

# Rappresentazione dell'informazione in un calcolatore

## Informazioni

- testi, numeri interi e reali, immagini, suoni, etc.;

## Come viene rappresentata l'informazione in un calcolatore?

- uso di tecnologia **digitale**: all'interno della CPU le informazioni vengono rappresentate da 2 possibili valori di tensione elettrica {V<sub>high</sub>, V<sub>low</sub>}
- In generale, a seconda del tipo di dispositivo considerato, i valori zero ed uno sono rappresentati:
  - da una tensione elettrica (alta, bassa);
  - da un differente stato di polarizzazione magnetica (positiva, negativa);
  - da luce e buio;
  - etc.
- Unità di informazione nel calcolatore: **bit**.

☞ Ogni informazione viene trasformata nel calcolatore in una sequenza di bit (forma **BINARIA**), cioè in una sequenza di 0 e 1.

00011010...

# Codifica dei numeri

Il sistema di numerazione che utilizziamo si dice **arabico** e fu introdotto in Europa dagli arabi nel Medio Evo.

- È **decimale** (o in base 10): esso rappresenta i numeri tramite sequenze di cifre che vanno da 0 a 9 (dieci cifre).
- È **posizionale**: il peso attribuito ad ogni cifra è funzione della posizione che occupa.

(Esistono anche sistemi **additivi**, in cui ogni unità è rappresentata da un unico simbolo o non-posizionali come quello romano).

☞ I sistemi posizionali consentono di rappresentare numeri grandi con un numero limitato di cifre, e di svolgere su di essi calcoli più efficienti.

## I sistemi di numerazione posizionale sono caratterizzati da una base b e un alfabeto $\alpha$

- **Alfabeto ( $\alpha$ ):** è l'insieme delle cifre disponibili per esprimere i numeri. Ad ogni cifra corrisponde un valore compreso tra 0 e (b-1).  
(Ad esempio, nella numerazione decimale l'alfabeto è  $\alpha=\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ )
- **Base (b):** è il numero degli elementi che compongono l'alfabeto. Ad esempio, nel caso decimale, b=10.

## Numerazione in base p

$$\alpha = \{0, 1, 2, \dots, p-1\}$$
$$b = p$$

un numero generico N in base p è rappresentato da una sequenza di cifre:

$$\mathbf{a_n a_{n-1} \dots a_1 a_0}$$

dove  $\mathbf{a_i} \in \alpha, \forall i=0, \dots, n$ .

( $\mathbf{a_n}$  è la cifra più significativa, mentre  $\mathbf{a_0}$  è la meno significativa)

## Codifica dei Numeri Naturali

Consideriamo l'insieme dei numeri naturali.

- Dato un sistema di numerazione posizionale  $\langle \alpha, b \rangle$ :

$$\alpha = \{0, 1, 2, \dots, p-1\}$$
$$b = p$$

- Sia  $a_n a_{n-1} \dots a_1 a_0$  la codifica di un numero naturale  $N$  in base  $p$ ; allora il valore di  $N$ , in base decimale è dato dalla formula:

$$N_p = a_n \cdot p^n + a_{n-1} \cdot p^{n-1} + \dots + a_1 \cdot p^1 + a_0$$

o, in forma più compatta:

$$N_p = \sum_{i=0, \dots, n} a_i \cdot p^i$$

- Con  $n$  cifre in base  $p$  è possibile rappresentare  $p^n$  numeri naturali diversi da 0 a  $p^n - 1$  (i due limiti si ottengono sostituendo a tutti gli  $n$  coefficienti  $a_i$  0 o  $p-1$  rispettivamente).

Infatti, il numero massimo si ottiene utilizzando la cifra massima ( $p-1$ ) per ogni posizione:

$$(p-1) \cdot p^{n-1} + (p-1) \cdot p^{n-2} + \dots + (p-1) \cdot p^1 + (p-1) =$$
$$= p^n - p^{n-1} + p^{n-1} - p^{n-2} + p^{n-2} - \dots - p + p - 1 = p^n - 1$$

## Esempi

### Conversione di un numero da base $b$ a base decimale

**Esempio 1:** Codifica **decimale**.  $b = 10$ ,  $N_{10} = 5870$

$$5870 = 5 \cdot 10^3 + 8 \cdot 10^2 + 7 \cdot 10^1 + 0 \cdot 10^0$$

**Esempio 2:** Codifica **binaria**.  $b=2$ ,  $\alpha=\{0,1\}$

$N_2 = 101001011$

$$101001011_2 = (1 \cdot 2^8 + 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0)_{10} = (331)_{10}$$

**Esempio 3:** Codifica **ottale**.  $b=8$ ,  $\alpha = \{0, 1, 2, 3, 4, 5, 6, 7\}$

$N_8 = (534)_8$

$$(534)_8 = (5 \cdot 8^2 + 3 \cdot 8^1 + 4)_{10} = (348)_{10}$$

**Esempio 4:** Codifica **esadecimale**:

$b = 16$

$\alpha = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

$N_{16} = B7F$

$$B7F_{16} = (11 \cdot 16^2 + 7 \cdot 16^1 + 15)_{10} = (2943)_{10}$$

## Conversione di un numero naturale in base 10, in base non decimale

La formula

$$N_p = a_n \cdot p^n + a_{n-1} \cdot p^{n-1} + \dots + a_1 \cdot p^1 + a_0$$

si può riscrivere come:

$$N_p = a_0 + p \cdot (a_1 + p \cdot (\dots + p \cdot (a_{n-1} + a_n \cdot p)) \dots)$$

