



# Rappresentazione dell'informazione

Prof. Alberto Borghese  
Dipartimento di Informatica  
[alberto.borghese@unimi.it](mailto:alberto.borghese@unimi.it)

Università degli Studi di Milano

Riferimenti al testo: Paragrafi 2.4, 2.9, 3.1, 3.2, 3.5 (codifica IEEE754)



## Sommario

### Rappresentazione binaria dell'Informazione

Sistema di numerazione binario

Conversione in e da un numero binario

Operazioni elementari su numeri binari: somma, sottrazioni.

I numeri decimali

Codifica IEEE754 dei numeri reali anche in forma denormalizzata.



## Il linguaggio



Per farsi capire da un calcolatore, occorre parlare la sua stessa lingua.



## Proprietà di potenze e logaritmi



$$2^K \times 2^M = 2^{(K+M)}$$

$$2^{K^M} = 2^{K * M} = 2^{M^K}$$

$$2^{-K} = \frac{1}{2^K}$$

Il logaritmo è l'operazione inversa dell'elevamento a potenza.

$$\log_2(2^M) = M$$

$$\log_2 K = -\log_2 \left( \frac{1}{K} \right)$$

$$\log_2 KM = \log_2 K + \log_2 M$$



## Rappresentazione dell'informazione



Non solo conteggio, ma anche enumerazione di oggetti....

Noi rappresentiamo gli oggetti tramite parole composte da un alfabeto di simboli: A,B,...,Z,0,1,...,9,...

- Diversi alfabeti possono essere usati per rappresentare gli stessi oggetti.
- I simboli degli alfabeti possono assumere diverse forme.
- Segni su carta, livelli di tensione, fori su carta, segnali di fumo.

.....



## Codifica dei caratteri alfanumerici




Quanti bit devono avere le parole binarie usate per identificare i 26 caratteri diversi dell'alfabeto inglese (es: A,B,...,Z)?

$$2^4 < 26 < 2^5$$

Quanti bit devono avere le parole binarie usate per identificare 26+26 oggetti diversi (es: A,B,...,Z, a,b, ..... z)?

$$2^5 < 52 < 2^6$$

Quanti bit servono per 100 oggetti?  $\text{ceil}[\log_2 100]$






|    |    |    |    |    |    |     |     |     |     |     |     |     |     |     |   |
|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| 0  |    | 32 | 64 | @  | 96 | `   | 128 | Ç   | 160 | á   | 192 | L   | 224 | œ   |   |
| 1  | ☐  | 33 | !  | 65 | Á  | 97  | a   | 129 | ü   | 161 | í   | 193 | l   | 225 | ß |
| 2  | ☐  | 34 | "  | 66 | B  | 98  | b   | 130 | é   | 162 | ó   | 194 | T   | 226 | Γ |
| 3  | ♥  | 35 | #  | 67 | C  | 99  | c   | 131 | â   | 163 | ú   | 195 |     | 227 | Π |
| 4  | ♦  | 36 | \$ | 68 | D  | 100 | d   | 132 | ä   | 164 | ï   | 196 | -   | 228 | Σ |
| 5  | ♣  | 37 | %  | 69 | E  | 101 | e   | 133 | à   | 165 | ñ   | 197 | +   | 229 | σ |
| 6  | ♠  | 38 | &  | 70 | F  | 102 | f   | 134 | ã   | 166 | æ   | 198 |     | 230 | μ |
| 7  | •  | 39 | '  | 71 | G  | 103 | g   | 135 | ç   | 167 | ë   | 199 |     | 231 | τ |
| 8  | ☐  | 40 | (  | 72 | H  | 104 | h   | 136 | ê   | 168 | ç   | 200 |     | 232 | ξ |
| 9  | ◊  | 41 | )  | 73 | I  | 105 | i   | 137 | ë   | 169 | ç   | 201 |     | 233 | θ |
| 10 | ☐  | 42 | *  | 74 | J  | 106 | j   | 138 | è   | 170 | ç   | 202 |     | 234 | Ω |
| 11 | ♠  | 43 | +  | 75 | K  | 107 | k   | 139 | í   | 171 | ½   | 203 |     | 235 | δ |
| 12 | ♠  | 44 | ,  | 76 | L  | 108 | l   | 140 | î   | 172 | ¼   | 204 |     | 236 | ∞ |
| 13 | ♠  | 45 | -  | 77 | M  | 109 | m   | 141 | ï   | 173 | í   | 205 | =   | 237 | ∞ |
| 14 | ♠  | 46 | .  | 78 | N  | 110 | n   | 142 | ñ   | 174 | «   | 206 |     | 238 | € |
| 15 | ♠  | 47 | /  | 79 | O  | 111 | o   | 143 | ñ   | 175 | »   | 207 |     | 239 | ∞ |
| 16 | ♠  | 48 | 0  | 80 | P  | 112 | p   | 144 | é   | 176 |     | 208 |     | 240 | ≡ |
| 17 | ♠  | 49 | 1  | 81 | Q  | 113 | q   | 145 | æ   | 177 |     | 209 |     | 241 | ± |
| 18 | ♠  | 50 | 2  | 82 | R  | 114 | r   | 146 | ff  | 178 |     | 210 |     | 242 | ≥ |
| 19 | !! | 51 | 3  | 83 | S  | 115 | s   | 147 | ô   | 179 |     | 211 |     | 243 | ≤ |
| 20 |    | 52 | 4  | 84 | T  | 116 | t   | 148 | ö   | 180 |     | 212 |     | 244 | ∫ |
| 21 |    | 53 | 5  | 85 | U  | 117 | u   | 149 | ò   | 181 |     | 213 |     | 245 | J |
| 22 | -  | 54 | 6  | 86 | U  | 118 | v   | 150 | û   | 182 |     | 214 |     | 246 | ÷ |
| 23 | ±  | 55 | 7  | 87 | W  | 119 | w   | 151 | ù   | 183 |     | 215 |     | 247 | ≈ |
| 24 | ↑  | 56 | 8  | 88 | X  | 120 | x   | 152 | ÿ   | 184 |     | 216 |     | 248 | ° |
| 25 | ↓  | 57 | 9  | 89 | Y  | 121 | y   | 153 | ÿ   | 185 |     | 217 |     | 249 | · |
| 26 | →  | 58 | :  | 90 | Z  | 122 | z   | 154 | ü   | 186 |     | 218 |     | 250 | · |
| 27 | ←  | 59 | ;  | 91 | [  | 123 | {   | 155 | ç   | 187 |     | 219 |     | 251 | ∫ |
| 28 | ↔  | 60 | <  | 92 | \  | 124 |     | 156 | £   | 188 |     | 220 |     | 252 | ∞ |
| 29 | ⊕  | 61 | =  | 93 | ]  | 125 | }   | 157 | ¥   | 189 |     | 221 |     | 253 | z |
| 30 | ▲  | 62 | >  | 94 | ^  | 126 | ~   | 158 | ℞   | 190 |     | 222 |     | 254 | ■ |
| 31 | ▼  | 63 | ?  | 95 | _  | 127 | △   | 159 | f   | 191 |     | 223 |     | 255 | ■ |

