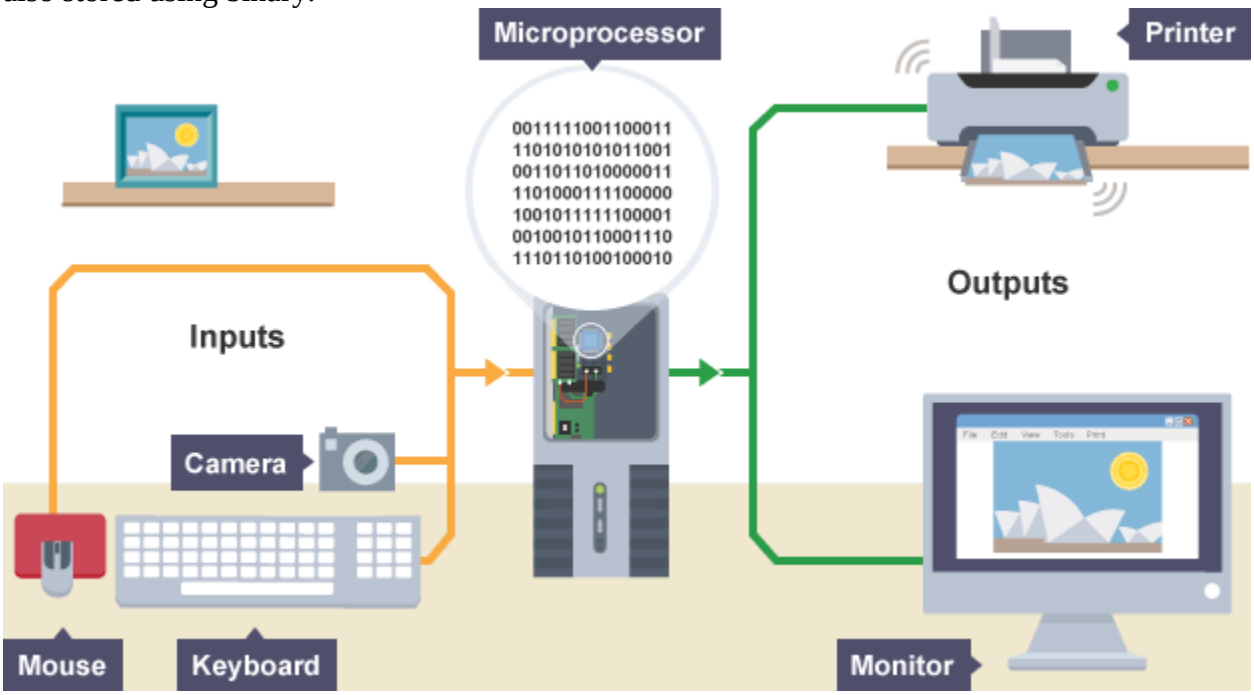
# Bits and binary

Computers use **binary** - the digits 0 and 1 - to store data. A binary digit, or **bit**, is the smallest unit of data in computing. It is represented by a 0 or a 1. **Binary numbers are made up of binary digits (bits)**, eg the binary number **1001**.
The circuits in a computer's processor are made up of billions of **transistors**. A transistor is a tiny switch that is activated by the electronic signals it receives. **The digits 1 and 0 used in binary reflect the on and off states of a transistor**.
Computer programs are sets of instructions. Each instruction is translated into **machine code** - simple binary codes that activate the **CPU**. Programmers write computer code and this is converted by a **translator** into binary instructions that the processor can **execute**.
All **software**, music, documents, and any other information that is processed by a computer, is also stored using binary.




# Encoding

Everything on a computer is represented as streams of binary numbers. **Audio, images and characters all look like binary numbers in machine code**. These numbers are encoded in different data formats to give them meaning, eg the 8-bit pattern **01000001** could be the number **65**, the character '**A**', or a colour in an image.

Encoding formats have been standardised to help compatibility across different platforms. For example:

- audio is encoded as audio file formats, eg mp3, WAV, AAC
- video is encoded as video file formats, eg MPEG4, H264
- text is encoded in character sets, eg ASCII, Unicode

- images are encoded as file formats, eg BMP, JPEG, PNG

The more bits used in a pattern, the more combinations of values become available. This larger number of combinations can be used to represent many more things, eg a greater number of different symbols, or more colours in a picture.

