INTRODUCTION TO

BASIC COMPUTER SCIENCE FOR DEVELOPERS

WHAT WE WILL LEARN

BINARY AND HEXADECIMAL NOTATIONS

ASCII AND UTF-8

LOGIC GATES AND BOOLEAN ALGEBRA

BINARY

In digital electronics and mathematics, binary is represented by a sequence of (*"1"s and "0"s)*. It's a base-2 number system as opposed to decimal which is base-10.

Computers use binary numbers because designing electronic circuits is easier when dealing with simple switches represented by two states, i.e. binary states: (*"On" and "Off"), ("True" and "False").*

Computers use binary to represent all data transported at the lowest levels. A 64 bit computer has a 64 bit data Bus that will transport 64 bits (binary digits) at once.

In the decimal system, there are 1s, 10s, 100s, ... positions (powers of 10 starting with 0)
In the binary system, there are 1s, 2s, 4s, 8s, ... positions (powers of 2 starting with 0)

```
Binary   Decimal equivalent
0     → 0
1     → 1
10    → 2
11    → 3
100   → 4
```

BINARY: CONVERTING TO DECIMAL

Let's convert **110011** to decimal:

(left-most bit) (right-most bit)

1	**1**	**0**	**0**	**1**	**1**	
32s digit	16s digit	8s digit	4s digit	2s digit	1s digit	
1 x 32	1 x 16	0 x 8	0 x 4	1 x 2	1 x 1	
32	16	0	0	2	1	Total: 51

Thus, 11011 in binary is equivalent to 51 in decimal.

BINARY: ARITHMETIC - HOW TWO BINARY NUMBERS ARE ADDED.

How do we add binary numbers? (*similar to decimal numbers*)

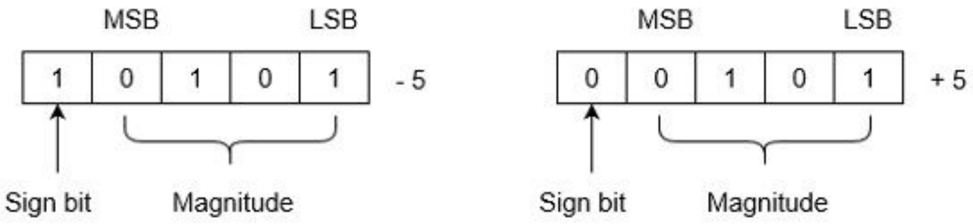
1+1=2, so the extra 1 carries over to next digit.

```
   11011
+     10
   11111
--------
```

BINARY: ARITHMETIC - NEGATIVE NUMBERS

We saw how positive numbers are represented in binary notation, but how to represent a negative binary number (-) ?
A negative binary number is represented by an extra bit or sign bit. There are numerous methods that we can use, two of which are described below:

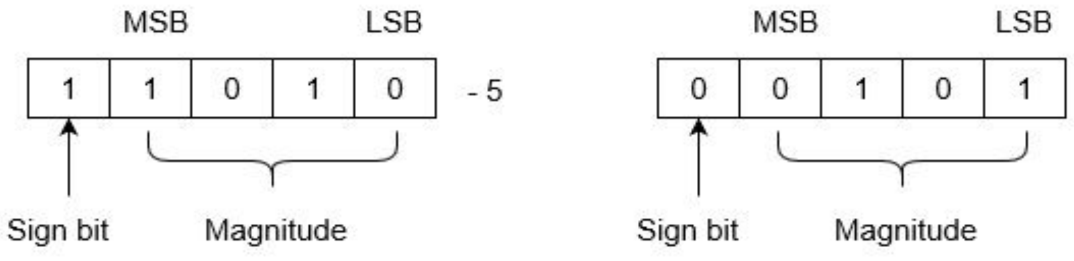
1. Signed Magnitude Method - using 5 bits register. The representation of -5 to +5 will be as follows:



In this method, number is divided into two parts: Sign bit and Magnitude. If the number is positive then sign bit will be 0 and if negative 1. Magnitude is represented with the binary form of the number to be represented.

BINARY: ARITHMETIC - NEGATIVE NUMBERS

2. 1's Complement Method - using 5 bits register. The representation of -5 and +5 will be as follows:



+5 is represented as it is represented in sign magnitude method. -5 is represented using the following steps:

(i) +5 = 0 0101

(ii) Take 1's complement of 0 0101 and that is 1 1010. MSB is 1 which indicates that number is negative.

MSB is always 1 in case of negative numbers.

