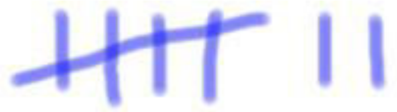
Binary numbers

1. Introduction to Numbers

Computers understand the world through binary numbers - sequences of 1s and 0s.

But what is a binary number? How does it relate to the numbers we use every day?
The answer is that both binary and our 'normal' numbers, called **denary**, are just different representations of the same thing.

What is the number shown below?



We are pretty sure that you would have instantly known that it was 7. But how did you know that? It doesn't look anything like a 7.

The point is that a number can be represented as any set of symbols as long as the rules are consistent. What we are basically trying to point out is that a 'number' can be represented in many different forms.

This section will look at how to convert numbers from the form we're most familiar with - denary - into binary and then back again.

Binary numbers

2. Denary and Binary

The two number systems we will discuss in this section are denary and binary.

The base of a number system describes how many symbols (agreed shapes) there are in that system. Therefore denary has ten symbols, and binary has two symbols.

There are others as well, for example hexadecimal which has sixteen symbols.

Base 10 number system (denary)

In our everyday lives we use a denary number system which has the number symbols 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.

This is called a base-10 number system, because there are ten symbols involved.

Here are some examples of denary numbers:

- 5
- 249
- 316
- 8715

Base 2 number system (binary)

You already know that computers need to use binary numbers to process data.

Binary is a base-2 type of number which has only two symbols, and these are chosen to be 1 or a 0

Here are some examples of binary numbers:

- 1
- 101
- 1101
- 11011001

Binary numbers

3. Denary number position and value

When working with any number system, the *position* of the digit or symbol is important in order to be able to calculate its value.

Let's look at the denary number 123.

The number on the far right, 3, is worth 3. But the number to the left of 3 isn't worth 2. Instead, it is worth 20. Because its position is one to the left of 3, it has been multiplied by 10, so it is (10 * 2)

Now think about the number 1 in 123. Again, this isn't worth the value of 1, instead it is multiplied by 100 because it is two to the left. Its actual value is (100 * 1)

The value of the decimal number 123 is arrived at by using the following calculation:

3 + (10 * 2) + (100 * 1)

In the denary system the rightmost position is labeled "position 0".

Each time you move one position left, the value of a digit is multiplied by 10.

So a digit in position 1 is multiplied by 10. A digit in position 2 is multiplied by 100.

The following table shows the value of a "1" digit, depending on its position within a denary number

Position	5	4	3	2	1	0
Multiplier	10^{5105}	10^{4104}	10^{3103}	10^{2102}	10^{1101}	10^{0100}
Value	100000	10000	1000	100	10	1

Binary numbers

4. Binary number position and value

Just like the denary system, the position of a digit in a binary number determines its actual value and just like denary, a binary number reads from right to left.

In the binary system the rule is that each time you move one position left, the value of a digit is multiplied by a power of 2, with the power being its position. So a digit in position 4 is multiplied by 2^{424}

The following table shows the value of a "1" digit, depending on its position within a binary number

Position	7	6	5	4	3	2	1	0
Multiplier	2^{727}	2^{626}	2^{525}	2^{424}	2^{323}	2^{222}	2^{121}	2^{020}
Value	128	64	32	16	8	4	2	1

For example a binary digit in position five needs to be multiplied by 32. A binary number with a value of 1 at position five looks like this (don't forget - the first position is position zero):

00100000

This represents thirty two as a binary number.

Binary numbers

5. Calculating the value of binary numbers

Position	7	6	5	4	3	2	1	0
Multiplier	2^{7}27	2^{6}26	2^{5}25	2^{4}24	2^{3}23	2^{2}22	2^{1}21	2^{0}20
Value	128	64	32	16	8	4	2	1

We can make use of the table above by placing each digit of the binary number in the correct column according to its position in the number

Example 1: what is the value of the binary number **00000001** ?

Place each digit of the number into the correct column, then multiply it by its multiplier

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

Multiplying a number by zero is still zero, so ignore any zeroes to the left of the leftmost 1 in the binary number

Answer: the number **00000001** represents 'one'

Working from the right: (1 * 1) = 1

Example 2: what is the value of the binary number **00000011**?

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

Answer: the number **00000011** represents 'three'

Working from the right: (1 * 1) + (2 * 1) = 3

Example 3: What is the value of the binary number **00100101** ?

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

Answer: the number **00100101** represents 'thirty seven'.

Working from the right: (1 * 1) + (2 * 0) + (4 * 1) + (8 * 0) + (16 * 0) + (32 * 1) = 37

Example 4: What is the value of the binary number **11111111** ?

Multiplier	128	64	32	16	8	4	2	1
Number	1	1	1	1	1	1	1	1

Answer: the number **11111111** represents 'two hundred and fifty five'. This is the largest number that can be represented with an 8 bit binary number i.e. a byte.

Workings: (1 * 1) + (2 * 1) + (4 * 1) + (8 * 1) + (16 * 1) + (32 * 1) + (64 * 1) + (128*1) = 255

Binary numbers

6. Denary to binary conversion

On the previous page we saw how a binary number has a value depending on the arrangement of 1s and 0s in the number.

It should be apparent that the same number can be represented in the denary system as well.

We shall now describe a method of converting any denary number from 0 to 255 into its equivalent binary number.

We will be using denary number 211 in our example

Step 1.

Start off with the table below that has all zeros in the second row

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

This is the equivalent of binary zero.

Step 2.

Is 211 larger than 128?

If yes, put a 1 underneath the 128 in the table. If not, leave it at 0

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

Step 3.

As we put a 1 under 128, subtract 128 from 211 to see what is left

211 - 128 = 83

Step 4.

We have 83 left

Is 83 larger than 64? If yes, put a 1 in the space under 64, if not, leave it at 0

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

Step 5.

We started with 83 and we put a 1 under 64. Thus, subtract 64 from 83 to see what is left:

83 - 64 = 19

Step 6.

We have 19 left.

Is 19 larger than 32? If yes, put a 1 in the space under 32, if not, leave it at 0

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

Step 7.

As 19 was NOT larger than 32, we left it at 0.

Is 19 larger than 16? If yes, put a 1 in the space under 16, if not, leave it at 0

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

Step 8.

We started with 19 and we put a 1 under 16. Thus, subtract 16 from 19 to see what is left:

19 - 16 = 3

Step 9.

We have 3 left.

Is 3 larger than 8? If yes, put a 1 in the space under 8, if not, leave it at 0

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

Step 10.

As 3 was NOT larger than 8, we left it at 0.

Is 3 larger than 4? If yes, put a 1 in the space under 4, if not, leave it at 0

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

Step 11.

As 3 was NOT larger than 4, we left it at 0.

Is 3 larger than 2? If yes, put a 1 in the space under 2, if not, leave it at 0

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

Step 12.

We started with 3 and we put a 1 under 2. Thus, subtract 2 from 3 to see what is left:

3 - 2 = 1

Step 13.

Is there a 1 left over? If yes, place a 1 in the space under 1, if not leave it at 0

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

There was a 1 left over and so a 1 was placed in the last position.

ANSWER:
Using the method above, the denary number has been converted to its equivalent binary **11010011**

Binary numbers

7. Binary to Denary conversion

We have looked at how to convert a denary or decimal number into binary. You also need to be able to do this conversion the other way around, i.e. convert binary into denary.

Let's look at a reliable method of doing this by using the binary number **10010101**

Again, we use a table to help us.

Place the binary number to be converted in the second row of the table

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

Step 1.

Multiply the number in the top row of column 1 by the number in the bottom row of column 1:

128	= 128
1	

Step 2:

Multiply the number in the top row of column 2 by the number in the bottom row of column 2:

64	= 0
0	

Step 3:

Multiply the number in the top row of column 3 by the number in the bottom row of column 3:

32	= 0
0	

Step 4:

Multiply the number in the top row of column 4 by the number in the bottom row of column 4:

16	= 16
1	

Step 5:

Multiply the number in the top row of column 5 by the number in the bottom row of column 5:

8	= 0
0	

Step 6:

Multiply the number in the top row of column 6 by the number in the bottom row of column 6:

4	= 4

1	

Step 7:

Multiply the number in the top row of column 7 by the number in the bottom row of column 7:

2	= 0
0	

Step 8:

Multiply the number in the top row of column 8 by the number in the bottom row of column 8:

1	= 1
1	

Step 9:

Add up all of the results from stages 1 to 8:

128 + 0 + 0 + 16 + 0 + 4 + 0 + 1 = 149

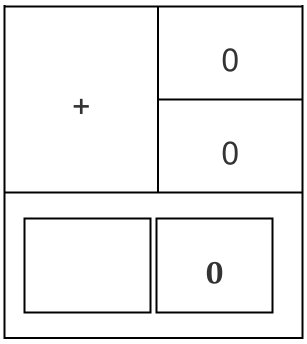
Therefore the binary number **10010101** has been converted into 149 denary (decimal)

Binary numbers

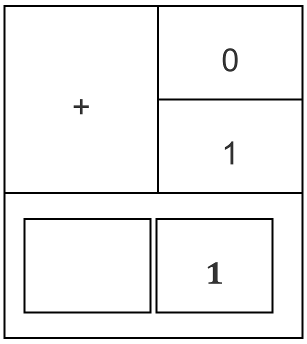
8. Adding binary numbers

Binary numbers can easily be added using a few addition rules. These are

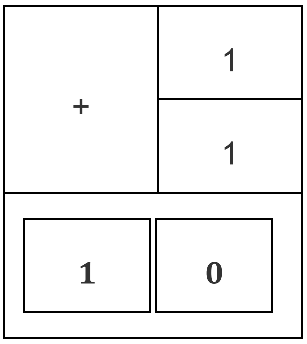
0 + 0 = 0



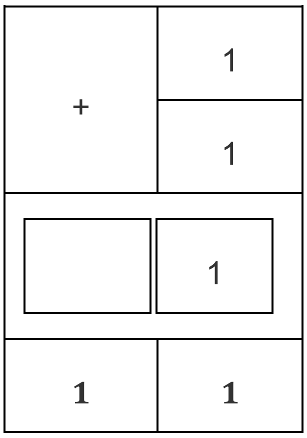
0 + 1 = 1



1 + 1 = 10



1+1+1 = 11



This last one is dealing with a possible carry 1 from the previous calculation

Binary numbers

9. Adding binary numbers - a worked example

We are going to add together the following two binary numbers: 00011101 and 10011011

Step 1

Start off by putting them on the top two rows in a four row table:

0	0	0	1	1	1	0	1
1	0	0	1	1	0	1	1

The third row will be used to hold any 'carry over' numbers

The fourth row will be used to hold the answer to the calculation

Step 2

We start adding numbers together going from the rightmost position.

Adding the column on the right hand side which contains the following:

1
1

From the rules on the previous page, we know that 1 + 1 =10.

We put a 0 on the bottom row of column 8 and we carry the 1 over to the third row of column 7 so we can use it in the next step (see table below)

Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
0	0	0	1	1	1	0	1
1	0	0	1	1	0	1	1
						1	
							0

Step 3

The column on the right (column 8) is now finished with. We move onto column 7.

Column 7 contains the numbers:

0
1
1

The top two rows are from the original number and the third row has been carried over from the previous calculation.

We add these three rows together. Adding 1 + 1 gives us the result 10.

We put 0 in the bottom row of column 7 and carry the 1 to the third row of column 6

Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
0	0	0	1	1	1	0	1
1	0	0	1	1	0	1	1

					1	1	
						0	**0**

Step 4

Column 7 is now finished with. We now work on column 6.

Column 6 contains the numbers:

1
0
1

We know that adding 1 + 1 gives us the result 10.

We put 0 in the bottom row of column 6 and carry over 1 to column 5.

Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
0	0	0	1	1	1	0	1
1	0	0	1	1	0	1	1
				1	1	1	
					0	**0**	**0**

Step 5

Column 6 is now finished with. We now work on column 5.

Column 5 contains the numbers:

1

1
1

We know that adding 1 + 1 gives us the result 10 then adding a third 1 gives 11.

We put 1 in the bottom row of column 5. Again, we can carry 1 over to column 4.

Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
0	0	0	1	1	1	0	1
1	0	0	1	1	0	1	1
			1	1	1	1	
				1	**0**	**0**	**0**

Step 6

Column 5 is now finished with. We now work on column 4.

Column 4 contains the numbers:

1
1
1

We know that adding 1 + 1 gives us the result 10 and a third 1 gives 11.

We put a 1 on the bottom row of column 4 and we carry the 1 over to the third row of column 3

Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8

0	0	0	1	1	1	0	1
1	0	0	1	1	0	1	1
		1	1	0	1	1	
			1	**1**	**0**	**0**	**0**

Step 7

Column 4 is now finished with. We now work on column 3.

Column 3 contains the numbers:

0
0
1

Here is the result: there is no carry in this case

Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
0	0	0	1	1	1	0	1
1	0	0	1	1	0	1	1
	0	1	1	0	1	1	
		1	**1**	**1**	**0**	**0**	**0**

Step 8

We now work on column 2:

0

0
0

Here is the result:

Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
0	0	0	1	1	1	0	1
1	0	0	1	1	0	1	1
0	0	1	0	0	1	1	
	0	**1**	**1**	**1**	**0**	**0**	**0**

Step 8

Finally, we work on column 1:

0
1
0

Here is the result:

Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
0	0	0	1	1	1	0	1
1	0	0	1	1	0	1	1
0	0	1	1	1	1	1	

1	**0**	**1**	**1**	**1**	**0**	**0**	**0**

After following all of those steps, the answer to adding the two binary numbers:

00011101 +
10011011

10111000

Question: What would have happened if Col 1 had the sum 1+1 in it? Where does the carry number go? See the binary overflow page that discusses this.