# Did you know?

In the early days of computing, the only way to enter data into a computer was by flicking switches or by feeding in punched cards or punched paper tape.

Since computers work using binary, with data represented as 1s and 0s, both switches and punched holes were easily able to reflect these two states - 'on' to represent 1 and 'off' to represent 0; a hole to represent 1 and no hole to represent 0.
Charles Babbage's Analytical Machine (in 1837) and the **Colossus**(used during the Second World War) were operated using punched cards and tapes. Modern computers still read data in binary form but it is much faster and more convenient to read this from microchips or from magnetic or optical disks.

# Bits and bytes

**Bits** can be grouped together to make them easier to work with. **A group of 8 bits is called a byte**.

Other groupings include:
- **Nibble** - 4 bits (half a byte)
- **Byte** - 8 bits
- **Kilobyte** (KB) - 1000 bytes
- **Megabyte** (MB) - 1000 kilobytes
- **Gigabyte** (GB) - 1000 megabytes
- **Terabyte** (TB) - 1000 gigabytes

Most computers can process millions of bits every second. A **hard drive's** storage capacity is measured in gigabytes or terabytes. **RAM** is often measured in megabytes or gigabytes.

# Amount of storage space required

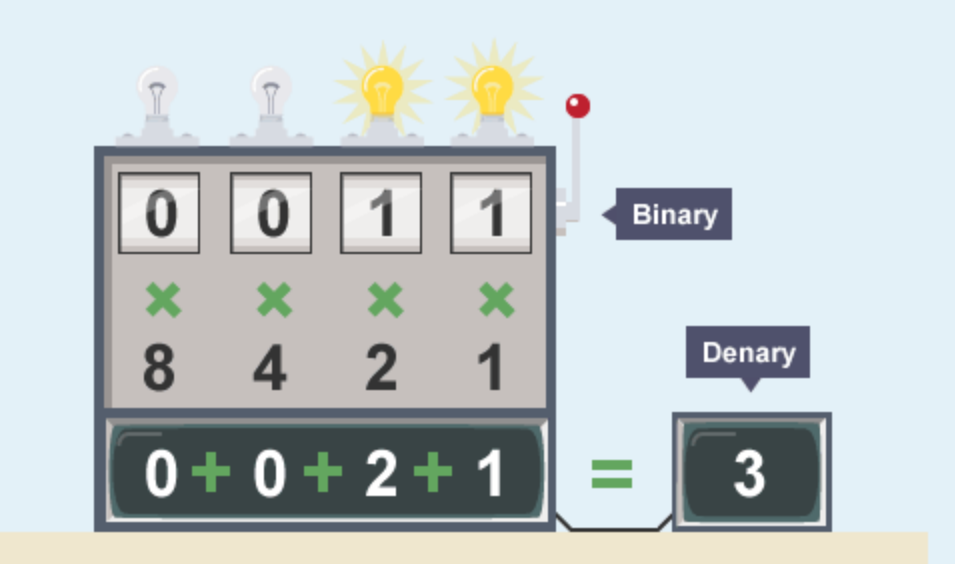
Different types of data require different amounts of storage space. Some examples of this follow:

| Data | Storage |
| --- | --- |
| One extended-ASCII character in a text file (eg 'A') | 1 byte |

| Data | Storage |
| --- | --- |
| The word 'Monday' in a document | 6 bytes |
| A plain-text email | 2 KB |
| 64 pixel x 64 pixel GIF | 12 KB |
| Hi-res 2000 x 2000 pixel RAW photo | 11.4 MB |
| Three minute MP3 audio file | 3 MB |
| One minute uncompressed WAV audio file | 15 MB |
| One hour film compressed as MPEG4 | 4 GB |

# Binary and denary

The **binary** system on computers uses combinations of 0s and 1s.
In everyday life, we use numbers based on combinations of the digits between 0 and 9. This counting system is known as **decimal**, **denary** or **base 10**.
A number **base** indicates how many digits are available within a numerical system. Denary is known as **base 10** because there are ten choices of digits between 0 and 9. For binary numbers there are only two possible digits available: 0 or 1. The binary system is also known as **base 2**.
All denary numbers have a binary equivalent and it is possible to **convert** between denary and binary.




