# COMPUTER ARCHITECTURE AND SOFTWARE EXECUTION PROCESS

## DATA REPRESENTATION

Bachelor in Artificial Intelligence, Data and Management Sciences
CentraleSupelec and ESSEC Business School - 2024/2025

**Idir AIT SADOUNE**
idir.aitsadoune@centralesupelec.fr

# PROCESSOR ARITHMETIC

- A **processor** is made of several logic circuits (more details in Chapter $4$).
- A logic circuit uses only two values $\rightarrow 0$ and $1$ (**bits**).
- All information handled by a processor must be encoded in **binary**.
  - integer numbers, real numbers, texts, pictures, ...
- For **natural numbers** $\rightarrow$ classic binary representation (see Chapter $1$).
- **Encoding** is required for other data types.

# OUTLINE



Back to the outline - Back to the begin

# OUTLINE



Back to the outline - Back to the begin

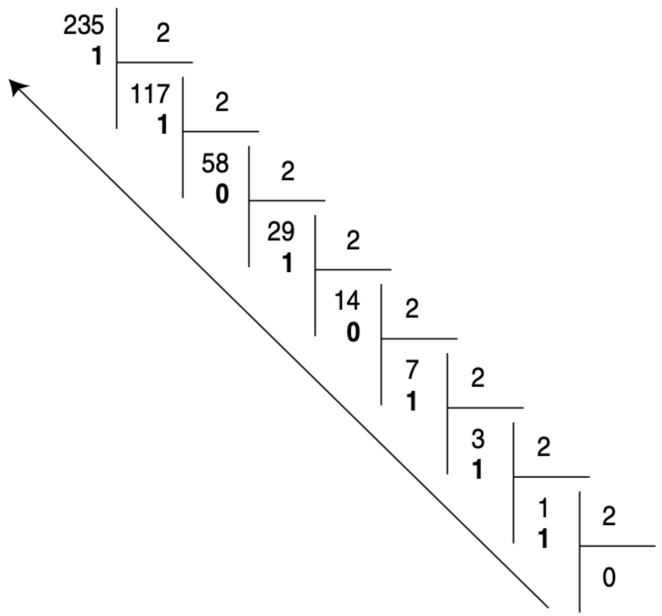
# FINITE PRECISION ARITHMETIC

- The processor only manipulates numbers with a **fixed representation size** $\rightarrow 1, 2, 4$ or $8$ bytes.
  - **Example**: Integer $\rightarrow 4$ bytes ($4 \times 8$ bits $= 32$ bits $\rightarrow 2^{32}$ different values)
- This means all data types have a **finite domain** $\rightarrow$ possible **overflows** in programming languages.
  - **Max. value** $\rightarrow 2147483647$
  - **Min. value** $\rightarrow -2147483648$
  - $2147483647 + 1 = ?$ ($= -2147483648$ next slides)

# NATURAL NUMBERS

## CLASSIC BINARY REPRESENTATION

- Generally, we use the binary representation, as detailed in **Chapter 1**.
- **Example** $\rightarrow$ case of using $32$-bit encoding ($235_{10}$)




  - We convert to binary and complete with zeros on the left to reach 32 bits.
  - $235_{10} = 00000000\ 00000000\ 00000000\ 11101011_2$

