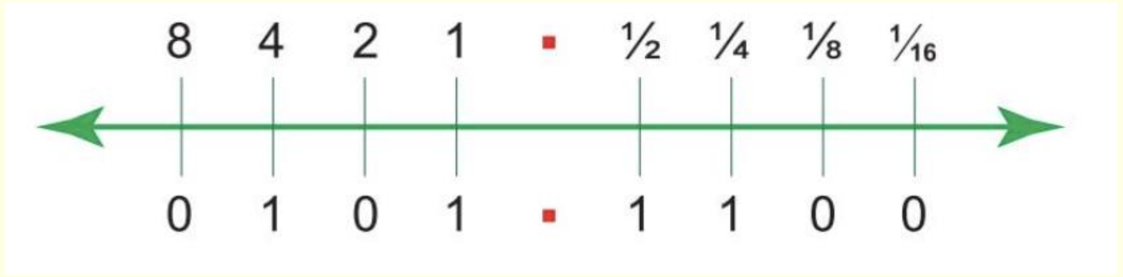
Learning Aims

Floating point arithmetic

- Represent positive and negative numbers with a fractional part in **fixed point** and **floating point** form
- Normalise un-normalised floating point numbers with positive or negative mantissas
- Add and subtract floating point numbers
- Explain underflow and overflow and describe the circumstances in which they occur

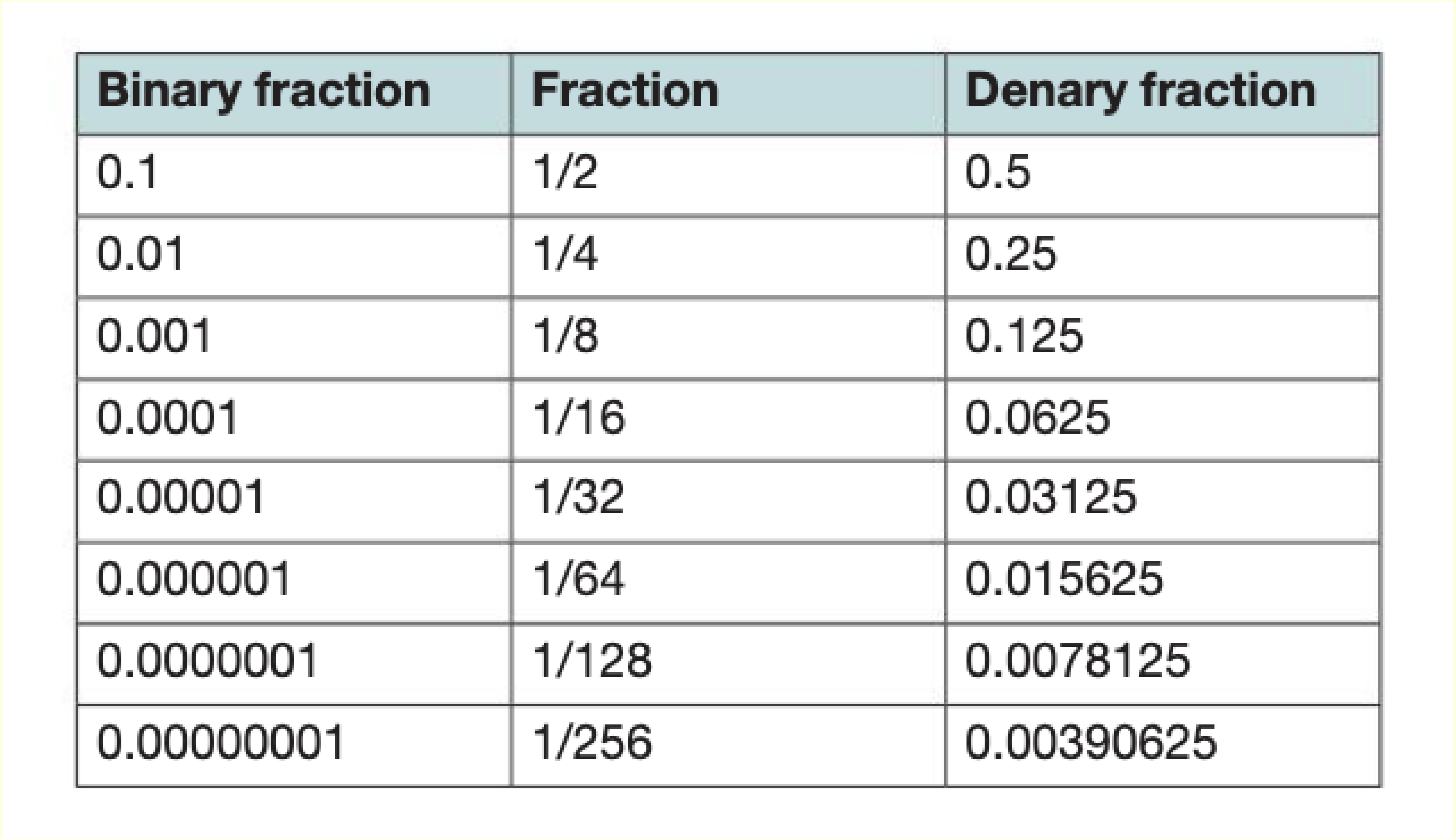
Fixed point format

- 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:



