Chapter 0
Input and Output

Without a way to get data in and results out, the CPU and memory are useless. The mechanism for getting data in and results out is called input and output. It may surprise you to learn that one of the most important technologies for input and output was invented long before electronic computers. Herman Hollerith invented methods for using cards with holes punched in specific locations for recording and later tabulating information. Hollerith's machines were used in the United States census of 1890. Punched cards were used for input and output for over a hundred years. Hollerith's company, The Computing-Tabulating-Recording Company, became IBM.

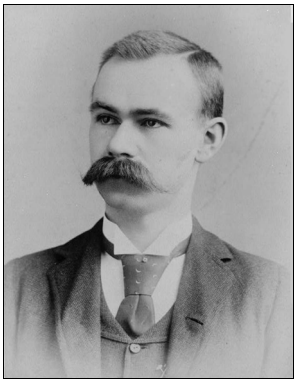

Figure 0-1
Herman Hollerith invented punched card tabulating equipment

When a program developer 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.

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, or periphery, 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 ****.

0.1 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. Even in that case, there must first be a command from the CPU.

0.1.1 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, with 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 between CPU and controller: 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.

0.1.2 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. Although an interrupt can arrive as a signal directly connected to the CPU, in modern computers, it is more likely to arrive through an interrupt controller that offloads some of the work from the CPU, or as a message on an input/output bus.

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 the operating system runs an interrupt handler routine to deal with the signal from the I/O device. Finally, the interrupted program is resumed.

For example, when the operating 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, such 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

---

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 at its next instruction.

a call to the operating system, which generally handles the error, usually by producing an error message and ending the failed program.

Sometimes it is necessary to suspend handling of interrupts. This is called **disabling interrupts** or **masking interrupts**, although the interrupt signals will be held until interrupts are re-enabled. One reason is to perform an atomic operation, such as setting a semaphore[2] for inter-process communication. Interrupts should only be disabled for very short periods, a few instruction times at most.

Some interrupts indicate serious errors, such as a memory error. These **nonmaskable** interrupts can occur even when interrupts are disabled.

0.1.3 I/O Completion

All input and output operations start with a command issued through the CPU, usually by the operating system. There are three ways of completing I/O operations. One is busy waiting, as described in Section 0.1.2. *** Busy waiting is found only in special circumstances because busy waiting means the CPU cannot do other work. Since most CPUs can execute millions of instructions in the time needed for one I/O operation, this is a serious drawback. The other two ways of completing I/O operations are interrupt-driven I-O and direct memory access I/O, or DMA.

Interrupt-Driven I/O

In interrupt-driven I/O, the I/O device or device controller raises an interrupt each time a character is available on input, or has been processed on output. The interrupt causes the operating system's device driver to run. On input, the device driver reads the character from the device's input data register and passes it to the appropriate application program or operating system routine. On output, the device driver places the next character in the output data register and signals the device that the next character is available, or concludes the I/O operation.

---

2 You will study semaphores in a course on operating systems. When one process is changing a semaphore, the operating system must ensure that no other process can read it until the change is complete.

Interrupt-driven I/O is used for character-oriented devices like keyboards, mice, and touch screens. While it might at first seem wasteful to interrupt a running program every time a character is typed or a mouse moves one increment, there are two important considerations. The first is that computer users expect immediate feedback when operating user interface devices. Second, the CPU can execute millions of instructions between such interrupts.

Direct Memory Access I/O

Block devices can be attached to the computer system using a direct memory access (DMA) controller which has access to the memory bus. When the CPU initiates a DMA I/O operation, it provides the address of the device, the location of the data on the device, perhaps as a logical block number, and whether a read or write is wanted. In addition, the CPU provides a memory address. The DMA controller transfers data from device to memory on a read, or from memory to device on a write. The CPU receives an interrupt only after the entire operation is complete. Instead of hundreds or thousands of interrupts, there is only one indicating the completion of the entire operation.

The trade-off is that, while a DMA device is using the memory bus, memory access is not available to the CPU. This drawback can be mitigated through using dual-ported memory, increasing the cost of the system. In systems where each DMA-capable device has its own DMA controller, there may be multiple devices contending for memory access.

