Hasan Hashim

10/26/2020

# Week 8 Readings

**7.4.1 I/o Control Methods:**

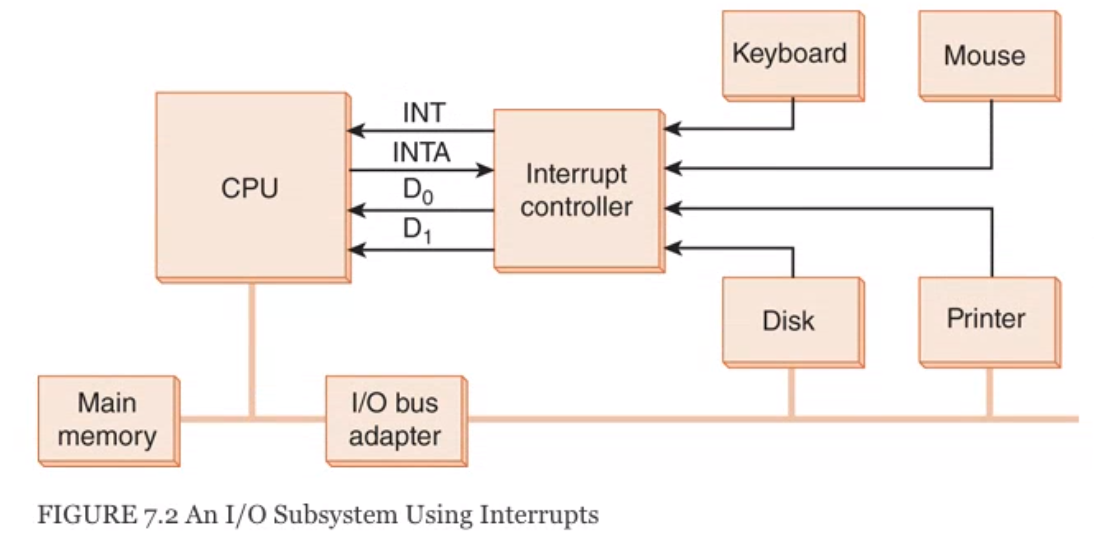
Dedicated I/O modules serve as interfaces between the CPU and its peripherals. They can control device actions, buffering data, performing error detection, and communicating with the CPU. One method isn't necessarily better than another, but the manner in which a computer controls its I-O greatly influences overall system design and performance.

**Programmed I/O:**

The way to connect with a CPU is through programmed I/O and this method is also called polled I/O or port I/O. The CPU continuously tracks (polls) the control register connected with each I/O port. Every bit in the control register will be ready for the byte to arrive in the port. When the processing is all done, the CPU will continue the polling of the control registers as the first time.

The advantage of using this approach is that we have programmatic control over the actions of each system. By changing a few lines of code, we can change the number and form of devices in the scheme, as well as their polling priorities and intervals. Constant polling is, however, a concern. The CPU is in a constant "busy wait" loop before an I/O request is initiated. It doesn't do any valuable work until I/O has been analyzed.

**Interrupt-Driven I/O:**

Interrupt-driven I/O is a more common and powerful control system. Interrupt-driven I/O can be thought of as a programmed I/O converse. Instead of the CPU constantly telling its connected devices if they have any input, the devices would inform the CPU when they have the data to send. The CPU can continue to execute other functions until the computer requesting the service sends an interruption to the CPU. Usually, these interrupts are created for any word of information that is transmitted. The setup of an example is seen in Figure 7.2. Any computer on the machine has access to the interrupt request line. The interrupt control chip has an input for each line of interrupts. When an interrupt line is invoked, the controller decodes the interrupt and lifts the Interrupt (INT) input to the CPU. 

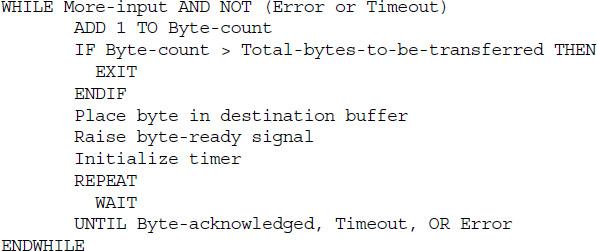
System programmers decide which devices can take priority over the others when more than one device causes disruption at the same time. The interrupt goals are then hardwired into the I/O controller, making them almost impossible to modify. Any machine that uses the same operating system and interrupt controller adds low-priority devices (such as the keyboard) to the same interrupt request line.

