Computer Systems Architecture

Assignment 1

The Devil is in the Data

George Hotten

Complete the Sentences – Converting from Denary

Numeric data is stored in a computer system in the following forms shown below. These are used because computers cannot represent numbers in traditional forms as computers function using electrical signals, which can only be on or off. This means computers can only store data as 0s and 1s. Therefore, base 2 was created to represent numbers with just two values: 0s and 1s.

Converting from Denary

Here is an example of converting 74 into the following number systems:

Binary (base 2)

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

74 in binary would be **01001010** as 64+8+2=74

Octal (base 8)

First, we divide 74 by 8.
74 ÷ 8 = 9.25

Now we identify the remainder.
8 × .25 = 2 – 9R2

Now we divide 9 by 8.
9 ÷ 8 = 1.125

Now we identify the remainder
8 × .125 = 1 – 1R1

Now we divide 1 by 8.
1 ÷ 8 = 0.125
With remainder of 8 × .125 = 1 – 0R1

Reading our results backwards and using the remainders (0R**1**, 1R**1**, 9R**2**), we can confirm 74 in octal is **112**.d

Hexadecimal (base 16)

Let's start by reusing our table from binary and splitting it into chunks of 4 bits.

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

Now from each 4-bits, we can produce a hex number.

From the first (furthest to the left) 4 bits, the only 'on' bit is from 4. So our first hex digit will be **4**.

In our second set of 4 bits, 8 and 2 are 'on'. By adding 8+2 we get 10, which cannot be represented in hex. Therefore, we switch it to an **A**, making that our second digit. This is because hex can only represent 0-9 in denary. Anything higher becomes a letter, up to F (15). For example, 10 – A, 11 – B, up until F – 15.

This confirms that 74 in hexadecimal is **4A**.

Complete the Sentences – ASCII

The characters on a computer keyboard are stored on a computer system in binary but represented as hexadecimal. For example, in ASCII A is represented as 0x41 (hex) but stored as 1000001. Hexadecimal is used to make it easier to read and understand ASCII.

"Computer" in ASCII

To help me, I will use the following ASCII table:

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Source: https://commons.wikimedia.org/wiki/File:ASCII-Table-wide.svg

C	o	m	p	u	t	e	r
43	6F	6D	70	75	74	65	72

Please continue overleaf.

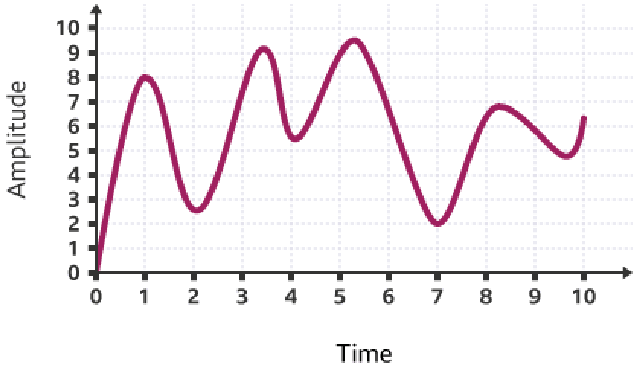
Sound and Bitmaps – how are they converted and stored?

Sound

As with all data on a computer, it must be stored in binary so computer can process it. This is done via an Analogue to Digital Converter (ADC). Microphones capture changes in air pressure, which is translated into electrical voltages, then digitised to bytes of data via the ADC.

Time and Amplitude

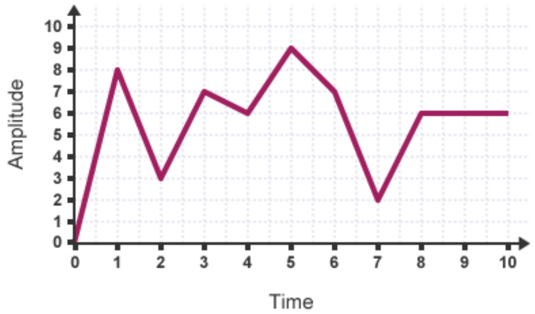
The ADC samples (see below) soundwaves received at a fixed rate and measures the amplitude (height) of the wave. The recorded sound at each sample is converted to the closest numeric equivalent.