- Eseguendo la **divisione intera** per p:

$$N_p \text{ div } p = (a_1 + p \cdot (\dots p \cdot (a_{n-1} + a_n \cdot p)) \dots) = Q_1$$

$$N_p \text{ mod } p = a_0 \quad [\text{resto della divisione intera}]$$

- Applichiamo la **divisione intera** per p sul risultato  $Q_1$  della divisione precedente:

$$Q_1 \text{ div } p = (a_2 + p \cdot (\dots p \cdot (a_{n-1} + a_n \cdot p)) \dots) = Q_2$$

$$Q_1 \text{ mod } p = a_1 \quad [\text{resto della divisione intera}]$$

- Ripetiamo il procedimento su  $Q_2, Q_3$ , etc. per ottenere le cifre rimanenti  $(a_2, a_3, \dots a_n)$ .

☞ In pratica, il procedimento da seguire è:

Sia N il numero.

1. Si divide N per la nuova base p; sia Q il quoziente ed R il resto.
2. Si converte R nella corrispondente cifra della nuova base p.
3. Si aggiunge la cifra così ottenuta a sinistra delle cifre ottenute in precedenza.
4. Se  $Q = 0$ , fine; Altrimenti poni  $N = Q$  e torna al passo 1.

## Conversione binaria

Si vuole convertire un numero  $N$  (in base 10), nella corrispondente rappresentazione in base 2.

- Applicando il procedimento visto, bisogna effettuare successive divisioni per 2.
- Il risultato è la sequenza di 0 e 1 ottenuti considerando i resti delle divisioni dalla meno significativa alla più significativa.

### Esempio: Convertire in forma binaria $N_{10}=331$

Divisione	Quoziente	Resto ( $a_i$ )
331 : 2	165	1
165 : 2	82	1
82 : 2	41	0
41 : 2	20	1
20 : 2	10	0
10 : 2	5	0
5 : 2	2	1
2 : 2	1	0
1 : 2	0	1

quindi:  $(331)_{10} = (101001011)_2$

### Esempio: Convertire in forma binaria $N_{10}=44$

Divisione	Quoziente	Resto ( $a_i$ )
44 : 2	22	0
22 : 2	11	0
11 : 2	5	1
5 : 2	2	1
2 : 2	1	0
1 : 2	0	1

quindi:  $(44)_{10} = (101100)_2$

## Conversione in base $p \neq 2$

### Esempio: Convertire in forma ottale $N_{10}=44$

Divisione	Quoziente	Resto ( $a_i$ )
44 : 8	5	4
5 : 8	0	5

quindi:  $(44)_{10} = (54)_8$

### Esempio: Convertire in forma esadecimale $N_{10}=44$

Divisione	Quoziente	Resto ( $a_i$ )
44 : 16	2	12 ( $C_{16}$ )
2 : 16	0	2

quindi:  $(44)_{10} = (2C)_{16}$

## Tabella Riassuntiva

Sistema di numerazione			
Decimale	Binario	Ottale	Esadecimale
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

## Conversioni da forma binaria a ottale ed esadecimale

Le rappresentazioni ottali ed esadecimali sono interessanti per la facilità di conversione dalla base due, e viceversa.

- Osserviamo che:

$$8 = 2^3$$

$$16 = 2^4$$

- La conversione da base 2 a base 8 si ottiene scomponendo il numero binario in **triple** di cifre binarie (partendo dalla meno significativa), e per ogni tripla ricavando la corrispondente cifra ottale.

$$(101100)_2 = (101.100)_2$$

$$(101)_2 = (5)_8$$

$$(100)_2 = (4)_8$$

$$\text{quindi } (101100)_2 = (54)_8$$

- La conversione da base 2 a base 16 si ottiene scomponendo il numero binario in **quadruple** di cifre binarie (partendo dalla meno significativa), e per ogni quadrupla ricavando la corrispondente cifra ottale.

$$(101100)_2 = (0010.1100)_2$$

$$(0010)_2 = (2)_{16}$$

$$(1100)_2 = (C)_{16}$$

$$\text{quindi } (101100)_2 = (2C)_{16}$$

## Aritmetica Binaria: operazioni sui naturali

### Definizione delle operazioni aritmetiche elementari

A	B	r/p	riporto	A + B	prestito	A - B
0	0	0	0	0	0	0
0	1	0	0	1	1	1
1	0	0	0	1	0	1
1	1	0	1	0	0	0
0	0	1	0	1	1	1
0	1	1	1	0	1	0
1	0	1	1	0	0	0
1	1	1	1	1	1	1

(r/p esprime il riporto/prestito della colonna precedente)

## Somma di due numeri binari naturali

Viene eseguita incolonnando i numeri e sommando tra loro i bit incolonnati, partendo dai meno significativi, in ordine di peso crescente.

☞ Per la somma di due numeri positivi di lunghezza K possono essere necessari K+1 posti. Se sono disponibili solo K cifre si genera un errore di **overflow** (o trabocco).

### Esempio:

$$(11011)_2 + (00110)_2$$

$$\begin{array}{r} 11011 + (27) \\ 00110 = (6) \\ \hline 100001 \quad (33) \end{array}$$

## Sottrazione di numeri binari naturali

Viene eseguita incolonnando i numeri e sottraendo tra loro i bit incolonnati, partendo dai meno significativi, in ordine di peso crescente.

