TECHNISCHE UNIVERSITÄT
CHEMNITZ

Professur für Schaltkreis- und Systementwurf
Stiftungsprofessur für Systemzuverlässigkeit in Elektromobilität und Energiemanagement

Introduction into Binary Number System, Data Representation & Boolean Logic

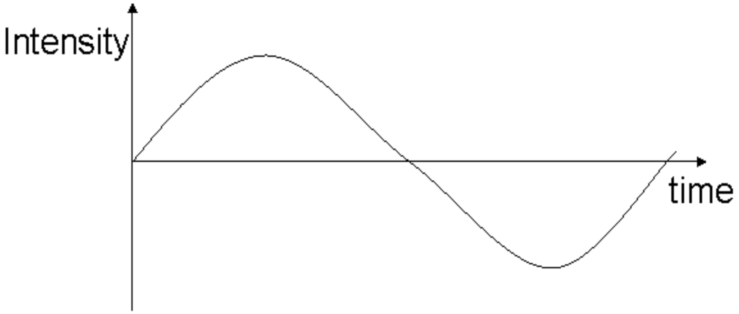
Prof. Dr.-Ing. Christian Paetz

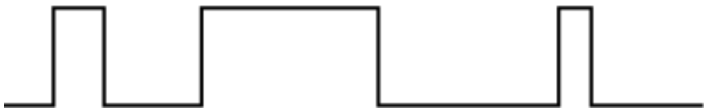
Objectives

- Describe the difference between analog signal and digital signal
- Introduce the binary number system and binary arithmetic
- Discuss how to convert between binary and decimal
- Explain how various data (positive integers, characters, colors) are represented in computers
- Reproduce the truth tables for the AND, OR, NOT and XOR Boolean operations and logic gates
- Trace the logic of the adder circuits composed of a few simple gates: half-adder, full-adder, multiple-bit-adder

Data Representation

- Data representation refers to the form in which data is **stored, processed** and **transmitted**
- Digital devices work with discrete data
- Analog devices work with continuous data





Analog Signal: continuous electrical signals that vary in time

Digital Signal: discrete electrical signals

Binary System in a Digital Computer

- Computers are made up of millions of switches
- Each switch has two states, „on“ or „off“
- This *binary system* is a natural and most easiest way for a computer to represent information and implement operations

Text:

```
CS1102 Introduction to Computer Studies
Computer programming is fun!
```

Binary:

```
01000011 01010011 00110001 00110001 00110000 00110010 00100000 01001001 01101110 01110100
01110010 01101111 01100100 01110101 01100011 01110100 01101001 01101111 01101110 00100000
01110100 01101111 00100000 01000011 01101111 01101101 01110000 01110101 01110100 01100101
01110010 00100000 01010011 01110100 01110101 01100100 01101001 01100101 01110011 00001010
01000011 01101111 01101101 01110000 01110101 01110100 01100101 01110010 00100000 01110000
01110010 01101111 01100111 01110010 01100001 01101101 01101101 01101001 01101110 01100111
00100000 01101001 01110011 00100000 01100110 01110101 01101110 00100001
```

Number Systems

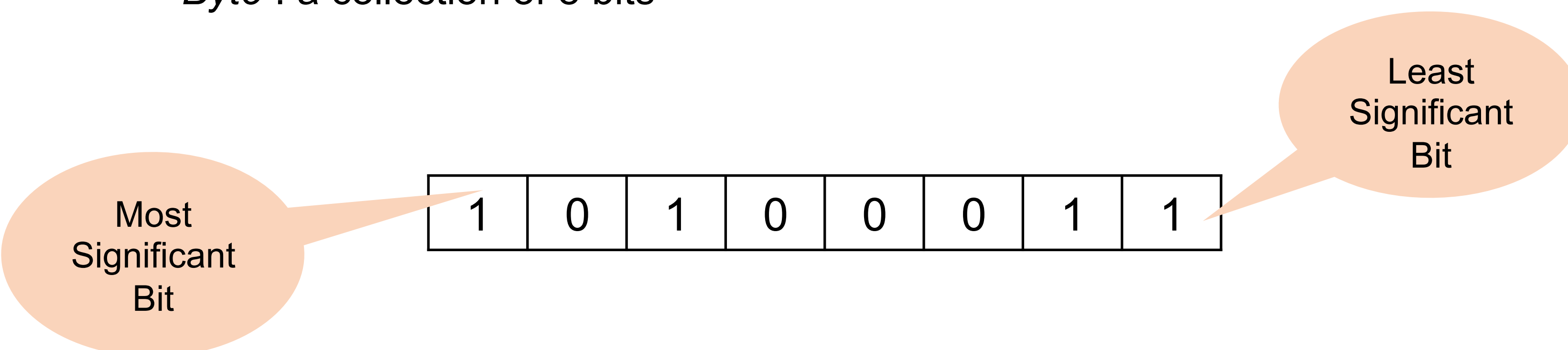
- Number system
  - Any system of representing numbers. Also called *numeral system*
  - The *base* of any number system is the number of digits in the system

