Computer hardware architecture

Bryan Ruddy

9 May 2022

Now that we have discussed data, we are prepared to go over the question of what computers do with it. Computers are everywhere in our lives today, but what, exactly, are they? After completing this module, you should be able to identify the major components that make up a computer along with their functions, describe the major performance measures for each of these components, and understand how the different combinations of these components can be used to solve problems.

A brief history of computing

The term "computer" dates to the 1600s, long before anyone dreamed of a machine that could perform calculations. Instead, it referred to a human occupation: a person who performs mathematical calculations. Originally tasked with jobs like preparing tables of logarithms and trigonometric functions, over time the range of tasks performed by computers expanded to cover the processing of astronomical observations and the calculation of artillery trajectories. Even as late as the 1950s and 1960s, human computers were essential in calculating trajectories for the American space program, as immortalized by the 2016 film *Hidden Figures*.

This space at the Royal Observatory in Greenwich, the Octagon Room, housed computers in the 1800s.[1]

In a quest for increased precision, increased speed, and reduced cost, mechanical computers were invented in the 1800s, followed by electromechanical computers in the 1930s. These computers were limited, in that they were built to solve only one problem or, at best, a limited set of problems, but they could be far more effective than humans in solving them.

Programmable computers came later, in the 1940s, and were designed to execute arbitrary programs - they could solve any problem that theoretically could have its solution computed. (Even the theory of what it means for a mathematical problem to be computable was only fully developed in the 1930s, by Alan Turing.) These early electronic computers used vacuum tubes; transistors were introduced in the 1950s, with far smaller sizes and lower energy consumption. Integrated circuits, combining multiple transistors micro-fabricated on a single piece of material, were developed in the early 1960s, and have since grown in size and speed so that they now easily hold billions of transistors per device, with each device performing trillions of mathematical operations per second.

A fully programmable mechanical computer was designed in the mid-1800s by Charles Babbage, but proved too complex to economically build. This is a tiny part of the mechanism, built 70 years later by his son as a four-function calculator.[2]

Through all of these developments the basic concept of what makes up a computer has remained largely unchanged, even if the physical devices performing those functions have not. Let's now consider what these key functions are, and discuss some of the key properties of them along with examples of devices that perform the

functions.

Computing

The basic idea of computing can perhaps most easily be explained with reference to human computers, as you have been trained to work as one in many of your math courses to date. The core idea of computing is to solve a mathematical problem by breaking it up into a number of elementary sub-problems, then combining those solutions to answer the original question. For example, in performing long division, you can break down the process of dividing two large numbers into a sequence of simple subtractions and multiplications, plus data copying and decision-making. An example of how you might do this is given below:

$$
\begin{array}{r}
0 \\
17\overline{)357} \\
3 \\
-0 \\
\hline
3
\end{array}
\qquad 3 - 17 < 0
$$

$$
\begin{array}{r}
02 \\
17\overline{)357} \\
3 \\
-0 \\
\hline
35 \\
-34 \\
\hline
1
\end{array}
\qquad
\begin{array}{l}
35 - 17 = 18 > 0 \\
18 - 17 = \ \ 1 > 0 \\
\ \ 1 - 17 < 0
\end{array}
$$

$$
\begin{array}{r}
021 \\
17\overline{)357} \\
3 \\
-0 \\
\hline
35 \\
-34 \\
\hline
17 \\
-17 \\
\hline
0
\end{array}
\qquad
\begin{array}{l}
17 - 17 = 0 \geq 0 \\
\ \ 0 - 17 =< 0
\end{array}
$$

1. Start from first digit of the dividend D, and with an empty output and remainder.
2. Append this digit of D to the remainder.
3. Set $n = 0$
4. Subtract the divisor d from the remainder.
5. If result is positive, increment n and go to step 4.
6. Append n to the end of the output.
7. Subtract $n \times d$ from the remainder.
8. Set the remainder equal to this result.
9. If this was not the last digit, move to the next digit and go to step 2.
10. Return the output.

