Motivation for IO handling Lab

Course : ET032G, Elektroteknik GR (A), Mikrodatorteknik at Mittuniversitetet, Sweden.

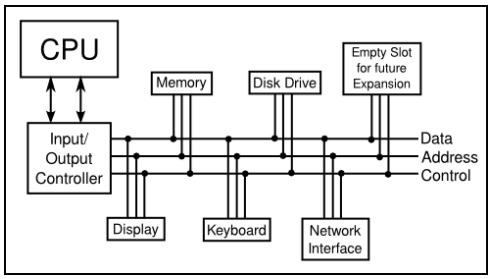
Extracted by : Muhammad Imran, Mid Sweden University, Sweden.

The CPU spends most of the time on fetching instruction from memory and executing them. The embedded platforms are not only combination of memory and CPU as there are other components such as

- Keyboard
- Mouse
- Printer
- Audio
- Temperature sensors
- Humidity sensors

All these devices are connected to the CPU via one or more buses which carries all sorts of information between the devices and the CPU. These wires carries address, data and control signals.

But how does the CPU know that the data is there?



CPU polling

One simple idea, which turns out to be not very satisfactory, is for the CPU to keep checking for incoming data over and over. Whenever it finds data, it processes it. This method is called **polling**, since the CPU polls the input devices continually to see whether they have any input data to report. Unfortunately, although polling is very simple, it is also very inefficient. The CPU can waste an awful lot of time just waiting for input.

To poll an I/O device, the CPU typically uses memory-mapped I/O. I/O devices are assigned memory addresses which aren't used by memory at all, but are instead, special addresses that the I/O device recognizes. (Memory mapped I/O is usually done when the CPU is started up). The CPU can then check the status of the device by reading a byte or word at some pre-determined address for that particular I/O device.

Problems with Polling

The problem with polling is that the CPU operates at a much faster speed than most I/O devices. Thus, a CPU can get into a busy wait, checking the device many times, even though the device is very slow.

Occasionally, polling is the right thing to do, especially if devices are quick enough. The main advantage of polling is that the CPU determines how often it needs to poll. An interrupt causes the CPU to "stop" and determine what device is interrupting.

Thus, if the CPU can be late handling the device, then polling prevents the CPU from being interrupted [1].

To avoid this inefficiency, **interrupts** are generally used instead of polling. An interrupt is a signal sent by another device to the CPU. The CPU responds to an interrupt signal by putting aside whatever it is doing in order to respond to the interrupt. Once it has handled the interrupt, it returns to what it was doing before the interrupt occurred.

For example, when you press a key on your computer keyboard, a keyboard interrupt is sent to the CPU. The CPU responds to this signal by interrupting what it is doing, reading the key that you pressed, processing it, and then returning to the task it was performing before you pressed the key[2].

Types of Interrupts

Hardware interrupt/External interrupt – Generated by an I/O device
Software interrupt/ Internal interrupt– Exception within a program
Program Generated – Used to transfer
control to the operating system

External Interrupts

- Most computers have a timer which interrupts the CPU every so many milliseconds.
- When you press keys or move a mouse, that generates that the interrupt and force the CPU to read the keystroke or mouse position.
- I/O devices tell the CPU that an I/O request has completed by sending an interrupt signal to the processor.
- I/O errors may also generate an interrupt.

Internal Interrupts

- When the hardware detects that the program is doing something wrong, it will usually generate an interrupt usually generate an interrupt. For example divide by zero
  - Arithmetic error - Invalid Instruction
  - Addressing error - Hardware malfunction
  - Page fault - Debugging
- A Page Fault interrupt is not the result of a program error, but it does require the operating system to get control.
- Internal interrupts are sometimes called exceptions.

Interrupt Service Routines

- When an interrupt occurs, execution starts in an interrupt service routine (ISR) or in an interrupt service routine (ISR) or interrupt handler.
- The ISR is almost always in the OS.
- The interrupt service routine processes the event or queues a program to process the event.
- After an external interrupt, the service routine will return to the program.

For ISR to be enabled, you need the following three conditions

- Set relevant interrupt control register for the peripherals you are using
- Enable the global interrupt flag
- The condition for the interrupt (for example pressing a touch button).
- References

[1]. http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/IO/polling.html
[2]. http://math.hws.edu/javanotes/c1/s2.html
[3]. http://williams.comp.ncat.edu/comp375/Interrupts.pdf