The IGCSE Computer Science (Theory) Revision Guide

Eason Qin (eason@ezntek.com), et al.

February 2025

Contents

Introduction

What is this guide?

You are looking at the final IGCSE Computer Science revision guide (also referred to as the CSRG), this time covering the whole syllabus for the IGCSE mock and final examinations. The first ever CSRG was for the Comp Sci G1 semester 2 examinations at OFS, and the second was for the Comp Sci G2 semester 1 examinations at OFS.

This document aims to cover everything you need to know for the final IGCSE Computer Science 0478 examinations, for the 2023 2025 batch of IGCSE CS students. It aims to deliver the content in a concise yet informative form, short but with enough explanation to help develop an understanding for the content. If highlighting the guide helps you, you may do so.

This document is also prepared in LaTeX, a high-quality typesetting system that is code-based. It is the de-facto standard for the communication and publication of scientific documents.[1]

This is revision one of the guide.

NOTE: All references to ”I”, ”Me”, ”Myself” and similar refer to the main author, Eason Qin.

For a quick tutorial as to how to use this guide, read [placeholder]

[1] Taken from the LaTeXwebsite.

License Notice

This means that you may do the following:

1. You must attribute me, i.e. state that the work was produced by us, the creators if you use it as a part of your work or teachings, or expand upon my work.
2. You may use the guide for any purpose, you may use it to teach yourself or teach others, whatever you like.
3. You may share the guide with anybody else with no restrictions.
4. If you want to create derivative works[2], you are allowed to do so, as long as if you put the exact same license on it. If it is not written in the text, it will be implied. If you would like the document in its raw, editable form, you may ask me.
5. You may then share it however you please. You can then add yourself to the authors list.
6. You must not make money off of it. Failure to comply means that I, and all other contributors may take any legal action on you if needed.

[2] Works based on this one.

Important Information

This is mostly targeted to certain individuals who carelessly read this guide while cramming.

1. The chapter and section markers ***will not correspond to the textbook directly!*** Some of the content is deemed too trivial to write about; for those sections I urge you to use your textbook.

2.

Notes

1. This revision text is authored by Eason Qin Luojia, with contributors listed on the cover page; including and not limited to Siddharth Harish ando Karthik Sankar.

2. Some excercises for the Chapter Ten content on Logic Gates may be pulled from the textbook[3], but some are also generated by the authors.

3. **Formal Citations and a bibliography are not provided**, as this is not an academic research document, but a reference booklet of notes from the IGCSE Computer Science 0478 course offered at my school, along with content from the textbook (as mentioned previously). Since the work is mostly produced from either directly pulling examples from the textbook (which will be annotated) or already synthesized information, no references for those points will be provided. If there is information that *must* be cited, including and not limited to extremely detailed data points, the source will be provided as a foot note. **In no case will MLA, APA, Harvard or any other form of formal academic referencing be used.**[4]

4. If there is an underlined portion of text, like so:

   Chatbots have mostly been replaced by LLMs, simply go to the AI section below

   is seen, and you have the **printed copy**, simply go to the section it says; do so via the .

[3] Cambridge IGCSE™ and O Level Computer Science, Second Edition, ISBN 9781398318281

[4] Legally, all licenses will be followed; i.e. if the document has a license that requires attribution, the attrbution will be provided, etc.

Chapter 1

Data Representation

IMPORTANT!

This section does **NOT** cover the entirety of chapter 1's content. PLEASE refer to the textbook for things you do not know.

1.1 Applications of Hexadecimal

There are several applications of the Hexadecimal number system in Computers, which include and are not limited to:

- Making binary easier to write/represent
- OS error codes
- MAC addresses
- IPv6 addresses
- Color codes

Making binary easier to write/represent

Given long binary sequences, such as `1011101010111110`, programmers may find it much easier to simply express the given example as `BABE`, which is 4 characters and not 16 when typing/writing. Converting between the two formats was covered in this section.

OS error codes

OS error codes are given when a program has an error and must exit. Sometimes, they are represented in hex.

MAC addresses

These uniquely identifies a device on the network. They use 48 bits of data in total (6 segments of 2 nibbles), and look like `AA:BB:CC:DD:EE:FF`. Refer to section 2.3 for details.

IPv6 addresses

Refer to 2.3 for details.

Color codes

Color codes are used to represent colors on a computer. They describe the amount that specific colors mix together. Red, Green and Blue are used as the base colors, as they are the additive color primary colors (not yellow!) like `#ffcc00`. 2 nibbles or one byte is used to represent the magnitude of the color itself, from 00-ff. The previous example consists of FF red, CC green and 00 blue, which produces a yellow color.

1.2 Text, Sound and Images

1.2.1 Character Encoding: ASCII and Unicode

Since computers only work with binary[1], the concept of ”text” in a computer system...does not exist. Instead, they are just sequences of special numbers. However, what dictates what number corresponds to what character?

A very influential solution made in the 1960s is called ASCII. It maps the numbers 0-127 to text characters and control codes (which controls the device that shows the characters to you). For example, capital A is 65 in denary, B is 66, and the text character for 1 is actually 49 in denary. The reason is that in ASCII formatted text, if the numbers 0-9 were actually for the text characters 0-9, there would not be a space to put very important control codes, like the ”return” character. Therefore, in the ASCII *representation* of all text characters, the text character for 1 is different from the number 1.

Unicode is like ASCII, but it supports a much wider range of valid characters. In hex, there are 11ffff characters; or 1.1 million in decimal. This means that every character in every script along with every emoji and piece of displayable text fits in this massive lookup table. The first 127 characters are 1:1 compatible with unicode, in fact.

[1] and hex, but that’s just binary. The same goes for denary.

1.2.2 Sound

Sound waves are vibrations in the are, and can actually represented by mathematical waves. In fact, all sound waves can be expressed as a sum of pure sine waves following the form $a * sin(bx + c)$. Therefore, every sound wave has an amplitude (how high the wave goes), and a frequency[2], measured in hertz (how often it repeats, repetitions per second). Let us suppose we have the following sound wave:

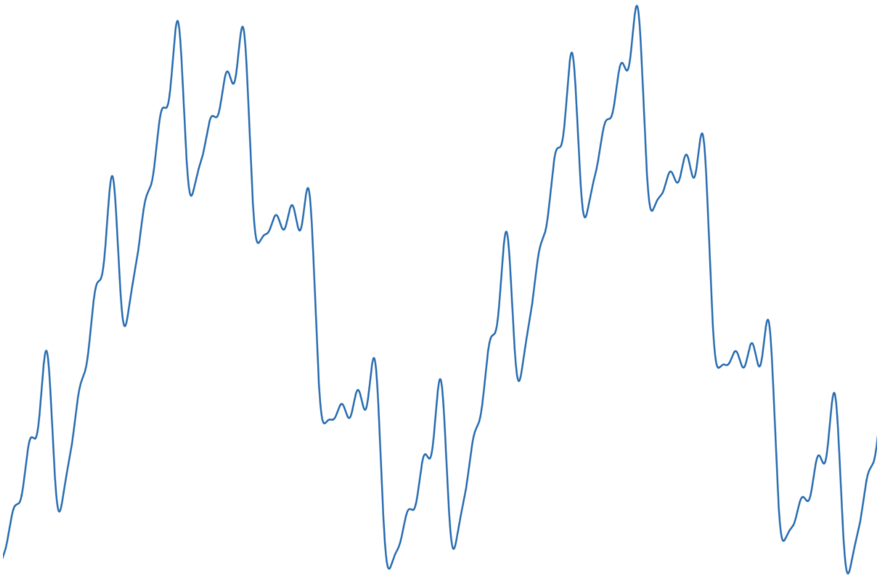


Figure 1.1: A sound wave.

Since figure 1.1 a pure mathematical function, it has infinite precision. At any given point of infinitely precision, the function is guaranteed to have a value. This is called **analog** data, because it is not given in discrete points, it is given as a smooth curve.

In order to convert this into digital data (represented in binary), it must pass through an Analog to Digital converter. This approximates the function into discrete data points. This device takes points on the curve at regular intervals and stores their height. Therefore, after the sound wave mentioned above passes through an ADC, it may look as follows:

[2] In math, periodicity.

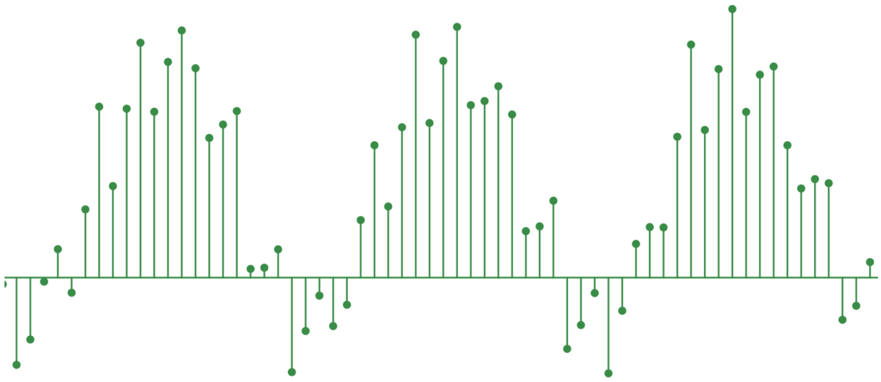


Figure 1.2: The sound wave, but sampled (now discrete)

with points being placed on the curve. Each one of those points' data values can now be stored in binary, like **`1001`**[3].

The **sampling resolution** is how accurate each one of the points are. If the sampling resolution is low (4 bytes), the range of data it can be in binary is low, meaning less accuracy. However, if the sampling resolution is high (44,100 bytes), each data point stored is much more accurate.

The **sampling rate** is how often these data points are taken. The diagram below illustrates this.

(turn over)

[3] in math terms, this is now a discrete function, as there are only some values at which the function is defined

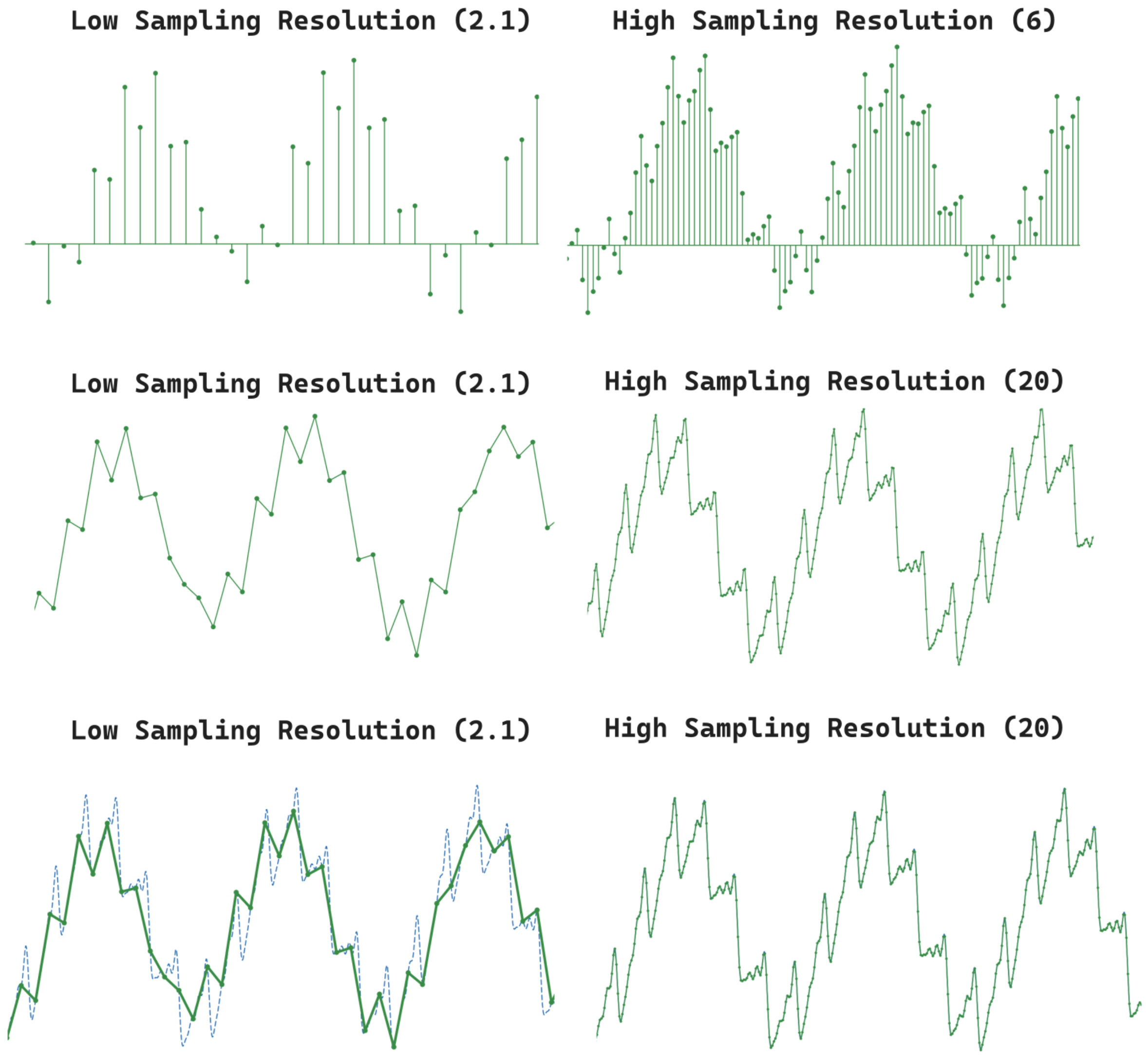


Figure 1.3: Sampling resolution

The first row of figure 1.3 demonstrates the difference between a badly sampled wave and a well sampled wave. The second row are the same points, but they are connected together; showing that the sampling rate affects the accuracy. The third row proves this. The blue line is the original function; the low sampling rate example does not approximate the function well, but on the high sampling rate example, one cannot see the blue line, as it is approximated so well.

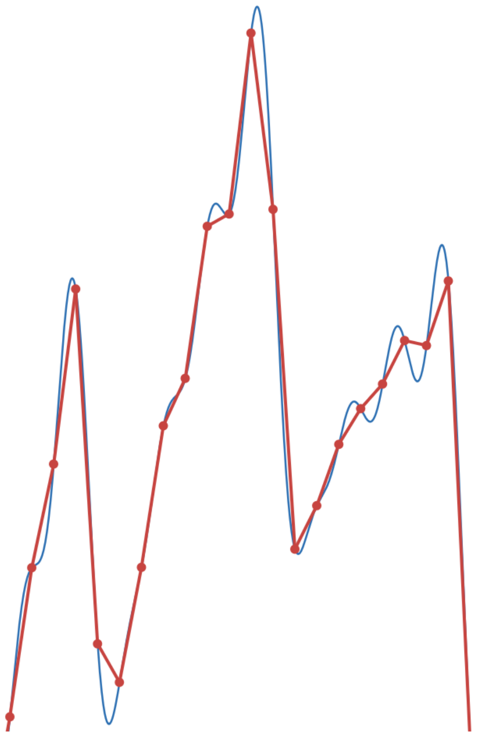


Figure 1.4: A zommed in rendition of the sound wave.

Sampling is never perfect, and can never mimic the nature of a continuous function. Figure 1.4 shows the curve with a sample rate of 8.5.

Finally, the **sample size** is how long the sample is; the length of the wave captured.

What all this means for a computer and human is that an audio file with a low sampling rate and resolution will sound worse (granier, less clear) as the curve is not approximated as well. An audio file with a higher sampling rate and resolution will sound better (clearer).

To calculate the file size, perform (everything except explicitly mentioned is in bytes):

```
file size = sample size * sampling resolution * sampling rate (Hz)
```

NOTE: All the images were generated with this desmos chart[4]*. Please play around with it! It teaches you a lot.*

[4] link: `https://www.desmos.com/calculator/qlduxmdckb`

1.2.3 Images

Images are actually represented as a grid of pixels. Each pixel can hold a color value. Consider this black and white image:

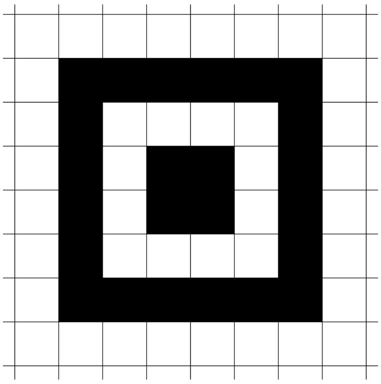


Figure 1.5: An 8x8 black-and-white grid

Figure 1.5 is a black-and white image. Let us suppose that each black pixel is represented as a 1 and each white is represented as a 0. In binary, the image would be represented as follows:

```
0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 0
0 1 0 0 0 0 1 0
0 1 0 1 1 0 1 0
0 1 0 1 1 0 1 0
0 1 0 0 0 0 1 0
0 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0
```

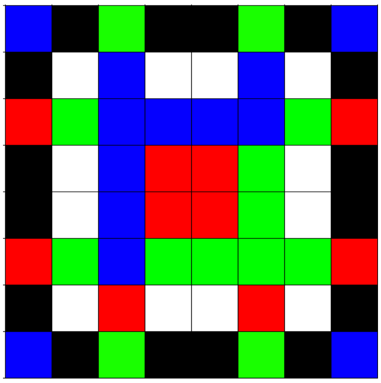


Figure 1.6: An 8x8 color ”image”

Now let us represent 1.6 in binary. We will now assume that black is 0 (the absence of color), 1 is red, 2 is green, 3 is blue and 4 is white. The representation now becomes:

```
3 0 2 0 0 2 0 3
0 4 3 4 4 3 4 0
1 2 3 3 3 3 2 1
0 4 3 1 1 2 4 0
0 4 3 1 1 2 4 0
1 2 3 2 2 2 2 1
0 4 1 4 4 1 4 0
3 0 2 0 0 2 0 3
```

The number of bytes used to store one pixel of data is called the *bit depth*. Since we used 0, 1, 2, 3, 4 to represent our image, we actually need a whole 8 bits or 1 byte to represent each pixel. This means that the bit depth is one byte (which gives us 8 colors). The resolution of figure 1.6 is the size of the image, which is 8x8 pixels, meaning 64 in total. As the resolution increases, more data can be held. To calculate the file size, perform:

```
file size (bytes/bits) = pixels wide * pixels high * bit depth (bytes/bits)
```

For us, it gives us 64 bytes of used space.

A higher resolutions means that images look sharper. Provided is an image from the textbook showing 5 images of the same wheel with varying resolutions.

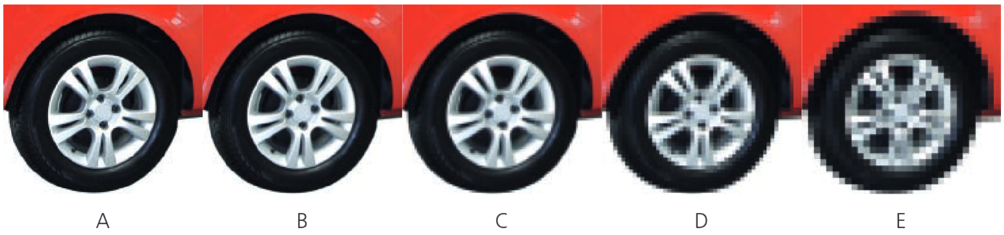


Figure 1.7: From the textbook: Five images of the same car wheel using different resolutions.

Image A is the sharpest-looking in figure 1.7, as it has more pixels than, say, Image E to store the same data. Therefore, it looks better.

1.3 File Size and Compression

1.3.1 File Size Units

The smallest unit of size in Computer Science is a **bit** (binary digit), and it is either a 1 or a 0. A byte is 8 bits, and a nibble is 4 (one hexadecimal digit). The table below introduces to you the rest of the units you need to know.

Number of Bytes	**Unit**
1000	Kilobyte (KB)
1,000,000	Megabyte (MB)
1,000,000,000	Gigabyte (GB)
1,000,000,000,000	Terabyte (TB)
1,000,000,000,000,000	Petabyte (PB)

Table 1.1: SI Base-10 based file size units

Table 1.1 shows the different units if going in increments of 1000. This is based on base-10 and is easier for humans to understand. However, since computers operate in base-2, it makes much more sense for them to go in increments of 1024, like so.