**Ipotesi:** si suppone che si generi sempre un numero positivo

$$\begin{array}{r} 1101 - (12) \\ 0011 = (3) \\ \hline 1001 \quad (9) \end{array}$$

## Moltiplicazione di numeri binari naturali

Si utilizza la stessa tecnica usata anche per i numeri in base 10 (somma e scorrimento):

$$A = (1011)_2 = (11)_{10}$$

$$B = (1101)_2 = (13)_{10}$$

$$A \times B = 1011 \times 1101 =$$

$$\begin{array}{r} 1011 \times \\ 1101 \\ \hline 1011 + \\ 0000_ + \\ 1011_ + \\ 1011_ \\ \hline 10001111 \end{array} \quad (143)$$

## Divisione di numeri binari naturali

Si usa la tecnica usata anche per i numeri in base 10 (differenza e scorrimento).

$$A = (101101)_2 = (45)_{10}$$

$$B = (11)_2 = (3)_{10}$$

Calcolare A/B:

$$\begin{array}{r|l} 101101 & 11 \\ \hline 00 & 01111 \\ 101 - & \\ \hline 011 & \\ 0101 - & \\ \hline 011 & \\ 100 - & \\ \hline 011 & \\ 011 - & \\ \hline 011 & \\ \hline 000 & \end{array}$$

Il risultato della operazione A / B è quindi  $(1111)_2 = (15)_{10}$



## Esercizi

Effettuare le seguenti operazioni aritmetiche tra numeri binari naturali, ipotizzando di lavorare con un elaboratore con lunghezza di parola (*word*) pari a un byte:

- **Differenza:**  $A - B$ ,  $A = (35)_{10}$ ,  $B = (22)_{10}$

$$(35)_{10} = (23)_{16} = (0010\ 0011)_2$$

$$(22)_{10} = (16)_{16} = (0001\ 0110)_2$$

$$\begin{array}{r} 0010\ 0011 \\ - \end{array}$$

$$\begin{array}{r} 0001\ 0110 \\ \hline \end{array} =$$

$$0000\ 1101 \text{ Prestito: } 0$$

$$\text{Risultato: } A - B = (0000\ 1101)_2 = (0D)_{16} = (13)_{10}$$

- **Somma:**  $A + B$ ,  $A = (42)_{10}$ ,  $B = (31)_{10}$

$$(42)_{10} = (2A)_{16} = (0010\ 1010)_2$$

$$(31)_{10} = (1F)_{16} = (0001\ 1111)_2$$

$$\begin{array}{r} 0010\ 1010 \\ + \end{array}$$

$$\begin{array}{r} 0001\ 1111 \\ \hline \end{array} =$$

$$0100\ 1001$$

$$\text{Risultato: } A + B = (0100\ 1001)_2 = (49)_{16} = (73)_{10}$$

- **Moltiplicazione:**  $A \times B$ ,  $A = (7)_{10}$ ,  $B = (12)_{10}$

$$(7)_{10} = (7)_{16} = (0000\ 0111)_2$$

$$(12)_{10} = (0C)_{16} = (0000\ 1100)_2$$

$$\begin{array}{r} 00000111 \times \\ 00001100 = \\ \hline 00000000 + \\ 00000000\_ + \\ 00000111\_ + \\ 000000111\_ \\ \hline 01010100 \end{array}$$

$$\text{Risultato: } (0101\ 0100)_2 = (54)_{16} = (84)_{10}$$

- **Divisione:**  $A / B$ ,  $A = (23)_{10}$ ,  $B = (7)_{10}$

$$(23)_{10} = (17)_{16} = (0001\ 0111)_2$$

$$(7)_{10} = (7)_{16} = (0000\ 0111)_2$$

$$\begin{array}{r} 1011 \quad | \quad 111 \\ \hline 000 \quad 011 \\ \hline 1011 - \\ \hline 0111 \\ \hline 01001 - \\ \hline 0111 \\ \hline 0010 \end{array}$$

$$\begin{array}{l} \text{Risultato: } \text{Quoziente} = (0000\ 0011)_2 = (3)_{10} \\ \text{Resto} = (0000\ 0010)_2 = (2)_{10} \end{array}$$

## Numeri relativi

Consideriamo l'insieme dei numeri relativi (**interi**)

$$\mathbb{Z} = \{-\infty, \dots, -1, 0, +1, \dots, +\infty\}$$

Vogliamo rappresentare gli elementi di questo insieme in forma binaria.

Avendo sempre un numero limitato di bit, possiamo utilizzarne uno per la rappresentazione del segno (+ o -).

### Rappresentazione con modulo e segno

Il primo bit di un numero intero viene utilizzato come bit di segno (0 positivo, 1 negativo). Gli altri bit indicano il modulo (valore assoluto) del numero.

#### Ad esempio:

- parole di 5 bit:

$$+5 = 00101$$

$$-10 = 11010$$

- parole a 16 bit:

$$+13 = 0000000000001101$$

$$-13 = 1000000000001101$$

## Rappresentazione con modulo e segno

- Tramite  $m$  cifre in base 2 è possibile rappresentare  $2^{m-1}$  numeri diversi, da  $-(2^{m-1} - 1)$  a  $+(2^{m-1} - 1)$ .

