Data Representation

How computers store numbers, text, sound and images in binary, how hexadecimal and two's complement work, and how files are measured and compressed.

In the world of computers, everything comes down to **just two numbers: 0 and 1**. This is called binary. Computers operate using binary numbers because they are built on digital circuits that recognize two states: on and off.

1.1 Number systems

The Role of Logic Gates

Inside every computer, there are tiny parts called logic gates. These act like traffic lights, deciding how binary data moves around. They make sure everything runs smoothly.

Keeping Data Safe with Registers

Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU. They're like little boxes where the computer stores information until it's needed again.

Denary Number System

Denary, also known as the decimal system, is the numerical language we use every day. **Denary is a base 10 system.** This means that each digit's value depends on its position in the number, with each position representing a power of ten. For example: the rightmost digit represents (10^0, which is 1). The next digit to the left represents (10^1, which is 10), and so on. Below is an example of how 2024 is represented in the denary system.

$$2 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 = 2024$$

Binary Number System

Binary is a base 2 system which means that we count in powers of two. This means that each digit's value depends on its position in the number, with each position representing a power of two. For example:

- The rightmost digit represents 2^0 (1).
- The next digit to the left represents 2^1 (2).
- The next digit represents 2^2 (4), and so on.

Note Academy

1 means that the power of 2 at that position is to be included in the sum, whereas 0 means that the power of 2 at that position is not to be included.

$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13$

Hexadecimal Number System

Hexadecimal is a base 16 system. In the hexadecimal system, each digit can represent values from 0 to 15, allowing for a wider range of numbers to be expressed using fewer characters compared to the decimal system.

Decimal	Hex
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Note Academy

NOTE

Memorise this table.

$2 \times 16^3 + 15 \times 16^2 + 10 \times 16^1 + 0 \times 16^0 = 12192$

The hexadecimal number `2FA0` is made of the digits 2, F (15), A (10) and 0, which is how the sum above is built.

Converting between number systems

TRY IT

Convert 10101101 to denary.

$1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$128 + 0 + 32 + 0 + 8 + 4 + 0 + 1 = 173$

TRY IT

Convert 54 to binary.

NOTE

To convert denary into binary, remember these two things:

1. Always think in powers of two (1, 2, 4, 8, 16, 32)
2. The power of two nearest to and lesser than the number is always on.

$32 + 16 + 4 + 2$

$2^5 + 2^4 + 2^2 + 2^1$

So the bits for 32, 16, 4 and 2 are on, and the bits for 8 and 1 are off: **110110**.

TRY IT

Convert 2FA0 to denary.

$2 \times 16^3 + 15 \times 16^2 + 10 \times 16^1 + 0 \times 16^0 = 12192$

TRY IT

Note Academy

> Convert 198 to hexadecimal.

Method 1: division. To convert the number 198 to hexadecimal, we can use the method of division.

198 divided by 16 equals 12 with a remainder of 6. In hexadecimal, we represent the remainder of 6 as 6 itself.

Next, we divide 12 by 16 which equals 0 with a remainder of 12. In hexadecimal, we represent the remainder of 12 as C.

Therefore, when we combine the **remainders from bottom to top**, the hexadecimal representation of 198 is **C6**.

Method 2: powers of 16.

Power	Value
16^0	1
16^1	16
16^2	256
16^3	4096

> **NOTE**
>
> Memorise this table and try to solve the question mentally.

Use these tips to get the answer:

1. Just like denary to binary, the power nearest to and lesser than the denary value will always be used.
2. Go from larger to smaller power. Guess which is the highest power that will be used and multiply it with a letter or number and get as close to the required number as possible. Then move on to the next power and repeat.

In this case 16^2 (256) cannot be used because it exceeds the number required. However, 16^1 (16) matches our criteria because it is lesser than and nearest to 198. If we multiply 16 by 12 it gives us 192. Any number bigger than that will exceed the required value. Hence at the place where 16 has the power of 1, C (12) will be placed. We now need 6 to reach our required value. Hence in the place of 16 with power 0, 6 will be placed. The final answer will be **C6**.