Number of Bytes	**Unit**
1024 (2^{10})	Kibibyte (KiB)
1,048,576 (2^{20})	Mebibyte (MiB)
1,073,741,824 (2^{30})	Gibibyte (GiB)
1,099,511,627,776 (2^{40})	Tebibyte (TiB)
1,125,899,906,842,624 (2^{50})	Pebibyte (PiB)

Table 1.2: SI Base-10 based file size units

This means that data can be represented in (2^{10}), (2^{20}), etc. which is more accurate, as devices measure memory in powers of 2, as they store binary bytes and not decimal 10s. This means that 1 Terabyte is only ~931 Gibibytes.

1.3.2 Compression

There are 2 main types of compression.

Lossy

This means that unnecessary data is eliminated from the file to reduce its size. Examples include:

- **JPEG**, which throws away some data (image quality/sharpness) to reduce file size. This may change the bit depth, etc.
- **MP3/MP4**, which are audio and video formats respectively that may throw away sampling rate, resolution and image quality for a video to save on file size.

Lossless (Run-length Encoding)

This means that no data is discarded to reduce the file size. RLE is a method in the syllabus you must know that collapses repeating sequences of the same data into one chunk of data. Consider the string `aaaaabbbccdddd`. Instead of storing it directly as their ASCII values, we can store

```
05 97 03 98 02 99 04 100
```

as there are 5 ASCII code 97s, 3 98s, 2 99s, and 4 100s, that correspond to 5×a, 3×b, 2×c, and 4×d. One can even compress images with this method. Consider figure 1.5. Its binary representation is shown here (1 is now black, and 0 is now white):

```
1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 1
1 0 1 1 1 1 0 1
1 0 1 0 0 1 0 1
1 0 1 0 0 1 0 1
1 0 1 1 1 1 0 1
1 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1
```

We can represent this in RLE format as:

```
81
11 60 11
11 10 41 10 11
11 10 11 20 11 10 11
11 10 11 20 11 10 11
11 10 41 10 11
11 60 11
81
```

which represents the same data, but now compressed (note that `81` means 8 1's). This also goes for colors, just substitute the different colors for numerical/binary values. Refer to page 37 in the textbook for more examples.

Chapter 2

Data Transmission

2.1 Types of Data Transmission

There exists several modes of data transmission:

- **Serial Transmission:** This is the process of sending data through one single channel, one bit at a time. It is slow, but reliable for small amounts of data. This is used by USB, the kind of connector that is used for flash drives, most modern peripherals like mice and keyboards, and other devices.

  All generations of USB[1] that make use of the USB-A conector use serial exclusively.

- **Parallel Transmission:** This is the process of sending multiple bits at once, thorugh multiple channels concurrently[2]. This is faster, but it requires more computational resources, and can be prone to errors if one lane is congested[3]. The data may also become skewed[4], and may cause unwanted issues.

- **Simplex:** This is when data can only be sent one way.

- **Half Duplex (HD):** This is when one device can send data at one time, like a walkie-talkie. The other device must wait until the data is fully sent; only then can they respond. This method is used in some radio communication protocols.

- **Full Duplex (FD):** This is when data can be sent both ways at the same time. This is like a phone call or video chat, where both people can talk simultaneously. This is used in video conferences, phone chats and real-time web applications.

[1] There are multiple generations of USB, please check this section for more information.

[2] At the same time.

[3] Thought problem: If parallel uses many data channels to send 4 bits, and the second channel is clogged somehow, will the data send, and will there be an error? The solution is trivial and explains this property.

[4] Arrive out-of-order.

2.2 USB

USB is a protocol and interface by which data is transmitted between computers and computer peripherals. it can be seen in many places, like between your computer and a mouse, a cable to connect your phone to the computer, USB microphones, audio devices, hubs, and other possible peripherals.

2.2.1 USB-A

Usually referred to as just USB or USB Type-A apart from the name USB-A, the cross-section of the connector typically looks like a black rectangle with an empty section with a silver coating around it, with 2 holes. Inside the cable, there exists 4 wires:

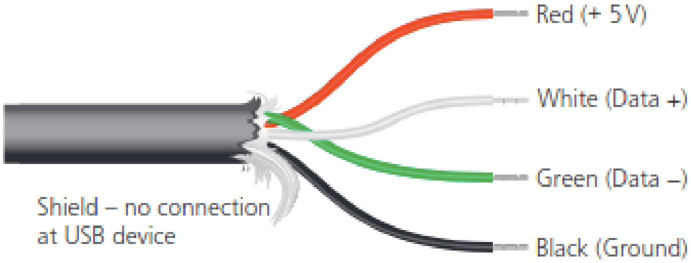


Figure 2.1: The inside of a USB cable, taken from the textbook

Figure 2.1 shows the 4 wires in a USB-A cable:

- A wire carrying **+5 volts**, usually in red,
- **Ground**, or the negative terminal usually in black
- **Green and white**, for serial communication (data- and data+).

And the corresponding port and connector looks like so:

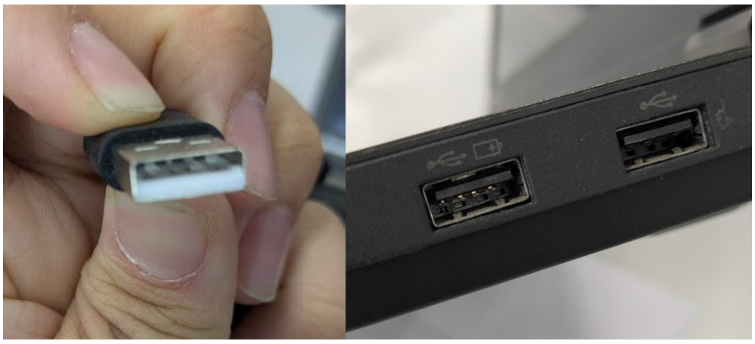


Figure 2.2: The connector and port of USB-A

Since this connector is by far the most common connector used in computers, you may very likely have one on your device. Simply check the sides and or back for an example[5].

[5] Unfortunately, if you possess a newer Apple computer, Apple took away the USB-A ports to make

2.2.2 Benefits and Drawbacks of USB-C

Note that one only needs to know a few advantages and disadvantages; some are quite specific.

Benefits include:

- USB-A is very common; seen on most computers and devices
- The connector itself is more durable due to its thickness
- Since the connector only fits in one way, incorrect connections cannot be made
- The USB protocol makes use of error correction[6][7]
- If one needs more ports, they can choose to connect a *USB Hub* which splits one USB port into many, similar to an Ethernet switch)

Drawbacks include:

- Standard USB-A only supports cables up to 5 meters, requiring extensions for cables to go further.
- The connector is one way, making it difficult to insert without looking at the port.
- Very early USB standards, like USB1 may not be supported by the latest computers.
- By far the most common USB-A standard, USB2, found in the vast majority of devices from 2000 till now, only supports 480Mbits/s, meaning that ethernet adapters and other applications that may require higher speed data transfer cannot be done.

2.2.3 USB-C

USB-C, USB Type-C, or just Type-C is the most common USB connector on newer computing devices; all Apple devices after ~2016 have USB-C, and computers between around ~2016 til ~2020 have *only* USB-C ports.

This connection is called USB, however the connector looks quite different from standard USB-A:

As seen in Figure 2.3, USB-C takes on more of a rounded shape, smaller than USB-A and is a *Symmetrical Connector*, meaning it can be inserted both ways.

you purchase adapters. Look for another device or an adapter for an example in that case

[6] As it should be.

[7] If any errors are found, the data is to be re-transmitted.

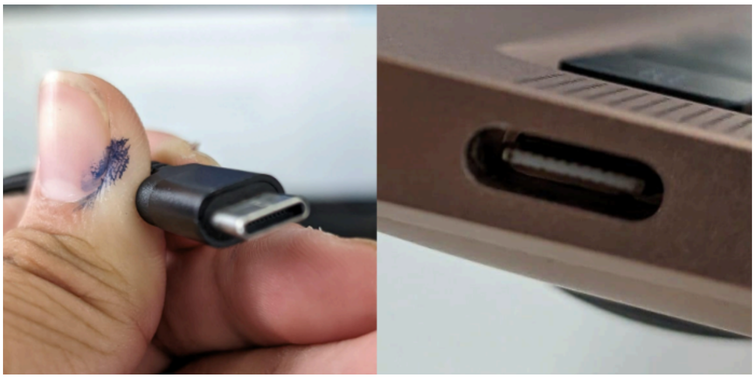


Figure 2.3: The connector and port of USB-C

Quirks

- USB-C can carry *A video signal*[8], meaning external monitors and TVs can be connected.
- USB-C supports a standard called USB-C PD (Power Delivery). The textbook[9] states USB-C can deliver a 20 volt 5 amp (therefore 100W) power signal to change devices at a high wattage.
- USB-C supports extremely high data transfer rates. The textbook states that it can deliver up to 40 gigabits to second, however the latest standard[10] supports up to 120 gigabits per second asymmetrically and 80Gbit/s symmetrically.
- USB-C is fully backwards compatible[11] with USB-A, one just needs a simple adapter with the correct lanes connected to make use of this feature.
- the USB-C connector is also used by Thunderbolt, which is a way to connect PCIe Express devices through USB[12]. Professional audio devices or graphics cards can be connected through this port.

2.2.4 Benefits and Drawbacks of USB-C

Note that one only needs to know a few advantages and disadvantages; some are quite specific.

Benefits include:

- USB-C can carry more types of signals, such as a video signal, power delivery, and Thunderbolt.
- USB-C can deliver much faster speeds for high volume data transfers, like between film cameras and computers.

[8] Through the DisplayPort standard.

[9] Since USB-C standards change relatively rapidly, the textbook version should be beter for the exams as that is in the syllabus

[10] USB4 Gen 2

[11] This generally means that a newer thing fully supports all the features of the older thing

[12] This is very niche and may not come up in tests, but it is good to know

- It can be inserted into a port both ways; it makes connecting devices in less accessible locations more convenient.
- It supports far more ports and devices, like DisplayPort, through adapters.

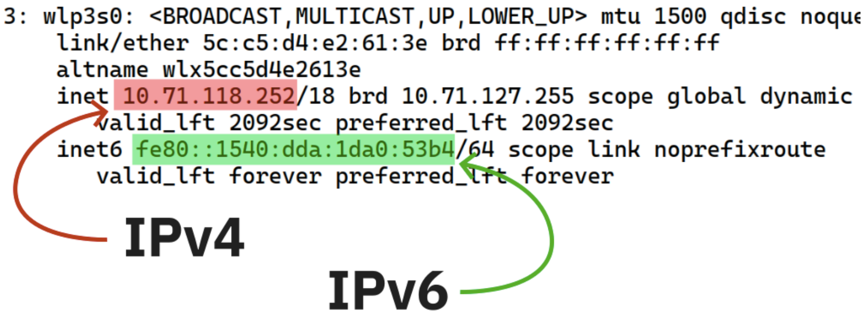
Drawbacks include:

- It is relatively rare on older devices; only around 2016-2018 did computer manufacturers put USB-C ports on their devices. A lot of them, even to this day do not support data transfer through the port; only power delivery.
- Since USB-C has so many standards (Video, USB, PCI through Thunderbolt, different levels of Power Delivery and even no-data cables), it is confusing as some ports are incapable of delivering thunderbolt, while others are incapable of delivering video, etc.
- USB-A has colors (White/Black and Blue for USB2 and 3 respectively) to determine the generation; and for USB type B the connectors are physically different to determine the generation. USB-C does not have any way of differentiating, so one must memorize the capabilities of their USB-C device to know the supported generation.

2.3 IP and MAC addresses

2.3.1 IP addresses

IP stands for the Internet Protocol, and IP addresses are like street addresses; in the sense that they represent locations on a network. There are 2 main types, IPv4 and IPv6 addresses.



```
3: wlp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noque
    link/ether 5c:c5:d4:e2:61:3e brd ff:ff:ff:ff:ff:ff
    altname wlx5cc5d4e2613e
    inet 10.71.118.252/18 brd 10.71.127.255 scope global dynamic
       valid_lft 2092sec preferred_lft 2092sec
    inet6 fe80::1540:dda:1da0:53b4/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Figure 2.4: Output from the Linux **`ip`** *utility, showing IPv4 and V6 addresses*

IPv4 Addresses

Are **32-bit values** represented as 4 3-digit numbers with dots between them, like `192.168.0.1` with each value ranging between 0-255. There are 2 tyes of IP addresses, *Public* and *Private* IP addresses.

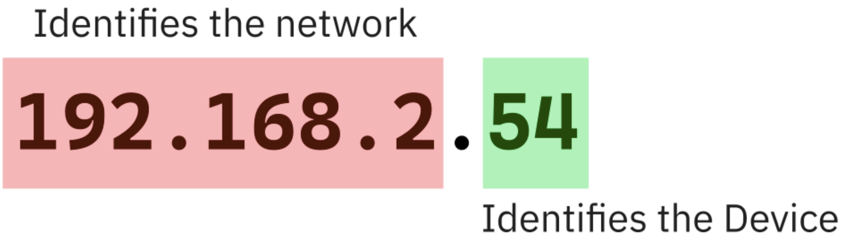


Figure 2.5: The two components of an IPv4 address

Figure 2.5 shows the two main components of IPv4 addresses; the first component determines the network itself, and the second component determines the device on the network uniquely.

In terms of private and public networks it will be covered more in **Chapter 4** and later sections about NATs.

Public IPv4 Addresses

These IP addresses denote the location of your home router or other routers on the public internet. All web servers, like the one in our school or your personal router has a public IP address accessible by everybody[13].

The only rule they must follow is that their addresses cannot be in the range of private IP addresses.

Private IPv4 Addresses

Private IP addresses are like public IP addresses, but instead of it being publicly accessible to everybody on the internet, they are only accessible to users connected to a private network, like a phone hotspot or the network created by your router. Every device on, say, your home network like your phone or your mom's work laptop have a private IP. These addresses are then mapped to public IP addresses through NATs (Network Address Translators); more details in section 2.3.5.

In the IPv4 address range, addresses are reserved for private networks to use, and they are:[14]

Class	IP Range	Most typically seen in
A	`10.0.0.0` to `10.255.255.255`	Large networks
B	`172.16.0.0` to `172.31.255.255`	Medium-sized networks
C	`192.168.0.0` to `192.168.255.255`	Small networks

Figure 2.6: Table of private IP classes

[13] although they do not necessarily respond with data all the time

[14] Taken from https://www.geeksforgeeks.org/private-ip-addresses-in-networking/

Note that you do not need to memorize the class letters nor the exact ranges, you just need to identify them, especially the most common one, class C which is between `192.168.0.0` to `192.168.255.255`.

These private IPs belong in a local network, where devices in the same approximate geographical location are connected to a router.

2.3.2 IPv4 Address Assignment

When you connect to the school network or your home WiFi network, you need to get an IP address in order for your device to communicate with the router, to transmit data to and from websites like Google. Without it, your device would be useless.

Static IP assignment

You must ask the router for an IP address; and there are 2 main methods. This one is called static IP assignment; this means that the IP address you have on the network does **not** change, and *your device chooses the IP address it has*. The IP you assign to yourself is now assigned to you for an indefinite[15] time.

Pros include:

- It is good for servers, because your location on the network does not change. Computers on the network will be able to find you easier.
- The connection process is much faster, as you don't have to ask for one.
- You get faster upload/download speeds (only if you use a static NAT![16])

Cons include:

- on a large private network, your location on the network is more easily identifiable, as your IP doesn't change.
- you cannot tell if the IP you assigned to yourself is available or not.[17]
- it is expensive to maintain, as the device at the address must constantly be on to be available.

Dynamic IP assignment

This is like static IP assignment, as you are receiving an IP address to identify yourself. But here, you use a protocol called DHCP (Dynamic Host Configuration Protocol)

[15] not infinite, just an unknown amount of time

[16] You will learn about NATs in a later section. Here, it means that a private address is directly mapped to the public address of your router.

[17] You can always ping the IP address to see if a device at that IP responds. If there is no response, there is no device at the IP.

to talk to the router nicely to ask for an available IP address. Every time you connect to the network, your device talks to the router for an IP. Sometimes it is different, and sometimes it is the same as before.

This is by far the most common method.

Pros include:

- On large public networks, it tends to change more often, making it more secure.
- It is a lot more convenient, as the device does not have to set their own IP before connecting.

Cons include:

- If you're connected to a video call or voice call through VoIP[18], if your IP address changes, it may disconnect the call[19].
- If your device is old and does not support the DHCP protocol, you cannot use it. In short, it may not support all devices.
- Connection time takes longer, you must send a `DHCPLEASE` message to the router for the IP, the router must calculate an IP, and the router must send a `DHCPBACK` with your IP. This typically takes 5 seconds but may take up to 25 seconds.

2.3.3 MAC addresses

MAC addresses are 48 bit numbers that identify your device uniquely on any given network. They are typically represented as 3 hexadecimal numbers:

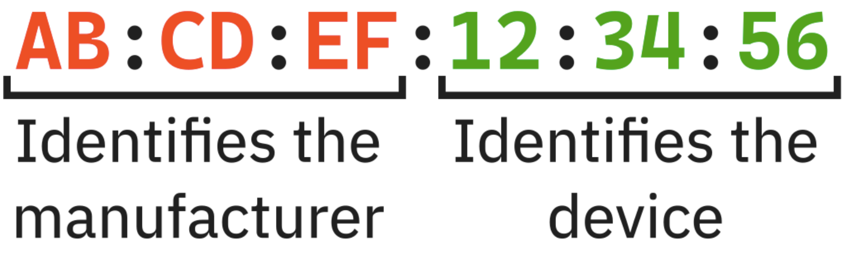
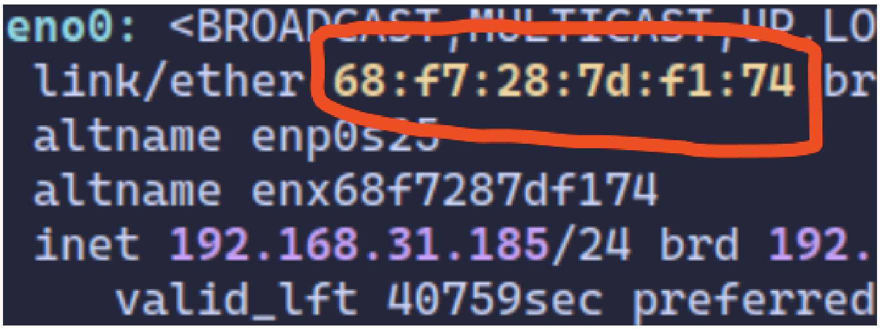


Figure 2.7: A breakdown of the portions of a MAC address.

The first 3 hexadecimal bytes (2 digits) identify the maker of the device[20], like Apple[21]. The last 3 digits uniquely identifies the device itself.

Figure 2.8 shows the MAC address of one of my computers. The first 3 components identify the manufacturer, and the last 3 uniquely identifiers my device.[22]

[18] Basically making phone or video calls through a network and not cell towers.
[19] It is not necessarily true, as your IP is not jumbled at regular intervals.
[20] Technically the NIC or Network Interface Card itself
[21] Not necessarily, again, it identifies the maker of the NIC.
[22] In this example, I have changed the default MAC address to a randomized one, so the manufacturer may be invalid if you try to look it up.



