Interrupt

What are Interrupts?

Interrupts are a response by the processor to a process/event that needs immediate attention from the software. It alerts the processor and servers a request for the CPU to interrupt the currently executing program/code when permitted, in order so that the event can be processed within good time. If the response is accepted from the processor, the processor will respond by suspending its current activities (saving its state), and thus executing a function called an interrupt handler to deal with the event.

This interrupt in activities and programs is only temporary, that is, unless the interrupt indicates a fatal error, otherwise the processor will resume all normal running actives after the interrupt handler finishes.

Interrupts are used by both hardware and software to indicate electric or physical state changes that require attention. Interrupts have a heavy usage in computer multi-tasking, especially in real-time computing, moreover, these systems that utilise interrupts are said to be “interrupt-driven”.

One of the advantages of interrupts is that they can be used to break an infinite loop, which can create memory leaks or cause a program to be unresponsive.

What is an Interrupt handler?

Interrupt handlers, also commonly known as Interrupt service routine (ISR), is a block of code that is associated with a specific interrupt condition.

Hardware and software interrupts or software exceptions initiate the interrupt handler, furthermore the interrupt handler is used to implement device drivers or transitions between protected modes of operation, such as system calls.

Interrupts have a number of functions, which ultimately depend on what triggered the interrupt in the first place, therefore the speed of which an interrupt handler completes its tasks also varies depending on what triggered the interrupt.

For instance, moving the mouse or pressing a key on the keyboard, will trigger interrupt handlers which read they key, or the position of the mouse, lastly the interrupt handlers will copy the information associated with those actions into the computers memory.

Types of Interrupts

Interrupts signals can be classified as software interrupts or hardware interrupts.

For any processor, the total number of interrupt types is limited by the architecture.

Hardware Interrupts:

All the devices are connected to the Interrupt Request Line (IRQ) or detected by devices embedded in processor logic (i.e. the CPU timer), to communicate that the certain device requires attention from the operating system, and if there is no operating system, then seek attention from “bare-metal” program which is running on the CPU.

Moving the mouse is considered to be using hardware interrupts.

Interrupt Request Line (IRQ): An IRQ refers to a hardware signal sent to the processor that temporarily stops a running program and allows the interrupt handler to run instead.

With regards to the processor clock, hardware interrupts can arrive asynchronously (transmission of data without the use of an external clock signal) with respect to the processor clock, and at any time during instruction execution.

That being said, all the hardware interrupt signals are conditioned by synchronising them to the processor clock, and acted upon only at instruction execution boundaries.

In many systems, the device that is causing the particular interrupt request can be identified, as each device is associated with a specific IRQ Three aspects relating to process of interrupt that will determining overall system performance of the process in handling interrupts.

* Interrupt latency: The amount of time that passes between the time that an interrupt request is received by the processor and the time that the processor takes action to begin processing the interrupt service routine. This delay is called interrupt latency.
* Interrupt processing time. This is the amount of time that the processor spends actually saving the machine state of the interrupted job and getting to service the interrupt. This is normally a small amount of time
* The amount of time taken to save machine registers which must be saved in order for the interrupt service routine to do its job. This time varies depending upon the complexity of the interrupt service routine.
* The costs of restoring all the machine state and returning to the interrupted routine.

with pipelining, there is an added complexity; as the interrupt is received, there could be a number of instructions still in the pipeline. The usual way to deal with this is to discard all instructions in the pipeline except for the last instruction in the write-back (WB) stage.

Interrupt in the different processor architecture

RISC

RISC processors are designed to execute a small set of simple instructions with high efficiency and have a simpler and more streamlined interrupt handling mechanism. It services both the hardware and software interrupt

RISC processors make use of the interrupt vector table. This is similar to a lookup table that holds addresses of the Interrupt Service Routines (ISRs). When an interrupt occurs, the processor retrieves a unique identifier associated with the interrupt type. This identifier is used as an index from the vector table to fetch the corresponding address from the IRS.

Before transferring control to the ISR, the processor takes steps to preserve the current execution state. This involves saving crucial information like the program counter, register values, and any other relevant context on a designated stack. This ensures that once the ISR finishes executing, the processor can restore the previous state and seamlessly resume normal program flow.

RISC processors also typically offer a mechanism to disable or enable interrupts. If certain sections of code must be executed without interruption the processor will temporarily disabling the interrupt, there is control over when and where interruptions can occur.

RISC processor can implement a priority system for handling interrupts. This allows the processor to address higher-priority interrupts before lower-priority ones, ensuring critical events receive immediate attention.

In a CISC (Complex Instruction Set Computer) architecture, interrupt handling operates through a slightly different mechanism compared to RISC architectures. CISC processors, known for their extensive and varied instruction sets, handle interrupts in a manner that may involve microcode routines and specialized mechanisms.

Unlike RISC architectures, CISC processors uses either microcode or firmware routines stored in ROM (Read-Only Memory) for managing interrupts. These routines are pre-defined sequences of instructions that are executed when specific interrupt conditions are met.

CISC instruction encoding is often variable-length, which can add complexity to handling complex operations like interrupt handling. Some instructions might implicitly trigger interrupt routines, while others may require explicit instructions or specific flags to initiate the interrupt handling process.

Similar to RISC architectures, CISC processors also require a way to preserve the current execution state before transferring control to the ISR. Such as putting programs such as program counter, register values ect on a designated stack. This ensures that once the ISR completes its execution, the processor can restore the previous state and seamlessly continue with the main program.

What is a microcode

Microcode is a low-level, hardware-level programming language (**microcode language or microcode assembly language**) that is used internally by a processor to execute complex instructions. In a CISC (Complex Instruction Set Computer) processor, microcode plays an important role in translating complex instructions into a series of simpler, more fundamental operations that the processor's control unit can execute.

CISC processors have a large and diverse set of complex instructions that can perform a wide range of operations. These instructions may involve multiple steps and operations. Microcode acts as an intermediary layer between the complex instructions and the actual hardware components of the processor.