COM SCI M151B Week 1

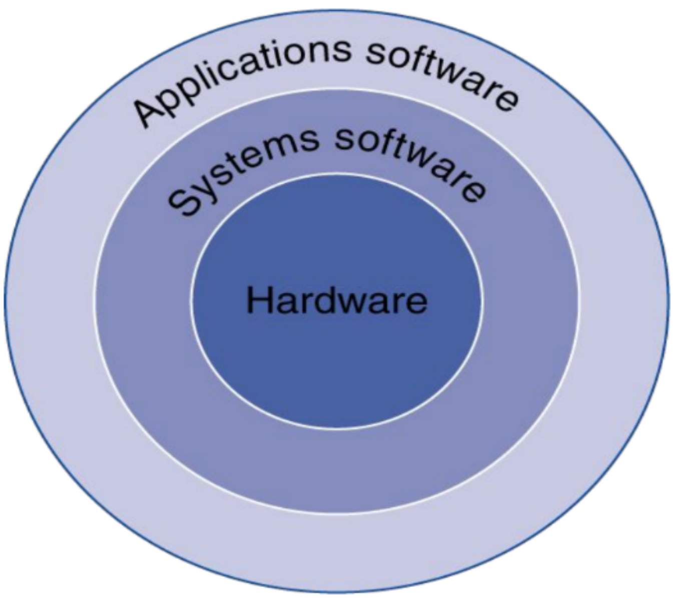
Aidan Jan

September 26, 2024

Using Abstraction

- We see a computer as a *box* with multiple layers of **abstraction**.
- Depending on which layer we want ot work on, we *abstract away* the irrelevant layers.
  - The *main benefit* is that we don't need to know the unnecessary details of the other layers in order to be able to work on our layer.

Computer Abstractions (simplified)



- Application software
  - Translation from algorithm to code
  - Written in high-level language (e.g., C, Java)
- System software
  - Compiler: translates high level language code to machine code
  - Operating system: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks and sharing resources

- Hardware
  - Processor memory
  - I/O controllers

`https://www.youtube.com/watch?v=_y-5nZAbgt4`

How do computers work?

- Theoretical and Historical Points of View
  - Turing machines and history of electronics and computers.

Level of Program Code

- High level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
  - Architecture-dependent
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data

How to maintain compatibility?

There are many different types of computer architecture, and many different languages applications are written in. How do we maintain compatibility?

- System software (OS) and software makes a contract to always give a program with **only a set of known instructions**, and the hardware promises to be able to run that.
- The **ISA** is the *interface* between hardware and software. This allows the HW and SW to change/evolve **independently**.
- Software sees:
  - Function description of hardware:
    1. Storage locations (e.g., memory)
    2. Operations (e.g., add)
- Hardware sees:
  - List of instructions and their order
- In this course, we will use RISC-V ISA.

Goals when designing computers

They must be efficient. What is *efficient*?

- Performance (often most important)
- Power consumption
- Cost
- Reliable and Secure

Past, Present, and Future

- Moore's Law:
  - We started with the ENIAC after World War II, and that computer had a size of 1000 cubic feet, used 125kW of power, only does 2000 additions per second, and had 48 kB of memory.
  - Now, we have computers taking up 1 cubic foot, consumes 250W of power, capable of 20B operations per second, with multiple GB of memory, for only less than 0.01% of the cost of the ENIAC.
  - Moore's Law states that every two years, the number of transistors (and therefore computational power) doubles. This suggests that the cost per transistor reduces each year.
- Dennard's Scaling Law:
  - According to Moore's law, the number of transistors on a chip doubles every two years, thus each transistor's area is reduced by 50%, or every dimension by 0.7x.
  - As a result, voltage is reduced by -30% (0.7x) to keep the electric field constant. $V = EL$
  - L is reduced, thus delays are reduced by -30%. ($x = Vt$)
  - Frequency is increased by +40% ($f = 1/t$)
  - Capacitance is reduced by -30% ($C = kA/L$)
- Scaling Power and Energy

  $$P = CV^2 f$$

  - Power consumption per transistor is decreased by -50%.
  - Power consumption of the entire chip stays the same! Except now it has more transistors.
- Where are these improvements coming from?
  1. Advancements in microelectronics and fabrication technologies.
  2. Advancements in architectural techniques
     - What this course is about!
     - This lead to an improvement by a factor of 25 vs. if we had only relied on (1.)