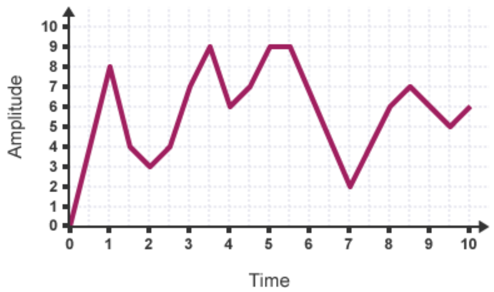
Sample rate

Sample rate is the number of samples of audio recorded per second. A higher sample rate means the amplitude is measured more times per second, and thus a higher quality audio file. Sample rate is measured in hertz.

Here is an example of a sample rate of 1hz.



Compared to a sample rate of 2hz.



The advantages of a higher sample rate are that it can increase the resolution of digital audio signals, and that it can allow for a greater range of frequencies to be captured. The disadvantages of a higher sample rate are that it can require more storage space, and that it can increase the amount of data that must be processed.

Bit depth

Bit depth is the number of bits used to store each sample. A typical bit depth is 16, allowing for a resolution of over 65,000 values. However, for better quality audio 24 bits are used, allowing for over 16 million possible values.

Bit rate

Bit rate is the number of bits that are processed per second. Higher bit rates mean more data can be processed per second allowing for higher quality audio. Bit rates are measured in kilobits per second and can be calculated by *sample rate × bit depth*.

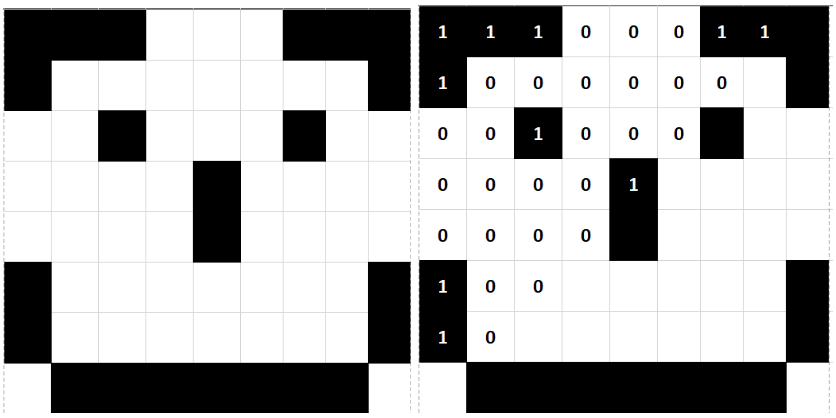
Bitmap Graphics

So that images can be stored on computers, they are broken down into picture elements (pixels).

Pixels and Colour

A pixel is the smallest individual element in a bitmap image. A pixel contains information about the colour of the image at that specific point. The colour is generally represented by a combination of three colours: red, green, and blue (RGB). This is stored in a binary value known as the "bit-plane". Every bit doubles the number of colours available. For example, 1 bit gives two colours, whilst 2 bits give 4 and 3 bits give 8.

A 1-bit image would give us monochrome: 0 for white and 1 for black.



This is an example of an 8 x 9 pixel image using 1-bit colour.

