Fixed point binary *(always stored using two’s complement)*

e.g. If we wanted to store 8.5 in binary ($8_{1/2}$)

-16	8	4	2	1	1/2	1/4	1/8
-2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
0	1	0	0	0	1	0	0

We can see that 8 bits have been used to store this value. This 8 bits range across the necessary base’s of two required to store 8.5. As you move left to right beyond the binary point (decimal point), the value will half as the power of two becomes more negative.

e.g. If we wanted to store -6.5 in binary ($-6_{1/2}$)

-16	8	4	2	1	1/2	1/4	1/8
-2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
1	1	0	0	1	1	0	0

This method of notation is called fixed point binary as the position of the decimal point is fixed on the number line. There is a limited range of numbers that this notation can represent – a maximum of 01111111 (0+8+4+2+1+(1/2)+(1/4)+(1/8) = 15.875) and the minimum – 10000000 = -16). This is because some digits are assigned to storing the fractional part of the number and so much larger ranges of numbers cannot be stored with this method.
Moreover, it is not entirely accurate as the rational numbers can only be represented using a limited range of base’2 numbers. E.g. if we wanted to store 1/3 (or 0.33333…..) (denominator is odd not even)

-16	8	4	2	1	1/2	1/4	1/8
-2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
0	0	0	0	0	0	1	0

This would be a reoccurring pattern of .0101010 then on after (0.25252525 ~0.33333). Given the number bits though we can’t accurately store 1/3.

Floating point binary (increases accuracy)

***Floating point refers** to the fact there is no fixed position of the decimal place, the number of digits before and after the decimal point can vary. Floating point numbers refer to numbers that are real (can be either rational or irrational).*

Say if we wanted to store a real number more accurately, we can move the place values of the same 8 bits. In other words, the binary point is floating up and down the number line (refer to definition on page 1). The advantage of floating point binary is that using only 8 bits, either the size of the number can be increased or the accuracy can be increased.

e.g.

8	4	2	1	● 1/2	1/4	1/8	1/16
2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
4 bits to the left --- 4 bits to the right							
-64	32	16	8	4	2	1	● 1/2
2^{-6}	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}
7 bits to the left --- 1 bit to the right							
-4	2	1	1/2	1/4	1/8	1/16	1/32
2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
3 bits to the left --- 5 bits to the right							

However, this means that two parts need to be stored: one to represent the actual binary value, and another to tell the computer where the binary point is on the number line. This is accomplished by splitting the 8 bits into 2 parts. One is called the mantissa and the other is the exponent.

Mantissa = the part of a floating point number representing the significant digits of this number.
Exponent = a binary value stored with the Mantissa that is used to interpret the of significant digits i.e. position the binary point.
-Both of which are stored using two's complement method (where the highest digit represents sign & magnitude).

Floating point numbers on computers can only store up to 23 digits (32-bit computers, or 56 on 64bit computers).

The Mantissa

$-2^?$	**🔴 $2^?$**	**$2^?$**	**$2^?$**	**🟢 $2^?$**
0	1	1	0	0

The Exponent

-4	**2**	**1**
0	1	1

This is equal to 2+1 = 3. Since we have positive +3 it means the starting binary point moves right 3 places. The binary starting point will always be between the 1st and 2nd most significant digit unless stated otherwise.

🔴 **= Starting binary point (imaginary/exponent of 0)**
🟢 **= Binary point (denoted by exponent)**

Therefore, the Mantissa is equal to 6 see →

-2^3	**2^2**	**2^1**	**2^0**	**🟢 2^{-1}**
0	1	1	0	0

Floating point binary e.g.2) & e.g.3)

Mantissa	**$-2^?$**	**$2^?$**	**$2^?$**	**$2^?$**	**$2^?$**	**Exponent**	**-2^2**	**2^1**	**2^0**
	0	1	1	1	0		0	0	1
	-2^1	**2^0**	**2^{-1}**	**2^{-2}**	**2^{-3}**		= 1 (+1 place to the right)		
	= 1+ ½+ ¼ → + 4/4 + 2/4 +1/4 = 7/4 = 1.75 or 1 & 3/4								

Mantissa	**$-2^?$**	**$2^?$**	**$2^?$**	**$2^?$**	**$2^?$**	**Exponent**	**-2^2**	**2^1**	**2^0**
	0	1	0	0	0		1	1	0
	We need a new number line! Simply back fill 0's to the left of the most significant digit						= -2 (2 places to the left)		

