PY410 / 505
Computational Physics 1

Salvatore Rappoccio

Programming

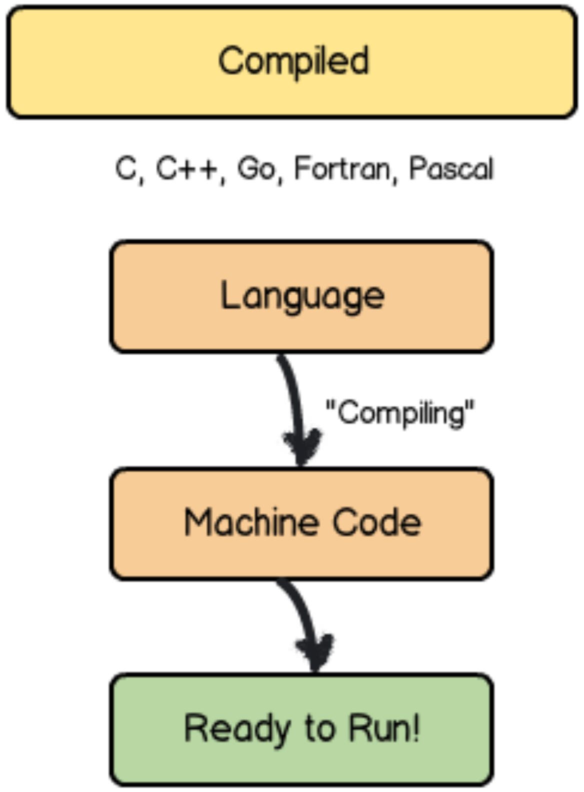
- We will now be learning how to program
- First you will learn C++, because it’s harder and gets you more in touch with the computer itself
- Then we will learn python, because it’s quick and easy with many practical applications

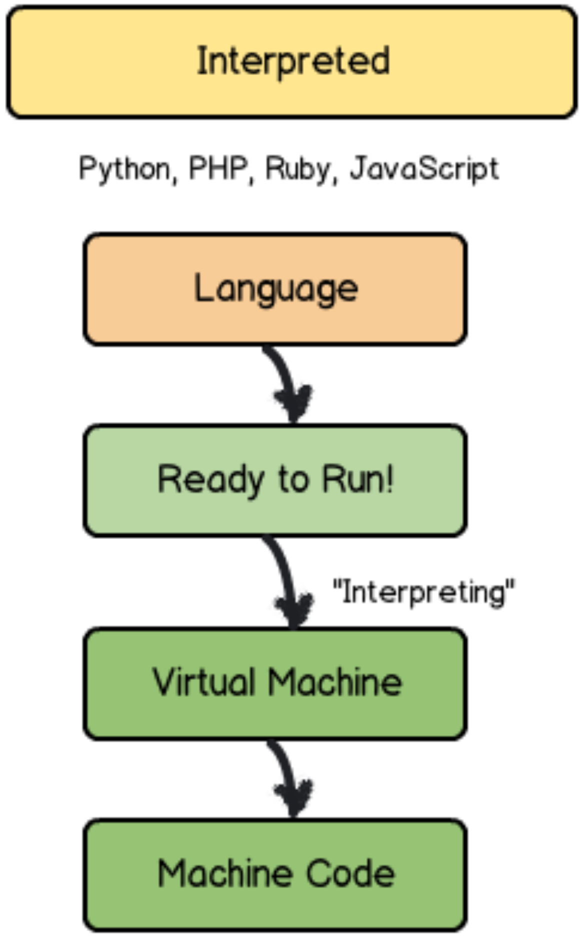
Computers

- When I was a lad, needed to know UNIX to check email
- Now, you just bark the order into your phone
- Somewhere, that order is transferred into currents and voltages on transistors
- How does it happen in between?

