2. Information Representation

Informática

Ingeniería en Tecnologías Industriales

Raúl Durán Díaz   Juan Ignacio Pérez Sanz
Álvaro Perales Eceiza

Departamento de Automática
Escuela Politécnica Superior

Course 2024–2025

Universidad de Alcalá

Rev: 1.5

Contents

Positional Representation

- Positional representation is based on the next theorem:

Theorem

Let $b > 1$ be a positive integer. Any positive integer n can be written in a unique way as

$$n = \sum_{j=0}^{k} a_j b^j = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0,$$

with $0 \leq a_j \leq b - 1$ for $j = 0, \ldots, k$, y $a_k \neq 0$.

- So we can write the positional representation of n as

$$n = (a_k, a_{k-1}, \ldots, a_0),$$

or just $a_k a_{k-1} \ldots a_0$.

Representation Bases

- As the theorem states, we can use any integer b as base to represent all integer numbers.
- Traditionally we use base $b = 10$, or *decimal*.
- However computers use base $b = 2$ or *binary* to make information process more efficient inside them.
- Ii is very common to use base $b = 16$ or *hexadecimal* as an easier and more compact way for humans to represent binary information

Rational numbers representation

- Rational numbers are always a ratio of two integers.
- To include the fractional part of a rational number, we can extend the positional system using the negative powers of the base:

$$n = \sum_{j=\ell}^{k} a_j b^j = a_k b^k + \cdots + a_1 b + a_0 + a_{-1} b^{-1} + \cdots + a_\ell b^\ell,$$

  with $\ell \leq 0 \leq k$.
- We can't represent exactly irrational numbers, (e.g. $\sqrt{2}, \pi, e$), so we take as an approximation the closets rational number that we can represent.

Rational numbers representation

- Let r be a rational number $r = \left[\frac{p}{q}\right]$ with $q = b^s$ where b is the base and s any positive integer. Then r can be expressed as:

$$r = \frac{p}{q} = \frac{\sum_{j=0}^{k} p_j b^j}{b^s} = \sum_{j=0}^{k} p_j b^{j-s}.$$

- If $k > s$, then r can be expressed as

$$r = \left(p_k p_{k-1} \cdots p_s, p_{s-1} \cdots p_0\right),$$

where $p_{s-1}, \ldots, p_0$ are the coefficients of the negative powers of b.

Base Change

- Let b_1 and b_2 be two different bases. Let (u, v) be a real number where u is the integer part and v is the fractional part.
- Then (u, v) can be represented with both bases:
  - With base b_1:
    $u = (p_{k-1}p_{k-2}\cdots p_0)_{b_1}$, $v = (,p_{-1}p_{-2}\cdots p_{-\ell})_{b_1}$,
    with $k, \ell > 0$.
  - With base b_2:
    $u = (q_{K-1}q_{K-2}\cdots q_0)_{b_2}$, $v = (,q_{-1}q_{-2}\cdots q_{-L})_{b_2}$,
    with $K, L > 0$.
- A very common task for computers is to pass from the representation in one base to the other (e.g. represent the decimal number 17 in binary).

Universidad de Alcalá

Rev: 1.5

Base Change

To obtain the integer part:

Divide successively $(u)_{b_1}$ by $(b_2)_{b_1}$. The remainders q_i are the digits of $(u)_{b_2}$ starting with q_0 until q_{K-1}.

To obtain the fractional part:

Multiply successively $(v)_{b_1}$ by $(b_2)_{b_1}$. After each multiplication, the integer parts q_i will form the digits of $(v)_{b_2}$ (from q_{-1} to q_{-L}). Before the next multiplication the previous integer part must be removed.

Example: Represent the decimal number 22.375 in binary (i.e. change from base 10 to base 2)

- Integer part: $u = 22$

dividend	quotient	remainder
22	11	0
11	5	1
5	2	1
2	1	0
1	0	1

- Fractional part: $v =, 375$

multiplicand	product	integer part
0,375	0,75	0
0,75	1,5	1
0,5	2	1

- Therefore the result is 10110.011

Inverse Base Change

- Just apply the opposite procedure or the positional formula

Example: Express the binary number 10110.011 in decimal

- Integer part: $u = 10110$

$$1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 22.$$

- Fractional part: $v =, 011$

$$0 \times 2^{-1} + 1 \times 2^{-2} + +1 \times 2^{-3} = 0.375.$$

Therefore the result is 22.375.