#### Ad esempio:

$$m = 3, \quad \text{da } -(2^2 - 1) = -3 \text{ a } +(2^2 - 1) = 3$$

$$m = 16, \quad \text{da } -32767 \text{ a } +32767$$

$$m = 32, \quad \text{da } -2147483647 \text{ a } +2147483647$$

☞ Abbiamo due rappresentazioni diverse per lo zero:

$$-0 = 10000$$

$$+0 = 00000$$

#### Ad esempio:

$m = 2 \rightarrow$  posso rappresentare  $2^2 - 1 = 3$  numeri:

Valore Binario	Valore Decimale
00	+0
01	+1
10	-0
11	-1

## Rappresentazione in complemento

Per semplificare le operazioni su interi (con segno) si adotta una rappresentazione dei **numeri negativi in complemento**.

### Complemento alla base

Dato un numero  $X$  in base  $b$  di  $n$  cifre, il complemento alla base è definito come:

$$b^n - X$$

### Esempi:

- $n = 2, b = 10, X = 64$   
Il complemento a 10 di 64 è  $10^2 - 64 = 36$
- $n = 5, b = 2, X = 01011$   
Il complemento a 2 di  $X$  è  $2^5 - X = 100000 - 01011 = 10101$

☞ In pratica, il principio è che la somma del numero più il suo complemento, dia una stringa di  $n$  cifre a zero.

### Complemento alla base -1

Dato un numero  $X$  in base  $b$  di  $n$  cifre, il complemento alla base -1 è definito come:

$$(b^n - 1) - X$$

### Esempio:

- $n = 5, b = 2, X = 01011$   
Il complemento a 1 di  $X$  è

$$(2^n - 1) - X = 11111 - 01011 = 10100.$$

☞ Osserviamo che il risultato è la sequenza di cifre che si ottiene complementando a 1 ogni singolo bit.

### Riassumendo:

Supponiamo di voler rappresentare un numero  $X$  tramite  $n$  cifre binarie.

- la rappresentazione in **complemento a 2** si ottiene sottraendo  $X$  da  $2^n$
- la rappresentazione in **complemento a 1** si ottiene sottraendo  $X$  da  $2^n - 1$ .

### In pratica:

- Il **complemento a 1** di un numero binario X (rappresentato come intero positivo) si ottiene complementando tutti i bit (cioè scambiando 0 con 1, e viceversa).
- Il **complemento a 2** si ottiene prima calcolando il complemento a 1, e poi sommandovi 1. (Infatti, si può scrivere l'operazione  $2^n - X$  come  $2^n - 1 - X + 1$ )

## Rappresentazione in complemento

**Esempio:**  $n = 4$

Intero	Modulo	Segno + modulo	Compl. a 1	Compl. a 2
-7	0111	1111	1000	1001
-6	0110	1110	1001	1010
-5	0101	1101	1010	1011
-4	0100	1100	1011	1100
-3	0011	1011	1100	1101
-2	0010	1010	1101	1110
-1	0001	1001	1110	1111
-0	0000	1000	1111	---

- **Rappresentazione in complemento a 1:** I numeri positivi sono rappresentati dal loro modulo ed hanno il bit più significativo a zero. I numeri negativi si ottengono complementando a 1 i corrispondenti moduli, segno compreso. Pertanto hanno il primo bit sempre a 1.
- **Rappresentazione in complemento a 2:** I numeri positivi sono rappresentati dal loro modulo ed hanno il bit più significativo a zero. I numeri negativi si ottengono complementando a 2 i corrispondenti moduli, segno compreso. Pertanto hanno il bit del segno sempre a 1 (in questo caso si ha una sola rappresentazione dello zero: 00000).
- Su 5 bit si possono rappresentare i numeri da -16 (10000) a +15 (01111).
- Il numero più piccolo rappresentabile (-16) non ha il corrispettivo positivo (per rappresentare +16 occorrerebbero 6 bit).

### Esercizi:

- Si rappresentino i seguenti numeri in complemento a 2 avendo 8 bit a disposizione:

	Modulo		Compl. a 1		Compl. a 2
-34	⇒ 0010 0010	⇒	1101 1101	⇒	1101 1110
-25	⇒ 0001 1001	⇒	1110 0110	⇒	1110 0110
+46	⇒	⇒		⇒	0010 1110
-23	⇒ 0001 0111	⇒	1110 1000	⇒	1110 1001
+66	⇒	⇒		⇒	0100 0010
+72	⇒	⇒		⇒	0100 1000
-120	⇒ 0111 1000	⇒	1000 0111	⇒	1000 1000
-10	⇒ 0000 1010	⇒	1111 0101	⇒	1111 0110

- Interpretare la sequenza di bit 1001 0001 1100 0101 come:

- a) numero naturale
- b) numero relativo in modulo e segno
- c) numero relativo in complemento a due

a) Interpretando la sequenza come **numero naturale**:

$$1001\ 0001\ 1100\ 0101 = 91C5_{16} = \\ = 9 \times 16^3 + 1 \times 16^2 + 12 \times 16 + 5 = 37317$$

b) Interpretando la sequenza come numero relativo in modulo e segno:

$$1001\ 0001\ 1100\ 0101 = -11C5_{16} = \\ = -(1 \times 16^3 + 1 \times 16^2 + 12 \times 16 + 5) = -4549$$

c) Interpretando la sequenza come numero relativo in complemento a due:

$$1001\ 0001\ 1100\ 0101$$

- ricaviamo il complemento a 1:

$$1001\ 0001\ 1100\ 0101 - 1 = 1001\ 0001\ 1100\ 0100$$

- complementiamo i bit:

$$-0110\ 1110\ 0011\ 1011 = \\ -6E3B_H = -(6 \times 16^3 + 14 \times 16^2 + 3 \times 16 + 11) = \\ -28219$$

## Operazioni aritmetiche sui numeri relativi

### Somma algebrica

Sfrutta la rappresentazione in complemento a 2.

