Unsigned binary number 10000101 to denary.

128	64	32	16	8	4	2	1
1	0	0	0	0	1	0	1

128 + 4 + 1 = 133

Convert the denary number 72 to an unsigned 8-bit integer.

128	64	32	16	8	4	2	1
0	1	0	0	1	0	0	0

Describe why two's complement may be preferable to sign and magnitude.

Calculations are more easily **performed** on two's complement

Two's complement allows for a (negligible) larger range of numbers to be stored.
(2's complement +127 to -128)
(Sign and magnitude 127 to -127)

Explain what is meant by the character set of a computer.

Normally equates to the symbols on a keyboard / digits / letters…

…that can be represented / interpreted / understood by a computer

May include control characters

Denary	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Hexadecimal:

1111 | 0000 Split into 2 nibbles

15 0 Convert to denary

F 0 Convert to Hex

Using the denary number 89 as an example, explain the relationship between binary and hexadecimal representations.

−Split the binary number in groups of 4
−Change each into a single value/(Hexadecimal) digit
−Digits which are between 10 and 15 are given letters A to F
−In this example: 0101 = 5 and 1001 = 9/Therefore 89 = 59(hex)

Using **two's complement** convert the denary number −43 into an 8 bit binary number. You must show your working.

Step 1 work out +43

-128	64	32	16	8	4	2	1
0	0	1	0	1	0	1	1

Step 2 Flip the bits after the first 1 from the right

-128	64	32	16	8	4	2	1
0	0	1	0	1	0	1	1
1	1	0	1	0	1	0	1

Using **sign and magnitude** convert the denary number −43 into an 8 bit binary number. You must show your working.

MSB	64	32	16	8	4	2	1
1	0	1	0	1	0	1	1

Binary Addition

0+ 0 =0
1+0 = 1
1+1 = 10
1+1+1 =11

Worked Examples

a) Add the binary numbers 01111011 and 01101000

Step 1: put the numbers together.

	0	1	1	1	1	0	1	1
+	0	1	1	0	1	0	0	0
	1	1	1	0	0	0	1	1
	1	1	1	1				

Step 2: write the answer.

11100011 (123+104=227)

b) Add the binary numbers 10110110 and 11000111.

Step 1 put the numbers together.

	1	0	1	1	0	1	1	0
+	1	1	0	0	0	1	1	1
	0	1	1	1	1	1	0	1
1					1	1		

There is an extra carry bit left over on this one. This is called overflow. It means that the two (in this case) 8-bit numbers added together need more than 8-bits. They need 9. Show this in the exam to make it clear you know what has happened.

Adding using two's complement numbers

Worked Example

The two large numbers when added together are too large to store in the 8-bit two's complement integer and the value overflows the available bits, creating a negative number.

	-128	64	32	16	8	4	2	1	Denary
	0	1	1	0	0	1	1	1	103
+	0	1	1	1	0	0	1	1	115
=	1	1	0	1	1	0	1	0	-38?
	1	1			1	1	1		

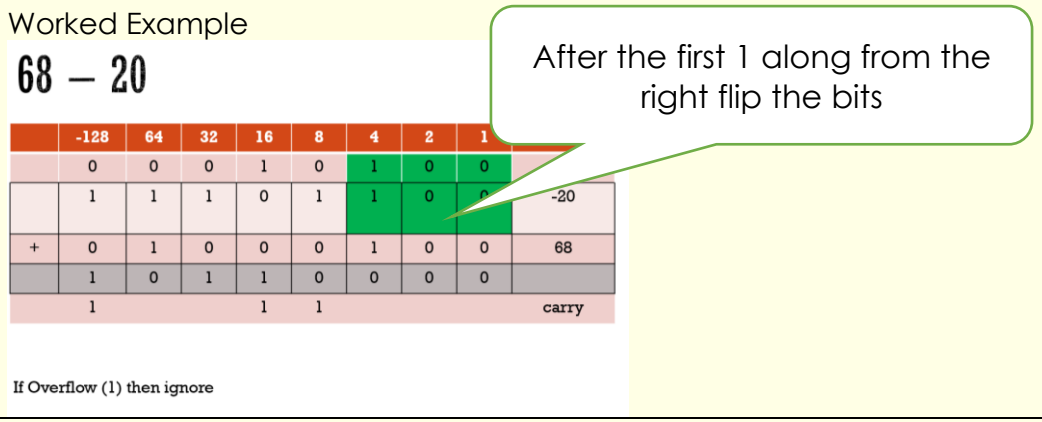
If the calculation were to result in a number that was too small to represent, then this would be called underflow.