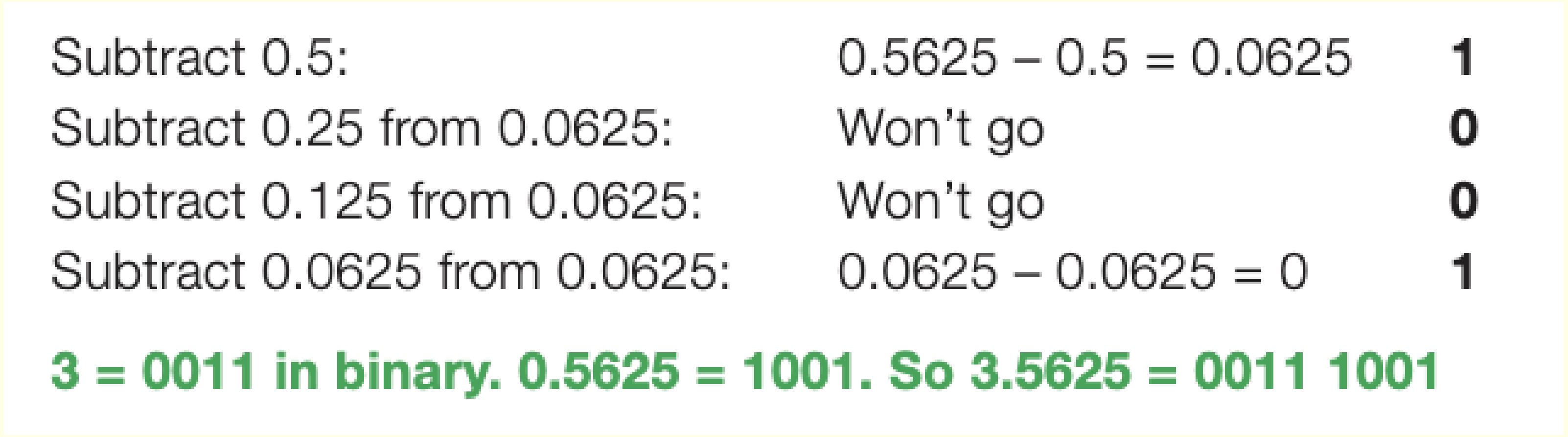
Binary fraction	Fraction	Denary fraction
0.1	1/2	0.5
0.01	1/4	0.25
0.001	1/8	0.125
0.0001	1/16	0.0625
0.00001	1/32	0.03125
0.000001	1/64	0.015625
0.0000001	1/128	0.0078125
0.00000001	1/256	0.00390625

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

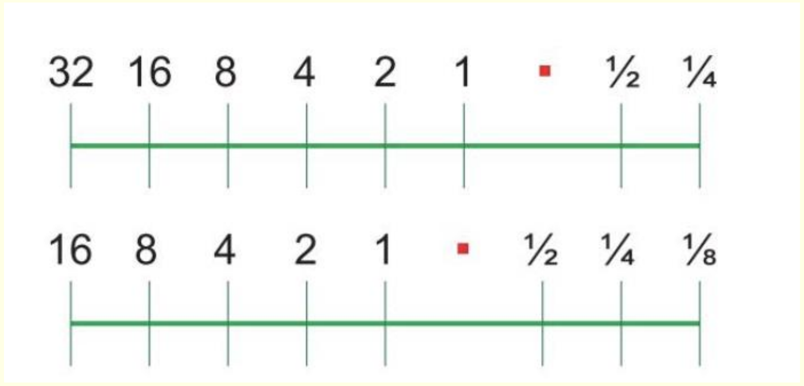
cannot be represented at all. 0.2, 0.3 and 0.4

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.

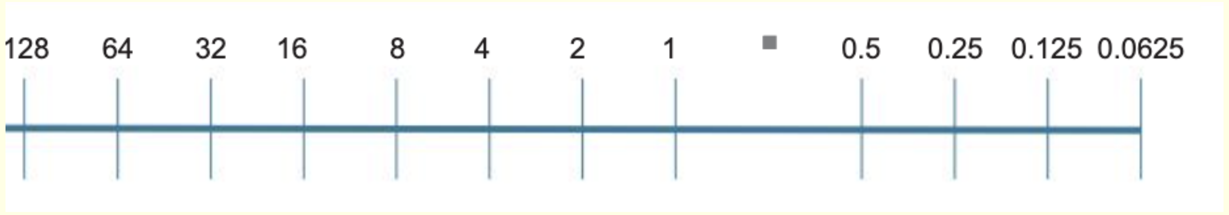


Problem 2

- 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?



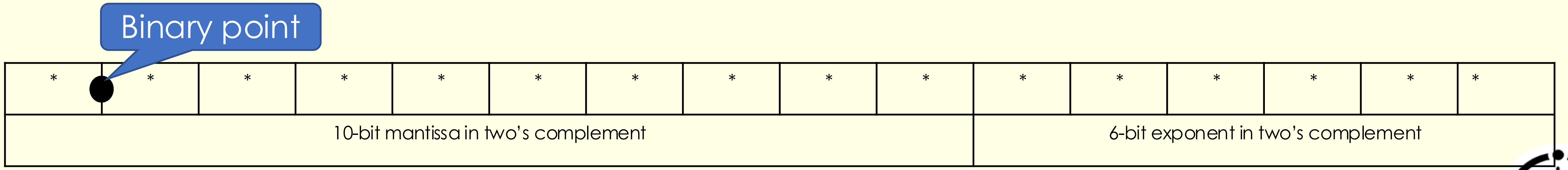
255.9375

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:

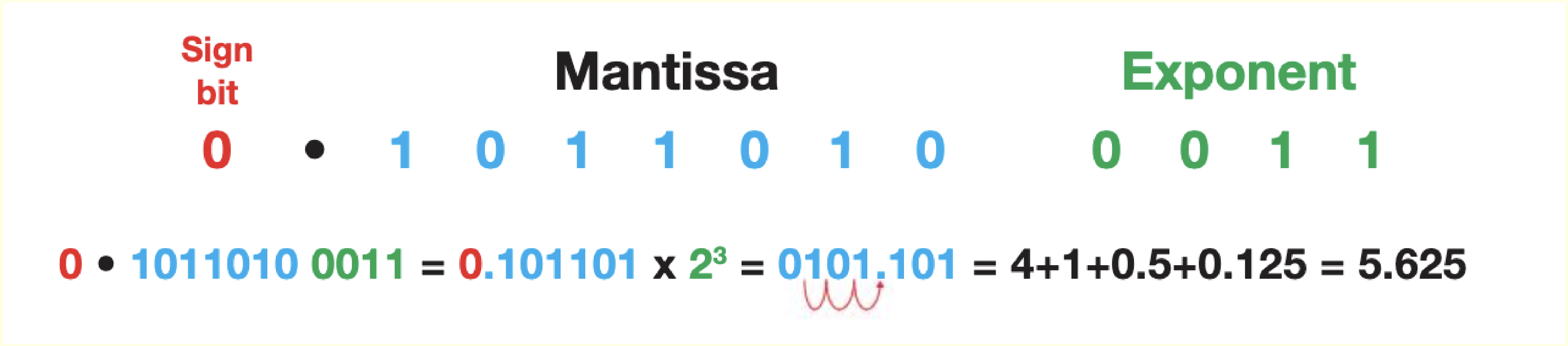
Column value	-32	16	8	4	2	1

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

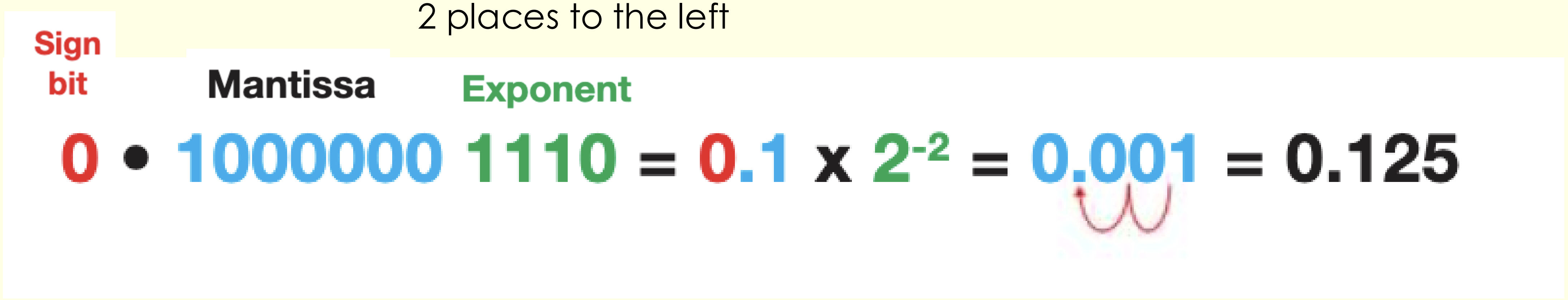
Convert the following floating point numbers to denary:

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

Floating Point Binary Numbers

- **Negative exponents**

Move the point
2 places to the left



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

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.

Normalisation

Normalisation is the process of moving the binary point of a floating point number to provide the maximum level of precision for a given number of bits.

This is achieved by ensuring that the first digit after the binary point is a significant digit.

To understand this, first consider an example in denary.

In the denary system, a number such as 5,842,13010 different ways can be represented with a 7-digit mantissa in many ways:

$0.584213 \times 10^{7} = 5{,}842{,}130$

$0.058421 \times 10^{8} = 5{,}842{,}100$

$0.005842 \times 10^{9} = 5{,}842{,}000$

The first representation, with a significant (non-zero) digit after the decimal point, has the maximum precision.

A number such as 0.00000584213 can be represented as 0.584213×10^{-5}

Normalising a positive binary number

- Having a large mantissa improves the accuracy with which a number can be represented but this would be entirely wasted if the mantissa contained a number of leading 0s.
- For this reason, floating point numbers are normalised.

Positive numbers	Negative numbers
No leading 0s to the left of the most significant bit and immediately after the binary point. Starts with 01 The binary fraction 0.000101 becomes 0.101 x 2^{-3} or 0101000000 111101	No leading 1s to the left of the mantissa. Starts with 10 Negative number 1.110010100 (10 bits) would become 1.00101 x 2^{2} or 1001010000 000010

A real number is stored as a floating-point number, which means that it is stored as two values: a mantissa, m, and an exponent, e, in the form m x 2^{e}.

Normalising a positive binary number

A positive number has a sign bit of 0 and the next digit is always 1.

Normalise the binary number 0.0001011 0101, held in an 8-bit mantissa and a 4-bit exponent.

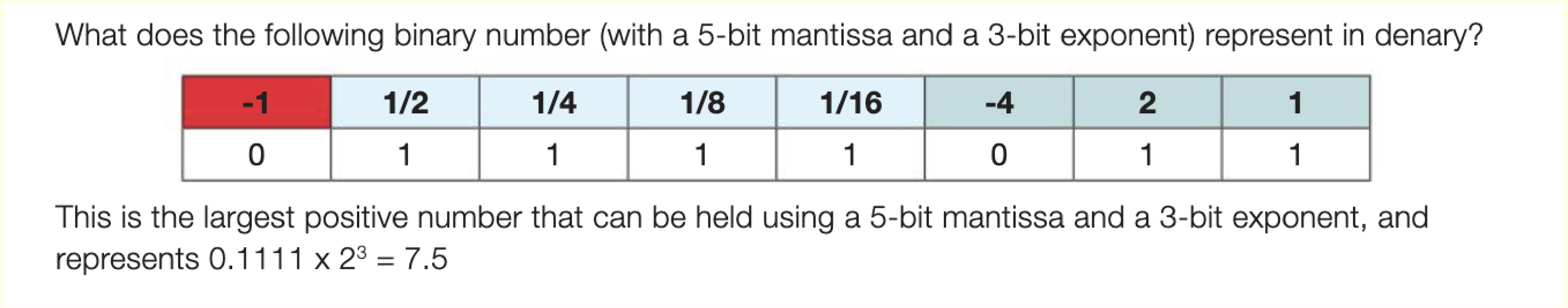
- The binary point needs to move 3 places to the right so that there is a 1 following the binary point.
- Making the mantissa larger means we must compensate by making the exponent smaller, so subtract 3 from the exponent, resulting in an exponent of 0010.
- The normalised number is 0.1011000 0010

A normalised negative number has a sign bit of 1 and the next bit is always 0.