# Place values

# Denary place values

Using the **denary** system, **6432** reads as six thousand, four hundred and thirty two. One way to break it down is as:

- **six** thousands
- **four** hundreds
- **three** tens
- **two** ones

Each number has a **place value** which could be put into columns. Each column is a power of ten in the base 10 system:

| Thousands 1000s ($10^3$) | Hundreds 100s ($10^2$) | Tens 10s ($10^1$) | Ones 1s ($10^0$) |
|---|---|---|---|
| 6 | 4 | 3 | 2 |

Or think of it as:
(**6** x 1000) + (**4** x 100) + (**3** x 10) + (**2** x 1) = **6432**

# Binary place values

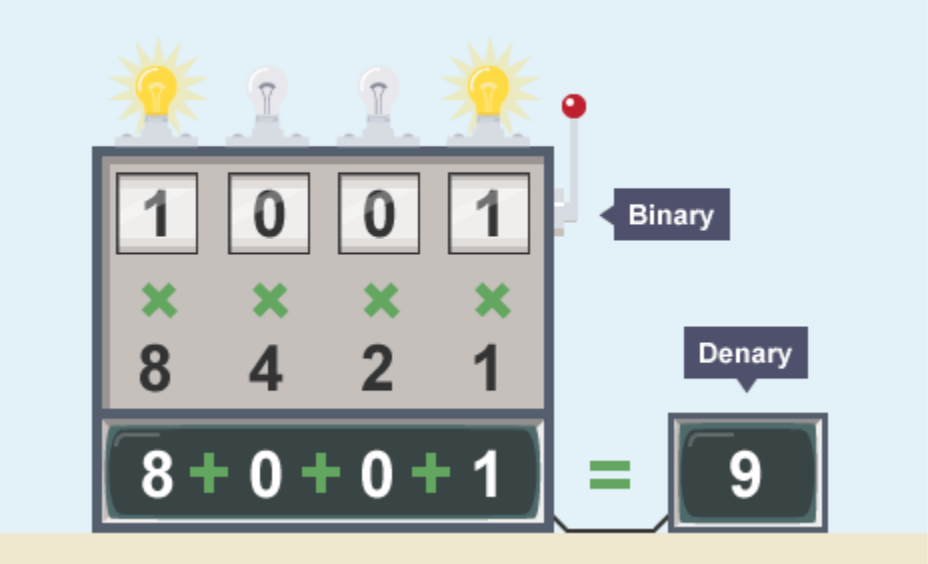
You can also break a **binary** number down into place-value columns, but each column is a power of two instead of a power of ten.

For example, take a binary number like **1001**. The columns are arranged in multiples of 2 with the binary number written below:

| Eights<br>8s<br>($2^3$) | Fours<br>4s<br>($2^2$) | Twos<br>2s<br>($2^1$) | Ones<br>1s<br>($2^0$) |
|---|---|---|---|
| 1 | 0 | 0 | 1 |

By looking at the place values, we can calculate the equivalent denary number.

