Data

THERE'S BEEN A LOT OF CONFUSION OVER 1024 vs 1000, KBYTE vs KBIT, AND THE CAPITALIZATION FOR EACH.

HERE, AT LAST, IS A SINGLE, DEFINITIVE STANDARD:

SYMBOL	NAME	SIZE	NOTES
kB	KILOBYTE	1024 BYTES OR 1000 BYTES	1000 BYTES DURING LEAP YEARS, 1024 OTHERWISE
KB	KELLY-BOOTLE STANDARD UNIT	1012 BYTES	COMPROMISE BETWEEN 1000 AND 1024 BYTES
KiB	IMAGINARY KILOBYTE	$1024 \sqrt{-1}$ BYTES	USED IN QUANTUM COMPUTING
kb	INTEL KILOBYTE	1023.937528 BYTES	CALCULATED ON PENTIUM F.P.U.
Kb	DRIVEMAKER'S KILOBYTE	CURRENTLY 908 BYTES	SHRINKS BY 4 BYTES EACH YEAR FOR MARKETING REASONS
KBa	BAKER'S KILOBYTE	1152 BYTES	9 BITS TO THE BYTE SINCE YOU'RE SUCH A GOOD CUSTOMER

Numerical data

- Numerical data is mainly used within scientific context in the modern computers… all the other data types are converted to numerical data to be elaborated

- Every try for elaborating non numerical data failed or was more inefficient than converting data to numerical data and then elaborate them.

How do we count?

$$252 = 2 \times 100 + 5 \times 10 \; + 2 \times 1$$
$$= 2 \times 10^2 \; + 5 \times 10^1 + 2 \times 10^0$$

The numbering system of the western world (*Hindu-Arabic numerical system*) is:

- decimal
- positional

Numeral systems

Non positional:

- Roman numerals (ex. V, L, D)
- Arithmetic operations are difficult
(ex. V + V = X)

Positional:

- Arabic (decimal)
- Mayan (Base 20)

Hybrid

- Chinese
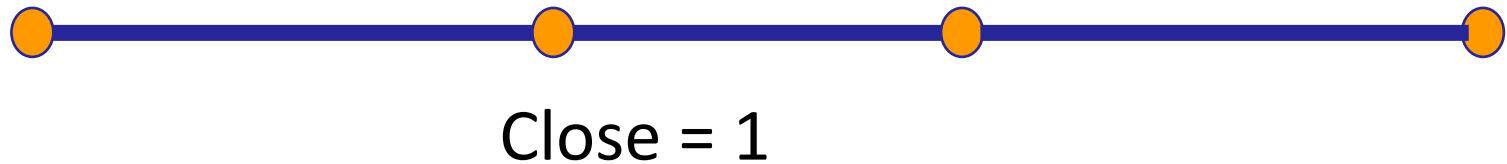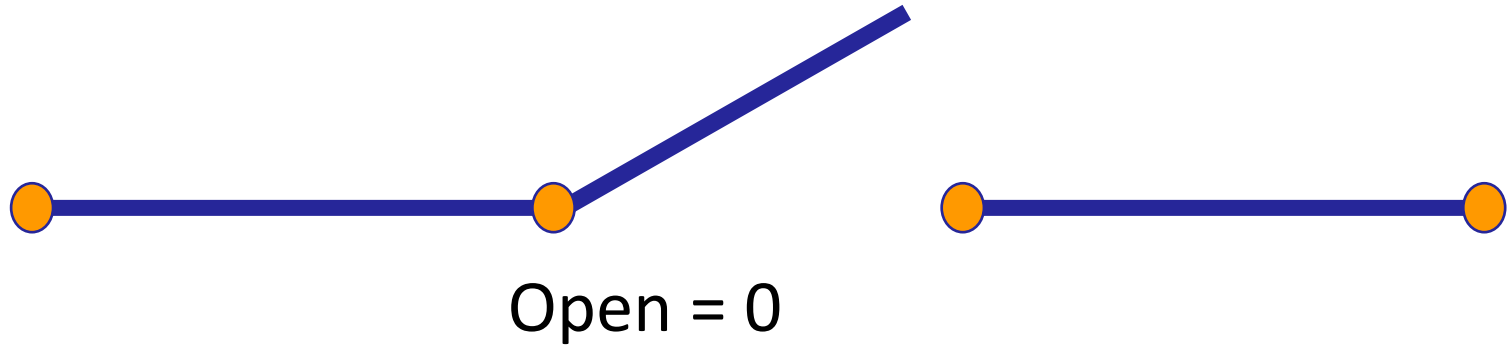
Positional numbering system in base B

