Computer Architecture

Keivan Navi

Cal Poly Pomona University

knavi@cpp.edu

Office hours: Tu/Th 5:25 Pm to 6:55 Pm

Office: 8-49

Computer Architecture

Interrupt

Polling

Interrupt/ Polling

- Interrupt and polling are two different methods used in computer systems to help devices communicate and synchronize.
- ***Polling:*** A method where the CPU continuously checks the status of all the devices to ensure the system is operating efficiently. Polling can be done at regular intervals or in a tight loop that uses a huge percentage of the processor's bandwidth. Polling can be resource-intensive and may cause a delay between when new data is available and when the client retrieves it. The advantage of Polling is that the CPU is always informed about the status of the devices and in the case of fault or failure it can react. Polling is like picking up your phone every few seconds to see if you have a call.

Interrupt

- ***Interrup***t: A method where devices signal when an event needs immediate attention. When an interrupt occurs, the CPU usually stops its current task and starts executing the interrupt handler. After completing the interrupt handler, the CPU resumes the previous task. Context switching can cause interrupt overhead. This is when the context of the current process and its thread must be saved when entering the interrupt handler and restored when exiting. Cache usage is another negative effect of context switching. Polling can be used in real-time systems, but it might not be ideal due to its inherent latency. Interrupts are often preferred for their immediate responsiveness in real-time applications.
- There are 3 major kind of interrupts:
- 1-Software Interrupts
- 2-Non-maskable Interrupts
- 3-Maskable Interrupts

Software Interrupt

- ***Software Interrupt***: A software interrupt is a signal from a program to a processor that an event needs immediate attention. It allows a program to stop executing and transfer control to an interrupt service routine (ISR). Software interrupts are used for a variety of tasks, including:
- **Reading input**: Reading input from a keyboard
- **Displaying text**: Displaying text on a screen
- **Accessing date and time**: Accessing the current date and time
- **Requesting OS services**: Requesting services from the operating system, such as opening a file or printing a message
- **Debugging**: Pausing a program at a specific point for debugging purposes
- Software interrupts are triggered when a program executes a special instruction, or when certain conditions are met. For example, a system call is a software interrupt that requests a service from the operating system. An exception is a software interrupt that occurs when the microprocessor encounters an error, such as division by zero.
- Software interrupts are essential for the operating system to respond to external devices. They allow programs to communicate directly with the main part of the operating system, which leads to more efficient software design and better resource management.

Non-Maskable Interrupt

- ***Non-Maskable Interrupt***: In computing, a non-maskable interrupt (NMI) is **a hardware interrupt that standard interrupt-masking techniques in the system cannot ignore**. It typically occurs to signal attention for non-recoverable hardware errors. Some NMIs may be masked, but only by using proprietary methods specific to the particular NMI. **A special NMI interrupt** has the highest priority and cannot be masked out by other interrupts. It is processed during critical hardware events such as a power loss.
- One example of where an NMI is useful is in an embedded system. Consider the possibility of failure where an electrostatic discharge sent the processor off into the weeds, where it could be stuck in a loop.

Maskable Interrupts

- A maskable interrupt is a type of interrupt that can be enabled or disabled by a programmer. The CPU can choose to ignore or enable a maskable interrupt.
- **The way they are triggered**
- Maskable interrupts can be triggered in two ways: level-triggering or edge-triggering
- **How they are used**
- All Interrupt Requests (IRQs) issued by I/O devices are maskable interrupts.
- **How they are controlled**
- Masking and unmasking interrupts can help optimize CPU utilization and power consumption.
- **How they are different from non-maskable interrupts**
- Non-maskable interrupts (NMIs) have a higher priority and cannot be disabled. NMIs are typically used for critical events like power loss.

What will happens if we need a lot of interrupts pins?

- Imagine we have 256 different devices that need the CPU to service them if it is needed. The time of service is not defined. That is we need interrupt signals to handle it. But it seems using 256 Pins of the CPU in this regard is a too much. Isn’t there any other way to reduce this huge amount of input signal to the CPU?
- There are to way to solve this problem:
- Priority Encoder
- Interrupt Handler

Priority Encoder

- Encoder:
- An Encoder is a combinational circuit that performs the reverse operation of a Decoder. It has a maximum of 2^n (some times it is less than 2^n) input lines and 'n' output lines, hence it encodes the information from 2^n inputs into an n-bit code. It will produce a binary code equivalent to the input, which is active High.
- For example instead of having
- 8= 2^3 the number of wires will be reduce to 3
- Instead of 1024= 2^{10} will have 10 wires.