That is:
(**1** x 8) + (**0** x 4) + (**0** x 2) + (**1** x 1) = 8 + 1 = **9**



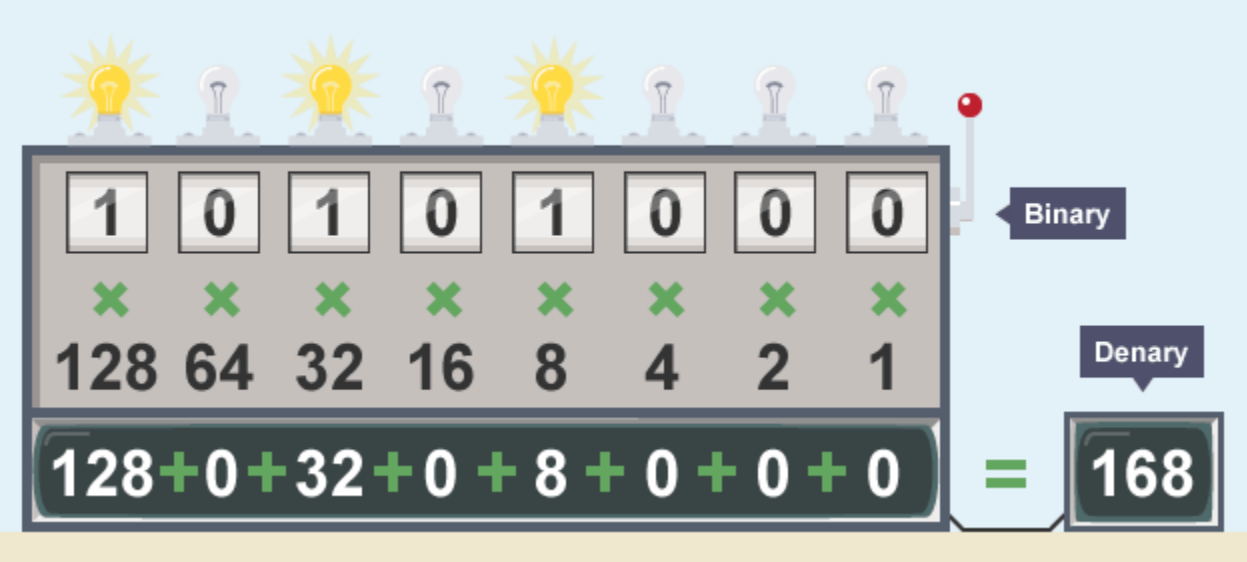

# Converting binary to denary

To calculate a large **binary** number like **10101000** we need more place values of multiples of 2.

- $2^7 = 128$
- $2^6 = 64$
- $2^5 = 32$
- $2^4 = 16$
- $2^3 = 8$

- $2^2 = 4$
- $2^1 = 2$
- $2^0 = 1$

In **denary** the sum is calculated as:
(**1** x 128) + (**0** x 64) + (**1** x 32) + (**0** x 16)+ (**1** x 8) + (**0** x 4) + (**0** x 2) + (**0** x 1) = 128 + 32 + 8 = **168**




The table below shows denary numbers down the left with their equivalent binary numbers marked out below the base 2 columns. Each individual column in the table represents a different **place value**equivalent to the base 2 powers.

# Converting denary to binary: Method 1

There are two methods for converting a **denary** (base 10) number to **binary** (base 2). This is method one.

# Divide by two and use the remainder

Divide the starting number by 2. If it divides evenly, the binary digit is 0. If it does not - if there is a remainder - the binary digit is 1.

Sorry, this clip is not available in your region or territory.

A method of converting a denary number to binary

**Download** Transcript

### Worked example: Denary number 83

1. **83** ÷ 2 = 41 remainder **1**
2. **41** ÷ 2 = 20 remainder **1**

3. **20** ÷ 2 = 10 remainder **0**
4. **10** ÷ 2 = 5 remainder **0**
5. **5** ÷ 2 = 2 remainder **1**
6. **2** ÷ 2 = 1 remainder **0**
7. **1** ÷ 2 = 0 remainder **1**

Put the remainders in **reverse** order to get the final number: **1010011**.

| 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|
| **1** | **0** | **1** | **0** | **0** | **1** | **1** |

To check that this is right, convert the binary back to denary:
(**1** x 64) + (**0** x 32) + (**1** x 16) + (**0** x 8) + (**0** x 4) + (**1** x 2) + (**1** x 1) = **83**

### Worked example: Denary number 122

1. **122** ÷ 2 = 61 remainder **0**
2. **61** ÷ 2 = 30 remainder **1**
3. **30** ÷ 2 = 15 remainder **0**
4. **15** ÷ 2 = 7 remainder **1**
5. **7** ÷ 2 = 3 remainder **1**
6. **3** ÷ 2 = 1 remainder **1**
7. **1** ÷ 2 = 0 remainder **1**

Put the remainders in **reverse** order to get the final number: **1111010**.