```
eno0: <BROADCAST,MULTICAST,UP,LO
 link/ether 68:f7:28:7d:f1:74 br
 altname enp0s25
 altname enx68f7287df174
 inet 192.168.31.185/24 brd 192.
    valid_lft 40759sec preferred
```

Figure 2.8: The MAC address of a computer, running Arch Linux.

Universally Administered

These addresses are by far the most common; they are the addresses assigned by manufacturers and universally identify the device. They are used to ensure the unique identification of the device throughout the globe, and avoids conflicts with other MAC addresses.

Locally Administered

These are a less commonly talked about form of MAC addresses, these are assigned to the computer by network administrators. These addresses have no guarantee of being completely unique from computer-to-computer, and are used in specific cases where addresses may conflict with each other.

Some organizations, such as mainframe or cluster computers, require MAC addresses to be in a certain format. LAAs allow for the MAC address to be changed, therefore allowing one to change the format. It may also be used to bypass a firewall, as some require the MAC addresses to be in a certain format.

Extra: LAA addresses also have a property where the 7th bit of the 1st byte of the MAC address is set to 1. UAA addresses have that bit set to 0.

2.3.4 Differences between IP and MAC addresses

MAC addresses	**IP addresses**
Identifies unique information about a device on a network	Identifies the location of something on a network
Unique for the device on the network	Only unique if it is a public IP. Private IPs are only unique to that network, and addresses across many private IPs may be the same, as they are powered by different routers.

48 bits	32 bits for IPv4, or 128 bits for IPv6
Can be locally or universally administered	Can be static or dynamic
Assigned by the manufacturer and is baked into the NIC	Static IPs are assigned by the connecting device. Dynamic IPs are assigned by the DHCP server, usually built into the router.

2.3.5 NATs (Network Address Translation)

By now you should be familiar with the concept of private versus public IPs. Private IP addresses only exist on internal private networks, and are separate from all public IPs. How do devices on private IPs then transmit data to websites on public networks?

NATs helps one accomplish this. They allow private IP addresses and public IP addresses to be tempoarily linked together to create a channel whereby data can flow. These are typically in your home router to map devices connected to it to the router's public IP address.

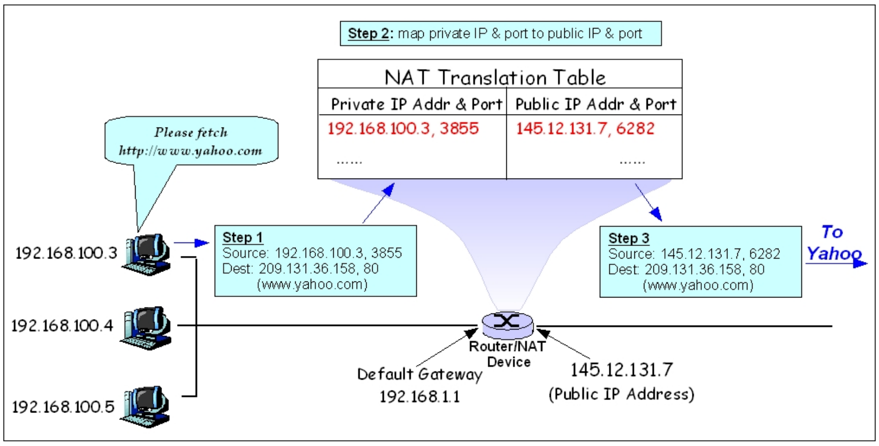


Figure 2.9: A diagram explaining how network addresses are mapped to public IPs through the NAT. Taken from Wikipedia.

As seen in figure 2.9, computers on the home network can request for websites (in this case, Yahoo, as the diagram is presumably very old), and the NAT in the router can add your device to a tempoary table that shows where traffic from one private IP should go, and vice versa.[23]

[23] Do not worry about the port; this allows 65535 channels of data to be open on one IP address, as one single IP is quite limiting.

2.4 Packet Switching

Sending data across large distances is used all the time, like in video chats, voice chats, uploading homework or simply loading HTML[24]. Typically, sending data like this is done through a stream of **data packets**, otherwise known as just packets.

A packet looks like the following:

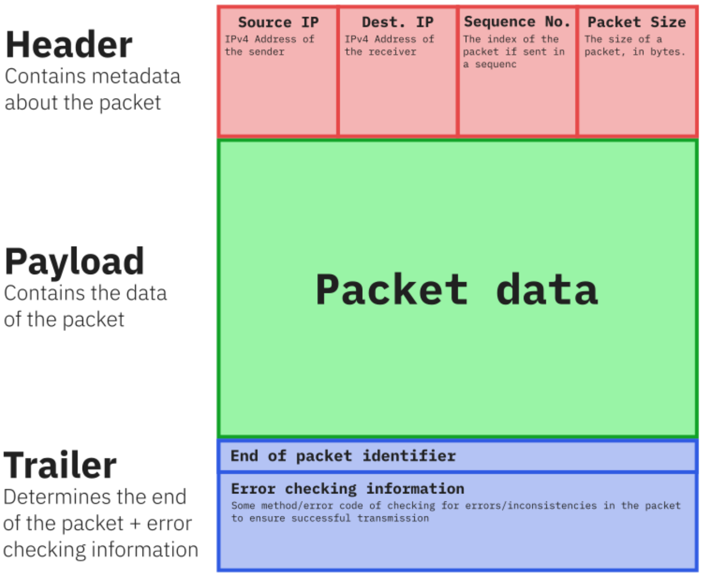

Header
Contains metadata about the packet
Source IP
IPv4 Address of the sender
Dest. IP
IPv4 Address of the receiver
Sequence No.
The index of the packet if sent in a sequenc
Packet Size
The size of a packet, in bytes.
Payload
Contains the data of the packet
Packet data
Trailer
Determines the end of the packet + error checking information
End of packet identifier
Error checking information
Some method/error code of checking for errors/inconsistencies in the packet to ensure successful transmission

Figure 2.10: The components of a data packet.

2.4.1 Components of a packet

Here are they:

- The Header:
  - The IP[25] of the sender
  - The IP of the receiver
  - The sequence number; If a lot of data must be sent throughout multiple packets, the *sequence number* makes sure that the packets' payloads are re-assembled in the correct order.
  - The packet size, in order to make sure the packet received is of the correct size.
- The payload, which consists of the binary data to be transmitted via the buffer.
- The trailer, which consists of:

[24] This markup language, commonly mistaken for a programming language is what lays out websites; it dictates the structure of the site, like the bricks for a house.

[25] IP Address; in this context, always IPv4 unless if specified

  - Some way of identifying the end of the packet. This is typically some special value like a null terminator[26]. The algorithm can then scan the data until it hit that character to extract the payload.
  - An error checking method. CRCs are used to check this.

2.4.2 Packet Switching

Packet switching is the next step after slicing up data into packets. This is done because sending data across large distances all at once is prone to failure (what happens if a slight portion of the data is corrupted? the data must be sent all over again, taking time).

This technique involves sending packets onto multiple paths that reach the same destination. They begin by being sent out onto the network; they travel seperately but at the same time and when all the packets arrive at the destination, they are put back together via the sequence number. The role of the router on each node is to determine the next router the packet should go to.

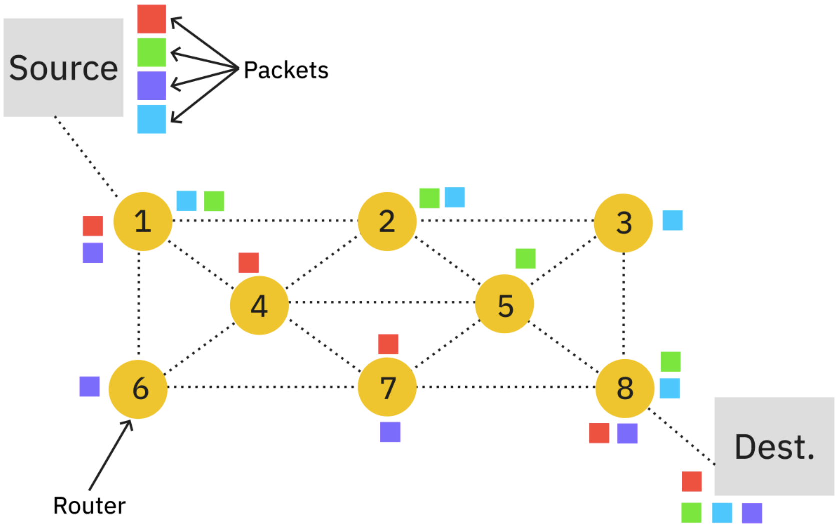


Figure 2.11: A diagram to illustrate packet switching.

Each yellow circle can be interpreted as an intersection on a road. The general term for these are nodes, but they are technically just routers.

Figure 2.11 has 4 packets that must be sent. The key points are:

- The path between the source and the destination are made up of the yellow circles; the nodes.
- Each packet takes a different path, i.e. the red packet takes 1-4-7-8, but the blue packet takes 1-2-3-8, etc.

[26] In programming, this is referred to as a *sentinel value*, which just means a value that carries some special meaning. An example is like in the C programming language, to tell the end of a string, a sentinel value of 0 is used to denote the string finished.

- The router at each node will decide the next router the packet should go to. The shortest *available* route is always taken but not the *absolute shortest* route.
- When the packets arrive, they will be put back in order according to the sequence number (not depicted).

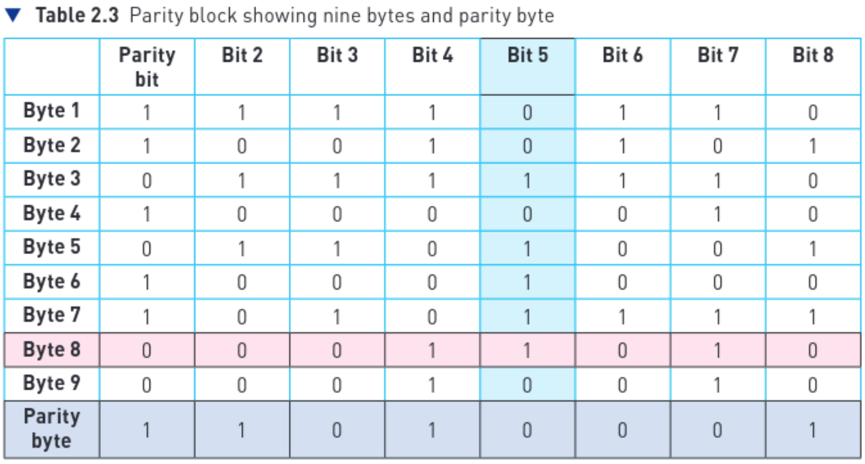
2.5 Error Correction

To make sure that data sent from a device is received on the other side correctly, without errors, error correction must be done to make sure the data is valid. Here are some methods:

- **Parity Checks:** This adds an extra bit (0 or 1) to a piece of data (usually a byte) based on the number of 1s already present. The number of 1s are totalled. Depending on if even or odd parity are agreed upon from the sender and receiver, if the number of 1s is even, the parity bit is 1, for odd parity the parity bit is 1 if the number of 1s is odd. If the receiver's calculated digit is the same as the digit sent, it indicates no error. However, this method can only detect single-bit errors and not multiple-bit errors.
- **Cyclic Redundancy Checks (Checksums, CRCs):** The data bits are added together, processed in a certain predetermined way, and the result (checksum) is appended to the data packet. The receiving device recalculates the checksum on the received data and compares it to the received checksum. If they match, the data is assumed to be error-free. This method is more robust than parity checks.
- **Echo Checks:** The sender sends the data packet and then waits for an exact copy of the packet back from the receiving device. This confirms successful data transfer but introduces a delay due to the back-and-forth communication.

2.5.1 Parity Blocks

These are like 2D parity checks. Each byte of data sent will have one bit used for parity, when the digit is calculated horizontally. Then, a whole other byte of data is also sent; with just parity bytes from parity checks from each vertical column.



▼ **Table 2.3** Parity block showing nine bytes and parity byte

	Parity bit	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7	Bit 8
Byte 1	1	1	1	1	0	1	1	0
Byte 2	1	0	0	1	0	1	0	1
Byte 3	0	1	1	1	1	1	1	0
Byte 4	1	0	0	0	0	0	1	0
Byte 5	0	1	1	0	1	0	0	1
Byte 6	1	0	0	0	1	0	0	0
Byte 7	1	0	1	0	1	1	1	1
Byte 8	0	0	0	1	1	0	1	0
Byte 9	0	0	0	1	0	0	1	0
Parity byte	1	1	0	1	0	0	0	1

Figure 2.12: From the textbook. "[The table] shows how the data arrived at the receiving end. It is now necessary to check the parity of each byte horizontally (bytes 1 to 9) and vertically (columns 1 to 8). Each row and column where the parity has changed from even to odd should be flagged:"

Figure 2.12 from the textbook shows this. An extra byte of information is used as parity bits for each column, and each row has one data bit dedicated to storing horizontal parity data. This allows exact errors to be pinpointed (bit flips).

2.5.2 Check Digits

Check digits are another form of error correction; usually involving checking for errors in numbered codes, such as barcodes.

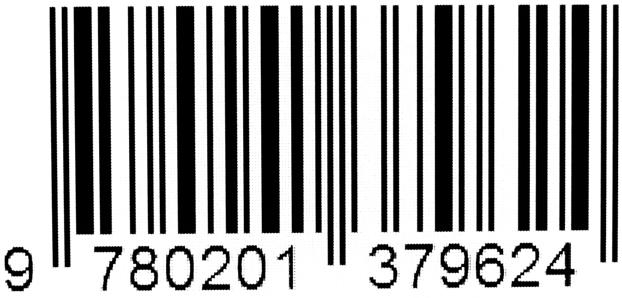


Figure 2.13: Barcodes consist of black and white bars that represent decimal numbers; written at the bottom.

typically, barcode readers that read these numbers may encouter stains on the physical code and such. Therefore, error correction is needed to make sure the data is not damaged to the point the barcode cannot be read.

ISBN-13

To calculate:

1. Add all odd digits
2. Add all even digits, and multipy the result by 3
3. Add the results from steps 1 and 2, and divide the result by 10
4. If the remainder is 0, use 0. Otherwise, subtract your result from step 3 from 10. This is your check digit.[27]

To verify:

1. Add all odd digits
2. Add all even digits, and multiply the result by 3
3. Add the results from steps 1 and 2, and divide the result by 10
4. If the remainder is 0, there is no error.

Modulo-11

To calculate:

1. You must calculate a *Max Weighting*. Take the length of your number data, and add 1 to get your weighting.
2. Take your first digit, and assign it to your weighting. Assign the next weighting to the max weighting minus one. Assign the next weighting after that to the max weighting minus 2, until you run out of digits.
3. Multiply each digit by the weight.
4. Add all the products from step 3 together.
5. Divide the result from step 4 by 11, and note down the remainder.
6. Subtract your remainder from 11.[28]. This is the check digit.

To verify:

1. perform steps 1-5.
2. If the remainder is 0, the check digit is correct.

[27] As an example, if you got 6 for step 3, your check digit is $10-6$ or 4.

[28] As an example, if you got 7 from step 5, your remainder is 4

2.5.3 Automatic Repeat Requests

In short, for small amounts of data, if the data received by the receiver is invalid, it automatically asks the sender to re-send the data.

This is a system used in networking to make sure the data received by the receiver over the network is valid. Before the data is sent, both sides agree on a timeout. The timeout determines the max time the data can take to be sent. The data is sent, and the timeout (like a stopwatch) begins. If the data is not received before the timeout is over, the data must be re-sent.

Acknowledgements are used for the sender and receiver to communicate about where the data is and the integrity of the data. If the data is received correctly, a *positive acknowledgement* is sent back to the sender. If the data is erroneous, a *negative acknowledgement* is sent. If there is no acknowledgement sent, the data is assumed to be lost in transit, and the dat must be resent.

Example

- Tom's computer wants to send Jerry's computer a data packet. Tom's computer establishes a timeout of 10 seconds and sends out the packet.
- If Jerry does not send an acknowledgement in time, the communication is cancelled.
- If Jerry does get the data, a positive or negative acknowledgement is sent. The process is repeated until there's a positive acknowledgement from Jerry.

2.6 Encryption

Encryption is a process by which data is obfuscated in a certain way that only the sender and receiver can see the data in question, but onlookers along the way cannot. The process of obfuscating is completely reversible given a consensus on which method to use. Two of them exist. In what matters to us, keys are used to achieve this process.

Symmetric Encryption

This is when one key is used to encrypt and decrypt the data. The key is applied to the data to encrypt it, and applied to the data to decrypt it. Although the approach is simple, if a hacker gets hold of the key, all data encrypted with the key is compromized.

First, they go to a secure location and exchange their keys. Then, they can exchange data:

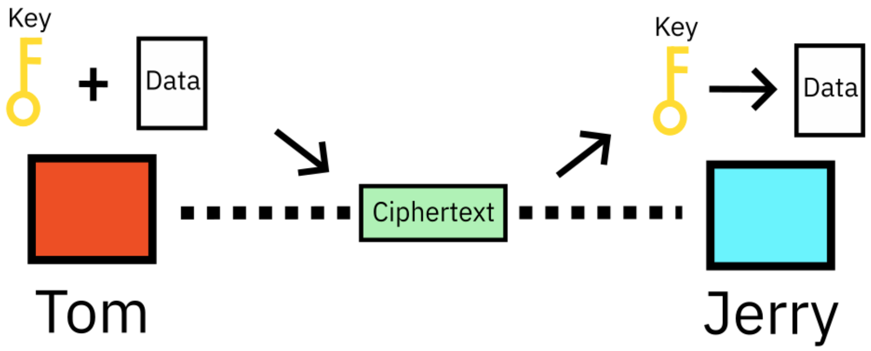


Figure 2.14: Tom and Jerry exchanges a message with the same key.

The same key is used both for encryption and decryption.

Asymmetric Encryption

This is when both the sender and the receiver have a pair of keys. One is used to decrypt data (the private key), and the other one is used to encrypt data (the public key). The public key is kept public, as anybody should be able to encrypt data for you. But only yourself can decrypt the data people garbled up for you. This means that only you can decrypt the data, which is the intent. If you want to encrypt data for someone else, you use their public key, and the other person has the private key to decrypt it.

Let's say Tom and Jerry now wants to make use of asymmetric encryption. To begin, they need each other's keys to encrypt data for one another. The process is called a key exchange (self explanatory).

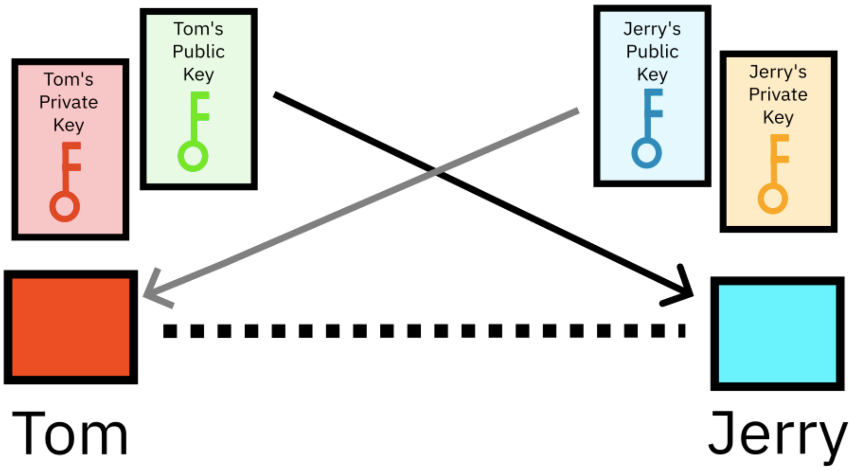


Figure 2.15: Tom and Jerry exchanges keys.

Then, if Tom wants to encrypt data for Jerry, Tom uses Jerry's public key, and Jerry decripts it with his private key.