**Il codice ASCII la rappresentazione dell'informazione alfanumerica**

- 8 bit
- 0-31 codici di controllo.
- 128-255 extended ASCII

A.A. 2019-2020 http://borghese.di.unimi.it/

## L'UNICODE

<http://www.unicode.org>. Codifica su 8, 16, 32 bit alfabeti diversi.

|            |                                      |                         |                                   |
|------------|--------------------------------------|-------------------------|-----------------------------------|
| Latin      | Malayalam                            | Tagbanwa                | General Punctuation               |
| Greek      | Sinhala                              | Khmer                   | Spacing Modifier Letters          |
| Cyrillic   | Thai                                 | Mongolian               | Currency Symbols                  |
| Armenian   | Lao                                  | Limbu                   | Combining Diacritical Marks       |
| Hebrew     | Tibetan                              | Tai Le                  | Combining Marks for Symbols       |
| Arabic     | Myanmar                              | Kangxi Radicals         | Superscripts and Subscripts       |
| Syriac     | Georgian                             | Hiragana                | Number Forms                      |
| Thaana     | Hangul Jamo                          | Katakana                | Mathematical Operators            |
| Devanagari | Ethiopic                             | Bopomofo                | Mathematical Alphanumeric Symbols |
| Bengali    | Cherokee                             | Kanbun                  | Braille Patterns                  |
| Gurmukhi   | Unified Canadian Aboriginal Syllabic | Shavian                 | Optical Character Recognition     |
| Gujarati   | Ogham                                | Osmanya                 | Byzantine Musical Symbols         |
| Oriya      | Runic                                | Cypriot Syllabary       | Musical Symbols                   |
| Tamil      | Tagalog                              | Tai Xuan Jing Symbols   | Arrows                            |
| Telugu     | Hanunoo                              | Yijing Hexagram Symbols | Box Drawing                       |
| Kannada    | Buhid                                | Aegean Numbers          | Geometric Shapes                  |

A.A. 2019-2020 http://borghese.di.unimi.it/



## Sommario



Rappresentazione binaria dell'Informazione

**Sistema di numerazione binario**

Conversione in e da un numero binario

Operazioni elementari su numeri binari: somma, sottrazioni.

I numeri decimali

Codifica IEEE754 dei numeri reali anche in forma denormalizzata.



## Tassonomia ed unità di misura



Hertz - numero di ciclo al secondo nei moti periodici (clock).

- MIPS - Milioni di istruzioni per secondo.
- MFLOPS - Milioni di istruzioni in virgola mobile (FLOating point) al secondo.

*Prefissi:*

k - chilo (mille:  $10^3$ ).

M - mega (un milione:  $10^6$ ).

G - giga (un miliardo:  $10^9$ ).

T – tera (1000 miliardi:  $10^{12}$ )

P – peta (1,000,000 miliardi:  $10^{15}$ )

m - milli (un millesimo:  $10^{-3}$ )

$\mu$  - micro (un milionesimo:  $10^{-6}$ )

n - nano (un miliardesimo:  $10^{-9}$ )

p – pico (un millesimo di  
miliardo:  $10^{-12}$ )

f – femto (un milionesimo di  
miliardo:  $10^{-15}$ )