> **TRY IT**
>
> Convert 10101101 to hexadecimal.

Note Academy

NOTE

To convert binary to hexadecimal, first convert the binary into denary and then convert it to hexadecimal. Similarly, to convert hexadecimal to binary, convert the hexadecimal value to denary and then to binary.

$$1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$128 + 0 + 32 + 0 + 8 + 4 + 0 + 1 = 173$$

The decimal number 173 can be converted to hexadecimal by dividing it by 16 and keeping track of the remainders.

1. Divide 173 by 16, which equals 10 with a remainder of 13.
2. The quotient (10) is then divided by 16, which equals 0 with a remainder of 10.

Now, match the remainders to their hexadecimal equivalents:

- 13 in hexadecimal is D.
- 10 in hexadecimal is A.

Therefore, the decimal number 173 converted to hexadecimal is **AD**.

Reasons to Use Hexadecimal Number System

1. Easier and quicker to understand
2. Easier to debug
3. Takes up less screen space
4. Less likely to make a mistake

Uses of Hexadecimal Number System

Hexadecimal is used where long binary numbers can be represented in a shorter form using hexadecimal. For example:

1. Memory addresses
2. Assembly language
3. Representing colour values

Addition in Binary

Here are the **basic rules** of binary addition:

Note Academy

Sum	Result
$0 + 0 = 0$	No carry
$0 + 1 = 1$	No carry
$1 + 0 = 1$	No carry
$1 + 1 = 10$	Result is 0 and carry 1 to the next higher bit

Given below are the **steps to follow** when adding binary numbers.

- **Align the numbers:** Write the numbers so their least significant bits (rightmost bits) are aligned.
- **Add each pair of bits:** Start from the rightmost bit and move to the left.
- **Carry over:** If a sum exceeds 1, carry the 1 to the next higher bit position.

Let's consider adding two 8-bit binary numbers. Here's an example:

```
  01011011
+ 00110101
----------
  10010000
```

Working from the right, one pair of bits at a time:

Step	Sum	Result	Carry
1	$1 + 1 = 10$	0	1
2	$1 + 0 + 1 = 10$	0	1
3	$0 + 1 + 1 = 10$	0	1
4	$1 + 0 + 1 = 10$	0	1
5	$1 + 1 + 1 = 11$	1	1
6	$0 + 1 + 1 = 10$	0	1
7	$1 + 0 + 1 = 10$	0	1
8	$0 + 0 + 1 = 1$	1	none

Note Academy

Overflow

Overflow occurs when the **result of an arithmetic operation exceeds the range that can be represented with a given number of bits**. In the case of 8-bit binary numbers, the range of representable numbers is from 00000000 (0 in decimal) to 11111111 (255 in decimal). Let's look at an example where overflow occurs:

```
  11011011
+ 10101010
----------
 1 10000101
```

The binary addition results in 110000101. Here, the result has 9 bits because the leading 1 represents an overflow beyond the 8-bit limit. Because more bits are required for the answer than are available, it would lead to an overflow.

Logical Binary Shifts

A logical binary shift involves moving all the bits in a binary number to the left or right, and filling the emptied bit positions with 0s. There are two types of logical shifts: left shift and right shift.

In a **left shift**, each bit in the binary number is shifted one position to the left, and a 0 is inserted at the rightmost bit.

```
Original:    0 1 0 1 1 0 1 1
Left shift:  1 0 1 1 0 1 1 0     (an additional 0 is added on the right)
```

> **The left shift by one bit multiplies the number by 2.**

For 01011011 (which is 91 in decimal), the left-shifted result is 10110110 (which is 182 in decimal).