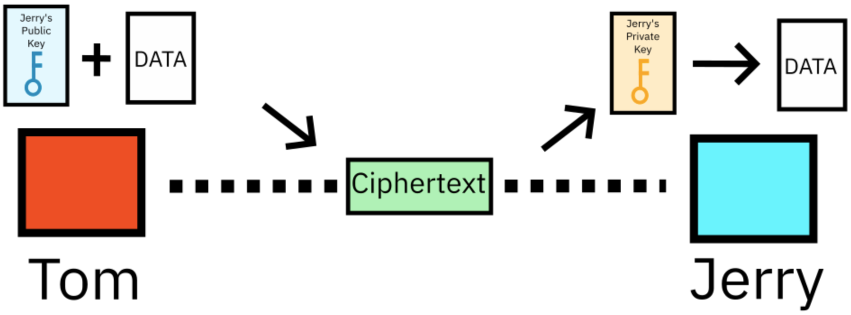


Figure 2.16: Tom sends data to Jerry.

And vice-versa.

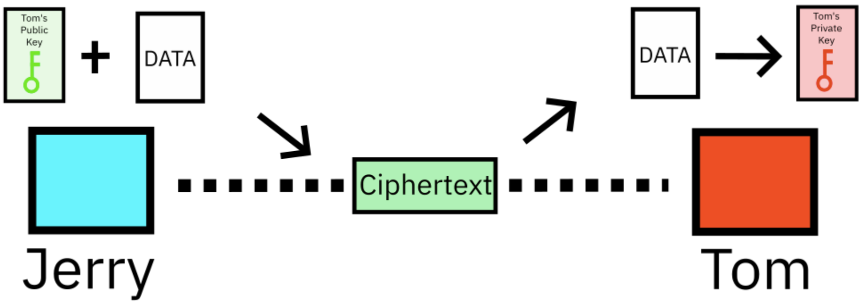


Figure 2.17: Jerry sends data to Tom.

Chapter 3

Hardware

3.1 Data Storage

Computers need to store data in order to function, like code, or data meaningful to you, like documents and images. Storage devices accomplish this. There are 2 main categories of storage devices, which will be covered here.

3.1.1 Primary vs Secondary Storage

The key difference is that **Primary Storage can be directly addressed by the CPU**, whereas **Secondary Storage must be accessed via the I/O controller** (see figure 3.10.) A more comprehensive table is given:

Primary Storage	**Secondary Storage**
Primary storage can be directly addressed by the CPU, like ROM and RAM.	Secondary storage cannot be addressed directly. Instead, the I/O controller must fetch the data for the CPU, and only then is the data visiblnne to the CPU.
Primary storage does not have to be loaded somewhere before it can be put into the CPU.	The data from secondary storage *must* be loaded into RAM first, then is it accessible to the CPU.

3.1.2 RAM

RAM or Random Access Memory; otherwise referred to as just Memory, stores *volatile data* that does not have to be on your hard drive. It only stores data that the computer uses when it is on. Examples of data that goes into memory include temporary values that programs must store; from simple integers to browser tabs. RAM is **Primary Storage.**

It has a property where it is directly addressable by the CPU. Check section 3.2.1

for details on exactly what that is, but it is essentially the brains of your computer. This makes RAM very fast.

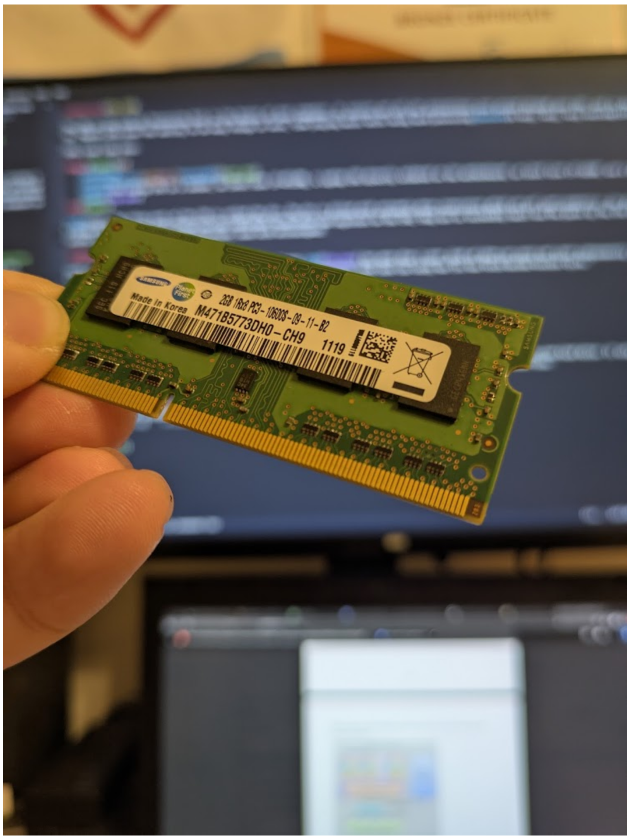


Figure 3.1: What a stick of LPDDR3 RAM (DRAM) looks like.

Some important information include:

- RAM can be freely written to, or read from. The data on RAM can be changed by the user, but not directly; through programs. As an example, opening a browser tab puts data into RAM, but you will not notice physical differences.
- RAM is ***volatile***. All data stored in it disappears when power to the RAM is lost.
- RAM is critical to your computer's function; as all code and data used by code is stored in RAM.
- Increasing the amount of RAM will boost the speed of your system, as the computer is able to store more of this temporary data at once in a fast location.

There are 2 main kinds of RAM, SRAM (S for static), and DRAM (D for dynamic).

Dynamic RAM

DRAM chips (the black squares that you see on the image) consist of transistors and capacitors. It is by far the most common type of RAM in all computers. The access time for DRAM is ~60 milliseconds.

They consist of:

- Capacitors hold a bit of information (0, or 1)
- Transistors act as switches, which allows the chip's control circuitry to read or write to the capacitors.

This must be constantly refreshed, as the capacitors cannot hold data for very long.

Benefits of DRAM include:

- Much cheaper to make than SRAM
- Consume less power on average
- They hold a larger total capacity (typically)
- DRAM in computers are upgradable in many cases. Modern laptops do not allow you to do so for the most part, but all desktops and some older laptops allow you to remove and exchange DRAM easily for upgrades/repairs.

Drawbacks of DRAM include:

- It needs constant refreshing to keep the capacitors charged
- It is slower than SRAM, by more than 2 times. This limits its applications.

Static RAM

SRAM chips are made of flip-flops that hold a constant one bit value. They do not need to be constantly refreshed, therefore. SRAM is used when speed is required, such as the CPU's cache. The access time is ~25 milliseconds.

Benefits of SRAM include:

- Much faster than DRAM due to less latency[1]
- Consumes less power
- Does not need to be constantly refreshed

Drawbacks of SRAM include:

- Much more expensive
- Very low capacity
- More complex and typically incompatible circuitry must be used to access the RAM.

[1] Delay.

Differences between DRAM and SRAM

Dynamic RAM (DRAM)	**Static RAM (SRAM)**
DRAM is slower	SRAM is faster
Uses transistors to control the flow of electrons, and capacitors store the binary 1s and 0s	Uses flip-flop circuits
Must be constantly refreshed to make sure the capacitors have charge	Does not need refreshing
Cheaper to make	Much more expensive to make
In a computer, the main memory uses it	The CPU's cache uses it
Less overall power consumption	More overall power consumption
Higher capacity	Lower capacity

3.1.3 ROM

ROM is like RAM, however, it stands for *read-only memory.* Like RAM (see section 3.1.2), it stores data that is quickly accessible to the computer; i.e. directly addressible by the CPU. However, the *key difference* is that **ROM is NOT erased when the computer powers off.** This is part of the reason why ROM is purely read-only. ROM cannot be erased.[2]

The main use of ROM in computers is to store the BIOS/UEFI. This is code that the CPU executes immediately when the computer starts up. It does critical checks to make sure all your peripherals are working (like your keyboard), applies specific security patches to your CPU that the CPU maker sees fit, and loads the OS (see section . Increasing the size of ROM does nothing in most systems. ROM is **Primary Storage.**

3.1.4 Virtual Memory

Virtual Memory is quite a complicated topic. In essence, it is memory made by combining physical RAM and space on your hard drive (HDD or SSD) called the swap space, swap area or swap file to create chunks of memory for each program to use independently.

A diagram illustrates this:

[2] Nowadays, computers do not use pure ROM chips anymore, as...they cannot be erased. Instead, they use EEPROM, or electrically-erasable and programmable ROM, which can be erased, but not as easily as RAM. You need a special device to erase an EEPROM, but it is doable. This means that manufacturing errors when making ROM can just be overwritten.

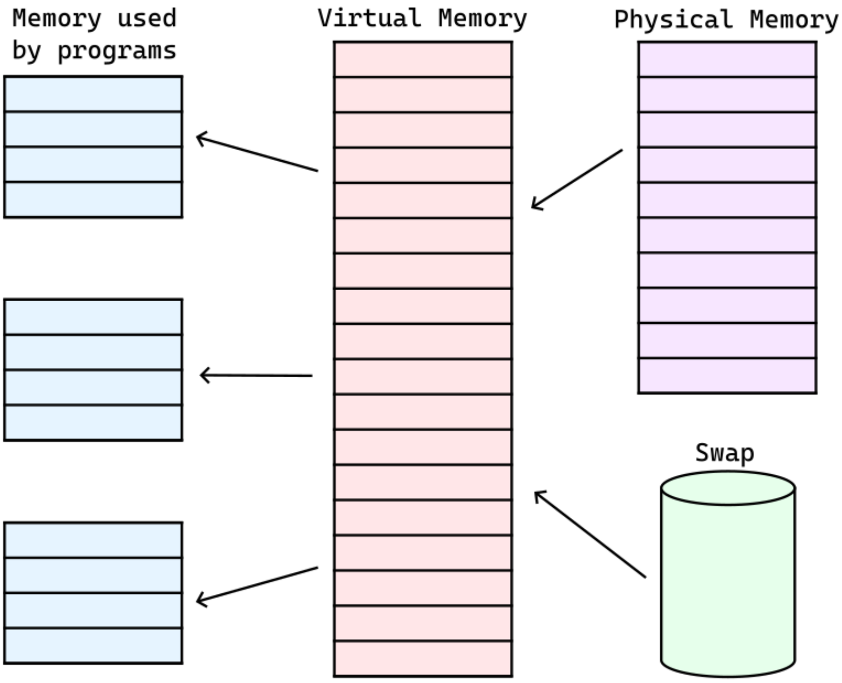


Figure 3.2: A diagram explaining virtual memory.

For a better explanation, head to section 4.1.1.

Each program gets its own slice of the virtual memory, so that memory regions do not accidentally overlap, causing unintended errors. This is also partly for security, so that malicious programs cannot read, say, the passwords in your browser. It is important to note that slices of virtual memory are not all of uniform size. Each slice's location in actual physical memory and in swap may be different.

The process of putting something explicitly into swap memory is called **paging in**, given one of those chunks in figure 3.2 is called a page. Moving something out is called **paging out**. Each slice of memory in the diagram (each block) is called a **page**. They are all equally sized.

3.1.5 HDDs (Hard disk drives)

HDDs, Hard Disk Drives or colloquially referred to as spin-drives or just hard drives are **large secondary storage** devices that has a metal casing and a spinning platter/disc inside of it. The disc inside is magnetized, and the direction at which the tiny magnets points represent binary 0s and 1s

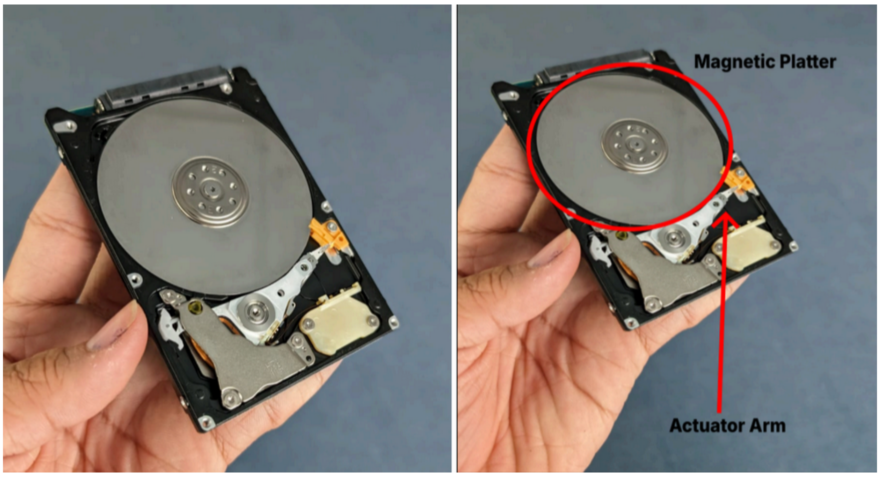


Figure 3.3: The inside of an HDD, with annotations explaining the parts.

Here are the important components:

- The magnetic platter is the medium the data is stored on.
- The actuator arm is what actually reads the data off of the medium.

These devices typically hold larger amounts of data, such as a lot of films, games, and are used in high-capacity storage devices and security camera backup systems. They can also be used to store the operating system (see section 4.1). Unlike optical media (see 3.1.8) hard drive is both the storage device and the medium.

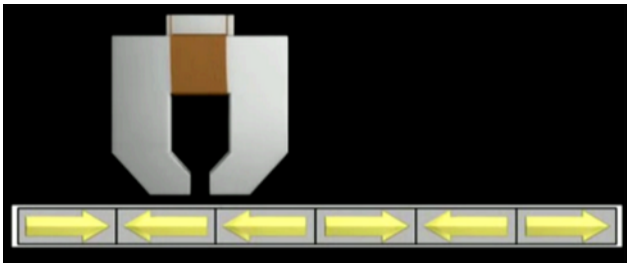


Figure 3.4: What the magnets inside an HDD looks like.

Pros

- They can hold a lot more data than SSDs, as the platters can be manufactured to have more density, along with being able to stack many on top of each other.
- They are a lot more affordable (as of 2025, the price gap has decreased by a large margin but still holds true).

Cons

- They have a lot of read-write latency (delays when reading or writing data), making one's computer a lot slower[3]. In short, they are slow.
- They are more prone to data rot, which is the corrosion of the platter in the HDD over time.
- They have more mechanical parts, making more failure points and making them a lot more fragile.

3.1.6 SSDs (Solid state drives)

SSDs or solid state drives, are like HDDs; they can store large amounts of data and are **secondary storage** devices. Instead of using magnetic discs as the medium, they use NAND flash chips, powered by transistors and NAND gates (see **??**.)

They are typically used in place of HDDs where fast storage is needed; like the startup disk/boot device used to store the operating system, and in mission-critical environments where fast data access is absolutely required. Using these in place of an HDD makes the computer feel snappier as there is no mechanical arm and spinning motor needed to access data.

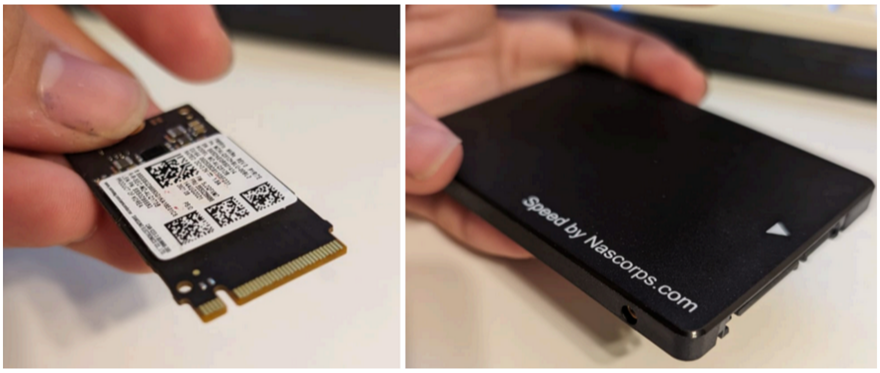


Figure 3.5: Two SSDs, one using the NVMe interface and one using SATA. Both are internal drives meant to be mounted into a computer's casing.

Pros

- They are much faster than HDDs, as they use non-mechanical NAND flash chips.
- They are less prone to data rot; SSDs typically last much longer than HDDs.
- They are less fragile than HDDs.
- They come in much more form factors and can fit almost everywhere.

[3] Only if used as the boot device; the drive at which the OS is stored.

Cons

- They are more expensive than HDDs.
- For backup purposes, they cannot store as much data as HDDs (>10 terabyte capacity SSDs don't really exist as of 2025). Buying smaller drives for backup still does not make sense, as it costs more per terabyte.
- Smaller SSDs and faster SSDs generate a lot of heat.

3.1.7 USB Mass Storage (Flash Drives)

These are like SSDs, but use a slightly different type of electronic flash chip. They do not consist of many moving parts and can carry small amounts of data in a small form factor. These are intended to be fully external, and uses the universal serial bus (USB, see section 2.2 for more details). These devices are very commonly seen for software installers that are not fully online, or for moving data through a physical device across computers. Figure 3.6 depicts it.

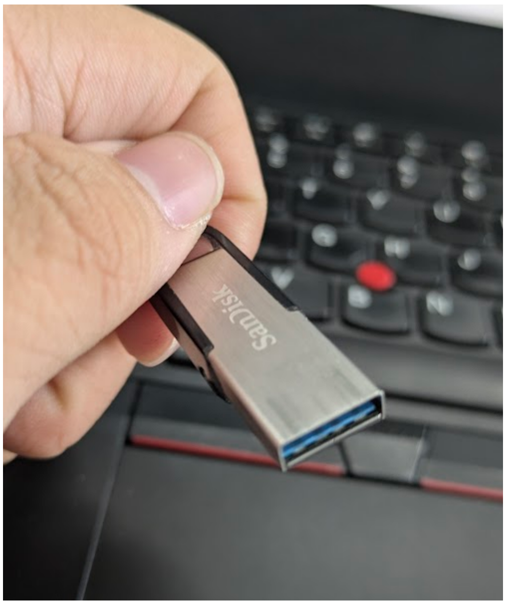


Figure 3.6: a USB flash drive.

3.1.8 Optical Media (CDs, DVDs, BDs)

Optical media look like discs with a hole in the middle for alignment. Unlike all the other storage devices, optical media just consists of the medium. You need a separate storage device, called an optical drive (or a DVD drive, CD drive etc.) to read them. The shiny side of the disc is what actually holds the data.

There are 3 types of discs, CDs, DVDs and BDs. CDs hold around 700mb of data, and was made for music. DVDs can hold around 4.7GB of data, and was made for standard definition films (lower resolution). BDs were made recently and is made for very high quality movies. It can hold anywhere between 25 and 100 GB of data. They are shown below:

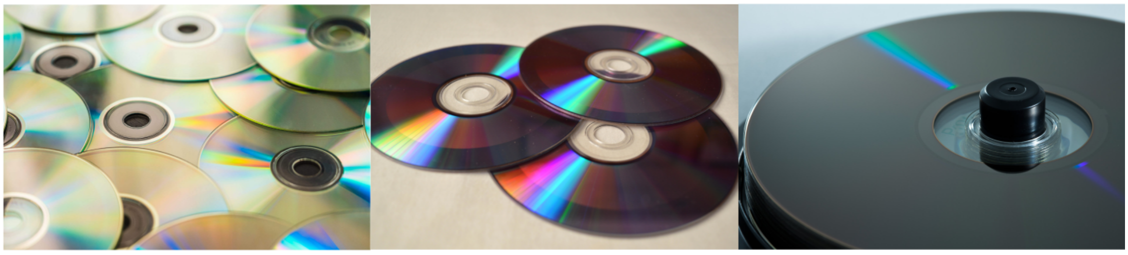


Figure 3.7: The back side of CDs, DVDs and BDs respectively.

Grooves and flat areas (pits and lands) are used to represent 1s and 0s on optical media. The image that depicts the DVD has a darker area; that is what data written to a DVD looks like. A laser is shone onto the troughs/flat areas, and the reflection angle is measured. Data can only be written to a CD **ONCE**[4] and cannot be rewritten[5]

3.1.9 Barcodes

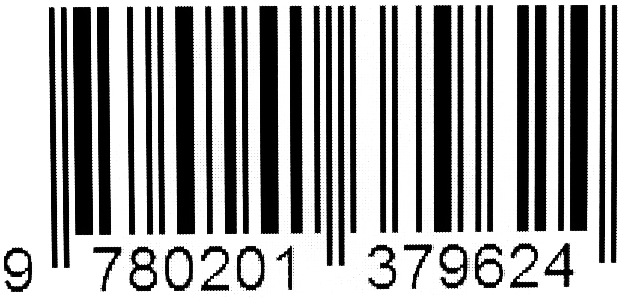


Figure 3.8: The back side of CDs, DVDs and BDsrespectively.

Barcodes are stick shaped codes usually printed on grocery products. They are everywhere; look for a drink bottle or a bag of vegetables; there is likely to be a barcode on it. Each black section represents a 1, and a whte represents a 0. There are 3 sections on the barcode, at the end, the middle and end to align the reader to not read false data.

[4] This is called burning.

[5] They can, but only for special CDs with the -RW suffix (-RE for blu-rays)

3.1.10 QR Codes

QR codes are like barcodes, but 2D. Consider the following QR code that stores some text:

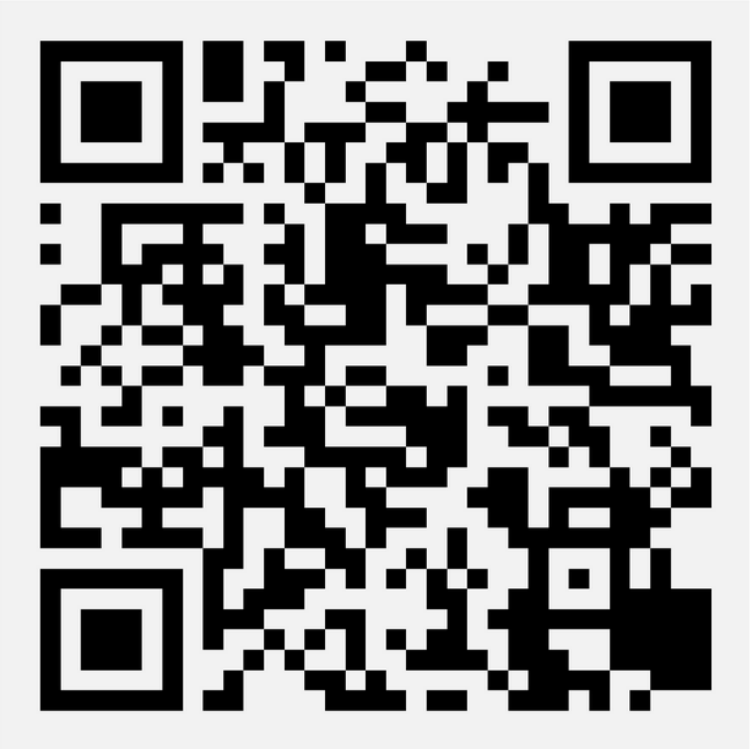


Figure 3.9: A QR code.

They are 2D matrices of squares that store small amounts of data; but more so than barcodes. They store binary data, and have error correction data encoded within them, to avoid errors in reading when the QR code is damaged. There are also large regions that act as anchors for QR code reading software to de-warp the QR code if scanned at an angle.

3.1.11 Cloud Storage

stub

3.1.12 Medium vs. Storage Device

The important distinction between a medium and a storage device is that **The medium is the physical material that stores data; the binary 1s and 0s** whereas **The storage device is an electronic piece of hardware that retreives and writes data to the medium.**

3.2 The Von Neumann Architecture

The von neumann architecture specifies a generic structure for CPUs and microprocessors to follow when they are designed. It dictates that the data used to store programs and the data used by the program (tempoary values, variables in memory) should co-exist in the same memory.

The following is a diagram of the von neumann architecture:

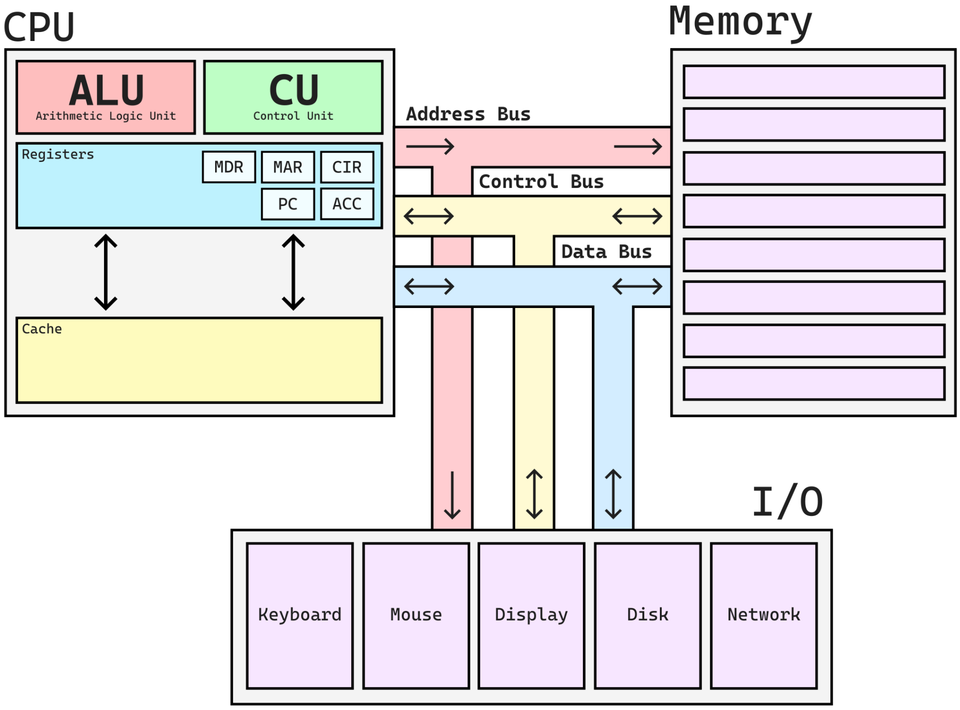


Figure 3.10: A bird's eye view of the von neumann architecture.

3.2.1 The CPU

The CPU or the Central Processing Unit is the brains of your computer. It carries out all the instructions ever passed through your CPU, and is the center of your computer's activity. Modern CPUs have the ability to do many things at once, like playing music while doing word processing[6]

CPUs look like this:

[6] A very, very, very oversimplified example.

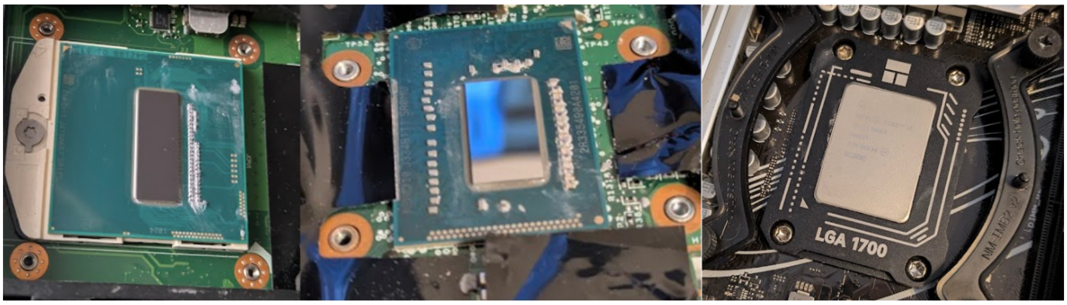


Figure 3.11: An Laptop CPU in a socket, an Intel Core i7-4712MQ, a Laptop CPU directly soldered on the motherboard, an Intel Core i7-3520M, and a desktop Intel Core i7-14700KF.