Characteristics:

- Digits: { 0, 1, 2, ..., B-1 }
- Weight of i-th digit: B^i
- Representation (natural numbers) on N digits

$$A = \sum_{i=0}^{N-1} a_i \cdot B^i$$

Bit and switches

A switch has two states
(Open/Close, OFF/ON, FALSE/TRUE)

Open = 0

Close = 1

Binary system

Base = 2

Digits= { 0, 1 }

Example:

$101_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$\quad\quad = 1 \times 4 + 0 \quad\quad + 1 \times 1$

$\quad\quad = 5_{10}$

$111_2 = 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$\quad\quad = 1 \times 4 + 1 \times 2 + 1 \times 1$

$\quad\quad = 7_{10}$

Other binary numbers

0	...	0	1000	...	8
1	...	1	1001	...	9
10	...	2	1010	...	10
11	...	3	1011	...	11
100	...	4	1100	...	12
101	...	5	1101	...	13
110	...	6	1110	...	14
111	...	7	1111	...	15

Terminology

BIT (BInary digiT)

0	1

BYTE = eight bits

00110110

WORD = n bytes

00001111
10101010

Terminology

1 0 1 1 0 1 1 0

MSb   LSb

Most   Least
Significant   Significant
bit   bit
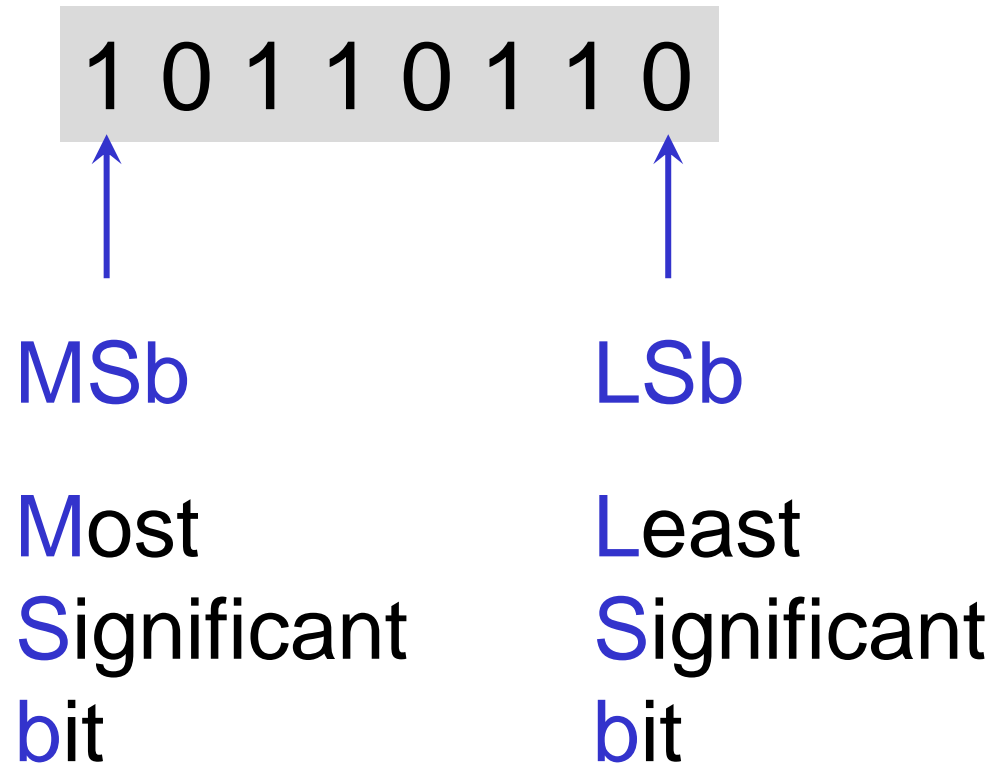
Some powers of 2

