Basic Computer Science Concepts Explained

Intro

Computers make no sense. Throw some metal in a box and boom [monke]. What the heck is going on here?

Inside your PC is a Central Processing Unit, or CPU. It's basically just a piece of silicon with billions of microscopic switches called transistors. Depending on the flow of electricity, they can be on or off, kind of like a light bulb, which gives us two states: 1 and 0. The value at one of these switches is called a "bit". One bit by itself doesn't really do much. But put them together, and magic starts to happen. A group of 8 bits is called a "byte" and can have 256 different combinations of 0s and 1s. Congratulations! We can now store information by counting in a system called "binary".

Binary

Every bit represents a power of 2, 1 meaning the power is included and 0 meaning it's not, so this number has 1 times 64, 1 times 4, and 1 times 1, which adds up to 69. This is nice, but for humans, hexadecimal is even better: It's often denoted by this 0x and is simply a more readable format than binary: Four binary bits can take any value from 0 to 15. Hexadecimal uses 0-9 and a-f to represent those values, so a group of four bits can be replaced by one hexadecimal digit.

Logic Gates

Now that we can store numbers, we just need computers to actually, you know, do something with them. Using transistors, you can make logic gates, which are electronic circuits that encapsulate logical statements. You can think of it as a lightbulb with two switches, where the light only turns on under certain conditions. For example, only if A AND B are on. By combining logic gates in a clever way, you can build circuits that perform calculations according to Boolean algebra, which is a system formalizing mathematical operations in binary.

ASCII

Even though computers understand 0s and 1s, for humans, it's not really all that useful. So, using a character encoding like ASCII, we can assign a binary number to each character. When you type an A on your keyboard, it gets translated into this binary code, and as soon as the computer sees this, it says: "Ah yes, that is a capital A.", and slaps it on the screen.

Operating System Kernel

How these devices fit together is handled by an operating system kernel, like Windows, Linux or Mac, which sits between computer hardware and applications and manages how they all work together, for example with device drivers.

Machine Code

Input devices allow you to give the computer instructions with the press of a button, but at the lowest level, computers only understand instructions in machine code, which is binary code telling the CPU what to do, and which data to use.

RAM

When it comes to following these instructions, the CPU is kind of like a genius, just with the memory of a demented goldfish. It can handle any instructions but it cannot store any data, so it's only really useful with random access memory or RAM. You can imagine it like a grid, where every box can hold one byte of information, which can be data or instructions, and has an address, so the CPU can access it in four steps: Fetch from memory, decode instructions and data and finally, execute and store the result. This is one machine cycle.

Fetch-Execute Cycle

Since a program is basically just a list of instructions in memory, to run it, the CPU executes them one by one in machine cycles until it's complete. Oh yeah did I mention that this happens, like, really fast? Modern CPUs can do billions of cycles every second, which are coordinated and synchronized by a clock generator. The speed of this clock is measured in GHz, and people often overclock their CPUs to improve performance, which is nice, but might just set your PC on fire.

CPU

What's ever crazier though, is that a CPU has multiple cores, which can all execute different instructions in parallel, so at the same time. Each core can be split into multiple threads, which also allows every single core to handle multiple instructions concurrently, so switch between them really quickly.

Shell

It doesn't matter how powerful a computer is if you have no way to give it instructions in the first place. Typing machine code by hand would probably make you go insane, but luckily, you don't have to: The kernel is wrapped in a shell, which is just a program that exposes the kernel to the user, allowing for simple instructions in a command line interface with text inputs.

Programming Languages

The best way to make a computer do something useful is with a programming language, which uses abstraction, so that instead of this, you can write code that looks like this, which is then converted into machine code for you. Some languages like Python use an interpreter, which directly tries to execute the source code line by line. Other languages like C or GO use a compiler, which converts the entire program into machine code, before putting it in a file the CPU can execute.