Binary numbers

10. Binary Shifts

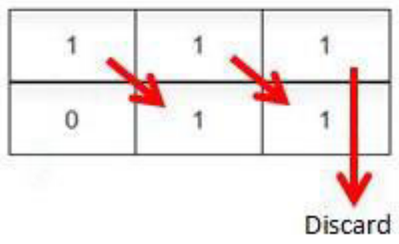
Binary shifts multiply or divide a binary number by 2

We've discussed how to add binary numbers together. The other arithmetic operations you may be asked to perform are to multiply or divide a binary number by 2. This is done using a **binary shift**, which is less complicated than it sounds.

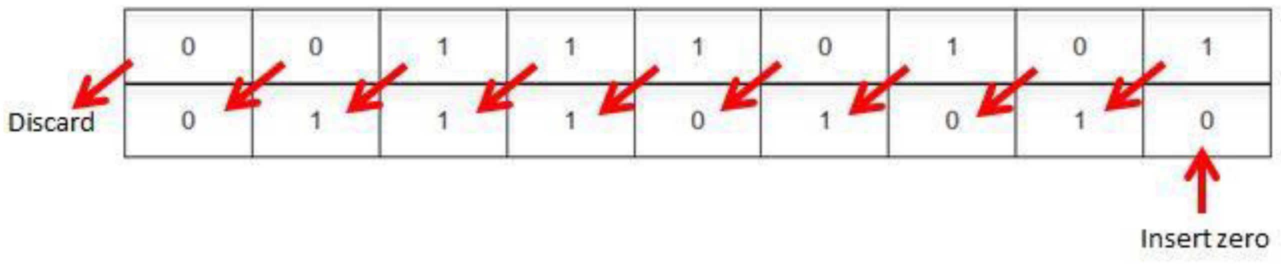
All it means is that every digit in the number is moved one place.

- To multiply, the digits are moved one place to the left. A "0" is inserted in the rightmost position.
- To divide, the digits are moved one place to the right, and the rightmost digit is discarded.

Binary shifts can't deal with fractions very well. If you are right-shifting "111" (denary 7), for example, it becomes "11" (denary 3).



And while humans don't mind adding another column to the left when multiplying, computers often have set sizes for numbers. A CPU that left-shifts a number will often drop the leftmost number, just like we drop the rightmost one when right-shifting.



This can cause issues of "binary overflow", which we will discuss shortly.

Binary numbers

11. Binary shifts - an example

The denary value of 001110101 = 117

With a leftwise shift every bit is moved up one and a zero is inserted to the rightmost bit, the original leftmost bit is discarded. The result is shown below

Left Shifted number

0	1	1	1	0	1	0	1	0

The new denary value of this is 011101010 = 234 which is twice the original 117

Shifting it again results in

1	1	1	0	1	0	1	0	0

In denary, this number is 111010100 = 468 which is four times the original

Now let's shift it once more, the answer should be 936

1	1	0	1	0	1	0	0	0

But 110101000 = 424 what has happened? The clue is that that 936 - 424 is 512 which of course is the 9th digit in a binary number. And as 9 bits can't be stored in an 8 bit register, an overflow has occured. This may or may not be an issue, depending on the purpose of the bit shifting

Binary numbers

12. Overflow

On an earlier page we added two binary numbers together and got a straightforward result:

Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
0	0	0	1	1	1	0	1
1	0	0	1	1	0	1	1
0	0	1	1	1	1	1	
1	**0**	**1**	**1**	**1**	**0**	**0**	**0**

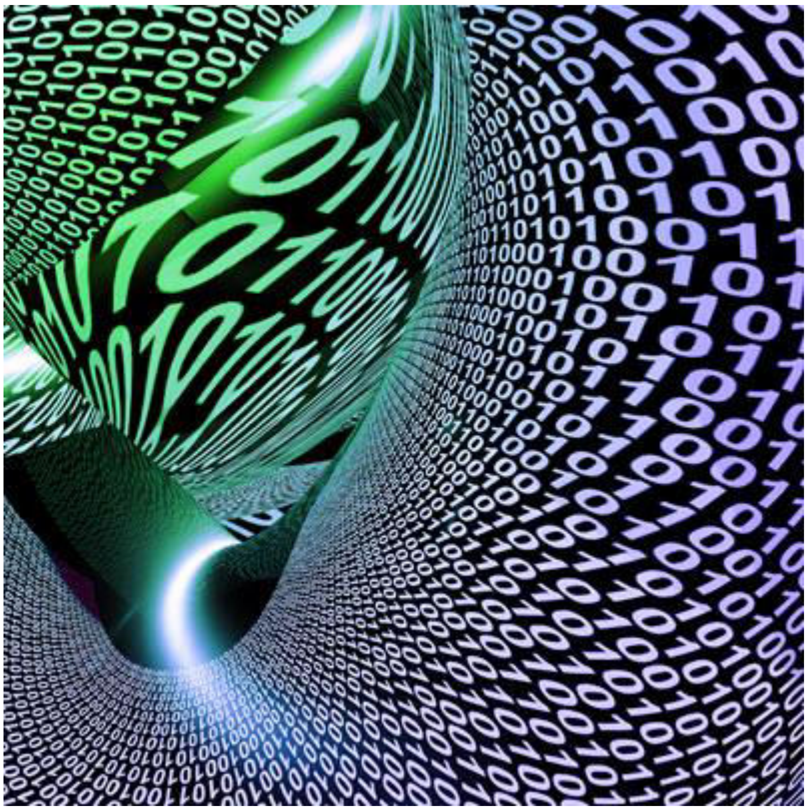
Now what would happen if the 8th digit (col 1) sum is 1 + 1? The carried 1 has nowhere to go and so is lost.

This problem is called 'overflow' and it causes the wrong answer.

Overflow is such a problem in a CPU that a special 'flag' inside the CPU is set when it happens.

A 'flag' is a single bit within a certain register and is used to mark an event happening.

The software carrying out the maths should read this flag to see if an overflow has occurred. If so, then it has to deal with the problem.



Binary numbers

13. Summary

- Number systems use a set of agreed symbols to mean a certain number - tally marks or otherwise
- Denary means a base 10 system, using the symbols 0 to 9
- Binary means a base 2 system, using the symbols 0 and 1
- With some simple rules it is perfectly possible to move from one number system to another
- Denary is for everyday maths, binary is for raw computer processing
- Bitwise shifts can be used to multiply or divide binary numbers quickly (if no overflow or underflow)

- Bitwise shifts can cause overflow which is indicated in a CPU by setting an overflow flag
- A 'flag' is a single bit within a certain register and is used to mark an event happening.

Hexadecimal numbers - conversion and purpose

1. Why Use Hexadecimal?

A computer, internally, can only handle binary numbers.

Such as 101001110 11011001010 110100101011 1010101 111001011 and so on.

However people find writing and reading binary numbers very awkward and error prone - we are just not built to handle long strings of 1s and 0s directly.

Therefore a more human-friendly system is used, namely the hexadecimal number system.

Denary numbers are base-10 because they use ten symbols, and binary is base-2 because it uses two symbols. Hexadecimal numbers are base-16 which uses sixteen symbols.

Hexadecimal is useful because it is much more readable to humans than binary but at the same time it still shares a lot of the qualities of binary. It is widely used in computing because it is a much shorter way of representing a byte of data (8 binary digits or bits).

If we were to represent a byte of data in binary, it would require 8 digits, e.g. 10100110.

However, that same byte of data could be represented in hexadecimal in just two digits e.g. "A6". This is much more compact and user friendly than a binary number.

Since a byte is 8 digits long it has a maximum value. In denary this is 255. The same number can be written as 11111111 in binary, or FF in hex.

2. Hexadecimal Symbols

The symbols used in hexadecimal are 0-9 and A-F. These correspond to the following numbers in denary:

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

When you go beyond 15, you run out of symbols. So, just like in denary and binary, you have to start again in the next number position.

Denary	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Hexadecimal	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F

Once we have gone past the first 16 numbers, a 1 is added in front of the next 16 numbers (similar to denary when we go from 0-9 and then move on to 10-19)

3. Hexadecimal Symbols cont.

The table on the previous page took us from denary 16-31. The next table is from 32-47. You can see that the corresponding hexadecimal numbers now have a 2 in front of them:

Denary	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
Hexadecimal	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F

See if you can complete the table for the next set:

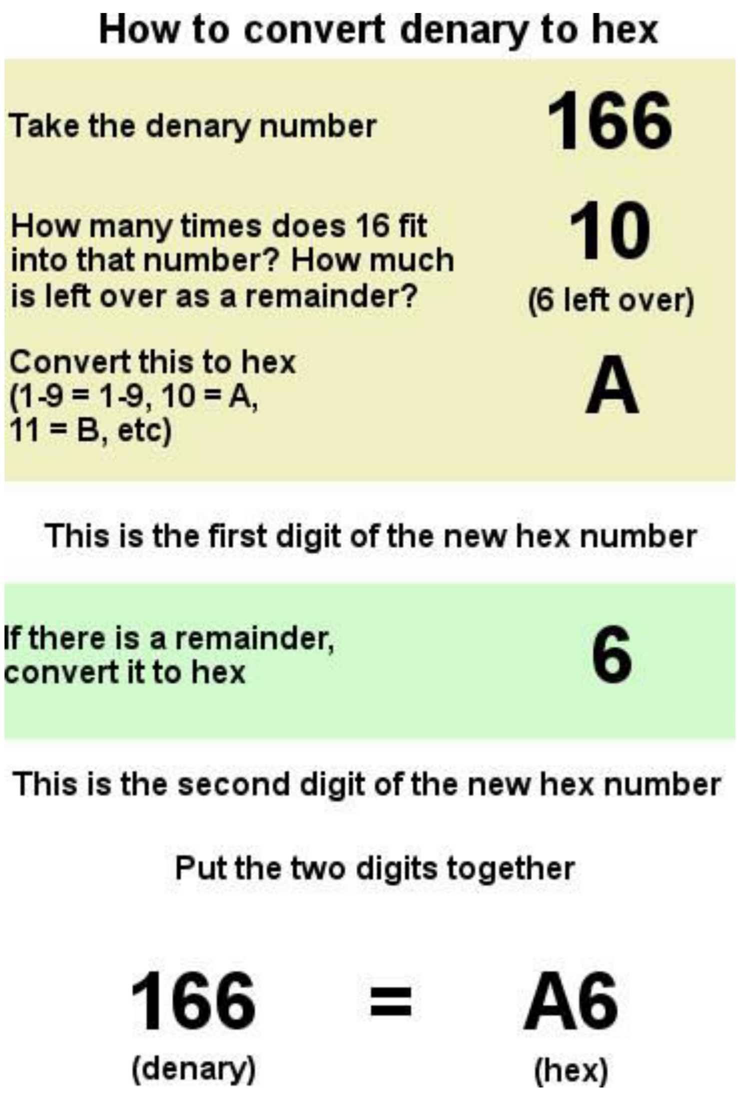
Decimal	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Hexadecimal																

Decimal	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Hexadecimal	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F

4. Denary to Hexadecimal Conversion

There is a quick algorithm to convert denary into hexadecimal that works on any sized denary number

Steps:



How to convert denary to hex

Take the denary number	166
How many times does 16 fit into that number? How much is left over as a remainder?	10 (6 left over)
Convert this to hex (1-9 = 1-9, 10 = A, 11 = B, etc)	A

This is the first digit of the new hex number

If there is a remainder, convert it to hex	6

This is the second digit of the new hex number

Put the two digits together

166 (denary) = A6 (hex)

Let's try another example. 106.

1. Divide the denary number by 16.

16 goes into 106 six times, with a remainder of 10.

2. Take the quotient value and look up the hex equivalent

The quotient is the number of times we could divide our original number by 16.

In our example, we divided it 6 times. 6 denary is 6 hex, so that's pretty simple.

We write 6 in the leftmost position

3. Take the remainder value and look up the hex equivalent.

You can use the hex tables on the previous two pages, or you can just remember that 1-9 are the same, while 10-15 are A-F.

Our example has a remainder of 10. "10" (denary) = "A" (hexadecimal)

This is the rightmost digit of our answer.
Our answer is **6A.**

Therefore, denary 106 = hexadecimal 6A

If you are working with a larger number you will need to repeat the steps until the quotient can no longer be divided by 16

5. Denary to Hexadecimal Conversion

Example 1: Convert 137 denary into hexadecimal

Comments	**Calculation**	**Quotient**	**Amount remaining**	**Remainder in Hex**
16 goes into 137, 8 times with a remainder of 9.	137 / 16	8	9	**9**
16 goes into 8, 0 times. So we have 8 remaining. The remainder in Hex is 8	8 / 16	0	8	**8**

Thus we write down the answer from the final column, in reverse order.
Denary 137 is Hexadecimal **89**

Example 2: Convert 75 denary into hexadecimal

Comments	**Calculation**	**Quotient**	**Amount remaining**	**Remainder in Hex**

Comments	Calculation	Quotient	Amount remaining	Remainder in Hex
16 goes into 75, 4 times with a remainder of 11.	75 / 16	4	11	**B**
16 goes into 4, 0 times. So we have 4 remaining. The remainder in Hex is 4	4/ 16	0	4	**4**

Thus we write down the answer from the final column, in reverse order.
Denary 75 is Hexadecimal **4B**

Example 3: Convert 257 decimal into hex

We chose this to show you how to deal with a remainder of zero as you work through the method

Comments	Calculation	Quotient	Remainder in Hex
16 goes into 257, 16 times with a remainder of 1.	257 / 16	16	**1**
16 goes into 16, 1 times with a remainder of 0	16 / 16	1	**0**

Comments	**Calculation**	**Quotient**	**Remainder in Hex**
16 goes into 1, 0 times with a remainder of 1	1 / 16	0	**1**

Thus the answer is the remainder rows going backwards up the table: **101** Hexadecimal

Example 4: Convert 1050 decimal into hex.

