

Learning Aims

- Normalise un-normalised floating point numbers with positive or negative mantissas



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 5,842,130 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 0.101×2^{-3} or 0101000000 111101	No leading 1s to the left of the mantissa. Starts with 10 Negative number 1.110010100 (10 bits) would become 1.00101×2^2 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 $m \times 2^e$.



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 $0.1111 \times 2^3 = 7.5$

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 $-1.0000 \times 2^3 = -1000.0 = -8$

Size of the mantissa will determine the **precision** of the number, and the size of the exponent will determine the **range** 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

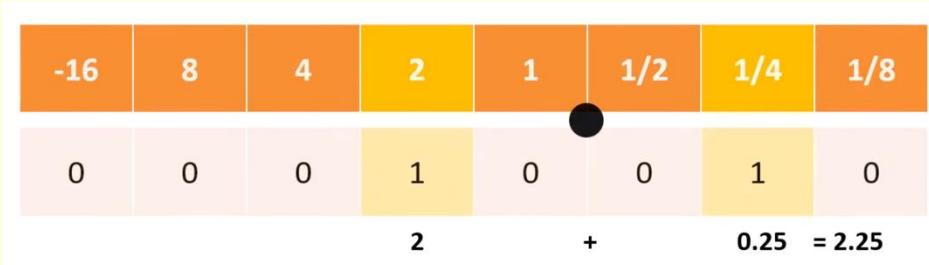
Flip bits 1.101100000

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

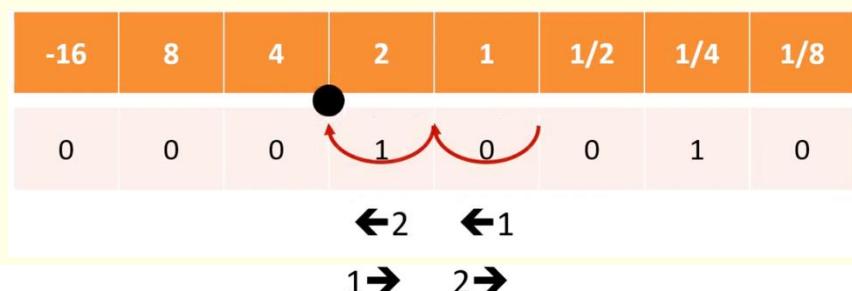
1.01100000×2^{-1} or 101100000 111111



Representing a positive normalised floating point number 2.25



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



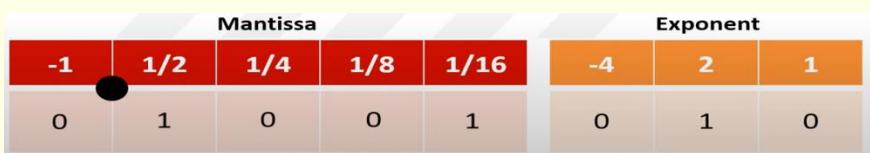
Step 2: move the binary point so it sits between the first 0 and 1

Normalised positive always starts with 01

Move 2 places to the left



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



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.

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

Step 1: Write out the positive version of the number

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

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

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

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

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

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

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 0.1110010×2^4
- Using a 4-bit exponent, $14.25 = 01110010\ 0100$

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

$$14.25 = 01110.010$$

$$-14.25 = 10001.110 \text{ (two's complement)}$$

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

$$10001.110 = 1.0001110 \times 2^4 = 10001110\ 0100$$



Spotting a normalised number.

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

1. 00110 011

Answer: 01100 010

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 can 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

