

# Class 01: Abstraction, Universal Computational Devices, Unsigned Integers

## Today's Goal

To learn that

- Computer systems can be understood at different levels of abstraction and what those levels of abstraction are.
- When we look at what calculations we can do with computers (ignoring hardware limits)
- All modern computers do the same kinds of calculations, just more- or less-efficiently.
- We know of no theoretical computer that's more powerful than our current computers (ignoring efficiency).

## Abstraction

- In some sense, computers don't do calculations, they move photons and electrons and other stuff around.
- We think of them as doing calculations because of our interpretation of these activities; we view things from some context and abstract away the parts that aren't part of the context.
- Textbook identifies levels of abstraction and transformations in computer systems:
  - Problem, Algorithm, Language (Program), Machine (Instruction Set Architecture), Microarchitecture, Circuits, Devices.
- Compiler translates high-level program into machine code (a sequence of bits).
- Machine runs machine code that uses particular instructions (from the ISA: Instruction Set Architecture). ISA is interface between software and hardware.
- The ISA instructions themselves can be broken down into sub-operations; this is the microarchitecture of the machine.
- The micro-operations and -operands are implemented using logic and arithmetic circuits (which deal with bits).
- The circuits are implemented using devices (such as transistors, switches) that fiddle with electrons.

---

## Modern Computers are Equivalent

- A 4-operation calculator isn't like a modern computer because you can't typically program it. Even programmable calculators only let you enter in a program; they don't let you create programs as the calculator runs.
- Modern computers are more powerful than 4-op calculators because they can create programs dynamically and execute them. (Compile and run a program.)
- Modern computers differ in speed, power used, heat given off, etc.
- But they're equivalent in the sense that they all do the same kinds of calculations.
  - Given enough time and memory, any computer can simulate another computer.

- E.g., the “Virtual PC” program ran on Power PC Macintoshes and made them simulate Intel hardware, so you could run Windows (just slowly :-).

## Turing Machines

- Alan Turing was a mathematician & cryptologist; he worked on the British Enigma project that broke Nazi codes and made them readable by the Allies.
- He invented what we call “Turing Machines” (TMs).
  - Very simple theoretical computer.
  - Has a tape with a read/write head.
  - You have instructions like “If we’re in state 72 and we see a \$, move the head right one symbol and go to state 53.”
- Unsurprisingly, you can simulate a TM with a modern computer.
- Surprisingly, TMs can simulate modern computers (verrrrry slowly).
  - You can design TMs that do arithmetic on integers and floating-point numbers, manipulate characters, and so on.
  - You could even simulate an Intel PC.
- In some sense, there’s only one TM you’d ever need to build.
  - The Universal TM simulates other TMs: You give it a description of the other TM and the other TM’s data, and the universal machine will do what the other TM would’ve done.

## Universal Computation Devices

- Turing’s Thesis: Every computation that can be done can be done by some TM.
    - This is not formally provable.
    - Since TMs and modern computers are equivalent (can solve the same set of problems), Turing’s Thesis is the same as saying that modern computers can do every conceivable kind of mechanical computation; TMs (and modern computers) are universal computation devices.
    - Sure looks true. Nobody’s found a mechanical, computational operation that a TM (or modern computer) can’t do. Even quantum computers (when they get built) will only solve the same set of problems we can solve today (but an awful lot faster!).
- 

## Unsigned Integers

- Our hardware represents data using bits; we use bits to represent binary numbers.
- Given  $n$  bits, there are  $2^n$  possible bit patterns.
  - We can use them to name  $2^n$  possible items.
  - For unsigned binary integers, we read the  $n$ -bit string as a base 2 number that’s  $\geq 0$ .
    - E.g. for 3 bits: 000, 001, 010, 011, 100, 101, 110, 111 are 0 through 7.
  - The bit positions (right to left) are  $2^0, 2^1, 2^2, \dots, 2^{n-1}$ .
  - So the largest unsigned  $n$ -bit number is 11...1, which represents  $2^{n-1} + 2^{n-2} + \dots + 2^1 + 2^0$ , which is  $2^n - 1$ .