Priority Encoder

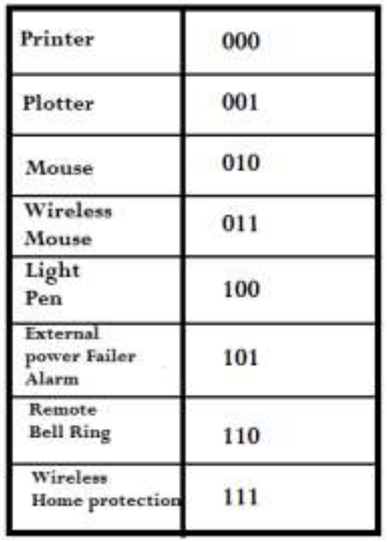
- A priority encoder is a circuit or algorithm that compresses multiple binary inputs into a smaller number of outputs, similar to a simple encoder. The output of a priority encoder is the binary representation of the index of the most significant activated line.
- That is between the data or signals to be serviced there is a serious hierarchy. In the case of two different devices asking the service at the same time, there will be no conflict.

Interrupt Handler

- **Handles simultaneous interrupts**
  - Receives interrupts while the CPU
  - handles interrupts and Maintains interrupt flags (CPU can poll interrupt flags instead and find out the status of the system.
- **Reduces the number of Wires and Pins**
  - CPU doesn’t need an interrupt wire to each particular peripheral. Instead the priority imposed by the CPU will be considered by the Interrupt Handler to manage them.

How can the CPU understand which device is asking service? (Vectored Interrupt)

- Imagine there is only one interrupt pin in the CPU which is connected to a priority encoder or interrupt handler. How can the CPU find out who is asking the service?
- After activating the interrupt signal, each device must introduce itself, in order to prevent any kind of conflicts.
- Imagine that we have only 8 devices.
- The needed codes must be from 0 to 7(000 to 111).

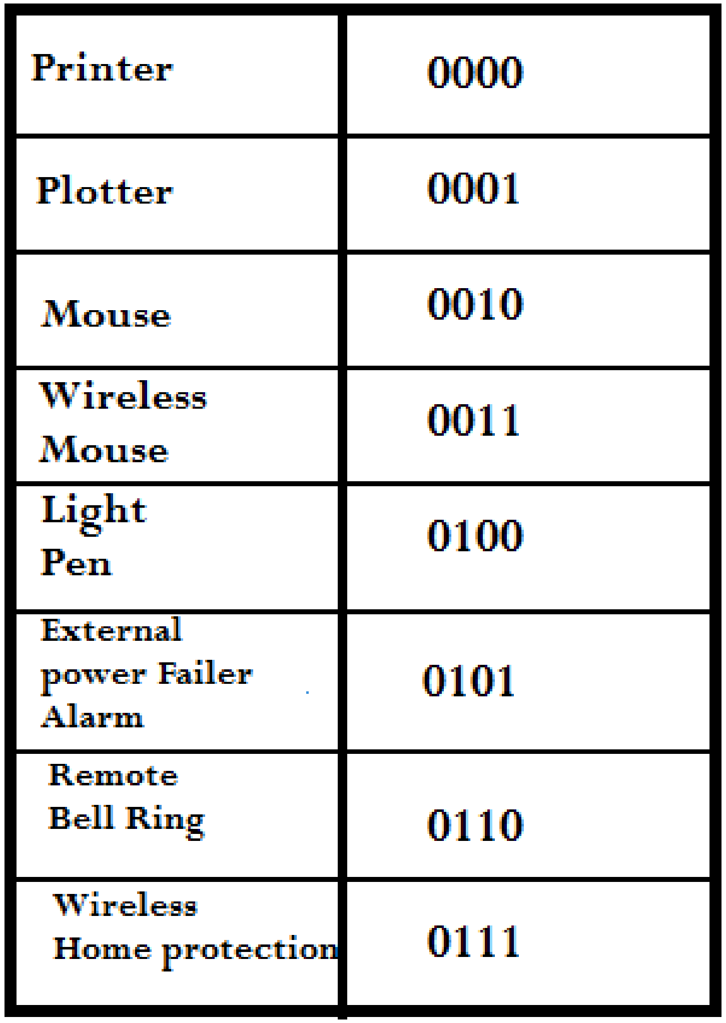


Printer	000
Plotter	001
Mouse	010
Wireless Mouse	011
Light Pen	100
External power Failer Alarm	101
Remote Bell Ring	110
Wireless Home protection	111

