BINARY

Number systems:

- *Natural number:* a positive whole number including zero
- *Integer:* any positive or negative whole number including zero. In theory integers are infinite but we have to limit it.
- *Rational number:* any number that can be expressed as a fraction or ratio of integers
- *Irrational number:* a number that cannot be expressed as a fraction or a ratio. The programmer must decide on the level of precision to use.

- *Ordinal Numbers:* a number used to identify a position relative to other numbers
- *Cardinal numbers:* a number that identify the size of something

Set (Maths) & Well-ordered set: a group of items with a defined order

Array (Computing): a data structure where are grouped together under a single identifier and are then accessed based on their position.

Number bases:

Bit = binary digit, given by the two states: electricity flowing (1) or not (0)

The number base m_n is usually indicated in the subscript

Denary = decimale, used for numbers by humans

Binary = binario, used for everything by the computer

Hexadecimal = esadecimale (base = 16), is sometimes used in computing for simplify the human view, for exemple colours #FF(red) 00(green) 00(blue), the digits are 0 to 9 plus A, B, C, D, E, F

Decimal			Binary				
Value	SI		Value	IEC		JEDEC	
1000	k	kilo	1024	Ki	kibi	K	kilo
1000^2	M	mega	1024^2	Mi	mebi	M	mega
1000^3	G	giga	1024^3	Gi	gibi	G	giga
1000^4	T	tera	1024^4	Ti	tebi	–	
1000^5	P	peta	1024^5	Pi	pebi	–	
1000^6	E	exa	1024^6	Ei	exbi	–	
1000^7	Z	zetta	1024^7	Zi	zebi	–	
1000^8	Y	yotta	1024^8	Yi	yobi	–	

If I have a n bytes machine the range is 2^n and the largest is $2^n - 1$

- MSB (most significant bit): the biggest bit (number further to left in binary)
- LSB : the lowest (units)

Convertions

Denary to binary:

Method 1

- Write out place holders
- Subtract the largest place holder value that is <= Denary No
- Put a 1 in that place holders column
- Repeat using the remainder

Method 2 (an algorithm easier programmed)

- Use on larger denary numbers
- Repeatedly divide Denary number by 2
- Remainders are either 0 or 1
- Remainders form the binary number
- Read remainders in reverse order

Denary to Hexadecimal:

- Convert Denary to Binary, group into blocks of 4 bits, interpret each group of 4 bits as Hex
- Dividing repeatedly by 16 and get the remainder

Binary coded decimal (BCD)

Early calculators only processed 4 bits, so with each 4 bits represent a digit. When doing an addition, if the two decimal first digits have a carry, do the normal addition and then add six.

436		0100	0011	0110
738		0111	0011	1000
	1		1	
1174	0001	**1011**	0111	**1110**
		0110		**0110**
		0001		0100
	0001	0001	0111	0100

Negative: 2’s Complement

Two’s complement is a method used to represent signed integers (positive or negative) in binary form. The first on the left is the Sign Bit and if we have 16 bits it value -2^{15}.

So there are two ways to convert a signed binary number in decimal:

1. Sum all the values considering the first one negative

-2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
0 if +ve 1 if -ve	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1

-32768 + (16384 + 8192 + 4096 + 2048 + 1024 + 512 + 256 + 128 + 64 + 32 + 8 + 4 + 2 + 1) = -32768 + 32751 = -17

2. If the first bit is 0 read as a normal positive binary otherwise flip all the bits (from 0 to 1 and vice versa), read it as a positive number, add one and multiply by -1

-95	
47	1
23	1
11	1
5	1
2	1
1	0
0	1

Read the remainders from the bottom up

1011111

Add leading zeros to make up 16 bits

000000000

1011111

0000000001011111

1111111110100000 Flip all the bits

+1 Add 1

1111111110100001

In the same way you can convert a negative number in a 2’s Complement signed one:

In the 2’s Complement:

- 1000000000000000 is the minimum number (-2^{15})
- 1111111111111111 = -1
- 0111111111111111 is the maximum number ($2^{15} - 1$)

Addition with 2’s Complement

It is done in the same way of normal binary numbers but there are some differences to remember:

- *Carry in the sign bit:* If adding two positive numbers I have got a carry in the sign bit I get an *overflow error*. This not happen if we are adding a positive and a negative when I can have a carry. Instead when adding two negative numbers I must have a carry in the sign bit.
- *Carry out of the left most bit:* If adding two numbers there is a carry out of the left most bit it’s ok, just ignore it.

Fixed Point – Binary fractions

They use an implied binary point. But they are not used because of their lack of precision when representing some number (e.g. 12.33).

Floating points

9.3x10^6 → 9.3 is the mantissa and 6 the exponent

In binary is the same, the binary point is after the first digit. The mantissa and exponent are usually two's complement.

01010000 / 0011 → from 0.101 the point is moved 3 places to the right

Compared to fixed points:

- Bigger ranges
- More accuracy
- Arithmetic is slow

Normalised number:

- For a positive number, the first bit after the comma must be a 1 not to waste bits and therefore accuracy (begins with 01)
- For a negative number, the first bit after the comma must be a 0 not to waste bits and therefore accuracy (begins with 10)

To normalise a number you shift right the mantissa until it become normalised and then subtract the number of shifts from the exponent.

One of the problem is that it cannot represent 0 with this notation. Therefore an exception is made for 0: if all the bits are 0 then that represent 0.

The set of number a floating point is made as: ______ 0 ______

If it goes outside the external bounds we get an overflow, if inside the range in the middle that cannot be represented we get an underflow.

- *Absolute error:* difference value can be represented-actual value
- *Relative error:* absolute error/actual value