2^0 = 1 2^{11} = 2,048

2^1 = 2 2^{12} = 4,096

2^2 = 4 2^{13} = 8,192

2^3 = 8 2^{14} = 16,384

2^4 = 16 2^{15} = 32,768

2^5 = 32 2^{16} = 65,536

2^6 = 64 2^{17} = 131,072

2^7 = 128 2^{18} = 262,144

2^8 = 256 2^{19} = 524,288

2^9 = 512 2^{20} = 1,048,576

2^{10} = 1,024 2^{30} = 1,073,741,824

Some powers of 2

$2^0 =$ 1 $2^{11} =$ 2,048

$2^3 =$ 8

$2^8 =$ 256

$2^9 =$ 512

$2^{10} =$ 1,024   → 1Ki / 1k

$2^{20} =$ 1,048,576   → 1Mi / 1M

$2^{30} =$ 1,073,741,824   → 1Gi / 1G

2^{10} =1,024 2^{30} = 1,073,741,824

Binary to decimal conversion

Apply the definition of the weighted sum of the binary numbers:

$$1101_2$$

$$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$= 8 + 4 + 0 + 1$$
$$= 13_{10}$$

Decimal to binary conversion

Algorithm for finding the binary representation of a positive integer:

1. **Divide by 2** the original value and record the **remainder**
2. If the **quotient** is **not zero**, continue to divide the new quotient by 2 and record the remainder
3. Once the **quotient equals 0**, the binary value consists of the **remainders** listed from right to left in the order they were recorded.

13	6	3	1	0	*quotients*
	1	0	1	1	*remainders*

$$13_{10} \; = \; 1101_2$$

Limits of the binary system (natural representation)

Considering natural numbers of N bits:
- 1 bit ~ 2 number ~ $\{\ 0,\ 1\ \}_2$ ~ $[\ 0\ \dots\ 1\ ]_{10}$
- 2 bit ~ 4 number ~ $\{\ 00,\ 01,\ 10,\ 11\}_2$ ~ $[0\dots3]_{10}$

Thus, in general for natural numbers of N bits
- distinct combinations

$$2^N$$

- Values intervals

$$0 \leq x \leq 2^N - 1 \qquad [\ \text{base 10}\ ]$$
$$(000\dots0) \leq x \leq (111\dots1) \qquad [\ \text{base 2}\ ]$$

Limits of the binary system (natural representation)

bit	symbols	min_{10}	max_{10}
4	16	0	15
8	256	0	255
10	1,024	0	1,023
16	65,536	0	65,535
32	4,294,967,296	0	4,294,967,295

Binary addition

Basic rules :

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 0 \quad (\text{carry} = 1)$$

Addition of binary numbers

The partial sum between the bits having the same weight is computed, including the propagation of the carry:

$$
\begin{array}{cccc}
1 & 1 & & \\
0 & 1 & 1 & 0 \quad + \\
0 & 1 & 1 & 1 \quad = \\
\hline
1 & 1 & 0 & 1 \\
\end{array}
$$

Binary Subtraction

- Basic rules:

$$0 - 0 = 0$$

$$0 - 1 = 1 \qquad \text{(borrow = 1)}$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

Binary Subtraction

The partial subtraction between the bits having the same weight is computed, including the propagation of the borrow:

$$
\begin{array}{r}
{\color{red}1\ 1} \\
1\ 1\ 0\ 1 \\
-\ 0\ 1\ 1\ 0 \\
\hline
0\ 1\ 1\ 1
\end{array}
$$

Overflow

- Term **overflow** indicates the *error* that occurs, in an automatic calculation system, when the result of an operation *can not be represented* with the *same code* and *number of bits* of the operands.