DATA: MEASURING

Bit - binary digit (0 or 1) - “Smallest unit of data”

Byte - 8 bits

Kilobyte (KB) - 1000 bytes - Kibibyte (KiB) - 1024 bytes

Megabit (Mb) - 1000^2 bits

Megabyte (MB) - 10^6 bytes - Mebibyte (MiB) - 1024^3 bytes

Gigabytes (GB) - 10^9 bytes - Gibibyte (GiB) - 1024^6 bytes

Terabytes (TB) - 10^{12} bytes

Petabytes (PB) - 10^{15} bytes

DATA: TRANSFER RATES - SPEED OF SENDING DATA

Average number of bits or bytes transmitted (over a wire or wireless) in digital telecommunication.

Usually described by ISPs (Internet Service Providers) using either Megabits per second or Megabytes per second (Mbps or MB/s). It's important to make the distinction, otherwise you may well end up with a slower speed than anticipated.

HEXADECIMAL NOTATION: INTRODUCTION

Binary in long sequences is difficult and cumbersome to read.

For software developers using 0s and 1s to represent our data would be extremely time consuming and tedious and prone to frequent errors and misrepresentations.

Developers use the Hexadecimal notation as a more compact form of encoding binary. Hexadecimal is a base 16 notation using numbers 0-9 and letters A-F.

However, computers cannot directly use hexadecimal notation to perform instructions. Hexadecimal notation is translated into binary before being processed. It is used as a convenience notation.

HEXADECIMAL NOTATION

Below is a comparison between binary and hexadecimal notation:

Binary (24 digits):	0001	1111	0000	0111	1000	1010
Hexadecimal (6 digits):	1	F	0	7	8	A

Every 4 bits → 1 hexadecimal digit (1 - 9, and A - F)

HEXADECIMAL NOTATION

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

Convert 1F048A to binary:

1 F 0 7 8 A

000111110000011110001010

HEXADECIMAL NOTATION: COMMON USES

Amongst the uses of hexadecimal by Computer Scientist and software developers are:

1. Representing Colors - #000000 = ‘black’ and #ffffff = ‘white’
2. Shorthand for Memory Locations - two hexadecimal digits for every byte as opposed to eight digits in binary.
3. Security hashes and secret keys
4. IPv6 Addresses
5. Represent Mac (Machine Access Codes)
6. Representing the Braille alphabet - Braille is a symbol language for the visually impaired.

ASCII NOTATION

We saw how numbers are represented in binary. How do we represent text, for example, text in our emails?

ASCII (American Standard Code for Information) is one solution: Ascii uses a byte (8-bits) for each character represented.

There are only 128 characters (since we have 8 digits, $2^8 = 127$).

H	**e**	l	**l**	o	**,**	_	**R**	i	**c**	h	**a**	r	**d**	!
72	**101**	108	**108**	111	**44**	32	**82**	105	**99**	104	**97**	114	**100**	33

ASCII NOTATION

The 128 Ascii characters can represent only the English alphabet.

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

UNICODE AND UTF-8

Ever wondered how computers manage to deal with all languages on websites, apps, keyboards, and output to printers etc. etc. Wonder no more, Unicode to the rescue.

ASCII can only support 128 characters for the English alphabet, computers needed a different solution for the many other languages out there, together using thousands of different characters, and then there are emojis! It’s a complex alphabet soup!.

Ü ج ಜ 😁

How do we represent them? We use Unicode and UTF-8

Unicode: Assigns a code for every character in the world.
UTF-8: Represents a unicode character in binary.

UNICODE: DEFINES CODE POINTS FOR EVERY CHARACTER IN THE WORLD

CHARACTER	CODE POINT
A	U+0041
a	U+0061
0	U+0030
9	U+0039
!	U+0021
Ü	U+00DC
ڃ	U+0683
ಚ	U+0C9A
😁	U+1F601

UTF-8: DEFINES BINARY REPRESENTATION OF EVERY UNICODE CODEPOINT

CHARACTER	CODE POINT	UTF-8 BINARY ENCODING
A	U+0041	01000001
a	U+0061	01100001
0	U+0030	00110000
9	U+0039	00111001
!	U+0021	00100001
Ü	U+00DC	11000011 10011100
ڃ	U+0683	11011010 10000011
□	U+2070E	11110000 10100000 10011100 10001110
😁	U+1F601	11110000 10011111 10011000 10000001

KEY ADVANTAGES OF UTF-8

- **Covers all languages**:

  Up-to 4 bytes allows the representation of millions of characters
- **Backward-compatible with ASCII**:

  Representation of “Hello, Richard!” is the same in ASCII and UTF-8
- **Spatial efficiency**:

  More frequent characters take less space

Unicode is probably one of the most useful concepts you can learn in software development, so go ahead and learn as much about it as you can.