This algorithm breaks down the operation of division to simpler, elementary operations of subtraction, addition, multiplication, and comparison. It also contains conditional branch points ("if" ... "go to"), and contains operations ("set") that store data in *memory*, in this case either written on a piece of paper (the remainder) or mentally tracked (n). These simple operations are easy to do, such that a reasonably-trained person can do so without steps in between, even for numbers where doing the division itself mentally would be extremely difficult. An expert might be able to use a more efficient process, by mentally dividing the expanded remainder in step 2 by the divisor rather than repeatedly subtracting. If the dividend and/or divisor were especially large, it might be necessary to store n on the paper, or even use multiple sheets of paper. In an extreme case, one might even need to store the paper in a filing cabinet, and

grab the right page as needed.

Practical computing

Modern electronic computers are almost unimaginably faster than human hand-calculations, but all of the same principles still apply. The different aspects of computing correspond to different kinds of physical computer hardware, which perform these aspects.

Elementary operations

Just as in the human example, electronic computer programs must be broken down and described using a small number of distinct operations. The set of distinct operations is called the *instruction set*, and a human-readable program using only these operations is said to be written in *assembly language*. Rather than being performed by the human brain, we use an electronic *processor* to perform these operations in a computer. Different processors have different instruction sets, just as an expert might be able to do some divisions of small numbers directly, or a younger child might have to repeatedly add rather than multiplying in one step.

In order to make the program readable by a processor, which can only work with binary data, the assembly code must be translated into *machine language* instructions. Machine language instructions are encoded as numbers, and when humans need to look at them they are written in hexadecimal. Each instruction in the instruction set corresponds directly with both a human-readable assembly language instruction and a binary machine language instruction.

It is almost never necessary for a human to examine the machine language code, due to this direct correspondence. To quote Douglas Hofstadter, "Looking at a program written in machine language is vaguely comparable to looking at a DNA molecule atom by atom."[3]

Typical kinds of elementary operation include the following:

- Store a constant value
- Transfer information between the processor and memory
- Arithmetic (add/subtract/multiply/divide)
- Logical operations (AND/OR/NOT)
- Bitwise operations (flip individual bits)
- Comparisons
- Call a function
- Start executing instructions elsewhere in the program (conditional or unconditional branch)

Data storage

Also like the human example, an electronic computer has different kinds of memory available to store results. Using memory takes time, whether it means remembering something only tracked in your mind or physically writing on a piece of paper. Some memory

is small but fast, like mentally keeping track of a small number; some is huge but slow, like a filing cabinet full of papers, and some is in between. There is usually a trade-off between the capacity of a type of memory and the ease with which it can be accessed. Each different memory type can be physically realized in different ways, with different kinds of electronic devices.

Communication

What may be less clear from the human example is that there is also a need to *transfer* data. We imagine solving a long division problem while sitting at a desk with a piece of paper, but what if you need to get your result to someone else? You could simply speak the number, but this takes about one second per digit, and only works if they are nearby. Alternatively, you could do your writing in large print so you could hold it up and they could read it from across the room, which is typically 3–4 times faster. Faster still, at least for you, might be to just fold the paper into an airplane so you can throw it to them—they could receive thousands of digits in only a few seconds this way! Computers likewise have a variety of electronic data transfer methods available, with different speeds and capabilities.

Just fun, or actually a fast method for data transfer?[4]

Finally, the human body conveniently comes with input (eyes, ears) and output devices (hands, legs, vocal cords), that make it easy to act upon results. For computers, this takes more work, and so a computer system must be equipped with appropriate *I/O devices* to interact with the outside world.

Computers

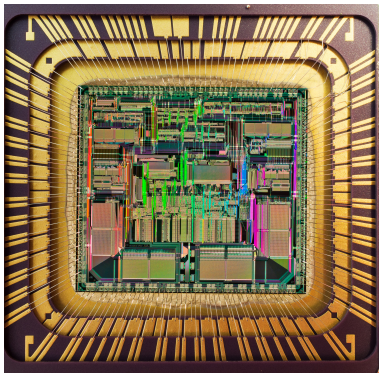
A computer must contain at least 1 processor, in order to perform computations; all practical computers will also include at least one form of memory, as well as at least one data transfer mechanism between memory and processor. To be useful, a computer will also include either a data transfer to other computers or at least one I/O device to translate computational results to the physical world. Most computers include both. In a typical *personal computer* (PC), each of these parts is one or more separate electronic devices connected together, and there are I/O devices provided specifically for human interaction. A *server* usually omits the I/O devices and only uses data transfer to other computers; a *supercomputer* is a very powerful kind of server incorporating many processors and memory devices. A *microcontroller* combines all of the parts of a computer in a single integrated circuit, but usually has I/O devices intended for interacting with the physical world rather than for interacting directly with humans. Microcontrollers also typically have much lower performance than a standard computer. Despite their low performance, however, microcontrollers are the most numerous kind of computer due to their simplicity and low cost.