Overflow

- In pure binary addition, overflow appears when:
  - Working with a fixed number of bits
  - There is a carry on the MSb

Overflow - example

- *Assumption*: Operation on numbers represented using only **4 bits**,

  in plain binary:

  $$\begin{array}{r} \textbf{0101} + \\ \textbf{1110} \\ \hline \end{array}$$

  overflow $\rightarrow$ **1**0011

Octal system

- base = 8
  (Sometimes indicated with Q as octal)
- digits = { 0, 1, 2, 3, 4, 5, 6, 7 }
- Useful to write binary numbers in compact form (3:1)

$$1\ 0\ 1\ 1\ 1\ 0\ 0\ 1_2$$

$$2\quad 7\quad 1_8$$

Hexadecimal system

- base = 16
  (Sometimes indicated with H for Hexadecimal)
- digits= { 0, 1, ..., 9, A, B, C, D, E, F }
- Useful to write binary numbers in compact form (4:1)

$$\underbrace{1\ 0\ 1\ 1}_{B}\ \underbrace{1\ 0\ 0\ 1}_{9}{}_{2}{}_{16}$$

Realtive numbers representation

+ 25 °C

- 9 °C

Signed numbers

- A number can be:
  - positive (+)
  - negative (−)

- Representing the sign in binary is easy, but…

  the simpler solution is not always the best solution!
  - Sign and Magnitude (S&M)
  - One's Complement (1C)
  - Two's Complement (2C)

Sign and magnitude representation

- One bit for the sign (usually the MSb):
  - 0 = positive sign (+)
  - 1 = negative sign (−)
- N-1 bits for the *absolute value* (also called *magnitude*)

Sign	*Magnitude*
1 bit	N-1 bit

Sign and Magnitude: Examples

Using a 3 bit representation :

$+ 3_{10} \rightarrow \quad 011_{S\&M}$

$- 3_{10} \rightarrow \quad 111_{S\&M}$

$000_{S\&M} \rightarrow \quad + 0_{10}$

$100_{S\&M} \rightarrow \quad - 0_{10}$

binary	Decimal
000	0
001	1
010	2
011	3
100	-0
101	-1
110	-2
111	-3

Sign and Magnitude issues

Issues:

- Double representation of zero (+ 0, - 0)

- Complexity of operations

  - Example: A+B

	A > 0	A < 0
B > 0	A + B	B - |A|
B < 0	A - |B|	- (|A| + |B|)

Sign and Magnitude: Limits

In a S&M representation on N bit:

$$- (2^{N-1} - 1) \leq x \leq + (2^{N-1} - 1)$$

Example:
- 8 bit   =   [-127 ... +127]
- 16 bit   =   [-32,767 ... +32,767]

One's Complement

Given a binary number A of N bits, the one's complement can be defined as :

$$\overline{A} = (2^N - 1) - A$$

Simply called *complement.*

It is usually indicated with an horizontal line upon the number or with the apostrophe (A').

One's Complement

Practical Rule:

One's complement of a binary number A can be obtained by complementing (i.e., inverting) all of its bits

Example:

$$A = 1011 \quad \rightarrow \quad \overline{A} = 0100$$

One's Complement: Example

Using a 3 bit representation :

$+ 3_{10} \rightarrow$ $\quad\quad$ 011_{1C}

$- 3_{10} \rightarrow$ $\quad\quad$ 100_{1C}

$000_{1C} \rightarrow$ $\quad\quad$ $+ 0_{10}$

$111_{1C} \rightarrow$ $\quad\quad$ $- 0_{10}$

binary	Decimal
000	0
001	1
010	2
011	3
100	-3
101	-2
110	-1
111	-0

Two's Complement

Given a binary number A of N bits, two's complement (2C) can be defined as :

$$\overline{\overline{A}} = 2^N - A = \overline{A} + 1$$

Two's Complement

Practical Rule:

Two's complement of a binary number A can be obtained by adding one to its one's complement

Example:

$$A = 1011 \rightarrow \overline{A} = 0100 \rightarrow \overline{\overline{A}} = \overline{A} + 1 = 0101$$

Two's Complement (bis)

Practical Rule (b):

The two's complement of a binary number is obtained starting from the LSb and copying all the bits up to the first "1" and then complementing the remaining bits.