Variables & Data Types

The most basic way to use data is with variables, which assigns a value to a name, which can then be reused and modified. Depending on the value, variables can have different data types. For text, there's single characters and strings of multiple characters. For numbers, there's integers, which can also be signed if they're negative, and floating point numbers for decimal values. They're called floating point, because the decimal point can float around to trade off precision with range. This is possible because they use scientific notation. It's some number times a power telling you where to put the decimal point, which is exactly what they look like under the hood. The actual number is stored with binary fractions. Some fractions like 1/3 can only be approximated in binary with an infinite sum, but, since memory is not infinite, you have to cut it off at some point, which leads to rounding errors, causing pretty weird calculations, sometimes. If these are not enough, long and double use twice the amount of memory to double the range of ints and floats. Some languages like Python automatically figure out which type a variable is, but in a language like C, you have to explicitly declare the type of a variable.

Pointers

The value of a variable is stored at some address in memory. Pointers are variables whose value is the memory address of another variable, which is denoted by this ampersand. So really, a pointer is just some chunk of memory, pointing to another chunk of memory. Since a memory address is just a number, you can add and subtract from it to navigate through individual bytes of memory. This is called "pointer arithmetic".

Memory Management

In some low-level languages like C, you have to manually allocate and free up memory once it's no longer used. This all happens in the heap, which is a part of memory that can dynamically grow and shrink as the program demands, which allows for more control but makes in incredibly easy to completely break your code. You could touch memory you're not supposed to, or that simply doesn't exist, which is known as a "segmentation fault". But also, if there's some chunk of memory that's no longer used, and you forget to free it or you have no way to access it anymore, that memory is no longer usable. This is called a "memory leak" and will make the program slow down and eventually crash. To avoid this mess, high level languages like Python have built in garbage collectors that manage memory for you. Different data types take up a different amount of memory. Integers are most often 4 bytes of memory. A single character is most often one byte of memory, and a string is just multiple character bytes, with a "NUL" character to signal the end of the string. Float has 4, and double has 8 bytes of memory.

Big O Notation

Now that we have data and memory management down, let's do something useful with it! To design programs that are efficient, we have to think about their time and space complexity. Analyzing time complexity means estimating the number of steps a program takes to run, and space complexity is about the amount of memory it requires. This is most often done with Big O notation. It simply describes the upper bound of how fast a program is expected to grow in terms of running time or memory consumption. For example, say we have a list of numbers. If we want to check if a number is in that list, we have to look at each number, so this program runs in O of N time, or linear time, where N is the number of items in the list. Because the number of checks scales linearly with the size of the list. But if the list is sorted, we can use binary search to cut the search space in half with every step, so it runs in logarithmic time, O of log N, which is way faster. But there's also factorial time, O of N factorial, which is very slow, because you have to perform every possible permutation of the input. It gets very slow, very quickly. To figure out the running time of a program, we'll often count the amount of operations and represent that with Big O notation. In this program, to find a number in a sorted list of size 8, we only need 3 steps. 8 = 2^3, so log 2 of 8 is 3, O of log N.

Control Structures

But just counting operations will only get you so far. To write interesting programs, you need control structures to, well, control the flow of your program. The most basic one is a conditional, also known as "if statements", which basically states: if this is true, do that. If not, do something else. When you have multiple conditions, there's "else if statements". All this is known as "branching".

Loops

Another type of control structure is a loop. Instead of running code just once, they repeat some code until a condition is met. The most common loops are for loops and while loops. A for loop repeats a chunk of code for a given amount of steps, and a while loop repeats a chunk of code until a condition is met.

Functions

For large programs, you'll often end up writing similar code multiple times. Instead of duplicating it, you can put that code in a function and call it by name. Functions take inputs, do some computations, and return the result. Instead of putting all your code in one huge function, it's often better to split it up into smaller, more manageable chunks. This process is called "modular programming".