A modern smartphone is really just a small PC with a touch-screen for I/O and a cellular radio for data transfer.

We will now discuss the different elements of computer hardware in more detail.

Processors

The processor is the part of a computer system that actually performs the computation. As such, it can be characterized by two properties: what instruction set does it offer, and how quickly can it do these instructions? A second question is that of the number and arrangement of multiple processors—modern PCs and phones have 2 or more main processors, housed together on a single integrated circuit (a *CPU*, or Central Processing Unit), with many more small processors used to manage memory, storage, and I/O devices. Even microcontrollers can contain multiple processors, potentially with different capabilities.



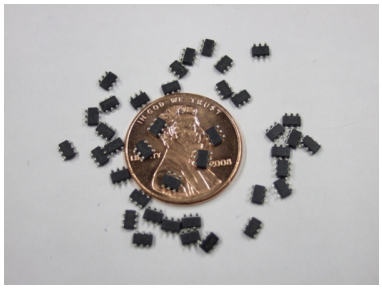
A 1990-era processor.[5]

Processor characteristics

As discussed in the module on information representation, processors differ in the availability of operations that work with larger numbers; a 64-bit processor will have many operations that work with 64-bit numbers, while an 8-bit processor will have few operations that can handle anything larger than 8 bits at a time. Processors working with larger numbers also typically have larger instruction sets, but different processor designs make different choices about which instructions to offer. For instance, a new smartphone will have a 64-bit processor designed by the British company ARM, with relatively few (442) different instructions. A PC will have a 64-bit main processor built by the companies AMD or Intel, based on a design from AMD that offers over twice as many (981) different instructions. Meanwhile, a blood glucose meter might be run using an 8-bit microcontroller designed by Intel in the 1980s called an 8051, which has just 49 different instructions available.

Intel made a poor choice in its own 64-bit design, and had to adopt its arch-rival's design instead!

Regardless of whether the instruction set is large or small, the instructions it contains can be combined to achieve any computational task. (In fact, even a single instruction is sufficient, if it's the right instruction!) When more instructions are available, they can be used to shorten the length of a machine language program, reducing the required storage as well as the number of instructions that need to be executed. In modern processors with large instruction sets, many of the instructions are used to perform computations on multiple pieces of data simultaneously, further reducing the required number of instructions.



Microcontrollers can be very small!.[6]

Processors also differ tremendously in the rate at which they can perform these operations. This is typically characterized by a *clock frequency*, measured in cycles per second (Hz); high-end and modern processors can typically execute one instruction per clock cycle, while older designs may need multiple cycles to complete one instruction. Some instructions, such as division, can take longer than

others even on modern processors. Other instructions are available on modern processors that compute several operations simultaneously, effectively increasing the calculation rate. Most modern processors will also anticipate the results of future calculations, to increase their speed; however, when these predictions are wrong, as might be the case around branching instructions, there is a delay of several clock cycles while the predictions are thrown out and results are calculated. Combining all these factors, the processor in a smartphone or PC will run with clock frequencies of several GHz, averaging up to 10 operations per clock cycle, while inexpensive microcontrollers like the one in a blood glucose meter may run at speeds as low as 12 MHz, and could require multiple clock cycles per instruction.

This prediction system was at the heart of two major security vulnerabilities in most processors announced in 2018, called Meltdown and Spectre.

How does this fit together? A processor with a larger instruction set may be able to perform a task in fewer instructions than one with a smaller instruction set, making it faster even at a lower clock frequency. For the most part, as a programmer you will not need to worry about this—compilers and interpreters are very effective at converting your code into the best set of machine code instructions for any given processor. However, this does mean that clock frequency is not a complete picture of processing speed: a processor with a faster clock might need more instructions to solve a problem than a processor with a slower clock, making it slower overall! Assuming equal levels of technology, a processor with a smaller number of instructions typically have a faster clock frequency than a processor with a larger instruction set.

