Lezione 2

Rappresentazione dell'informazione

A. Borghese, F. Pedersini

Dip. Informatica (DI)
Università degli Studi di Milano

Definizione di **rappresentazione** di informazione:

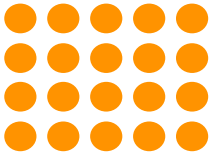
- Corrispondenza tra informazione I e parola P(I) composta da cifre a_i di un alfabeto A di simboli

$$\mathbf{I} \longrightarrow P(\mathbf{I}) = \{a_i\}, \quad a_i \in \mathbf{A}$$

ALFABETO A = $\{a_i\}$: insieme dei simboli per la rappresentazione

Esempi: {A … Z} ; {0 … 9} ; {0 , 1}

- I simboli dell'alfabeto possono essere di varia natura (segni su carta, suoni, livelli di tensione, fori su carta, segnali di fumo…)
- Diversi alfabeti possono essere usati per rappresentare la stessa informazione



Informazione (quantità)

$S = \{0 \cdots 9\}$ $S = \{0,1\}$ $S = \{a \cdots z\}$

20 10100 *venti*

esempi di rappresentazione dell'informazione

Capacità di rappresentazione

- Dato un **alfabeto S**=$\{s_1, s_2, \ldots s_N\}$ composto da **N simboli**, quante "informazioni diverse" (quanta informazione) riesco a rappresentare con una sequenza di **k** cifre di questo alfabeto?

$$C = N^k \qquad \textbf{C: capacità di rappresentazione}$$

Quanti oggetti posso rappresentare con **k bit**?
S={0,1} ➔ N=2 ➔ C = (2 x 2 x 2… x 2) ➔ **$C = 2^k$** oggetti
Quanti oggetti posso rappresentare con **k cifre decimali**?
C = (10 x 10 x 10… x 10) ➔ **$C = 10^k$** oggetti

Problema inverso:

- Date **C** informazioni diverse, quante cifre dell'alfabeto **S** (di **N** simboli) mi servono per poterle rappresentare tutte?

$$k = \log_N C \qquad k \text{ intero} \rightarrow \qquad k = \sup(\log_N C)$$

*Quanti bit mi servono per rappresentare **C** oggetti diversi?*
*Es: **C = 21**: (A,B,…,Z)* **$2^4 = 16 < 21 < 32 = 2^5$** ➔ **5 bit**

Numerazione

*Se l'informazione da rappresentare è una **quantità**,*
*allora la rappresentazione è detta **numerazione***

– Gli **N** elementi della base rappresentano: quantità elementari

B = $\{s_0, s_1, \ldots, s_{N-1}\}$ ➔ quantità: ***0 , 1 , … , N–1***

- Numerazione DECIMALE
  - **B = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} – Base=10**
- Numerazione ESADECIMALE
  - **B = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F} – Base=16**
- Numerazione BINARIA:
  - **B = {0, 1} – Base=2**
- Numerazione OTTALE:
  - **B = {0, 1, 2, 3, 4, 5, 6, 7} – Base=8**

Sistemi di numerazione

Un sistema di numerazione consiste di:

- una base ***B*** di N elementi, rappresentanti le quantità elementari
- una regola di associazione: parola ←→ quantità

- Sistema di **numerazione a conteggio**
  - Ogni cifra rappresenta sempre lo **stessa quantità** (quella elementare)

    es: numerazione romana: **I, V, X, L, C, D, M**

    quantità elementari: 1, 5, 10, 50, 100, 500, 1000

  - **Regola di associazione:** somma delle quantità

$$E:\langle c_k\ c_{k-1}\cdots\ c_0\rangle \qquad E=\sum_{i=0}^{k} val(c_k)$$

Numerazione posizionale

- Sistema di **numerazione a conteggio**
  - Ogni cifra rappresenta sempre lo **stessa quantità** (quella elementare)
  - es: numerazione romana: **I, V, X, L, C, D, M**