Normalise the binary number 1.1110111 0001, held in an 8-bit mantissa and a 4-bit exponent.

- Move the binary point right 3 places, so that it is just before the first 0 digit. The mantissa is now 1.0111000
- Moving the binary point to the right makes the number larger, so we must make the exponent smaller to compensate. Subtract 3 from the exponent. The exponent is now 1 – 3 = -2 = 1110
- The normalised number is 1.0111000 1110

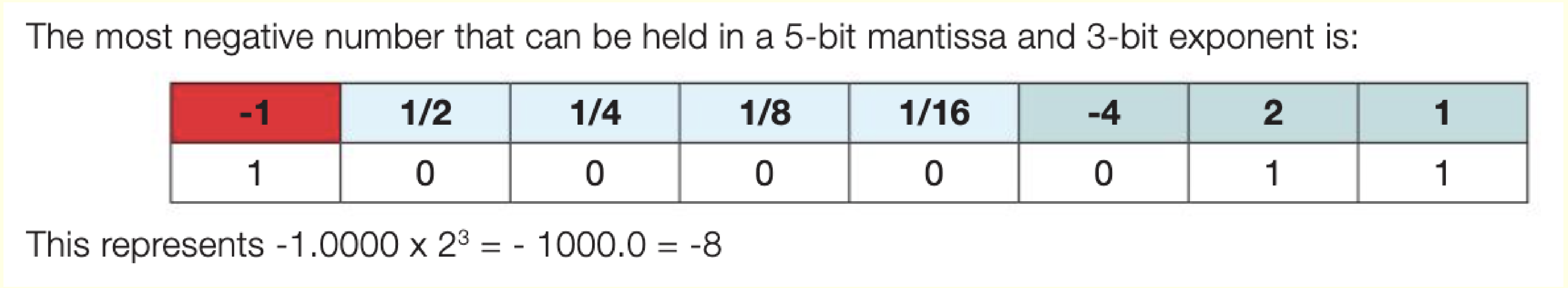
Largest positive and negative number



What does the following binary number (with a 5-bit mantissa and a 3-bit exponent) represent in denary?

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

This is the largest positive number that can be held using a 5-bit mantissa and a 3-bit exponent, and represents 0.1111 x 2^3 = 7.5



The most negative number that can be held in a 5-bit mantissa and 3-bit exponent is:

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

This represents -1.0000 x 2^3 = - 1000.0 = -8

Size of the mantissa will determine the of the number, and the size of the exponent will determine the of numbers that can be held.

Example: Representing a negative number

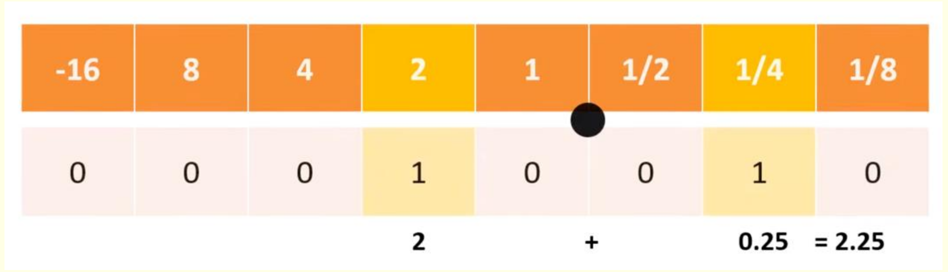
- To represent the value -0.3125 in floating point form using 10-bit two's complement mantissa and 6-bit two's complement exponent in normalised form, convert the decimal to binary:

0.3125 =	0.010100000
Flip bits	1.101100000

Now normalise by floating the binary point to remove the leading 1s in the mantissa after the binary point:

1.011000000 x 2^{-1} or 1011000000 111111

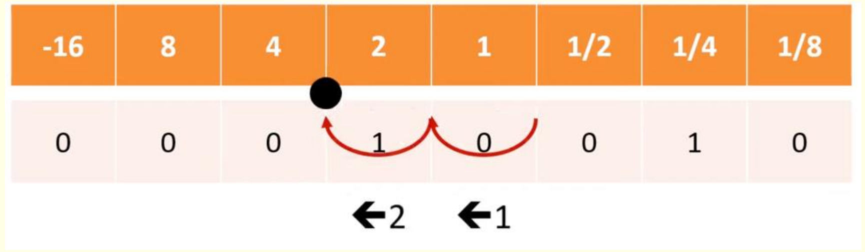
Representing a positive normalised floating point number 2.25



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

2 + 0.25 = 2.25

Step 1: Write out 2.25 on a standard fixed point number line



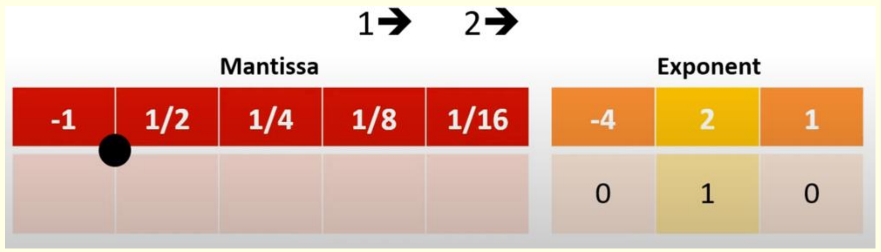
-16	8	4	2	1	1/2	1/4	1/8
0	0	0	1	0	0	1	0

←2 ←1

Step 2: move the binary point so it sits between the first 0 and 1

Normalised positive always starts with 01

Move 2 places to the left



1→ 2→

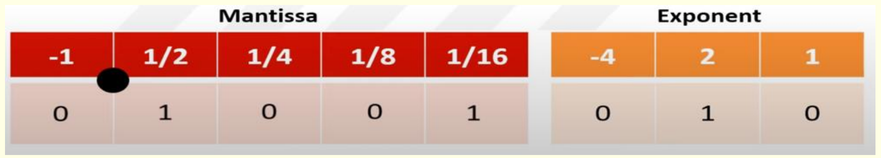
Mantissa

-1	1/2	1/4	1/8	1/16

Exponent

-4	2	1
0	1	0

Step 3: Work out what the exponent should be – 2 places to the right to put the binary point back to the correct position.