# Approssimazione



| Multipli del bit         |         |           |                         |         |          |   |
|--------------------------|---------|-----------|-------------------------|---------|----------|---|
| Prefissi SI              |         |           | Prefissi binari         |         |          |   |
| Nome                     | Simbolo | Multipli  | Nome                    | Simbolo | Multipli |   |
| <a href="#">kilobit</a>  | kbit    | $10^3$    | <b>kibibit</b>          | Kibit   | $2^{10}$ | 1024 bit                                  |
| <a href="#">megabit</a>  | Mbit    | $10^6$    | <a href="#">mebibit</a> | Mibit   | $2^{20}$ | 1 024 Kib                                 |
| <a href="#">gigabit</a>  | Gbit    | $10^9$    | <a href="#">gibibit</a> | Gibit   | $2^{30}$ | 1 048 576 Kib = 1 <a href="#">gibibit</a> |
| <a href="#">terabit</a>  | Tbit    | $10^{12}$ | <a href="#">tebibit</a> | Tibit   | $2^{40}$ | 1 024 Gbit                                |
| <a href="#">petabit</a>  | Pbit    | $10^{15}$ | <a href="#">pebibit</a> | Pibit   | $2^{50}$ | 1 024 Tbit                                |
| <a href="#">exabit</a>   | Ebit    | $10^{18}$ | <a href="#">exbibit</a> | Eibit   | $2^{60}$ | 1 024 Pbit                                |
| <a href="#">zettabit</a> | Zbit    | $10^{21}$ | <a href="#">zebibit</a> | Zibit   | $2^{70}$ | 1 024 Ebit                                |
| <a href="#">yottabit</a> | Ybit    | $10^{24}$ | <a href="#">yobibit</a> | Yibit   | $2^{80}$ | 1 024 Zbit                                |



# Terminologia



Bit = binary digit.

- 1 byte = 8 bit.
- 1kbyte =  $2^{10}$ byte = 1,024 byte
- 1Mbyte =  $2^{20}$ byte = 1,048,576 byte.
- 1Gbyte =  $2^{30}$ byte = 1,073,741,824 byte.
- 1Tbyte =  $2^{40}$ byte = 1,099,511,627,776 byte.

- Parola (word) numero di bit trattati come un unicum dall'elaboratore.
- Le parole oggi arrivano facilmente a 64bit (Itanium).



## Numerazione Simbolica



Sistema di numerazione mediante simboli (numerazione romana: I, V, X, L, C, M) il cui valore non dipende dalla posizione: e.g. XXXI = 31, XI = 11....

Sistema di numerazione posizionale (decimale): **cifra + peso**.  
Il peso è la base elevata alla posizione della cifra.

1 ha un valore diverso nelle due scritture:

100

1000



## Numerazione Posizionale



Alfabeto della numerazione:

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9} numerazione araba decimale.

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F} numerazione esadecimale.

{0, 1} numerazione binaria.

Sistemi di numerazione binario, ottale ed esadecimale.

Conversioni decimale -> binario e viceversa.



## Codifica posizionale di un numero



Fondata sul concetto di **base**:  $B = [b_0, b_1, b_2, b_3, \dots]$ .

Ciasun elemento,  $N$ , può essere rappresentato come combinazione lineare degli elementi della base: 
$$N = \sum_k c_k b_k$$

Esempi:

$$\bullet 764,3_{10} = 7 \times 10^2 + 6 \times 10^1 + 4 \times 10^0 + 3 \times 10^{-1} = 764,3 \quad b_k = B^k = 10^k$$

$$\bullet 12,21_{10} = 1 \times 10^1 + 2 \times 10^0 + 2 \times 10^{-1} + 1 \times 10^{-2} = 12,21 \quad b_k = B^k = 10^k$$

$$\bullet 100,11_2 = 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 4,75 \quad b_k = B^k = 2^k$$



## Osservazioni sulla numerazione binaria



Il linguaggio di un elaboratore elettronico è fatto di due segnali: **on** e **off**, rappresentati dai simboli **1** e **0** (**alfabeto binario**).

- Sia le istruzioni che i dati sono rappresentati da *parole* di numeri binari.
- Un alfabeto binario non limita le funzionalità di un elaboratore a patto di avere parole di lunghezza sufficiente.
- 00000011001010001101000000100000 rappresenta un'istruzione di addizione in MIPS su 32 bit (add \$k0, \$t0, \$t9).





## Codifica binaria



Quanti oggetti diversi possiamo rappresentare con parole binarie di  $k$  bit?

- Con una parola di 1 bit rappresentiamo 2 oggetti (1 bit ha due possibili valori).
- Supponiamo di avere parole di  $k-1$  bit. Quanti oggetti riescono a rappresentare?

$2^{k-1}$  oggetti.



## Esempio di codifica binaria



- Quanti oggetti diversi possiamo rappresentare con parole binarie di 3 bit?

|   |     |   |
|---|-----|---|
| 0 | 000 | A |
| 1 | 001 | B |
| 2 | 010 | C |
| 3 | 011 | D |
| 4 | 100 | E |
| 5 | 101 | F |
| 6 | 110 | G |
| 7 | 111 | H |



## Sommario



Sistema di numerazione binario

Rappresentazione binaria dell'Informazione

**Conversione in e da un numero binario**

Operazioni elementari su numeri binari (somma, sottrazione e moltiplicazione intera).

I numeri decimali

Codifica IEEE754 dei numeri reali anche in forma denormalizzata.



## Conversione da base n a base 10



Un numero  $N = [c_0, c_1, c_2, c_3, \dots]$  in base 10,  $B = [b_0, b_1, b_2, b_3, \dots]$  si trasforma in base n,  $R = [r_0, r_1, r_2, r_3, \dots]$ , facendo riferimento alla formula:

$$N = \sum_k c_k b_k = \sum_{k=0}^{N-1} d_k r^k$$

- ciascuna cifra k-esima viene moltiplicata per la base corrispondente:  
 $r_k = n^k$ .