Comments	**Calculation**	**Quotient**	**Remainder in Hex**
16 goes into 1050 65 times with a remainder of 10. The 10 is converted into hex A	1050 / 16	65	**A**
16 goes into 65, 4 times with a remainder of 1	65 / 16	4	**1**
16 goes into 4, 0 times with a remainder of 4	4 / 16	0	**4**

Thus the answer is the remainder rows going backwards up the table : **41A**

6. Hexadecimal to Denary Conversion

Now let's try converting hexadecimal numbers back into denary.

Just like you have learned with denary and binary numbers, the position of a digit determines its value. For example, denary 23 means that it is actually: (2 x 10) + (3 x 1)

Hexadecimal works in the same way. The position of its digits determines the value. The only difference is that we use powers of 16 rather than powers of 10:

Hexadecimal Digit position	3	2	1	0
Multiplier	163163	162162	161161	160160
Denary value	4096	256	16	1

For two-digit hexadecimal numbers you multiply the value of the first digit by 16, and the value of the second digit by 1.

For instance, 23 Hexadecimal is actually:

(2 x 16) + (3 x 1) = 35 denary

While F2 Hex = (15 x 16) + (2 x 1) = 242 denary

7. Position of numbers

Here is the denary table from the previous page. We have put our number 23 into the bottom row of the table.

Denary Digit position	3	2	1	0
Multiplier	103103	102102	101101	100100
Denary value	1000	100	10	1
Our number			**2**	**3**

To work out the value of any number, hex, denary or binary, the basic rule is that you always work from right to left.

So looking at the table above, you can see that the 3 is in position 0 and the 2 is in position 1.

8. Hexadecimal to Denary Conversion

Example 1: convert 21 Hexadecimal to its denary equivalent

Hexadecimal Digit position	3	2	1	0
Multiplier	163163	162162	161161	160160
Value of multiplier	4096	256	16	1
Hexadecimal number to be converted			**2**	**1**
Calculation			2 x 16^1	1 x 16^0
Result			32	1
Completed conversion (denary number)	**32 + 1 = 33**			

Starting from the right hand side, take the 1 (from 21) and multiply it by 1 (16^0) =1

Move one position to the left - which is our number 2 (from 21)
Multiply 2 by 16 (16^1)= 32

Check there are no further digits to the left. If we have done the final calculation then add up all of the results:

1 + 32 = 33

So 21 Hexadecimal is 33 in denary

Example 2: work out 457 Hexadecimal as denary

Start at the right hand side

7 x 1 = 7

5 x 16 = 80

4 x 256 = 1024

Answer is 7 + 80 + 1024 = 1111 denary
So 457 in Hexadecimal is 1111 in denary

Example 3: work out 586 Hexadecimal as denary

Start at the right hand side

6 x 1 = 6

8 x 16 = 128

5 x 256 = 1280

Answer is 6 + 128 + 1280 = 1414 denary
So 586 in Hexadecimal is 1414 in denary

9. Binary to Hexadecimal Conversion

It has been mentioned earlier, hexadecimal is mainly used in computing as a very convenient way of expressing a binary number in a compact form. It is quite straightforward to move between hexadecimal and binary numbers, although we have to do it in two separate stages.

The first step is to break the binary number into groups of four digits, starting from the right. For example, 11010001 would be broken into 1101 and 0001.
The term for a group of four binary digits is 'nibble' (we have a section describing this here)

If the leftmost chunk is less than four digits long, add zeros to the front of it until it is four digits long. Thus 111000 becomes 0111, 1000
The next step is to break each of these four-digit numbers into denary. We learned how to do this **in an earlier page**.

Since the binary nibbles are four digits long, each nibble will be between 0 and 15 when converted into denary.

Denary	7	6	5	4	3	2	1	0
Binary equivalent	0111	0110	0101	0100	0011	0010	0001	0000

Denary	15	14	13	12	11	10	9	8
Binary equivalent	1111	1110	1101	1100	1011	1010	1001	1000

So 11010001 becomes 1101 and 0001. When converted into denary, these become 13 and 1.

Once we have the two denary numbers for our original binary version we have finished step 1 and can move onto step 2.

Step 2 is where we change the two denary values into hexadecimal.

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

Step 1 gave us the denary values of 13 and 1.

Using the table above we can see that 13 (denary) is D in hexadecimal

And 1 in denary is 1 in hexadecimal

The final step is to join these two hexadecimal values together so our final result is D1

So we have converted binary 111000 into its hexadecimal equivalent of D1

To summarise the process:

- Break the binary number into four digit chunks. If the leftmost chunk is less than four digits long, add zeros to the front of it.
- Convert each nibble to denary
- Convert each denary number to hexadecimal
- Recombine the nibbles

10. Binary to Hexadecimal Examples

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

Example: What is 11011110 as a hex number?

Step 1.

Split the binary number into four digit chunks, like this

1101 1110

Step 2.

Now convert each chunk into denary

13........14

Step 3.

Then finally convert these directly to hexadecimal

D ...... E

Answer: 11011110 (binary) = DE (hexadecimal)

DE is much shorter and easier to read than 11011110, so you can see why this kind of conversion would be useful to programmers who have to deal with binary.

Let's look at some other examples.

Example: Convert 10011111 to hexadecimal

10011111 breaks down into 1001 and 1111

1001 (binary) = 9 (denary) = 9 (hexadecimal)

1111 (binary) = 15 (denary) = F (hexadecimal)

So 10011111 = 9F

Example: Convert 101100 to hexadecimal

101100 = 0010 and 1100

0010 = 2 = 2

1100 = 12 = C

101100 = 2C

Example: Convert 1110 to hexadecimal

1110 is already a single four-digit chunk

1110 = 14 = E

1110 = E

11. Hexadecimal to binary conversion

To convert hexadecimal to binary, you simply reverse the process we talked about on the last page:

- Convert each digit into denary
- Convert each denary value into binary
- Recombine the binary chunks into a single number

Hexadecimal	7	6	5	4	3	2	1	0
Denary	7	6	5	4	3	2	1	0
Binary	0111	0110	0101	0100	0011	0010	0001	0000

Hexadecimal	F	E	D	C	B	A	9	8
Denary	15	14	13	12	11	10	9	8
Binary	1111	1110	1101	1100	1011	1010	1001	1000

Example: what is E3 hexadecimal as a binary number?

E3 = E and 3

E (hexadecimal) = 14 (denary) = 1110 (binary)

3 (hexadecimal) = 3 (denary) = 0011 (binary)

1110 and 0011 becomes 11100011

Example: what is D1F Hexadecimal as a binary number?

D1F = D and 1 and F

D (hex) = 13 (dec) = 1101 (bin)

1 (hex) = 1 (dec) = 0001 (bin)

F (hex) = 15 (dec) = 1111 (bin)

1101 and 0001 and 1111 becomes 110100011111

12. Summary

- Number systems use a set of agreed symbols to mean a certain number - tally marks or otherwise
- The three number systems for this syllabus are denary, binary and hexadecimal
- Denary means a base 10 system 0 to 9
- Binary means a base 2 system 1 or 0
- Hexadecimal a base 16 system with A,B,C,D,E,F for values above 9
- With some simple rules it is perfectly possible to move from one number system to another
- Three number systems (denary, binary, hexadecimal) are used because it is convenient to do so
- Denary is for everyday maths, binary is for raw computer processing and hexadecimal is for human convenience in dealing with binary values such as 0101010110

Characters

1. Introduction

Many of the main functions of a computer require it to be able to handle text. Of course, to a computer, all data is actually sequences of binary numbers.

Therefore text needs to be converted into binary numbers in order for the computer to be able to process it.

This section will describe how text or characters are represented in binary form.

Topics covered include ASCII, characters sets and Unicode.

Characters

2. Standard ASCII

You already know that computers can only process binary numbers i.e. "0" and "1".

While this works perfectly well for the computer, imagine how hard it would be to write your essays or emails using only 0s and 1s. How would you know if you had made a spelling mistake? How could you proofread your work?

To solve this, a system had to be developed that would allow computer operators to use the full set of alphabetic characters, both upper and lower case, numbers and symbols.

The system uses binary codes to represent each character, number and symbol in the chosen language.

The most commonly used system is called the 7 bit standard ASCII code short for (The **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange)

7 bits gives you 2727 possible values. This is 2 x 2 x 2 x 2 x 2 x 2 x 2, or 128. So the ASCII character set goes from 0-127. The character set is broken down like this :-

ASCII	
Lower case letters of the alphabet	26
Upper case letters of the alphabet	26

Number symbols	10
Punctuation marks and white space	33
Control codes such as carriage return and line feed	32
TOTAL SET	**127**

Characters

3. ASCII continued

In ASCII every character is represented by a binary number, e.g:

0	**1**	**0**	**0**	**0**	**0**	**0**	**1**

.

The 8 bit ASCII code below represents the upper case letter **A**:

0	**1**	**0**	**0**	**0**	**0**	**0**	**1**

The 8 bit ASCII code below represents the lower case letter **a**:

0	**1**	**1**	**0**	**0**	**0**	**0**	**1**

If you wanted to represent the word **JOHN** in ASCII, it would look like this:

01001010 01001111 01001000 01001110

The word **JOHN** would take 4 bytes of memory to store (1 byte per character).

Since computer architecture is built around the byte (which has 8 bits) the 7-bit ASCII table is a little inconvenient.

So ASCII characters are padded out to 8 bits, with the first bit always set to 0. This makes memory management a little easier.

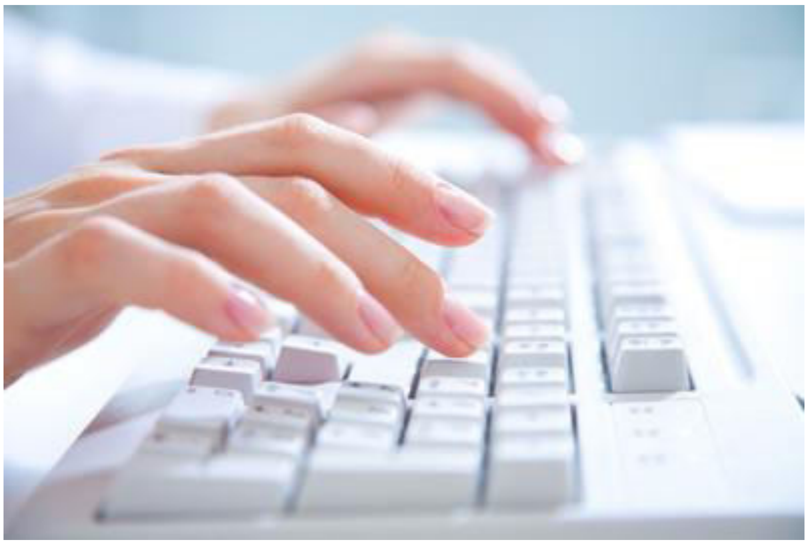
There is, however, an extension to ASCII, called "Extended ASCII", that makes use of this spare digit to fit in another 128 characters commonly used in other languages.

Characters

4. Character Sets

People enter data into a computer by means of an input device such as a keyboard.

Imagine a prankster has levered off all the plastic key-tops on a keyboard and then put them back in a different arrangement, so all of the letters on the keys were switched around. What would happen if you plugged it in?



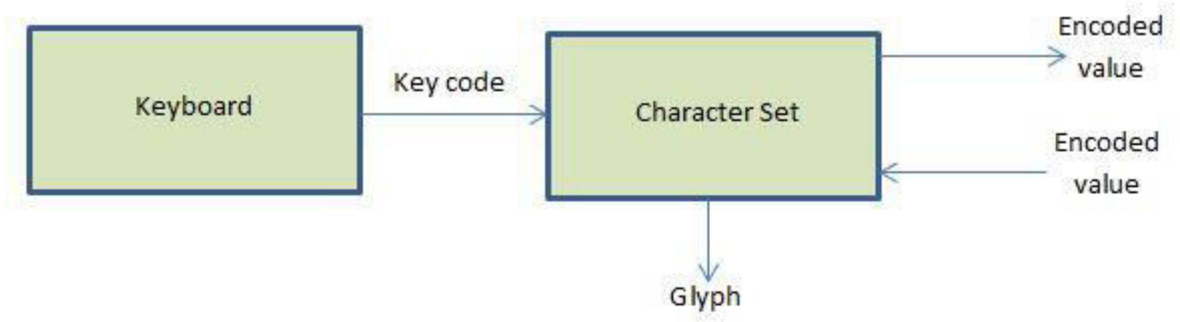
The answer is that the computer would continue to treat that keyboard as if the keys were still arranged in the proper way. If you pressed the key in the bottom left of the number pad, it would enter "0" even if you'd changed the key to read "Q".