**Memory-Mapped I/O:**

The architectural decisions taken with respect to the system's I/O control mechanism have an important impact on the overall system architecture. If we plan to use programmed I/O, it will be to our benefit to set up separate busses for memory traffic and I/O traffic so that continuous polling does not interfere with memory access.

In every system it will be different, like in the small systems the low\_level data will be transferred under I/O controllers that are already built in the devices.This way will work fine because the CPU will not care about weather the device is ready or counting the bytes in a transfer, or calculating error-correcting codes.

**Direct Memory Access:**

As we knotece in the programmed I/O and interrupt-driven I/O, the CPU moves data from and to the I/O device in the system. In the I/O, the CPU runs a code instructions similar to the one in the bottom pseudocode:

This is the concept behind Direct Memory Access (DMA). If a machine uses DMA, the CPU unloads time-consuming I/O instructions. To impact a switch, the CPU will supply the DMA controller with the location of the bytes to be transferred, the number of bytes to be transferred, and the destination computer or memory address. This exchange normally takes place via the CPU 's unique I/O registers. Once the right values are stored in the memory, the CPU alerts the DMA subsystem and continues with the next step, while the DMA takes care of the specifics of the I / O. After the I / O is finished (or ends in error), the DMA subsystem signals the CPU by sending it another interrupt.

**Channel I/O:**

As we know that programmed I/O handles a one byte in a time of course that will depend on the device and a system. Slower machines, such as keyboards, produce more interrupts per amount of bytes than disks or printers. DMA methods are all block-oriented, interrupting the CPU only after the transition of a set of bytes has been accomplished (or failed). After the DMA signals the completion of the I / O, the CPU can send it the address of the next memory block to be read from or written to. In the case of loss or failure, the CPU shall be entirely responsible for taking the necessary action. With channel I/O or more I/O processors monitor separate I/O pathways which are called channel paths. The channel paths are used for the slow devices for example terminals and printers. I/O channels are powered by small CPUs called I/O processors (IOPs) that are designed for I/O service. Unlike DMA circuits, IOPs are capable of running programs that provide arithmetic-logic and branching instructions.

**7.4.2 Character I/O Versus Block I/O:**

When we press a key in the a keyboard the motion of a sequence of actives will be a single event and also no matter how fast you type it will be a single event. Each key controls a small switch that closes the connection in a conductor matrix that runs horizontally and vertically below the keys. When a key click closes, the keyboard circuitry reads a different scan message. The scan code is then sent to the serial interface circuit, which converts the scan code into a character code. In a keyboard buffer that is stored in low memory, the gui positions the character code. In the queue, the characters wait quietly before they are recovered by a program (or until the queue is reset) one at a time. In blocks, magnetic disks and tapes store data. Consequently, handling disk and tape I / O in block units makes sense. Block I / O lends itself to processing DMA or channel I / O. Blocks, depending on the individual hardware , software, and implementations involved, may be various sizes. When a device is being calibrated for optimal efficiency, deciding an appropriate block size may be a significant task. High-performance systems manage big blocks more easily than small blocks are handled. Even the slow systems should be okay to manage bytes in the smaller blocks, otherwise, this system might become unresponsive to the user input during the I/O.

**Difference between Serial and Parallel Transmission:**

**Serial Transmission**: The data-bit moves bi-directionally from one machine to another machine. One bit flows at one pulse of the clock in this transmission. In Serial Transmission, at a time that has a start and stop bit, 8 bits are passed.

**Parallel Transmission**: From one computer to another computer, several bits move together concurrently. To relay the bits, parallel transmission is faster than serial transmission. For short distances, parallel transmission is used.

**Guiding Questions:**

* Pick an I/O device (keyboard, scanner, USB thumb drive, etc.). Describe what I/O control method would be best for the device, and whether it should use character or block I/O.

The best I/O control method for Keyboard is Interrupt-Driven I/O. Because it controls the input and the output activity and it also sends a signal an example of that is a keyboard. The Processor is continuously operating on its assigned tasks here. When an input is available, such as when someone types a key on the keyboard, to take care of the input data, the CPU is distracted from its task. Without inspecting the input devices, the CPU will operate constantly on a job, allowing the devices themselves to interrupt it if necessary.

* Based on the "Difference between Serial and Parallel Transmission" table in the web link, give an example of an I/O scenario where serial transmission would be better than parallel transmission. Justify your answer.

An example I can think about is a printer. It will be faster using Parallel Transmission than Serial Transmission using it because it is a short distance. As we know that Parallel Transmission is used for a shot distance. Also it is not cast efficient.