Multi-processing

A greater challenge for the programmer is that all but the smallest and simplest computers now use multiple processors. Where these processors are contained in a single device, we refer to them as *cores*, but they still operate independently. When there are multiple processors, the program code will only use one of them unless it has been specifically written to divide tasks among several processors, a concept called *parallel programming* that you explored in the Performance module of this course. However, this arrangement does allow several independent programs to run simultaneously, keeping them from interfering with each other. Where there are simple programs that help to operate the computer hardware, they can be run on small, simple processors dispersed throughout the computer, making the job of the operating system simpler.

Why multiple cores? Around 2010, the effective speed of single processors stopped increasing, even as their size, cost, and power consumption continued to decrease. The only way to obtain higher performance from a computer became the addition of more processors.

Many computers also use specialized devices called *GPUs* (Graphics Processing Unit) to assist with the calculations needed to render 3-D graphics. These contain a large number (thousands) of simple 32-bit processors in a single device, with limited sets of instructions that are optimized for numerical calculation using floating-point numbers rather than decision-making. These processors are connected together so that groups of processors operate

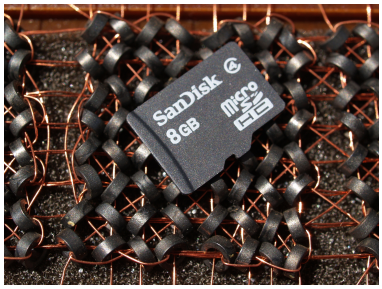
in lockstep, performing the same operations on different data. By using so many processors together, enormous rates of computation are possible, over 10 trillion operations per second! However, getting data into and out of so many processors is very challenging, as we will discuss later, and so this rate of computation is only available when many manipulations must be done on the same data. Application programs must therefore be specially written to take advantage of these capabilities. While GPUs were originally designed for graphics, they are now widely used any time many floating-point calculations are needed, especially in machine learning and AI.

New Zealand had the #482 supercomputer in the world as of November 2018, when it was commissioned, with 18,560 CPU cores and running 900×10^{12} operations per second. You might be able to use it in your part IV project!.[7]

For maximum performance, a supercomputer can contain thousands of individual CPUs and GPUs, and with specially-written software can solve problems that would need thousands of years to solve on a PC. The fastest supercomputer in the world today can perform 140×10^{15} operations per second across over 2 million processor cores, an average of 60 billion operations per second per processor. For comparison, the earliest electronic computers in the 1940s could only perform a few thousand operations per second, using a single processor!

Memory

As part of their design, processors include a very small amount of data storage (under 256 bytes) to hold the values being manipulated by the instructions, called *registers*. However, much more than this is needed to represent practical programs and the data they manipulate. Also, once the computer is turned off this data will disappear forever. Thus, storage devices need to be included in a computer, to hold more data and to preserve data across power cuts.



8 GB of memory from 2008, on top of 8 bytes of memory from 1960.[8]

Storage devices have three key properties that influence their performance: the rate at which data can be read from or written to them (*data rate*), the delay before beginning to read or write a piece of data (*latency*), and the capacity. Devices with larger capacity tend to have slower data rates and/or longer latencies. Faster speeds and lower latencies also come with higher costs per unit of storage capacity. There is thus a hierarchy of memory in a computer, with small amounts of fast memory and large amounts of slow memory, arranged to minimize cost while maximizing performance.

The performance of a computer is often determined by the performance of its memory, rather than its processor. For instance, consider an 8-bit processor that can perform 1 million operations per second, and has an operation that fetches 1 byte of data from memory. If this processor is attached to a memory with a latency of 5 microseconds, and a data rate of 500 kilobytes per second, then it will take 7 operations for a piece of data to actually arrive at the processor, once it has been requested. (5 operations waiting for the latency, plus two for the actual data transfer.) If the very next op-

Modern processors have several tricks used to reduce the severity of this problem, but slow storage devices like hard drives are too slow for any such tricks to fix the problem.

eration needed the data, the processor has to wait and do nothing until the data arrives! Thus, in writing a program one must pay attention to how and when data are used and stored.

Memory devices

There are two main kinds of storage device: *volatile* storage, which is erased when the computer loses power, and *non-volatile* storage, which can maintain its data long-term. Typically, volatile storage is faster but lower in capacity than non-volatile storage.