Universidad de Alcalá

Rev: 1.5

What is a *codification*?

- From chapter 1:

Definition

Codification: is a bijective correspondence among the elements of two sets

Observation

As it is bijective (i.e. one-to-one and onto) we can identify the elements of the first set using the ones of the second set.

More formally ...

- Let A and B be two sets and let $f: A \to B$ be a function.

Definition

We can say that B *codifies* A by f if f is *bijective*

- If the sets are provided with an inner operation $(A, +)$, $(B, \oplus)$:

Definition

If $f(a+b) = f(a) \oplus f(b)$ for any $a, b \in A$, then we have a *faithful representation* (or *codification*)

- Example: We obtain the same result adding two numbers in decimal or binary representations:
  $2 + 4 = 6$, $0010 + 0100 = 0110$, and $6_{10} = 0110_2$

Universidad de Alcalá

Rev: 1.5

Modulo Operation

Definition

Let $m > 0$. Then the modulo operation with two integer numbers, $b = a \pmod{m}$, is the remainder of a divided by m.
(therefore $a = q \cdot m + b$, for some integer q)

Example

- $7 \pmod{2} = 1$, as $7 = 3 \times 2 + 1$
- Clocks work modulo 12 or 24 hours.

Universidad de Alcalá

Rev: 1.5

Operations in $\mathbb{Z}$ and B

- The set of all integers is $\mathbb{Z}$
- B_w is the set of all binary numbers with w digits
  There are 2^w binary numbers with w digits (e.g. for $w = 2$ there are 2^2 binary numbers $\{00, 01, 10, 11\}$
- Codification of integers is a bijective correspondence $R \rightarrow B$ where R is a subset of $\mathbb{Z}$
- We want also a faithful representation, that is, that operations in R correspond to operations in B obtaining the same result (e.g. $2 + 4 = 6$, $0010 + 0100 = 0110$).

Integer Representation

- The number of bits that a computer uses to store binary numbers is the *width* or *size* of a *word*,
- Usually is 8, 16, 32, or 64 bits.
- In programming languages, each size receives a name, for instance in C language:

`char`	⇒	8 bits.
`short int`	⇒	16 bits.
`int`	⇒	32 bits.
`long int`	⇒	64 bits.

Universidad de Alcalá

Summary of different binary representations

Fixed point	Unsigned binary	
	Signed binary	With sign bit
		One's complement
		Two's complement
		Excess-Z
Floating point	Integer significand	
	Fractional significand	

Universidad de Alcalá

Unsigned binary

- Corresponding function is simply the formula to change to base 2:

$$\begin{array}{rcl} f: R & \rightarrow & B \\ n & \mapsto & (x_{w-1}, \ldots, x_0)_2 \end{array}$$

  such us $n = \sum_{i=0}^{w-1} x_i 2^i$.
- For w bits, the set $R = \{0, 1, \ldots, 2^w - 1\}$ is codified as $0 \mapsto (0 \cdots 0)$, $\ldots$, $2^w - 1 \mapsto (1 \cdots 1)$ (positives and 0)
- Example: for $w = 3$, $\{0, \cdots, 2^3 - 1\} \mapsto \{000, \cdots, 111\}$
- It is a faithful representation

Universidad de Alcalá

Rev: 1.5

Signed binary

- Add an extra bit at the left to express the sign (0 for positive, 1 for negative)
- Therefor for w bits we can represent the set $R = \{-2^{w-1} + 1, \ldots, 2^{w-1} - 1\}$.
- Example: $-3_{10} = 1011_2$
- It is NOT a faithful representation as 0 can be represented in two ways $(+0, -0)$, and therefore is not bijective.