Example:

$$A = 10110 \quad \rightarrow \quad \overline{\overline{A}} = 01010$$

Two's Complement: Example

Using a 3 bit representation :

$+\ 3_{10} \rightarrow$ $011_{S\&M}$

$-\ 4_{10} \rightarrow$ $100_{S\&M}$

$000_{2C} \rightarrow$ $+\ 0_{10}$

$100_{2C} \rightarrow$ $-\ 4_{10}$

binary	Decimal
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Encoding a value in two's complement

Algorithm encoding a **negative** number in 2C:
1. Get the **positive pure binary value** using the **predefined bits**
2. **Complement** the number (**1C**)
3. **Add 1**.

EX: transform the decimal number -**13**$_{10}$ to binary using the 2C representation on 5 bits.

$\qquad$ 1. 13_{10} = 01101
$\qquad$ 2. 1C = 10010
$\qquad$ 3. Add 1: 10010 + 1

$\qquad\qquad$ - 13_{10} = **10011** 2C

Getting the decimal number from a value in two's complement

Given a negative number in **2C**, then:
1. **Complement** the number (**1C**)
2. **Add 1**
3. Convert the binary number to decimal.

EX: which is the decimal number represented on 5 bits by:
   10011 (2C)

        1. 1C = 01100
        2. Add 1: 01100 + 1 = 01101
        3. $01101_2 = 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13$
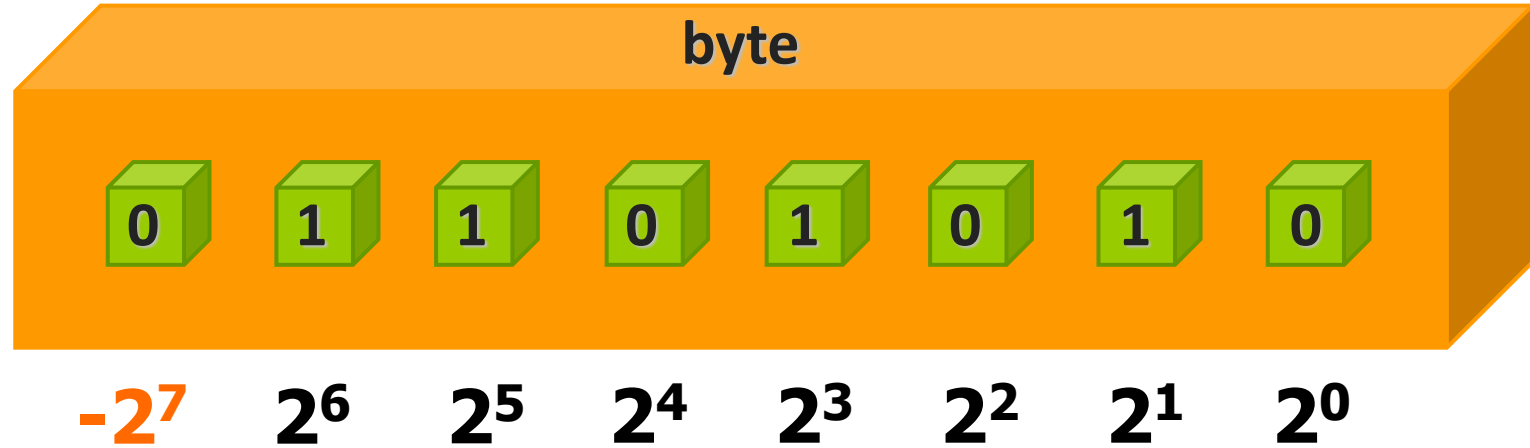   then, **10011** (2C) = $- 13_{10}$

Encoding a value in two's complement -2

- in a two's complement (2C) number of N bits:
  - the MSb has a negative weight (-2^{N-1})
  - the rest of the bits have a positive weight according to its position

