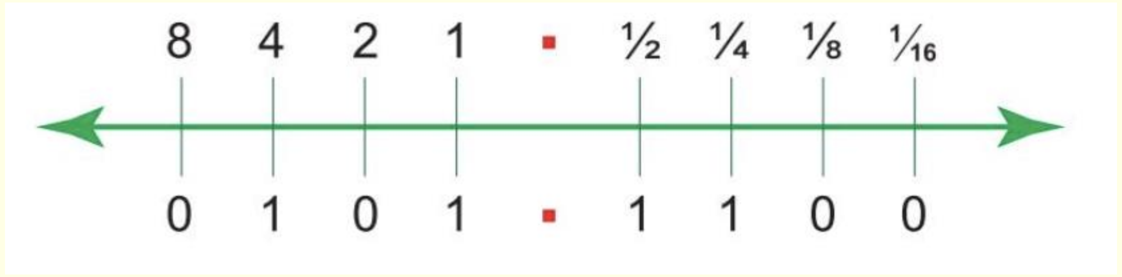
Learning Aims

- Represent positive and negative numbers with a fractional part in **fixed point** and **floating point** form

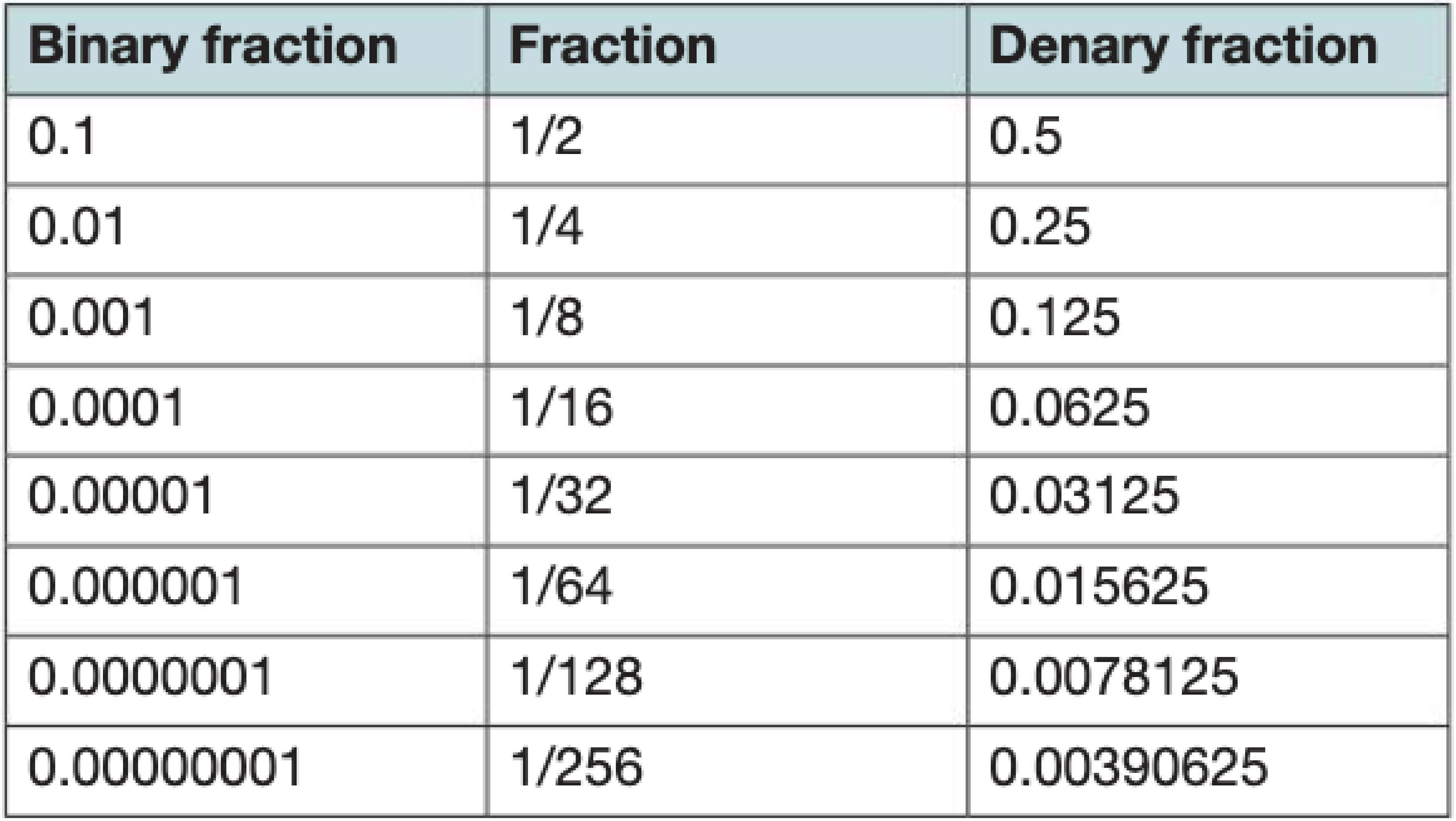
Fixed point format

- Fixed point binary numbers can be a useful way to represent fractions in binary. A binary point is used to
- separate the whole place values from the fractional part on the number line:



In the binary example above, the left hand section before the point is equal to 5 (4+1) and the right hand section is equal to ½ + ¼ (¾), or 0.5 + 0.25 = 0.75. So, using four bits after the point, 0101 1100 is 5.75 in denary.