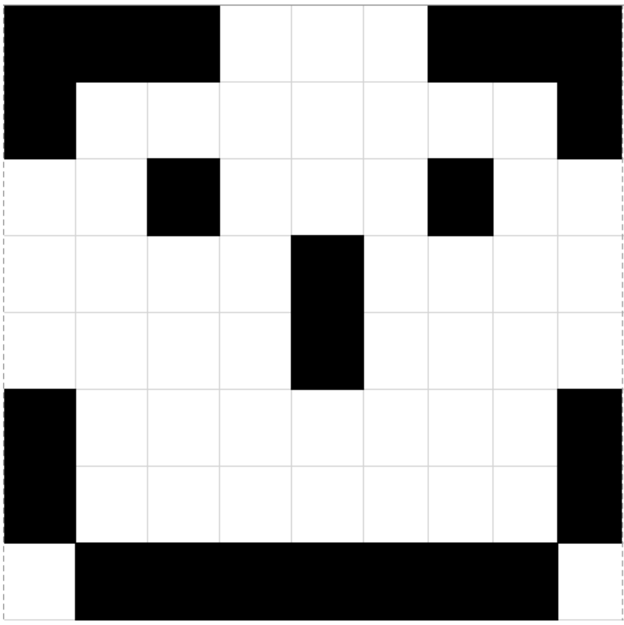
Resolution and Pixels

The resolution is the number of pixels in an image, meaning the more pixels you have the higher quality the image. Usually, resizing a bitmap image makes it more pixelated as the pixels must be expanded to fill the extra space.

The number of pixels in an image can be measured in pixels or megapixels. For example, a photo that has a resolution of 3024 x 4032 would have a total of 12,192,768 pixels or 12 megapixels.

Image Storage

These images are stored with scan lines, meaning each line is encoded left to right, top to bottom.



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

Our image's scan lines would look like this.

Image Metadata

To enable the computer to interpret the image, the following needs to be stored:

- Colour depth – this is the number of bits that represent each pixel.
- Resolution – the is the width and height of the image in pixels.

File Compression

When increasing the resolution and colour depth of an image, the file can skyrocket in size. This makes it difficult for images to be shown on the web for people with slower connections or making it more difficult to send and receive images. This is where compression comes in.

Lossy Compression

This compression loses some quality, usually by reducing the colour depth from 24 bits to 8bits. A typical lossy compression filetype is a JPEG.

Lossless Compression

This compression loses no quality, and functions by using a data compression algorithm that allows the receiver's device to fully reconstruct the image. A typical lossless compression filetype is a PNG.

Please continue overleaf.

Converting between Number Systems

Denary to Binary

123

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

64+32+16+8+2+1 = 123

01111011

252

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

128+64+32+16+8+4 = 252

11111100

9.125

Step one: 9 to binary

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

1001

Step two: .125 to binary

Calculation	Result	Binary (1 if whole number)
.125 x 2	0.25	0
.25 x 2	0.5	0
.5 x 2	1	1
.0 x 2	0	0
.0 x 2	0	0

00111

1001.00100*2^0

Normalization: 1.00100100*2^3

Converting the exponent – 2^3

3 + 127 = 130

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

10000010

To now format this, ensuring the mantissa 23 bits long (add extra zeros if it doesn't!):

Sign: 0 = positive, 1 = neg	Exponent	Mantissa (fractional)
0	10000010	100100100

Our number is then finally 01000001010010010000000000000000

Binary to Denary

1101010

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

64+32+8+2 = **106**

0111000

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

32+16+8 = **56**

011.011

8	4	2	1	0.5	0.25	0.125	0.0625
0	0	1	1	0	1	1	0

2+1+0.25+0.125 = **3.375**

Binary to Hexadecimal

1101010

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

4+2 = 6 | 8+2 = 10

6A

0111000

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

2+1=3 | 8

38

1000111

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

4 | 4+2+1 = 7

47

Denary to Hexadecimal

123

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

4+2+1 = 7 | 8+2+1 = 11

7B

252

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

8+4+2+1 = 15 | 8+4 = 12

FC

541

8	4	2	1	8	4	2	1	8	4	2	1
2048	1024	512	256	128	64	32	16	8	4	2	1
0	0	1	0	0	0	0	1	1	1	0	1

2 | 1 | 8+4+1 = 13

21D

Completing Floating Point Binary

111110100.011111

Normalization: 1.11110100011111*2^8

Converting the exponent – 2^8

8 + 127 = 135

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

10000111

To now format this, ensuring the mantissa 23 bits long (add extra zeros if it doesn't!):

Sign: 0 = positive, 1 = neg	Exponent	Mantissa (fractional)
0	10000111	11111010001111

Our number is then finally 01000011111111010001111100000000

1000011

Normalization: 1.000011*2^6

Converting the exponent – 2^6

6 + 127 = 133

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

10000101

To now format this, ensuring the mantissa 23 bits long (add extra zeros if it doesn't!):

Sign: 0 = positive, 1 = neg	Exponent	Mantissa (fractional)
0	10000101	1000011

Our number is then finally 01000010110000110000000000000000

110111000.100111

Normalization: 1.10111000100111*2^8

Converting the exponent – 2^8

8 + 127 = 135

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

10000111

To now format this, ensuring the mantissa 23 bits long (add extra zeros if it doesn't!):

Sign: 0 = positive, 1 = neg	Exponent	Mantissa (fractional)
0	10000111	110111000100111

Our number is then finally 01000011110111000100111000000000

How are Floating Point Numbers Represented in Binary?