The main kind of volatile storage in use is called *RAM*, for Random-Access Memory. Every piece of data in RAM can be accessed with the same latency, making its use highly predictable. Computers have two main kinds of RAM: *cache*, which is located on the same device as the processor, and *main memory*, which is shared across all of the processors in the computer. The cache is designed with the processor, so that code running on the processor that can work with data in the cache effectively cannot see any delay caused by cache speed. When the code needs data from outside the cache, however, it will be delayed while the data are fetched from main memory. Main memory is very fast, but still has a latency of about 100 nanoseconds and a data transfer rate of 10 to 20 GB/s in modern PCs and smartphones.

Microcontrollers often only have one block of RAM, with performance similar to cache.

There are several different kinds of non-volatile storage available, with widely varying performance. Some kinds of non-volatile memory are *read-only* (ROM): they have fixed content, set at the time of manufacture or during the first use. This can include both integrated circuits, that otherwise act like RAM, and removable devices, like CDs, DVDs, and Blu-ray, which have low data rates (75 MB/s at most for Blu-ray) and long latencies (up to 1 second). While ROM historically had an important role to play, other storage technologies have leapt ahead in terms of performance and cost, such that its usage has been steadily decreasing. The main advantage it retains over other technologies is archival value—optical disks are rated for lifetimes of hundreds of years, much longer than the drives that read them! (Re-writable optical discs are also available, but are no longer cost-effective or of acceptable performance compared to other options in 2022.)

Read-only memory in practice.[9]

Other non-volatile storage looks much like RAM, made of special transistors on an integrated circuit that are capable of changing state in response to the application of a high voltage. The most common form of this storage today is called *flash memory*, which can only be read and erased in blocks of hundreds of bytes but can be manufactured in large capacities relatively cheaply. Flash memory has a limited lifespan, typically only surviving 100,000 write cycles, but this is sufficient for many applications. It also may only retain data for a few years, depending on environmental conditions, and should therefore not be used for archival storage.

Flash memory is used both in fixed applications, where it is a

permanent part of the computer, and removable, where it can be used to transfer data between computers. The most common example of a fixed application is in a *solid-state drive*, used as the main non-volatile storage for a PC or mobile phone. Solid-state drives currently support read and write speeds of up to 3 GB/s, with latencies as low as tens of microseconds in high-end drives. Removable flash memory is used in devices that connect to standard computer ports, such as USB drives, and in special memory cards that fit in more compact spaces. As of this writing, these memory cards are available with the highest physical data storage density of any device—a microSD card can store over 500 GB in a similar volume to a fingernail, with a mass of only 250 mg. SD cards offer speeds up to 90 MB/s, though 10 MB/s is more common; latency when reading is typically a few milliseconds, but when writing it may be as large as hundreds of milliseconds.

The other main form of non-volatile storage uses the magnetization of iron-bearing materials as its storage medium. The most common form of this today is the hard disk drive, which contains a set of rapidly-spinning, magnetically-coated disks in a controlled environment. Hard drives are much more cost-effective than flash memory for storage of large amounts of data, but have lower data rates and worse latency. For example, in 2020 a low-performance 1 TB hard drive could be had for about $75, while the least expensive 1 TB SSD cost about $250, or over 3 times more. (The price differential was a factor of 6 at a capacity of 4 TB!) However, the hard drive transfers data 3 times more slowly, and has up to 1000 times greater latency (about 10 ms) compared to the SSD.

Magnetic media are also used for removable storage, most iconically in the form of the floppy disk. Even though the venerable floppy disk is long out of use, magnetic media sees use behind the scenes to archive large amounts of data on magnetic tape. Tape has been used for many decades as a form of high-capacity storage, even though it can have a latency measured in minutes. Tape remains the cheapest way to store very large (hundreds of TB) amounts of data, costing about half the price of a hard drive, offering a comparable speed to a hard drive, and coming with a longer lifetime for archival storage. In data centres, robotic tape libraries are used to store petabytes (thousands of terabytes) of data, accessible in just a few minutes at all times.