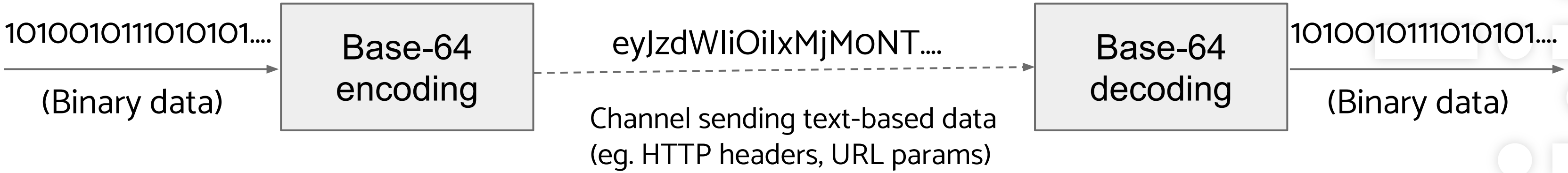
Click the link for a great in-depth introduction to Unicode - A great introduction to Unicode

SUMMARY: HOW HUMANS AND COMPUTERS REPRESENT NUMBERS AND TEXT

WHAT HUMANS CAN UNDERSTAND	WHAT COMPUTERS CAN UNDERSTAND
48300 (Base-10)	1011**1100**1010**1100** (Binary) B **C** A **C** (Hex)
FETHİYE (Latin alphabet)	01000110**01000101**01010100**01001000**11000100**10110000**01011001**01000101** UTF-8 46**45**54**48**c4**b0**59**45** Hex

BASE-64: ENCODING AND DECODING

Base-64 is a group of algorithms to represent raw binary data as textual data, for sending this data in a channel that only supports text-based data.
All data is encoded into 64 characters.



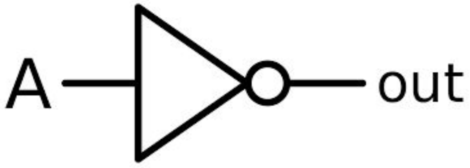
LOGIC GATES AND BOOLEAN ALGEBRA

Logic gates are basically switches that derive a logical state/output depending on their inputs. The state is determined by a set of logical operations based on Boolean Algebra.

In computer science Truth Tables are used to represent logical expressions of True and False states using a set of Boolean Operators.

The most common Boolean operators are **AND**, **OR** and **NOT**. Each operator has a standard symbol that can be used when drawing logic gate circuits.

NOT GATE



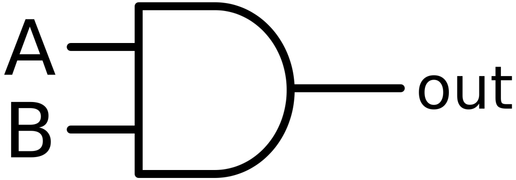
IF A is not true THEN
Do something...
END

Example:
IF speed is higher than 90 THEN
Reduce speed
END

Truth table

A	NOT A
false	true
true	false

AND GATE



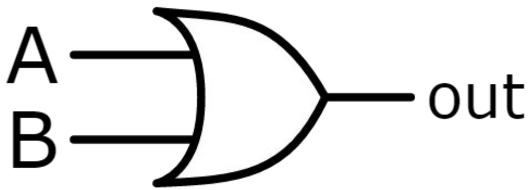
IF A is true AND B is true THEN
Do something...
END

Example:
IF it is raining AND it is cold THEN
Take a coat
END

Truth table

A	B	A AND B
false	false	false
false	true	false
true	false	false
true	true	true

OR GATE



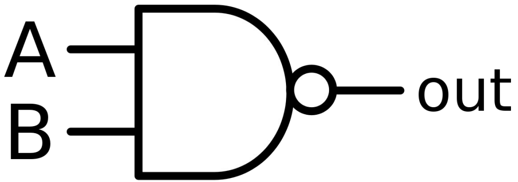
IF A is true OR B is true THEN
    Do something...
END

Example:
IF it is raining OR it is too sunny THEN
    Take an umbrella
END

Truth table

A	B	A OR B
false	false	false
false	true	true
true	false	true
true	true	true

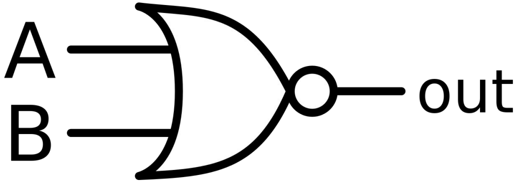
NAND GATE



Truth table

A	B	A AND B
0 / false	0 / false	1
0 / false	1	1
1	0 / false	1
1	1	0 / false

NOR GATE



Truth table

A	B	A NOR B
false	false	true
false	true	false
true	false	false
true	true	false

COMBINING OPERATIONS

A	B	C	(NOT(A) AND B) OR NOT(C)
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0