# NATURAL NUMBERS

## BINARY-CODED DECIMAL (BCD)

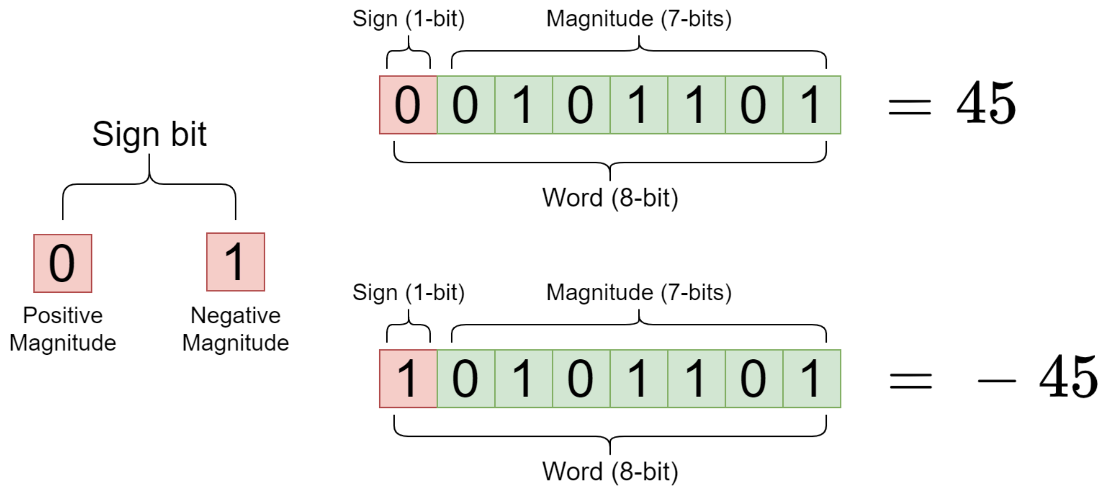
- **BCD** is a **class of binary encodings** of decimal numbers where each digit is represented by a fixed number of bits ($4$ or $8$ bits).
- **Example** $\rightarrow$ case of using $32$-bit encoding (each digit is represented by $8$ bits)
  $214 = \ ?$

| 0 | 2 | 1 | 4 |
|---|---|---|---|
| 00000000 | 00000010 | 00000001 | 00000100 |

  - $214_{10} = 00000000\ 00000010\ 00000001\ 00000100_{BCD}$

# INTEGER NUMBERS

## SIGNED NUMBER REPRESENTATION




- **A signed number** is represented by the bit pattern corresponding to
  - the sign of the number for the sign bit (the most significant bit) $\longrightarrow$ (set to $0$ for a positive number and to $1$ for a negative number)
  - the magnitude of the number (or absolute value) for the remaining bits.

✘ Two representations of $0$ ($00000000 = 10000000$)

✘ Arithmetic operations **cannot be implemented** in electronic circuits.

# INTEGER NUMBERS

## ONES' COMPLEMENT

**To represent -34 in 1's complement form**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| + 34 | = | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| | | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ | ↓ |
| - 34 | = | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |

**(1's complement of + 34)**

- **Positive numbers** are the same simple, binary system used by sign-magnitude.
- **Negative values** are the bit complement of the corresponding positive value.

✘ Like sign-magnitude representation, ones' complement has two representations of 0 (00000000 = 11111111)

✘ Arithmetic operations are **not intuitive**.

# INTEGER NUMBERS

## TWO'S COMPLEMENT

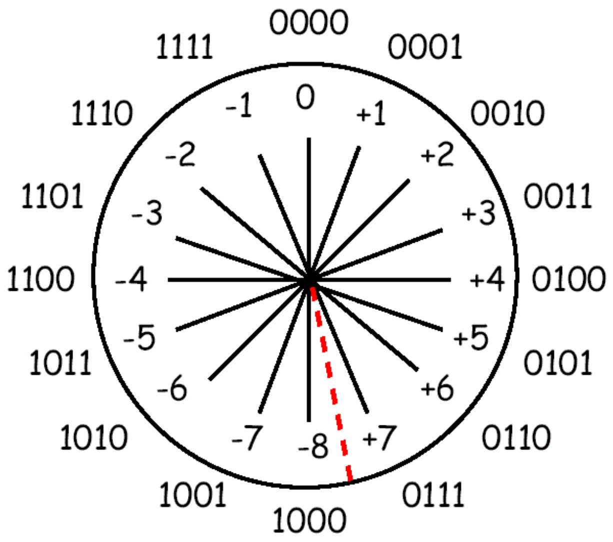
**To represent -34 in 2's complement form**