- i valori così ottenuti sono sommati per ottenere il numero in notazione decimale.

$$\begin{aligned} \mathbf{101\ 1101\ 0101_{due}} &= 1x2^{10} + 0x2^9 + 1x2^8 + 1x2^7 + 1x2^6 + 0x2^5 + \\ &1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = \\ &1024 + 256 + 128 + 64 + 16 + 4 + 1 = 1493 \end{aligned}$$



## “Spelling” di un numero



Vogliamo rappresentare  $1493_{\text{dieci}}$

**Unità**                     $1493 = 10 \times 149 + 3$     ← Cifra meno significativa

**Decine**    (10x)     $149 = 10 \times 14 + 9$

**Centinaia** (100x)     $14 = 10 \times 1 + 4$

**Migliaia**    (1000x)     $1 = 10 \times 0 + 1$                     ← Cifra più significativa

$$1493 = 3 \times 1 + 9 \times 10 + 4 \times 100 + 1 \times 1000$$



## “estrazione” delle cifre decimali



Vogliamo estrarre le cifre di  $1493_{\text{dieci}}$ . Porto le cifre alla destra della virgola:

$$1493 / 10 = 149,3$$

→ esamino 149 → 3 unità

$$149 / 10 = 14,9$$

→ esamino 14 → 9 decine

$$14 / 10 = 1,4$$

→ esamino 1 → 4 centinaia

$$1 / 10 = 0,1$$

→ termina → 1 migliaia



## Meccanismo di “estrazione”



Vogliamo estrarre le cifre di  $1493_{\text{dieci}}$ . Porto le cifre alla destra della virgola.

Utilizzo la divisione intera per la base 10, il resto rappresenta la cifra decimale meno significativa.

$$1493 / 10 = 149 \text{ con } R = 3 \rightarrow 3 \text{ unità}$$

$$149 / 10 = 14 \text{ con } R = 9 \rightarrow 9 \text{ decine}$$

$$14 / 10 = 1 \text{ con } R = 4 \rightarrow 4 \text{ centinaia}$$

$$1 / 10 = 0 \text{ con } R = 1 \rightarrow 1 \text{ migliaia}$$

**Termina perchè non è rimasto nulla del numero.**



## Conversione base 10 -> base 2 “estrazione” delle cifre binarie



Vogliamo rappresentare  $1493_{\text{dieci}}$  in binario:  $10111010101_{\text{due}}$

$$1493 / 2 = 746 + 1 \quad \leftarrow \text{Bit meno significativo (LSB)}$$

$$746 / 2 = 373 + 0 \quad 373 * 2 + 0 = 746$$

$$373 / 2 = 186 + 1 \quad 186 * 2 + 1 = 373 \Rightarrow (186 * 2 + 1) * 2 = 373$$

$$186 / 2 = 93 + 0$$

$$93 / 2 = 46 + 1$$

$$46 / 2 = 23 + 0$$

$$23 / 2 = 11 + 1$$

$$11 / 2 = 5 + 1$$

$$5 / 2 = 2 + 1$$

$$2 / 2 = 1 + 0$$

$$1 / 2 = 0 + 1 \quad \leftarrow \text{Bit più significativo (MSB)}$$



## Perché funziona?



Prendiamo il numero  $E_0$  e dividiamo per 2.

Se  $E_0$  è pari il resto,  $R_0$ , sarà 0, altrimenti sarà 1. Infatti:

$$\text{int}(E_0 / 2) * 2 + R_0 = E_0.$$

Esempio:

$$E_0 = 5_{10} = ?_2$$

$$R_0 = \text{resto}(5/2) = 1$$

Chiamiamo  $E_1 = \text{int}(E_0 / 2)$  e procediamo.

$$E_1 = \text{quoz}(5/2) = 2$$

Prendiamo  $E_1$  e dividiamo per 2. Chiamiamo  $E_2 = \text{int}(E_1 / 2)$ . Se  $E_1$  è pari il resto,  $R_1$ , sarà 0, altrimenti sarà 1.  $E_1 = \text{int}(E_1 / 2) * 2 + R_1$

$$R_1 = \text{resto}(2/2) = 0$$

Chiamiamo  $E_2 = \text{int}(E_1 / 2)$  e procediamo.

$$E_2 = \text{quoz}(2/2) = 1$$

Prendiamo  $E_2$  e dividiamo per 2. Se  $E_2$  è pari il resto,  $R_2$ , sarà 0, altrimenti sarà 1.  $E_2 = \text{int}(E_2 / 2) * 2 + R_2$ .

$$R_2 = \text{resto}(1/2) = 0$$

Si procede fino a quando  $E_i$  non è  $< 2$  ( $=1$ ).

$$E_0 = R_0 + R_1 * 2 + R_2 * 2^2 \\ = 1 \ 0 \ 1$$

$$E_0 = \text{int}[E_0 / 2] * 2 + R_0.$$

$$E_0 = \text{int}[E_1] * 2 + R_0.$$

$$E_0 = \text{int}[\text{int}(E_1 / 2) * 2 + R_1] * 2 + R_0.$$

$$E_0 = \text{int}[\text{int}(E_2) * 2 + R_1] * 2 + R_0.$$

$$E_0 = \text{int}[\text{int}[\text{int}(E_2 / 2) * 2 + R_2] * 2 + R_1] * 2 + R_0. \quad \text{int}(E_2 / 2) = 0$$

<http://borghese.di.unimi.it/>



## Conversione base 10 -> base n: algoritmo



Un numero  $x$  in base 10 si trasforma in base  $n$  usando il seguente procedimento:

- Dividere il numero  $x$  per  $n$
- Il resto della divisione è la cifra di posto 0 in base  $n$
- Il quoziente della divisione è a sua volta diviso per  $n$
- Il resto ottenuto a questo passo è la cifra di posto 1 in base  $n$
- Si prosegue con le divisioni dei quozienti ottenuti al passo precedente fino a che l'ultimo quoziente è 0.
- l'ultimo resto è la cifra più significativa in base  $n$



## Esercizi



Dati i numeri decimali 23456, 89765, 67489, 121331, 2453, 111010101

- si trasformino in base 3
- si trasformino in base 7
- si trasformino in base 2
  
- Dati i numeri  $23456_7$ ,  $121331_5$ ,  $2453_8$ ,  $111010101_2$
- convertire ciascuno in decimale e in binario



## Codifica esadecimale



Il codice esadecimale viene utilizzato come forma compatta per rappresentare numeri binari:

- 16 simboli: 0,1,...,9,A,B,...,F
- Diverse notazioni equivalenti:  
 $0x9F$   
 $9F_{16}$   
 $9F_{hex}$

$$0x9F = 9 \times 16^1 + 15 \times 16^0 = 159_{10}$$



## Conversione esadecimale -> binario



Vogliamo rappresentare 9Fhex in binario. E' semplice.