Computers

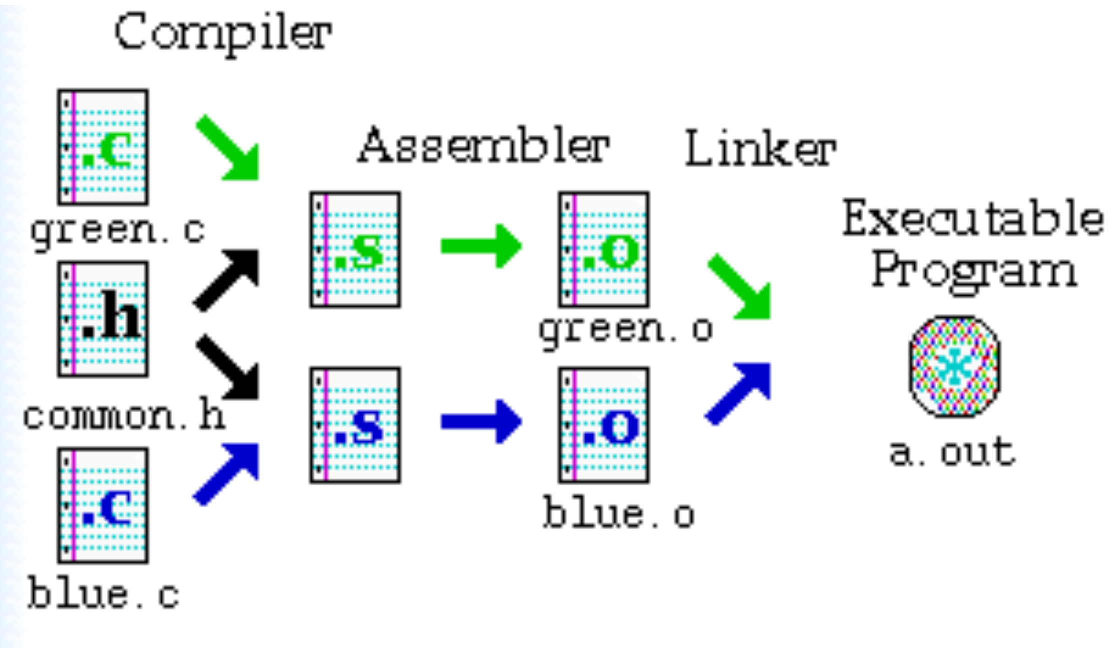
- Your voice is being recognized by software that does a waveform analysis
  - We'll look at the basics of waveform analysis later in the semester
- That software is written in some high level language that translates commands to instructions for your computer chip
- Two main modes :
  - Compiled
  - Interpreted