    (di valori: 1, 5, 10,50,100,500,1000)

$$E:\langle c_k\ c_{k-1}\cdots\ c_0\rangle \qquad E=\sum_{i=0}^{k} val(c_k)$$

- Sistema di **numerazione posizionale**
  - In un numero a più cifre, ogni cifra rappresenta una **quantità diversa**, a seconda della sua **posizione**

  - **Valore della cifra = quantità elementare x peso(posizione)**

    es: la cifra "1" ha un valore diverso nelle parole **100** e **1000**

$$E:\langle c_k\ c_{k-1}\cdots\ c_0\rangle \qquad E=\sum_{i=0}^{k} val(c_i)\cdot b_i \quad , \quad b_i = N^i$$

Peso: b_i

Codifica posizionale *di un numero*

In numerazione, un alfabeto di ***N elementi*** è detto **BASE**:

Base N: $B_N = \{ b_0, b_1, b_2, \dots, b_{N-1} \}$

rappresentante i valori: $\{ 0, 1, 2, \dots, N-1 \}$

Ciascun numero **E**, può essere rappresentato come combinazione lineare degli elementi della base:

$$E : \langle c_k \; c_{k-1} \cdots \; c_0 \rangle \qquad E = \sum_{i=0}^{k} val(c_i) \cdot b_i \quad , \quad b_i = N^i$$

b_k sono i PESI delle cifre, di valore: $\mathbf{b_k = N^k}$

Esempi di basi e pesi di numerazione:

base 2:	B_2 = {0,1}	valori: {0,1}	pesi: {…16, 8, 4, 2, **1**, ½, ¼, …}
base 10:	B_{10} = {0, 1, …, 9}	valori: {0…9}	pesi: {…100, 10, **1**, 0.1, 0.01, …}
base 3:	B_3 = {□★○},	valori: {0,1,2}	pesi: {… 27, 9, 3, **1**, 1/3, 1/9, …}

Esempi: $12_{10} = 1 \cdot 10^1 + 2 \cdot 10^0$; $100_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 4_{10}$

Conversione ***base 10 → base N***

Algoritmo di conversione di un numero ***x*** in base 10 in base *N:*

- $i = 0$
- Divido (div. intera) il numero x per N
- **Resto** della divisione: cifra i-esima in base N
- $i = i+1$
- **Quoziente** della divisione ➔ x

si prosegue fino a che il quoziente $x = 0$

(l'ultimo resto è la cifra più significativa del numero in base N)

Conversione base 10 → base 2

Esempio: vogliamo rappresentare 1492_{10} in base 2:

```
1492 = 2 x 746 + 0   ← Bit meno significativo
746  = 2 x 373 + 0
373  = 2 x 186 + 1
186  = 2 x 93  + 0
93   = 2 x 46  + 1
46   = 2 x 23  + 0
23   = 2 x 11  + 1
11   = 2 x 5   + 1
5    = 2 x 2   + 1
2    = 2 x 1   + 0
1    = 2 x 0   + 1   ← Bit più significativo
```

senso di lettura

$1492_{10} = 10111010100_2$

Conversione: base n → base 10

Conversione da base N a base 10

Semplice applicazione della formula di codifica posizionale:

Es. numero a **k cifre**, **in base *n***: $\mathbf{E} = \langle c_{k-1}\ c_{k-2}\ \ldots\ c_0 \rangle$

$b_i = n^i$

si trasforma in base 10 calcolando E con la formula:

$$E = \sum_{i=0}^{k-1} c_i \cdot b_i = \sum_{i=0}^{k-1} c_i \cdot n^i, \quad n = 10$$

Esempio:

$$\mathbf{101\ 1101\ 0100}_2 = 1x2^{10} + 0x2^9 + 1x2^8 + 1x2^7 + 1x2^6 + 0x2^5 + 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 0x2^0 =$$

$$1024 + 256 + 128 + 64 + 16 + 4 = \mathbf{1492}_{10}$$

Esercizi

- Dati i numeri decimali: 121331, 2453, 11101, si trasformino in base 3, in base 7, in base 2
- Convertire in base 10 i numeri:
  3456_7 , 121331_5 , 2453_8 , 111010101_2
- Data la base: B = {O⊙★♪#✔}:
  - convertire in tale base il numero: 120_{10}
  - convertire in decimale il numero: ⊙♪✔★

