# E2011: Theoretical fundamentals of computer science
# Topic 3: Numeral systems

Vlad Popovici, Ph.D.

Fac. of Science - RECETOX

# Outline


Figure: Numerals - from Wikipedia

# Introduction

Electronic computers/calculators:

- analogic computers
- *digital computers*
- hybrid computers

# Introduction

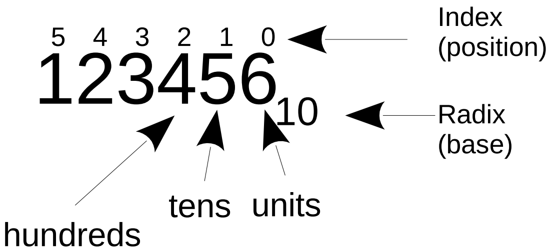
Electronic computers/calculators:

- analogic computers
- *digital computers*
- hybrid computers


Figure: An analogic computer - oscilloscope

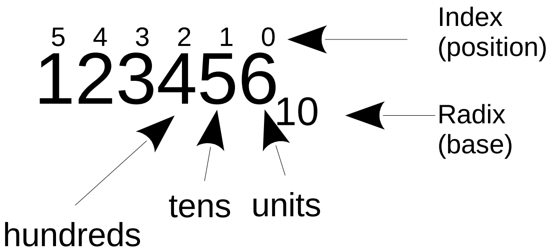
# Positional notation

Can be traced back to the work of Archimedes (3rd century BC).
Only in 12th century, the decimal notation was introduced in Europe (Fibonacci).



# Positional notation

Can be traced back to the work of Archimedes (3rd century BC).
Only in 12th century, the decimal notation was introduced in Europe (Fibonacci).




$$123456_{10} = 1 \times 10^5 + 2 \times 10^4 + 3 \times 10^3 + 4 \times 10^2 + 5 \times 10^1 + 6 \times 10^0$$

How do we extract the digits from a number (radix 10)?

123456

Divide by 10:

quotient

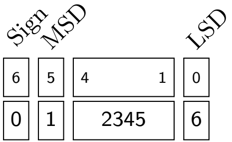
12345

remainder

6

# Representation

EXAMPLE (123456)




Sign: if present, whether it is a positive or negative integer
MSD: most significant digit
LSD: least significant digit

# Binary systems (Base-2)

$$
\begin{aligned}
&1010011010_2 = \\
&= 1 \times 2^9 + 0 \times 2^8 + 1 \times 2^7 + 0 \times 2^6 \\
&+ 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\
&= 512 + 128 + 16 + 8 + 2 \\
&= \mathbf{666_{10}}
\end{aligned}
$$


TRANSLATION: 666

- digits (base-10) $\longleftrightarrow$ bits (base-2)
- *kilobit (Kb)* $= 1000 = 10^3$ bits
- *megabit (Mb)* $= 1000\text{kb} = 10^6$ bits
- *giga*, *tera*, *peta*, *exa*, *zetta*,...
- *kibibit* $= 1024 = 2^{10}$ bits
- *mebibit* $= 1024\text{Kibit} = 2^{20}$ bits

- **8 bits = 1 byte**
- 1 KB $=$ 1024 bytes $= 2^{10}$ bytes $=$ 8192 bits $\neq$ 1 Kb
- 1 MB $=$ 1024 KB $= 2^{20}$ bytes
- GB, PB, TB, EB...

From decimal to binary: $42_{10} =?_2$

$$
\begin{aligned}
42/2 &\longrightarrow \text{quotient: } 21 \text{ remainder: } 0\\
21/2 &\longrightarrow \text{quotient: } 10 \text{ remainder: } 1\\
10/2 &\longrightarrow \text{quotient: } 5 \text{ remainder: } 0\\
5/2 &\longrightarrow \text{quotient: } 2 \text{ remainder: } 1\\
2/2 &\longrightarrow \text{quotient: } 1 \text{ remainder: } 0\\
1/2 &\longrightarrow \text{quotient: } 0 \text{ remainder: } 1
\end{aligned}
$$

...and read from last to first remainder:

$$42_{10} = 101010_2$$

...or quicker (for small numbers):

$$\begin{aligned} 42_{10} &= 32 + 8 + 2 \\ &= 2^5 + 2^3 + 2^1 \\ &= 100000_2 + 1000_2 + 10_2 = \\ &= 101010_2 \end{aligned}$$

$00000_2 = 0_{10}$

$00001_2 = 1_{10}$

$00010_2 = 2_{10}$

$00011_2 = 3_{10}$

$00100_2 = 4_{10}$

$00101_2 = 5_{10}$

$00110_2 = 6_{10}$

$00111_2 = 7_{10}$

$01000_2 = 8_{10}$

$01001_2 = 9_{10}$

$01010_2 = 10_{10}$

$01011_2 = 11_{10}$

$01100_2 = 12_{10}$

$01101_2 = 13_{10}$

$01110_2 = 14_{10}$

$01111_2 = 15_{10}$

$10000_2 = 16_{10}$

# Important properties

- with $n$ bits, maximum number representable is

$$2^0 + 2^1 + 2^2 + \cdots + 2^{n-1} = 2^n - 1$$

- it follows that data is represented with limited precision

# Bits, bytes, words...

- byte (8 bits) is the basic data unit
- 1 byte is used to represent basic characters (ASCII)
- 2 bytes are used for extended/international caracters (Unicode)
- 1 integer value may be represented on 2/4/8/16 bytes: defines the "word"-size for a given computer → depends on the *architecture*
- short- and long-words have half-/double- the size of the word
- there are also "doublewords", "quadwords"

# Examples

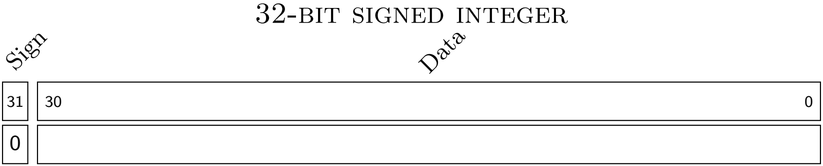
## ASCII

- American Standard Code for Information Interchange.