- The number system most commonly used in are:

Decimal - 10 digits	0,1,2,3,4,5,6,7,8,9
Binary - 2 digits	0,1
Octal – 8 digits	0,1,2,3,4,5,6,7
Hexadecimal – 16 digits	0,1,2,3,4,5,6,7,8,9,A,B,C,D, E,F

Bit & Byte

- Computers operate on binary numbers
  - *Bit* (short for „*B*inary dig*IT*“)
    - the smallest unit of information
    - either **1** or **0**
    - representing <u>numbers, text characters, images, sounds, instructions and others</u>
  - *Byte* : a collection of 8 bits



- Kilobytes (KB): 2^{10} = 1,024 bytes
- Megabytes (MB): 2^{20} = 1,024 KB = 1,048,576 bytes
- Gigabytes (GB): 2^{30} = 1,023 MB = 1,073,741,824 bytes
- Terabytes (TB): 2^{40} = 1,024 GB = 1,099,511,627,776 bytes
- Petabytes (PB): 2^{50} = 1,024 TB = 1,125,899,906,842,624 bytes

Decimal: base-10 number system

Hundreds	Tens	Ones		Tenths	Hundredths
10^2	10^1	10^0		10^{-1}	10^{-2}
3	**7**	**5**	**.**	**1**	**5**

$3*10^2 = 3*100 =$ 300.
$7*10^1 = 7*10 =$ 70.
$5*10^0 = 5*1 =$ 5.
$1*10^{-1} = 1*.1 =$ 0.1
$5*10^{-2} = 5*.01 = +$ 0.05
375.15

Formula: $\sum$DIGIT * BASE$^{\text{POSITION \#}}$

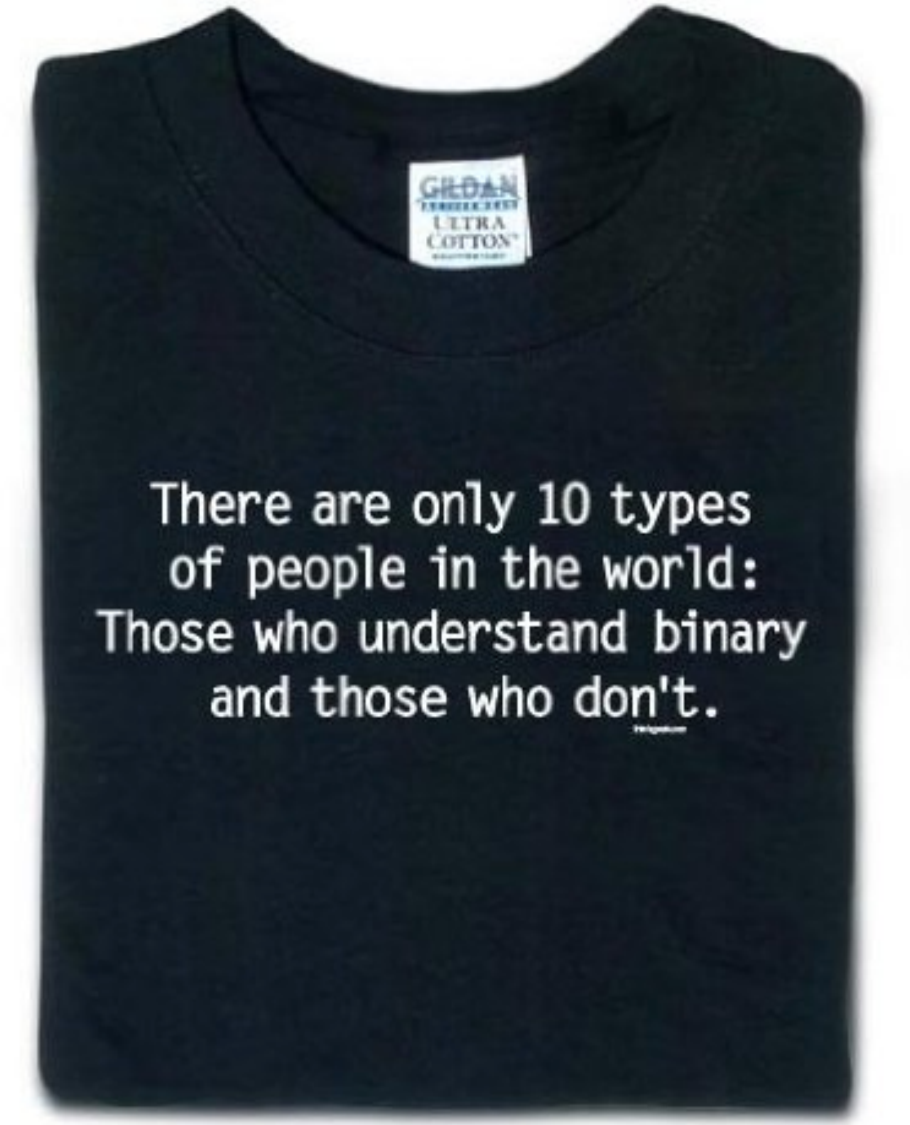
Binary: base-2 number system

- **Formula: $\sum$DIGIT * $2^{\text{POSITION \#}}$**

2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	
1	**1**	**1**	**0**	**1**	**0**	**1**	
16	+8	+4	+0	+1	+0	+0.25	= 29.25