Mantissa

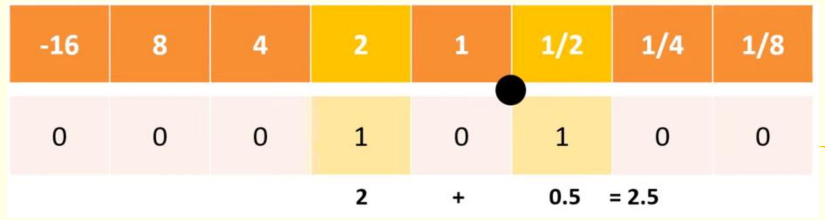
-1	1/2	1/4	1/8	1/16
0	1	0	0	1

Exponent

-4	2	1
0	1	0

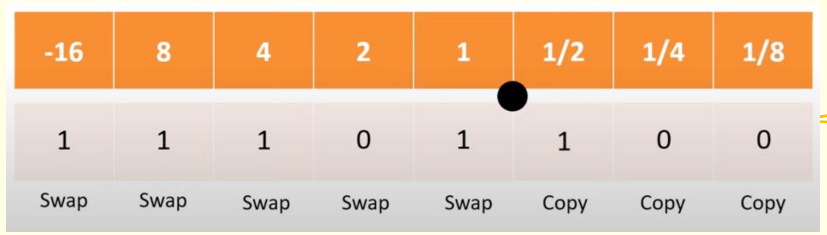
Step 4: store the mantissa

Representing a negative normalised floating point number -2.5



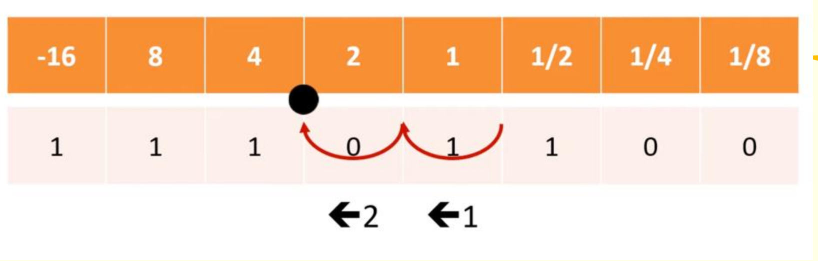
-16	8	4	2	1	1/2	1/4	1/8
0	0	0	1	0	1	0	0
			2	+	0.5	= 2.5	

Step 1: Write out the positive version of the number



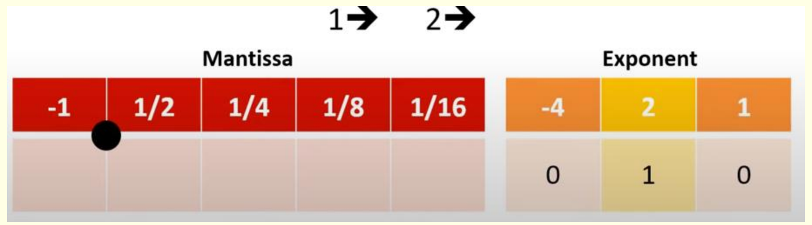
-16	8	4	2	1	1/2	1/4	1/8
1	1	1	0	1	1	0	0
Swap	Swap	Swap	Swap	Swap	Copy	Copy	Copy

Step 2: convert number to negative
Flip the bits after the first 1



-16	8	4	2	1	1/2	1/4	1/8
1	1	1	0	1	1	0	0
			←2	←1			

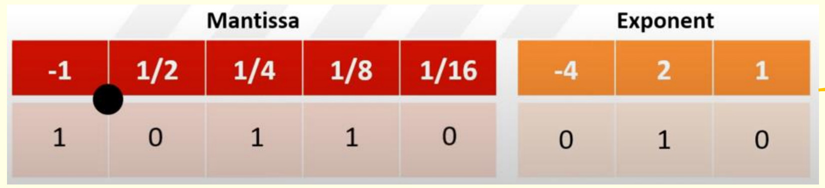
Step 3: next move the floating point to the first 10.



1➔ 2➔

Mantissa					Exponent		
-1	1/2	1/4	1/8	1/16	-4	2	1
					0	1	0

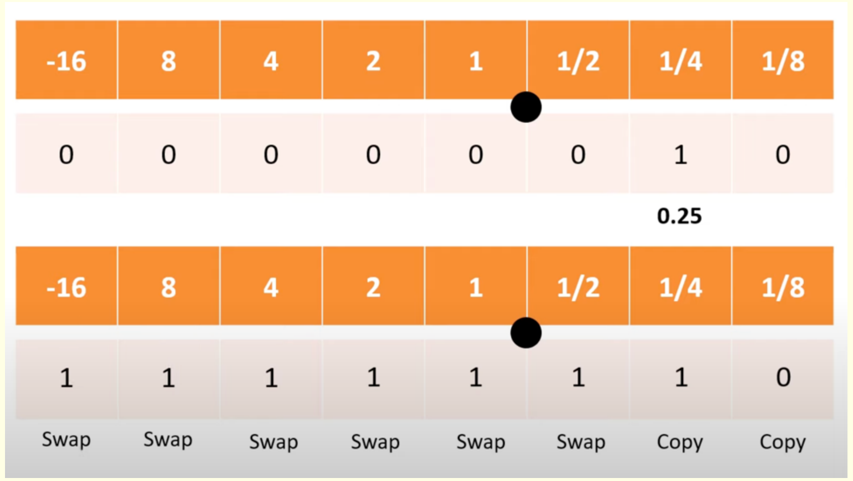
Step 4: work out exponent, 2 to the right