In a **right shift**, each bit in the binary number is shifted one position to the right, and a 0 is inserted at the leftmost bit.

```
Original:     0 1 0 1 1 0 1 1
Right shift:  0 0 1 0 1 1 0 1      (an additional 0 is added on the left)
```

> **The right shift by one bit divides the number by 2 (ignoring the remainder).**

Note Academy

For 01011011 (which is 91 in decimal), the right-shifted result is 00101101 (which is 45 in decimal).

The bit that is removed during a shift is lost.

Two's Complement Integers

Two's complement is a method for representing signed integers in binary form. It allows for straightforward binary arithmetic and makes it easy to determine the sign of a number.

> **The leftmost bit is the sign bit. 1 means it is negative, 0 means it is positive.**

$$-128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = -1$$

Given above is a representation of a two's complement 8-bit integer (all eight bits set to 1). The leftmost bit, in this case the 8th bit, is negative whereas all others are positive. This is how a negative number can be represented in binary. The **range is from −128 to +127** for an 8-bit integer.

> TRY IT
>
> Convert the positive denary number 23 to an 8-bit two's complement integer.

Convert 23 to binary: 23 = 10111

Ensure it fits in 8 bits: 23 = 00010111

The leftmost bit is 0, indicating a positive number. The two's complement of 23 is simply 00010111. **The representation of a positive integer in two's complement is the same as in regular binary.**

> TRY IT
>
> Convert −23 to an 8-bit two's complement integer.

Method 1

1. Convert 23 to binary: 23 = 10111
2. Ensure it fits in 8 bits: 23 = 00010111
3. Invert the bits: 11101000
4. Add 1: $11101000 + 1 = 11101001$

Final answer is **11101001**.

Method 2

$$-128 + 64 + 32 + 8 + 1 = -23$$

Note Academy

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

By remembering the values represented by each bit of the 8-bit integer, and using mental maths, you can turn the bits on and off to reach the final value. In this case, the 8th, 7th, 6th, 4th, and 1st bits need to be kept on.

1.2 Text, sound and images

Representing Text in Computers

A **character set** contains all the symbols and characters that the computer can use and assigns a unique number (binary/hexadecimal) to each character/symbol.

- **ASCII** (American Standard Code for Information Interchange) is a character encoding standard used to represent text in computers.
- It assigns a unique binary number to each character.
- ASCII uses 7 bits to represent each character, allowing for 128 unique characters.
- **Unicode** is a character encoding standard that extends ASCII to include a vast array of characters from different languages, symbols, and even emojis.
- Unicode was developed to address the limitations of ASCII, which cannot represent non-English characters or many symbols.

Differences between ASCII and Unicode

1. ASCII has fewer characters than Unicode
2. ASCII covers a limited set of languages
3. Unicode includes many more languages
4. ASCII has 128 characters
5. Unicode can represent over a million characters
6. ASCII requires 7 bits per character
7. Unicode requires more bits per character than ASCII (for example 16 or 32 bits)

Representing Sound in Computers

Sampling involves measuring the sound wave at regular intervals and recording these measurements as binary data.

A sound wave is sampled by measuring its amplitude at regular intervals. Each measurement is called a **sample**. Each sample is then converted into a binary number, which represents the amplitude of the

Note Academy

sound wave at that specific point in time.

- **Sample rate**, also known as sampling frequency, is the number of samples taken per second during the sampling process.
- It is measured in Hertz (Hz).
- Higher sample rates result in more samples per second, capturing more detail from the sound wave.
- A sample rate of 44.1 kHz means 44,100 samples are taken each second.
- **Sample resolution**, or bit depth, is the number of bits used to represent each sample that provides the variation in amplitude that can be stored for each sample.
- Higher sample resolution allows more precise representation of the sound wave's amplitude.
- It also provides a greater dynamic range, which means more detailed and accurate sound reproduction.
- A 16-bit resolution means each sample is represented by a 16-bit binary number.

Higher sample rates and resolutions result in more accuracy and **larger file sizes**. This is because more data is recorded per second, and each sample uses more bits.

Representing Images in Computers

A **pixel** is the smallest unit of a digital image. Each pixel represents a single point in the image and has a specific colour. The quality and detail of an image depend on the number and arrangement of pixels.