- **Octal and Hexadecimal**
  - Octal – base 8 number system
  - Hexadecimal – base 16 number system
  - for example: $26_{10} = 11010_2 = 32_8 = 1A_{16}$

Binary	Octal	Decimal	Hexa-decimal
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F


GILDAN
ULTRA
COTTON
There are only 10 types
of people in the world:
Those who understand binary
and those who don't.

Decimal Integer to Binary

- Convert a positive decimal *integer* to a binary using **repeated division**
  - **Step 1** – divide the value by two and record the remainder
  - **Step 2** – continue to divide the quotient by two and record the remainder, until the newest quotient becomes zero
  - **Step 3** – the binary representation is the remainders listed from bottom to top in the order they were recorded

E.g., what's the binary for integer 13?

13 ÷ 2 = 6 ··· 1

Dividend Divisor *Quotient* *Remainder*

6 ÷ 2 = 3 ··· 0

3 ÷ 2 = 1 ··· 1

1 ÷ 2 = **0** ··· 1

Answer: 1 1 0 1

Decimal Fraction to Binary

- Convert a positive decimal *fraction* to binary using **repeated multiplication**
  - **Step 1** – multiply the fraction by two and record the integer digit of the result
  - **Step 2** – disregard the integer part and continue to multiply the fraction part by two, until the newest fraction part becomes point zero **or there is a repeated digit pattern**
  - **Step 3** – the binary representation is the integer digits listed from top to bottom in the order they were recorded
- Like decimal fractions, some binary fractions are *periodic* where a sequence of digits behind the decimal point (the period) is endlessly repeated
  - E.g. $0.6_{10} = 0.\underline{1001}1001_2 \ldots$

E.g., what's the binary for integer 0.375?

0.375 x 2 = **0**.75

0.75 x 2 = **1**.5

0.5 x 2 = **1**.0

Answer: **0. 0 1 1**

TECHNISCHE UNIVERSITÄT CHEMNITZ

Exercise: Real Numbers to Binary, Fixed Point

- A *real number* is a number that has a decimal point (called *floating point number* in computing languages)
- Convert decimal 13.375 to binary:
  - 13 = 1101
  - 0.375 = 0.011
  - 13.375 = 1101.011

Unfortunately, floating is not represented like this in computers

- More examples:
  - 133.3 = ?
  - -1345.57 = ?
- Standardize the representation of real numbers

Standardize Real Number Representation

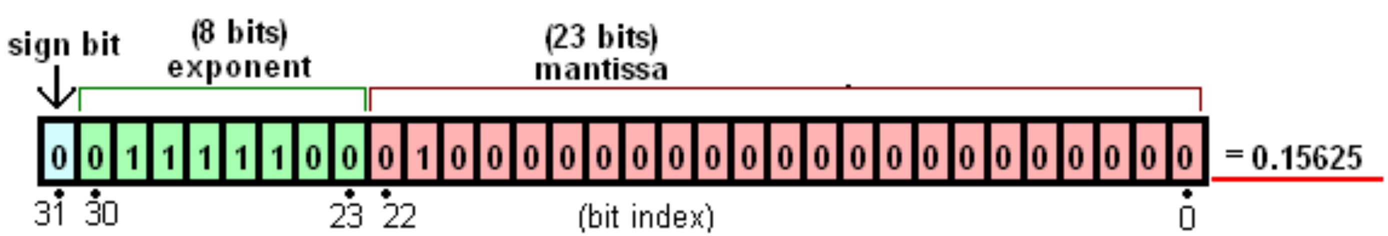
- Normalized representation of real numbers:

  20,000 -> 2.0 x 10^4; -0.0034 ->

  101.01 -> 1.0101×2^2; -0.00101 -> - 1.01×2^{-3}

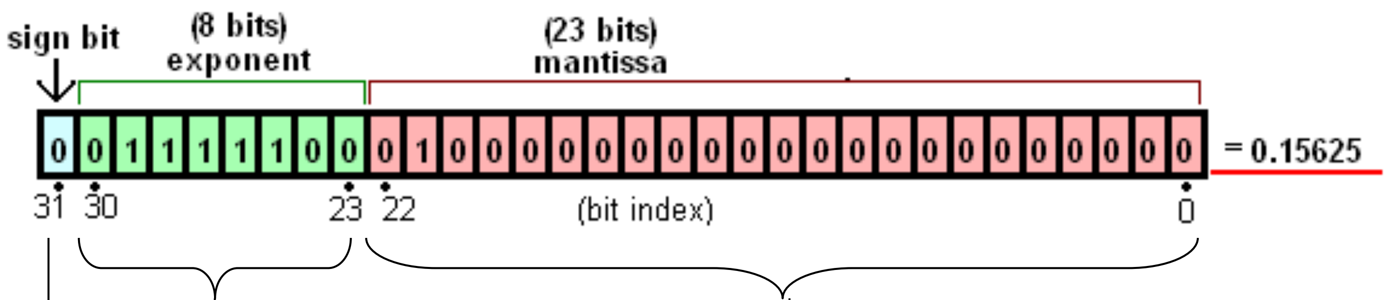
-3.4 is mantissa
-3 is exponent

- The integer part of a standard floating point number is always ‚1' , it is omitted in the representation
- Each real number has 3 parts: sign, exponent, fraction
- IEEE 754 (reference [4]): IEEE Standard for Floating-Point Representation of Normalized Binary Numbers



IEEE Floating Point Standard

$0.15625 = 1.01 \times 2^{-3}$



Let s = +1 (positive numbers) when the sign bit is 0
s = −1 (negative numbers) when the sign bit is 1

Let e = exponent - 127

Let m = 1.mantissa in binary

The number's value v is: $v = s \times 2^{e} \times m$

1101.011 -> 1.101011x2^3 -> 010000010101011000……

Binary Arithmetic

- Rules of Binary Addition
  - 0 + 0 = 0
  - 0 + 1 = 1
  - 1 + 0 = 1
  - 1 + 1 = 0, and carry 1 to the next more significant bit

  - E.g.

$$\begin{array}{rl} & 00011010 = 26_{10} \\ + & \underline{00001100} = 12_{10} \\ & 00100110 = 38_{10} \end{array}$$

- Rules of Binary Subtraction
  - 0 – 0 = 0
  - 0 – 1 = 1, and borrow 1 from the next more significant bit
  - 1 – 0 = 1
  - 1 – 1 = 0

  - E.g.

$$\begin{array}{rl} & 00110011 = 51_{10} \\ - & \underline{00010110} = 22_{10} \\ & 00011101 = 29_{10} \end{array}$$

Binary Arithmetic

- Rules of Binary Multiplication
  - 0 x 0 = 0
  - 0 x 1 = 0
  - 1 x 0 = 0
  - 1 x 1 = 1, and no carry or borrow bits
  - E.g.

```
      00101001    (41_10)
x     00000110     (6_10)
      00000000
      00101001
    00101001
    0011110110   (246_10)
```

- Binary Division: repeated process of subtraction
  - E.g.

```
                     1 1 0 1 1      (27_10)
1 0 1  )  1 0 0 0 0 1 1 1     (135_10)
(5_10)    -  1 0 1
               1 1 0
             - 1 0 1
                 0 1 1
               -     0
                   1 1 1
                 -   1 0 1
                       1 0 1
                     -   1 0 1
                             0
```

function divide(N, D)
R := N
while R ≥ D **do**
  Q := Q + 1
  R := R - D **end**
return (Q, R) **end**

Bytes Representing Signed Integers

- 2's Complement representation for signed integers
  - Designed to simplify the binary arithmetic, allowing the computer to perform all arithmetic operations using only addition
  - Based on the idea: y - x = y + (-x)

➢ Convert an negative integer to 2's complement :

1. Convert the number to binary
2. Negate each bit (0 →1, 1 → 0)
3. Add 1 to the binary

E.g., Convert -5_{10} to 2's complement using 4-bits ?

0101 (+5)

1010

1011 (-5)

Subtraction with 2‘s Complement

- Subtracting x from y ("y - x") with an n-bit 2's complement representation
  - Represent x in 2's complement
  - Add y and (-x)
  - Discard any bits greater than n

  - E.g., compute: “7 – 1” using 2's complement?

$$\begin{array}{rl} 0\,1\,1\,1 & (+7_{10}) \\ +\;\;1\,1\,1\,1 & (-1_{10}) \\ \hline 1\,0\,1\,1\,0 & (+6_{10}) \end{array}$$

Discard this overflow bit

Binary	Decimal
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	+0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

Bytes Representing Text

- Each character (letter, punctuation, etc.) is assigned a unique binary number
  - *ASCII* - American Standard Code for Information Exchange (primarily for English)
  - *Unicode*: represent the major symbols used in languages world side
  - E.g.,