Computers

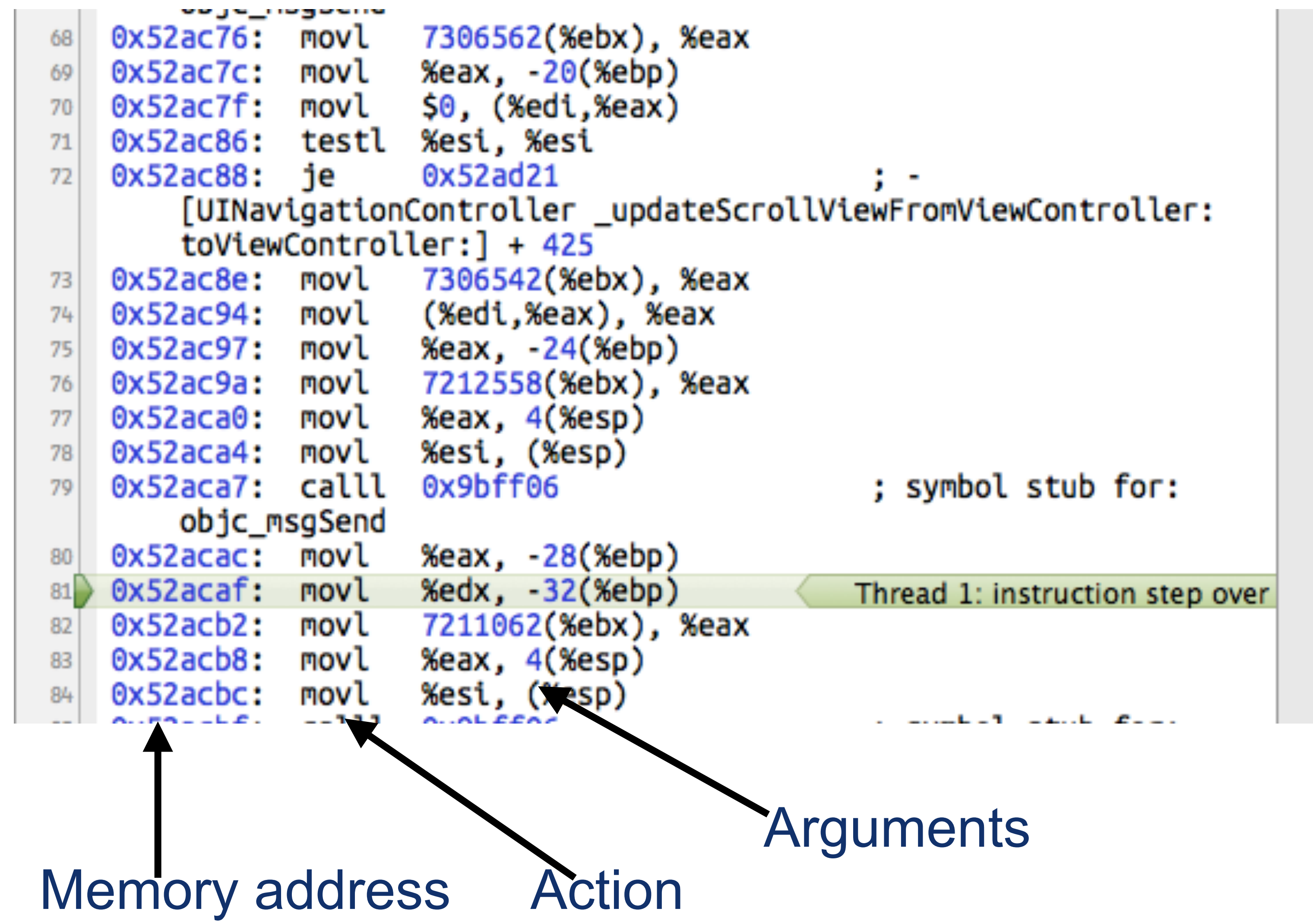
- Compiling in C++
- Code written in C++ is compiled
- Outputs “assembly” code



- Assembly code is linked together
- Then merged into the executable

Computers

- Assembly code is what you need to UNDERSTAND, but not necessarily to WRITE
  - Mnemonic for specific chipset instructions
    - Architecture specific (AMD, Intel, etc)
- Basic instructions
  - “move contents of Register 1 into Register 2”
  - “Add contents of Register 1 and Register 2 together, store in Register 3”
  - THESE are the important issues when concerning yourself with writing code
    - Speed, memory efficiency, etc, primarily impacted



Computers

- So we need to understand the guts of what is going on before we learn to use them correctly
- Everything in the computer is stored in a specific memory location (address)
- It is ultimate a set of dual-state transistors, storing “on” or “off”, interpreted as “one” or “zero” in binary:

...	...
Address 3	11101000
Address 2	00000000
Address 1	10010111
Address 0	01101001

Computers

- Binary is base 2
- If you haven't encountered anything other than base 10, we are going to cover it now

- Instead of place values as $10^0, 10^1, 10^2, 10^3, 10^4, \ldots$
- we have place values $2^0, 2^1, 2^2, 2^3, 2^4, \ldots$

- These are BINARY DIGITS (BI) + (TS)