- in this way, the MSb determines the number sign:
  - $0 = +$ $1 = -$

- examples (two's complement numbers on 4 bits):
  - $1000_{2C} = -2^3 = -8_{10}$
  - $1111_{2C} = -2^3 + 2^2 + 2^1 + 2^0 = -8 +4 +2 +1 = -1_{10}$
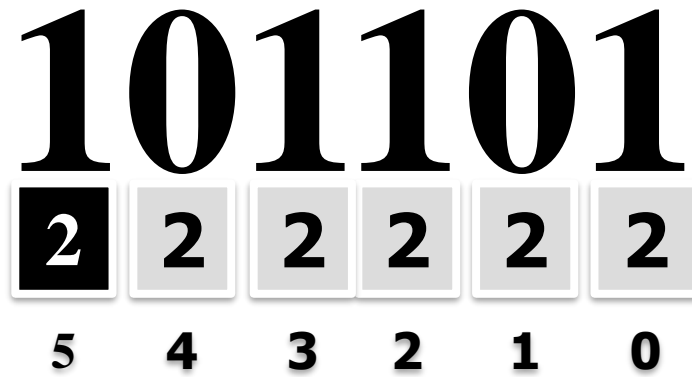 - $0111_{2C} = 2^2 + 2^1 + 2^0 = 7_{10}$

Two's complement

byte

| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |

-2^7 $\quad$ 2^6 $\quad$ 2^5 $\quad$ 2^4 $\quad$ 2^3 $\quad$ 2^2 $\quad$ 2^1 $\quad$ 2^0

Two's complement (example)

$$101101$$

2	2	2	2	2	2
5	4	3	2	1	0

-32+8+4+1= -19

Two's Complement

- The two's complement (2C) representation is the most adopted in today's processors, since simplifies the circuit implementation for the basic arithmetic operations

- It is possible to apply the binary rules to all of the bits composing the number

Addition and Subtraction in Two's Complement

- The addition and subtraction are done directly, without considering the signs of the operands

$$\bullet A_{2C} + B_{2C} \rightarrow A_{2C} + B_{2C}$$
$$\bullet A_{2C} - B_{2C} \rightarrow A_{2C} - B_{2C}$$

Addition and Subtraction in Two's Complement

• subtraction is done by adding the two's complement (2C) of subtrahend to minuend

$$A_{2C} - B_{2C} \rightarrow A_{2C} + \overline{\overline{B_{2C}}}$$

(note: during the subtraction B is already encoded in 2C but applies the 2C transformation)

Addition in two's complement - example

00100110 + 11001011

```
00100110 +
11001011 =
--------------
11110001
```

verify: 38 + (− 53) = − 15

Subtraction in two's complement - example

00100110 – 11001011

00100110 +
00110101 =

01011011

verify: 38 – (− 53) = 91

Overflow two's complement addition

Operands with different signs: overflow never occurs

Operands with the same sign: Overflow occurs if the result has different sign

In any case, carry is always neglected on MSb

Note: it is not needed to define the overflow for the subtraction in two's complement because it is always computed with an addition.

Overflow in two's complement

```
0101 +                    1110 +
0100 =                    1101 =
---------                 ----------
1001                      11011 =
                           1011
```

overflow!   *OK*

(5 + 4 = 9)   (-2 + -3 = -5)

Impossible in 2C on 4 bits

Overflow two's complement subtraction

3 − (-7)	-1 − (7)
0011 +	1111 +
0111 =	1001 =
----------	----------
1010	11000 =
	1000
overflow!	*OK* (-8)

+10 is not possible in
   2C on 4 bits

Real numbers representation (Floating Point)

Scientific notation

3.14	$0.314 \times 10^{+1}$
0.0001	0.1×10^{-3}
137	$0.137 \times 10^{+3}$

S = sign

M = mantissa

E = exponent
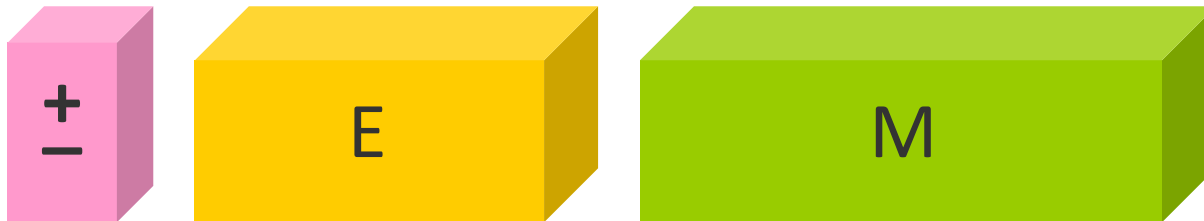
$$N = \pm\ M \times 2^{E}$$