Universidad de Alcalá

Rev: 1.5

Excess-Z binary representation

- Simply add a positive integer $Z > 0$: $n \mapsto n + Z$, $n \in R$. Assuming that $n + Z \geq 0$, we can represent $R = \{-Z, \ldots, Z - 1\}$.
- Use unsigned binary representation to express the result

$$n + Z = \sum_{i=0}^{w-1} x_i 2^i.$$

- Typically for w bits we choose $Z = 2^{w-1}$
- It is used to represent the exponential in floating point representation (see below)

Universidad de Alcalá

Rev: 1.5

Excess-Z binary representation

- It is NOT a faithful representation:
  Let $n, m \in R$

$$
\begin{array}{rcc}
n & \mapsto & n+Z \\
+ & & + \\
m & \mapsto & m+Z \\
\hline
n+m & \not\mapsto & n+m+2Z,
\end{array}
$$

i.e. it is necessary to subtract Z to get the correct result in R

One's Complement -1C- binary representation

- Positive 1C numbers are the same than in signed binary (SB)
  $+5_{10} = 0101_{SB} = 0101_{1C}$
- To get 1C representation of a negative number swap all bits ($0 \rightarrow 1, 1 \rightarrow 0$) of the corresponding positive signed binary:
  $-5_{10} = 1101_{SB} = 1010_{1C}$
- Range of representation $R_{1C} = \{-2^{w-1} - 1, \ldots, 2^{w-1} - 1\}$
- It is NOT a faithful representation as it is not bijective because the number 0 can be represented in two ways ($+0, -0$)
- Much less used than 2C

Two's Complement -2C- binary representation

- Positive 2C numbers are the same than in SB
  $+5_{10} = 0101_{SB} = 0101_{1C} = 0101_{1C}$
- To get the 2C representation of a negative number
  - Obtain 1C
  - Add $+1$
  - $-5_{10} = 1101_{SB} = 1010_{1C} = 1011_{2C}$
- To know the magnitude of a negative 2C number, compute its 2C again to obtain the corresponding positive

Two's Complement -2C- binary representation

- Range of 2C representation $R_{2C} = \{-2^{w-1}, \ldots, 2^{w-1} - 1\}$.

$$
\begin{array}{ccc}
-2^{w-1} & \mapsto & (1, 0, \ldots, 0)\,, \\
 & \cdots & \\
-1 & \mapsto & (1, 1, \ldots, 1)\,, \\
0 & \mapsto & (0, 0, \ldots, 0)\,, \\
1 & \mapsto & (0, 0, \ldots, 1)\,, \\
 & \cdots & \\
2^{w-1} - 1 & \mapsto & (0, 1, \ldots, 1)\,.
\end{array}
$$

- It is **UNIVERSALLY USED** by computers:
  - It is bijective and faithful with $\{+, -, \times, \div\}$ operations
  - To subtract is very easy: just add the 2C of the number

Universidad de Alcalá

Rev: 1.5

Floating point representation

- The idea is to save space without loosing accuracy by means of moving the coma and changing the exponent: (decimal example: $0.00027 \times 10^{-2} = 2.7 \times 10^{-6}$)
- Each number x is represented as $x = \pm m \times b^e$, where

m	significand or mantissa
b	base
e	exponent

Example

$$a = (1.001)_2 \times 2^{-5}$$
$$b = (1.001)_2 \times 2^{7}$$

Universidad de Alcalá

Rev: 1.5

Floating point format

- The typical format to represent a floating point number is:

s	exponent	mantissa

- *Sign* 0 $\rightarrow$ positive, 1 $\rightarrow$ negative.
- *Exponent*: Integer expressed in Z-excess with $Z = 2^{w_e - 1}$, where w_e is the number of bits to store it.
- *Significand or mantissa*:
  - *Integer*: not used
  - *Fractional*: It is generally *normalized* such as the integer part is just one significant bit ($\neq 0$)

Universidad de Alcalá

Rev: 1.5

Floating point examples

Example

- $a = 1.001 \times 2^{-5}$. Exponent is $e = -5$ and the mantissa $m = 1.001$ is already normalized (1 in the integer part)
- $a = 10.01 \times 2^{-6}$. Exponent is $e = -6$ and $m = 10.01$ is not normalized (two bits in the integer part)
- $a = 0.1001 \times 2^{-4}$. Exponent is $e = -4$ and $m = 0.1001$ is not normalized (the integer part is 0)

By the way: $a = \frac{(1001)_2}{2^3} \times \frac{1}{2^5} = \frac{9}{2^8} = 0.03515625$.