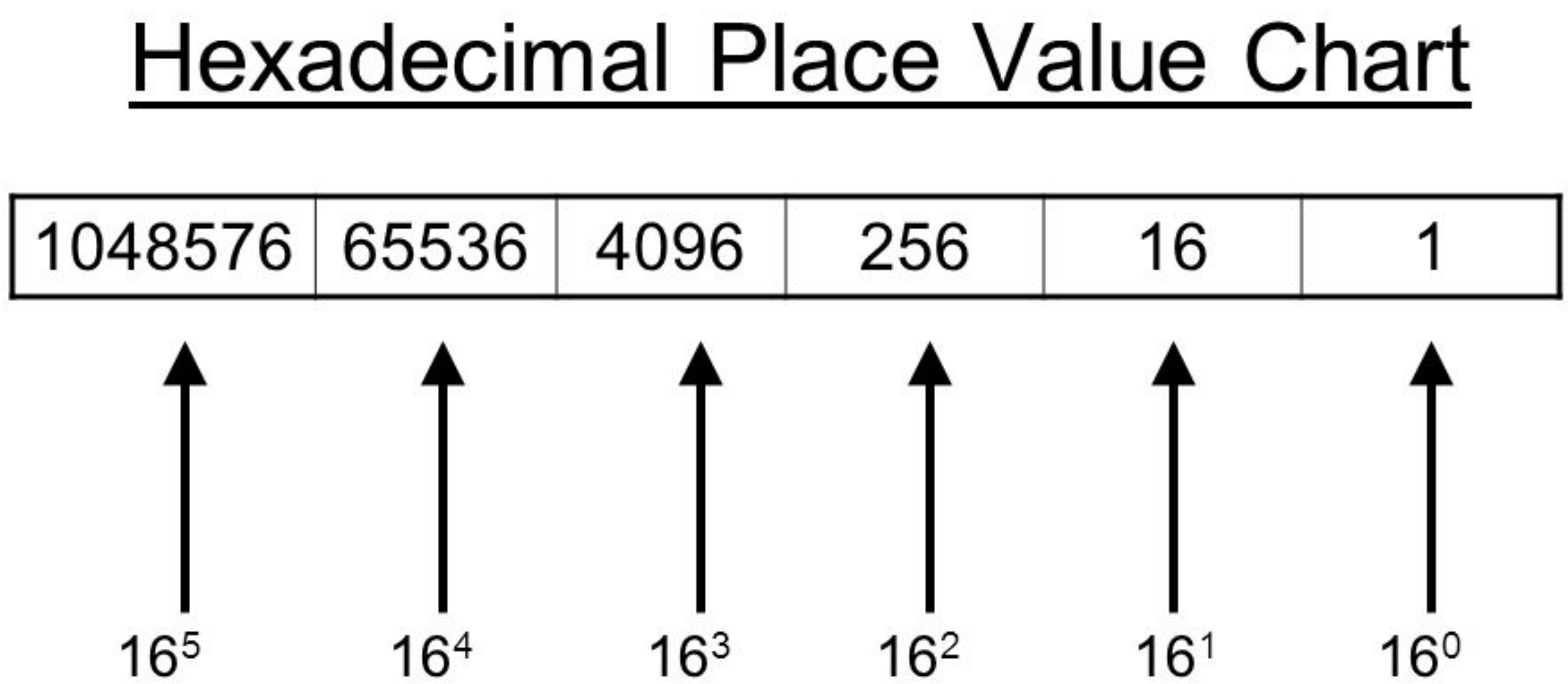
- = Bits

Binary Value 1001b

2^3	2^2	2^1	2^0
8 Eights place	4 Fours place	2 Twos place	1 Ones place
1	**0**	**0**	**1**

Computers

- Of course, writing a LOT of 1's and 0's is pretty cumbersome, so we usually abbreviate binary (preface "0b" in C++14) with base-8 (octal, preface "0") or base-16 (hexadecimal, preface "0x")



- Each set of 4 bits is one hexadecimal number:
- 0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f

Computers

- Examples :

Decimal	Binary	Octal	Hexadecimal
1	“0b1"	“01"	“0x1"
4	“0b100"	“04”	“0x4”
10	“0b1010"	“012"	“0xa”
15	“0b1111”	“017"	“0xf"
16	“0b10000"	“020”	“0x10”
32	“0b100000"	“040”	“0x20"
40	“0b101000"	“050"	“0x28”

Computers

There are
10 types
of people
in the world:

Those who
understand binary,
and those
who don't.