Floating points can be represented in binary using the IEEE 754 standard, which was first introduced in 1985. The standard is made up of 3 components:

Sign of the Mantissa

Represents if the number is positive, 0, or negative, 1.

Exponent

Represents the power of the normalized mantissa.

Normalized Mantissa

This is the floating-point number. As we only have two digits in binary, a normalized mantissa has only a single 1 before the decimal point.

Conversion Example

As an example, I will convert 37.25671 to binary using the IEEE 754 standard.

Step one: convert the digits before the decimal point to binary

First, we convert 37 into binary.

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

This leaves us with the binary number **00100101**.

Step two: convert the digits after the decimal point to binary

This is done by multiplying the number by 2 a certain number of times for the precision we want for our number. After multiplying the initial .25671, we take the result's numbers after the decimal point and multiply that by 2, and repeat. If the number is a whole number, it will be represented by a 1, else a 0.

Calculation	Result	Whole Number	Binary
.25671 x 2	0.51342	no	0
.51342 x 2	1.02684	yes	1
.02684 x 2	0.05368	no	0
.05368 x 2	0.10736	No	0
.10736 x 2	0.21472	no	0

This leaves us with the binary number **01000**.

This can now be represented as $00100101.01000 \times 2^0$.

Step three: normalize the number

We now normalize the number so the decimal point is in front of the first 1.

Our number is now 1.0010101000 – with the leading zeros removed.

As we moved the decimal point 5 places, our exponent will be 5, making our new number: 1.0010101000×2^5.

Step four: convert the exponent

To convert the exponent, we must first add it to 127. In this example, 5 + 127 = 132. We now convert this into binary.

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

This means our exponent is **10000100**.

Step five: formatting the number

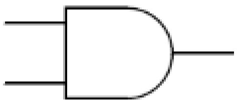
Finally, we can format our number in the following order: sign, exponent and mantissa, ensuring that the mantissa has a total of 23 bits.

Sign: 0 = positive, 1 = neg	Exponent	Mantissa (fractional)
0	10000100	10010101000

This leaves our final number as 01000010010010101000000000000000.

Logic Gates and Truth Tables

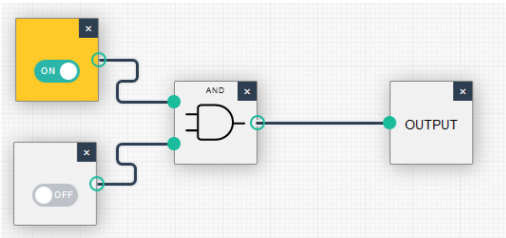
AND

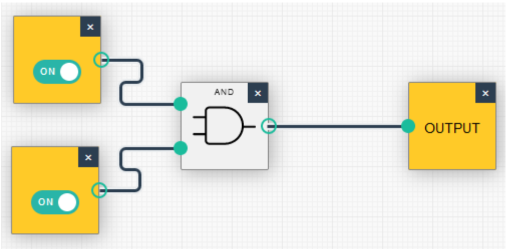


AND

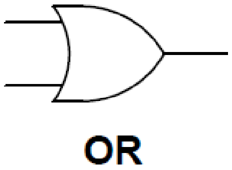
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

The AND gate takes two inputs, and only outputs a 1 when both inputs are 1.