**+ 34 = 0 0 1 0 0 0 1 0**

**1 1 0 1 1 1 0 1** **(1's complement of + 34)**

**+ 1**

**- 34 = 1 1 0 1 1 1 1 0** **(2's complement of + 34)**



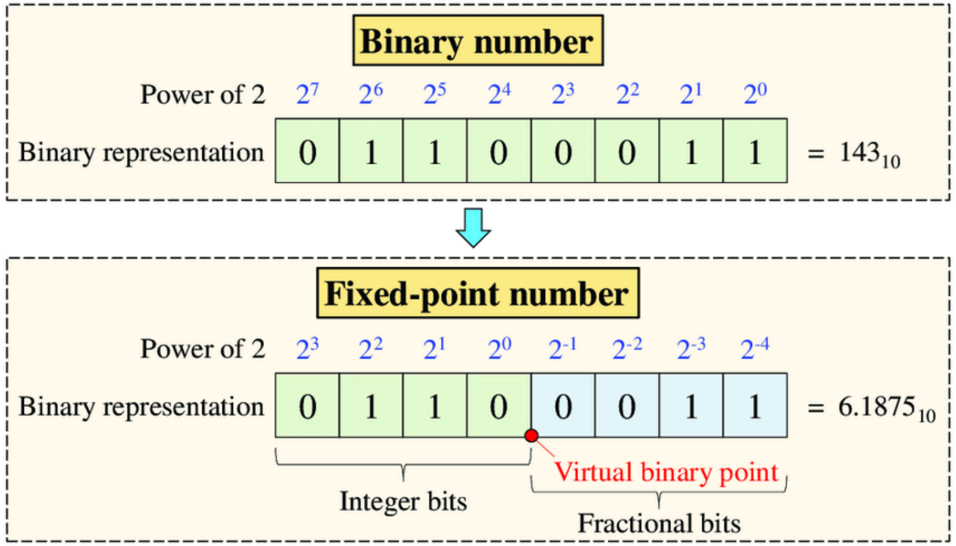

- The **two's complement** of an integer is computed by
  1. starting with the binary representation of the corresponding positive value;
  2. inverting all bits - changing every 0 to 1, and every 1 to 0;
  3. adding 1 to the entire inverted number, ignoring any overflow

✔ Only one representation for 0

# REAL NUMBERS

## FIXED-POINT ARITHMETIC

- **A fixed-point** representation of a fractional number is essentially an integer that is to be implicitly multiplied by a **fixed scaling factor**.




- $6.1875_{10} = 0110.0011_2$ (with 8 as a fixed scaling factor)
  - $2^2 + 2^1 = 4 + 2 = 6$
  - $2^{-3} + 2^{-4} = 0.125 + 0.0625 = 0.1875$

# OUTLINE



Back to the outline - Back to the begin

# REAL NUMBERS

## FLOATING-POINT ARITHMETIC

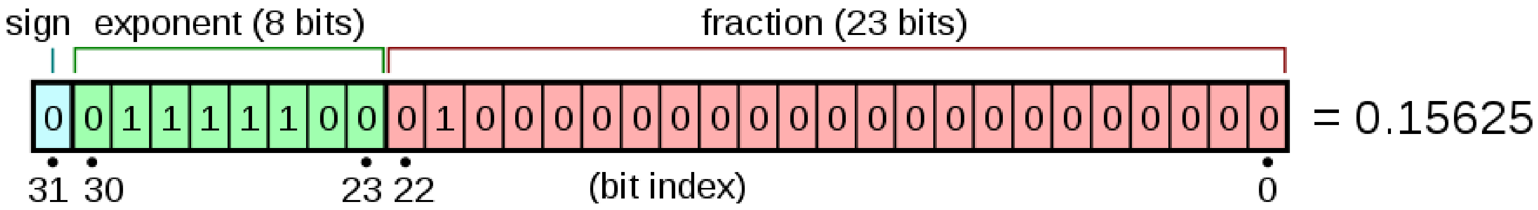
- Arithmetic that represents **real numbers** using an integer with a fixed precision (significand), scaled by an integer exponent of a fixed base.

$$x = 3.14159265359 = \underbrace{314159265359}_{\text{significand}} \times \underbrace{10}_{\text{base}}{}^{\overbrace{-11}^{\text{exponent}}}$$