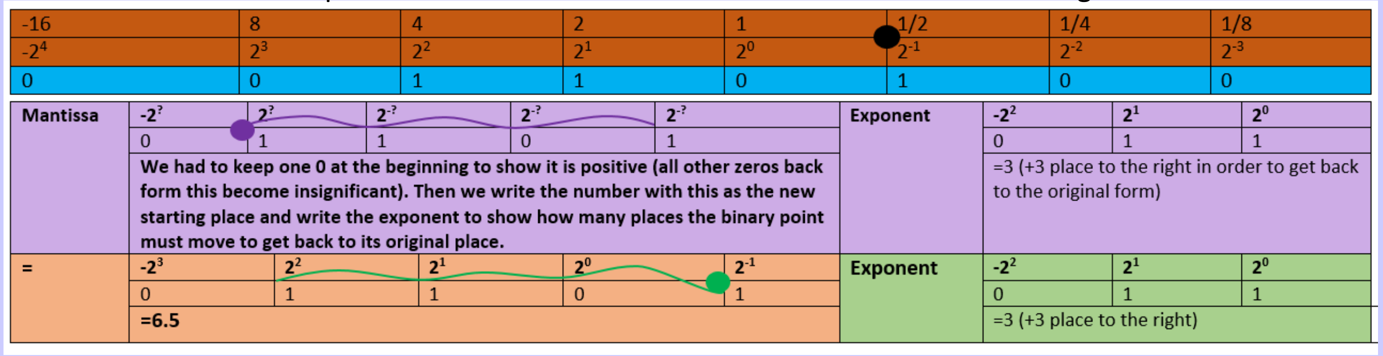
	-2^0	**2^{-1}**	**2^{-2}**	**2^{-3}**	**2^{-4}**	**2^{-5}**	**2^{-6}**
	0	**0**	**0**	**1**	**0**	**0**	**0**
	= 1/8 = 0.125						

In this example we had to backfill some 0's to allow for the number to be represented. If the number is positive (starts with a 0) we back fill with 0's but if the number starts with a 1 (it is negative) we just backfill with 1's

Binary Normalisation

One problem with floating point binary is that any given number can be represented using a variety of different combinations of Mantissa and exponent. A unified way is required to make it more efficient and set the convention. For example, the decimal 1 can be represented as: 00100 010 or 00010 011 or 01000 001 or many other combinations.

The correct (normalised) version is actually the third example, 01000 001, since all positive binary numbers should begin with 01 and all negative binary numbers should begin with 10 and then you know the numbers are normalised.
Therefore the third example is the correct normalised version of the number 1 using 8 bits.



-16	8	4	2	1	1/2	1/4	1/8
-2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
0	0	1	1	0	1	0	0

Mantissa	**$-2^?$**	**$2^?$**	**$2^{-?}$**	**$2^{-?}$**	**$2^{-?}$**	**Exponent**	**-2^2**	**2^1**	**2^0**
	0	1	1	0	1		0	1	1
	We had to keep one 0 at the beginning to show it is positive (all other zeros back form this become insignificant). Then we write the number with this as the new starting place and write the exponent to show how many places the binary point must move to get back to its original place.						=3 (+3 place to the right in order to get back to the original form)		
=	**-2^3**	**2^2**	**2^1**	**2^0**	**2^{-1}**	**Exponent**	**-2^2**	**2^1**	**2^0**
	0	1	1	0	1		0	1	1
	=6.5						=3 (+3 place to the right)		

●	= Not normalised
●	= Normalised position

The purple shows the correct representation for the number 6.5 (normalised floating point binary.

Normalising -2.5 in floating point binary

-16	8	4	2	1	1/2	1/4	1/8
-2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
1	1	1	0	1	1	0	0

Mantissa	**$-2^?$**	**$2^?$**	**$2^?$**	**$2^?$**	**$2^?$**	**Exponent**	**-2^2**	**2^1**	**2^0**
	1	0	1	1	0		0	1	0
We move the non-normalised binary point into a position so that the negative number begins with 10 (convention of a negative number in two's compliment). This would give an exponent of +2 to get it back to the original position (2 to the right)							=2		

Checking this: Exponent = 2 so binary point moves two to the right. The mantissa (1.0110) → 101.10 = -4 + 1 + ½ = -2.5 CORRECT

Normalising -0.25 (-1/4) in floating point binary

	-16	8	4	2	1	1/2	1/4	1/8
	-2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
+0.25 =	0	0	0	0	0	0	1	0
-0.25 =	1	1	1	1	1	1	1	0

Mantissa	**$-2^?$**	**$2^?$**	**$2^?$**	**$2^?$**	**$2^?$**	**Exponent**	**-2^2**	**2^1**	**2^0**
	1	0	0	0	0		1	1	0
We must begin with a 10 in order for this to be seen as a negative in two's compliment. Therefore we must move the binary point 2 places to the right when writing in normalised form, this involves filling in digits to the left with 0's (insignificant). The exponent is now -2 as we need to move it 2 places to the left to get back to the original.							=-2 (moves -2 places or 2 places to the left to get back to the original)		

Checking this: Exponent = -2 (infill with 1's before this -insignificant) as this will move the binary point 2 places to the left so the Mantissa 1.0000 → 1.110000 i.e. 1.1100 = -1 + ½ + ¼ = -4/4 + 2/4 + ¼ = -1/4 = -0.25 CORRECT

We normalise numbers to maximise the accuracy of the value stored and to avoid multiple representations of the same number.

Positive numbers in normalised form: always begin 01

Negative numbers in normalised form always begin 10