23-25 – Number Systems

Friday, November 9, 2018 10:14 AM

Types of number

See Maths notes for types of number.

Units

bibyte is in **bits.
Kibibyte, mebibyte, gibibyte and tebibyte store 2^{10} (1024), 2^{20}, 2^{30} and, 2^{40} respectively.
Kilobyte, megabyte, gigabyte and terabyte store 10^3, 10^6, 10^9, 10^{12} respectively.

Changing Base

Remember this table for hexadecimal lookup:

Binary	Hex
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

See the number base converter for confirmation:
https://joeiddon.github.io/random/number_bases

Two's Complement

https://en.wikipedia.org/wiki/Two%27s_complement

The two's complement of an *N*-bit number is defined as its complement with respect to 2^N. For instance, for the three-bit number 010, the two's complement is 110, because 010 + 110 = 1000.

You can speed up this process by flipping all the bits and adding 1.

In a Two's Complement number system, you can reverse the sign of a number by taking its complement. E.g. from above, we see that –2 is represented as 110 since it is the two's complement of 2 which is 010.

Consider the following decimals and their representations in two's complement using 3 bits.

Decimal	Binary
3	011
2	010
1	001
q	000
-1	111
-2	110
-3	101
-4	100

Decimal -> Two's Complement Representation

Forget the sign, represent as binary then take the two's complement of this number if it was negative.

E.g.
To represent -5, we convert 5 to binary, 0101, and then since –5 is negative, we take the Two's Complement of 0101 which is 1011.

Two's Complement Representation -> Decimal

Convert the binary from the second bit onwards to decimal then if the top bit is set (MSB), subtract the negative of its weight.

E.g. To convert 1100, we consider 100 in binary which is 4, and then since the top bit is set, we minus its weight which is 8 here, so 1100 is 4–8 which is –4.

Remember to truncate the result of calculations!

Fixed-Point Numbers

https://en.wikipedia.org/wiki/Fixed-point_arithmetic

A fixed point numbers are used when a computer lacks a floating point unit (FPU) or if fixed-point performs better performance or accuracy.

A fixed-point data type is an integer that is scaled by an implicit *scaling factor* which is determined by the data type.

E.g.
1.23 can be represented as 1230 with a scaling factor of 1/1000.
1,230,000 can be represented as 1230 with a scaling factor of 1000.

Unlike floating-point data types, the scaling factor is the same for all value of the same type and does not change during computation.

The maximum value is the largest value that can be represented by the underlying integer type multiplied by the scaling factor; and similarly for the minimum value.

Normalisation

The process of making your float's mantissa fit that of the data type you are trying to represent it in.

E.g. In standard form, the mantissa must be between 1 and 10.

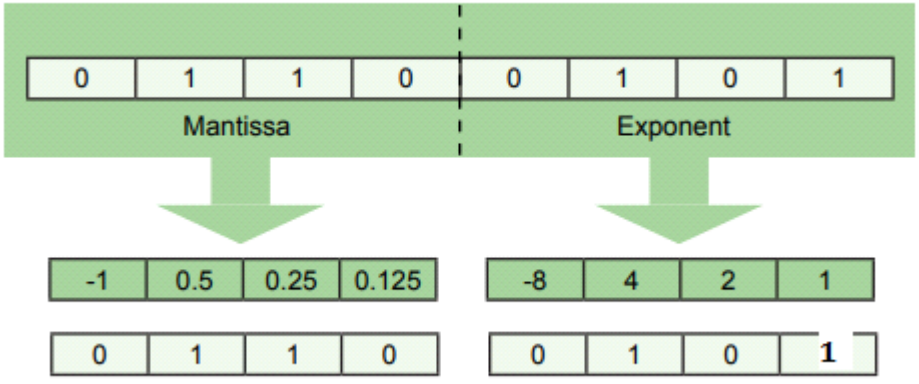
Floating-Point Numbers

https://en.wikipedia.org/wiki/Half-precision_floating-point_format

Floating point-numbers are just like standard form, but in base 2.

They are stored as a pre-determined length mantissa (fraction) and exponent. Both these are floats that must be interpreted by the Two's Complement system.

E.g.
With an 8-bit float which has a 4-bit mantissa and 4-bit exponent, consider:



So the exponent is 5 which means to shift the radix point to the right by 5 places.

I.e. 0.110 goes to 1100.0 in binary which is 24 in denary.

Overflow and Underflow

An overflow is when the result of an arithmetic calculation is larger than the maximum value that can be stored in a data type.

Similarly, an underflow is when the result is smaller than the minimum value.

Absolute and Relative Error

These represent how badly your number has been represented.

Absolute error is the difference between the true value and the represented value.

Relative error is the absolute error / true value.

Some practice:

1) unsigned integer is an integer (whole number) that is non-negative
2) a) 01000000 b) 11000000 c) 01100100 d) 10011100
3) 10101
4) a) 1100 + 1000 = 10100 b) 11001 + 01111 = 01000 = 8
5) Fixed point is more memory efficient, but you have no flexibility to represent really big or really small numbers.
6) 3.25 = 11.01 in binary
7) 10.11 = 2.75 in denary
8) a) 56 << 9 = 28672 b) 0.101 = 0.625
9) a) 13.5 = 1101.1 b) -3.75 = -11.11 = 00.01