Si sommano i numeri per colonna incluso il bit del segno, ignorando l'eventuale riporto sul bit del segno.

### Esempi:

0 000101 (+5)	0 000101 (+5)
0 001000 (+8)	1 111000 (-8)
0 001101 (+13)	1 111101 (-3)
1 111011 (-5)	1 111011 (-5)
0 001000 (+8)	1 111000 (-8)
(1)0 000011 (+3)	(1)1 110011 (-13)

- Consideriamo 8 bit: (il numero è compreso fra -128 e 127):

$$70 = 01000110$$

$$70 + 70 =$$

$$\begin{array}{r} 01000110 + \\ 01000110 = \\ 10001100 \text{ (OVERFLOW)} \end{array}$$

$$\begin{array}{l} -70 - 70 \text{ cioè } (-70) + (-70) = \\ -70 \text{ in complemento a 2: } 10111010 \end{array}$$

$$\begin{array}{r} 10111010 + \\ 10111010 = \\ (1)01110100 \text{ (OVERFLOW)} \end{array}$$

☞ Si può verificare **overflow** solo quando gli operandi hanno lo stesso segno (l'operazione produce un numero di segno opposto!!).

### Sottrazione tra numeri relativi

Se i numeri sono rappresentati in complemento a 2 l'operazione di sottrazione si effettua mediante somma (vantaggio della rappresentazione in complemento a 2).

### Esercizio:

Effettuare le seguenti operazioni, supponendo di operare con una rappresentazione dei numeri su un byte e in complemento a due:

$$\begin{array}{lll} -34 + 25 & -34 + 46 & -23 - 34 \\ 66 + 72 & -120 - 10 & \end{array}$$

### Soluzione

I valori indicati risultano così rappresentati:

$$\begin{array}{ll} -34 & 1101 \ 1110 \\ +25 & 0001 \ 1001 \\ +46 & 0010 \ 1110 \\ -23 & 1110 \ 1001 \\ +66 & 0100 \ 0010 \\ +72 & 0100 \ 1000 \\ -120 & 1000 \ 1000 \\ -10 & 1111 \ 0110 \end{array}$$

Da cui:

$$1101 \ 1110 \ -34$$

$$\underline{0001 \ 1001 \ +25}$$

$$1111 \ 0111 = -00001001 = -9$$

$$1101 \ 1110 \ -34$$

$$\underline{0010 \ 1110 \ +46}$$

$$(1)0000 \ 1100 = +00001100 = +12$$

1101 1110 -34

1110 1001 -23

(1)1100 0111 = -00111001 = -57

0100 0010 +66

0100 1000 +72

1000 1010 = -01110110 = -118  
⇒ OVERFLOW

1000 1000 -120

1111 0110 -10

(1)0111 1110 = +01111110 = +126  
⇒ OVERFLOW

## Moltiplicazione e divisione tra numeri relativi

- Utilizzare i numeri in valore assoluto ed utilizzare le operazioni viste sui numeri naturali (somma/differenza e scorrimento).
- Il segno si determina in base al segno degli operandi.

☞ L'applicazione delle operazioni direttamente sui numeri in complemento è in generale scorretta.

Infatti:  $(-3) \times (+3)$

I valori indicati risultano così rappresentati con 4 bit in complemento a 2:

-3 = - 0011 = 1101  
+3 = 0011

Il risultato del calcolo, -9, non è rappresentabile su soli 4 bit. Supporremo disponibile allo scopo una parola di un byte.

```
  1101 ×
  0011 =
-----
  1101 +
  1101_ +
  0000__ +
  0000___
-----
 00100111
```

che, su un byte, è interpretabile come numero positivo, di valore 39.



## Operazione di shift

### Shift verso sinistra

- Dalla rappresentazione dei numeri discende immediatamente che lo scorrimento verso sinistra di tutte le cifre del numero di una posizione con l'inserimento di uno zero nella posizione di destra equivale a moltiplicare il numero per la base.

☞ Lo scorrimento di  $k$  posizioni verso sinistra equivale a moltiplicare il numero per  $b^k$ .

### Shift verso destra

- Lo scorrimento (shift) verso destra di tutte le cifre del numero di una posizione con l'inserimento di uno zero nella posizione di sinistra, equivale a dividere il numero per la base (cioè a moltiplicare il numero per  $b^{-1}$ ).

☞ Lo scorrimento di  $k$  posizioni verso destra equivale a moltiplicare il numero per  $b^{-k}$ .

## Numeri frazionari

Sono numeri reali compresi fra zero e 1;

### Rappresentazione dei numeri frazionari

Sia:

$$\alpha = \{0, 1, 2, \dots, p-1\}$$
$$b = p$$

Dato un numero frazionario  $N$ , la sua rappresentazione in base  $p$  è data da una sequenza di cifre:

$$0.a_1 a_2 a_3 \dots a_n$$

dove  $a_i \in \alpha, \forall i=0, \dots, n$ .