Subtracting using two's complement numbers

We convert the number to be subtracted into a negative two's complement number and add them together.

If you get an overflow when subtracting, you lose the 1 value. This will still give the correct positive two's complement value in the 8-bits.

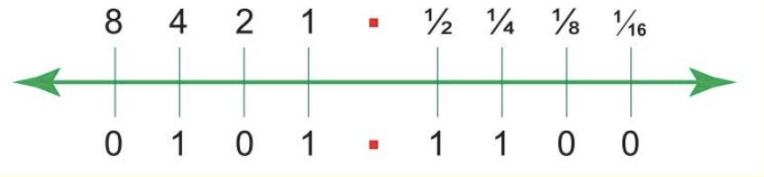
Worked Example



Representing Real Numbers

Fixed point binary numbers

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:**



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

How is 19.25 represented using a single byte with 3 bits after the point?

16	8	4	2	1	0.5	0.25	0.125
1	0	0	1	1	0	1	0

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,

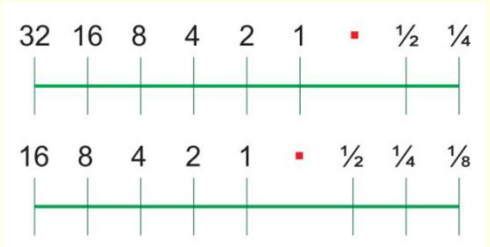
For example, will require an infinite number of bits to the right of the point. The number of fractional places would therefore be truncated and the number will not be accurately stored, causing rounding errors.

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.

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 binary

Represents numbers using a **sign**, **mantissa**, and **exponent** for greater range.

Working Example if mantissa and exponent are positive.

Move the point
3 places to the right

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

$0 \bullet 1011010\ 0011 = 0.101101 \times 2^3 = 0101.101 = 4+1+0.5+0.125 = 5.6$

Worked Example if exponent is negative

Move the point
2 places to the left

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

Negative mantissa

Flip to bits from the first 1 from the right

Move point 5 places to the right

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

Accuracy and Range

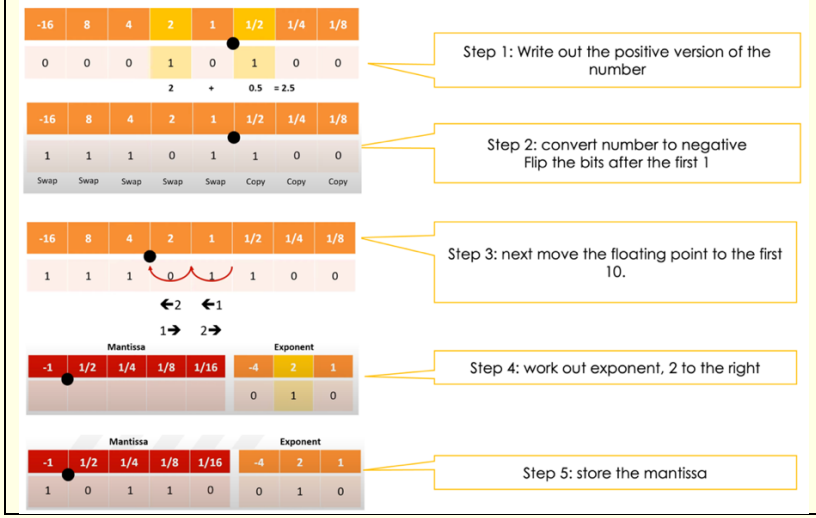
- With floating point representation, the balance between the range and precision depends on the choice of numbers of bits for the mantissa and the exponent.
- A large number of bits used in the mantissa will allow a number to be represented with greater accuracy, but this will reduce the number of bits in the exponent and consequently the range of values that be represented.

Why use normalisation?