Codifica esadecimale

Codifica esadecimale: base 16

molto utilizzata in alternativa alla codifica binaria

16 simboli: S_{16} = {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

Valori: 0 … 15

Notazioni comunemente utilizzate: $9F_{16}$, 0x9F , 9Fhex

Perché base 16?

$16 = 2^4$ ➔ 1 cifra esadecimale = 4 bit

4 cifre binarie tradotte in 1 cifra esadecimale

0	1	2	3	4	5	6	7
0000	0001	0010	0011	0100	0101	0110	0111

8	9	A/10	B/11	C/12	D/13	E/14	F/15
1000	1001	1010	1011	1100	1101	1110	1111

Conversione binario – esadecimale

Esempi:
conversione di 1101011_{due} in esadecimale:

- 1011_{due} → B_{hex}
- 110_{due} → 6_{hex} (viene aggiunto un "leading" 0)
- 1101011_{due} → $6B_{hex}$

conversione da esadecimale a binario

Ogni cifra esadecimale convertita in un numero binario di 4 cifre:

- 9F hex → $1001\ 1111_2$

0	1	2	3	4	5	6	7
0000	0001	0010	0011	0100	0101	0110	0111

8	9	A/10	B/11	C/12	D/13	E/14	F/15
1000	1001	1010	1011	1100	1101	1110	1111

Rappresentazione binaria di numeri negativi

- Codifica a **modulo e segno**: il primo bit indica il segno, il resto il numero in modulo.
- Codifica in **complemento a 1**: il numero negativo si ottiene cambiando 0 con 1 e viceversa.
- Svantaggi:
  - Ridondanti: doppia codifica per lo 0
  - Scomode per calcolo automatico
- Codifica in **complemento a 2**: il numero negativo si ottiene cambiando 0 con 1 e sommando 1.

Modulo e segno	
dec	bin
0	00
+1	01
0	10
−1	11

Compl. a 1	
dec	bin
0	00
+1	01
−1	10
0	11

Compl. a 2	
dec	bin
0	00
+1	01
−2	10
−1	11

Complemento a 2

Notazione in complemento a 2

Regola di costruzione su ***N*** bit:

Dato numero ***E*** *compreso nel range:* $-2^{N-1} \le E \le +2^{N-1}-1$

- *Se* $E \ge 0$ ➔ *codifico:* E
- *Se* $E < 0$ ➔ *codifico:* $E_{C2} = 2^N + E$

N = 3 $-2^2 \le E \le +2^2-1$	
codifica	valore
0 0 0	**0**
0 0 1	**1**
0 1 0	**2**
0 1 1	**3**
1 0 0	**–4**
1 0 1	**–3**
1 1 0	**–2**
1 1 1	**–1**

Proprietà:

- Il bit più significativo (**MSB**) corrisponde al **segno**
- Comoda inversione di segno
  1. inverto tutti i bit
  2. aggiungo "1"
- Comodo per calcolo automatico: sottrazione fatta come somma.

  A – B = A + (–B)

Sottrazione fra interi

- Sfruttando la rappresentazione dei numeri negativi, gestisco la sottrazione come una somma:

  A – B = A + (–B)

Esempio: 11 – 13 = 11 + (–13)

- Rappresentandoli in complemento a 2:

  $+11_{10}$ → 01011_2
  -13_{10} → 10011_2

```
      1 1  ← riporti
0 1 0 1 1 +
1 0 0 1 1 =
-----------
1 1 1 1 0  → –2₁₀
```

Vantaggio della rappresentazione in complemento a 2:

la somma torna anche con i numeri negativi

➔ posso fare somma e differenza con lo stesso procedimento, quindi con lo stesso **circuito**!