Binary and Hexidecimal

	16^3	16^2	16^1	16^0
1	0	0	0	1
10	0	0	0	a
64	0	0	4	0
66	0	0	4	2
1035		4	0	b

0x1

0xa

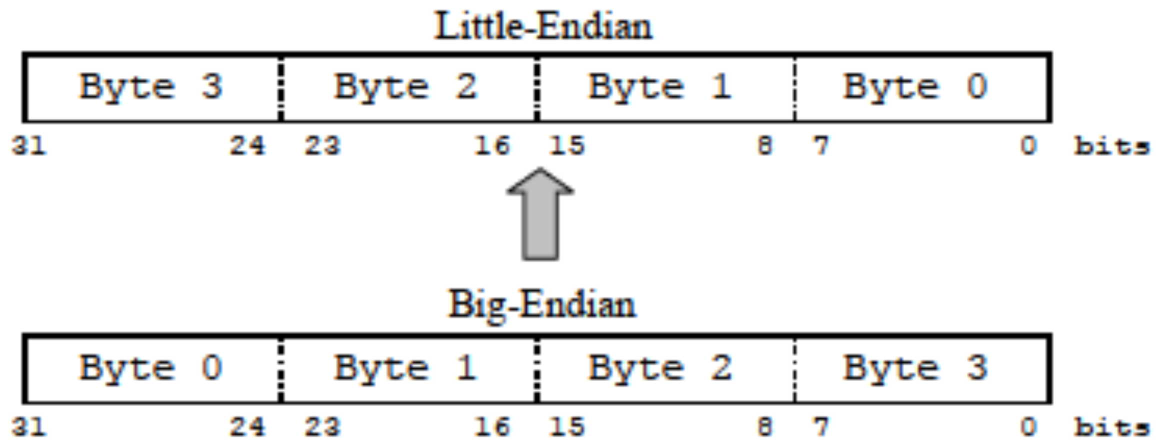
0x40

0x42

Computers

- How to represent the bits on the actual transistors, though?
- Could start from left to right, or from right to left
- Some architectures do one, some the other

- This is which "end" to start from : the "Endian" debate
  - From Gulliver's Travels, as in which end of the soft boiled egg to crack
- "Big end -ian" : most significant digit is first
- "Little end -ian" : least significant digit is first



Computers

- How about NEGATIVE numbers?
- Well, uhh… easy too, right?
- Decimal -2.2 = binary -10.10
- But how to represent that? Need another bit to indicate the sign!
  - Let’s say it’s the first bit, we have to sacrifice one of them:
  - Decimal -2 = binary 110
    - The first “1” counts as a negative sign

Computers

- Great, how about operations on these?
- Decimal 4 + 3 = 7 ===> binary 100 + 011 = 111
- Decimal 4 - 3 = 1 ===> binary 100 - 011 = 001
- Uhh, but shouldn’t “4 - 3” be the same as “4 + (-3”)?
  – Uhhh…
  – Decimal : 4 + (-3) ===> binary 100 + 111
  – Crap. Doesn’t work!
    - 100 + 111 = 1011 binary = 11 decimal, but want this to be 1!

Computers

- Solution : two's complement
  - https://en.wikipedia.org/wiki/Two's_complement

- Two's complement = subtracting the number from 2^N:
  - Example : 10 decimal = 01010 binary
  - Two's complement = 10000 - 01010 = 10110
  - Compare to the "naive" implementation of "1" in the first place plus the number "10" :
  - "naive"   : 1 1010
  - "2's comp": 1 0110

Computers

- Easier way : you flip the bits, then add 1 :

- decimal -10 = binary -01010
- Flip the bits : 10101
- Add one : 10110

- Ta-daaa! Like magic, but it’s math!

Computers

- Arithmetic still works as you'd think in two's complement:

- Look at a 4-bit byte (max is 1111)
- Decimal 10 - 3 = 10 + (-3) = binary 01010 - 00011
- In two's comp : (01010) + (11101) = 100111
  - but only have 4 bits, the first is truncated, so this is 0111 = 7!

- So simple arithmetic still works in two's complement, no need for special instructions for the chip!

Two's Complement

	7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	0	1
5	0	0	0	0	0	1	0	1
-5	1	1	1	1	1	0	1	1
10	0	0	0	0	1	0	1	0
-10	1	1	1	1	0	1	1	0
64	0	1	0	0	0	0	0	0
-64	1	1	0	0	0	0	0	0

- Number = 100000
- Flip the bits : 0111111
- Add 1: 1000000

Computers

- Storage of numbers on computers is done in binary
- Combining some 4-bit registers is called a “byte”
  - Yes, like “gigabyte”.

