How a Computer Works

Learning CS with Python Series - Day 1

Let’s Design a Simple Circuit

- Imagine two wires:
  - One on top with voltage (think of this as pressure)
  - One goes to ground
- A resistor hangs from the top wire, connected to a switch then connected to ground
  - So you have two states based on the condition of the switch

Let’s Design a Simple Circuit

- Now replace the physical switch with a transistor
    - A transistor is an automatic/electronic switch that can be turned on or off based on current to one of its three connections
- What we just described is an inverter

But with just that inverter...

- Gates:
  - NOR Gate: Inverter plus one transistor (if either is 1, output 0)
  - OR Gate: NOR Gate + Plus an Inverter (if either is 1, output 1)
  - AND Gate: Inverter + NOR Gate (If both is 1, output 1)
  - XOR Gate: Combination of 4 NOR gates

Storing Bits - Flip-Flop

- Let's say you have two inverters
  - The output of inverter 1 goes into the input of inverter 2
  - The output of inverter 2 goes into the input of inverter 1
  - This is stable, it can "store" a 1 or 0 indefinitely
- Now, stick some gates in-between the two inverters, this allows you to change the value

Example: Performing Addition

- So you want to add two bits:
  - Use an XOR Gate (with some circuitry for the carry)
    - If one of the two bits is 1, then the value is 1
    - If both of the two bits are 0, then the value is 0
    - If both of the two bits are 1, then the value is 0 with a “carry”
  - Connect a bunch of these together using the carry

So, this is all there is?

- Yes! A computer doesn't do much, it just doesn't do much really fast
  - I'm simplifying, but much of the last 40/50 years can be thought of as just making the circuits smaller and cheaper

Transistor Counts

- Intel 8080 -> 4.5 Thousand
- Pentium -> 3.1 Million
- Core i7 -> 2.3 Billion

Machine Language

- So we have some storage (memory)
- We have some basic tasks (such as addition) that can be performed by the hardware
- Now, let’s introduce the concept of a program counter

Program Counter

- The counter starts at the top (line 0) and increments down a memory space (document) line by line
- Each line contains a “word”
- A word is a combination of an action and a memory address
- There is a special register of data called the accumulator

What is a “word”

Op Code

Memory Address

0011 0 1010010001000100

Redirect Bit

Focus on the OP Code

- A four bit OP Code would allow for 16 instructions
- Each of these instructions corresponds to circuits in the processor (just like we talked about with the addition operation)

Example OP Codes

- 0000
- 0001
- 0010
- 0011

What do they do?

- Add
- Subtract
- Load the Accumulator
- Store Accumulator to Memory
- I/O Instructions
- Logic and Flow Control (Skip and Jump)

Assembly Language

- Adds a layer of English on top OP Codes
- No one wants to remember 0000 to load the accumulator. So instead we type:
  - ADD *memory address*
  - SUB *memory address*
  - LDA *memory address*
  - STA *memory address*

Assembly Language

- We then take this document and run it through an “assembler” which turns it into machine language
  - It is a one-to-one relationship, each line corresponds to a binary line of machine language
  - An assembler is the simplest form of “compiler”

Moving to higher languages

- When you compile C, objective-c, Small Talk, whatever, your converting your code into series of machine language “words”
- Unlike assembler, it is a one-to-many relationship. Each line of your code represents many lines of machine code.
- But not all commands are equal. Some operations (like addition) take very few words. Others take many.

So, when I write something in Python, I'm really writing machine code?

- Yes! And you need to be aware of how it translates so you know what is efficient, and what is not.
  - When you store something in a variable, you are really converting it to binary and storing it in a particular address of memory
  - When you do an if statement, you are really using a comparison operation with a jump operation
  - etc

Data types

- Think about it: there is really only one type - integer
- But we can “fake” it for some other types
  - If we group 8 bits together we can represent 255 different things, let’s say we map those to characters of the alphabet
    - Using this method a bunch of “bytes” (8 bit groups), make up a “string”

Data types

- Floats (binary decimal numbers) can be represented by taking 1 bit to represent the sign, some number of bits to represent the exponent (e.g. 8) and the rest to represent the fraction (e.g. 23)