Extra: The shiny part of the CPU is called the die. The die is filled with extremely dense transistors which are all semi-conductive. The die actually does the processing. For those who wonder, no, you cannot see the individual transistors and registers on the CPU; they are so incredibly small and the parts of the CPU are so incredibly small it is impossible to reverse-engineer its architecture; even with microscopes.

Here are some important points:

- CPU's are also called microprocessors. They are interchangeable.
- They carry out all the instructions that your computer must do to work.
- They exist in phones, tablets, computers, game consoles, etc.
- They are responsible for handling input, calculating results and producing outputs[7]
- A CPU is a type of integrated circuit; a large network of transistors in a single unit

Inside the CPU, there are many important components that are critical to its function. Figure 3.10 and 3.12 shows this. Each section that follows explains each part of the diagram. For information on Memory and I/O (or IO), See section 3.1.2 and 3.3 respectively.

[7] Example: Getting the input $5+3$ from the user, calculating the result and giving it back to you. This is oversimplified.

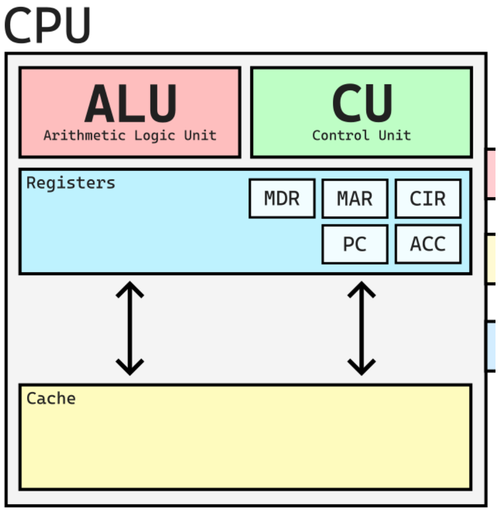


Figure 3.12: The CPU on the inside

3.2.2 The CU, ALU and Registers

The Control Unit, or CU is like the boss of your CPU. All operations that involve sending data to and from memory or interacting with I/O devices like the mouse and keyboard (refer to figure 3.10) is done by the control unit. It is also responsible for decoding instructions for the CPU to execute.

The ALU or the Arithmetic Logic Unit is responsible for performing all mathematical and logical operations, like addition, multiplication and so on, along with AND, OR, NOT, etc. For more information on logical operations, see section **??**.

Registers, like RAM is for fast data storage. The main difference between registers and main memory is that registers are built into the CPU itself. Registers store very little data, typically only 64 bit integers; but are much, much faster than main memory. They also typically serve special purposes. In the diagram, there are 5 important registers that are important for CPU operation.

- **The Program Counter (PC)** is a special register that stores the *address* of the current instruction that is being run by the CPU.
- **The Accumulator** is another special register in the CPU that stores intermediate results of arithmetic or logical operations, like the result of $3 + 5$ or $1 OR 0$. It can accumulate data over time for longer chains of instructions[8]
- **The Current Instruction Register (CIR)** stores the current instruction being executed. This differs from the PC, as the PC stores the *location* of the instruction, whereas the CIR stores the instruction itself.
- **The Memory Address Register (MAR)** stores addresses in memory that must be read from or written to.

[8] In modern CPUs, there is no accumulator. There are so many general-purpose registers that any one can act as an accumulator.

- **The Memory Data Register (MDR)** stores the data that must be read from or written to memory. This differs from the MAR, as the MAR stores *locations* (addresses), whereas the MDR stores the locations themselves.

General purpose registers also exist, which are registers that can be used for anything.

Extra for experts

The von neumann architecture in this syllabus is covered in a great way, but this does not reflect modern computers. For instance, there are many more special registers in a modern computer. For a CPU architecture called `aarch64` or `arm64` (very popular; used in Apple Macs, phones and some game consoles), there are a few more[9]

- **The Stack Pointer (SP)** stores the location of the stack. The stack is where large-sized variables are stored in code; it is slower but more flexible than registers as it can hold more data. The stack pointer stores where this continuous chunk of memory begins.
- **The Link Register (LR)** stores the address of the last executed instruction before a branch. Branches are instructions that tell the CPU to go to a specific location to execute instructions, then go back. Before the program counter is set to where the new instructions are, the link register is set so that the CPU knows where to return to after executing the branch.
- **The Current Program Status Register (CPSR)** stores important information about instruction results. After, let's say a comparison (such as `5 > 3`), the result is stored in the CPSR. Then, later instructions read it to execute conditional branches (if statements in code).

3.2.3 Cache

Cache simply stores commonly used data and instructions. If there are many similar operations in a row with similar results, the operations themselves and the results of them are put in cache. This means that similar calculations do not have to be done over and over again, instead it is stored in cache (or cached).

Cache is made of SRAM and is inside the CPU, therefore it is fast (faster than RAM!)

3.2.4 Buses

Buses[10] are channels by which data flows. Refer to figure 3.10 for details.

There are 3 buses to speak of in the von neumann architecture:

[9] This may be specific to aarch64, but other architectures have them too.

[10] For the brits, Busses

- **The Address Bus** is the channel for memory addresses. When the CPU wants to read something from memory, the address is transmitted through the address bus. It is unidirectional, meaning that it is only between the CPU and RAM. It is as wide as memory addresses can be. On modern computers, this is 64 bits wide, or 8 bytes.
- **The Control Bus** is the channel for control signals. It carries signals from the control unit to memory and/or input and output devices (via the I/O controller, pictured in the diagram). These signals are for the CPU, Memory and I/O to coordinate and cooperate. An example could be the CPU telling the RAM that it wants to read data, or telling I/O that it wants to send data to a device. The bus is only 8 bits wide usually, as these only pass simple signals. It is bidirectional, meaning that data goes both ways.
- **The Data Bus** is the channel for actual data. Data that the keyboard sends to the CPU, or data that the CPU sends to memory for writing, like numbers. It is bidirectional for both reads and writes, and the width of the data sent depends, ranging from 8 bits (usually) to 64, or even higher in newer architectures.

3.2.5 The Fetch-Decode-Execute Cycle

The fetch-decode-execute cycle is the cycle that the CPU follows to execute instructions over and over again. The cycle itself is as follows:

Fetch

Is when the CPU gets the next instruction to run from memory. *The instruction at the program counter ends up at the current instruction register.* In terms of the busses,

- The CPU places the program counter's value on the **address bus**, because it's an address.
- The CPU sends a read signal to memory over the **control bus**, to read the data at the address.
- Memory responds by sending the instruction at the program counter over the **data bus**
- The data is caught in the current instruction register.

Decode

The instruction is decoded in the control unit.

Execute

The control unit executes the instruction. For arithmetic instructions and comparisons, it is sent to the ALU. For memory-related instructions, data is sent to and from memory via the CU and the buses.

There is a great video by Tom Scott on this, watch it here.

3.2.6 The Clock Speed

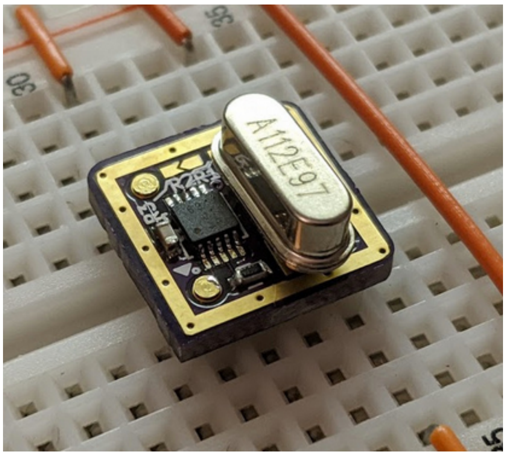


Figure 3.13: A crystal clock generator, which generates electrical pulses.

The clock speed is the rate at which the Fetch-Decode-Execute cycle is ran at. Figure 3.13 shows a crystal clock generator, which generates electrical pulses. On each pulse, the CPU does either a fetch, decode or execute. Fetches and Executes may take more than one pulse to run, but simply assume each stage takes one pulse.

Since this describes an event that happens regularly, we use hertz (times per second) to measure this. In modern computers and phones, CPUs tend to run at around 3.5 to 4.5 Gigahertz, which means 3.5 billion to 4.5 billion fetches/decodes/executes per second, which is very fast.

3.2.7 Cores

Cores are like Sub-CPUs in each CPU. Each CPU shown in figure 3.11 has more than 1 core. Each of them can carry out tasks independently of each other; despite how they can communicate together. For those who have already read this whole section, each core can carry out its own fetch-decode-execute cycle.

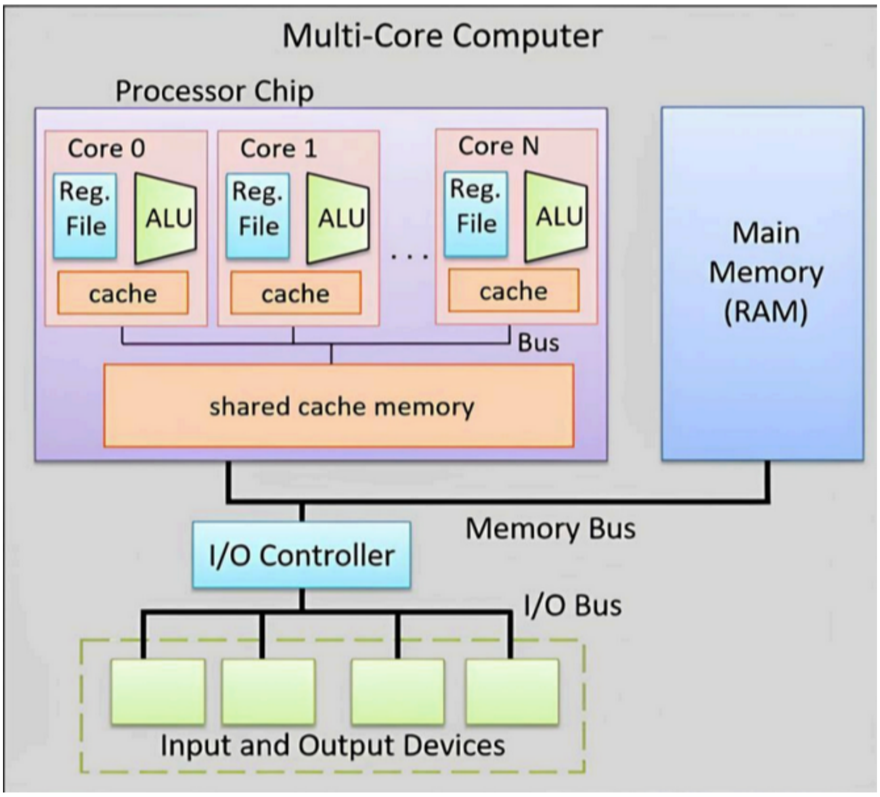


Figure 3.14: A diagram showing what cores look like in a CPU.

Each core talks to each other, and every core can communicate with every other core. They are bound together tightly in the CPU; knowing exactly how they communicate is not a part of the syllabus and is too complicated to break down.

There are some special names for multi-core CPUs with lower core counts. **Quad core CPUs** have 4 cores. **Dual core CPUs** have 2 cores. **Single core CPUs** have 1 core, as per the name.

3.2.8 Increasing CPU speed

There are 3 ways:

1. **Increasing the clock speed** increases the amount of F/D/E cycles run per second, meaning more can be done per second; implying a faster chip.
2. **Increasing cache** means that more common instructions and their results can be stored at one time, preventing constantly recalculating these instructions.
3. **Increasing the amount of cores** means that there are more units that can run the F/D/E cycle per second, meaning more things can be done concurrently.

3.3 Input and Output Devices

3.3.1 Input Devices

Input devices detect actions from the user. For example, a touchscreen detects the action of the user touching the screen, and a button detects the action of the user pressing the switch down. Examples of input devices include:

- **Barcode Scanners**, which scans barcodes
- **QR Code Scanners**, which scans QR codes
- **Keyboards**, for text input
- **Mice (Optical Mice)**, which gives user precise position based inputs into a computer.
- **Touchpads/Trackpads**, which does the same thing as a mouse, but it revolves around the user moving their finger on a sensitive surface.
- **Digital Cameras**, which capture the outside world as binary data
- **Microphones**, which record sound data and convert it into digital sound waves.
- **Scanners** captures 2D images or 3D models of a physical object in a precise way.
  - 2D scanners scan 2D documents and such to create a typically black-and-white image that is sharp. Similar to a camera, but works best for documents.
  - 3D scanners employ complex technology to create 3D models of objects you place in it.
- **Touchscreens**, which allows the user to touch the screen itself to allow less-precise but human gesture controls for a computer/phone/tablet.
  - Resistive Touchscreens have a flexible layer on top of the display. When the user presses the screen, the bottom of the display can detect where the user pressed on the flexible layer.
  - Capacitive Touchscreens detect touches by seeing if there is a conductive material on the surface of it, which is electrically charged. Since your finger is very minimally conductive, it can detect it as a touch.
  - Infrared Touchscreens detect touches by shining a row and column of closely-packed infrared lasers. When your finger blocks 2 beams, the x and y coordinates are sent to the computer.

Note that details about the devices themselves arent provided. Please research on your own or refer to the textbook.

3.3.2 Output Devices

Output devices allows a computer system to create a tangible reaction or an event in the real world, like displaying data or starting a water pump.

- **Displays**, or screens are found in laptops, TVs, tablets, phones, etc. Their role is obvious.
- **Projectors** are like displays, but they generate light beams that shine the computer's output into a surface, like a wall, whiteboard or a projector screen.
- **Actuators**, including things like solenoids and pumps creates motion that is controlled by a computer. As an example, a solenoid is like a piston; it pushes things.
- **Printers** convert image data from a computer[11] and puts it onto paper. There are 3 types of printers.
  - Laser Printers shine a laser onto toner, which is a very fine powder that acts as a dye onto a piece of paper, basically burning it into the paper. Almost all workplaces and schools uses them.
  - Inkjet Printers shoot very small jets of ink onto paper. They produce better colors than laser, which is why they are used more in art institutions.
  - Dot-matrix Printers are very old and typically only used in airports. They are the loudest, and make a high pitched screeching noise. You may have seen them near boarding gates; they punch very shallow holes and fill them with ink in a matrix of dots. The resolution and quality is low.
- **Speakers** broadcast computerized binary sound data.
- **3D printers** are like 2D printers, but they use a filament that is heated up to produce accurate reproductions of 3D models in a computer/

3.3.3 Sensors

Sensors detect changes in the surrounding environment, i.e. they *sense* the things happening around it. As data in the real world is always analog, it must always go through an ADC (Analog to Digital Converter, *NOT a DAC!*)

- **Temperature Sensors** sense temperature.
- **Humidity Sensors** sense the concentration of water in the air.
- **Acoustic Sensors** detect specific sounds, like the sound of glass breaking in a criminal-infested museum.
- **Pressure Sensors** detect how hard an object has been pressed. Pressure sensors that detect the pressure of a container do also exist, but they are usually not digital.

[11] Fun fact: text must be rendered as pixels before it can be printed, so it is image data!

- **Gas Sensors** detects the concentration of a specific gas in the air.
- **An accelerometer** detects the acceleration of an object, like sudden shakes or movements of a phone.
- **A Gyroscope** detects the movement of an object in space, including forward and backward movements, up and down movement and rotation. In essence, a more precise accelerometer.
- **An active infrared sensor** detects if beams of infrared light are blocked or not. A signal will be sent out if it is.