Mantissa					Exponent		
-1	1/2	1/4	1/8	1/16	-4	2	1
1	0	1	1	0	0	1	0

Step 5: store the mantissa

Representing a negative normalised floating point number -0.25

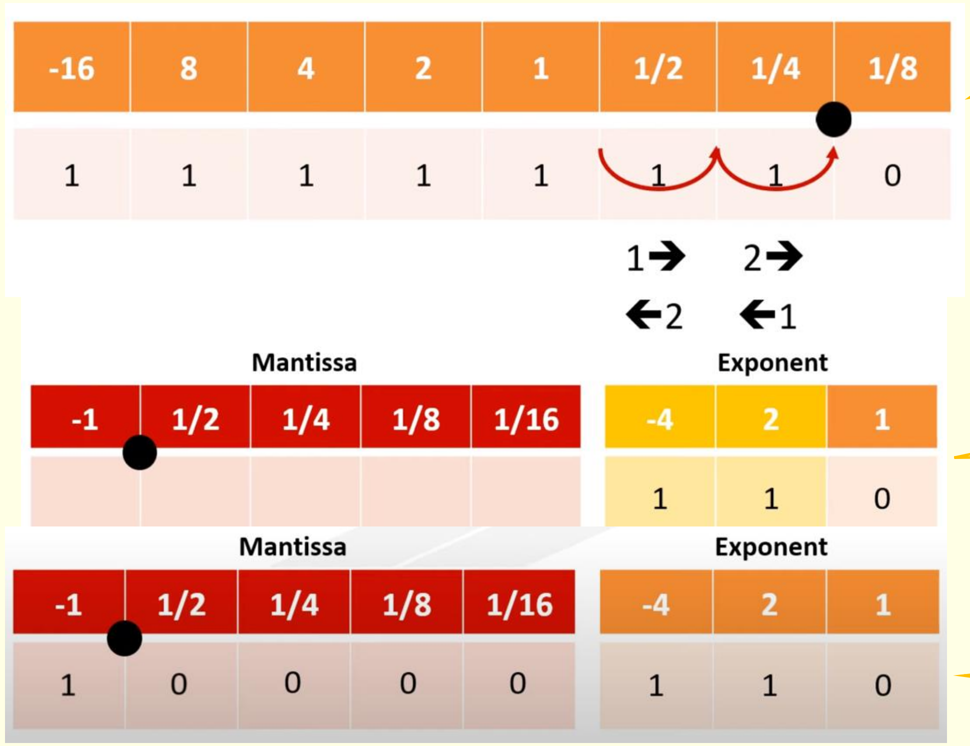


-16	8	4	2	1	1/2	1/4	1/8
0	0	0	0	0	0	1	0
						0.25	

-16	8	4	2	1	1/2	1/4	1/8
1	1	1	1	1	1	1	0
Swap	Swap	Swap	Swap	Swap	Swap	Copy	Copy

Step 1: Write out the positive version of the number

Step 2: convert number to negative
Flip the bits after the first 1



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

Mantissa

-1	1/2	1/4	1/8	1/16

Exponent

-4	2	1
1	1	0

Mantissa

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

Exponent

-4	2	1
1	1	0

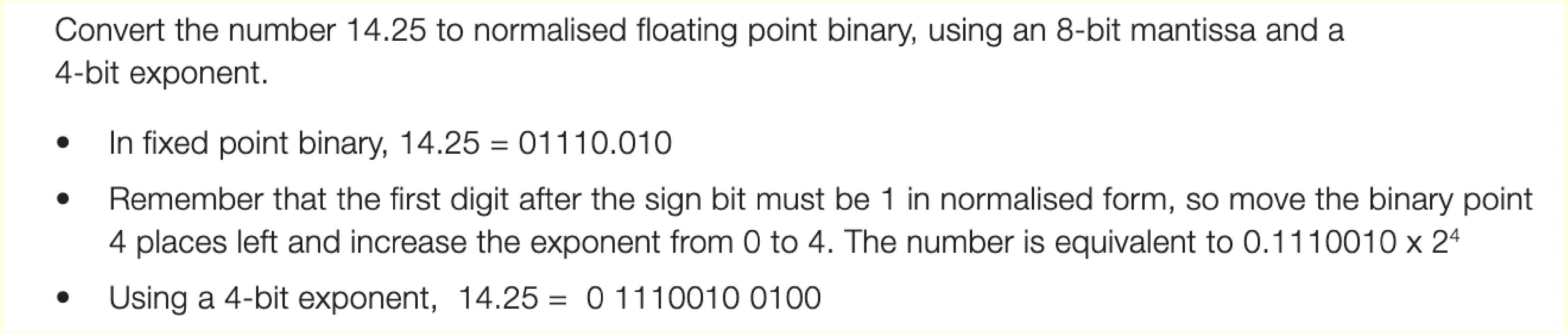
Step 3: next move the floating point to the first 10.