- 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.
- Normalisation provides the maximum precision/accuracy with the bits available
- Multiplication is more accurate as there is less truncation of decimal numbers

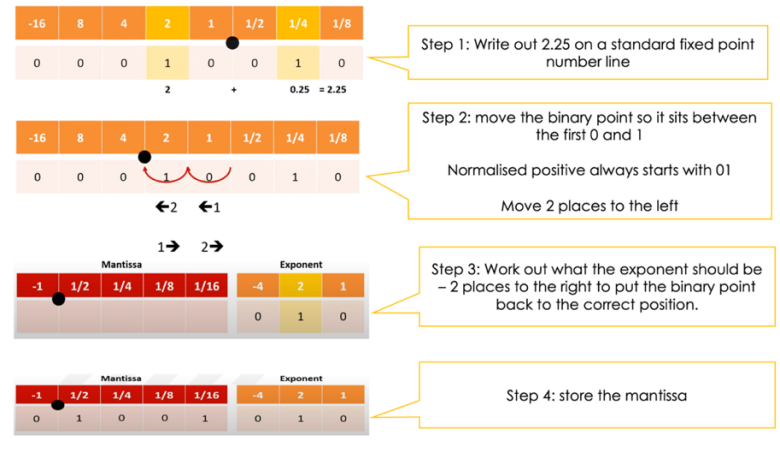
Positive numbers	Negative numbers
No leading 0s to the left of the most significant bit and immediately after the binary point. Starts with 01 The binary fraction 0.000101 becomes 0.101 x 2^{-3} or 0101000000 111101	No leading 1s to the left of the mantissa. Starts with 10 Negative number 1.110010100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

Representing a negative normalised floating point number -2.5

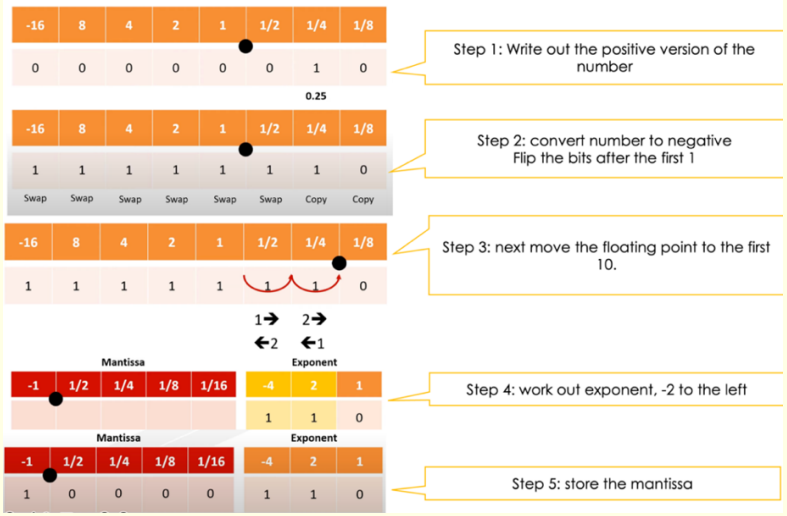


Floating point normalisation

Representing a positive normalised floating point number 2.25



Representing a negative normalised floating point number -0.25



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

1. 00110 011
2. 01100 010

Answer: If the left 2 bits change sign, then that indicates the number is normalised.

They can both represent 3 decimal. But the first one is not normalised but the second one is normalised.

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.

Show the subtraction of these two floating point binary numbers.
Both numbers are stored in a normalised floating point format, using 6 bits for the mantissa and 4 for exponent.
You should show your result in the same format.
Show your working out.

011010 0011 – 010010 0010

Step 1 work out Both exponents
0011 = 3
0010 = 2

Step 2 Both mantissas shifted
0.11010 3 to the right
0110.10
0.10010 2 to the right
010.010

Step 3 Align mantissas

0110.10
0010.01

Step 4 Binary subtraction:

0110.10

Convert 0010.01 - 2's complement
1101.11

Add:
0110.10
1101.11
10100.01

Step 5 normalise

0100.01 Move 3
0011

Correct mantissa 010001

Correct exponent 0011

Character Sets

Explain how codes are used to represent a character set.

- Each symbol has a (binary) code / number…
- …which is unique.
- Number of bits used for one character = 1 byte
- Example code: ASCII / Unicode…
- …uses 8 bits /16 bits per character

Use of more bits for extended character set
ASCII and Unicode.
Similarity:
Both (use binary) to represent characters / are character sets
The first 7/8 bits of Unicode is the same as ASCII (overlaps)
Difference:
ASCII has fewer characters (128/256) / Unicode has more characters
ASCII is 7/8 bits whereas Unicode can be larger 16/32 / can have variable sized characters
ASCII limited to Latin / English / European characters whereas Unicode can represent other symbols (e.g. Chinese/Cyrillic/Emojis)