts can be described in several steps:

1. **Interrupt Generation:** An event, such as a request from an input/output device, generates an interrupt.
2. **Source Identification:** The interrupt number is associated with the specific source of the interrupt. Each interrupt source has a unique interrupt number.
3. **Accessing the Interrupt Vector Table:** The interrupt number is used as an index to access the interrupt vector table. This table is typically stored in memory and contains memory addresses corresponding to the interrupt handlers.
4. **Control Transfer:** The processor transfers control to the interrupt handler whose address is indicated in the interrupt vector. This leads to the execution of the code specific to handling the interrupt.
5. **Interrupt Handling:** The interrupt handler takes care of processing the event associated with the interrupt. This may include manipulating data, communicating with the device, or other necessary actions.



Remarque

- The use of interrupt vectors simplifies the management of different interrupt sources in a system.
- Each type of interrupt is associated with a specific memory address, making selective interrupt handling easier.
- This mechanism is commonly used in modern hardware architectures to efficiently organize interrupt management.

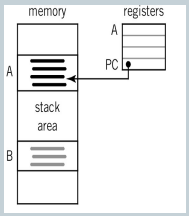
Advantages: The microprocessor immediately recognizes the device that triggered the interrupt.
Disadvantage: It is necessary to manage priorities (simultaneous deposits of 2 vectors on the b

6. Detection and Handling of an Interrupt in a Simple System

1. Interrupt Detection
2. Context Saving
3. Identifying the Cause of the Interrupt
4. Interrupt Acknowledgment
5. Interrupt Handling
6. Restoring the Context of the Interrupted Program

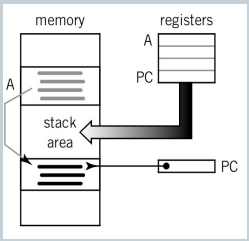
6. Detection and Handling of an Interrupt in a Simple System

1. **Interrupt Detection**: This occurs when the processor identifies that an external or internal event requires attention. It can be triggered by a signal from a hardware device, a special software instruction, etc.
2. **Context Saving**: Store the current state of the processor, including register values, the instruction pointer, and other relevant information. This ensures the interrupted program can resume execution later without loss of information.
3. **Identifying the Cause of the Interrupt**: This involves determining the specific source that triggered the interrupt. It is done by checking specific registers, interrupt tables, or other dedicated mechanisms.



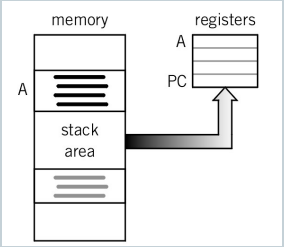
6. Detection and Handling of an Interrupt in a Simple System

4. **Interrupt Acknowledgment**: Inform the hardware that the processor has acknowledged the interrupt and is ready to handle it. This may involve sending a return signal to the interrupt source.
5. **Interrupt Handling**: The system executes the appropriate interrupt handler, which is code designed to manage the specific event that triggered the interrupt.

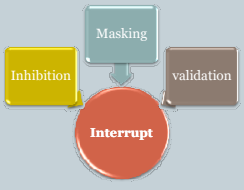


6. Detection and Handling of an Interrupt in a Simple System

6. **Restoring the Context of the Interrupted Program:** Once the interrupt handling is complete, restoring the context involves retrieving the saved state of the processor (step 2) to resume the execution of the interrupted program from where it was paused.



7. Hierarchical Interrupt Systems



7. Hierarchical Interrupt Systems

19

Interrupt Inhibition

- It is the temporary disabling of interrupts.
- When an interrupt is inhibited, the processor temporarily ignores interrupt signals, thus preventing their handling during this period.
- Inhibition is often used to ensure data integrity in critical situations or during the execution of certain routines that are sensitive to interrupts.

7. Hierarchical Interrupt Systems

20

Interrupt Masking

- Occurs when certain interrupts are temporarily disabled or ignored by the system, usually based on their priority level.
- Lower-priority interrupts may be masked during the handling of a higher-priority interrupt.
- This helps manage the interrupt hierarchy by ensuring that the most important interrupts are handled first.

7. Hierarchical Interrupt Systems

21

Interrupt Validation

- Occurs when the system decides that the interrupt is valid and must be processed.
- It may be related to checking certain conditions, such as the status of a device or the presence of a specific event.
- Once validated, the interrupt is typically acknowledged, and its handling can begin.

8. Detection and Handling of an Interrupt in a Hierarchical System

22

- **Interrupt Detection in a Hierarchical System:** Involves the recognition of a disruptive event by the hardware or software. Interrupts are typically associated with priority levels, and the system responds accordingly, either by suspending or modifying the ongoing execution.
- **Interrupt Handling in a Hierarchical System:** Involves the orderly management of interrupts based on their priority level. The interrupt handling routines associated with each level are called sequentially, starting with the highest interrupt level. This ensures a prompt response to the most critical interrupts.

9. Interrupt Level Encoding

23

- The encoding of interrupt levels depends on the specific architecture of the processor.
- Each processor manufacturer may have its own way of representing interrupt levels in hardware.

Example: For x86, interrupt levels are often encoded using the status register.

- In the status register, the interrupt flag (IF) controls whether hardware interrupts (maskable) are enabled or disabled.
- When the interrupt flag is set to 1, interrupts are enabled, and when the bit is set to 0, interrupts are disabled.
- The lowest interrupt level (the highest priority level) is generally the highest-priority hardware interrupt level.

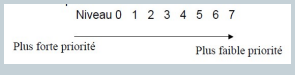
Example 1: Interrupt Hierarchy

24

The processor has a single INT (interrupt) pin to receive interrupts.
What happens if:

- ✓ Two interrupts arrive at the same time?
- ✓ An interrupt occurs while another interrupt is being handled?

Note: A priority (level) concept is introduced between interrupts.
Example: 8 levels.



Example 1: Interrupt Hierarchy

25

What to do if:

Q. Two interrupts arrive at the same time?

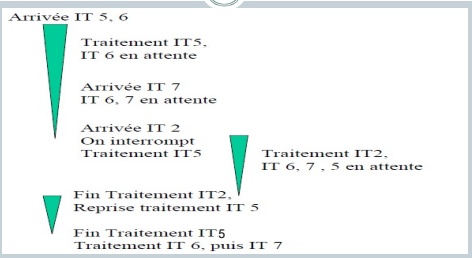
A. The one with the highest priority is handled first.

Q. An interrupt occurs while another interrupt is being handled?

A. It interrupts the current processing only if it has a higher priority.

Example 2: Interrupt Hierarchy

26



Example 2: Interrupt Hierarchy

27