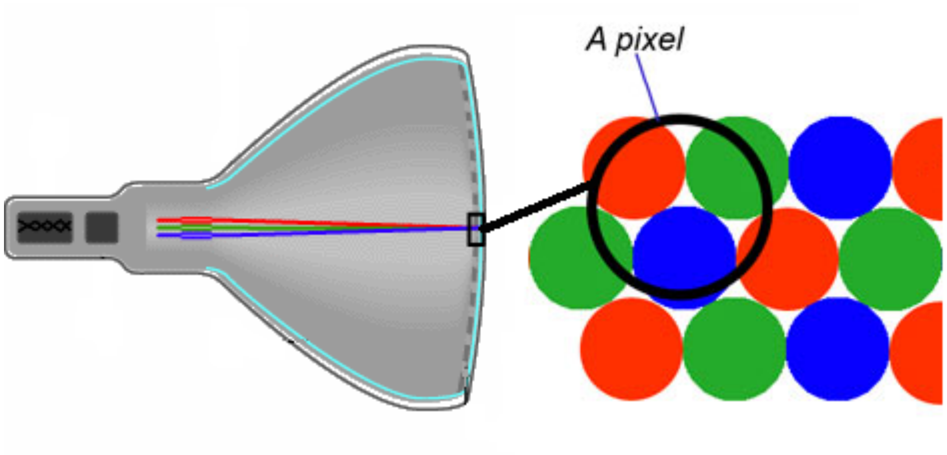
    Text: H e l l o !

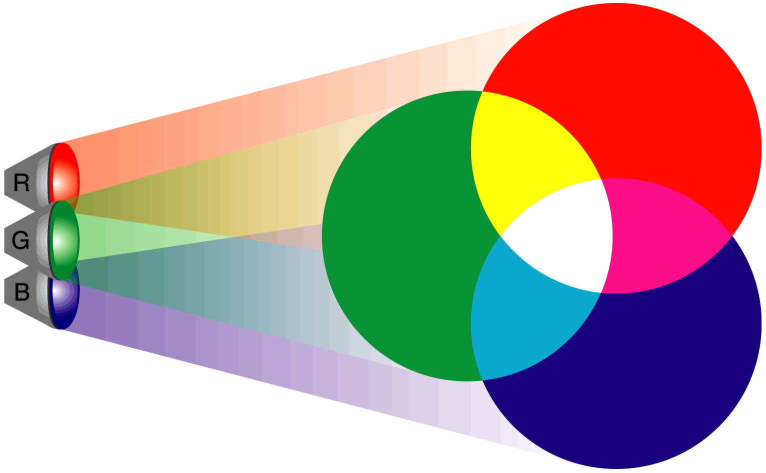
    ASCII: 48 56 6C 6C 6F 21

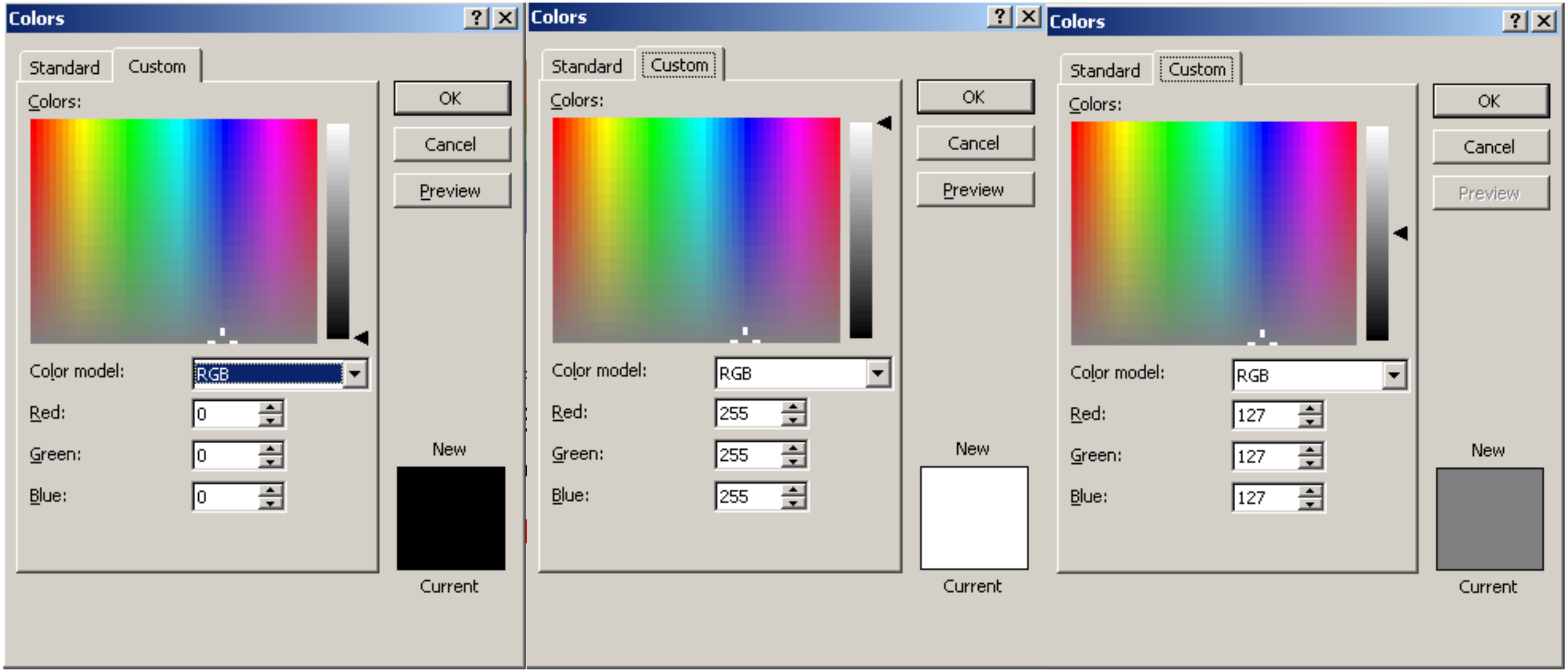
Decimal	Hex	ASCII	Decimal	Hex	ASCII	Decimal	Hex	ASCII	Decimal	Hex	ASCII
0	0	NUL	32	20		64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	"	66	42	B	98	62	b
3	3	ETX	35	23	#	67	43	C	99	63	c
4	4	EOT	36	24	$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	'	71	47	G	103	67	g
8	8	BS	40	28	(	72	48	H	104	68	h
9	9	HT	41	29	)	73	49	I	105	69	i
10	A	LF	42	2A	*	74	4A	J	106	6A	j
11	B	VT	43	2B	+	75	4B	K	107	6B	k
12	C	FF	44	2C	,	76	4C	L	108	6C	l
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	E	SOH	46	2E	.	78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	□