Complemento a 2

Rappresentazione grafica della notazione in complemento a 2:

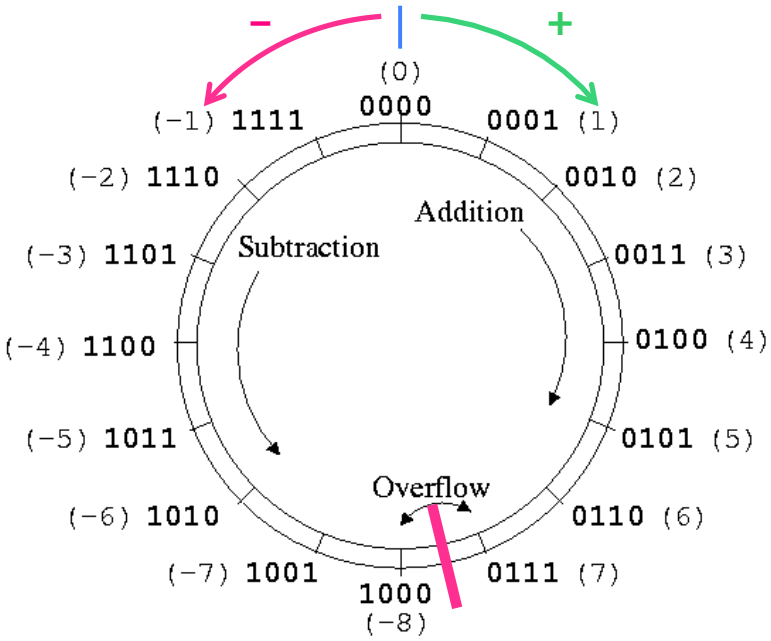
Rappresentazione su un cerchio

+ : senso orario

– : senso antiorario

I calcoli sono corretti se non si oltrepassa la linea rossa: **overflow**

Esempio:
N = **4** → **–8** ≤ E ≤ **+7**



Capacità di rappresentazione interi

In sintesi, quindi:
rappresentazione di **numeri interi in un elaboratore:**

N bit → 2^N valori rappresentabili

Interi "unsigned" (senza segno) → rappr. come numeri binari naturali
Da 0 (00…0) a $2^N - 1$ (111…1)

Interi "signed" (con segno) → rappresentati in complemento a 2
Da -2^{N-1} (100…0) a $2^{N-1} - 1$ (011…1)

Standard C/C++: **`int`** impiega 4 byte (32 bit) → **N=32**

`(signed) int`: -2^{31} (–2.147.483.650) ≤ E ≤ $+2^{31}-1$ (+2.147.483.649)
`unsigned int`: **0** ≤ E ≤ $+2^{32}-1$ (+4.294.967.295)

Conversione di un numero frazionario: x,y da base 10 a base *N*:

$$x,y = x + 0,y$$

Esempio:
3,14 = 3 + 0,14

parte intera

parte frazionaria

Per la parte intera ***x*** → vedi algoritmo precedente

Per la parte frazionaria ***0,y***:

- $i = 1$
- Moltiplico $0, y$ per N
- Parte **intera**: cifra decimale i-esima in base N
- $i = i+1$
- Parte **frazionaria** → $0,y$
- Si prosegue fino a che $y = 0$

Problema: potrebbe **non finire mai**!

Esempio: conversione base 10 → base 2

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

10 : 2 = 5, **0**
5 : 2 = 2, **1**
2 : 2 = 1, **0**
1 : 2 = 0, **1**

0,75 x 2 = 1.5 →**1**
0,5 x 2 = 1.0 →**1**

→ ...,11
(parte frazionaria)

→ 1010,...
(parte intera)

Errori di approssimazione:
arrotondamento , troncamento.