Real numbers representation (Floating Point)

The computer system stores the following elements:
- Sign
- Exponent
- Mantissa

IEEE-754 Format

- Mantissa in the form "1. ……" (max value < 2)
- Exponent base is 2

https://float.exposed/

- IEEE 754 (SP) single precision, 32 bits:

sign	exponent	mantissa
1 bit	8 bit	23 bit

- IEEE 754 (DP) double precision, 64 bit:

sign	exponent	mantissa
1 bit	11 bit	52 bit

IEEE-754 SP: range of values

underflow (0)

-10^{38} -10^{-38} 0 10^{-38} 10^{38}
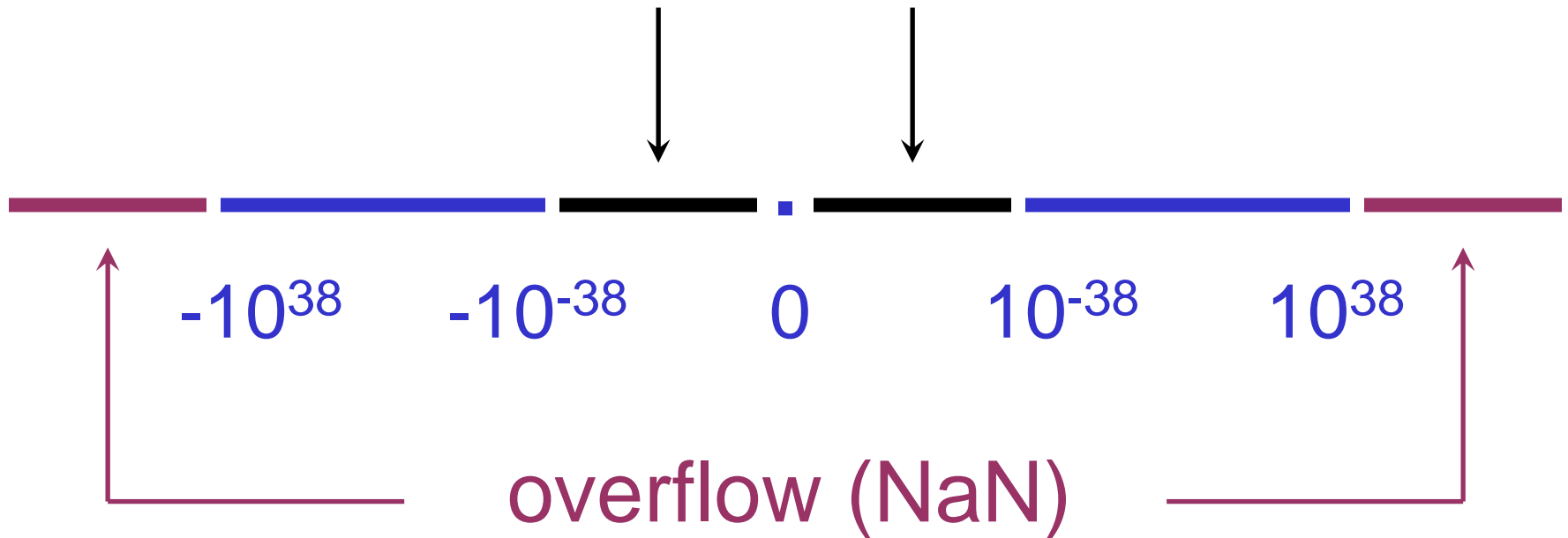
overflow (NaN)