Il valore di  $N$  è dato dalla formula:

$$N_p = a_1 \cdot p^{-1} + a_2 \cdot p^{-2} + \dots + a_n \cdot p^{-n}$$

o, in forma più compatta:

$$N_p = \sum_{i=1, \dots, n} a_i \cdot p^{-i}$$

### Ad esempio:

- Base decimale:  $p = 10$ ,  $N_{10} = 0.587$

$$(0.587)_{10} = 5 \times 10^{-1} + 8 \times 10^{-2} + 7 \times 10^{-3}.$$

- Binario:  $p = 2$ ,  $N_2 = 0.1011$

$$(0.1011)_2 = (1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4})_{10} = (0.6975)_{10}$$

## Conversione di un numero decimale frazionario in base $\neq 10$

$$N_p = a_{-1} \cdot p^{-1} + a_{-2} \cdot p^{-2} + \dots + a_{-n} \cdot p^{-n}$$

- Moltiplico  $N_p$  per  $p$ :

$$N_p \cdot p = a_{-1} + a_{-2} \cdot p^{-1} + \dots + a_{-n} \cdot p^{-n+1}$$

Il risultato è un **numero reale**  $>1$ :

- Parte **intera** ( $N_p \times p$ ) =  $a_{-1}$
- Parte **frazionaria** ( $N_p \times p$ )  
 $= a_{-2} \times p^{-1} + \dots + a_{-n} \times p^{-n+1} = M_1$

- Applichiamo ancora la **moltiplicazione** per  $p$  sulla parte frazionaria  $M_1$  ottenuta nella divisione precedente:

- Parte **intera** ( $M_1 \times p$ ) =  $a_{-2}$
- Parte **frazionaria** ( $M_1 \times p$ )  
 $= a_{-3} \times p^{-1} + \dots + a_{-n} \times p^{-n+2} = M_2$

- Ripetiamo il procedimento su  $M_3$ ,  $M_4$ , etc. per ottenere le cifre rimanenti ( $a_{-3}$ ,  $a_{-4}$ , ..  $a_n$ ).

☞ In pratica, il procedimento da seguire è:

**Sia N il numero frazionario.**

- 1. Si moltiplica N per la nuova base; sia I la parte intera e M la parte frazionaria.**
- 2. Si converte I nella corrispondente cifra della nuova base.**
- 3. Si aggiunge la cifra così' ottenuta a destra delle cifre ottenute in precedenza (la prima cifra immediatamente a destra della virgola)**
- 4. Se F=0 (oppure si sono ottenute le cifre richieste) fine; Altrimenti poni N = F e torna al passo 1.**

## Conversione binaria di numeri frazionari

- Si ottiene effettuando successive moltiplicazioni per due.
- Il risultato è la sequenza di zeri e uni ottenuti considerando le parti intere delle moltiplicazioni dalla più significativa alla meno significativa.

### Esempi:

- Convertire in base 2:  $N_{10} = 0.875$

Moltiplicazione	Parte Frazionaria	Parte Intera ( $a_i$ )
$0.875 \times 2$	0.75	1 ( $a_1$ )
$0.75 \times 2$	0.5	1 ( $a_2$ )
$0.5 \times 2$	0	1 ( $a_3$ )

quindi  $(0.875)_{10} = (0.111)_2$

- Convertire in base 8:  $N_{10} = 0.875$

Moltiplicazione	Parte Frazionaria	Parte Intera ( $a_i$ )
$0.875 \times 8$	0	7

quindi  $(0.875)_{10} = (0,7)_8$

- Convertire in base 16:  $N_{10} = 0.875$

Moltiplicazione	Parte Frazionaria	Parte Intera ( $a_i$ )
$0.875 \times 16$	0	14 (E) <b>16</b>

quindi  $(0.875)_{10} = (0.E)_{16}$

## Conversione di numeri frazionari in base non decimale

- Non sempre si ottiene una conversione esatta: ad esempio convertiamo  $(0.8)_{10}$  in binario.

Moltiplicazione	Parte Frazionaria	Parte Intera ( $a_i$ )
$0.8 \times 2$	0.6	1
$0.6 \times 2$	0.2	1
$0.2 \times 2$	0.4	0
$0.4 \times 2$	0.8	0
$0.8 \times 2$	0.6	1
...	...	...

quindi  $(0.8)_{10} = (0.\underline{1100})_2$  **periodico**

- Uno stesso numero può quindi avere un numero finito di cifre in una base ed un numero infinito in un'altra.
- Per questo, nella rappresentazione interna può essere introdotto un **errore di troncamento**.

## Rappresentazione dei numeri reali

Consideriamo l'insieme  $\mathcal{R}$  dei numeri reali.

Ogni elemento di  $\mathcal{R}$ , in generale, può essere espresso come somma di un intero con un numero frazionario.

In pratica, un numero reale è individuato univocamente da:

- una parte intera I
- una parte frazionaria F

### Rappresentazione dei reali in virgola fissa

Un numero prefissato di cifre viene dedicato alla parte intera ed a quella frazionaria (**rappresentazione in virgola fissa**).