Universidad de Alcalá

Rev: 1.5

ANSI/IEEE 754 Standard representation

- MOST EXTENDED standard to represent floating point numbers in computations.
- Defines the size in bits of each field.
- Normalized mantissa $\rightarrow$ just one integer bit always $= 1$. Therefore is never stored (*implicit bit*)
- There are two sizes::
  - Simple precision floating point, `float`, total size $= 32$ bits.
  - Double precision floating point, `double`, total size $= 64$ bits.

Universidad de Alcalá

Rev: 1.5

ANSI/IEEE 754 Standard. Special values

- **Zero** cannot be represented, so it is chosen by convention to be the number with all bits $= 0$ (otherwise would be 1.0×2^{-127} for `float` and 1.0×2^{-1023} for `double`.
- **Infinity**. By convention two different codes are chosen to represent $\pm\infty$ (0/1 for sign, exponent all 1's, mantissa al 0's).
- **NaN**. Not a Number. Undefined result after some operation (for instance 0/0). Represented as well by a particular code.

ANSI/IEEE 754 Standard

	simple	doble
Total Size	32 bits	64 bits
Mantissa	$23 + 1$ bits	$52 + 1$ bits
Exponent	8 bits	11 bits
Excess	$2^7 - 1$	$2^{10} - 1$
Minimum	$2^{-126} \simeq 1.2 \times 10^{-38}$	$2^{-1022} \simeq 2.2 \times 10^{-308}$
Maximum	$2^{128} - 2^{-127} \simeq 3.4 \times 10^{38}$	$2^{1024} - 2^{-1023} \simeq 1.8 \times 10^{308}$
Zero	$e + exc = 0\ ,\ m = 0$	$e + exc = 0,\ m = 0$
Infinity	$e + exc = 255,\ m = 0$	$e + exc = 2047,\ m = 0$
NaN	$e + exc = 255,\ m \neq 0$	$e + exc = 2047,\ m \neq 0$

Universidad de Alcalá

Rev: 1.5

Alphanumeric Information Representation

- Alphanumeric Information is codified with character tables.
- Each element is represented by a binary code
- Each table defines the number of bits to represent each character.
- There are different standards:
  - ANSI/ASCII.
  - ISO8859-XX.
  - Unicode, UTF-8, UTF-16.
  - BM/EBCDIC.

ANSI/ASCII-7 table

- 7 bits are used to codify 128 alphanumeric characters.

Examples:

Character	"0"	"1"	...	"9"	"A"	...	"Z"
ASCII-7 code	48	49	...	57	65	...	90

ISO8859-15 table

- 8 bits to codify 256 alphanumeric characters
  - First 128 are the same than in ASCII-7
  - Last 128 are Western language characters

Examples:

Character	"é"	...	"è"	...	"û"	...
ISO8859-15 code	130	...	138	...	150	...

Universidad de Alcalá

Rev: 1.5

UTF-8 table

- It uses variable length codes, from 8 to 16 bits.
- For codes smaller than 128 is fully compatible with ASCII-7
- It allows to codify character of many languages, including Easter ones

Character	"é"	...	"è"	...	"û"	...
UTF-8 code	0xC3A9	...	0xC3A8	...	0xC3BB	...

Universidad de Alcalá

Rev: 1.5

Character Chains

To store character chains in memory another aspect must be considered:

- How to codify the chain length. Three main methods
  - Terminator method
  - Length indicator method
  - Descriptor method

Universidad de Alcalá

Rev: 1.5

Terminator method

- A special character is used to indicate the end of the chain. Typically 0 is used.
- To access the chain it is only necessary to know the address of the first character.

Example

To represent the string `"Hi!!"` with ISO8859-15 table we use five bytes:

H	i	!	!	0

Length indicator method

- The first (or first and second) byte(s) of the chain indicate(s) its length.
- To access the chain it is only necessary to know the address of the first character.
- This method limits the maximum length of the chain.

Example

To represent the string `"Hi!!"` with ISO8859-15 table we use five bytes:

4	H	i	!	!

Universidad de Alcalá

Rev: 1.5

Descriptor method

- Chain characters are written alone from a memory position onward
- To access the chain it is necessary to know the address of the first character AND its length. These two data together form the *descriptor*

Example

To represent the string `"Hi!!"` with ISO8859-15 table we use four bytes:

H	i	!	!