Characters are represented on 8 bits (1 byte):

- 'A': $65_{10} = 41_{16} = 0100\ 0001_2$, 'B': $0100\ 0010_2$,...
- '0': $48_{10} = 30_{16} = 0011\ 0000_2$,...
- etc

# Examples

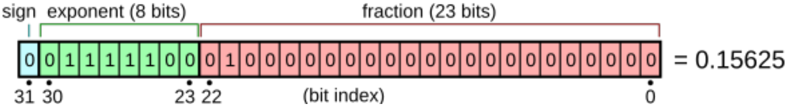



- if "Sign" is reserved, the range is $-2^{31}-1$ to $2^{31}-1$, i.e. $-2,147,483,647$ to $2,147,483,647$
- if "Sign" is not reserved, the range is 0 to $2^{32}-1$, i.e. 0 to $4,294,967,295$

# Examples

## IEEE 754

- technical standard for floating point arithmetic




$$(-1)^{b_{31}} \times 2^{(b_{30} \ldots b_{23})_2 - 127} \times (1.b_{22} \ldots b_0)_2$$

$$= (-1)^{\text{sign}} \times 2^{(E-127)} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right)$$

# Additions in base-2

$$13 + 23 = ?$$

| | | | | | | |
|---|---|---|---|---|---|---|
| carry: | **1** | **1** | **1** | **1** | **1** | |
| | | 0 | 1 | 1 | 0 | 1 |
| + | | 1 | 0 | 1 | 1 | 1 |
| = | 1 | 0 | 0 | 1 | 0 | 0 |

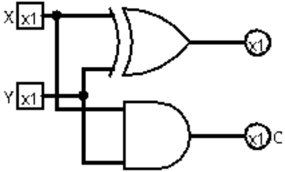
# Binary arithmetic and logical circuits

Example: single-bit adder:

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 0 \text{ carry: } 1$$

| x | y | sum | carry |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

$$\text{sum} = x \oplus y \text{ (XOR)}$$
$$\text{carry} = x \cdot y$$



# Bitwise operations

Bitwise operations manipulate strings of bits:

- *bitwise-NOT*: for example, on 4 bits: $\text{NOT}\,7 = 8$ $(\text{NOT}\{0111\} = 1000)$
- *bitwise-AND*: example: $1010\&0111 = 0010$
- *bitwise-OR*, *bitwise-XOR*, etc.
- test if a number is even/odd: check whether the least significant bit (index 0) is 0/1

# Bitwise operations

- *logical shift*: insert 0s to the left (shift right) or to the right (shift left). Example (on 4 bits):

$$1011 << 1 = 0110 \text{ left shift with one position}$$
$$1011 >> 1 = 0101 \text{ right shift with one position}$$

- left shift with one position is equivalent to a multiplication by two
- right shift with one position is equivalent to a (integer) division by two

# Bitwise operations

- *arithmetic shift*: insert 0s to the right (shift left) and duplicate the most significant bit at left (shift right). Example (on 4 bits):

$$1011 <<< 1 = 0110 \text{ left shift with one position}$$
$$1011 >>> 1 = 1101 \text{ right shift with one position}$$

# Hexadecimal system (Base-16)

- need more symbols for hexa-digits: A, B,...,F

$$00000_2 = 0_{10} = 0_{16}$$
$$00001_2 = 1_{10} = 1_{16}$$
$$00010_2 = 2_{10} = 2_{16}$$
$$00011_2 = 3_{10} = 3_{16}$$
$$00100_2 = 4_{10} = 4_{16}$$
$$00101_2 = 5_{10} = 5_{16}$$
$$00110_2 = 6_{10} = 6_{16}$$
$$00111_2 = 7_{10} = 7_{16}$$

$$01000_2 = 8_{10} = 8_{16}$$
$$01001_2 = 9_{10} = 9_{16}$$
$$01010_2 = 10_{10} = A_{16}$$
$$01011_2 = 11_{10} = B_{16}$$
$$01100_2 = 12_{10} = C_{16}$$
$$01101_2 = 13_{10} = D_{16}$$
$$01110_2 = 14_{10} = E_{16}$$
$$01111_2 = 15_{10} = F_{16}$$
$$10000_2 = 16_{10} = 10_{16}$$

- 4 bits correspond to 1 hexa-digit
- most of the time, we use hexa notation as it is more compact
- in many cases, hexa strings/number are prefixed by **0x** to make clear their meaning
- example: HTML color specification R,G,B: $\#AFA077$:

$$R = 0xAF = 1010\ 1111_2 = 10 \times 16 + 15 = 175$$
$$G = 0xA0 = 1010\ 0000_2 = 160$$
$$B = 0x77 = 0111\ 0111_2 = 119$$

# Questions?