Problems

- Fix number of encoding bits
- Numbers are represented as sequences of digits
- Problems:
  - Representation interval
  - Overflow
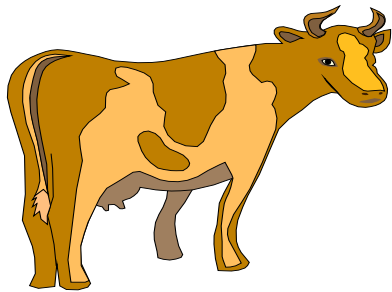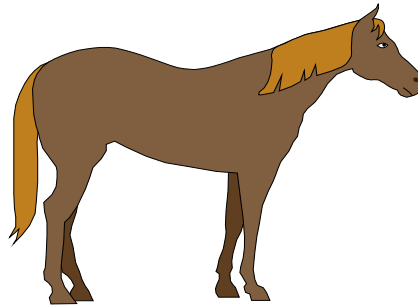 - Precision

Elaborating non-numerical information

Nel mezzo del cammin di nostra vita ...

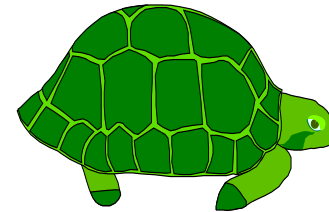# Non-numerical information

If in a finite quantity, you can put in correspondence with integer numbers.

00   01   10

Numerical data representation

- Assuming a data composed of N bits…
  - It is possible to encode 2^N different elements
  - Useful for different numerical representations

- Example (using 3 bits):

Binariy elements	000	001	010	011	100	101	110	111
Natural numbers	0	1	2	3	4	5	6	7
Relative num. (S&M)	+0	+1	+2	+3	−0	−1	−2	−3
Relative num.(1C)	+0	+1	+2	+3	−3	−2	−1	−0
Relative num.(2C)	+0	+1	+2	+3	−4	−3	−2	−1

Characters

- There exists standard encoding, since it is more often used information:
  - ASCII (American Standard Code for Information Interchange)
  - EBCDIC (Extended BCD Interchange Code)
  - UNICODE

ASCII Code

- Used also in telecommunications
- Uses 7 bit (8 bit) for the representation:
  - 52 alphabetic characters (a...z, A...Z)
  - 10 digits (0...9)
  - Punctuation symbols (,;!?...)
  - Control characters

```
CR      ( 13 )          Carriage Return
LF,NL   ( 10 )          New Line, Line Feed
FF,NP   ( 12 )          New Page, Form Feed
HT      ( 9 )           Horizontal Tab
VT      ( 11 )          Vertical Tab

NUL     ( 0 )           Null
BEL     ( 7 )           Bell
EOT     ( 4 )           End-Of-Transmission

. . . . . . . . . .
```

ASCII Code (Cont.)

01000001	A	00100000	
01110101	u	01110100	t
01100111	g	01110101	u
01110101	u	01110100	t
01110010	r	01110100	t
01101001	i	01101001	i
00100000		00100001	!
01100001	a		

NEWLY DISCOVERED BINARY LOVE LETTER FROM BILL GATES TO HIS HIGH SCHOOL SWEETHEART

Extended ASCII Code

- The characters with ASCII code higher than 127 are not standard.

- In practice, the characters with MSb=1 are used for:
  - Control characters
  - Local alphabets (example: ñ à á ä ã æ)
  - Graphic characters (example: § ¨ ♥ ♠ $\propto$ $\Leftrightarrow$ $\sqrt{}$)

UNICODE and UTF-8

Unicode expresses all the chars of all the world languages (more than one million).

UTF-8 is the most used Unicode:

- 1 byte for US-ASCII chars(MSb=0)
- 2 byte for Latin chars with diacritical symbols, Greek, Cyrillic, Armenian, Jewish, Arabic, Syrian e Maldivian
- 3 byte for other commonly used languages
- 4 byte very rare chars
- Recommended by IETF for e-mails

Representing a text in ASCII format

- Characters codified in ASCII code

- Every line terminates with the "line terminator" character:
    - MS-DOS and Windows = CR + LF
    - UNIX = LF
    - MacOS = CR

- Pages sometimes separate by FF.

PDF

- Portable Document Format (PDF) is an open standard for document exchange. This file format, created by Adobe Systems in 1993, is used for representing documents in a manner independent of application software, hardware, and operating systems.

- Each PDF file encapsulates a complete description of a fixed-layout flat document, including the text, fonts, graphics, and other information needed to display it.