A useful table with some denary fractions and their equivalents is given below:



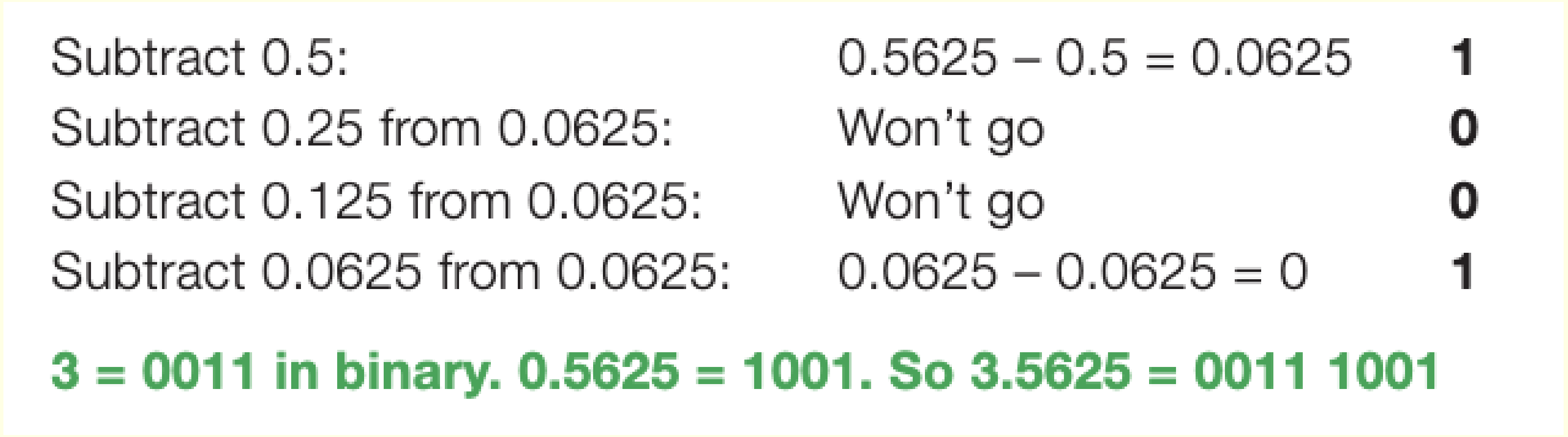
Binary fraction	Fraction	Denary fraction
0.1	1/2	0.5
0.01	1/4	0.25
0.001	1/8	0.125
0.0001	1/16	0.0625
0.00001	1/32	0.03125
0.000001	1/64	0.015625
0.0000001	1/128	0.0078125
0.00000001	1/256	0.00390625

Converting a denary fraction to fixed point binary

To convert the fractional part of a denary number to binary, you can employ the same technique as you
would when converting any denary number to binary.

Take the value and subtract each point value from the amount until you are left with 0.

Take the example 3.5625 using 4 bits to the right of the binary point:



Subtract 0.5:	0.5625 – 0.5 = 0.0625	**1**
Subtract 0.25 from 0.0625:	Won't go	**0**
Subtract 0.125 from 0.0625:	Won't go	**0**
Subtract 0.0625 from 0.0625:	0.0625 – 0.0625 = 0	**1**

3 = 0011 in binary. 0.5625 = 1001. So 3.5625 = 0011 1001

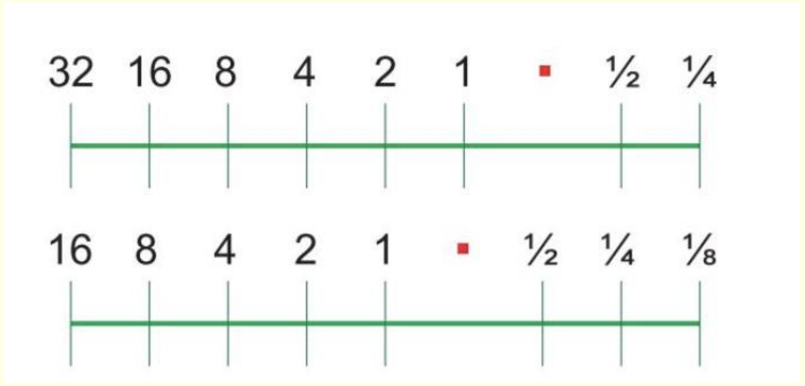
Problem 1

It is worth noticing that this system is not only **less accurate** than the denary system, but some fractions
cannot be represented at all. 0.2, 0.3 and 0.4

for example, will require an infinite number of bits to the right of the point.