Quindi:

- Si calcola la rappresentazione della parte intera nella base data con la formula:

$$N_p = a_n \cdot p^n + a_{n-1} \cdot p^{n-1} + \dots + a_1 \cdot p^1 + a_0$$

- Si calcola la rappresentazione della parte frazionaria nella base data (tenendo conto del numero di cifre disponibili) con la formula:

$$N_p = a_{-1} \cdot p^{-1} + a_{-2} \cdot p^{-2} + \dots + a_{-n} \cdot p^{-n}$$

- Si giustappongono le due rappresentazioni:  
<rappresentazione I> . <rappresentazione F>

### Ad esempio:

$331.6975_{10} = 00101001011.10110_2$

(riservando 11 bit per la parte intera e 5 per quella frazionaria).

☞ La precisione è variabile e può essere scarsa per numeri di valore prossimo allo zero.

## Rappresentazione dei numeri reali in virgola mobile

- Ad un numero reale  $r$  vengono associati due numeri:
  - $m$  = **mantissa**
  - $n$  = **esponente** (o caratteristica)

### Mantissa

- È un numero frazionario con segno. Il suo valore è quindi compreso nell'intervallo  $]-1, +1[$ .
- Mantissa **normalizzata**: se la prima cifra dopo la virgola è diversa da 0. Il valore assoluto della mantissa, in questo caso,  $\in [1/p, 1[$

### Esponente

- È un intero con segno.

☞ la mantissa e l'esponente sono legati dalla relazione:

$$r = m \times b^n$$

dove  $b$  è un numero intero che indica una base utilizzata per la notazione esponenziale (in generale, se  $p$  è la base del sistema di numerazione,  $p = b$ ).

## Rappresentazione in virgola mobile

### Esempi:

- $p = b = 10, r = -331.6975$

$r$  si rappresenta con  $m = -0.3316975$  e  $n = 3$

- Conversione in binario, virgola mobile ( $p = b = 2$ ):  
 $r = 12.5_{10}$

$$12.5_{10} = 1100.1_2$$

$$m = 0.11001_2 \quad n = 100_2$$

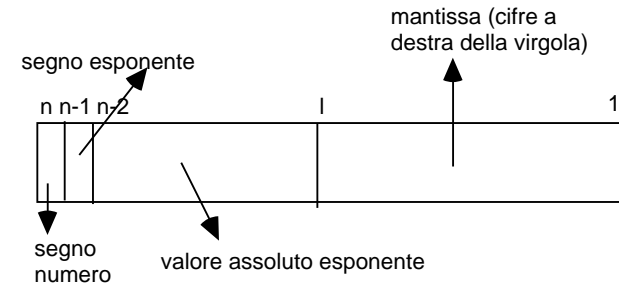
- Conversione in binario, virgola mobile ( $p = b = 2$ ):  
 $r = 26.5_{10}$

$$26.5_{10} = 11010.1_2$$

$$m = 0.110101_2 \quad n = 101_2$$

## Rappresentazione binaria in virgola mobile

Per ogni numero reale vengono utilizzati  $n$  bit:



### Ad esempio:

Supponiamo di avere a disposizione **32 bit**, di cui:

- 1 bit per il segno della mantissa;
- 1 bit per il segno dell'esponente;
- 8 bit per il valore assoluto dell'esponente;
- 22 bit per la mantissa.

⇒ Rappresentazione in virgola mobile di  $r = 26.5$

$$r = 0.110101_2 \times 2^{101_2}$$

esponente			mantissa																				
0	0	00000101	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

⇒ Rappresentazione in virgola mobile di  $r = 0.3125$

$$r = 0.101_2 \times 2^{-12}$$

esponente			mantissa																				
0	1	00000001	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

## Rappresentazione binaria in virgola mobile

- Poiché la mantissa comincia sempre con 1, si può utilizzare questo come bit di segno.
- Invece di riservare un bit per il segno dell'esponente, in alcuni casi si adotta la rappresentazione in complemento a 2 dell'esponente.

### Esempio: complemento a 2 per l'esponente

32 bit:

- 1 bit segno mantissa;
  - 8 bit valore dell'esponente (in compl. a 2);
  - 23 bit mantissa.
- $r = 26.5 = 0.110101_2 \times 2^{101_2}$

esponente    mantissa	
0	00000101   1101010000000000000000

- $r = 0.3125 = 0.101_2 \times 2^{-1_2}$

esponente    mantissa	
0	11111111   1010000000000000000000

## Esercizio

Rappresentare su 16 bit (di cui 8 di esponente in complemento a due, uno per il segno e 7 di mantissa) i seguenti valori:

0.75      0.1      13.33

## Soluzione

a)  $r=0.75$

Parte **intera**: 0

Parte **frazionaria**: algoritmo delle moltiplicazioni successive. Si ha:

$$0.75 \times 2 = 1.5 \Rightarrow F = 0.5, I = 1$$

$$0.5 \times 2 = 1.0 \Rightarrow F = 0.0, I = 1$$

$$0.0 \times 2 = 0.0 \Rightarrow F = 0.0, I = 0$$

...

Il valore richiesto è dunque:

$$\langle \text{parteIntera} \rangle . \langle \text{parteFrazionaria} \rangle = 0.11$$

$\Rightarrow$  Non è necessaria alcuna normalizzazione.