Byte Representing Colors

- A monitors screen is divided into a grid of small units called ***pixels***
- The more pixels per inch, the better the *resolution*, the sharper the image
- All colors on the screen are a combination of red, green and blue (RGB), just at various intensities
- *“True color” systems* require 3 bytes or 24 bits per pixel
  - There are also 4-bit and 8-bit color systems

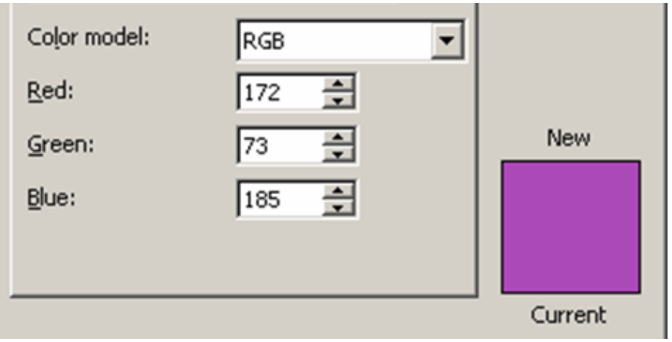


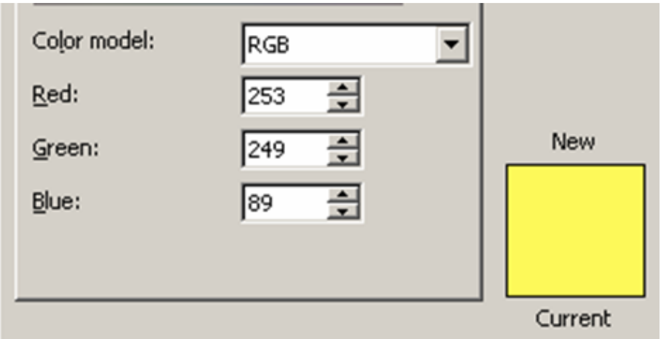




- Each color intensity of red, green and blue represented as a number from 0 through 255
- Black has no intensity or no color and has the value (0, 0, 0)
- White is full intensity and has the value (255, 255, 255)
- Between the two extremes is a whole range of colors and intensities
- Grey is somewhere in between (127, 127, 127)