The number of fractional places would therefore be truncated and the number will not be accurately stored, causing **rounding errors**.

In our denary system, two denary places can hold all values between .00 and .99.
With the fixed point binary system, 2 digits after the point can only represent 0, ¼, ½, or ¾ and nothing in between.

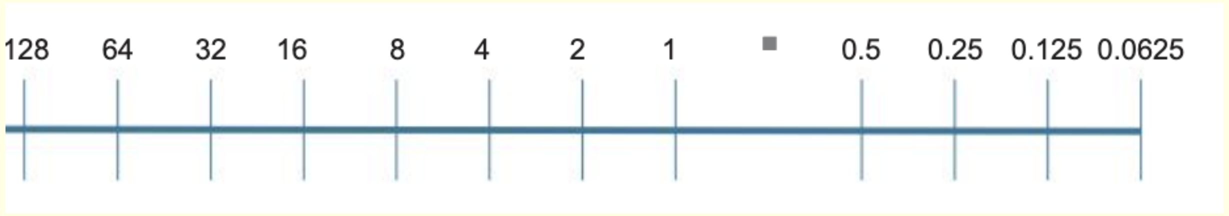


Problem 2

- The range of a fixed point binary number is also limited by the fractional part.
- For example, if you have only 8 bits to store a number to 2 binary places, you would need 2 digits after the point, leaving only 6 bits before it.
- 6 bits only gives a range of 0-63.
- Moving the point one to the left to improve accuracy within the fractional part only serves to half the range to just 0-31.
- Even with 32 bits used for each number, including 8 bits for the fractional part after the point, the maximum value that can be stored is only about 8 million.
- Another format called floating point binary can hold much larger numbers, with greater accuracy.
- Floating point form is covered in the next lesson

Recap

- What are the largest and smallest unsigned numbers that can be held in two bytes with four bits after the point?



255.9375

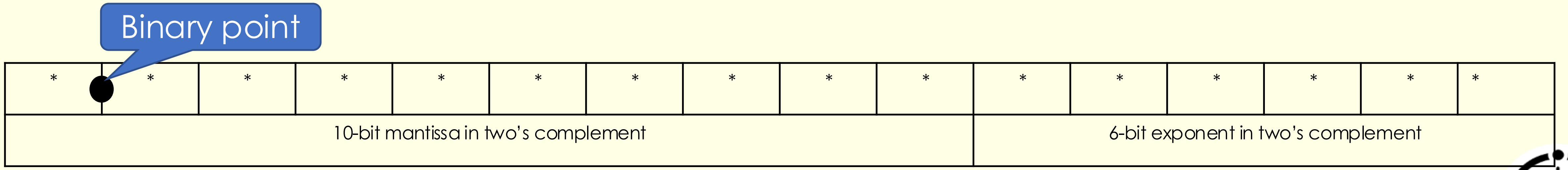
Floating point binary numbers

- Using 32 bits (4 bytes), the largest fixed point number that can be represented with just one bit after the point is only just over two billion.

- Floating point binary allows very large numbers to be represented.

Floating Point

- To represent denary fractions (decimals), it is customary to use a standard form so 123.456 is written as 1.23456 x 10^2 and 0.00167 as 1.67 x $10^{-3.}$
- The power of 10 shows how many places the decimal point has ‘floated’ left or right in the number to make the **standard form.**
- The first part of these representations is called the **mantissa** and the power to which the 10 is raised, the **exponent**.
- In binary we use a similar standard form called **floating point**, for example a 16-bit floating point number may be made up of a **10-bit mantissa** and a **6-bit exponent**.


Binary point
* * * * * * * * * * * * * * * *
10-bit mantissa in two’s complement
6-bit exponent in two’s complement

- Real numbers have fractional parts to them; in binary these fractional parts are ½, ¼, $^1/_8$, and so on.
- So the column values associated with the mantissa are:

*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
10-bit mantissa in two's complement										6-bit exponent in two's complement					

Binary point

Column value	-1	½	¼	$^1/_8$	$^1/_{16}$	$^1/_{32}$	$^1/_{64}$	$^1/_{128}$	$^1/_{256}$	$^1/_{512}$
	-1	0.5	0.25	0.125	0.0625	0.03125	0.015625	0.0078125	0.00390625	0.001953125

- The column values for the exponent are:

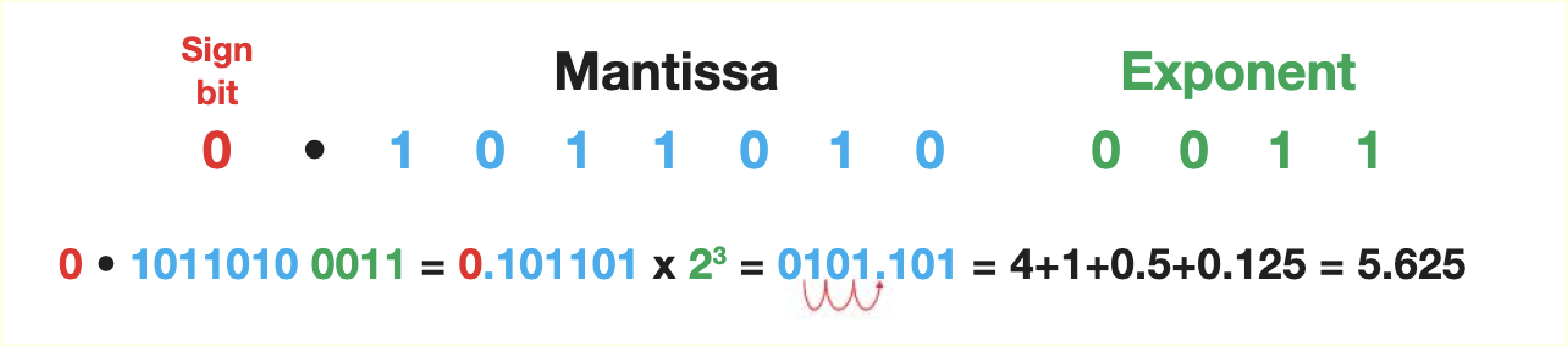
Column value	-32	16	8	4	2	1

Floating Point Binary Numbers

Floating point binary allows very large numbers to be represented.

Positive Exponent

Move the point
3 places to the right



Sign bit		Mantissa							Exponent			
0	•	1	0	1	1	0	1	0	0	0	1	1

0 • 1011010 0011 = 0.101101 x 2^3 = 0101.101 = 4+1+0.5+0.125 = 5.625

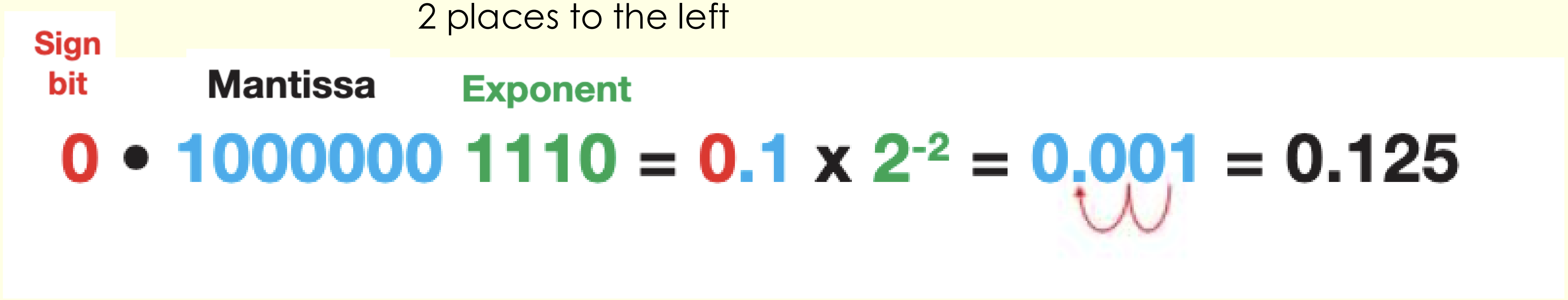
Convert the following floating point numbers to denary:

(a) 0 • 1101010 0100
(b) 0 • 1001100 0011

Floating Point Binary Numbers

- **Negative exponents**

Move the point
2 places to the left



Sign bit

Mantissa

Exponent

$$0 \bullet 1000000\ 1110 = 0.1 \times 2^{-2} = 0.001 = 0.125$$

Convert the following floating point number to denary:

0 • 1100000 1110

Floating Point Binary Numbers

Handling negative mantissas

Flip the bits from the first 1 from the right

Move point 5 places to the right

$$1 \bullet 0101101\ 0101 = -\,0.1010011 \times 2^{5} = -10100.11 = -\,20.75$$

Find the twos complement of the mantissa. It is 0.1010011, so the bits represent -0.1010011

Translate the exponent to denary, 0101 = 5

Move the binary point 5 places to the right to make it larger. The mantissa is -010100.11

Translate this to binary

16	8	4	2	1	1/2	1/4
1	0	1	0	0	1	1

The answer is -20.75.