- The CPU or the operating system will define a code for each device. The following table is a good example:

Another difference between Priority Encoder and Interrupt Handler

- Consider this table of interrupt services:
- After the activation of the interrupt signal a code of 1000, it is considered as out of the range code.
- The interrupt handler will send an out of the range signal to the CPU.
- The priority Encoder will need additional circuits to send an out the range signal to the CPU.

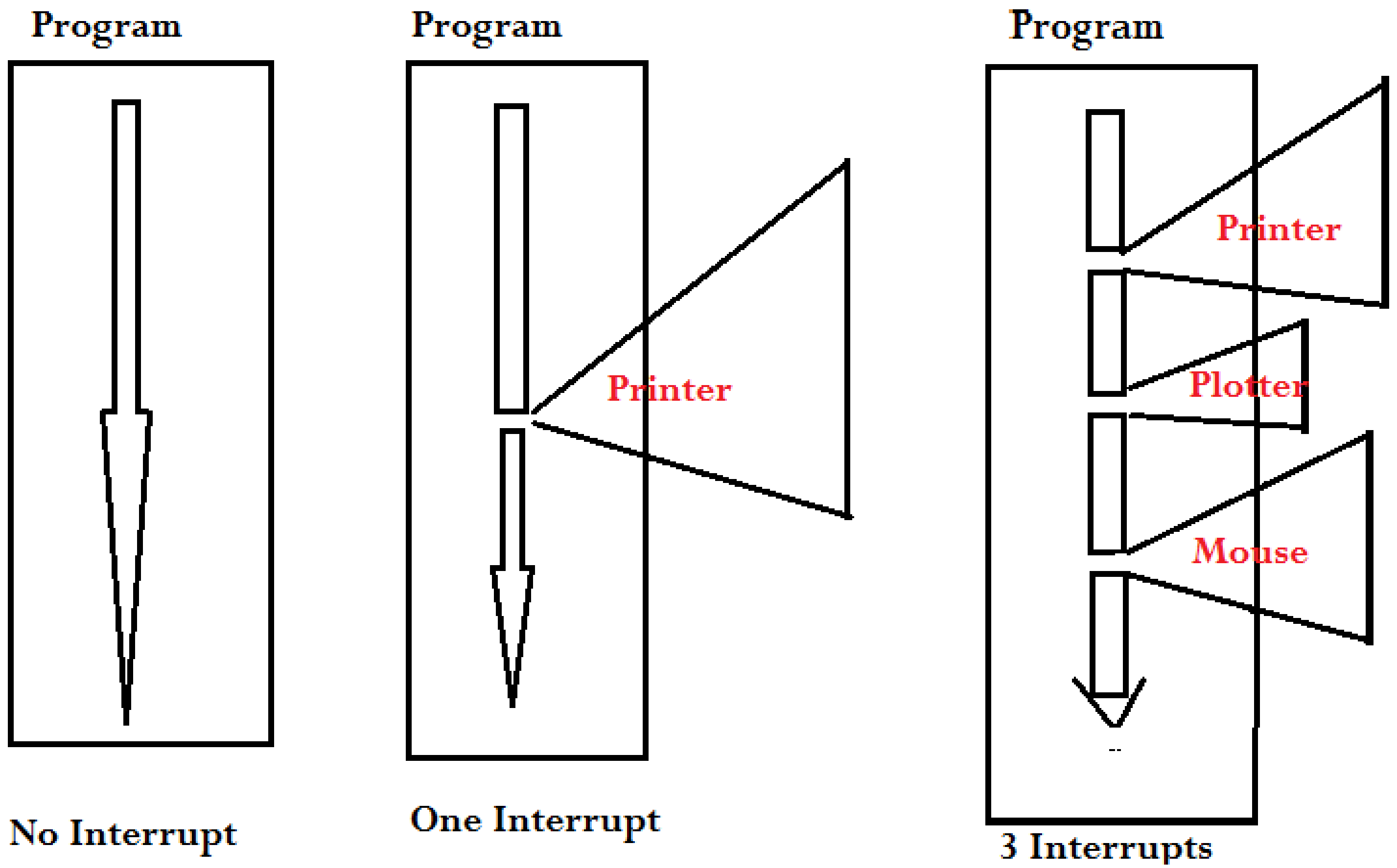


Printer	0000
Plotter	0001
Mouse	0010
Wireless Mouse	0011
Light Pen	0100
External power Failer Alarm	0101
Remote Bell Ring	0110
Wireless Home protection	0111