- **A passive infrared sensor** detects IR heat radiation.

3.4 Network Hardware

Stub

stub

3.5 Embedded Systems

stub

Chapter 4

Software

4.1 The Operating System and Kernel

A computer without an operating system is like a human without a soul. Without an OS installed on your computer, it would be no more than a fancy metal/plastic brick with silicon in it.

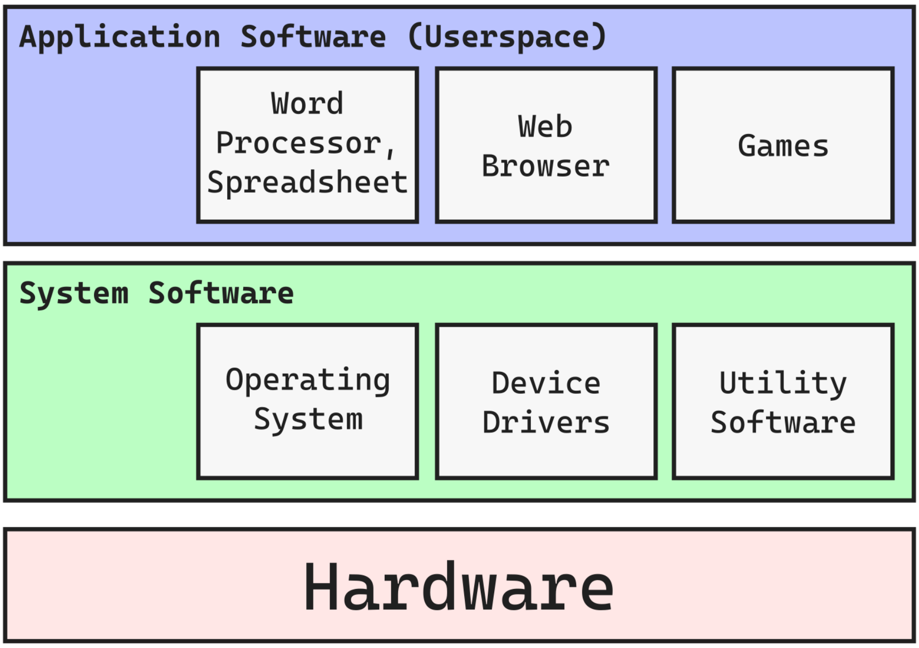


Figure 4.1: A very simplified software stack that Cambridge expects you to know.

Figure 4.1 shows the software that exists on your computer. All of this is stored in secondary storage on your boot device.

4.1.1 The OS and System Software

The operating system itself is a large set of programs that mainly run directly on the CPU, that gives the user an inteface to interact with the computer and connected devices, along with a mechanism to run programs, among others. The tasks that the OS carries out include:

1. Loads device drivers for hardware and creates abstractions for application and utility software.
2. Handles interrupts
3. Creates and manages virtual memory (see section 3.1.4) for application and utility software
4. Manages all files, including system files, files used by applications/programs and your files.
5. Allows for multitasking
6. Manages Hardware Peripherals
7. Manages user accounts and their permissions on the computer.
8. Provides a Human-Computer Interface (HCI)
9. Provides Mouse/Keyboard input.

Major Operating Systems include Microsoft Windows, Apple's macOS, Linux (GNU/Linux), and the *BSD family, among others. This is the reason why the interface you get when you use a Lenovo laptop, for example is different from if you use a Macintosh; the operating systems they use is different.

The kernel is simply a part of the operating system that creates the abstractions for hardware, so that userspace software can interact with it. On the list provided previously, items 1-5 is handled by the operating system, and the rest through special system software that is commonly grouped together with the OS.

Device Drivers

Device Drivers, or just Drivers are pieces of software that enables hardware devices to communicate with the rest of the operating system. These drivers are what makes your devices, like your keyboard, mouse, monitor, camera, and even storage devices to function correctly. All devices that connect to your computer have drivers.

In more complex terms, they are abstractions of the hardware and the raw electrical interfaces they provide for software written in higher-level languages to interact with.

Multitasking

Multitasking allows computers to carry out more than one task at a time. Each task is referred to as a *process* (plural: processes). All of these processes at the end of the day share their resources with every other process (therefore the rest of the computer). A process on your computer could be the process drawing website content to your screen in your web browser, and another being your music player.

However, processes can clash, i.e. they request to be run at the exact same time. The CPU simply cannot run more than one instruction at the same time; even on multi-core processors they share one set of buses. Therefore, the CPU must do the following:

- processes are allocated the computer's attention for a specific time.
- The process can be interrupted while it is running.
- The process is given a priority. Processes with more priority get more attention; they get more time to access the rest of the computer's resources.
- Different processes are placed into attention at different times and they are alternated between (not at a specific rate, though).

This mechanism of swapping between processes very quickly gives one the impression processes run at the same time. This strategy is called preemptive multitasking. The technique of actually managing these resources is called *scheduling*, it takes place within the operating system.

Interrupts

Interrupts are special signals that tells the CPU to immediately stop what it's doing to carry out something. There are two main types of interrupts, hardware and software interrupts.

Hardware Interrupts

This is when I/O devices create data that the CPU must look at. This happens on:

- Keyboard Presses
- Mouse movements
- Timer interrupts (for multitasking)
- Hardware failure (overheating)

To elaborate on *timer interrupts*, this is the mechanism that tells the CPU to swap between different tasks.

Software Interrupts

This is when software ran on the CPU itself generates interrupts. This happens in very specific cases, like:

- Dividing by Zero
- Accessing memory out of bounds[1].
- Two processes trying to read/write to the same mory at the same time (known as a race condition).
- Extra: Asking the operating to do something (system call/syscall).

System calls, or syscalls is out of the syllabus, but is important to all software. In order for programs to do literally anything, like acessing I/O devices, reading and writing to disk, printing text, reading text and accessing the network, it has to ask the operating system[2] to do it, because it is the only thing that has the privelege of doing such things (security reasons). Doing this creates a type of interrupt that forces the CPU to give extra attention to it.

Buffers

Buffers are just a general concept in programming that means a memory space used to put temporary data/collect data, only for it to be used and cleared later. In terms of OSes, buffers are used to maintain the state of programs before and after interrupts. Before an interrupt is called, the state of the CPU and virtual memory in the active process is saved to a buffer. After the interrupt, the memory is restored. The same happens when switching between processes for multitasking.

Servicing Interrupts

When interrupts are received, they must be *serviced*. Here are the procedures:

1. The state of the Program Counter (see **??**), all registers and a portion of memory is saved.
2. The interrupt service routine (ISR) is called, which is special code that is executed by loading the beginning of the code to the program counter. This makes the CPU start reading instructions from the ISR.
3. The ISR decides what must happen from the type of the interrupt.
4. The state of the registers, PC and memory is restored.

[1] Known as a segmentation fault. Since virtual memory is split into ”containers” (see figure 3.2), when you access memory outside, the operating system disallows you to do so by sending an interrupt.

[2] The kernel, to be more specific.

5. Normal execution continues.

Buffers and interrupts are typically used together in an OS to facilitate multitasking and interrupt handling. The textbook makes mention of a printer and how it uses buffers and interrupts. **The example is very oversimplified.**

Sending data to the printer from the CPU takes fractions of miliseconds. However, actually printing the data out can take around half a minute. If the CPU were to wait for this I/O operation to fully complete, the CPU would be stuck on one instruction telling the OS to print a single byte of data, meaning all other processes would not be able to run. The naïve solution would be to simply suspend the printing process and switch to another one, continuing multitasking. However, when the CPU returns to printing the document, it starts exactly where it left off, meaning it woult take even longer to print the document.

The solution is to save all the printing data into a print buffer, and instead of swapping between other processors and *live-producing the bytes*, it simply reads a region of memory (the buffer) that has the data already prepared. This means that printing can happen at near-normal speed.

Virtual Memory and Memory Management

The operating system manages memory in the sense that it is responsible for creating virtual memory. Since the OS has direct access to memory (Direct Memory Access or DMA in the Linux kernel), and all I/O devices (the hard drive), it combines them in such a way that a pool of memory is made. When a process is created, a slice of that memory is made and is mapped to a full *address space* for the process[3]. This is known as memory mapping.

- This is so that programs cannot access each other's memory, so that if a virus is on your computer, it cannot simply find your browser's memory space and read the passwords.
- Processes also do not have to worry about accidentally overlapping with each other's space, causing interrupts.
- Memory can also be added and removed as needed. If the program needs more memory, it can take any piece of memory it needs from physical RAM or from the hard drive, and can tack itself onto the process's address space. This means that no drop of memory is wasted (ultimate rule in Computer Science).

This mechanism of dynamically adding/removing space for programs means that physical RAM does not have to be *contiguous*, i.e. one next to each other. Instead it can look something like as follows:

[3] Not all of the space is accessible; it is 64 bits as that is how wide the address bus is.

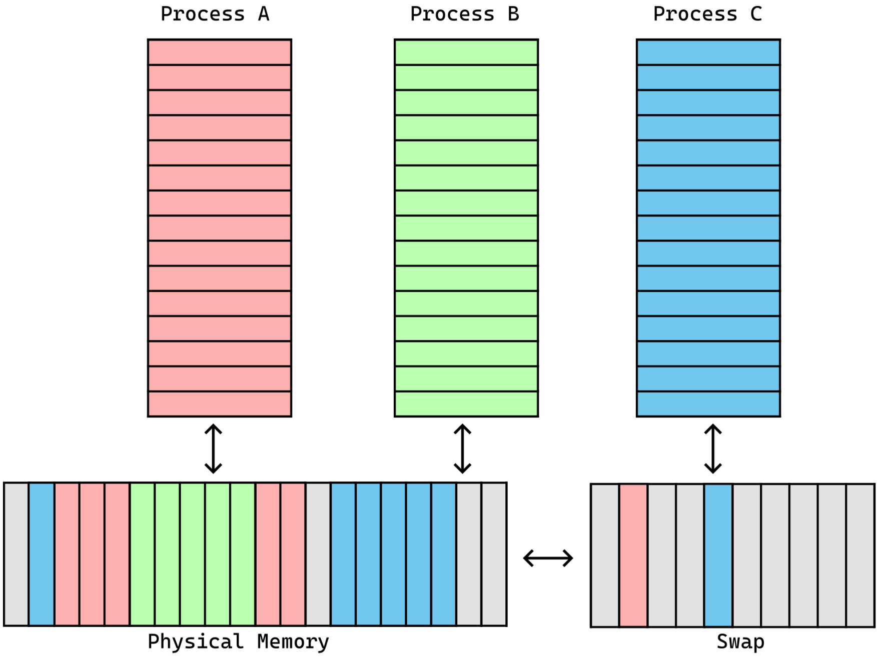


Figure 4.2: Memory mapping.

NOTE: each chunk of memory is called a **page**. Each page is the same size as another.

File Management

File management in general is a complex topic, and due to time constraints it cannot be explained in as much detail as Virtual Memory. The general takeaways are:

1. The hard drive is a hard slab of uselessness without a filesystem.
2. Filesystems specify a set of rules that determine how the disk should be split into parts. Since files don't exist, nor do folders, the file system makes them exist. It specifies how files and folders should be laid out on disk so that the OS can interpret it as files and folders for its operation.
3. The filesystem dictates things like file names and extensions (filename.pdf = filename is the file name and .pdf is the extension)
4. Copying, moving, renaming, etc. is all handled by the OS through the filesystem.
5. Filesystems can also have access control, meaning that different permissions (read, write, execute) can be set on files so that only specific users can can do the mentioned operations on them. As an example, teachers on a computer should be able to read and write to the folder of assessments, but students should only be able to read to the assessments.
6. The operating system reads these files from the filesystem into memory for operation.

What is a file, exactly?

This is actually something that comes up in exams quite a bit. After reading the previous section, you know that files don't exist on hardware, it is defined by software; the filesystem you use, the system by which you store files.

Files are distinct pieces of data that usually exist on secondary storage with a name, and sometimes a file extension. On the device itself, it exists in some location, and that location along with its file name is located in the filesystem header (in many cases, the **file allocation table**). This means that under the hood, files have 2 parts.

Hardware Peripheral Management

The operating system interacts with hardware devices. In summary:

1. The OS can communicate with all I/O devices through device drivers.
2. Devices are actually represented as files on the filesystem (but not stored on the hard disk!). The kernel keeps the file open for itself to write to, and no other thing must interfere.
3. The OS controls them by handling all the necessary interrupts (see section 4.1.1), and handles all buffers and queues.

For more examples, refer to page 158 of the textbook.

User Accounts and Permissions

User accounts are a concept you should be familiar with. Accounts on a computer are not dissimilar from accounts on, say, Google; different users have different usernames and passwords on a computer system. This is useful for identifying different people, separating different users' data, and permissions management.

The OS manages all user accounts and sets up techniques to manage passwords and personal folders (home folders, what stores the desktop, documents, etc.) and manages who can access it. The OS creates different levels of privelege by segregating different users and granting them the ability to do different tasks based on their usernames and the user groups they are placed in. Some will have the privelege to write to the whole startup disk, while others only have privelege to write to their own home folder. Some will also have administrator priveleges while others will not.

This is not dissmilar to classroom management systems like the one at our school; teachers can create and set homework while we can only read them.

Human-Computer Interfaces (HCI)

HCI's allow the user to interact with the computer through an interface. There are 2 types of interfaces used in the modern day, **Command-line Interfaces (CLI)** and **Graphical User Interfaces (GUI)**. They allow the user to perform tasks the operating system is capable of doing (everything mentioned above), like launching programs, managing files and users.

GUIs

GUIs, or Graphical User Interfaces is the most common type of HCI out there.

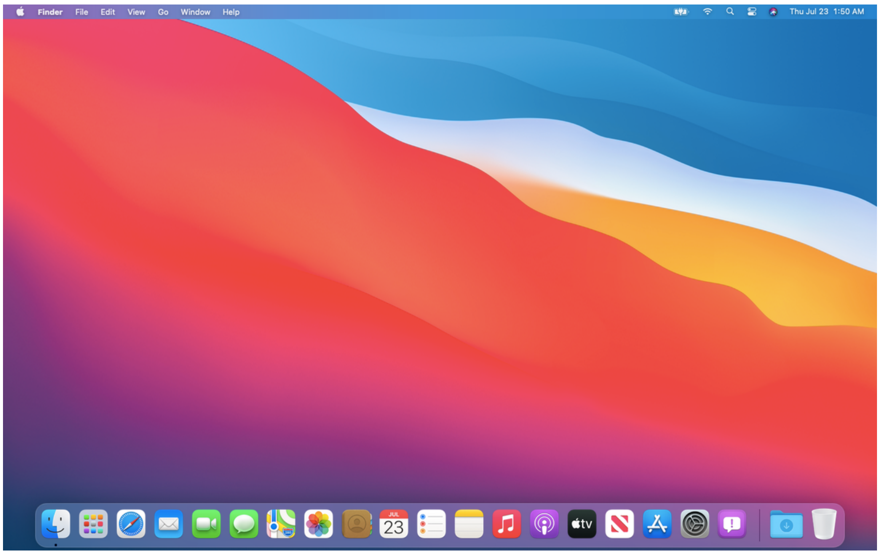


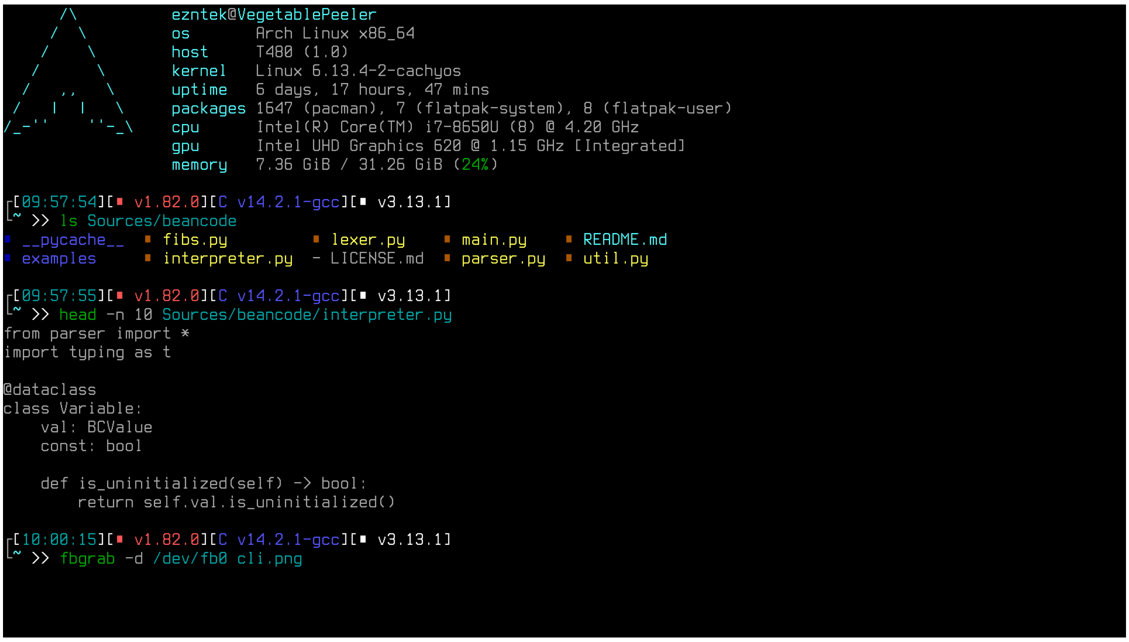
Figure 4.3: The desktop of macOS 11 "Big Sur".

They usually use a group of technology called WIMP (Windows, Icons, Menus and Pointer) which includes all the elements mentioned previously to give you an interface with menus, windows, icons and a pointing device to navigate them. This was developed for use on personal computers (its origins are disputed) by several companies separately.

There are also post-WIMP interactions. This is used on both phones, tablets and some laptops with touchscreens here instead of using a pointer, the device uses less menus and more icons, buttons and gestures powered by the touchscreen to let the user navigate it. Both iOS and Android uses these. The most notable desktop OS that does not have post-WIMP interactions is macOS.

CLIs

CLIs, or command line interfaces was the most prevalant HCI before GUIs were idealized. Even now, there are operating systems (Linux and BSDs) that allow the user to skip the WIMP stack and directly perform all tasks with a CLI.

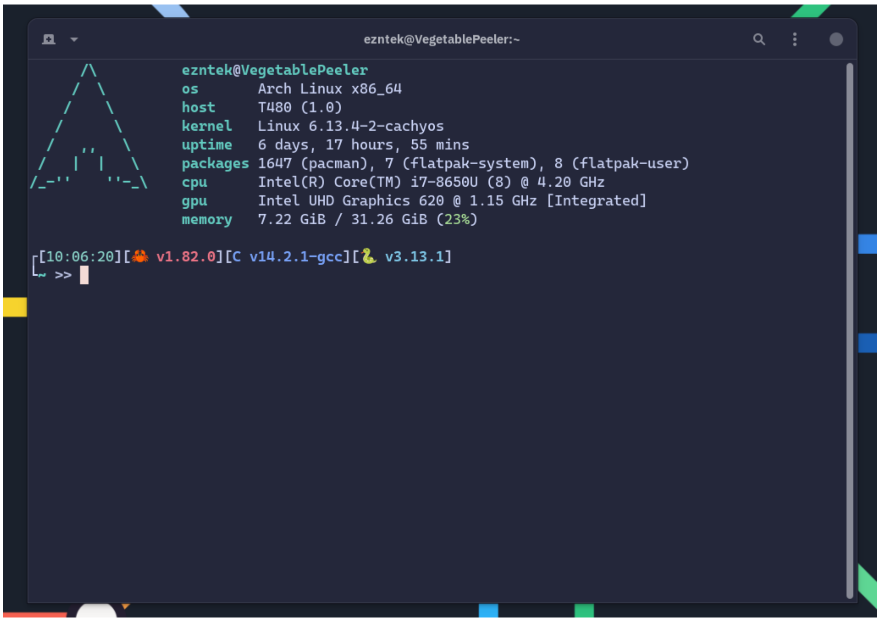