The reason for this is that the computer doesn't know what's printed on the keys. It can only tell them apart by the signals that are sent when a key is pushed. These signals tell the computer the *position* of the key on the keyboard, not what is written on it.

Characters

5. Character Sets

When you press a key on your keyboard the '*character set*' translates the signal produced from the key's position into its proper binary code value. And going the other way, the character set maps code values back into their correct symbol (glyph) for display.



A **character set** is the mapping between characters and their identifying code values.

If you use an European language keyboard, for example, then the "Latin Alphabet No 1" character set may be installed on the computer to translate the keys into the right language symbols for you (there are other character sets that would also do the same job).

A completely different language such as Japanese would use a different character set and of course the middle-row third key would have a different symbol on it.

A **character set** is NOT a font.

A character set is not a font or a typeface. The character set tells the computer what a character is - whether it's a 3 or a T. The font or typeface determines how those characters look, but a 3 is a 3 whether it's in bold or italic, Times New Roman or Arial.

So far, we have discussed 1 byte character sets such as ASCII and extended ASCII. But these can only describe 256 symbols. Some languages have far more than this. For instance some far-east languages have more than 12,000 characters!

So the logical way of handling the problem is to use a character set that uses more than 1 byte. The next page describes multi-byte methods.

Characters

6. Unicode

A 1 byte character scheme can only represent 256 symbols. This is fine for many individual languages which is why ASCII is so popular.

But a 2 byte scheme can represent 65,000+ characters (two to the power of 16). This is more than enough to hold every currently used language, in the world, in one place.

Imagine that you own a company that sells computers all over the world. Naturally every customer will want to type in their own language. What to do? You want a system that can handle every possible written language - when the computer gets turned on for the first time they simply choose the language that the operating system needs to handle.

This is why 2 byte character sets were developed.

A very popular 2 byte (16 bit) encoding standard is called '**Unicode**'. Unicode can handle any language and it does so by the user selecting a specific '**code page**' which is one portion of the total unicode space. Each code page represents the chosen language.

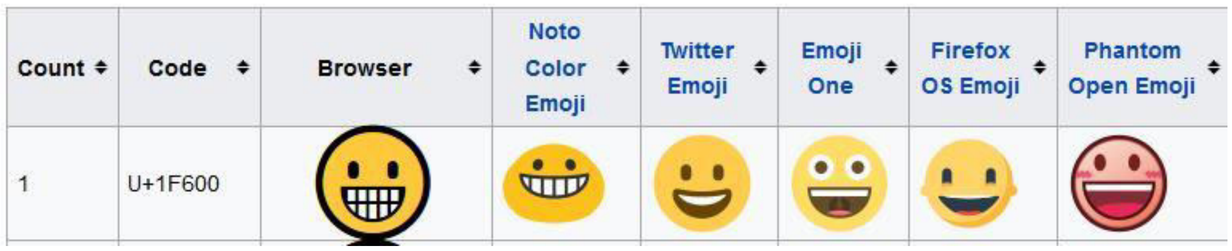
For example code page 1253 within Unicode represents the Greek language.

If the person in Greece has a greek keyboard and the greek codepage is selected within the operating system then the correct characters appear on the screen when they press a certain key on the keyboard.

Extended Unicode

A later version of Unicode uses even more bits - 21 bits - in order to include ancient languages such as Egyptian Hieroglyphics and even emojis.

For example did you know that the Unicode code for a smiley face is U+1F600



Count	Code	Browser	Noto Color Emoji	Twitter Emoji	Emoji One	Firefox OS Emoji	Phantom Open Emoji
1	U+1F600	😀	😀	😀	😀	😀	😀

courtesy of Wikimedia Commons

Characters

7. Conclusion

To summarise, the number of bits defines the limit of how many characters can be represented within any scheme.

The 7 bit ASCII scheme allow for 128 symbols (2 to the power of 7 = 128) which is fine for many Latin languages such as English.

The 8 bit extended ASCII scheme allows for 256 symbols (2 to the power of 8 = 256) although unfortunately there are a number of extended ascii schemes which could cause confusion.

Moving on to multi-byte schemes such as 16 bit Unicode, 16 bits allows for 65,535 symbols to be represented. This allows for all current written languages to be represented under one scheme.

A further extension of Unicode expanded the scheme to 21 bits, so offering over 1 million symbols, (1,114,112 items in the range Hex 0 to Hex 10FFFF).

This is enough space for even dead languages such as Egyptian Hieroglyphics to be represented. It even handles emojis

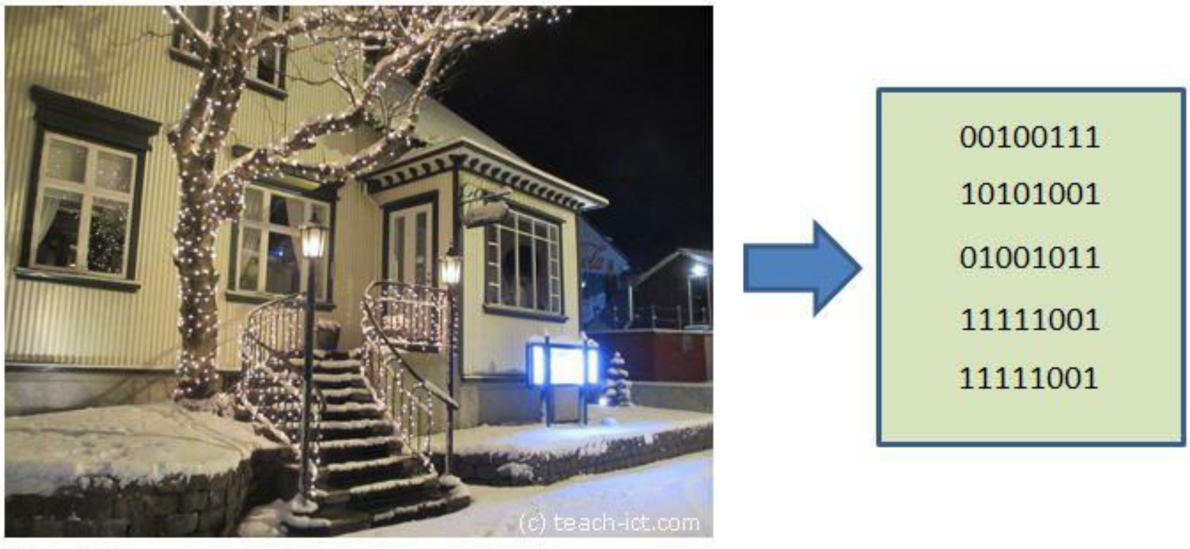
1. Introduction

We see the world in smooth shades of colour, each blending into the next. Our vision is *analogue*. Computers, though, can only interpret *digital* signals, where every value is either 1 or 0.

So how do we convert between the two?

There needs to be a way of representing an image as a set of binary numbers, so that it can be stored in a file or displayed on a screen.

In this section we will look in detail at the process.



© teach-ict.com

We will also discuss what is meant by terms such as colour depth and resolution, and how these relate to the file size.

Finally, we will be discussing metadata - additional data added to the image on top of instructions on how to render it.

2. Input devices

The two most popular types of input devices for creating an image in binary form are the flatbed scanner and the digital camera (including the one built into your smart phone)

Flatbed scanner.

A physical photograph can be copied into a binary file by using a flatbed scanner. The photo is placed face down on the glass, then a scanning head slowly moves across it. The head moves into its first position, and takes a reading of all the colours it senses in a narrow strip. Then it moves slightly and takes another set of readings. This continues until the head has scanned the whole photograph.

The binary data is stored as a file. This file can then be imported into an image editing application for processing.

Digital Camera

A digital camera has an image sensor that creates binary files directly.

These files are then stored in the camera's memory card or transferred by USB cable and Wi-Fi into a computer.

3. Pixels

As we mentioned earlier, computers can only handle 1s and 0s. They need absolute values, rather than a mishmash of different colours.

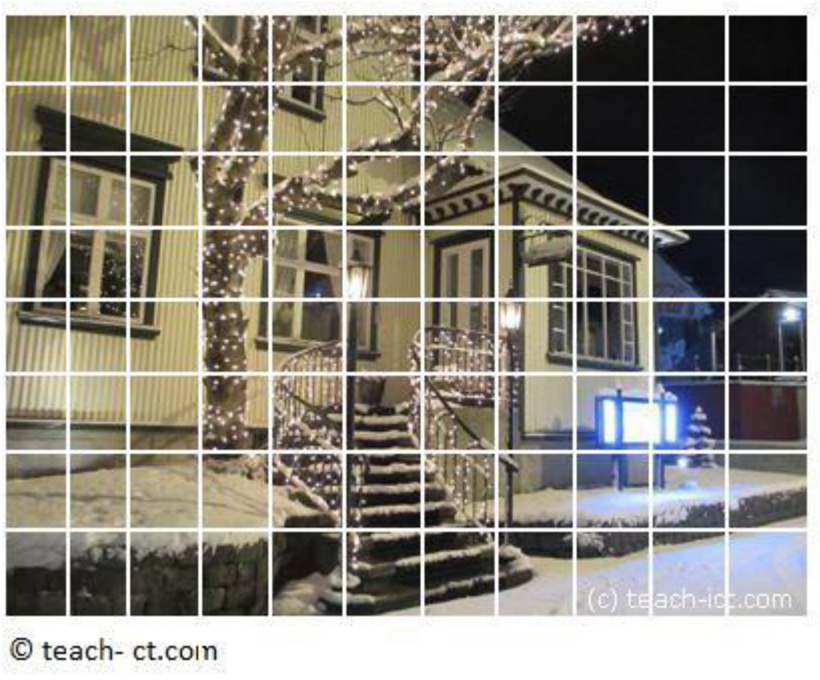
These values also need to be arranged in an organised manner, so that the computer can tell the position of every one.

The simplest way of doing this is with a grid, with a single value stored in one square of the grid.

So how do you do this with a photograph, which has millions of different colours?

The first step is mostly the same: make a grid.

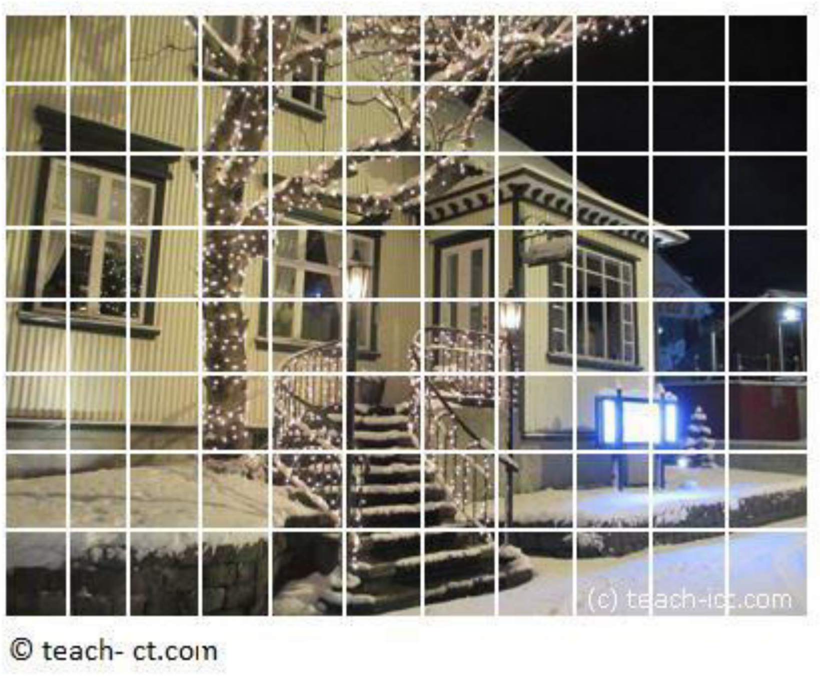
We've overlaid the image below with a grid of lines. This divides it up into smaller sections.



The picture has effectively been divided into 11 horizontal elements (columns) and 8 elements vertically (rows). This is what happens when an image is digitised - it is divided up into many individual elements.

4. Pixels cont.

However, we haven't broken the image down far enough. Each block still contains multiple colours. We need to increase the number of rows and columns to allow only a *single* colour to be present within each box.



Now you are getting close to a true digital picture.

With only one colour per box, you can represent each using just a single binary number.

The sections of a grid like this are called 'pixels' which is short for 'picture element'. And by increasing the number of pixels and making them very small, the image looks completely smooth.

Bitmap Image

A bitmap image file represents an image as a series of pixels. If you were to zoom in or expand a bitmap image enough you would be able to see each pixel.

A popular bitmap image file format is BMP. It has data on every single pixel and so bmp files can be very large compared to other file formats.

A BMP file is 'uncompressed' as it has the complete data set for the image.

Other file formats such as JPG, compress the data and so some non-vital pixel information is discarded. This is called 'lossy compression', so although some quality is lost, the file sizes are much smaller.

Megapixels

A megapixel is a unit of resolution equal to one million pixels.