Once ubiquitous, now remembered only as the "save" icon.[10]

Memory usage

Some types of memory are managed automatically by the processor hardware or the operating system. For instance, application programs cannot directly choose to keep information in the cache—this is decided by the processor itself. Non-volatile storage is managed by the operating system to appear as a set of files, which can be manipulated by applications, but applications do not access the raw binary data as stored on the device. Further details of how the

operating system and the internals of the programming language manage storage will be discussed in the module on computer software architecture.

There is one important way in which applications do need to manage their own use of storage, however. This relates to the use of non-volatile storage, particularly in a data centre environment: when can a piece of data safely be transferred to slower, cheaper storage? This must be determined for every different application, to balance user experience with economics, and further detail is beyond the scope of this course.

A problem like this is an excellent place to apply operations research, to find the optimal strategy!

Data transfer

All of the different parts of a computer system are connected together by data transfer devices, often called *data buses* when they use wires, which must convert information into a physical signal to move it from one place to another, then interpret it at the other end from a signal back to information. Data transfer devices can also be used to connect multiple computer systems together, to exchange data.

Today, almost all modern data transfers work *serially*: they break up the data to be transferred into individual bits, which are sent one-by-one from source to destination. Data can be transferred faster by using several such connections simultaneously, with as many as 32 simultaneous connections used in the fastest links inside a modern PC. The data transfer device manages the conversion of data in standard chunks of bytes to and from these streams of serial data.

This is a slight over-simplification—many data transfer methods used for long-distance communication represent data as more complex symbols, where one observable state at a moment in time can have as many as 15 bits of information encoded by it.

Like storage devices, data buses are also characterized by their latency and their data transfer rate. The latency indicates the delay before a transfer can begin, while the data rate indicates the time it will take a given amount of data to be transferred once the transfer begins. Typically, data buses that must travel a shorter physical distance will support higher data rates and lower latencies; data buses connected to many devices will also suffer a latency penalty compared to buses with fewer devices, due to the likelihood that multiple devices will attempt to use the bus at the same time (a *collision*) and due to the mechanisms used to prevent this.

The table below summarizes a number of different commonly-used data transfer mechanisms, including their latency, data transfer rate, and usual purpose. Note the use of bits per second (b/s) rather than bytes (B/s), due to the serial nature of the links.

Mechanism	Rate	Latency	Application
DDR4	192 Gb/s	≈10 ns	Main memory to processors
PCIe 4.0	256 Gb/s	≈1 μs	Processors to I/O, storage, and other data links
SATA 3.0	6 Gb/s	≈5 μs	Storage devices
USB 3.1	10 Gb/s	≈1 μs	External storage and I/O
USB 2.0	480 Mb/s	125 μs	External storage and I/O
USB 1.1	12 Mb/s	1 ms	External storage and I/O
Ethernet (10G)	10 Gb/s	≈5 μs	Local interconnections
UFB Fibre	1 Gb/s	≈10 ms	Long-distance interconnections
ADSL	10 Mb/s	≈10 ms	Long-distance interconnections
I^2C	400 kb/s	Low	Embedded systems
SPI	10 Mb/s	Low	Embedded systems
LTE (mobile)	326 Mb/s	≈10 ms	Long-distance wireless links
WiFi 5	7 Gb/s	≈1 ms	Medium-distance wireless interconnections
Bluetooth 4.2	25 Mb/s	≈5 ms	Short-range wireless interconnections
Typing	35 b/s	Low	Data input
Reading	117 b/s	≈1 s	Data output

You will not be expected to memorize the individual data transfer mechanisms listed here, only to have a general sense for the performance available. Note that these latencies apply for only a single link—when communicating on a network, the latency may be much longer. Also, in long physical connections the delay due to the finite speed of light may become significant.

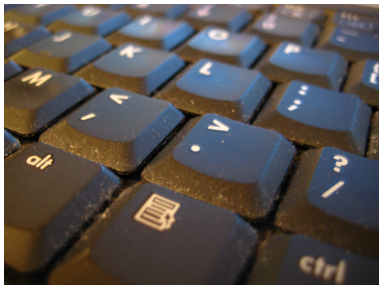
Finally, note that sometimes the best way to transport data is to do so literally, by moving a removable storage medium. It can often be faster to carry a hard drive than to transfer files over the internet. In an extreme case, consider carrying a milk jug full of microSD cards on an airplane. You can get almost anywhere in the world in 24 hours (ignoring pandemic restrictions), carrying 1.6 PB (16,000 TB) of data, an effective data rate of 1.5 Tb/s. The latency of this approach, however, would be a very high 24 hours. You might also have to develop special equipment for reading and writing many cards at once, given that inexpensive cards only support data rates of 80 Mb/s! While it sounds absurd, this very kind of approach is commercially available: Amazon will send a truck full of hard drives to collect your data, hundreds of times faster than any available long-distance communications link could carry it.