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

0,76 x 2 = 1.52 →**1**
0,52 x 2 = 1.04 →**1**
0.04 x 2 = 0.08 →**0**
0.08 x 2 = 0.16 →**0**
0.16 x 2 = 0.32 →**0**
0.32 x 2 = 0.64 →**0**
0.64 x 2 = 1.28 →**1**
0.28..?

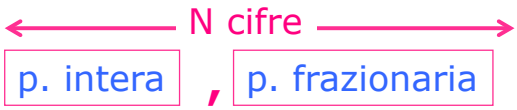
→ ,1100001...
errore = 0.28 • 2^{-8}

Rappresentazione di numeri decimali

Dato un certo numero di cifre **N** (finito) per codificare il numero decimale, esistono due tipi di codifiche possibili:

Virgola fissa (fixed point):

- lascio la virgola dov'è
- date N cifre, le divido tra parte intera e parte frazionaria



Esempio: N=8 cifre decimali → 4 p. intera | 4 p. frazionaria

CAPACITÀ:	MIN: 0,0001	MAX: 9999,9999
RISOLUZIONE:	ΔE = 0,0001	costante!

- Capacità: 9 ordini di grandezza.
- Risoluzione insufficiente per numeri piccoli, ma esagerata per numeri grandi

Rappresentazione di numeri decimali

Virgola mobile (floating point):

- sposto la virgola dove mi fa più comodo (es. 0,xxxxxx) utilizzando la rappresentazione normalizzata (mantissa + esponente)

$$127{,}35 = 0{,}12735 \times 10^{3}$$

mantissa; esponente

- date N cifre, le divido tra mantissa ed esponente

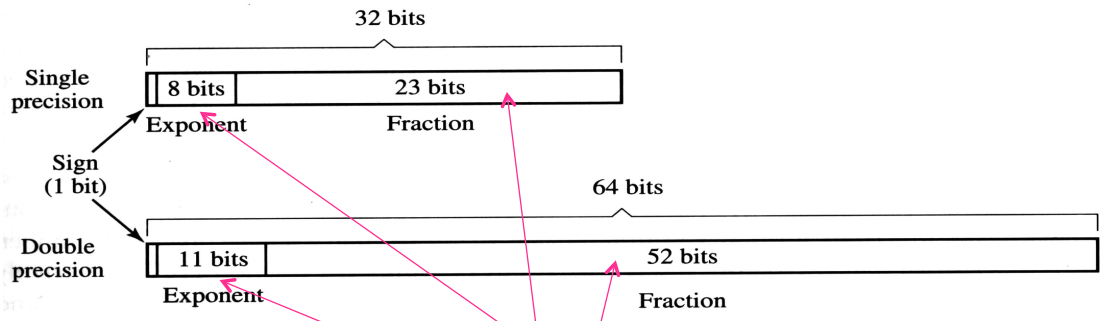
Esempio: N=8 cifre decimali → 6 mantissa | 2 esponente

CAPACITÀ:	Min: $0{,}000001\times10^{-50}$	Max: $0{,}999999\times10^{+49}$
RISOLUZIONE:	$\Delta E = 0{,}000001\times10^{ESP}$	varia con l'esponente

- Capacità molto maggiore che in virgola fissa (100 ord. grandezza)
- La risoluzione è proporzionale all'ordine di grandezza del numero

Lo standard: IEEE-754 (1980)

Standard **IEEE-754**:
rappresentazione binaria di numeri frazionari in virgola mobile



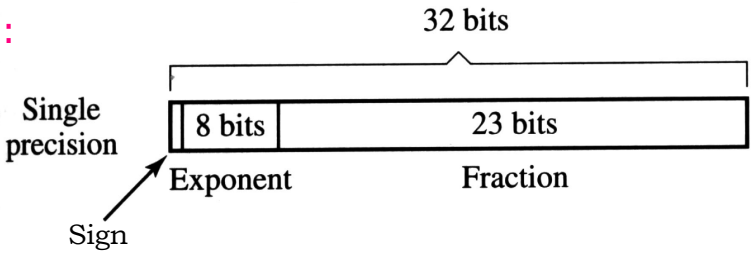
- **SINGLE** precision: **32 bit**; **DOUBLE** precision: **64 bit**
- Memorizzata solo la parte frazionaria della mantissa, in formato