# FLOATING-POINT ARITHMETIC

## IEEE 754 STANDARD (BINARY32)

- Sign bit $\rightarrow 1$ bit
- Exponent part $\rightarrow 8$ bits ($exponent + 127$)
- Significand precision $\rightarrow 24$ bits ($23$ explicitly stored)




$$(-1)^{sign} \times 2^{exponent-127} \times 1.fraction$$

- $sign = 0$
- $exponent = 01111100_2 = 124$
- $fraction = 2^{-2} = 0.25$
- $(-1)^0 \times 2^{124-127} \times 1.25 = 2^{-3} \times 1.25 = 0.15625$

# FLOATING-POINT ARITHMETIC

## IEEE 754 STANDARD (BINARY32)

$$0.15625 = ?$$

- We need a **normal form** $(-1)^{sign} \times 2^{exponent-127} \times 1.fraction$
- $0.15625 \times 2 = 0.3125$
- $0.3125 \times 2 = 0.625$
- $0.625 \times 2 = 1.25$
- $0.15625 = 1.25 \times 2^{-3}$
  - ➠ $exponent = -3 + 127 = 124 = 1111100_2$
  - ➠ $fraction = 0.25 = 2^{-2} = 0.01_2$
  - ➠ $0.15625 = 0.01111100\ 01000000000000000000000_2$

# FLOATING-POINT ARITHMETIC

## IEEE 754 STANDARD (BINARY32)

$$6.02 \times 10^{23} = ?$$

- We need a **normal form**
  $(-1)^{sign} \times 2^{exponent-127} \times 1.fraction$
- $6.02 \times 10^{23} = 1.991809 \times 2^{78}$
  (we get this after 78 successive divisions per 2)
  ➠ $exponent = 78 + 127 = 11001101_2$
  ➠ $fraction = 0.991809 = 0.11111101111010011111000_2$
  ➠ $6.02 \times 10^{23} = 0\ 11001101\ 11111101111010011111000_2$

# OUTLINE



Back to the outline - Back to the begin

# ASCII CHARACTER ENCODING

- All computer **data** is seen as a (finite) **sequence of bytes** (character)
  - **Example:** A file is a sequence of bytes, regardless of its contents
- **American Standard Code for Information Interchange** - **ASCII**
  - **ASCII** is a character encoding standard for electronic communication.
- **ASCII** codes represent text in computers, telecommunications equipment, and other devices.
- The first edition of the **ASCII standard** was published in 1963.

# ASCII CHARACTER ENCODING

- ASCII has just $128$ code points ($7$ bits + $1$ as parity bit).
  - Of the $2^7 = 128$ codes, $33$ were used for controls, and $95$ for printable characters

| Hex | Value | Hex | Value | Hex | Value | Hex | Value | Hex | Value | Hex | Value | Hex | Value | Hex | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | NUL | 10 | DLE | 20 | SP | 30 | 0 | 40 | @ | 50 | P | 60 | ` | 70 | p |
| 01 | SOH | 11 | DC1 | 21 | ! | 31 | 1 | 41 | A | 51 | Q | 61 | a | 71 | q |
| 02 | STX | 12 | DC2 | 22 | " | 32 | 2 | 42 | B | 52 | R | 62 | b | 72 | r |
| 03 | ETX | 13 | DC3 | 23 | # | 33 | 3 | 43 | C | 53 | S | 63 | c | 73 | s |
| 04 | EOT | 14 | DC4 | 24 | $ | 34 | 4 | 44 | D | 54 | T | 64 | d | 74 | t |
| 05 | ENQ | 15 | NAK | 25 | % | 35 | 5 | 45 | E | 55 | U | 65 | e | 75 | u |
| 06 | ACK | 16 | SYN | 26 | & | 36 | 6 | 46 | F | 56 | V | 66 | f | 76 | v |
| 07 | BEL | 17 | ETB | 27 | ' | 37 | 7 | 47 | G | 57 | W | 67 | g | 77 | w |
| 08 | BS | 18 | CAN | 28 | ( | 38 | 8 | 48 | H | 58 | X | 68 | h | 78 | x |
| 09 | HT | 19 | EM | 29 | ) | 39 | 9 | 49 | I | 59 | Y | 69 | i | 79 | y |
| 0A | LF | 1A | SUB | 2A | * | 3A | : | 4A | J | 5A | Z | 6A | j | 7A | z |
| 0B | VT | 1B | ESC | 2B | + | 3B | ; | 4B | K | 5B | [ | 6B | k | 7B | { |
| 0C | FF | 1C | FS | 2C | , | 3C | < | 4C | L | 5C | \ | 6C | l | 7C | \| |
| 0D | CR | 1D | GS | 2D | - | 3D | = | 4D | M | 5D | ] | 6D | m | 7D | } |
| 0E | SO | 1E | RS | 2E | . | 3E | > | 4E | N | 5E | ^ | 6E | n | 7E | ~ |
| 0F | SI | 1F | US | 2F | / | 3F | ? | 4F | O | 5F | _ | 6F | o | 7F | DEL |