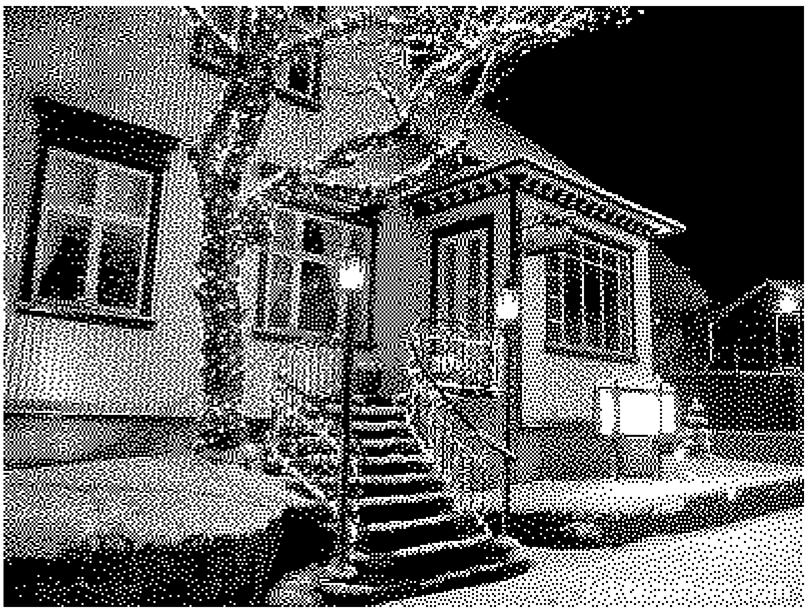
You will see digital camera makers promote the quality of their product by quoting how many 'megapixels' their camera sensor has. For example a 20 megapixel camera will be able to take more refined pictures than a 5 megapixel one, simply because it is using more pixels within each image.

5. Greyscale

The previous page described how a binary number can represent a single, specific colour.

Each colour needs to have a unique value, in order to tell it apart from other colours. So the greater the range of colours you want to represent, the greater the range of binary numbers you need.

At one extreme is a system with only two allowed numbers: '1' and '0'. With just two numbers you can only represent two colours - such as black and white.



The photo above was converted to a bitmap image with three colours - black, one shade of grey and white. All the fine details have been lost as three colour values are not enough to make an accurate representation.

You can make increasingly more detailed images by allowing more and more shades of grey. Such images are called "greyscale".

But no matter how many shades of grey you allow, it doesn't help you show colour.

6. Colours

We saw on the previous page that you can make images more and more detailed by assigning more binary numbers to represent each shade. But this will only leave you with a very detailed greyscale image.

To make the jump to coloured images, it's important to know something about the way that humans see the world. The cells in our eye can only pick up three possible colours: red, green and blue, and pass on to our brains how intense each of those colours is at any given point. Every colour that we see is just our brains adding up the values of red, green, and blue.

So colours can be represented by knowing three values: how much red to add, how much green to add, and how much blue to add. Computers can handle that! We give each pixel 1 byte of memory for each of those three values.

A byte has 256 possible values, so the Red byte can represent any of 256 different shades of red. The same with the green byte and the same with the blue byte.

With a 3 byte number system there are 256 x 256 x 256 = 16.8 million possible colours. The 3 bytes can be stored separately or they could be combined to form a single 3-byte (24 bit) number.

This image has a blend of colours but they can all be broken down into the three primary colours for each pixel in the image. For example pixel no 1. in this image (top left) is a kind of dark beige and can be represented by Red = 169, Green = 89, Blue = 65

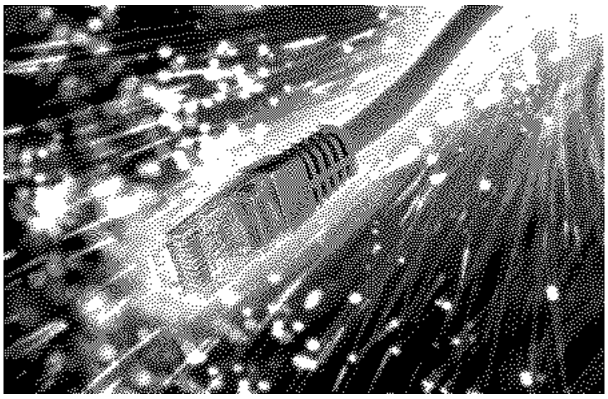
7. Colour depth

On the previous page we covered how you can have up to 16 million colours per pixel. Whilst that gives you a lot of flexibility, it can result in very large image files.

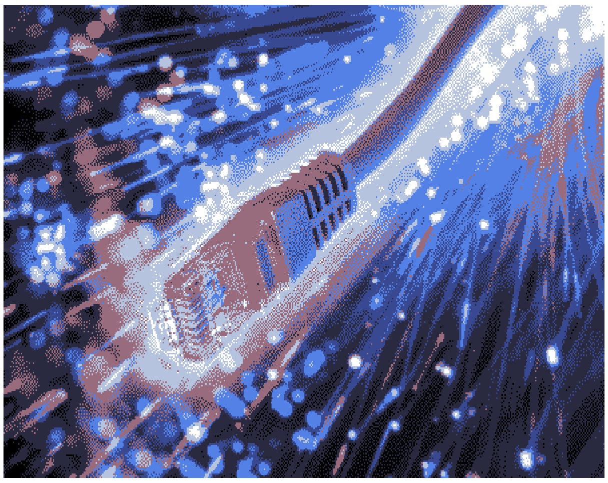
You may not be aware, but you don't have to always use 3 bytes per pixel. The downside of this is that you will limit the range of colours that each pixel can show.

The number of bits per pixel used to represent a colour is called **colour depth**.

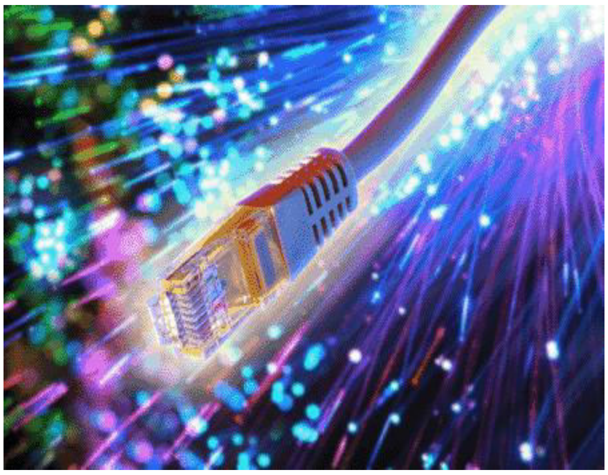
Have a look at these images as the colour depth is increased (RGB = Red, Green, Blue)



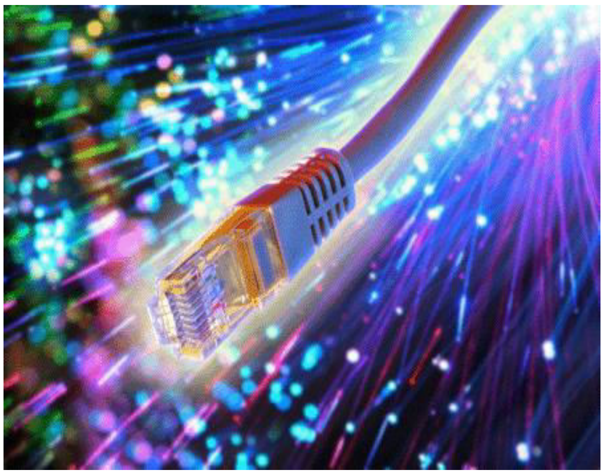
2 colours: There are only two colours in this image. It has a colour depth of **1 bit** i.e. The first colour is a single combination of RGB and the second colour is another combination of RGB.



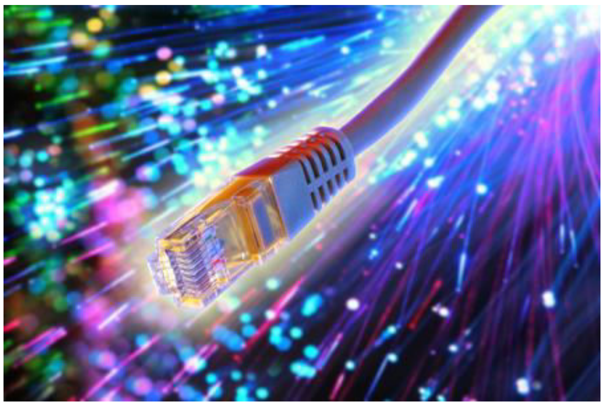
8 colours: Now this image is using 8 colours so it is a great improvement on the previous image in terms of accuracy. It has a **colour depth of 3 bits.**



64 colours. This has a colour depth of **6 bits**.



255 colours. It is has a colour depth of **8 bits**. This was the quality of older VGA colour monitors



Full colour. This has a colour depth of **24 bits**. And can represent 16 million colours. This is the quality of modern SVGA colour monitors.

As you can see, the greater the colour depth, the better the colour rendering. Of course there is a limit because the output device must be able to render the colour in the first place.

For example, printers have only a certain number of ink cartridges and this means that they can only handle colours that their blended inks can produce.

The colour depth of the computer's video card determines how many colours can be rendered on screen.

File size

It should be apparent that the greater the colour depth, the larger the image file is going to be because it is storing more bits per pixel.

8. Resolution

Image resolution is the number of pixels per inch. The higher the number of pixels (dpi) the greater the resolution.

The higher the resolution the more refined the image i.e. a greater amount of detail can be seen. The lower the resolution, the lower quality the image.

For example, in terms of video quality

480p - this is 480 pixels vertical, which is a basic You Tube quality upload. Note: it is good enough for non-blockbuster viewing such as personal interviews or vblogs.

720p - this is normal quality streaming video

1080p - this is the HD or 'Hi-Def' quality of video which just means 1,080 lines of detail per image.

However, the higher the resolution, the more information needs to be stored about the image and so the larger the file becomes.

Let's compare some images to demonstrate.

Below is a resolution of 72 dpi image (as seen in the screenshot below) - it is 131.4 Kb in size.

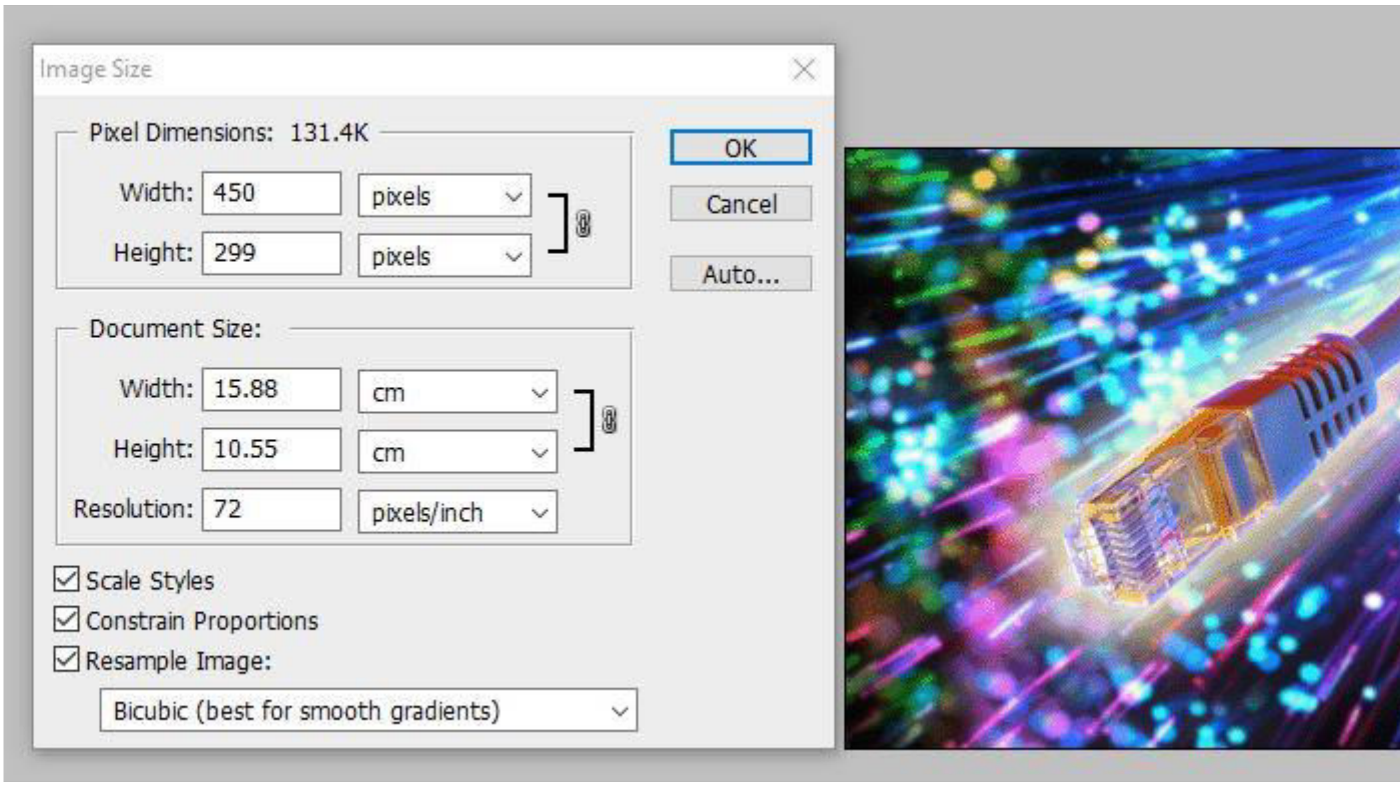


Image Size

Pixel Dimensions: 131.4K

Width: 450 pixels

Height: 299 pixels

Document Size:

Width: 15.88 cm

Height: 10.55 cm

Resolution: 72 pixels/inch

☑ Scale Styles

☑ Constrain Proportions

☑ Resample Image:

Bicubic (best for smooth gradients)

Now, let's alter the image to have a resolution of 300 pixels per inch

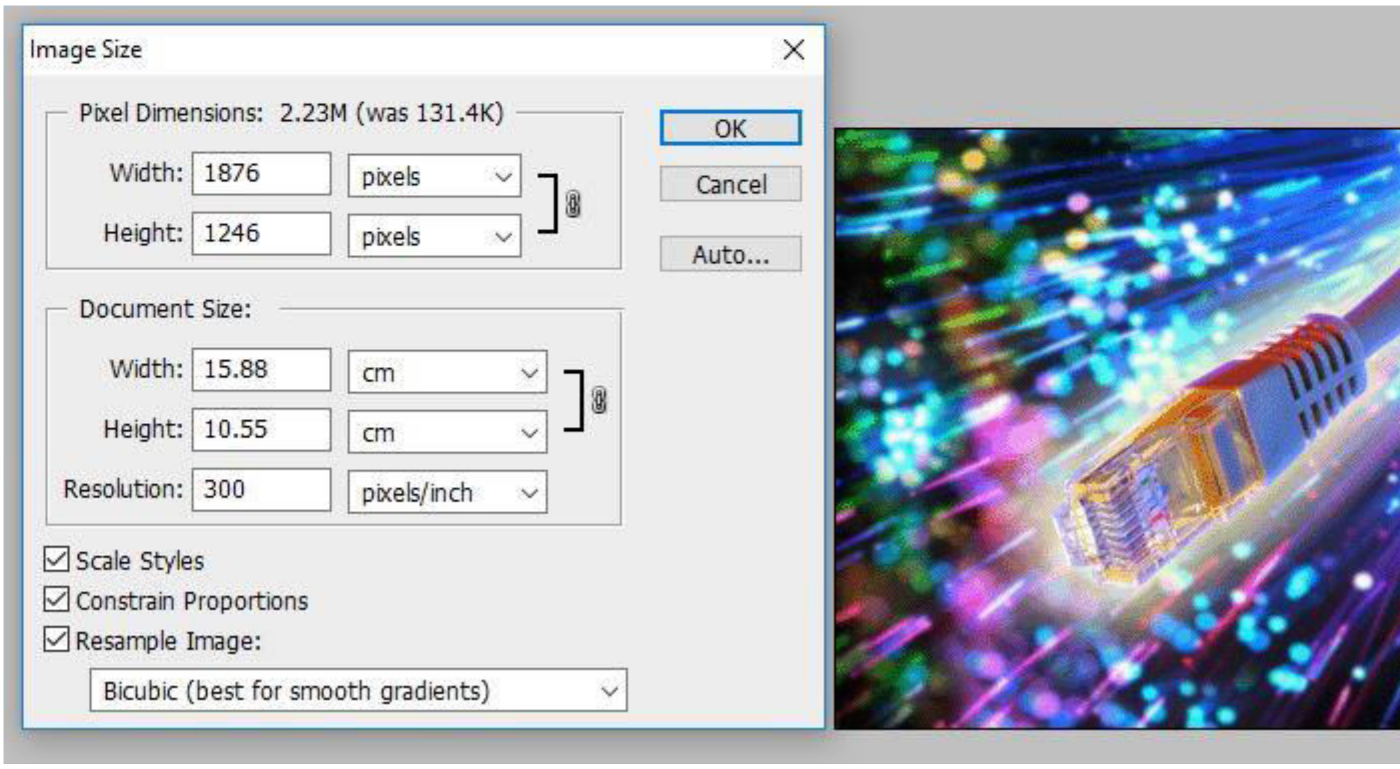


Image Size

Pixel Dimensions: 2.23M (was 131.4K)

Width: 1876 pixels

Height: 1246 pixels

Document Size:

Width: 15.88 cm

Height: 10.55 cm

Resolution: 300 pixels/inch

☑ Scale Styles

☑ Constrain Proportions

☑ Resample Image:

Bicubic (best for smooth gradients)

Notice that the file size (quoted after "Pixel Dimensions" at the top left of the image) has jumped to 2.23 Megabytes, because more data is being stored. Yet it looks exactly the same as the 72dpi one from before. This is because you are using the computer screen to view both - proving the point that 72dpi is good enough for screen view.

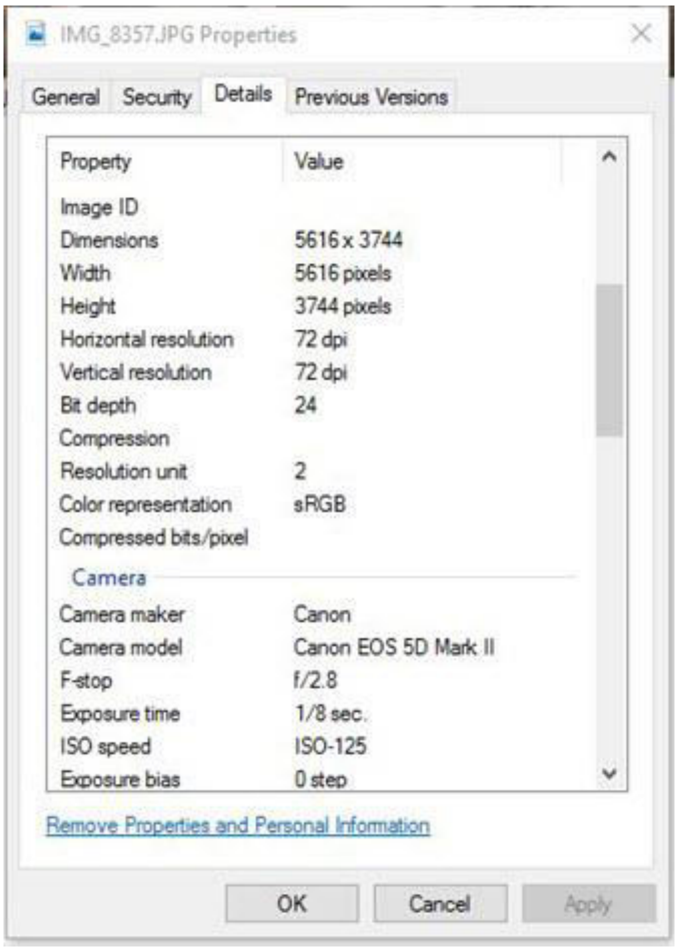
Only if you printed the image on a high-quality printer *would* you see the difference between the two images.

9. Metadata

Digital images are stored as data files. Metadata is the information that is stored about each image.

Metadata can include details like

- Width in pixels
- Height in pixels
- Horizontal resolution in dpi
- Vertical resolution in dpi
- Colour (bit) depth
- Dimensions


IMG_8357.JPG Properties

General | Security | Details | Previous Versions

Property	Value
Image ID	
Dimensions	5616 x 3744
Width	5616 pixels
Height	3744 pixels
Horizontal resolution	72 dpi
Vertical resolution	72 dpi
Bit depth	24
Compression	
Resolution unit	2
Color representation	sRGB
Compressed bits/pixel	
Camera	
Camera maker	Canon
Camera model	Canon EOS 5D Mark II
F-stop	f/2.8
Exposure time	1/8 sec.
ISO speed	ISO-125
Exposure bias	0 step

Some metadata such as that shown above, help an application to render the image properly.

Different image file formats hold their own metadata. For example, high quality digital cameras can place metadata into the image file that describes the name of the camera, the aperture and speed settings and even the GPS coordinates where the picture was taken.

In Windows, if you right-click on an image file a pop-up box appears, now select details tab and it shows all the metadata contained in the image file. This one happens to contain camera settings as well as basic image information.

10. Summary

- A bitmap image is made up of pixels

- A pixel is the smallest element of an image
- The colour of a pixel can be stored as a binary number
- Any colour can be produced from a blend of the three primary colours Red, Green and Blue
- A byte allows 255 shades of a primary colour to be stored
- A popular colour system is to have a byte for Red, a byte for Green and a byte for Blue
- The 3 byte combination can describe 16 million colours
- Common input devices to capture images are the flatbed scanner and digital camera
- BMP is an uncompressed file format for storing images
- JPG is a lossy compression file format that allows for very compact file sizes
- Colour depth is the number of bits used to store the colour of a pixel
- The higher the colour depth, the larger the image file
- Resolution is the ability of an image to resolve two theoretical dots lying next to each other
- Resolution is measured in dot per inch
- The higher the resolution, the larger the file size
- Metadata is extra information stored in the image file to help render it properly
- Metadata includes pixel width and height, along with resolution and colour depth

Sound

1. Introduction

A very common requirement of computers is to store and play back recorded sound, which of course includes music.

A 'real' sound is analogue in nature, you listen to sound with analogue ears where tones and rhythms blend together smoothly.

But computers are digital, they can only deal with binary numbers.

Therefore if sound is to be handled by a computer, the analogue sound from any source has to be converted into a set of binary numbers i.e. you 'digitise' the sound with some input device.

This is called 'sampling' the sound.

This section will discuss sampling and related topics in more detail.

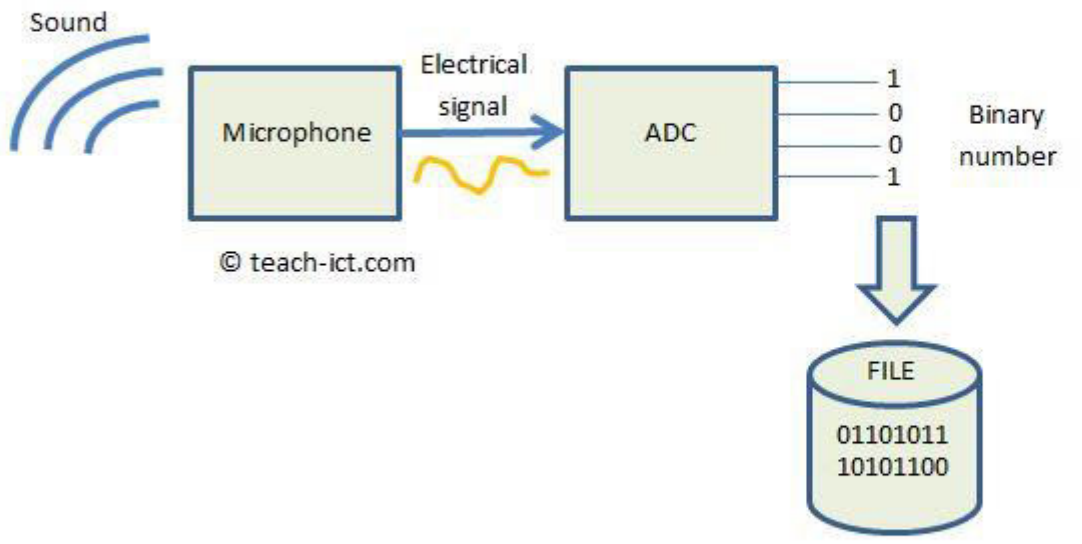
Sound

2. Recording and Replay

There are a number of devices that are involved in capturing analogue sound and converting it in to a digital file for use by a computer:

Microphone - Recording sound

A microphone is an input device for sensing incoming sound. It converts sound into an electrical signal. This signal is then passed on to the ADC (Analogue-to-Digital converter) in the sound card.



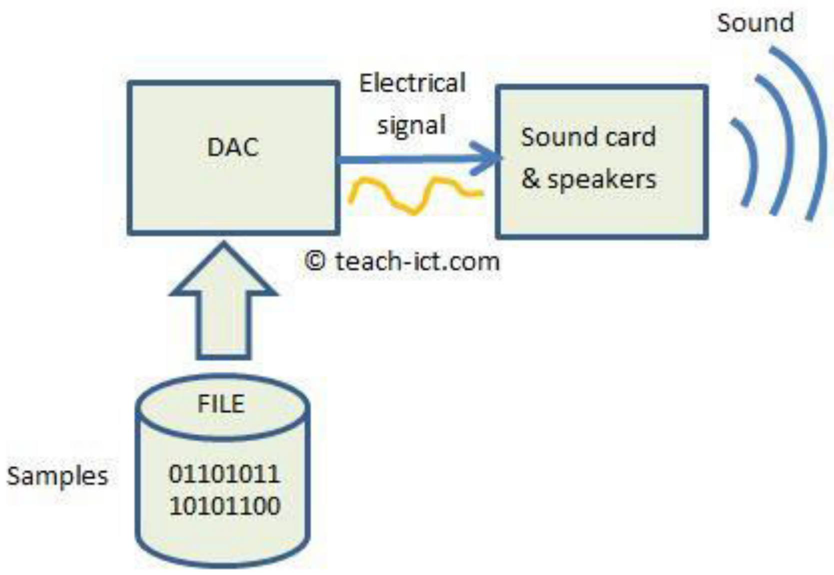
Sound card - Converting sound

A sound card has two jobs. It has to be able to convert an analogue signal into a digital one for storage. It also has to be able to convert stored digital signals into an electrical signal to send to the speakers.

The first job is performed by the Analogue-to-Digital Converter (ADC). The second is performed by the Digital-to-Analogue Converter (DAC). We will discuss how this happens in later pages.

Speakers - Replaying sound

To replay a sound, the file is opened and the binary numbers are fed through the DAC (digital-to-analogue converter) which is then amplified in some way (internal or external amplifier) to produce enough power for the speakers or ear-phones you are using.