- How many bits are in a byte?
  - Architecture dependent, but typically 8

- 8-bit byte : 00111100
  - Can represent numbers from (decimal) 0 - 255
  - (Hexadecimal : 0x0 - 0xff)

- How do you store bigger numbers?
  - put bytes together

Computers

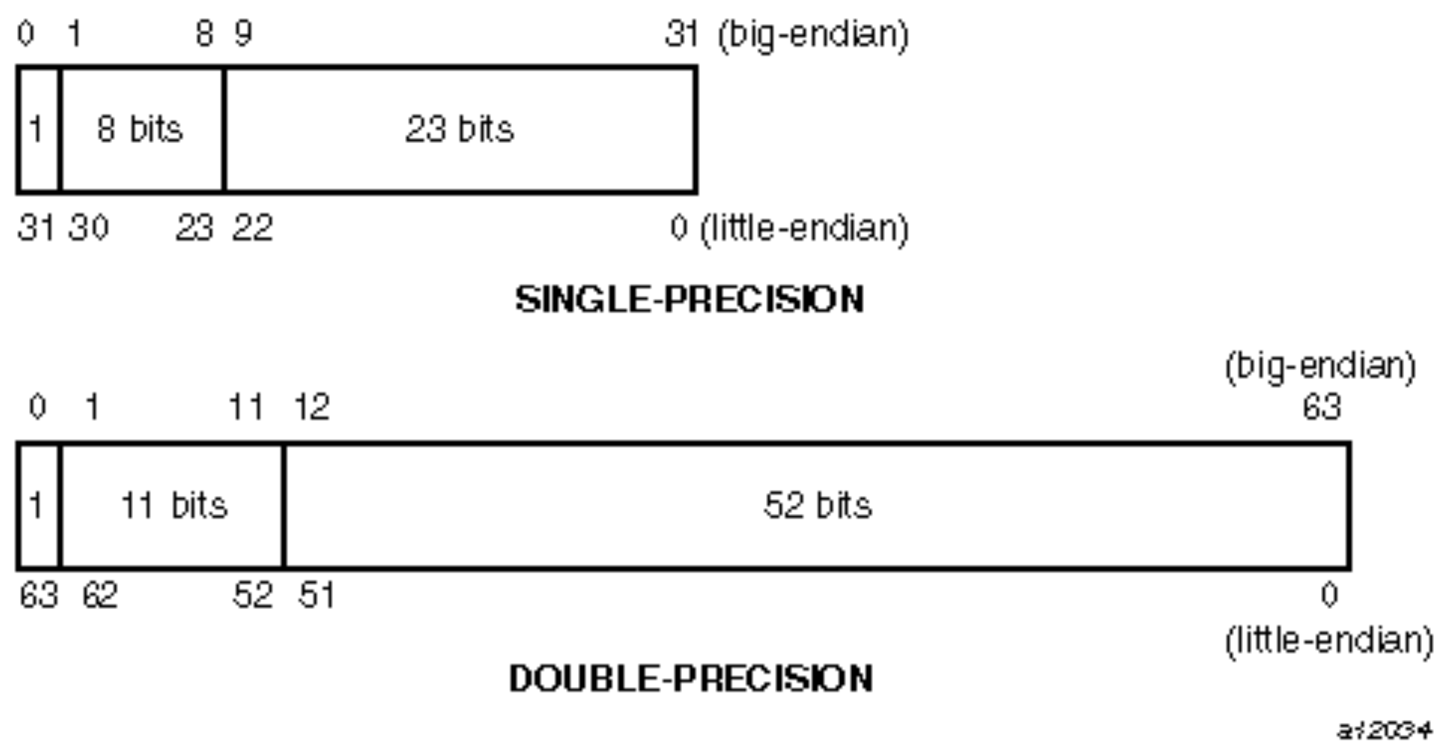
- The range of values you can represent depends on if you need it to be signed or unsigned, integer or floating point:

# Bytes	Signed?	Min	Max
1	Unsigned	0	255
1	Signed	-128	127
2	Unsigned	0	65,535
2	Signed	-32,768	32,767

Computers

- Well, that's great for integers, but what about decimals?
  - Decimal 2.2 = binary 10.10
- But in binary, how do we specify the "decimal place" (binary place?)
- Use scientific notation : 1.35e5

- Use some bits for the exponent, some bits for the mantissa
- Can pick either two bytes (single precision) or four bytes (double precision)