  1, xxxxxx… • 2 EXP

- Rappresentazione polarizzata dell'esponente:
  127 per **singola** precisione (1 viene codificato come: 1000 0000)
  1023 in **doppia** precisione (1 viene codificato come: 100 0000 0000)

Codifica standard IEEE-754

IEEE 754 – Single precision:
(32 bit)



Esempio: N = −10,75

1. Conversione a binario: $-10{,}75_{10} = -1010{,}11_2$
2. Normalizzazione: $\pm 1{,}xxxxxx$ $\quad -1{,}01011 \times 2^3$
3. Codifica del segno: 1 = "−" ; 0 = "+"
4. Calcolo dell'esponente in rappresentazione polarizzata:

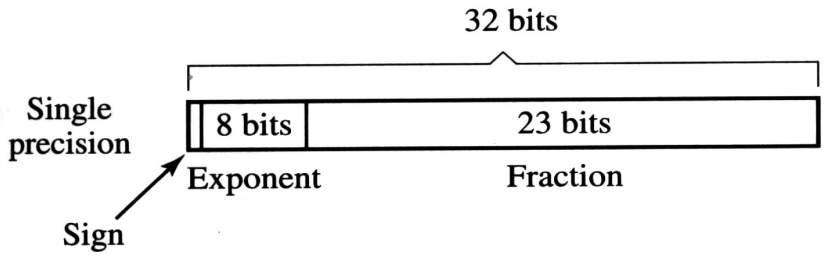
$$e = 3 + 127 = 130_{10} = 10000010_2$$

-10.75_{10} =

1 ___8___ _____ 23 _____

= 1 | 1000 0010 | 01011000 00000000 0000000

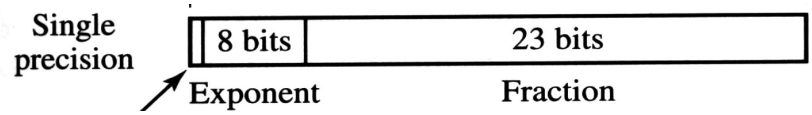
IEEE 754: Configurazioni **notevoli**



Numero	Mantissa	Esponente
0	= 0…0	0000 0000
∞	= 0…0	1111 1111
NaN (*Not-a-Number*)	≠ 0…0	1111 1111
Num. denormalizzato	≠ 0…0	0000 0000

Range esponenti: **1…254** → $-126 \le e \le +127$

Range valori (32 bit): $1{,}00\ldots0 \cdot 2^{-126} \le x \le 1{,}11\ldots1 \cdot 2^{+127}$
$1.175\ldots \cdot 10^{-38} \le x \le 3.4028\ldots \cdot 10^{+38}$



MIN_float: $E_0 = 1{,}00\ldots00 \cdot 2^{-126} = 1.175\ldots \cdot 10^{-38}$

Float successivo: $E_1 = 1{,}00\ldots01 \cdot 2^{-126} = \text{MIN\_float} + 2^{-126-23}$

→ Risoluzione float: $\Delta E = E_1 - E_0 = 2^{-126-23} = 2^{-149} = 1.41298\ldots \cdot 10^{-45}$

***Discontinuità** tra **ZERO** e **MIN_float** !!*

Soluzione: **numeri denormalizzati**

Esponente: 00…0 → si pone pari a: −126 → $\mathbf{2^{-126}}$
Mantissa: $m_1 \ldots m_{23}$ → **0, $m_1 \ldots m_{23}$** (anziché: **1,** $m_1\ldots$)

Numero: 0 , $m_1 \ldots m_{23} \cdot 2^{-126}$

→ MIN_float (denorm.): $0{,}00\ldots01_2 \cdot 2^{-126} = \mathbf{2^{-149} = \Delta E} \approx 1.41_{10} \cdot 10^{-45}$