Numbers

Momchil Ivanov

12.04.2017

Bases

Base	Elements	Name
2	0,1	binary (bin)
10	0,1,2,3,4,5,6,7,8,9	decimal (dec)
16	0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f	hexadecimal (hex)

Table : Bases.

Integer numbers

Range of unsigned integers: from 0 to $(2^n - 1)$, where n is the number of bits.
Range for signed integers: from (-2^{n-1}) to $(2^{n-1} - 1)$.

Bits	Sign	Minimum value	Maximum value
4	no	0	15
8	no	0	255
16	no	0	65,535
32	no	0	4,294,967,295
64	no	0	18,446,744,073,709,551,615
4	yes	-8	7
8	yes	-128	127
16	yes	-32,768	32,767
32	yes	-2,147,483,648	2,147,483,647
64	yes	-9,223,372,036,854,775,808	9,223,372,036,854,775,807

Table : Range of integer numbers.

Integer numbers in 32 bit representation

Base conversion of unsigned numbers from binary to decimal

$$\sum_{i=1}^{32} b_i 2^{i-1} \quad , \tag{1}$$

where b_1 is the right most bit. What about the signed integers? Try to understand the negative numbers, here $b_{32} = 1$ is the sign bit.

base 10	base 2
-2	11111111111111111111111111111110
-1	11111111111111111111111111111111
0	00000000000000000000000000000000
1	00000000000000000000000000000001
2	00000000000000000000000000000010
3	00000000000000000000000000000011

Table : Signed integer numbers in 32 bit representation.

Integer numbers in binary representation

In python you can display the binary representation of a decimal number using

bin(123456)

or the hexadecimal representation with

hex(123456)

In scientific calculators (calc.exe in Windows) you can switch the representation from the menu. In the arbitrary precision calculator "calc" from the Virtual Machine, you can switch the representation with

config("mode","binary")

or

config("mode","hex")

Floating Point Numbers

Single precision (32 bit).

Bits	Length	Meaning
1–23	23	fraction (mantissa)
24–31	8	exponent
32	1	sign

Table : Bit field (right most bit is b_1).

Formula for computing the decimal representation of a floating point number

$$(-1)^{\text{sign}} \left(1 + \sum_{i=1}^{23} b_{24-i} 2^{-i}\right) \times 2^{e-127}. \tag{2}$$

Floating Point - Examples

Base 10	Base 2
-1.0	10111111100000000000000000000000
0.0	00000000000000000000000000000000
1.0	00111111100000000000000000000000
2.0	01000000000000000000000000000000
4.0	01000000100000000000000000000000

Table : Floating point numbers in 32 bit representation.

Floating Point - Examples

Base 10	Base 2
1.0000000000000000	00111111100000000000000000000000
1.0000001192092896	00111111100000000000000000000001
1.0000002384185791	00111111100000000000000000000010
1.0000003576278687	00111111100000000000000000000011
1.0000004768371582	00111111100000000000000000000100
1.0000005960464478	00111111100000000000000000000101
1.0000007152557373	00111111100000000000000000000110
1.0000008344650269	00111111100000000000000000000111
1.0000009536743164	00111111100000000000000000001000
1.0000010728836060	00111111100000000000000000001001
1.0000011920928955	00111111100000000000000000001010
1.0000013113021851	00111111100000000000000000001011
1.0000014305114746	00111111100000000000000000001100

Table : Floating point numbers in 32 bit representation.

Floating Point - Python

Print the bit representation of the floating point 1.1234 in Python

```
import struct
bin(struct.unpack('Q',struct.pack('d',1.1234))[0])
```

The output here

'0b1111111111000111111001011100100100011101000101001110001110111'

Note that this code skips leading zero bits, i.e. try to see what 0.0 is in binary.

Precision

Besides single precision (32 bit) there is also double precision (64 bit) and quadruple precision (128 bit). Most processors (as well as recent GPUs) can handle single and double precision floating point numbers. Older GPUs can handle only single precision. Most computing software comes with both precisions. There is also arbitrary precision software ("calc" in the VM).

Type	Significant decimal digits
single	6–9
double	15–17
quadruple	33–36

Table : Precision of floating point representations.

For python refer to

`https://docs.python.org/2/tutorial/floatingpoint.html`

`https://docs.python.org/2/library/stdtypes.html`

Floating Point - Examples

Compute

$$01000000010010010000111111011011.$$

Floating Point - Examples

Compute

$$01000000010010010000111111011011.$$

The answer is somewhere near π

$$3.1415927410125732.$$

References

Basics

http://en.wikipedia.org/wiki/Single-precision_floating-point_format

http://en.wikipedia.org/wiki/Double-precision_floating-point_format

http://en.wikipedia.org/wiki/Quadruple-precision_floating-point_format

https://docs.python.org/2/tutorial/floatingpoint.html

https://docs.python.org/2/library/stdtypes.html

Advanced reading

David Goldberg, *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, Computing Surveys (1991).

https://ece.uwaterloo.ca/~dwharder/NumericalAnalysis/02Numerics/Double/paper.pdf