Use of DMA also presents a problem of *cache coherence*. If a memory location is cached and a DMA operation overwrites that portion of memory, the cached value no longer matches memory. The problem of cache coherence can be mitigated by having the cache subsystem monitor the memory address lines and invalidate any cache entries that are written by direct memory access. This is called a snooping cache, and increases hardware cost. Since DMA operations are initiated by the operating system, the OS is aware of memory locations that will be overwritten, and can invalidate any cache entries for those locations. In that case, the penalty is time rather than money.

In addition to storage, video, sound, and network devices may employ DMA to speed up I/O operations. In some architectures, DMA can be used to move blocks of data from one memory location to another.

0.1.4 I/O Buses

A bus is a collection of electrical conductors or optical fibers that transmits data, address information, control signals, and sometimes power over distances from millimeters or less up to one or a few meters at relatively high speeds. There are buses internal to chips themselves, such as the CPU and memory chips, buses on the circuit board that holds the CPU and other components, and buses that connect peripheral devices, either within the computer enclosure or external to it. The buses that connect peripheral devices are I/O buses.

Principle characteristics of I/O buses are signaling speed and whether data are transmitted one bit at a time or several bits in parallel. If multiple devices are connected to the same bus, there must be a mechanism for bus arbitration to share access to the bus among the multiple connections. Some buses transmit power as well as data and control signals. There have been numerous bus standards for various purposes since the 1980s. The computer industry is now focusing on relatively few modern standards.

Twenty-first century I/O buses transmit data serially, that is, one bit at a time. The reason is a problem called bus skew. Earlier buses attempted to achieve high transfer rated by using multiple data conductors and transmitting eight or more bits at a time, each on its own conductor.[3] For all the bits to arrive at the destination device at the same time, the conductors must be exactly the same length or nearly so. For relatively slow signaling speeds, that wasn't a problem. As bus speeds increased, bus skew became a problem. A bit that arrived late became a part of the following byte of data, introducing errors. If bits are transmitted serially, one after another, they necessarily arrive in the order they were sent.

---

3 Such a bus is called a parallel bus.

There are three main types of I/O buses in current use, the universal serial bus (USB), the serial ATA bus (SATA), and the PCI express (PCIe) bus. Two other bus types, high-definition multimedia interface (HDMI) and DisplayPort are used primarily for display devices.

Universal Serial Bus

The universal serial bus (USB) is primarily an external bus. It was developed in the 1990s to address the problem that every device such as keyboard, printer, game controller, or modem connected to a computer had a different interface. To keep users from plugging a device into the wrong interface, the connectors were made different, requiring different cables. Several hardware and software companies led by Ajay Bhatt of Intel, came together in 1994 to develop a common interface for such devices. Goals of the group included ease of use, robust connectors, simplicity of interface design, and not adding manufacturing cost to computers. The interface should also supply power to devices with low power requirements like keyboards and mice. Devices should be hot-swappable, that is, it should be possible to plug and unplug devices with power on.

Although Apple was not one of the original developers of USB, the fact that the iMac, released in September 1998, had only USB ports attracted attention and USB became widely accepted. The line of Apple Macs announced in late 2020 may be the first computers to incorporate USB4. The USB standard has evolved through four generations. The USB Implementers Forum currently maintains the standard and assigns USB vendor IDs.

Every USB device has a 16-bit vendor ID and a 16-bit product ID assigned by the vendor (manufacturer) that in combination make a unique identifier. Together these identify a suitable device driver for the device. Every device also has an eight-bit device class, such as human interface device, printer, or mass storage device.

The original design of USB included a single host per bus, almost always a computer, and some number of peripheral devices. One computer can support more than one USB host, and so more than one bus. One physical connection

can include more than one function; for example, one device with a single connection might include both a camera and a microphone. All activity on the bus is controlled by the host.

The design of USB includes hubs, which allow connecting multiple peripheral devices to one host or upstream connection. The host is called the root hub and is at the top of the connection hierarchy. Additional hubs can be connected downstream from the host, up to seven levels. Each hub can connect multiple peripheral devices, up to a total of 127 devices.