- **Resolution** refers to the number of pixels in an image, typically expressed as width × height (e.g. 1920 × 1080 pixels).
- Higher resolution means more pixels, which generally results in a more detailed and clearer image.
- Higher resolution increases the file size because more pixel data needs to be stored.
- **Colour depth**, or bit depth, is the number of bits used to represent the colour of each pixel.
- Higher colour depth allows for more colours to be represented, resulting in more vibrant and accurate images.
- For example, 8-bit colour depth can represent 256 colours.
- Higher colour depth increases the file size because more data is needed to represent each pixel's colour.

1.3 Data storage and compression

Units of Data Storage

Bit is the smallest unit of data in computing, representing a binary value of 0 or 1.

Nibble is a group of 4 bits, e.g. 1101.

Note Academy

Byte is a group of 8 bits.

Name	Symbol	Relationship with other units
kibibyte	KiB	1024 bytes
mebibyte	MiB	1024 kibibytes
gibibyte	GiB	1024 mebibytes
tebibyte	TiB	1024 gibibytes
pebibyte	PiB	1024 tebibytes
exbibyte	EiB	1024 pebibytes

The units above are the ones the syllabus uses, and calculations must use 1024. The decimal units below use 1000 and are shown for comparison.

Name	Symbol	Relationship with other units
kilobyte	KB	1000 bytes
megabyte	MB	1000 kilobytes
gigabyte	GB	1000 megabytes
terabyte	TB	1000 gigabytes
petabyte	PB	1000 terabytes
exabyte	EB	1000 petabytes

Note Academy

TRY IT

An image is stored on a computer. The image is 16-bit colour and is 100 pixels high and 150 pixels wide. Calculate the file size of the image in bytes.

$$\frac{16\times100\times150}{8} = 30000 \text{ bytes}$$

- **16:** each pixel uses 16 bits.
- **100 × 150:** the total number of pixels.
- **8:** 1 byte has 8 bits.

TRY IT

> A sound file has a 40 kHz sample rate, 8-bit resolution and is 30 seconds long. Calculate the file size in mebibytes.

$\frac{40000\times8\times30}{8\times1024\times1024} \approx 1.14$ mebibytes

- **40000 × 8 × 30:** 40000 samples are taken each second. Each sample is 8 bits long. This is done 30 times.
- **÷ 8:** 1 byte has 8 bits.
- **÷ 1024, then ÷ 1024 again:** to convert bytes into kibibytes, then into mebibytes.

Dividing by 1024 only once gives the size in kibibytes: 1171.9 KiB.

Data Compression

Data compression is a technique used to **reduce the size of files and data**. Below are some of the reasons why data is compressed:

1. Compressed files require **less bandwidth to transmit over a network**.
2. Storing compressed files means using **less disk space**, allowing more files to be stored on a given storage medium.
3. Compressed files **can be transmitted more quickly over networks** because they contain less data.

- **Lossy compression** reduces file size by permanently removing some data, which can lead to a **loss in quality**.
- It is often used for multimedia files like images, audio, and video, where some loss of quality is acceptable for significantly smaller file sizes.
- Common lossy compression formats include JPEG, MP3, and MPEG.

For images and videos, **lowering the resolution or the number of colours** can reduce file size. For example, changing a high-definition image to a lower resolution or reducing the colour palette from millions of colours to just a few.

For audio, **decreasing the sample rate or the bit depth** can significantly reduce file size. For example, converting audio from 44.1 kHz to 22 kHz or from 16-bit to 8-bit.

Lossless compression reduces file size **without losing any data**. It is used when it is important to retain the original data exactly, such as in text documents and executable files. Common lossless compression formats include ZIP and PNG.

Run Length Encoding (RLE) is a simple form of lossless compression that replaces sequences of the same data value with a single data value and a count. For example, the string "AAAABBBCCDAA" would be compressed to "4A3B2C1D2A".

Note Academy