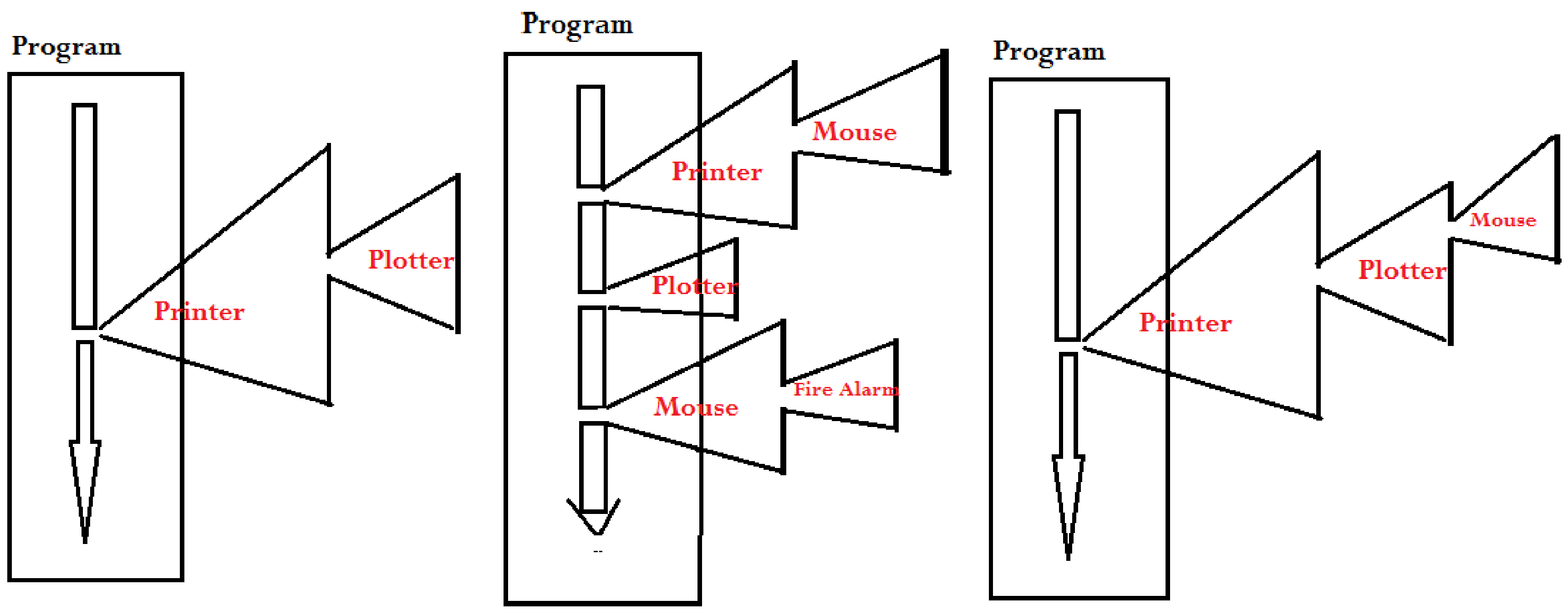
Nested Interrupt Handeling

- ***Nested interrupt handling*** is a software technique that allows a processor to prepare to accept another interrupt before it finishes handling the current one. This enables the prioritization of interrupts and improves the latency of high priority events. However, it also increases the complexity of the system.
- An interrupt is a signal to the processor that indicates an event that needs immediate attention. Interrupts can be generated by hardware or software. Peripheral devices, such as sensors, timers, communication modules, or keyboards, can generate hardware interrupts.

