Chapter 0
Input and Output

Without a way to get information in and results out, the CPU and memory are useless. When a program developed needs an input or output operation, almost always the mechanism is to use a function that's a part of the programming language. Dig a little further and you will find that the programming language calls on the operating system to perform the operation.

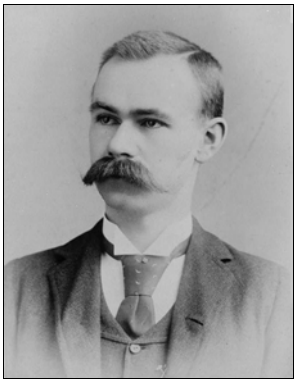

Figure 0-1
Herman Hollerith invented punched card tabulating equipment; punched cards were used for input and output for over a hundred years. Hollerith's company, The Computing-Tabulating-Recording Company, became IBM.

Although application program developers rarely do input or output at the machine level, those who develop operating systems do. An understanding of the process will help the application program developer write better and more efficient programs. In short, we will study what CPU instructions make input and output happen and how they work, something that is glossed over in LMC and, to an extent, in TBC.

The input and output devices are sometimes called **peripherals** because they might be considered to form an outer circle around the computer and memory. Input and output is frequently shortened to I-O.

Networks

From the viewpoint of the CPU, the operating system, and application programs, a network interface controller is like any other I-O device. However, the network interface is a gateway to communication with potentially millions of other devices. From the viewpoint of the information technology professional, the external connection of the network interface is likely to be more demanding of attention than its nature as an I-O device. For that reason, we cover networks separately in chapter ****.

0.2 Input and Output Concepts

The central processing unit can initiate input or output operations. Sometimes input operations happen as a direct result of an action by a program running on the CPU. Reading a block of data from a disk is an example. In other cases, a program must be able to "listen" to a device until it presents input. The CPU can't read a character from the keyboard until someone presses the key.

0.2.1 Architecture for Input and Output

The CPU, and programs running on the CPU, need to communicate with the computer's input and output devices, or more usually, device controllers. Device controllers offload some of the work of managing a class of devices from the CPU, leaving more CPU resources for other tasks. Four categories of information need to be exchanged: device address information, commands from the CPU to a device, status information from the device to the CPU, and data, which may flow in either direction depending on whether an input or output operation is taking place.

Memory-Mapped and Port-Mapped I-O

There are two architectural approaches to exchanging information, memory-mapped I-O and port-mapped I-O. In memory-mapped I-O, some of the addresses that would otherwise be part of main memory instead refer to device registers. Often these addresses are at the highest part of the logical address space, where one might not expect actual memory to be installed in any case. It is up to the address decoder (see Section 3.5.2 ****) to sort out which addresses are actual memory locations and which refer to device registers.

In port-mapped I-O, there is a separate address space of I-O ports. This has the advantage of allowing the full physical memory address space to be used for memory and of separating I-O devices, which are usually slower than memory, from the operation of the memory subsystem. The disadvantage is that separate instructions for input and output are needed, as with the Intel x86 IN and OUT instructions.

It is possible to combine the two approaches, as in some early small computers in which display memory was mapped into the main memory address space, but other input and output was handled separately.

Regardless of the addressing approach chosen, the actual operation of input and output is nearly the same.

Block and Character Devices

Some I-O devices transfer data a character at a time and some transfer larger blocks of data. It is obvious that the keyboard transfers data a character at a time. Someone running a word processing program presses a key and the resulting character appears on the screen. It may be less obvious that the mouse sends data for every increment that the mouse moves. The frame buffer for monitor output is a character device.

Magnetic disks, solid-state disks, flash storage, and similar devices are block devices. Such devices have a physical block size established when the device is manufactured, or possibly when it is first initialized by the operating system. All transfers are in multiples of that physical block size. You cannot read or write a single byte on disk. To update one byte, the entire block where it is located must be read, the byte changed, and the entire block written back to disk. For disks, blocks are likely to be 512 or 4,096 bytes. Solid state drives (SSDs) present a special problem for the developer of device drivers: they can only be erased a block at a time, and the blocks are big, up to half a megabyte.

Interrupts

The CPU could command an I-O device to take some action, then repeatedly check the device status to see whether the action was complete. Such a process is called *busy waiting* because the CPU can't do anything else while it is repeatedly checking the I-O device. It is clearly better to have the I-O device send a signal when the requested operation is complete. Such a signal is called an **interrupt**. An interrupt is an electronic signal to the CPU that changes the flow of instructions. In other words, it interrupts what the CPU was doing in order to do something else. At the level of instructions, an interrupt will cause the

machine state[1] to be saved so that the interrupted program can be resumed. Then an interrupt handled routine is run to deal with the signal from the I-O device. Finally, the interrupted program is resumed.

For example, when the operation system is initialized, it may command the keyboard controller to read a character. The associated interrupt is not generated until a key is pressed. The keyboard interrupt handler will translate the code from the keyboard into a character code and pass it along to the active program. Until the key is pressed the CPU can do other work. As this example shows, the time when an interrupt will occur is unpredictable.

A **software interrupt** is a signal generated within the CPU itself that changes the flow of instructions. Software interrupts are caused by the execution of special instructions, as when an application program wants to call on the operating system for some service.

A **trap**, as discussed in Chapter 3 **** is a third kind of interrupt and happens when an error like arithmetic overflow or division by zero occurs. A trap causes a call to the operating system, which generally handles the error by generating an error message and ending the failed program.

0.2.2 I-O Buses

1 Saving the machine state means saving the program counter, the contents of the registers, and possibly a small amount of other information. Since the program counter holds the address of the next instruction, restoring the machine state will cause the interrupted program to resume.

0.2.3 Programmed I-O

0.2.4 Interrupt-Driven I-O

0.2.5 Direct Memory Access I-O

0.2.6 Scheduling and Device Priority

Includes cycle-stealing and bus contention

0.3 Storage Devices

0.1.1. Hierarchy of Storage – Access Time and Transfer Rate
Includes sequential vs. direct access
0.1.2. Magnetic Disks
0.1.3. Solid-State Disks and Flash Drives
0.1.4. CD and DVD Devices
0.1.5. Magnetic Tape
0.1.6. Redundancy – RAID and ZFS

0.4 Displays and Printers

0.5 Other Devices

Includes keyboards, mice, touch screens, scanners, cameras, etc.

0.6 Summary of Learning Objectives

*This section of the chapter tells you the things you should know about, but not **what** you should know about them. To test your knowledge, discuss the following concepts and topics with a study partner or in writing, ideally from memory.*

0.7 References