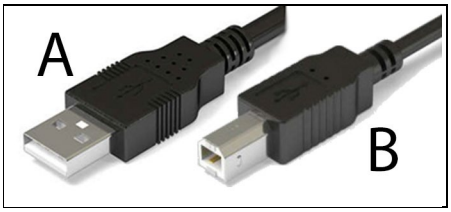

Figure 0-2
USB cable showing host connector A and device connector B.

Original USB cables had a host or upstream end, called A, and a device or downstream end, called B, with incompatible connectors. This was necessary because the bus provides power, and the different connectors prevent connecting two power sources together. The two connectors were designed so that they could only be inserted face up, preserving the orientation of the electrical conductors within the cable.

USB 1.1 had a 1.5 megabit per second low speed mode which did not require shielded cable and was used for devices like mice, which need a thin, flexible cable. The specification also had a twelve megabit per second high speed mode for devices like printers. The high speed mode required a thicker, shielded cable.

USB 2.0 introduced a 480 Mb/sec high speed mode and maintained backward compatibility with older devices. The specification also included mini and micro USB connectors for devices like smartphones and tablet computers, for which the standard connectors were too thick.

One of the most interesting features of USB 2.0 was USB On the Go, also called USB OTG or dual role mode. It is useful to have smartphones and tablets emulate mass storage for connection to desktop or laptop computers. Doing so requires them to have the role of peripheral devices. It is also useful to connect devices like keyboards and printers to smartphones and tablets which requires

them to have the role of hosts. USB OTG solved that problem by allowing a device to be either a host or a peripheral device. That is accomplished by using a five-pin connector in which the fifth pin, called the ID pin, is connected to ground on the host or A end and unconnected on the B end. An addition to the USB protocol allows a role swap.

USB 3.0 introduced five additional pins in plugs and sockets that are backward compatible with USB 2.0. USB 3.0 connectors with the additional pins are marked by blue inserts. The additional pins provide two more pairs of shielded signal conductors and, as of USB 3.2 Gen 2x2, provide transfer speeds of 20 gigabits per second.

The USB Power Delivery standard, USB PD, was completed in 2012. It allows a powered device and an unpowered device to negotiate the amount of power to be delivered by exchanging data between the two devices. Version 3.0 of USB PD can provide up to 100 watts of power at 20 volts and five amperes.

Type	USB Version	Cable Length
CC2	USB 2.0	≤ 4 m.
CC3G1	USB 3.2 Gen1 and USB4 Gen2	≤ 2 m.
CC3G2	USB 3.2 Gen2 and USB4 Gen2	≤ 1 m.
CC4G3	USB4 Gen 3	≤ 0.8 m.

Table 0-1
USB-C Cables
Current carrying capacity is indicated by a suffix
USB 3.0 Promoter Group, 2019

The USB-C connector was finalized by the USF-IF in summer, 2014. It removes the distinction between A and B cable ends. Because the pins are rotationally symmetrical, the connector has no concept of face up or face down; it can be plugged in either way. The distinction between power supplying and power consuming ends is now handled within the devices themselves rather than by requiring different connectors.

Not all cables with USB-C connectors are alike. There are four defined types of USB-C cables as shown in Table 0-1. Each type has a variant that can deliver power at three amperes and another that can deliver power at five amperes, a total of eight variations. All but the USB 2.0 three ampere version are required

to include an E-Mark[4] or electronic mark circuit in the cable's connector that can be interrogated to find the capabilities of the cable. "Full featured" USB-C cables have 18 electrical conductors, including two shielded pairs for data and a power conductor that will support three or five amperes of current.

Active cables include circuitry in addition to the E-Mark circuit that amplifies and conditions the data signals to allow cable lengths longer than defined in the specification. All fiber optic USB-C cables are necessarily active cables.

There are cables with USB-C connectors on the market with unsafe combinations of connectors that could allow connection of two power sources, and even cables that are reported to have damaged equipment to which they were connected. Cables that allow more current than they are designed to carry could overheat and cause fires. The USB-IF has a program for certification of cables. Looking for the "USB Certified" mark is one way to avoid substandard cables.

The USB4[5] specification, announced in 2019, provides up to 40 gigabits per second; USB-C cables are required. USB4 includes Intel's thunderbolt specification, which Intel released to the USB-IF for use in USB devices. Thunderbolt compatibility is likely to be found mainly in desktop and laptop computers and less frequently in smartphones or tablets. In addition to fast data transfer, USB4 provides mechanisms for tunneling DisplayPort signals.