C++

- This brings us finally to C++
- The first concept we need to understand is the data type
  - Tells the computer how you are going to INTERPRET the bits on the register
- Example : Computer gives you 11110110100111010111011010011101
- What the heck is that?
  - Is it a signed integer? -157452643
  - Is it an unsigned integer? 4137514653
  - Is it a float? -1.5968679E33
  - Is it a double? -1.59686791004826 87E33

C++

- Since computing (and nature) is hard, you have to understand what the computer is giving you and how you are interpreting it
- This is why C++ is "strongly typed" (you need to tell the computer how you are representing the number)

Type	Size	Min	Max
bool	1 bit	0	1
unsigned char	1 byte	0	255
char	1 byte	-128	127
unsigned int	2 bytes	0	65,535
int	2 bytes	-32,768	32,767
unsigned long int	4 bytes	0	4,294,967,295
long int	4 bytes	-2,147,483,648	2,147,483,647
float	4 bytes	1.2E-38	3.4E+38
double	8 bytes	2.3E-308	1.7E+308

C++

- There is an annoyance, though : the standard specifies that “int”, for instance, must be AT LEAST 16 bits (2 bytes). It can be more, and sometimes is!

- This is ARCHITECTURE DEPENDENT
  - size of int on linux+intel != size of int on mac+amd
  - http://en.cppreference.com/w/cpp/language/types

C++

Type specifier	Equivalent type	Width in bits by data model				
		C++ standard	LP32	ILP32	LLP64	LP64
short	short int	at least 16	16	16	16	16
short int						
signed short						
signed short int						
unsigned short	unsigned short int					
unsigned short int						
int	int	at least 16	16	32	32	32
signed						
signed int						
unsigned	unsigned int					
unsigned int						
long	long int	at least 32	32	32	32	64
long int						
signed long						
signed long int						
unsigned long	unsigned long int					
unsigned long int						
long long	long long int (C++11)	at least 64	64	64	64	64
long long int						
signed long long						
signed long long int						
unsigned long long	unsigned long long int (C++11)					
unsigned long long int						

C++

Type	Size in bits	Format	Value range	
			Approximate	Exact
character	8	signed (one's complement)	-127 to 127[note 1]	
		signed (two's complement)	-128 to 127	
		unsigned	0 to 255	
	16	unsigned	0 to 65535	
	32	unsigned	0 to 1114111 (0x10ffff)	
integral	16	signed (one's complement)	$\pm 3.27 \cdot 10^{4}$	-32767 to 32767
		signed (two's complement)		-32768 to 32767
		unsigned	0 to $6.55 \cdot 10^{4}$	0 to 65535
	32	signed (one's complement)	$\pm 2.14 \cdot 10^{9}$	-2,147,483,647 to 2,147,483,647
		signed (two's complement)		-2,147,483,648 to 2,147,483,647
		unsigned	0 to $4.29 \cdot 10^{9}$	0 to 4,294,967,295
	64	signed (one's complement)	$\pm 9.22 \cdot 10^{18}$	-9,223,372,036,854,775,807 to 9,223,372,036,854,775,807
		signed (two's complement)		-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
		unsigned	0 to $1.84 \cdot 10^{19}$	0 to 18,446,744,073,709,551,615
floating point	32	IEEE-754	$\pm 3.4 \cdot 10^{\pm 38}$ (~7 digits)	• min subnormal: $\pm$ 1.401,298,4 $\cdot 10^{-47}$ • min normal: $\pm$ 1.175,494,3 $\cdot 10^{-38}$ • max: $\pm$ 3.402,823,4 $\cdot 10^{38}$
	64	IEEE-754	$\pm 1.7 \cdot 10^{\pm 308}$ (~15 digits)	• min subnormal: $\pm$ 4.940,656,458,412 $\cdot 10^{-324}$ • min normal: $\pm$ 2.225,073,858,507,201,4 $\cdot 10^{-308}$ • max: $\pm$ 1.797,693,134,862,315,7 $\cdot 10^{308}$

C++

- tl;dr :
  - Data types are hard
  - They are designed for efficiency
  - C++ types you will use the most :
    - char
    - int
    - unsigned int
    - float
    - double
  - You need to be aware of the intricacies and difficulties, though… you WILL run into this if you do any serious numerical computing in the future.