- Ogni simbolo viene convertito in un numero binario di 4 cifre:

9hex --> 1001<sub>due</sub>

Fhex --> 1111<sub>due</sub>

9Fhex --> 10011111<sub>due</sub>

- È sufficiente ricordarsi come si rappresentano in binario i numeri decimali da 0 a 15 (o derivarli)



## Conversione binario -> esadecimale



Da binario ad esadecimale si procede in modo analogo:

- Ogni gruppo di 4 cifre viene tradotto nel simbolo corrispondente:

Esempio: convertire 1101011<sub>due</sub> in esadecimale:

1011<sub>due</sub> --> B<sub>hex</sub>

110<sub>due</sub> --> 6<sub>hex</sub>

1101011<sub>due</sub> --> 6B<sub>hex</sub>

Viene aggiunto un "leading" 0

00000011001010001101000000100000 - add \$k0, \$t0, \$t9

0x0328d020



## Codifica dei numeri interi



Viene replicato il bit più significativo

Codifica su 16 bit:

Numeri naturali:  $11_{10} = 1011_2 = 0000\ 0000\ 0000\ 1011$

Replico il primo bit, parte integrante del numero

Numeri relativi:  $+5_{10} = 0101_2 = 0000\ 0000\ 0000\ 0101$

Numeri relativi:  $-5_{10} = 1011_2 = 1111\ 1111\ 1111\ 1011$

**Replico il primo bit, quello del segno**



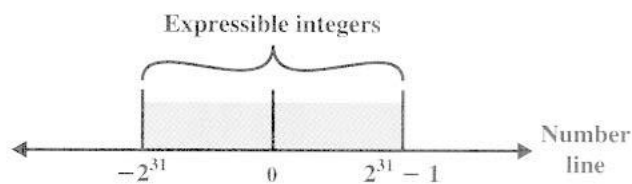
## Capacità di rappresentazione: Numeri Interi



Interi con segno su N bit. Range:  $-2^{N-1} \leq n \leq 2^{N-1} - 1$ .

Esempio: Visual C++. Intero è su 4byte (word di 32 bit):

$$-2^{31} = -2.147.483.650 \leq n \leq 2.147.483.649 = 2^{31} - 1$$



(a) Twos complement integers





## Sommario



Sistema di numerazione binario

Rappresentazione binaria dell'Informazione

Conversione in e da un numero binario

**Operazioni elementari su numeri binari: somma, sottrazione**

I numeri decimali

Codifica IEEE754 dei numeri reali anche in forma denormalizzata.



## Somma



$$\begin{array}{r} 111 \quad \leftarrow \text{Riporto} \\ 1011 + \\ 110 = \\ \text{-----} \\ 10001 \end{array}$$

Vorrei definire solo l'operazione di somma e non utilizzare la sottrazione



## Numeri negativi: complemento a 1



I numeri negativi sono complementari ai numeri positivi:  $a + (-a) = 0$

Codifica in complemento a 1: il numero negativo si ottiene cambiando 0 con 1 e viceversa.

|           |     |    |                           |
|-----------|-----|----|---------------------------|
| Problema: | 000 | 0  | Doppia codifica per lo 0. |
|           | 001 | 1  |                           |
|           | 010 | 2  |                           |
|           | 011 | 3  |                           |
|           | 100 | -3 |                           |
|           | 101 | -2 |                           |
|           | 110 | -1 |                           |
|           | 111 | 0  |                           |



## Numeri negativi: complemento a 2



I numeri negativi sono complementari ai numeri positivi:  $a + (-a) = 0$

Codifica in complemento a 2: il numero negativo si ottiene cambiando 0 con 1 e viceversa, e sommando 1.

|     |    |                                |
|-----|----|--------------------------------|
| 000 | 0  |                                |
| 001 | 1  | negativo: $110 + 1 = 111 = -1$ |
| 010 | 2  | negativo: $101 + 1 = 110 = -2$ |
| 011 | 3  | negativo: $100 + 1 = 101 = -3$ |
| 100 | -4 |                                |
| 101 | -3 |                                |
| 110 | -2 |                                |
| 111 | -1 |                                |

$$(x_{31} \times -2^{31}) + (x_{30} \times 2^{30}) + (x_{29} \times 2^{29}) + \dots + (x_1 \times 2^1) + (x_0 \times 2^0)$$

**NB La prima cifra è il bit di segno.**



## Perché complemento a 2?



- La rappresentazione in complemento a due deve il suo nome alla proprietà in base alla quale la somma senza segno di un numero di  $n$  bit e del suo complemento è pari a  $2^n$  (peso del bit  $n+1$ )

$$7 + (-7) = \quad \quad \quad 0111 + 1001 = 10000 \quad \quad \quad 2^4$$

$$5 + (-5) = \quad \quad \quad 0101 + 1011 = 10000 \quad \quad \quad 2^4$$

- e quindi il complemento (o negazione) di un numero  $x$  in complemento a due è pari a  $2^n - x$ , ovvero il suo complemento a 2.