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

To check that this is right, convert the binary back to denary:
(**1** x 64) + (**1** x 32) + (**1** x 16) + (**1** x 8) + (**0** x 4) + (**1** x 2) + (**0** x 1) = **122**

**The binary representation of an even number always ends in 0 and an odd number in 1.**

# Converting denary to binary: Method 2

There are two methods for converting a **denary** (base 10) number to **binary** (base 2). This is method two.

# Take off the biggest $2^n$ value you can

Remove the $2^n$ numbers from the main number and mark up the equivalent $2^n$ column with a 1. Work through the remainders until you reach zero. When you reach zero, stop and complete the final columns with 0s.

Sorry, this clip is not available in your region or territory.

A method of converting a denary number to binary

Download Transcript

### Worked example: Denary number 84

First set up the columns of base 2 numbers. Then look for the highest $2^n$number that goes into 84.

1. Set up the columns of base 2 numbers
2. Find the highest $2^n$ number that goes into 84. The highest $2^n$ number is 26 = **64**
3. 84 – 64 = 20. Find the highest $2^n$ number that goes into 20. The highest $2^n$ number is 24 = **16**
4. 20 - 16 = 4. Find the highest $2^n$ number that goes into 4. The highest $2^n$ number is 22 = **4**
5. 4 - 4 = 0
6. Mark up the columns of base 2 numbers with a 1 where the number has been the highest $2^n$number, or with a 0:

| 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|
| **1** | **0** | **1** | **0** | **1** | **0** | **0** |

Result: **84** in denary is equivalent to **1010100** in binary.

To check that this is right, convert the binary back to denary:
(**1** x 64) + (**0** x 32) + (**1** x 16) + (**0** x 8) + (**1** x 4) + (**0** x 2) + (**0** x 1) = **84**

# Bit number patterns

Computer systems and files have limits that are measured in **bits**. For example, image and audio files have **bit depth**.
The bit depth reflects the number of **binary** numbers available, similar to the number of combinations available on a padlock. The more wheels of numbers on a padlock, the more combinations of numbers are possible. The greater the bit depth, the more combinations of binary numbers are possible. Every time the bit depth increases by one, the number of binary combinations is doubled.

A 1-bit system uses combinations of numbers up to one place value (1). There are just two options: 0 or 1.

A 2-bit system uses combinations of numbers up to two place values (11). There are four options: 00, 01, 10 and 11.

| Bit depth | Max (binary) | Max (denary) | Combinations available |
|---|---|---|---|
| 1 | 1 | 1 | 2 |
| 2 | 11 | 3 | 4 |
| 3 | 111 | 7 | 8 |
| 4 | 1111 | 15 | 16 |
| 5 | 11111 | 31 | 32 |

A 1-bit image can have 2 colours, a 4-bit image can have 16, an 8-bit image can have 256, and a 16-bit image can have 65,536.

# Binary combinations

These tables show how many binary combinations are available for each bit size.

# One bit

| | | Binary number |
|---|---|---|
| | | Place value 1 |
| Denary number | 0 | 0 |
| | 1 | 1 |

Maximum binary number = 1

Maximum denary number = 1

Binary combinations = 2

# Two bit

| | | Binary pattern | |
|---|---|---|---|
| | | Place value 2 | Place value 1 |
| Denary number | 0 | 0 | 0 |
| | 1 | 0 | 1 |
| | 2 | 1 | 0 |
| | 3 | 1 | 1 |

Maximum binary number = 11

Maximum denary number = 3

Binary combinations = 4

# Three bit

| | | Binary pattern | | |
|---|---|---|---|---|
| | | Place value 4 | Place value 2 | Place value 1 |
| Denary number | 0 | 0 | 0 | 0 |
| | 1 | 0 | 0 | 1 |
| | 2 | 0 | 1 | 0 |
| | 3 | 0 | 1 | 1 |
| | 4 | 1 | 0 | 0 |
| | 5 | 1 | 0 | 1 |
| | 6 | 1 | 1 | 0 |
| | 7 | 1 | 1 | 1 |

Maximum binary number = 111

Maximum denary number = 7

Binary combinations = 8