Byte Representing Colors



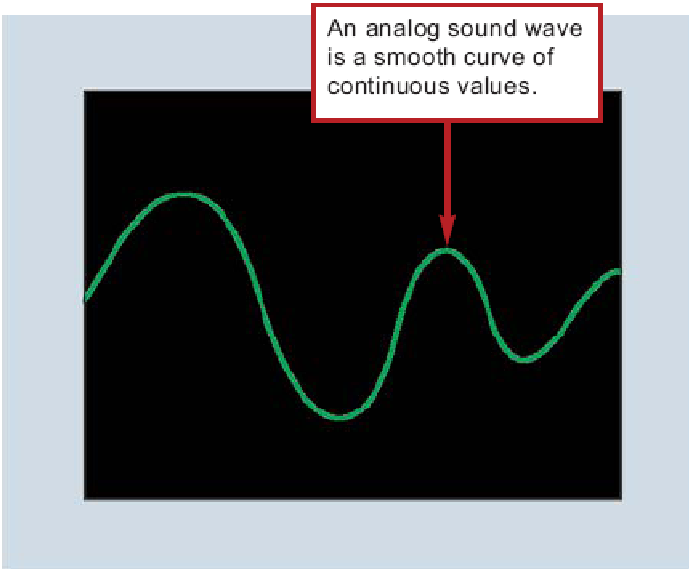


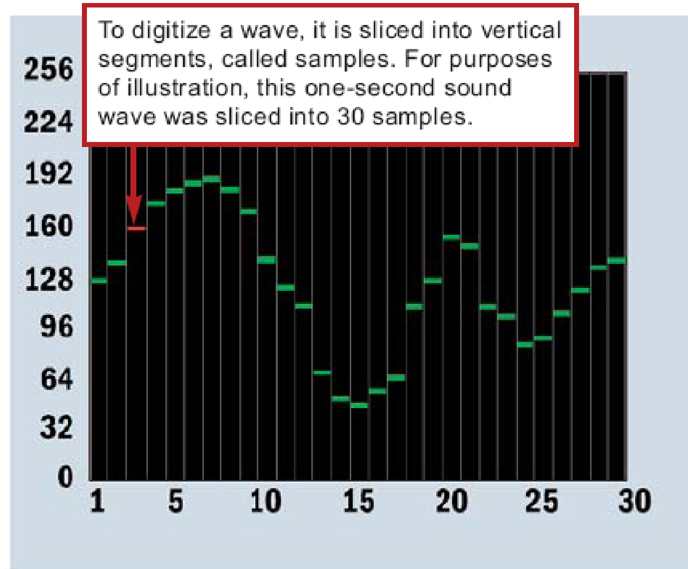
- Let’s convert these colors from Decimal to Hexadecimal

	Red	Green	Blue	
Purple:	172	73	185	#AC49B9
Yellow:	253	249	88	#FDF958

Note: in HTML, sometimes text or background color is defined in hexadecimal notation.

Byte Representing Sound





Sample	Sample Height (Decimal)	Sample Height (Binary)
1	130	10000010
2	140	1000110
3	160	10100000
4	175	10101111
5	185	10111001

The height of each sample is converted into a binary number and stored. The height of sample 3 is 160 (decimal), so it is stored as its binary equivalent—10100000.

Boolean Logic

Boolean Operations

- Boolean operation: an operation that manipulates one or more true/false values
  - True = 1, False = 0
  - Specific operations: AND, OR, NOT, XOR (exclusive or), NAND, NOR, XNOR (exclusive nor)
- Gate: a tiny electronic device that computes a Boolean operation
  - Often implemented as (small) electronic circuits
  - Provides the building blocks from which computers are constructed

Logic Gates

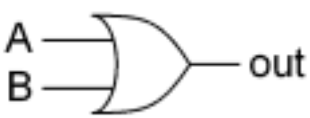
AND gate

The output is True when both inputs are True; otherwise, the output is False.

Input		Output
A	B	A **AND** B
0	0	0
0	1	0
1	0	0
1	1	1

False AND False is False
False AND True is False
True AND False is False
True AND True is True

OR gate



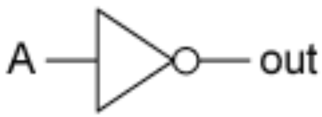
The output is False if both inputs are False; otherwise, the output is True.

Input		Output
A	B	A **OR** B
0	0	0
0	1	1
1	0	1
1	1	1

False OR False is False
False OR True is True
True OR False is True
True OR True is True

Logic Gates

NOT gate or inverter

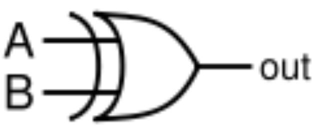


Input	Output
A	**NOT** A
1	0
0	1

NOT True is False

NOT False is True

XOR gate

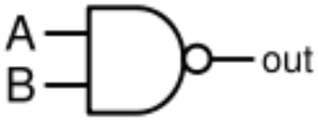


The output is True if either, but not both, of the inputs are True.

Input		Output
A	B	A **XOR** B
0	0	0
0	1	1
1	0	1
1	1	0

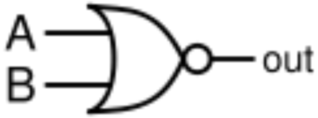
Logic Gates

NAND gate



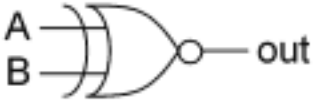
Combination of an AND gate with a NOT gate

NOR gate



Combination of an OR gate with a NOT gate

XNOR gate



Combination of an XOR gate with a NOT gate.
The output is True if both of the inputs are True or both of the inputs are False.

Review of gates

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

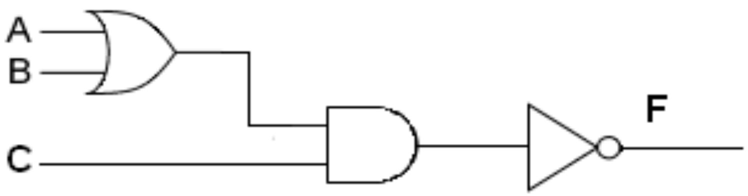
Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

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

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

From Logic Gates to Logic Circuit

- What does the following circuit compute?