# EXTENDED ASCII

- **Extended ASCII** is a repertoire of character encodings that include the original **ASCII** character set, plus up to 128 additional characters.
- In 1987, the **ISO** ( **International Organization for Standardization**) published a set of standards for **8-bit ASCII** extensions, **ISO 8859**
  - **ISO 8859-1**: for the most common Western European languages.
  - **ISO 8859-2**: for Eastern European languages.
  - **ISO 8859-xxx**: ...

# THE UNICODE STANDARD

- **Unicode** is a text encoding standard maintained by the **Unicode Consortium** designed to support the use of text in all of the world's major writing systems.
- **Unicode** is used to encode the vast majority of text on the Internet, including most web pages.
- 149 813 code points in the last published version.
(15.1, September 2023)

# THE UNICODE STANDARD

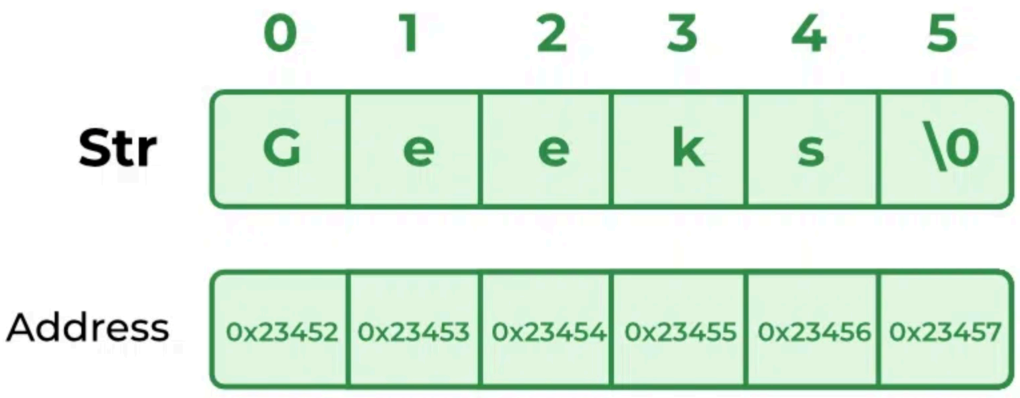
| Forme | Used bits | Code points |
|---|---|---|
| 0xxxxxxx | 7 | 0 to 127 |
| 110xxxxx 10xxxxxx | 11 | 128 to 2 047 |
| 1110xxxx 10xxxxxx 10xxxxxx | 16 | 2 048 to 65 535 |
| 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx | 21 | 65 536 to 1 114 111 |


UTF-8

# STRINGS ENCODING

- The most used representation is a **character array**.
  - but an array is not directly manipulated by the processor.

- The processor needs to know the address of the beginning of the array and the index of the element it wants to access.
  - the first memory word contains the number of characters,
  - or the **ASCII** code 0 (**NULL**) indicates the end of the string (like in **C**)



# THANK YOU

Version PDF des slides

Retour à l'accueil - Retour au plan