# Input/Output

Lecture Notes – Week 7

*G.  MacKenzie, I. Bojanova, Z. Xu*

**Introduction**

This session concentrates on the following:

* Programmed I/O
* Interrupts
* Direct Memory Access
* Device controllers

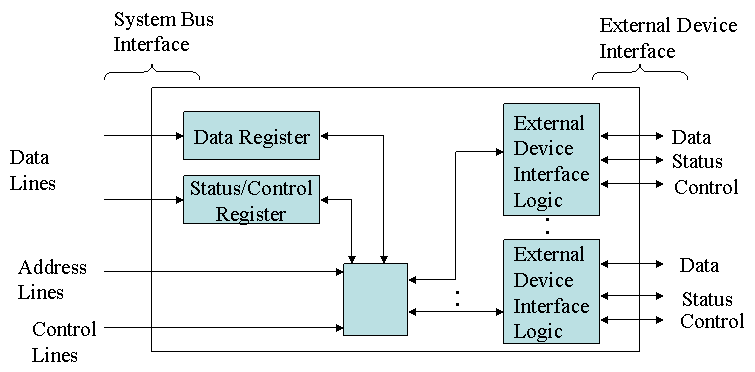
**Overview**

External devices are not generally connected directly into the bus structure of the computer – refer also Figure 7.1. Since there is a wide variety of peripheral devices that require different logic interfaces.

|  |
| --- |
| http://polaris.umuc.edu/~ibojanov/CSMN%20611/Session8_files/FIG.07.04.jpg |

***Figure 7.1 - High-level view of a system bus***

Because of a likely mismatch of data types and different data representations, it is impractical to expect the CPU to “know how” to control each device. The I/O module relieves the CPU of I/O management by providing a standard interface to the CPU and the bus that is tailored to a specific I/O device and its interface requirements. This interface typically consists of control, status, and data signals as shown in Figure 7.2.



***Figure 7.2 - I/O Module Block Diagram***

The I/O module is located between the *system bus* and *peripheral devices* and is used for:

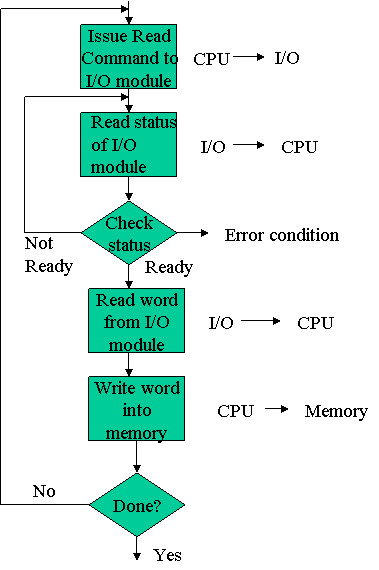
* Control and Timing
* CPU communication
* Command decoding Data: data are exchanged between the processor and I/O module
* Status reporting
* Address recognition
* Device Communication
* Commands Status information Data
* Data Buffering
* Error Detection

**Programmed I/O**

Programmed I/O is an I/O operation in which CPU issues an I/O command to the I/O module, and the CPU is in direct control of the operation. But the CPU must wait until the I/O operation is completed before it can perform other tasks. I/O task completion is indicated by a change in the module status bits, and the CPU must periodically poll the module to check its status.

1. The CPU issues an I/O command to an I/O module and switches to some other useful work.
2. The I/O module will interrupt the CPU to request service when it is ready to exchange data with the CPU. Note that the steps in the interrupt processing are handled by both *software* and *hardware*.
3. The CPU executes the data transfer.
4. The CPU then resumes its former processing.

As a result of the speed difference between a CPU and the peripheral devices (orders of magnitude), programmed I/O wastes a large amount of CPU processing power, since the CPU slowed to the speed of the peripheral. There are advantages, however, in that programmed I/O is simple to implement and requires very little special software or hardware. Figure 7.3 illustrates programmed I/O operation.