Input			Output
A	B	C	F
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	0
1	0	0	**1**
1	0	1	0
1	1	0	**1**
1	1	1	0

Single-Bit Adder (1)

- Half-adder
  - A circuit that performs an addition operation on two binary digits
  - Produces a sum and a carry value which are both binary digits
- Logic combinations of single-bit sum

$$\begin{array}{cccc} 0 & 0 & 1 & 1 \\ \underline{+0} & \underline{+1} & \underline{+0} & \underline{+1} \\ 0 & 1 & 1 & 10 \end{array}$$

Carry-out

Sum = A XOR B

Carry-out = A AND B

Input		Output	
A	B	Sum	Carry-out
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

A half-adder

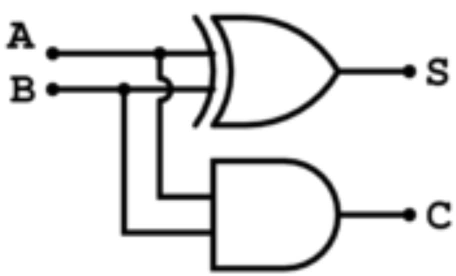


Image extracted from reference [6]

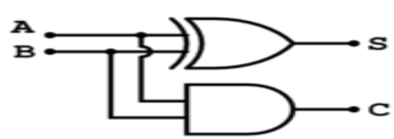
Single-Bit Adder (2)

- Full-adder
  - A logic circuit that performs an addition $A + B + C_{in}$
  - Produces a sum and a carry out
  - Note: $A + B + C_{in} = (A + B) + C_{in}$

Input			Output	
A	B	C_{in}	Sum	C_{out}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	er 0	1
1	1	1	1	1

Sum = (A XOR B) XOR C_{in}

C_{out} = (A AND B) OR (C_{in} AND (A XOR B))



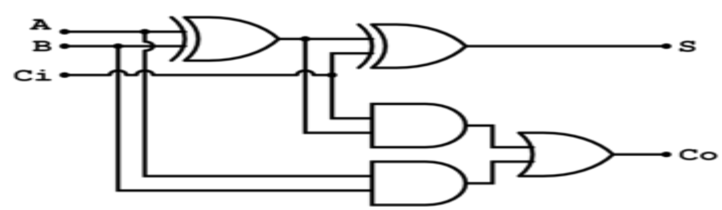
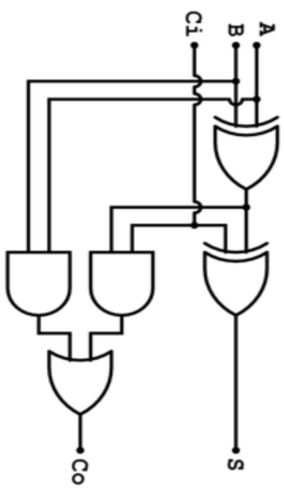
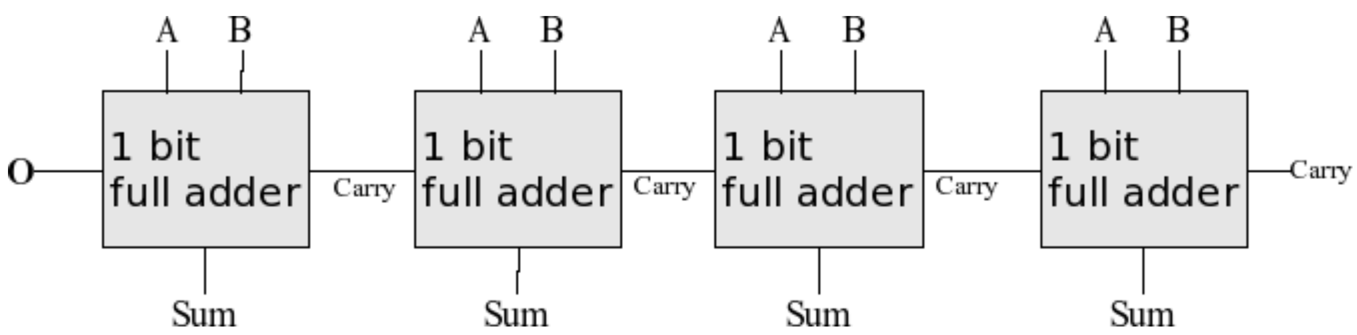


Image extracted from reference [6]

Multiple-Bit-Adder

- Using several full adders to add multiple-bit numbers
  - Each full adder inputs a C_{in}, which is the C_{out} of the previous adder
  - This kind of adder is a ripple-carry adder, since each carry bit "ripples" to the next full adder.





Summary

- Binary number system is the language which only the computer understands
- Conversion between the binary "language" and the language we already understand: the decimal system
- Various data (signed or unsigned integers, real numbers, text, multimedia or even instructions) are represented as binary inside computers
- Computers are built on a set of strict logic (Boolean logic); complex circuits that perform particular functions are constructed using the basic logic gates