Delicious, delicious data...[11]

Amazon offers a truck full of hard drives as a service—100 PB of data on wheels![12]

I/O devices

For a computer to be useful, it must include some way to obtain data to work on (input), as well as some way to give the results of its calculations to the user (output). In some cases, this can be accomplished by using removable non-volatile storage, or by transferring the data to a different computer. In others, however, the data are generated by a device that functions as part of the computer, or sent to such a device to make some change in the physical world. These devices are called input/output devices, or *I/O devices* for short.



An input device you use almost every day.[13]

There are a wide range of different I/O devices that could be used. In a PC, these include devices like the mouse/trackpad and keyboard for input, and a display and audio generator for output. In an *embedded system*, a computer system (often consisting of just a microcontroller) contains I/O devices that relate to the physical world, like sensors that perform measurements and actuators that produce physical motion, and the computer system forms a smaller part of an overall mechanical or electrical system.

In most situations, where an operating system is available you do not need to worry about the details of how the I/O devices communicate with the other parts of the computer system in writing your code. The suppliers of the I/O devices write programs called *drivers* that form part of the operating system and handle this communication. However, if you have built a new I/O device for your system, or you are programming a small embedded system that does not use an operating system, you will need to learn how the device fits in with the rest of the system.

Other input devices, as on this Apple Watch, can measure biological signals. [14]

One common way that I/O devices are used is one in which they act like memory: a special set of memory addresses is set aside for their use, and changes to memory contents at that address directly relate to activity by the device. This method, called *memory mapping*, is common in microcontrollers, and requires the I/O device to be connected to the same data transfer mechanism as is used by the main memory. You will work with a memory-mapped device in the lab. Other devices use specific communications protocols to send and receive commands and data, over slower data transfer mechanisms, where they cannot be connected to the same data transfer mechanism as the memory.

Figure sources:

[1]By ChrisO, from `https://commons.wikimedia.org/wiki/File:Royal_observatory_greenwich.jpg` (GFDL license)

[2]By the Science Museum, London, from `https://collection.sciencemuseum.org.uk/objects/co62246/henry-babbages-analytical-engine-mill-1910-analytical-engine-mills` (CC BY-SA-NC 4.0 license)

[3]Hofstadter, D. R. (1979). *Gödel, Escher, Bach: An Eternal Golden Braid* (p. 290). New York: Basic Books.

[4]By Akkana, from `https://commons.wikimedia.org/wiki/File:Paperairplane.png` (GFDL license)

[5]By Gregg M. Erickson, from `https://commons.wikimedia.org/wiki/File:Motorola68040die.`

jpg (CC BY 3.0 license)

[6]From https://electronics.stackexchange.com/a/84851

[7]From https://www.nesi.org.nz/services/high-performance-computing/platforms (UoA copyright)

[8]By Daniel Sancho, from https://commons.wikimedia.org/wiki/File:8_bytes_vs._8Gbytes.jpg (CC BY 2.0 license)

[9]From https://commons.wikimedia.org/wiki/File:DVD-Video_bottom-side.jpg (Public domain)

[10]By Fredrik Alpstedt, from https://www.flickr.com/photos/alpstedt/9328234633/ (CC BY-SA 2.0 license)

[11]By Randall Munroe, from https://what-if.xkcd.com/31/ (CC BY-NC 2.5 license)

[12]By Amazon, from https://www.wired.com/2016/12/amazons-snowmobile-actually-truck-hauling-huge-hard-drive/

[13]By Tom Stefanac, from https://commons.wikimedia.org/wiki/File:Computer_keyboad_slanted_angle_dec07.jpg (GFDL license)

[14]By Fletcher, from https://commons.wikimedia.org/wiki/File:Apple_Watch_Series_3_Sensors.jpg (GFDL license)