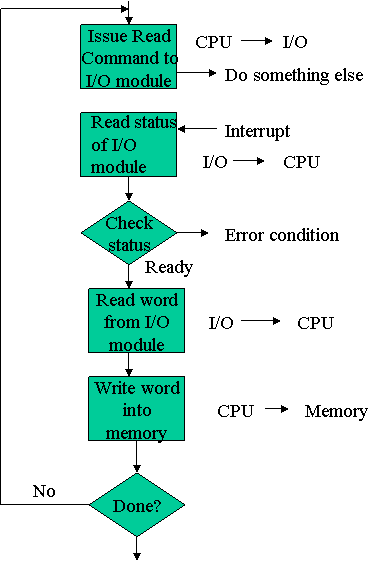
***Figure 7.3 - Programmed I/O Operation***

There are two primary alternatives for I/O addressing: isolated and memory mapped. Characteristics are described below:

* Isolated (standard) I/O
* Address space of the I/O module is isolated from memory address space
* Separate instructions are used to perform I/O
* Typical control lines include read/write lines to specify direction of data transfer and IO/M lines to switch address reference between memory and I/O
* Memory mapped I/O
* I/O devices are integrated into the normal memory address space
* All of the memory accessing instructions can be used to access the I/O peripherals
* “Cost” is the loss of “real” memory addresses (Not a big problem today with the huge address spaces in modern systems)

**Interrupts**

To reduce the time spent on I/O operations, the CPU can use an interrupt-driven approach where it issues an I/O command to the I/O module and continues with its other tasks while the module is performing its task. The module signals the CPU when the I/O operation is finished, and the CPU responds to the interrupt by executing an interrupt service routine and then continues on with its primary task. The CPU recognizes and responds to interrupts at the end of an instruction cycle. The Interrupt-Driven I/O technique, illustrated in Figure 7.4, is used to support a wide variety of devices.



***Figure 7.4 - Interrupt-Driven I/O Operation***

With interrupt-driven I/O, there are several design issues that arise if there are multiple I/O modules. How does the CPU determine which device caused the interrupt? Or, if multiple interrupts occur at the same time, which is processed first? One solution is to provide multiple interrupt signal lines for a system, but this is practical only for small numbers of interrupts. Another approach is to use multiple interrupts but use one interrupt for more than one device. This requires some sort of device polling to determine which requested service, or the requesting device can place an ID on the bus. Finally, there should be a determination scheme to prioritize among multiple interrupts. (For example, processing a keyboard input would probably carry higher priority than servicing a printer output request).

**Direct Memory Access**

Both programmed and interrupt-driven I/O require the continued involvement of the CPU in ongoing I/O operations. Direct memory accessing (DMA), (refer to Figure 7.5,) takes the CPU out of the task except for initialization of the operation. Large amounts of data can be transferred between memory and the peripheral without severely impacting the CPU performance.

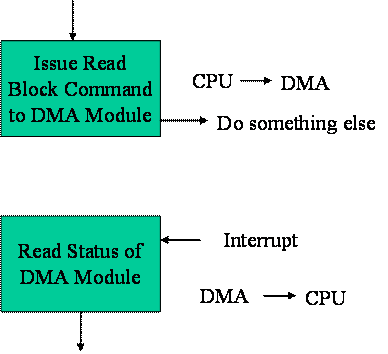
|  |
| --- |
| http://polaris.umuc.edu/~ibojanov/CSMN%20611/Session8_files/FIG.07.02.jpg |

***Figure 7.5 – An example of a DMA configuration***

1. The CPU issues a command to the DMA module.
2. The CPU continues with other work.
3. The DMA module transfers the entire block of data, one word at a time, directly to or from memory, without going through the CPU.
4. When the transfer is complete, the DMA module sends an interrupt signal to the CPU.

The process, as illustrated in Figure 7.6, involves the following steps:

* CPU initializes DMA module
* Read or write operation defined
* I/O device involved
* Starting address of memory block
* Number of words to be transferred
* CPU then continues with other work



***Figure 7.6 - DMA Process***

DMA essentially operates by “stealing” bus cycles from the CPU. It accesses memory to retrieve a data word and forwards the word to the I/O peripheral. In practice, it uses the bus when the CPU is not using it, so there is no impact on the CPU performance.

Please see also:

* [DMA Definition](http://whatis.techtarget.com/definition/0,,sid9_gci213903,00.html)
* <http://www.science.unitn.it/~fiorella/guidelinux/tlk/node87.html>

**I/O Controllers**

I/O controllers sit between CPU and I/O devices.

Location-wise, it can be part of the main system, independent, or part of the I/O devices. Also, one controller can be used to control multiple similar devices. It is the CPU helper that executes commands from CPU, reports status of I/O devices, and controls I/O devices to free up CPU.

References

Englander, Irv (2003). *The Architecture of Computer Hardware, Systems Software & Networking: An Information Technology Approach.* John Wiley and Sons.