Example of Interrupts



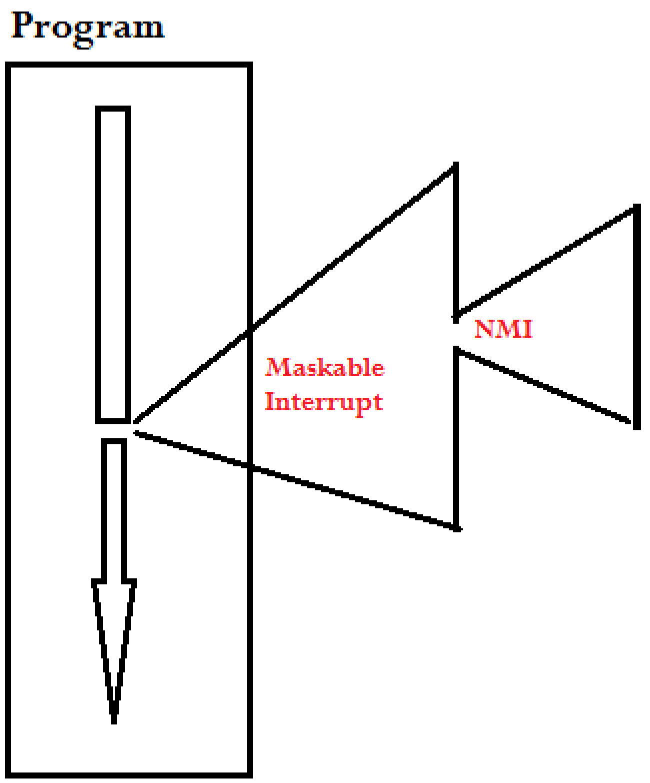
Example of Nested Interrupts

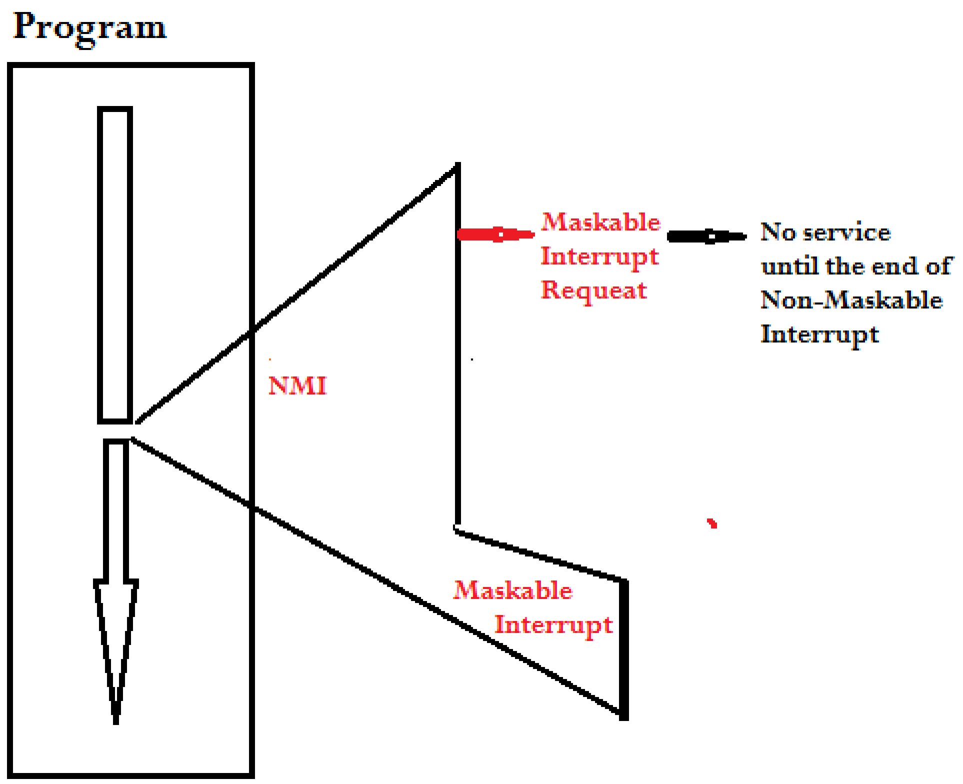


- In these examples the CPU permits Nested Interrupts without any restriction.

Non-Maskable (NMI) interrupt versus Maskable Interrupt

- NMI is serviced when running Maskable interrupts
- Maskable interrupts won't be serviced until the end of NMI




Program
NMI
Maskable Interrupt Requeat
No service until the end of Non-Maskable Interrupt
Maskable Interrupt