Computers & Programming I

September 11, 2018

This lecture was given from the slides in Bb, so feel free to cross reference. I had minor additional details that he gave in class, which weren’t on the slides. I will highlight them.

Professor revised binary addition and ones and twos compliment. This was a visual demonstration. The video provided below is similar to what he went over. Just primarily focus on one’s and two’s compliments:

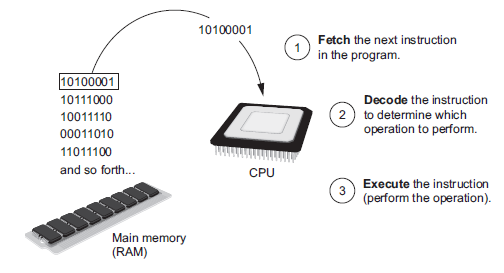
<https://www.youtube.com/watch?v=Xaj8oV8P860>

You just need to know of two’s compliment, not necessarily how to do it.

* We use two’s code to represent negative numbers
* We multiply by 2 when we have fractions.
* Real numbers are encoded using floating point position.
* The most important coding scheme is ASCII
  + ASCII is limited to only 128 characters
* Unicode coding scheme is becoming standard with ASCII
  + It can be used to represent different languages
* Digital describes any device that stores data as binary numbers
  + Digital images are composed of pixels
    - To store images, each pixel is converted to a binary number representing the pixel’s color
  + Digital music is composed of sections called samples
    - To store music, each sample is converted to a binary number

**How A Program Works**

* CPU is designed to perform simple operations on pieces of data
  + reading data, adding, subtracting, multiplying, and dividing numbers
* To carry out meaningful calculation, CPU must perform many operations
* Program must be copied from secondary memory to RAM each time CPU executes it
* CPU executes program in cycle
  + Fetch: read the next instruction from memory into CPU
  + Decode: CPU decodes fetched instructions to determine which operation to form
  + Execute: perform the operation
* Vonn Neumann developed this structure.



**Assembly and Machine Language**

* Assembly language: uses short words (mnemonics) for instructions instead of binary numbers
* Easier for programmers to work with
* Assembler: translates assembly language to machine language for execution by CPU

**High Level Languages**

* Low-level language: close in nature to machine language
  + Example: assembly language
* High-Level language: allows simple creation of powerful and complex programs
  + No need to know how CPU works or write large number of instructions
  + More intuitive to understand

**Compilers and Interpreters**

* Programs written in high-level languages must be translated into machine language to be executed
* Compiler: translates high-level language program into separate machine language program
  + Machine language program can be executed at any time
* Interpreter: translates and executes instructions in high-level language program
  + Used by Python language
  + Interprets one instruction at a time
  + No separate machine language program
* Source code: statements written by programmer
  + Syntax error: prevents code from being translated