Step 4: work out exponent, -2 to the left

Step 5: store the mantissa

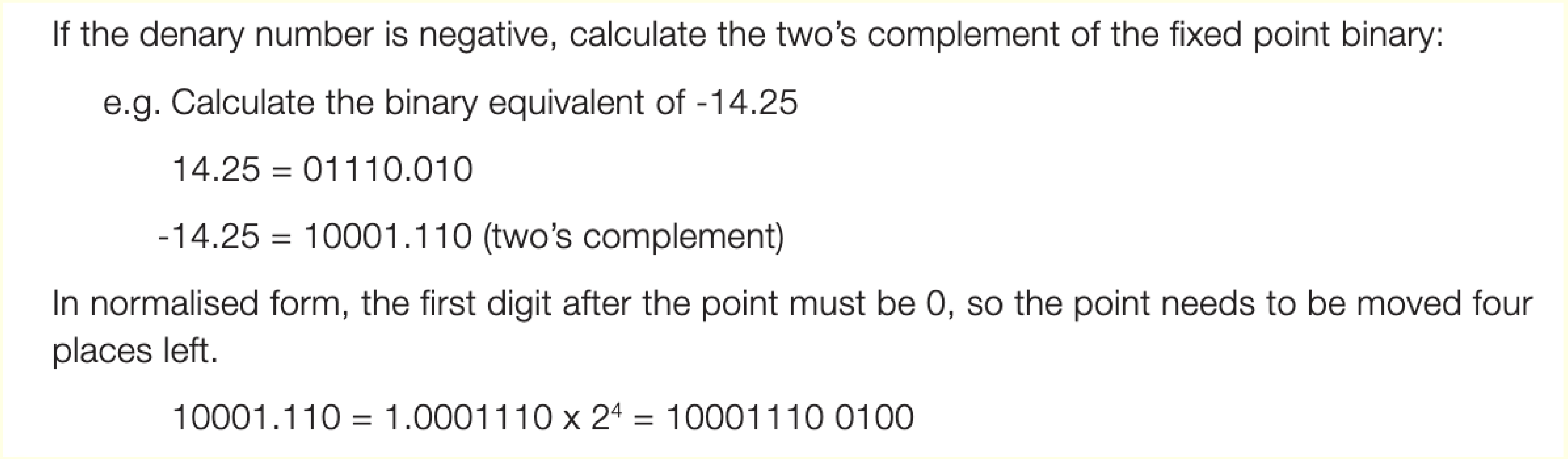
Converting from denary to normalised binary floating point

- To convert a denary number to normalised binary floating point, first convert the number to fixed point binary.



Convert the number 14.25 to normalised floating point binary, using an 8-bit mantissa and a 4-bit exponent.

- In fixed point binary, 14.25 = 01110.010
- Remember that the first digit after the sign bit must be 1 in normalised form, so move the binary point 4 places left and increase the exponent from 0 to 4. The number is equivalent to 0.1110010 x 2^4
- Using a 4-bit exponent, 14.25 = 0 1110010 0100



If the denary number is negative, calculate the two's complement of the fixed point binary:

e.g. Calculate the binary equivalent of -14.25

14.25 = 01110.010

-14.25 = 10001.110 (two's complement)

In normalised form, the first digit after the point must be 0, so the point needs to be moved four places left.

10001.110 = 1.0001110 x 2^4 = 10001110 0100

Spotting a normalised number.

- Which of these are normalised numbers (8 bit scheme, 3 bit exponent, uses twos complement)

1. 00110 011
2. 01100 010

Answer:

The first one is not normalised but the second one is normalised.

They can both represent 3 decimal.

In the first example the binary number is 0.0110 and the exponent is 3 so move the binary point three places to the right. You get 11.0 which is 3 decimal

In the second example the binary number is 0.1100 with the exponent 2 so move the binary point two places to the right and you still get 11.00 which is once again 3 decimal. But note that you now have **2 binary bits after the point**, which indicates you have **more precision** available.

Normalisation – Range and Accuracy

- With floating point representation, the balance between the **range** and **precision** depends on the choice of numbers of bits for the mantissa and the exponent.

- A large number of bits used in the mantissa will allow a number to be represented with **greater accuracy**, but this will **reduce** the number of bits in the exponent and consequently the **range of values** that be represented.

Key Points

- To normalise a floating point number, we ‘float’ the binary point to be in front of the first significant digit and adjust the exponent accordingly.
- We **normalise numbers** in this way to **maximise the accuracy** of the **value stored** and to **avoid multiple representation of the same number.**
- The accuracy of a floating point number depends on the number of digits in the mantissa.
- **More digits in the mantissa** means **fewer in the exponent**, meaning a **smaller range of values can be stored**.
- There is always a trade-off between the range and the accuracy when choosing the size of the mantissa and exponent in a floating point number.

Steps for normalising a floating-point number

Normalise the following numbers, using an 8-bit mantissa and a 4-bit exponent

(a)0.0000110 0011

Step 1: Work out the exponent: + 3

Step 2: Normalise the floating point

Move floating point 4 places to the right 00000.110

As we moved it right then to put it back it would be 4 to the left -4

Step 3 Work out the new exponent

Current value (+ or -) New Exponent +3 -4 = -1 1111

Floating point addition and subtraction

Before looking at these operations in binary, we can gain an understanding of the principles involved in floating point arithmetic by looking at equivalent calculations in denary.

In denary, when adding two numbers involving decimal points, we first have to line up the points.

$$\begin{array}{rr} & 132.156 \\ + & 1.0318 \\ \hline & 133.1878 \end{array}$$

The rules for addition and subtraction can be stated as:

1. **Line up the points by making the exponents equal**
2. **Add or subtract the mantissas**
3. **Normalise the result**

In their "normalised form", the two numbers above would be represented as:

$$.132156 \times 10^3$$

$$.103180 \times 10^1$$

Worked Example: Craig n Dave approach

Add together the two binary numbers given below, leaving the result in **normalised** floating point binary form.

-1 2

0●	1	1	0	0	0	0	0	1	1	1	1		0●	1	1	0	1	0	0	0	0	0	1	0

The rules for addition and subtraction can be stated as:

1. **Line up the points by making the exponents equal**
2. **Add or subtract the mantissas**
3. **Normalise the result**