Sound

3. What is sound

From your physics or science class you probably remember that sound is actually the subtle movement of air (vibration) causing your eardrum to move in sympathy.

Your brain senses this eardrum movement as 'sound'.

Sound is measured in 'cycles per second' or 'Hertz'. This is the **frequency** of sound. The higher the frequency, the higher-pitched it sounds.

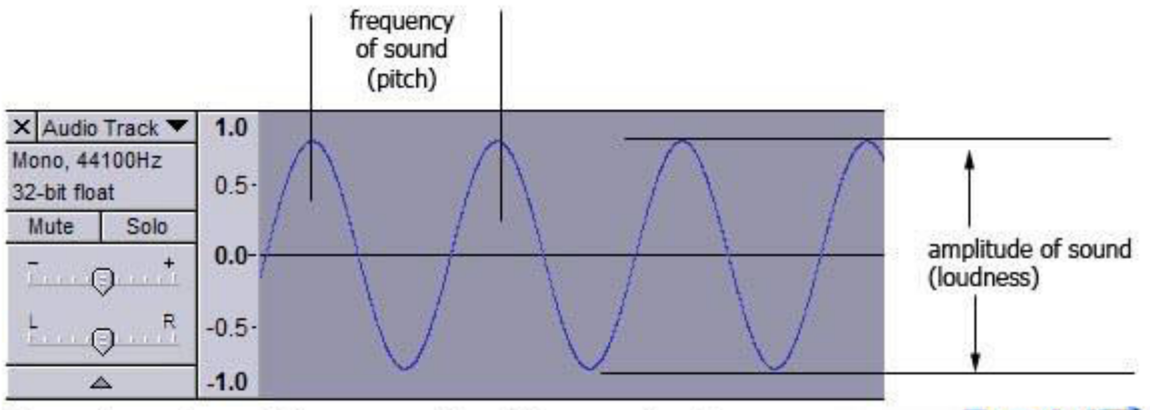
If you have headphones, listen to the two pure tones in the table below. One is low frequency and the other is high frequency.

Sound frequency

Low frequency (about 200Hz).mp3	This is a low frequency pure tone
High frequency (about 10KHz).mp3	This is a high frequency tone

Human ears (young ones at least!) can hear from 20 Hz to 20,000 Hz. But, as you get older, you lose the ability to hear tones on the higher end of the scale.

As well as the *frequency* of vibration, there is also the *loudness* of the sound. Sound is a wave - the loudness is in effect, the height of the wave. This is called the '**amplitude**' of the sound:

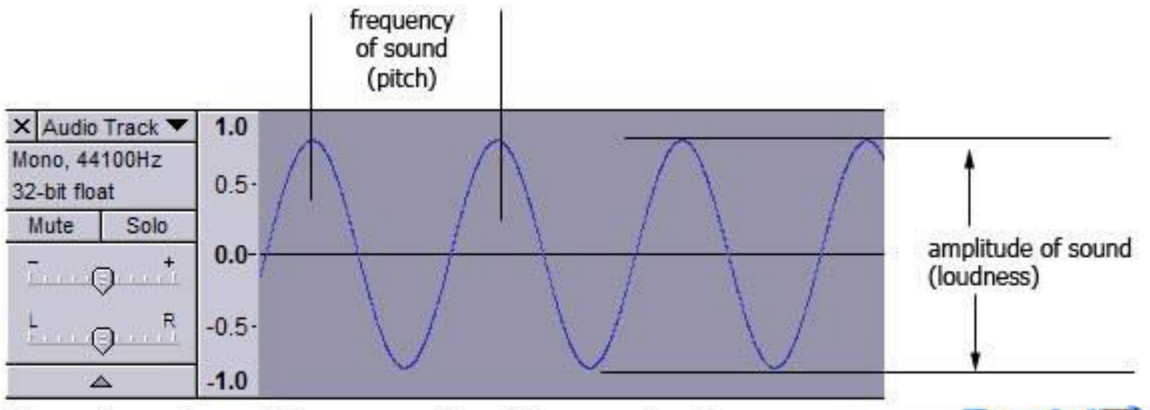


Sound as viewed in a sound editing application

4. Sampling Sound

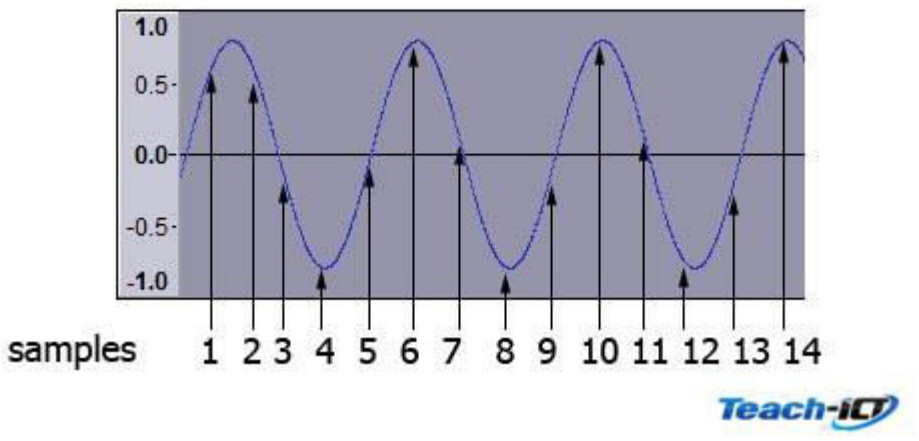
Sampling is a method of converting an analogue sound signal into a set of binary numbers that are then stored in a digital file.

The diagram below shows a typical sound wave:



Sound as viewed in a sound editing application

The idea of 'sampling' is to take measurements of the amplitude of the sound wave at various points in time. Like this:



Each of the samples above are measuring the amplitude of the signal at that instant. This results in a set of numbers like this:

Sample number	Measurement
1	0.6
2	0.5
3	-0.3
4	-0.8
5	-0.1

Sound

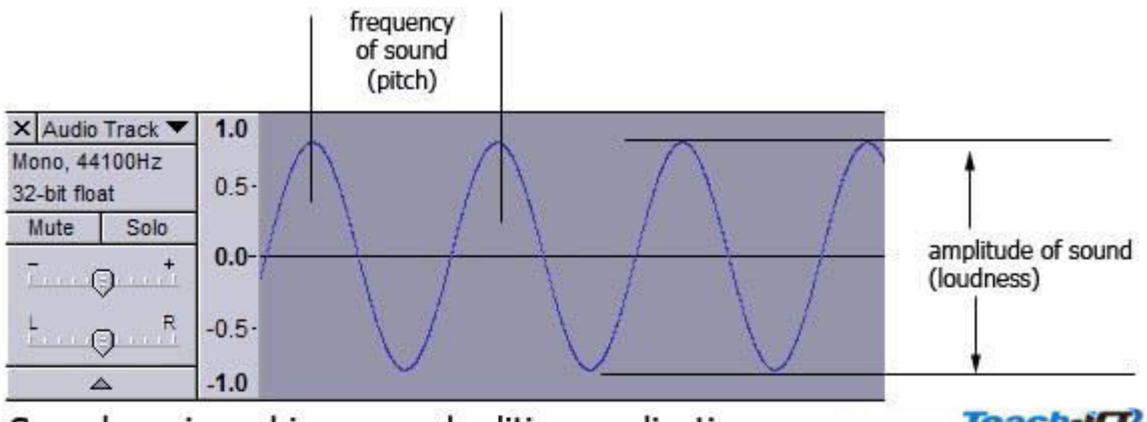
5. Storing Samples

The table below shows the set of sample measurements from the previous page.

Sample number	Measurement
1	0.6
2	0.5
3	-0.3
4	-0.8
5	-0.1

Each value is then converted into the equivalent binary number. The whole collection of samples is then stored in a digital file.

There may be tens of thousands of samples making up the complete sound track. So the longer the sound track - the larger the file becomes.



Sound as viewed in a sound editing application

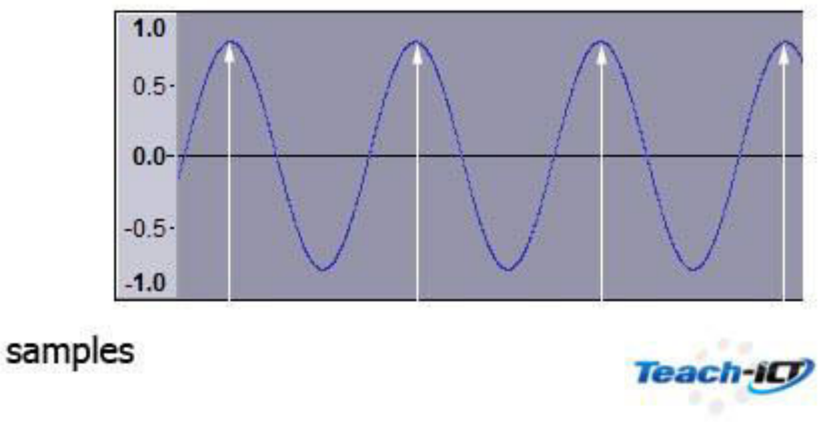
In the diagram above, do you notice the number on the top left of 44100 Hz? This is the **sample rate.** This is the rate at which samples are taken in this example - 44,100 samples per second.
The sample rate has a great effect on the *quality* of the recording as will be explained on the next page and of course it directly affects the *size* of the sound file, the higher the sample rate the larger the file.

6. Sample rate

This is the rate at which samples are taken - stated in Kilohertz (thousand times per second).

What happens if the samples are taken too slowly compared to the sound signal?

Have a look at the extreme example below. where the white arrows are regularly spaced samples.



The table of values recorded are shown below.

sample	Measurement
1	0.7
2	0.7
3	0.7

Every value is the same! All the frequency information of the sound has been missed. This shows that there is a minimum rate at which samples need to be taken in order to faithfully reproduce the original signal.
The minimum sample rate is at least twice the highest frequency in the signal.

So for a CD that has sound going up to 20KHz, the minimum sampling rate is 40 KHz. Things are not perfect, so the music industy uses 44.1KHz

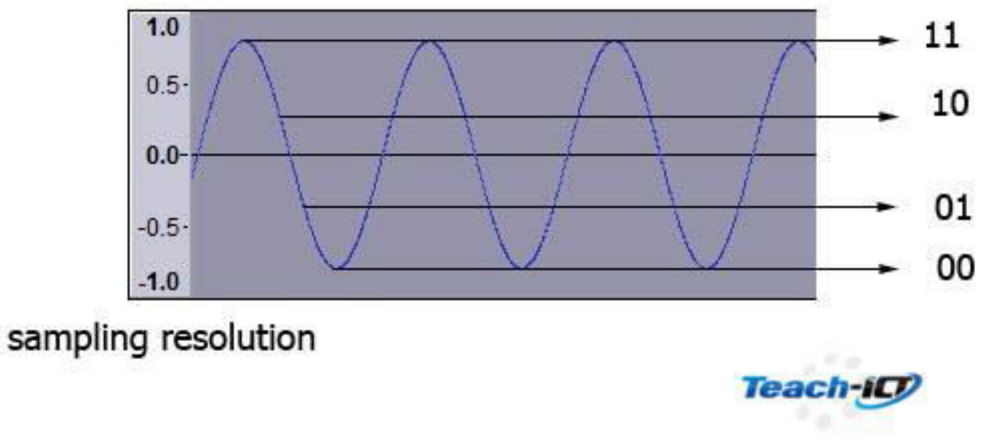
7. Bit depth

We have discussed the importance of sample rate in order to capture frequency information correctly. But there is another vital thing to consider as well.

Namely the *accuracy* of the measurement.

Bit depth is the number of bits used per sample

A 2 bit binary number can only represent 4 possible values.



In the diagram above, a sample can only have four values as represented by the 2 bit number. This is far too crude for most applications.

For low quality applications, it is normal to use 8 bit resolution.

With careful adjustment of the sound level, 8 bits will give you 256 discrete levels, with the maximum value 1111 1111 representing the maximum value of the sound, and 0000 0000 representing the minimum value of the sound.

If you have headphones, listen to the effect of bit depth and sample rate in the recordings below

8 KHz 8 bit.mp3	0.2 Mb	Telephone
22KHz 8bit.mp3	0.4 Mb	Basic
44 KHz - 16 bit.mp3	1.0 Mb	CD quality

The first example used a sample rate of 8 thousand samples per second (8KHz) and a bit depth of 8 bits. It is only barely acceptable quality - but it is the smallest file size.

The second example also uses a bit depth of 8 but used a sample rate of 22 thousand samples per second (22KHz) - the quality of the sound is far better but the file size is twice as big. In real life, this is a good compromise between quality and file size. For example recording an audio interview with someone, where the voice is clear but it does not need to be perfect quality

The third example is the industry standard for recording high quality music - it uses 44KHz sample rate and 16 bit depth. The quality is superb but the file size is large.

So there is a balance to be struck between sample rate, bit depth, sound quality and file size

Sound

8. Channels

When you are digitising sound, it is important to think about how the sound is going to be played back. Humans use two ears to hear, so it is possible for us to locate where a sound is coming from.

In order to reproduce this effect in an audio file, there needs to be at least two channel recordings, one for the left ear and the other for the right ear. This is called a stereo recording.

'Surround sound' is also popular to get this space sensation in which case 6 or more channels are recorded.

The simplest sound recording is the single channel or mono recording - it has the benefit of smallest file size.