Serial ATA Bus

PCIe Bus

The PCIe root bridge hides the complexity of a packet network and presents the CPU with something that looks like a bus.

https://www.fpga4fun.com/PCI-Express4.html

4 This is not the same as the European Union automotive safety E-Mark.

5 The official spelling does not include a space.

Connection to the CPU

The original design for microcomputers involved a bus connection at the CPU package, with pins for data, address, control, and status signals. The CPU bus connected to multiple input/output controllers for different classes of I/O devices. Some designs limited the bus clock speed to that of the slowest device on the bus, and all required bus arbitration circuitry to control access to the bus and prevent multiple devices from trying to use the bus at the same time.

Moore's Law allowed designs to evolve so that fewer physical components provided greater functionality. By the beginning of the 21st century, designs for personal computer class machines used a bridge architectures, with a high-speed interconnection device called a **host bridge** connected directly to the CPU's bus pins. The connection to the CPU bus was called the **front-side bus** to distinguish it from other buses in the computer design. High-speed devices, at first main memory and graphics devices, were connected directly to the host bridge. The widespread adoption of the PCI express bus resulted in designs that included a PCIe root node as an attachment to the host bridge.

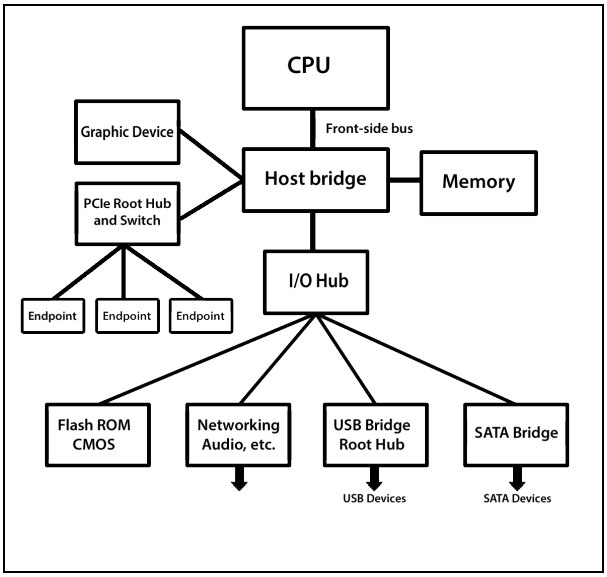

Figure 0-3
Computer organization with bridge architecture

By the second decade of the 21st century, increasing transistor density allowed the functions of the host bridge to be migrated to the CPU die, and so packaged with the CPU cores. Still later, some designs replaced the functions of the host bridge with a PCI Express root complex with integrated memory controller.

This design presents the CPU with a bus interface but connects devices, including graphics devices, with the packet-switch PCIe interface.

0.2 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. Redundancy – RAID and ZFS

0.1.4. Solid-State Disks and Flash Drives
https://blog.westerndigital.com/nvme-important-data-driven-businesses/
embedded MultiMediaCard (eMMC) and solid-state drive (SSD) storage have a lot in common, including the use of NAND flash memory, SSDs almost always deliver superior performance and are available in far larger sizes for bulk storage. In most cases, if you're not shooting for a budget PC, we recommend opting for an SSD over eMMC storage.

slightly longer tl;dr - writing to a NAND cell requires juicing it with some electric current in order to change its state (from 1 to 0 or 0 to 1), and over time, some charge is trapped in the NAND gate's dielectric layer. The trapped charge changes the NAND gate's resistance, which increases the amount of current required to make the cell change state. Eventually, the amount of current required to cause the NAND gate to flip is greater than the amount of current the drive's electronics can deliver, and that cell is marked bad. Over a long enough time—which varies greatly depending on the quality of the NAND and the strategies the drive's electronics use to spread writes out among cells—the drive reaches a point where it doesn't have enough cells to function and it dies.

0.1.5. CD and DVD Devices

0.1.6. Magnetic Tape

0.3 Displays and Printers

0.4 Other Devices

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

0.5 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.*

0.6 References