```
      /\          eznte k@VegetablePeeler
     /  \         os       Arch Linux x86_64
    /    \        host     T480 (1.0)
   /      \       kernel   Linux 6.13.4-2-cachyos
  /   ,,   \      uptime   6 days, 17 hours, 47 mins
 /   |  |   \     packages 1647 (pacman), 7 (flatpak-system), 8 (flatpak-user)
/_-''    ''-_\    cpu      Intel(R) Core(TM) i7-8650U (8) @ 4.20 GHz
                  gpu      Intel UHD Graphics 620 @ 1.15 GHz [Integrated]
                  memory   7.36 GiB / 31.26 GiB (24%)

┌[09:57:54][■ v1.82.0][C v14.2.1-gcc][■ v3.13.1]
└~ >> ls Sources/beancode
■ __pycache__  ■ fibs.py         ■ lexer.py    ■ main.py    ■ README.md
■ examples     ■ interpreter.py  - LICENSE.md  ■ parser.py  ■ util.py

┌[09:57:55][■ v1.82.0][C v14.2.1-gcc][■ v3.13.1]
└~ >> head -n 10 Sources/beancode/interpreter.py
from parser import *
import typing as t

@dataclass
class Variable:
    val: BCValue
    const: bool

    def is_uninitialized(self) -> bool:
        return self.val.is_uninitialized()

┌[10:00:15][■ v1.82.0][C v14.2.1-gcc][■ v3.13.1]
└~ >> fbgrab -d /dev/fb0 cli.png
```

Figure 4.4: Linux in tty mode, meaning there is no GUI.

CLIs, instead of GUIs do not make use of the mouse pointer, and there are no graphics, like windows and icons. Instead, you type comands into a **console** or **terminal** that perform very specific tasks, like making a directory with permissions or opening a text file and editing it.

Power users and programmers like you use CLIs, as you can directly communicate with the computer and operating system without needing to dig for menus here and there and constantly using the mouse to perform basic tasks. Instead, you only use the keyboard and type in the exact commands to do the exact things you must do all at once. CLIs also lets you automate complex tasks in the form of scripts, which run many commands one after another. You can use a **Terminal Emulator** to emulate a full CLI inside a window, like so:



```
     /\          eztek@VegetablePeeler
    /  \         os       Arch Linux x86_64
   /    \        host     T480 (1.0)
  /      \       kernel   Linux 6.13.4-2-cachyos
 /   ,,   \      uptime   6 days, 17 hours, 55 mins
/   |  |   \     packages 1647 (pacman), 7 (flatpak-system), 8 (flatpak-user)
/_-''    ''-_\   cpu      Intel(R) Core(TM) i7-8650U (8) @ 4.20 GHz
                 gpu      Intel UHD Graphics 620 @ 1.15 GHz [Integrated]
                 memory   7.22 GiB / 31.26 GiB (23%)

┌[10:06:20][🦀 v1.82.0][C v14.2.1-gcc][🐍 v3.13.1]
└~ >> 
```

Figure 4.5: A terminal emulator on Linux, which allows one to emulate a full terminal in a GUI.

4.1.2 System Software

What sits on top of the OS includes system software. These are pieces of software that allows hardware to run correctly, provides useful functions and allows the user to communicate with the computer. There are many examples of system software:

- **Compilers, Interpreters and Linkers** will be covered in section 4.4, but for now, know that this is what turns your code into machine code instructions that can be used in the fetch-decode-execute cycle.
- **Device Drivers** are pieces of software that directly run on the CPU and is a part of the kernel; that makes peripherals like your display, mouse and keyboard function correctly. They also provide an interface so that a userspace program can use it, like drawing to the display or reading the position of the mouse.
- **Other Examples** include antiviruses (provides protection against malicious software), defragmentation software (compacts space on HDDs), screenshot software, screensavers, your lock screen, a file browser, file backup software, etc.

For more detailed examples of system software, refer to the textbook in the corresponding section.

4.2 System Startup

System startup is defined as the process the computer takes from being complteely shut off (no current passing through the CPU) to running the operating system, able to execute multiple processes. The process by which this happens is:

- **The BIOS** is loaded into RAM. This is stored on a small ROM (EEPROM) chip on the motherboard, and it contains important code that performs hardware initialization. This means that all the hardware connected to the I/O controller, like all the ports, the display, keyboard, mouse and even RAM is initialized and acknowledged.
- **The BIOS loads the boostrap loader/bootloader.** The BIOS is able to locate the operating system on disk as it just initialized it, it loads the first few sections of it, which contains the *kernel*[4].
- The rest of the OS starts up, along with userspace software. See section 4.3 for a discussion. After the core of the OS is loaded into RAM, it re-initializes the hardware by loading more advance device drivers. It then starts up whatever system services that must be ran, and then ar GUI interface.

The BIOS is also called **the firmware**, which is defined as a program that provides low level control for devices.

The BIOS is stored in a special type of ROM that can be reprogrammed, so that it can be updated if it must be but in normal circumstances is read-only. It is usually only ~16MiB.

4.3 Userspace Software

Userspace software differs from the software mentioned in section 4.1. Cambridge refers to this simply as application software, but the real distinction comes from *whether the program has direct access to physical memory and devices, or if the program must run through virtual memory and device drivers.*

Examples of Application Software[5] include and are not limited to:

- **Word processors**, like Microsoft Word or Google Docs
- **Spreadsheet software**, like Google Sheets or Excel
- **Web browsers**, like Firefox and Chrome
- **Control and Measuring software**, which connects to scientific instruments, sensors and such to measure physical quantities in the real world, and controls embedded systems.
- **Multimedia Apps**, like video and audio players such as VLC
- **Photo and Video editors**, like Adobe Photoshop, Premiere, or others like GIMP and DaVinci Resolve

[4] For a discussion on the kernel, see section 4.1. The kernel is the very core of the OS and handles all the system-related tasks the OS does, like peripheral management, filesystem, virtual memory, and multitasking

[5] counts as userspace software!

- **Graphics Manipulation Software** is mentioned in the textbook, but is identical to a Photo Editor. They edit photos (raster/bitmap images). There is also software to edit vector images.

Extra: The more technical definition of userspace software is that userspace software does not have direct access to memory and I/O. Kernel-space software can make use of the Von Neumann architecture directly; i.e. directly place things on buses, read and write to physical memory, and access I/O. Userspace software must access memory via virtual memory, and I/O through device drivers. This means that a lot of system software is actually userspace.

4.4 Translators, Compilers and Interpreters

Translators (dissmilar to Google Translate or DeepL) translates programming languages into different programming languages. All of this is done so that at one point, code that you write can be translated into the raw instructions that your CPU understands, known as **machine code**. There are 3 main types of translators that you must be aware of for your final exam.

4.4.1 High-level and Low-level languages

High level (HLLs) and Low level languages (LLLs) are categories used to describe the amount of control the programmer has over the CPU.

High-level languages

HLLs allows the programmer to focus on the problem itself, without the prgorammer needing to know anything about the CPU the code will be ran on. They are designed with programmers in mind; they provide human-like features like if statements, loops, procedures and other things a human would find natural to have when giving a machine instructions.

- Read and understand the code better, as it is closer to English
- Write code faster
- Debug faster
- Less maintenance when the code is deployed.

Prominent HLLs include **Python, Java, Go, Rust, JavaScript**. Disputed ones include C, C++ and Rust.

Low-level languages

These languages give programmers more control over the CPU and the memory that is being manipulated in the program. To use these languages, you at least need to know how memory works, and sometimes how the CPU executes code. Prominent examples include **Assembly Language, Machine Code** and disputed ones include C, C++ and Rust[6].

Some features include:

- Programs run much faster, as they are closer to the native CPU
- Programming takes longer
- One must worry both about the problem at hand and the CPU used to run it
- Debugging is slower, as the code is more complex
- More maintenance as bugs are harder to spot

4.4.2 Types of translators

Assemblers

Assemblers are the most simple type of translator. They take assembly language, like so:

```
.data
txt:
    .ascii "hello, world!\n"

.text
.global _start
_start:
    mov x8, #64
    mov x0, #1
    ldr x1, =txt
    mov x2, 14
    svc #0
    mov x8, #93
    mov x0, xzr
    svc #0
```

and translates it into machine code that can be used in the fetch-decode-execute cycle, all at one time. Machine code is represented entirly in hexadecimal, like so:

[6] They are disputed because you don't really need to know how the underlying CPU works and the native assembly it takes, but you do need to know exactly how RAM and manual memory management works

```
00000000: 0808 80d2 2000 80d2 c100 0058 c201 80d2  .... ......X....
00000010: 0100 00d4 a80b 80d2 e003 1faa 0100 00d4  ................
00000020: 0000 0000 0000 0000                     ........
```

(output from the `xxd` utility)

However, hexadecimal is hard to read, so programmers programming directly for the CPU use assembly. They can manipulate the fetch-decode-execute cycle directly and do exact things with the CPU, while writing in a better language than straight-up binary/hexadecimal.

An Executable is defined as any piece of machine code that can be run in the fetch-decode-execute cycle.

Compilers

Compilers are similar to assemblers, in that they translate things all at once. Compilers take **source code**, which is the code you write, and it converts it into executable machine code. They translate all the code at once. Source code is also sometimes called a translation unit. A prominent compiled language is C, here is a C program that does the same thing as the assembly given above (prints ”hello world”):

```c
#include <stdio.h>

int main(void) {
    printf("hello, world!\n");
    return 0;
}
```

This will translate into machine code that can be used in the fetch-decode-execute cycle. Compilers are typically used by low-level languages. However, languages like Go have a compiler, and Go is high-level. Compilers are programs in themselves.

Advantages include:

- They generate code that runs much faster.
- All errors are reported all at once.
- Compiled programs take up less space in memory.
- Compiled programs do not require an interpreter to run them.

Disadvantages include:

- Compiling large projects takes a very long time.
- Writing, testing and debugging programs written in a compiled language is harder and takes more time. It typically requires special tools like debuggers and address sanitizers.

Interpreters

Interpreters are unlike Compilers, as Compilers translate the whole piece of code all at once and emits all errors all at once. Interpreters, on the other hand, runs the code on a line-by-line basis[7], and only emits an error when it hits a line that has it. It does not tell you about all the errors in the file all at once.

Python is a prominent interpreted programming language, here is a Python program that does the same thing as the C program above:

```
print("hello, world!")
```

Interpreters are usually only used for HLLs, as you do not have to directly manage memory. Interpreters are also programs in themselves. The program, therefore, can simply do all the memory for the HLL, so the source code the interpreter reads does not have to manage its memory.

Advantages include:

- They are easier to write, test and debug.
- They are simpler to run.
- Maintenance is easy, as there are typically fewer bugs produced in interpreted languages.

Disadvantages include:

- They are slower than compiled languages.
- They take up more space on disk and more memory, as one must also store the interpreter somewhere.
- Programs cannot be run without the interpreter.
- Errors are reported gradually as the program runs.

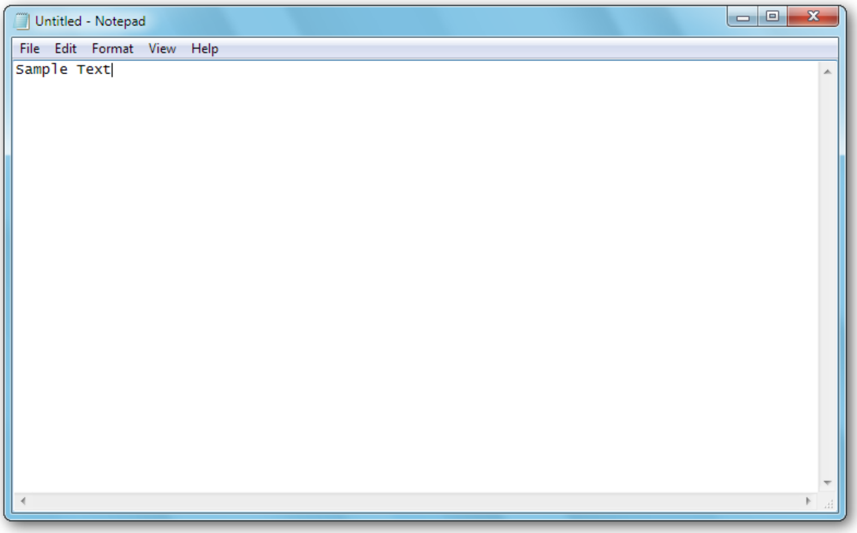
4.5 Development Environments

There are many ways to actually author code. There are 3 main ways of doing so.

Text Editors

They are typically the most simple. They are meant to just edit *plain text*, which means raw ASCII/Unicode data without any formatting data. On macOS, the default text editor is TextEdit, on Windows it is Notepad, and on Linux...it depends.

[7] This is technically untrue, but all you must know is that this is true.



Sample Text

Figure 4.6: A screenshot of Windows Notepad.

They typically do not have any convenience features for programmers, which is why people don't typically use them.

IDEs (Integrated Development Environments)

These are much more sophisticated than just text editors. They are like text editors, but tailored to writing code. They usually have tabs and nice features like a compiler/interpreter built into them, a debugger, error diagnostics, autocompletion, auto-documenters/documentation generators and pretty-printing.

For a more thorough discussion on IDEs, look at page 170 on the textbook.

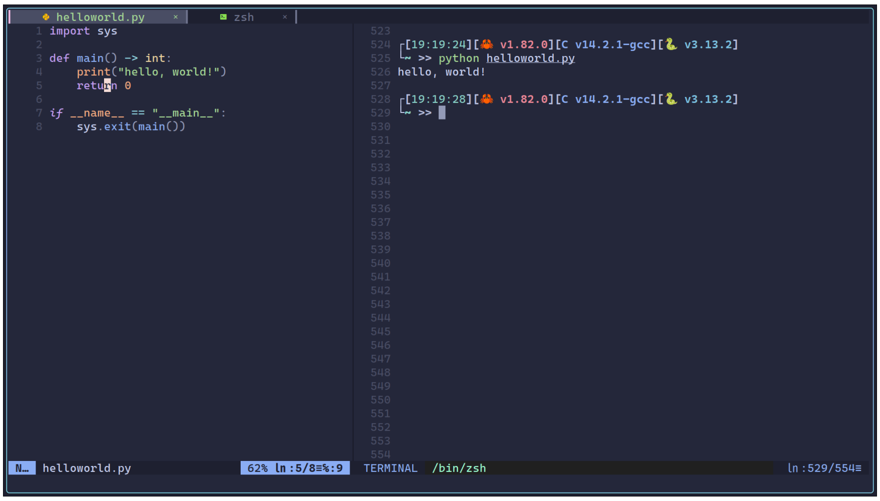
Code Editors

These are more sophisticated than just text editors, as they have features like syntax highlighting[8] and coding-specific features. They are not talked about often, as most people just use IDEs. This is also not in the syllabus, but is interesting to know.

Personally, I[9] use **`neovim`** on Linux, it is based in the terminal. It looks as follows:

[8] ”fancy colors”.

[9] Eason Qin.



```python
import sys

def main() -> int:
    print("hello, world!")
    return 0

if __name__ == "__main__":
    sys.exit(main())
```

```
┌[19:19:24][🦀 v1.82.0][C v14.2.1-gcc][🐍 v3.13.2]
└~ >> python helloworld.py
hello, world!