As you see - there is a direct compromise between sound file size and the quality of the sound experience.

Sound

9. Bit rate

The sample rate, bit depth and number of channels is combined to what is called the 'bit rate'.

Bit rate

The bit rate is the amount of information required to describe one second of a sound. The formula for working out the bit rate is shown below:

Bit rate = sample rate x bit depth x number of channels

To work out the bit rate for one second of audio, multiply the number of samples taken in one second by the number of bits needed for each sample for each audio channel, and then multiply by the number of channels. The unit for bit rate is "bits per second".

A CD-quality audio recording with two (i.e. stereo) channels requires:-

Bit rate = 44,100 x 16 x 2 = 1,400,000 bits per second (1.4 Mbps)

This is quite a high bit rate. You would quickly produce very large files with a bit rate this high. Thankfully, *data compression* can be used to vastly reduce file size without sacrificing too much quality.

MP3 compression can reduce an uncompressed CD-quality bitrate of 1.4 Mbps to about 192 Kbps and yet retain almost the same quality.

Sound

10. File size

Now that you know how to work out the bit rate - the number of bits per second of audio - it's quite straightforward to work out the total size of an audio file. You simply take the bit rate and multiply it by the length of the file in seconds.

Storage calculation

File size = bit rate* x length

Bit rate* = sample rate x bit depth x number of channels

This will give you an answer in bits (though it's usually convenient to convert this further to bytes).

For example, if these are the details - what is the storage requirement?

- Sample rate: 44.1 KHz
- Bit depth: 16 bits
- Number of channels 2
- Recording time: 180 seconds

Storage = Sample rate x bit depth x channels x time

By slotting in the details above we get

Storage = 44100 x 16 x 2 x 180 = 254 million bits

As there are 8 bits to a byte. The storage for an uncompressed 3 minute, CD quality recording is:

Bytes of storage = 254 Megabits / 8 = 31 Megabytes.

Sound

11. Metadata

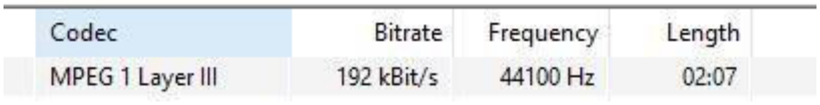
In order for an audio player to play back a sound file, the file itself needs to contain some additional basic information as well as the samples. This extra information is called 'metadata' and it is usually embedded at the beginning of the file.

Basic metadata

- Sample rate

- Bit rate
- Duration
- Codec

Here is a screen shot of the metadata in a typical MP3 sound file:



Codec	Bitrate	Frequency	Length
MPEG 1 Layer III	192 kBit/s	44100 Hz	02:07

The Codec is a shortening of **Cod**e **Enc**ode and describes the algorithm used to create and play back the file. In this case the codec is the MPEG 1 Layer III algorithm.

Additional Metadata

File formats such as MP3 support a large number of extra metadata. For mp3 the metadata are also called 'tags'. Here is a short list of common tags:

- Title of song
- Artist
- Track number
- Year
- Album

Sound

12. Summary

- Sound is an analogue wave in the air
- Music is usually composed of many frequencies from 20Hz to 20KHz
- Sampling involves taking a measurement of the wave at an instant in time
- The measurement is converted into a binary number to represent the wave at that instant
- The longer the sampling lasts, the larger the file will be.
- You need at least twice the highest frequency of the wave to be able to reproduce it accurately
- The standard CD quality sampling rate is 44.1 KHz

- The faster the sampling, the more binary numbers it creates per second and so the file size gets bigger

- Bit depth is how many bits are used per sample

- A bit depth of 8 bits allows 256 levels of the wave to be measured

- A bit depth of 16 bits allows 32,000 levels to be measured, much more accurate

- Doubling the bit depth will double the file size - each sample is now two bytes instead of one

- The higher the bit depth, the better the perceived quality of the sound

- Bit Rate (bits per second) is the product of sample rate, sample bit depth and channels

- 16 bit depth is good enough for CD quality sound if also sampled at 44KHz to capture the highest frequencies.

- An uncompressed file format is WAV and a compressed format is MP3

Compression

1. Introduction

Compression is the process of reducing the size of a data file whilst still retaining most, or even all, of the original information. You can compress most types of file, including documents, music files, video and image files.

Data streaming services compress their stream in real-time to reduce the amount of bandwidth needed to provide the service. If they streamed uncompressed data, the service would be very slow and high in bandwidth requirements.

Why do we compress files? The main reasons are:

- Less storage space required for files
- Faster download and uploads
- Smaller file attachments for email
- Coping with slow links when streaming

We will look at compression in more detail over the next few pages.

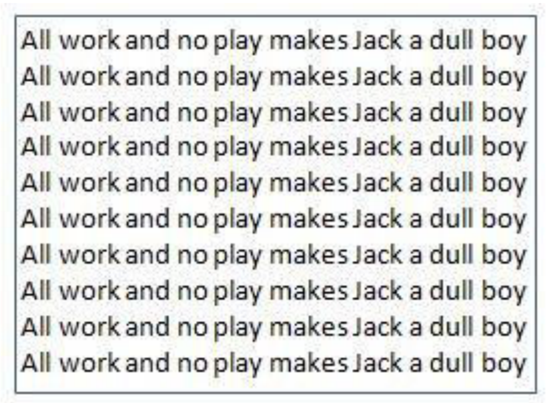
Compression

2. Basic principle

Compression depends on *patterns* being present in the information. This is true of text, image, video or music data.

A pattern implies that some of the data is identical, but is just located in a different place in the file.

Consider a text file that just contains the same statement over and over again.



All work and no play makes Jack a dull boy
All work and no play makes Jack a dull boy
All work and no play makes Jack a dull boy
All work and no play makes Jack a dull boy
All work and no play makes Jack a dull boy
All work and no play makes Jack a dull boy
All work and no play makes Jack a dull boy
All work and no play makes Jack a dull boy
All work and no play makes Jack a dull boy
All work and no play makes Jack a dull boy

The original data file has a 100 words in it, ignoring the spaces. But the *information* (as opposed to the data) is really just the first sentence containing the ten words "All work and no play makes Jack a dull boy". This sentence is repeated 10 times.

A text compression algorithm would spot this pattern and create a file that effectively only contains the first 10 words plus some instructions about repeating it.

Therefore 100 words become just 10 along with a few additional details. As a result, this has compressed the file by a factor of almost ten.

The efficiency of compression is a simple formula: -

$$\text{Compression ratio} = \frac{\text{Original data size}}{\text{Compressed data size}}$$

Compression is also used by video and music streaming services to reduce the bit rate per second. The compression formula then becomes

$$\text{Compression ratio} = \frac{\text{Original data rate}}{\text{Compressed data rate}}$$

Compression

3. Lossy compression

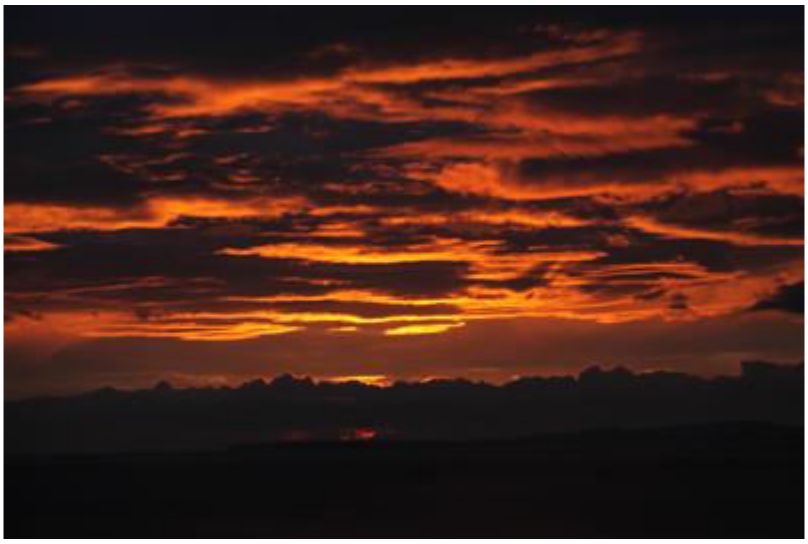
If file size is an issue, then it may be acceptable to *disregard* some of the original information. That way, less data needs to be stored. This is called '**lossy compression**'.

Lossy compression depends on patterns being present within the information, as mentioned on the previous page. The algorithm looks at the data, tries to identify the patterns and decides how much it can throw away without noticeably affecting the quality of the data.

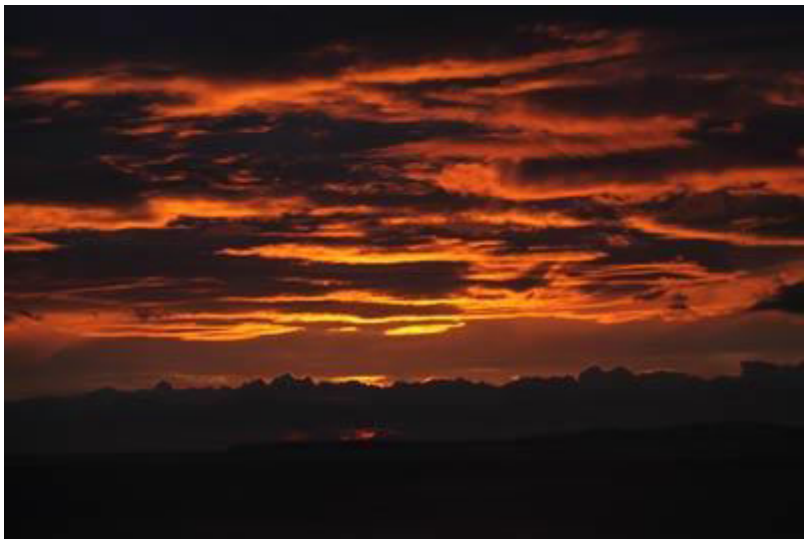
Different file types can be compressed to different amounts. Some information can be safely discarded, while other information may be essential to retain.
For example a document needs to store every character. Imagine the word *fury* and *furry* in a sentence, they mean something completely different - yet they are only one character different - you cannot afford to lose that single character.

On the other hand a music file will have sounds at frequencies that the human ear cannot hear. Those frequencies can be safely discarded without the listener ever noticing their absence.

Here is an example of lossy compression using images:



File size 75Kb
Quality set to 100%



File size 13Kb
Quality set to 45%

Would you say that the 45% quality jpeg is much worse than the 100% quality one?

It is slightly more blurry, but probably not enough to matter in most circumstances. But losing the information has resulted in the file size being reduced from 75Kb down to 13Kb. This means it would take up a lot less storage space on your device and also, if you were opening a web-page, it would render 5 times faster.

For a video file this is even more significant. Perhaps a film would take an hour to download in uncompressed format whilst a well-compressed version would be five times quicker (12 minutes).

Here are some very popular lossy compression standards for music, image and video

- Music : MP3 files
- Image: JPG files
- Video: MP4 files

Compression

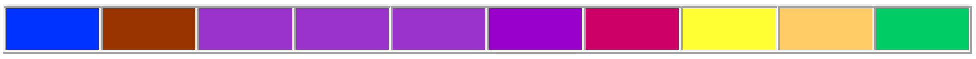
4. Lossless compression

But what if you need to make a file smaller while keeping everything?
The answer is to use **lossless compression**.

Lossless compression reduces a file's size with no loss of quality. It is less effective at reducing file sizes than lossy compression, but the trade-off is that there's no loss of quality.

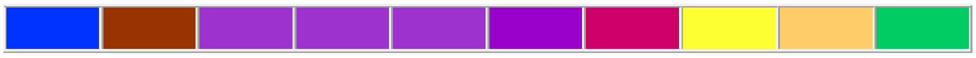
This is done by using patterns.

For example, a simple way to store the following colour information is to store a byte for every item

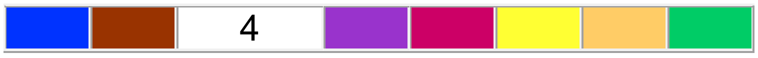


You could store the ten colours as 10 bytes. There is no compression - there is a byte for every colour.

Now consider this - notice that there are 4 identical purple squares one after another?



So there is a **pattern** within the information - we could take advantage of this pattern by just declaring how many purples-in-a-row there are within the file. Like this



The file now contains 8 bytes - a 20% saving in file size, and yet no information has been lost.

A software application could read this file and notice the '4' in it, it then knows that the next item needs to be repeated 4 times. So the output is exactly the same as the original information.

- PNG is an example of a lossless image format
- WAV and FLAC are lossless audio formats

- There are very few lossless video formats - lossy compression is much more popular here as even with compression, lossless video files can be huge.
- A very popular lossless compression format for storing any kind of file is .ZIP.

Compression

5. Summary

- Compression means to reduce the size of a data file
- Lossless compression does not lose any data
- Lossy compression disregards some information but degrades the quality of playback or viewing
- Lossless compression is typically used for text and spreadsheet files
- Lossy compression is typically used for image, music and video files
- Compression ratio is the original size divided by the compressed size
- Lossy compression is generally more effective at reducing file size. It has a higher compression ratio.
- Data streaming compression means to reduce the bit rate of the video or music stream
- For video, a popular lossy compression format is MP4
- For images, a popular lossy compression format is JPEG
- For music, a popular lossy compression format is MP3
- For lossless compression of data files, a popular choice is ZIP