Data Structures

To manipulate data, you need data structures. A data structure is just a way of organizing and storing data, to make it easier to work with. The simplest one is an array, which is a fixed-size list of elements of the same type. They're stored in contiguous memory, so you can access any element in constant time, O of 1. But inserting or deleting an element in the middle is slow, because you have to shift the other elements. That's O of N. To solve this, there's linked lists, which store elements in separate objects, with a reference to the next element. This makes inserting and deleting elements in the middle faster, O of 1, but accessing elements is slower, because you have to follow the references. That's O of N. There's also stacks, which are LIFO, or "last in, first out", so you can only access the last element. And queues, which are FIFO, or "first in, first out", so you can only access the first element. And hash tables, which store key-value pairs in an array, where the key is hashed to an index. This allows for fast access, O of 1, but can be slow if there are hash collisions, which is when multiple keys are hashed to the same index. And trees, which are hierarchical structures, where each element has a parent and children. And graphs, which are like trees, but with cycles, where an element can have multiple parents. Different data structures have different strengths and weaknesses, so it's important to choose the right one for the job.

Algorithms

To manipulate data structures, you need algorithms. An algorithm is just a step-by-step procedure for solving a problem. Some of the most common algorithms are sorting algorithms, which sort elements in a list. The simplest one is bubble sort, which repeatedly swaps adjacent elements that are out of order. It's easy to understand, but very slow, O of N squared. A faster one is quicksort, which picks a pivot element, and partitions the list into elements smaller than the pivot, and elements larger than the pivot, and recursively sorts them. It's fast, O of N log N, but can be slow if the pivot is chosen poorly. There's also searching algorithms, like linear search, which checks each element in a list, O of N, and binary search, which only works on sorted lists, and cuts the search space in half with each step, O of log N. And graph algorithms, like breadth-first search, which explores all the neighbors of a node before moving on to the next level, and depth-first search, which explores as far as possible along each branch before backtracking. Different algorithms have different strengths and weaknesses, so it's important to choose the right one for the job.

File Systems

To store data permanently, you need a file system. A file system is just a way of organizing and storing files on a storage device, like a hard drive or SSD. The simplest one is FAT, or "file allocation table", which uses a table to keep track of which blocks of data belong to which files. It's simple, but has limitations, like a maximum file size of 4 GB. A more advanced one is NTFS, or "new technology file system", which uses a master file table to keep track of files, and supports features like file compression and encryption. There's also ext, or "extended file system", which is used by Linux, and APFS, or "Apple file system", which is used by macOS. Different file systems have different strengths and weaknesses, so it's important to choose the right one for the job.

Network Protocols

To communicate with other computers, you need network protocols. A network protocol is just a set of rules for how data is transmitted over a network. The simplest one is Ethernet, which is used for local area networks, or LANs. It uses a protocol called CSMA/CD, or "carrier sense multiple access with collision detection", to avoid data collisions. A more advanced one is TCP/IP, or "transmission control protocol/internet protocol", which is used for the internet. It uses a protocol called TCP to ensure reliable data transmission, and a protocol called IP to route data packets to their destination. There's also HTTP, or "hypertext transfer protocol", which is used for the web, and FTP, or "file transfer protocol", which is used for transferring files. Different network protocols have different strengths and weaknesses, so it's important to choose the right one for the job.

Operating Systems

To manage all of this, you need an operating system. An operating system is just a program that manages computer hardware and software, and provides common services for computer programs. The simplest one is DOS, or "disk operating system", which provides a command line interface for running programs and managing files. A more advanced one is Windows, which provides a graphical user interface, or GUI, for running programs and managing files. There's also macOS, which is used by Apple computers, and Linux, which is used by servers and supercomputers.

Different operating systems have different strengths and weaknesses, so it’s important to choose the right one for the job.

Conclusion

And that’s it! Now you know the basics of computer science. Of course, there’s a lot more to learn, but this should give you a good foundation to build on. Happy coding!