$$2^4 - 7 = 1001$$

$$2^4 - 5 = 1011$$



## Doppia negazione



I numeri negativi sono complementari ai numeri positivi:  $a + (-a) = 0$

Segue che  **$-(-a) = +a$**

Codifica in complemento a 2: il numero negativo si ottiene cambiando 0 con 1 e viceversa, e sommando 1.

|    |    |               |
|----|----|---------------|
| 00 | 0  |               |
| 01 | 1  | $10 + 1 = 11$ |
| 10 | -2 |               |
| 11 | -1 |               |

Esempio:

$$-(-2)_{10} = +2_{10}$$

$$-(10)_2 \Rightarrow \text{Complemento a 1} \Rightarrow 01 \Rightarrow \text{Sommo 1 (complemento a 2)} \Rightarrow 10_2 = 2_{10} \text{ c.v.d.}$$



## Sottrazione



Somma i seguenti 2 numeri  $11 + (-13)$ :

$$01011_2 = 11_{10}$$

$$10011_{10} = -13_{10}$$

E' equivalente ad effettuare la differenza:  $11 - 13$ .

$$\begin{array}{r} 00110 \\ 01011 + \\ 10011 = \\ \hline 11110 \end{array} \rightarrow -2_{10}$$



## Sommario



Sistema di numerazione binario

Rappresentazione binaria dell'Informazione

Conversione in e da un numero binario

Operazioni elementari su numeri binari: somma, sottrazione.

**I numeri decimali**

Codifica IEEE754 dei numeri reali anche in forma denormalizzata.



## Numeri decimali rappresentazione in fixed point



Numeri reali per il computer non sono i numeri reali per la matematica!!  
E' meglio chiamarli float (numeri decimali), sono in numero finito.

Dato un certo numero di bit (stringa) per codificare il numero float, esistono due tipi di codifiche possibili:

Rappresentazione in fixed point.

La virgola è in posizione fissa all'interno della stringa.

Supponiamo di avere una stringa di 8 cifre, con virgola in 3a posizione:

$$\begin{aligned} 27,35 &= + | 27,35000 \\ -18,7 &= - | 18,70000 \\ 0,001456 &= + | 00,00145(6) \end{aligned}$$



## Numeri decimali rappresentazione floating point



Rappresentazione come mantissa + esponente.  $E = \sum_k c_k b_k = \sum_{k=-M}^N c_k B^k$

Esempio di **rappresentazioni equivalenti**:

$$627,35 = 62,735 \times 10^1 = \mathbf{6,2735 \times 10^2} = \mathbf{0,62735 \times 10^3} =$$

$$10^1 \times (6 \times 10^0 + 2 \times 10^{-1} + 7 \times 10^{-2} + 3 \times 10^{-3} + 5 \times 10^{-4}) = 10^2 \times \mathbf{1,2735}$$

In grassetto viene evidenziata la rappresentazione normalizzata.

Vengono rappresentati numeri molto grandi e molto piccoli.

Supponiamo di avere una stringa di 8 cifre. 5 per la mantissa, 2 per l'esponente:  $0,001456 = + | 1456 | - | 03 |$

E' in virgola mobile, perchè la prima cifra prima della virgola ha un peso diverso a seconda dell'esponente.



## Conversione base 10 -> base n: algoritmo



Un numero  $x.y$  in base 10 si trasforma in base  $n$  usando il seguente procedimento.

Per la parte intera,  $x$ , si applica l'algoritmo visto in precedenza:

- Dividere il numero  $x$  per  $n$
- Il resto della divisione è la cifra di posto 0 in base  $n$
- Il quoziente della divisione è a sua volta diviso per  $n$
- Il resto ottenuto a questo passo è la cifra di posto 1 in base  $n$
  
- Si prosegue con le divisioni dei quozienti ottenuti al passo precedente fino a che l'ultimo quoziente è 0.
  
- l'ultimo resto è la cifra più significativa in base  $n$



## «Estrazione» delle cifre decimali contenute dopo la virgola



Vogliamo estrarre le cifre di  $0,3672_{\text{dieci}}$ . Porto le cifre alla **sinistra** della virgola:

|                      |                 |                   |
|----------------------|-----------------|-------------------|
| $0,3672 * 10 = 3,67$ | → esamino 0,672 | → 3 decimi        |
| $0,672 * 10 = 6,7$   | → esamino 0,72  | → 6 centesimi     |
| $0,72 * 10 = 7,2$    | → esamino 0,2   | → 7 millesimi     |
| $0,2 * 10 = 2,0$     | → termina       | → 2 decimillesimi |



## Conversione base 10 -> base 2

*“estrazione” delle cifre binarie dopo la virgola*



Vogliamo rappresentare  $0,625_{\text{dieci}}$  in binario:  $0,101_{\text{due}}$

$$\begin{aligned} 0,625 * 2 &= 1,250 && \Rightarrow 1 \\ 0,250 * 2 &= 0,500 && \Rightarrow 0 \\ 0,500 * 2 &= 1,000 && \Rightarrow 1 \\ 0,000 &&& \end{aligned}$$

$$1*2^{-1} + 0*2^{-2} + 1*2^{-3} = \frac{1}{2} + \frac{1}{8} = 0,5 + 0,125 = 0,625$$



## Conversione base 10 -> base n: algoritmo per la parte frazionaria



Un numero  $x,y$  in base 10 si trasforma in base  $n$  usando il seguente procedimento.