┌[19:19:28][🦀 v1.82.0][C v14.2.1-gcc][🐍 v3.13.2]
└~ >>
```

Figure 4.7: A screenshot of neovim running on Hyprland.

Chapter 5

The Internet and Its Uses

5.1 The Internet and the Worldwide Web (WWW)

The internet is one of the most influential creations of society. It has impacted all of us in some way. This section will focus on it. The internet at its core is just a bunch of computers connected together over a very large wide-area network (WAN), hence the words INTERconnected NETwork.

5.1.1 The Internet vs the Worldwide Web (WWW)

Internets come from the words Interconnected Network; they are literally just computers connected together that can share data amongst themselves. If I took 5 laptops and connected each of them to each other, that would technically be an internet. The internet has some protocols (rules) that are defined for computers to communicate in a uniform way.

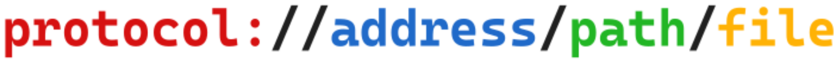
- The internet that we know of is the largest collection of computers that are connected together. They consist of servers, which provide the websites and services, and they are all connected to each other in some way.
- The internet is more of a concept that a tangible thing; the servers that you can touch are just computers. The connection between the servers is what the internet really is. However, **it still relies on a physical infrastructure**

The World Wide Web (or WWW) is just a part of the internet that we can access. The world wide web consist of the things that sit on top of the internet, all the web pages and services like Google, YouTube, Discord, Reddit, Instagram, etc. are a part of the WWW. The WWW also provides extra protocols on top of the internet's protocols that relate to the content on the WWW, like how websites should be written, encryption, etc.

In summary, the internet are the connections between physical infrastructure that provide the technology for computers to communicate, and the WWW are the webpages, content and protocols that sits on top of the internet's protocols.

5.1.2 URLs (Uniform Resource Locators)

Web browsers, or just browsers allows you to view content on the WWW. Browsers interpret HTML**Hypertext Markup Langauge**, which is a specific programming language that defines the basic component and the text in a webpage, think the bricks in your house. To access these files, the browser must visit a URL, which locates a location on the WWW. They look as follows:



protocol://address/path/file

Figure 5.1: The parts of a URL

The protocol is typically either `http` or `https`.

The domain (or web address) is the website's name, like `google.com`.

- The domain host (or the subdomain) is what goes before the website's name, like `images.` in `images.google.com`.
- The domain name, which is `google`.
- The domain suffix/domain type, which contains `.com`, `.net`, etc. Sometimes it is a country code, like `.uk`, `.us` or `.cn`, for example.

The path are like the directions leading up to the file that is to be loaded. Here, it is just one thing, but you can have something like:

`https://www.cambridgeinternational.org/why-choose-us/benefits-of-a-cambridge-education/`

where the path is 2 parts, `why-choose-us` and `benefits-of-a-cambridge-education`. You can also think of it like sections leading up to a paragraph in a textbook.

The file is the document or the actual data for the browser to load. It can be an html file or something else, you will see suffixes like `.jsp` and `.aspx`. It is a lot like files on your hard drive.

5.1.3 HTTP and HTTPS

HTTP, or the hypertext transfer protocol, is a set of rules that defines how data should be sent over the internet. It determines how data should be sent between a client (somebody visiting the server), and a server. It includes details like error checking and how clients/servers should respond to each others' requests. It is then up to web browsers and web server software[1] to comply with HTTP for it to be compatible with existing infrastructure.

[1] software that provides the actual website and the HTML behind it to the client.

HTTPS is like HTTP, but uses TLS (Transport Layer Security) or SSL (Secure Socket Layer)[2] to communicate. This means that instead of sending the raw bytes of the HTTP request, containing potentially sensitive information, it is encrypted so that onlookers cannot modify or read the data. the S in HTTPS means its secure.

5.1.4 Web browsers

These are programs that are capable of reading HTML files, displaying website layouts, playing sound, video and viewing PDFs, among others. They have the ability to connect to the internet and allows the user to freely access the worldwide web. You are likely using one right now; Google Chrome and Mozilla Firefox are very popular ones.

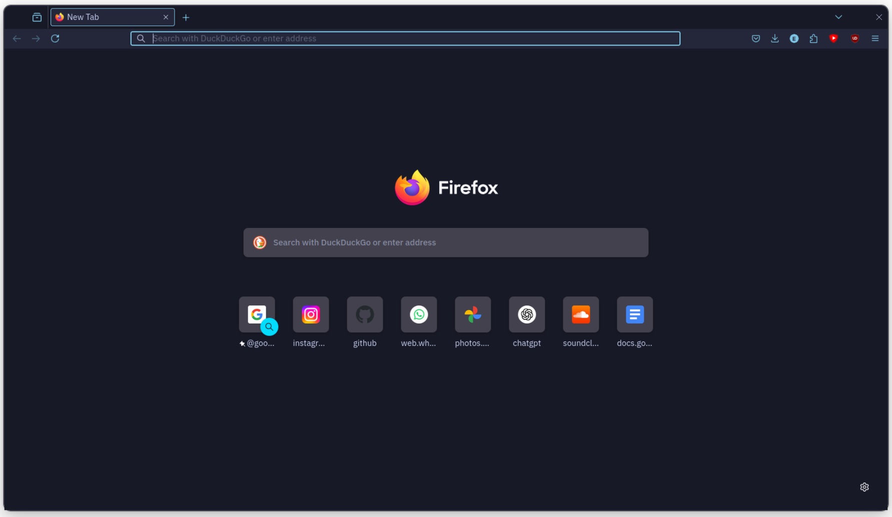


Figure 5.2: A screenshot of the homescreen of the Firefox Browser.

They do several things:

- Loads webpages at URLs.
- Keeps a history of all the website the user visited.
- They allow the user to navigate forward and backward.
- They make use of cookies.
- They support hyperlinks, which are links to other webpages/locations on the document.
- Data is stored inside of a cache.

and more.

[2] All TLS-compatible software are compatible with SSL, as TLS was designed to be compatible with SSL

5.1.5 Loading Website Data

In order to load HTML data from servers, it must find the IP address of the server first. However, website names are typed into the browser in their domain form, and not the IP address form. It must go through a **DNS** (Domain Name Server) first, which is (essentially) a large lookup table between domains and IP addresses. The typical procedure for your computer to load data from a server looks as follows:

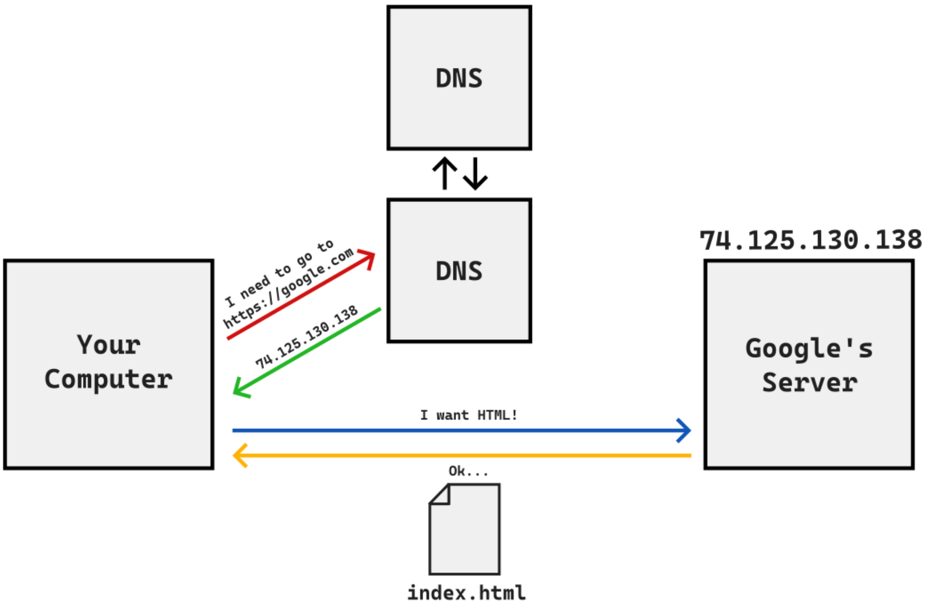


Figure 5.3: How a DNS works.

1. Your computer requests the DNS for the IP address of the URL, in this case Google.com.
2. The DNS may communicate with other nearby DNSes, and returns to the computer the IP address that your computer needs to load the resource.
3. Your computer then requests that webpage from the server, in this case Google.
4. The server then responds with the corresponding data.

5.1.6 Cookies

These are like blobs used to store tempoary, small bits of information that relate to a website and its features.

In summary, cookies:

- Allows websites to remember user data, including emails and payment information, so that they don't have to reinsert it into the website every time it's needed.

- Are used as memory for webpages, like a webpage's memory on the user's computer used all for itself.
- Can track internet habits and web histories, along with other personal information.
- Can be used in targeted advertising.
- Stores user preferences (like dark mode, etc.)

There are 2 main kinds of cookies, and they are as follows:

Session Cookies

These are the most common type of cookie. These don't have an expiry date attached to it, and they are stored in RAM a not to the disk. Hence, they only exist when the browser ies left open.

An example of session cookies use would be the shopping basket on a shopping site like Lazada. When you add items to the cart, it stores the contents of the cart in a cookie, so that when you relaod the page or visit other items, the cart is not emptied. They do not actually collect any information about the computer and do not identify the user.

Persistent Cookies

Unlike session cookies, these persist by being stored on disk. These are usually used to store website settings, like making ChatGPT stay in dark mode. These cookies have an expiry date. When the expiry date of a persistent cookie is reached, or if the user deletes it, only then is it cleared.

These are used for:

- Saving personal information, like credit cards
- Tracking your preferences on websites
- Storing login details

And others. Persistent cookies allows the web server to not store as much data as they usually would need to, as things like login details can be stored on the user's computer and can be loaded every time the site is visited.

Since these remain on disk, they can be used to **target users** by sending these cookies to other websites for advertising. They can also track your browser history and other personal data. Hackers can target persistent cookies on disk to breach your data, as well.

Loading Cookies

1. Every time a page is loaded by your web browser, it looks for any cookies.
2. When it finds cookies, it loads both the persistent cookies stored on disk and session cookies which are stored in RAM (tempoary).
3. When the website is closed, the persistent cookie is backed up on secondary storage.
4. When the web browser itself is closed, the session cookies are deleted from memory.

For more details, see page 185 of the textbook.

5.2 Digital Currency

stub

5.3 Cybersecurity

stub

Chapter 6

Emerging Technologies

6.1 Automated Systems

Automated systems are a mix of software and hardware that control things or do tasks that have to do with the real world. They are mainly designed to work without a person needing to operate it.

A recap on sensors

- **Temperature Sensors** which measures the surrounding temperature
- **Humidity Sensors** which measures humidity
- **Acoustic Sensors** which detects sound, e.g. the sound of a thief breaking into a museum
- **Pressure Sensors** which measures pressure and can be used, for example, to check if something has been pressed
- **Active Infrared Sensors** which detects if there is any obstruction by sending and receiving an IR signal to and from the receiver respectively.
- **Passive Infrared Sensors** which detect IR heat radiation. All objects that are hot in terms of temperature all emit some amount of Infrared light
- **Gyroscopes**[1] detects the orientation of an object in 3D space (i.e. phones, cars, etc.).
- **Accelerometers** detects the acceleration of an object in 2D space (i.e. phones, cars, etc.). Almost all modern gyroscopes can act as accelerometers.

The typical automated system involves:

- Sensors constantly provide readings

[1] Many types, 3 rotating wheels are the simplest. MEMS (vibrating structure) used in phones.

- Programs (run on the microprocessor[2] of the device and processes the readings. If the data is analog, it invokes an ADC[3]
- The program either logs it or invokes an actuator[4] to reflect something.

Let's think of a common automated system in many of our houses, a Roomba or an autonomous vacuum cleaner.

- The sensors in the Roomba, i.e., proximity sensors, gyroscopes, light sensors, etc. sense for obstacles and sudden jolts (i.e., picked up by human or violently kicked by dog) and constantly feed the device with data
- The Program running on the CPU likely runs the following code:
  - If there is an obstacle in front, turn some number of degrees until nothing is in front
  - If there is a sudden jolt detected, do not power on the motor, and do not power on the vacuum.
  - Otherwise, power on the motor and continue sucking the floor for dust.
- The actuators include the motors and the vacuum pump that sucks up dust.

6.1.1 Applications

Automated systems have a lot of applications (Taken from textbook and syllabus).

- **Agriculture**
- **Transport**
- **Industry**
- **Video Games** (Helping gamers with their workflows?)
- **Home Decoration and tools** (Roomba's? Smart LED lights or bulbs? Smart home devices?)

6.1.2 Advantages of Automated Systems

- **Automated systems tend to be a lot faster** than a human in terms of response time for necessary actions
- **Automated systems tend to be safer** as it is more likely to respond to critical events, therefore sparing humans from the hassle

[2] Like a CPU, it is an integrated circuit on a single chip

[3] An analog to digital converter, this converts continuous analog data with an infinite decimal range to binary 1's and 0's with a finite and defined range, so a computer can make sense of it.

[4] Another term tested in the previous examination, these are devices that can be controlled via a program that change the surrounding environment; like a solenoid that pushes things around, or a (bigger) piston, pump, valve, etc.

- **Automated systems tend to be more productive** as they are efficient and specialized to do one thing, therefore typically making less errors than humans do
- **Automated systems tend to be more consistent** as they are usually programmed to do the exact same thing over and over again, meaning that the outcomes reflect it
- **Automated systems tend to use less resources** as they are programmed specifically to use exact amounts of inputs to produce outputs, with fewer chances of error.

6.1.3 Disadvantages of Automated Systems

- **They are expensive to set up** as they require a lot of human effort to set up, program, and maintain
- **They must be thoroughly tested**
- **Bugs in the system can cause it to stall**, if the developers were not careful. This can do things like halt production in a factory-related setting.
- **They can be maintenance-intensive** as these computerized systems tend to cause issues if they are not being watched to a degree, with necessary actions being taken on the fly
- **Automated systems that connect to networks are subject to cyber-attacks**, which may halt critical processes.

In call cases, automated systems can help the vast majority of people in these industries speed up boring and repetitive tasks. More info can be found in the textbook (pages 218 229 all contain examples).

6.2 Robotics

In call cases, automated systems can help the vast majority of people in these industries speed up boring and repetitive tasks. More info can be found in the textbook (pages 218 229 all contain examples).

6.2.1 Applications of Robotics

Robots can be found in a few industries (some taken from textbook). Please do not worry about memorizing all of them, The syllabus simply requires you all to know that robots could be applied in these areas, not know the exact details. They are provided for further information/higher mark questions that require elaboration.

- Industrial Purposes, like putting together parts of a car or managing warehouse stock, or labelling plastic bottles.

- In the home, like roombas.
- Military, like drones that can detect enemy planes
- Transportation, like autonomous vehicles, or e-concierges
- Agriculture, to harvest crops and so on
- Medicine, like surgery robots

6.2.2 Characteristics

- They can **sense their surroundings**:
  - Sensors (see section 3.3.3) are used for this.
  - These sensors lets robots detect things about their environment. It helps them detect things like the shapes and sizes of objects, their masses, how far away they are, how hot/cold they are, what shape they are, etc.
- They **have moving components**:
  - Robots are typically used to automate repetitive tasks.
  - Since these repetitive tasks may contain picking things up and/or moving these things around, things such as solenoids, hydraulic pumps and systems, motors, wheels, cogs, etc. to help these computerized parts move.
  - (Taken from textbook) They can have different attachments to help them carry out specific tasks, such as welding, spraying, cutting, lifting, among others.
- They **can be programmed**:
  - With code written in a programming language, a device known as a *microcontroller* can be controlled. Nowadays, a project called MicroPython allows the simple and versatile Python programming language to be on these microcontrollers.
  - These microcontrollers typically have general-purpose IO pins, or GPIO pins[5]. They typically contain serial interfaces or PWM[6][7] interfaces which can be controlled by Python code that directly talk to the components, such as robotic arms or motors, to do real world tasks. All microcontrollers have a microprocessor, which is a type of integrated circuit on a single chip.

6.2.3 Important Notes

(Headings are taken directly from the textbook).

[5] Not necessarily needed for your exam.

[6] Pulse-width Modulation. Not necessarily required for the exam itself.

[7] This allows components, such as bulbs or motors, to change their intensity by varying the overall voltage they receive. This technology involves quickly turning on and off a steady supply of voltage to change the overall amount of voltage it receives.

- Robots and AI are not the same thing! AI tends to possess more human-like intelligence features, however robots do not strictly include them. However, AI systems may be used to control a robot, in some cases.
- Do not confuse software robots and hardware robots! Hardware robots have things like pistons, solenoids, motors, gears, etc. and interact with the physical world through a microcontroller of some sort. A software robot can also be a thing that does a repetitive task, but it does not interact with the physical world, however more the online world. Software robots may include (Examples from textbook):
  - Web Crawlers, or as the textbook states; WebCrawlers, which automatically "surfs" the internet and collects information about web pages.
  - Chatbots, which seem to have a human-like conversation with you. However, they have mostly been phased out by LLMs, see section 6.3.

6.3 Artificial Intelligence (AI)

Note: AI shills and knowledgeable people, some of this information may be completely outdated or grossly oversimplified; this is simply what the syllabus requires us to know.

Artificial intelligence is a branch of computer science and a popular field of research that goes into how computers can simulate the "intelligence" of Human beings. A large portion of it is to let these systems adopt the mental process of acquiring, synthesizing and rephrasing knowledge in a human language for humans to interact with.

These systems all occur in humans, but through A LOT of statistics[8] they can be replicated by a machine to an extent. They are benchmarked against things like reasoning, Language generation, sight, among others; Despite how AIs cannot truly replicate these human actions and reason for themselves[9].

6.3.1 Characteristics

AI, at its core, is simply a collection of data and rules, generated from a dataset (one word). However, different types of AI systems make use of said data in different ways, but they are all processed, synthesized through a machine algorithm that mimics that of humans; and regurgitates it.

All you must know is that they can be split into 3 categories:

- **Narrow AI** are AI systems that specialize into one narrow area, and typically perform better than humans in that area.

[8] AI is simply applied statistics, not black magic. Ask your math teacher (who may ask a smarter math teacher), it heavily uses a concept known as regression (estimating relationships) and a lot of calculus.

[9] This may change in the future, however the best LLM technology like ChatGPT 4o and o1 all rely on transforming the inputs given directly; all the "reasoning" it has is derived from a human's natural reasoning that the LLM simply re-spits.

- **General AI** are AI systems that performs similarly but not superior to a human in genral cases. They typically cover broader ranges of operations, but are not as good as real professionals.
- **Strong AI** is when a machine has equal or better performance than a real human in a broad range of tasks. However, it has not been reached yet.

6.3.2 Reasoning

This is a process in AI systems where a system provides/generates conclusions on a set of data.

It is developed by feeding an AI system a set of factually correct data[10]. In the context of a kitchen, a chef learns in culinary school details on how to cook the best-tasting plate of fried rice. All these facts are provided to the chef, and after enough training, the chef is able to learn the information and then apply it to be able to make the perfect plate of stir-fried noodles, or maybe even pasta. The chef would then naturally change its techniques to adapt it to the desired result.

Through this training, AI can deduce facts and reason that if these facts hold true for creating this outcome, these similar facts should be true for creating a similar outcome. It can quickly do so as these systems tend to be specialized in doing that specifically, and then make relatively factual predictions based on the factual inputs. This is how an AI system can adapt.[11]

6.3.3 Examples

- Smart home assistants, such as Apple's Siri, Google's Assistant, and Microsoft's now-discontinued Cortana. The program interacts with the user by using speech-to-text models[12] and then analyses the patterns in them to provide a result.
- Translation systems such as Google Translate, DeepL, etc.
- Math-solving AIs such as Wolfram Alpha, MathPile, or word problems through OpenAI for natural language processing (commonspeak).
- Large Language Models, or LLMs, like ChatGPT, Claude, LLaMa, etc. These are large transformer AI systems that take text as input and are trained[13] on those data points[14], and then are transformed based on a prompt that generates text.

[10] A big problem in AI is that bias always exists. There is no way to get a large, diverse set of perfectly factual data. If the example of the kitchen and chef cooking fried rice is read, maybe the instructor of the culinary school really likes the food spicy, so the chef ends up making all his dishes spicy.

[11] Further details are not provided as it is out of the syllabus.

[12] These are AI systems that detect common speech patterns and correlates them to text, therefore translating phrases and words that are spoken to text form.

[13] Through building general knowledge, followed by fine-tuning to specialize into a certain field.

[14] Including supervised learning, self-supervised learning, and reinforcement learning.

6.3.4 Expert Systems

These are developed to mimic human knowledge and experiences, and uses them to solve problems that would normally require a human expert. The expert system will produce a conclusion and may output the probability of accuracy of its conclusion. As of current, they are outdated and have largely been replaced by Machine Learning (see section 6.3.5.)

There are two parts to an expert system, the user interface which is a portion of code for the user to interact with (usually the explanation system), and the actual functional portion (the part that does the work), consisting of the inference engine, rules base, and the knowledge base.

Knowledge Base

- Is a collection of facts accepted from multiple experts and expert resources in the chosen field.
- Represents data through objects and their attributes. For example, the object 'Dog' would contain the following attributes: 4 legs, 2 eyes, 50 km/s running speed, is fluffy, swims, lives on land.

Object	Attr. 1	Attr. 2	Attr. 3	Attr. 4	Attr. 5	Attr. 6	Attr. 7
Dog	4 legs	2 eyes	50km/h speed	Fluffy Weapon	Can swim	Lives on Land	Mammal
Cat	4 legs	2 eyes	32km/h speed	Weapon	Can swim	Lives on Land	Mammal
Salmon	0 legs	2 eyes	32km/h speed	It's just a fish!	Can swim	Lives in Water	Just a Fish

...and many more objects. Note that "Attr." denotes attribute.

Rules Base

- It is a collection of rules to analyze the knowledge base and to solve problems.
- The inference engine uses it to make conclusions, which can then be used to solve problems or answer the user's questions
- The rules base contains logic that the inference engine uses to draw conclusions. E.g. IF moving = false AND breathing = false THEN status = "Dead". Logical methods like these are then evaluated by the inference engine to evaluate if status = "Dead" or "Alive"

Inference Engine

- It is the main, problem-solving element of expert systems.
- It uses the logic stored in the *rules base* to identify and answer user queries with the most relevant info stored in the *knowledge base*.
- One may compare it as working like a search engine on the *knowledge base*.
- The inference engine tries to find the object in the knowledge base that best matches the user input, with the rules base being used to provide the search logic.

Explanation System

It is the part of the expert system that explains to the user the reasoning it generated, and draws conclusions/recommendations.

Overall Structure

The user interface, explanation system and the inference engine are considered to be in the expert system shell, since it is the layer that handles interaction between the user and the data. The knowledge base and the rules base do not handle any part of the interaction, but is used by the inference engine as it handles the user interaction. The flow of data generally follows: Input screen → Expert system → Output Screen.

For the expert system to find the object from the above table that a user is interested in, it could ask questions on almost every attribute. Though, with repeating values in certain rows, we can optimize the number of questions we ask the user. Attribute 2 — the number of eyes – stays the same for all the rows in our table. So the questions may be the type of creature (Mammal or fish), and if it can swim. Extra questions may be added to aid the user if they know different information about the object, such as the speed or if it ”is a fluffy weapon”. This is done to allow multiple paths to reach to a conclusion. (Accessibility purposes)

Setting up Expert Systems

- Information is gathered through human sources or written sources such as textbooks, research papers or the internet
- The above information fills the knowledge base in the object-attribute format
- A rules base needs to be created to analyze user queries against the knowledge base
- User interface needs to be developed to allow the user and expert system to communicate

- Once set up, it needs to be fully tested by running the system with fully known outcomes, so results can be compared and changes made to the expert system.

6.3.5 Machine Learning

Machine learning is another type of AI, similar to expert systems, in that it is *a technique for machines to replicate human behavior*. Unlike expert systems, which uses relatively simple conditionals to make decisions based on a small database, machine learning develops a *neural network*. It is done by feeding the training program a lot of data and letting the computer categorize things based on patterns.

This is like letting the computer train from past experiences in order to predict certain future events and to take decisions from previous scenarios. Due to a neural network being built that contains all this data stored logically, they respond quickly and are quite efficient when trained.

Here's an example from the textbook: Consider a search engine, like Google. This is how your typical searches go:

1. Open your search engine and enter your search term
2. Search engine presents you with multiple pages of information
3. You look through the options on the first page, and when you reach the end, you go to the second one, so on.

However, these pages may not be presented in *the most efficient way*, they may just be a random assortment of links. However, we can use machine learnin g here.

We could build a neural network to recognize the relationship between a search term and search results. Given a completely random set of results; the URL that the user chooses to visit can be stored along with the search term, creating a correlation that this search term is related to this URL more **strongly than others**. Here's an example case:

1. User wants to know about dog toys and therefore searches "top 10 dog toys".
2. The newly-built search engine presents all the articles it sees fit, in the order it best sees fit.
3. When the user clicks an article, like "The Best Dog Toys for Adult Dogs!", the search engine will correlate that search term, "top 10 dog toys", with the article, like "The Best Dog Toys for Adult Dogs". It will create a link between the articles with a **weight**, basically dictating how strong the correlation is.
4. As more and more people search for dog toys and choose the articles they are interested in, the data on correlation between search terms and articles will get more accurate and broad. These weights are reinforced, naturally placing the most relevant article (largest weight) as the first option.

Extra Information: This is a form of *reinforcement learning*, where the search engine learns from user activity by clicking websites. Each time the user clicks an article, the model correlates the article and the term closer, almost like a reward for the computer as it knows the correlation further. **You will not need to know the different types of ML yet.**[15]

[15] New for 2027 batch IB CS kids: well ML is a topic in IB CS now :P