0●	1	1	0	0	0	0	0	
●0	1	1	0	0	0	0	0	-1 move 1 to left

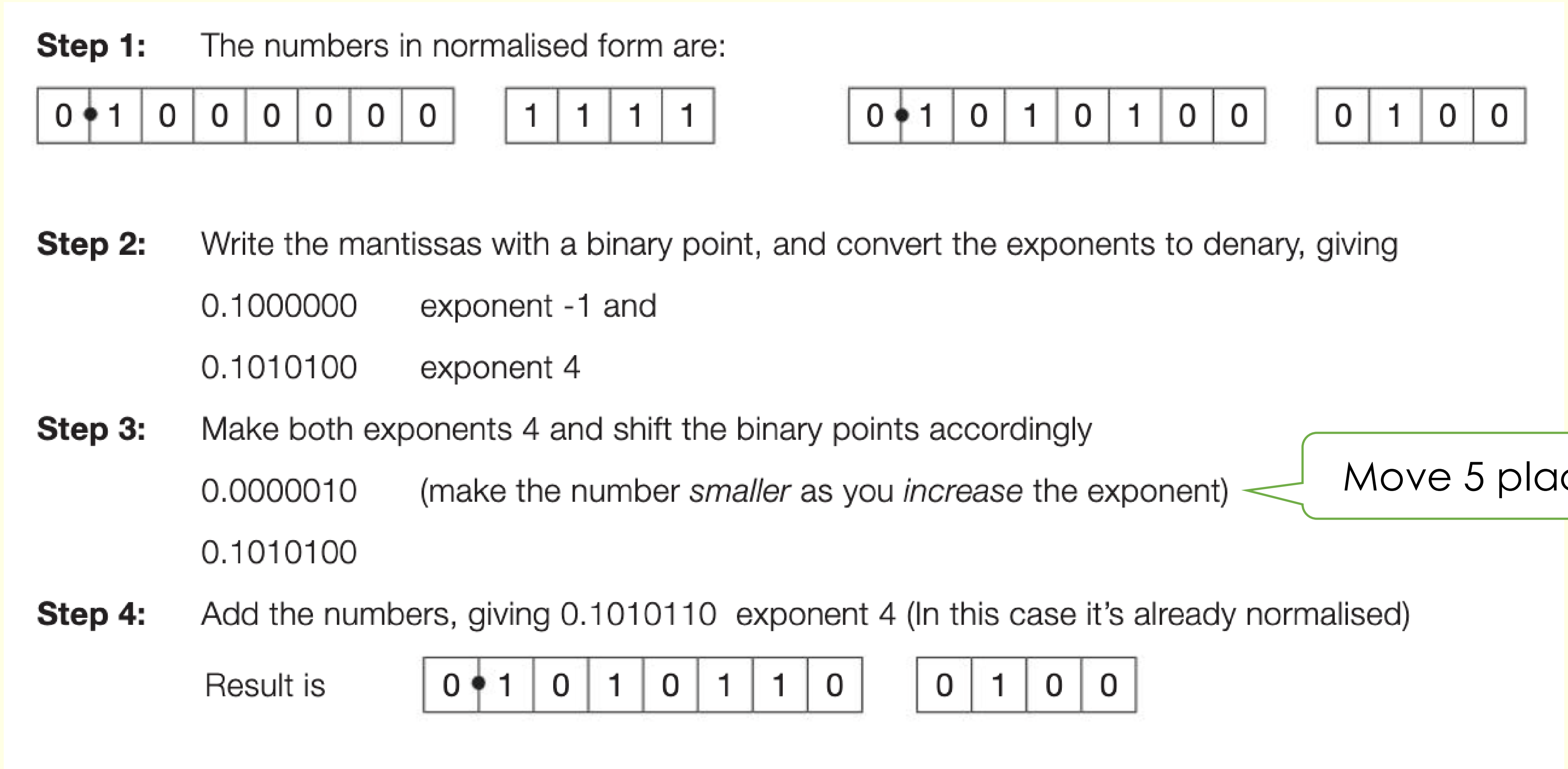
0●	1	1	0	1	0	0	0	
0	1	1●	0	1	0	0	0	2 move 2 to right

0	0●	0	1	1	0	0	0	**Line up the points by making the exponents equal**
1	1●	0	1	0	0	0	0	
1	1●	1	0	1	0	0	0	**Add**
		1						

0●	1	1	1	0	1	0	0	0	0	1	0	Normalise move 2 places to the left + 2

Worked Example – Alternative approach

- Convert the denary numbers 0.25 and 10.5 to normalised floating point binary form using an 8-bit mantissa and a 4-bit exponent.
- Add together the two normalised binary numbers, giving the result in normalised floating point binary form.



Step 1: The numbers in normalised form are:

0 •	1	0	0	0	0	0	0		1	1	1	1

0 •	1	0	1	0	1	0	0		0	1	0	0

Step 2: Write the mantissas with a binary point, and convert the exponents to denary, giving

0.1000000 exponent -1 and

0.1010100 exponent 4

Step 3: Make both exponents 4 and shift the binary points accordingly

0.0000010 (make the number *smaller* as you *increase* the exponent)

0.1010100

Step 4: Add the numbers, giving 0.1010110 exponent 4 (In this case it's already normalised)

Result is

0 •	1	0	1	0	1	1	0		0	1	0	0

Move 5 places to the left

Worked Example - Subtraction

Subtract the second of the two numbers given below from the first, giving the result in normalised floating point binary form.

0•	1	0	0	0	1	0	0		0	1	1	0

0•	1	0	0	0	0	1	0		0	1	0	1

Step 1: Convert the exponents to denary, giving

0.1000100 exponent 6 and

0.1000010 exponent 5

Step 2: Make both exponents 6 and shift the binary point of the second number accordingly

0.1000100 exp 6

0.0100001 exp 6 (make the number *smaller* as you *increase* the exponent)

Step 3: Find the twos complement of the second number

Step 4: Add the numbers

0.1000100

1.1011111

(1)0.0100011 exp 6 (ignore the carry)

Now normalise the number by moving the binary point right 1 place, which increases the number, and decrease the exponent by 1

Result is

0•	1	0	0	0	1	1	0		0	1	0	1

Underflow and overflow

- Underflow occurs when a number is too small to be represented in the allotted number of bits. If, for example, a very small number is divided by another number greater than 1, underflow may occur and the result will be represented by 0.
- Overflow occurs when the result of a calculation is too large to be held in the number of bits allocated.

Notes for exam…

- Learn the approach we have used here for the exam.
- Don't check your answers using ChatGPT as it will use an alternative way of working out floating point and normalising a floating point. At the hardware level, there are different methods for storing floating-point binary numbers, including IEEE 754.
- Use https://www.advanced-ict.info/interactive/normalise.html to check your answers when doing revision.