Per la parte frazionaria,  $y$ :

- Moltiplicare il numero  $y$  per  $n$
- La prima cifra del risultato coincide con la cifra di posto 1 dopo la virgola.
- Si elimina la parte intera ottenuta e si considera la nuova parte frazionaria.
- La parte frazionaria ottenuta viene moltiplicata per la base  $n$ .
- La prima cifra del risultato coincide con la cifra di posto 2 dopo la virgola.
- Si prosegue con le moltiplicazioni della parte frazionaria fino a quando non diventa 0 o non si **esaurisce la capacità di rappresentazione.**



## Conversione base 10 -> base n: algoritmo per la parte frazionaria



Un numero  $x,y$  in base 10 si trasforma in base  $n$  usando il seguente procedimento.

Per la parte frazionaria,  $y$ :

- Moltiplicare il numero  $y$  per  $n$
- La prima cifra del risultato coincide con la cifra di posto 1 dopo la virgola.
- Si elimina la parte intera ottenuta e si considera la nuova parte frazionaria.
- La parte frazionaria ottenuta viene moltiplicata per la base  $n$ .
- La prima cifra del risultato coincide con la cifra di posto 2 dopo la virgola.
- Si prosegue con le moltiplicazioni della parte frazionaria fino a quando non diventa 0 o non si esaurisce la capacità di rappresentazione.

A./

.unimi.it/



## Errori di approssimazione



Esempio:  $10,75_{10} = 1010,11_2$

Esempio:  $10,76_{10} = 1010,1100001..._2$

10 : 2 => 0  
5 : 2 => 1  
2 : 2 => 0  
1 : 2 => 1  
##### 1010,

0,75 \*2 => 1  
(1),50 \*2 => 1  
(1),00 =>  
##### 11

0,76 \*2 => 1x2<sup>-1</sup>  
(1),52\*2 => 1x2<sup>-2</sup>  
(1),04\*2 => 0x2<sup>-3</sup>  
(0),08 \*2 => 0x2<sup>-4</sup>  
(0),16 \*2 => 0x2<sup>-5</sup>  
(0),32 \*2 => 0x2<sup>-6</sup>  
(0),64 \*2 => 1x2<sup>-7</sup> (2<sup>-7</sup> = 0,0078125)  
(1),28 .....

**Errori di approssimazione:  
arrotondamento e troncamento.**

Con 7 bit di parte fraz, rappresento:  
 $0,5+0,25+0,0078125 = 0,7578125$   
Errore = 0,0011875





## Sommario



Sistema di numerazione binario

Rappresentazione binaria dell'Informazione

Conversione in e da un numero binario

Operazioni elementari su numeri binari: somma, sottrazione.

I numeri decimali

Codifica IEEE754 dei numeri reali anche in forma denormalizzata



## Standard IEEE 754 (1980)



<http://stevehollasch.com/cgindex/coding/ieeefloat.html>

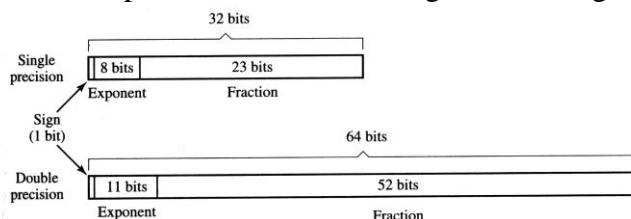


Figure 2-10 Single-precision and double-precision IEEE 754 floating point formats.

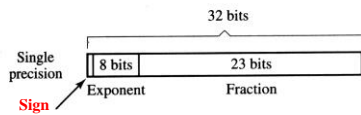
Rappresentazione polarizzata dell'esponente:

Polarizzazione pari a 127 per singola precisione =>  
1 viene codificato come 1000 0000.

Polarizzazione pari a 1023 in doppia precisione.  
1 viene codificato come 1000 0000 000.

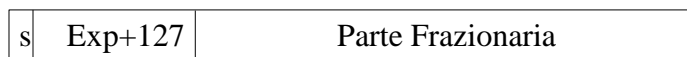


# Codifica mediante lo standard IEEE 754

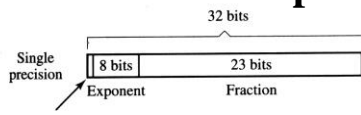


Esempio:  $N = -10,75_{10} = -1010,11_2$

- 1) Normalizzazione:  $\pm 1,xxxxxx \times 2^e$     Esempio:  $-1,01011 \times 2^3$
- 2) Codifica del segno    1 = -    0 = +
- 3) Calcolo dell'esponente, *exp*.



# Calcolo dell'esponente in notazione polarizzata



Esempio:  $N = -10,75_{10} = -1010,11_2 = -1,01011_2 \times 2^3$

**Sign**

Calcolo dell'esponente, *e*, in rappresentazione polarizzata (si considerano solo 254 esponenti sui 256 possibili, compreso tra -126 e +127):

|                 | <b>Codifica</b>        | <b>Exp effettivo del numero</b> |
|-----------------|------------------------|---------------------------------|
|                 | 1111 1111 = 255 →      | Codifica riservata              |
|                 | 1111 1110 = 254 →      | +127                            |
|                 | <b>1000 0010 = 130</b> | <b>+3</b>                       |
|                 | 1000 0001 = 129 →      | +2                              |
|                 | 1000 0000 = 128 →      | +1                              |
| Polarizzazione: | 0111 1111 = 127 →      | 0                               |
|                 | 0111 1110 = 126 →      | -1                              |
|                 | 0000 0001 = 1 →        | -126                            |
|                 | 0000 0000 = 0 →        | Codifica riservata              |



## Configurazioni notevoli nello Standard IEEE 754

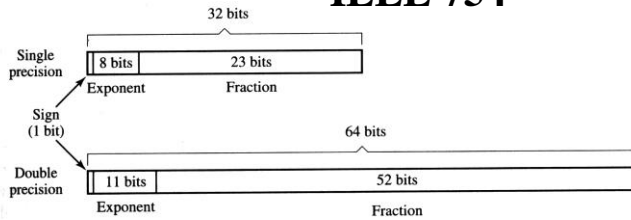


Figure 2-10 Single-precision and double-precision IEEE 754 floating point formats.