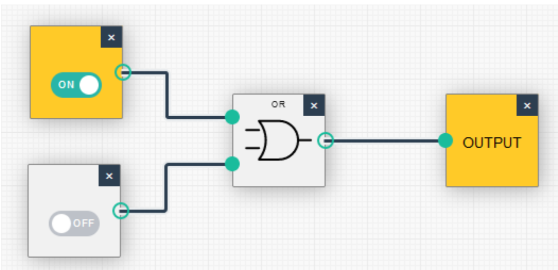


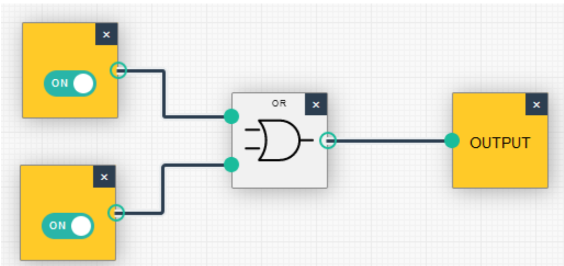
OR



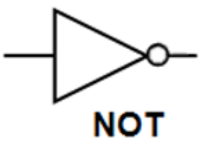
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

The OR gates takes two inputs and will output a 1 when either input is a 1.





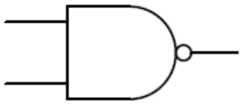
NOT



A	Output
0	1
1	0

The NOT gate takes only one input and will output the opposite of its input. For example, if 0 is inputted 1 is outputted.

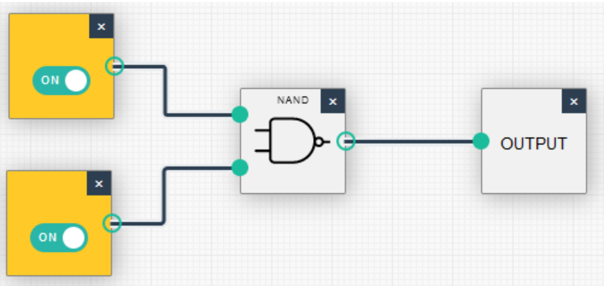
NAND

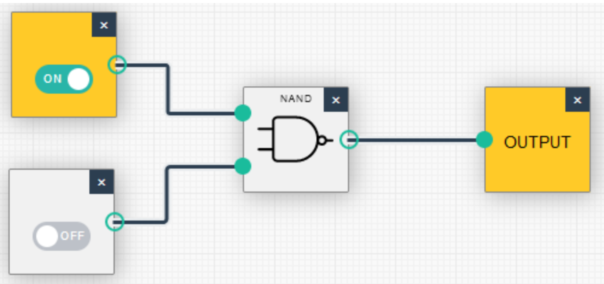


NAND

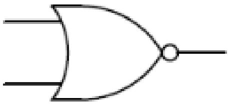
A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

Unlike the AND gate, the NAND gate requires that both inputs are **not** on to output a 1. If both inputs are a 1, a 0 will be inputted.





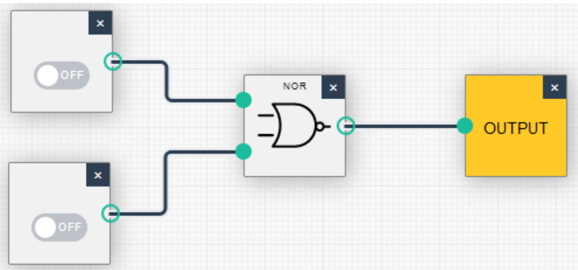
NOR

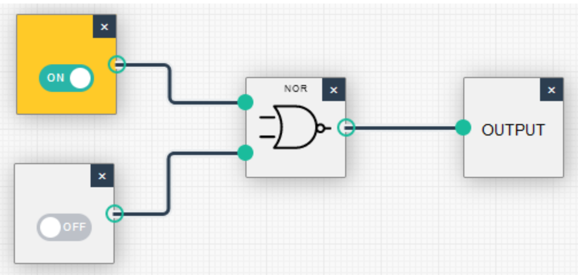


NOR

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

Unlike the OR gate, the NOR gate requires that **all inputs are off** to output a 1. If any input is a 1, the output will be a 0.





XOR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

The XOR gate will only output 1 if one input is a 1 whilst the other is a 0. If both inputs are 0 or 1, the output will be a 0.