Configurazioni notevoli:

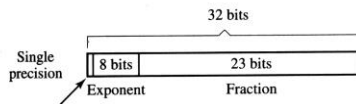
|           |                      |                      |
|-----------|----------------------|----------------------|
| 0         | Mantissa: 0          | Esponente: 00000000  |
| $+\infty$ | Mantissa: 0          | Esponente: 11111111. |
| NaN       | Mantissa: $\neq 0$ . | Esponente: 11111111. |

Range degli esponenti (8 bit):  $1--254 \Rightarrow -126 \leq \text{exp} \leq +127$ .

Numeri float:  $1.0 \times 2^{-126} = 1.175494351 \times 10^{-38} \div 3.402823466 \times 10^{38} = 1.1\dots 11 \times 2^{127}$



## Capacità dello Standard IEEE 754



Range degli esponenti (8 bit):  $1--254 \Rightarrow -126 \leq \text{exp} \leq +127$ .

Minimo float (in valore assoluto!):  $1.0 \times 2^{-126}$

Massimo float:  $1.1111\ 1111\ 1111\ 1111\ 1111\ 1111 \times 2^{+127}$

Capacità float:

Minimo:  $1.175494350822288 \times 10^{-38}$  (1.175494350822288e-038)

Massimo:  $3.402823466385289 \times 10^{38}$  (3.402823466385289e+038)

Discontinuità tra Minimo\_float e 0 il delta è  $1.175494350822288 \times 10^{-38}$

Si può fare di meglio?

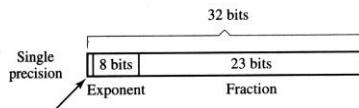
Numero denormalizzato      Mantissa:  $\neq 0$       Esponente: 00000000



# Denormalizzazione nello Standard IEEE 754



Esempio di numero denormalizzato:  $0,000001 \times 2^{-126}$



Range degli esponenti (8 bit):  $1--254 \Rightarrow -126 \leq \text{exp} \leq +127$ .

Minimo float (in valore assoluto!):  $1.0 \times 2^{-126} = 1.17549435082288 \times 10^{-38}$

Tuttavia abbiamo anche la mantissa a disposizione. Se troviamo una codifica per cui possiamo scrivere  $0,0000\ 0000\ 0000\ 0000\ 0000\ 001$ , otteniamo un numero più piccolo (in valore assoluto!) pari a:  $2^{-(23-126)} = 1.401298464324817 \times 10^{-45}$ .

Discontinuità tra Minimo\_float e 0 diventa  $2^{-149} = 1.401298464324817 \times 10^{-45}$

Configurazioni notevoli:

|                       |                      |                      |
|-----------------------|----------------------|----------------------|
| 0                     | Mantissa: 0          | Esponente: 00000000  |
| $+\infty$             | Mantissa: 0          | Esponente: 11111111. |
| NaN                   | Mantissa: $\neq 0$ . | Esponente: 11111111. |
| Numero denormalizzato | Mantissa: $\neq 0$   | Esponente: 00000000  |



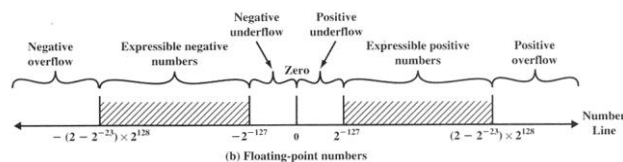
# Risoluzione della codifica dei reali



Distanza tra due numeri vicini.

**Fixed point:** Risoluzione fissa, pari al peso del bit meno significativo.

Esempio su 8 bit:  $+1111,101$  la risoluzione per tutti i numeri sarà:  $1 \times 2^{-3} = 0,125$



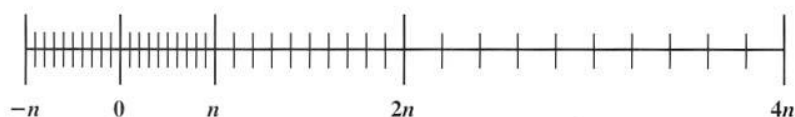
**Floating point:** Risoluzione *relativa* fissa, pari al peso del bit meno significativo.

Il bit meno significativo è in 23a posizione in singola precisione  $\Rightarrow 2^{-23}$ , ne consegue che la risoluzione sarà  $2^{-23}$  volte il numero descritto.

Esempi:

$100, \dots = 1,000 \times 2^2 \Rightarrow$  La risoluzione sarà  $2^{-23} \times 2^2 = 2^{-21}$

$1,0 \times 2^{-126} \Rightarrow$  La risoluzione sarà  $2^{-23} \times 2^{-126} = 2^{-149}$





## Sommario



Sistema di numerazione binario

Rappresentazione binaria dell'Informazione

Conversione in e da un numero binario

Operazioni elementari su numeri binari: somma, sottrazione

I numeri decimali

Codifica IEEE754 dei numeri reali anche in forma denormalizzata.