Rappresentazione finale di 0.75:

esponente    segno    mantissa	
00000000	0   1100000

b)  $r=0.1$

Parte **intera**: 0

Parte **frazionaria**: algoritmo delle moltiplicazioni successive. Si ha:

$$0.1 \times 2 = 0.2 \Rightarrow F = 0.2, I = 0$$

$$0.2 \times 2 = 0.4 \Rightarrow F = 0.4, I = 0$$

$$0.4 \times 2 = 0.8 \Rightarrow F = 0.8, I = 0$$

$$0.8 \times 2 = 1.6 \Rightarrow F = 0.6, I = 1$$

$$0.6 \times 2 = 1.2 \Rightarrow F = 0.2, I = 0$$

...

$\Rightarrow$  rappresentazione **approssimata** poiché si ottiene un numero periodico.

$$0.1_{10} \Rightarrow 0.00011 = 0.0001100110011...$$

• Normalizzando:

- mantissa = 0.1100110

- esponente =  $-(11) = 11111100 + 1 = 11111101$

esponente	segno	mantissa
11111101	0	1100110

Si noti che la mantissa viene troncata al numero disponibile di bit *solo dopo aver normalizzato*.

$\Rightarrow$  C'è di un *errore di troncamento*: il numero realmente rappresentato non è 0.1, ma:

$$0.110011 \times 2^{-3} = (1/2 + 1/4 + 1/32 + 1/64) \times 1/8 = 0.099609375$$

c)  $r=13.33$

Parte **intera**:  $13 = (1101)_2$

Parte **frazionaria**: 0.33 con algoritmo delle moltiplicazioni successive:

$$0.33 \times 2 = 0.66 \Rightarrow F = 0.66, I = 0$$

$$0.66 \times 2 = 1.32 \Rightarrow F = 0.32, I = 1$$

$$0.32 \times 2 = 0.64 \Rightarrow F = 0.64, I = 0$$

$$0.64 \times 2 = 1.28 \Rightarrow F = 0.28, I = 1$$

$$0.28 \times 2 = 0.56 \Rightarrow F = 0.56, I = 0$$

$$0.56 \times 2 = 1.12 \Rightarrow F = 0.12, I = 1$$

...

quindi:  $r = 13.33 = 1101.010101...$

• Normalizzando:

- mantissa = 0.1101010101...

- esponente =  $+4 = 100$

esponente	segno	mantissa
00000100	0	1101010

Valutiamo l'errore: il numero realmente rappresentato vale:

$$0.1101010 \times 2^4 = (1/2 + 1/4 + 1/16 + 1/64) \times 16 = 53/64 \times 16 = 13.25$$



## Precisione nella rappresentazione dei numeri reali

Si può, in generale, osservare che:

- Quanto maggiore è il numero di bit riservati alla mantissa tanto maggiore è il numero di cifre significative che possono essere memorizzate (precisione).
- Quanto maggiore è il numero di bit riservati all'esponente tanto maggiore è l'ordine di grandezza della cifra che può essere rappresentata.

### Precisione

È data dal numero di cifre in base 10 rappresentabili con la mantissa.

### Ad esempio:

Se la mantissa è rappresentata da 20 bit, la precisione è di 6 cifre (max valore rappresentabile dalla mantissa  $2^{20}-1$  circa  $= 10^6 = 1000000$ ).

- Le cifre meno significative della mantissa che non possono essere rappresentate nel numero di bit a disposizione vengono eliminate mediante **troncamento** od **arrotondamento**.

## Troncamento e arrotondamento

### Esempio

$$r = 1029_{10} = 0.10000000101 \times 2^{11}$$

con:

- 10 bit per la mantissa
- 5 bit per l'esponente

Rappresentazione:

- con **troncamento**:

$$0\ 01011\ 1000000010 (= 1028)$$

- con **arrotondamento**:

$$0\ 01011\ 1000000011 (= 1030)$$

### Esempio

$$r = 0.8_{10} = 0.110011001100... \times 2^0 \text{ (periodico)}$$

con:

- 10 bit per la mantissa
- 5 bit per l'esponente

Rappresentazione:

- con **troncamento** (e con **arrotondamento**):

$$0\ 00000\ 1100110011 (\sim 0.7998)$$

## Valori massimi e minimi rappresentabili

Dipendono dall'esponente e dal metodo di rappresentazione.

Se il valore assoluto dell'esponente è rappresentato da 10 bit:

- il **massimo numero reale** è (in valore assoluto):

$$1 \times 2^{1023} \sim 10^{307}$$

- il **minimo numero reale** è (in valore assoluto):

$$0,1_2 \times 2^{-1023} \sim 0,5 \times 10^{-307}$$

### Accuratezza della macchina

È il più piccolo numero che, aggiunto ad 1.0 produce un risultato diverso da 1.0.

Tipicamente, per  $n=32$ , è circa  $3 \times 10^{-8}$

## Operazioni aritmetiche con numeri reali

### Somma

Si opera nel modo seguente:

- 1) si aumenta l'esponente del più piccolo, spostando contemporaneamente la mantissa a destra, fino a che i due esponenti sono uguali
- 2) si sommano le due mantisse insieme
- 3) si normalizza il risultato

In questo caso si possono generare errori:

### Errore di incolonnamento

Si manifesta nel sommare numeri con esponenti diversi.

### Esempio:

Supponiamo di lavorare in base 10 con 5 bit per la mantissa normalizzata.

$$0.12465 \times 10^1 + 0.32999 \times 10^{-2} = \\ 0.12465 \times 10^1 + 0.00032 \times 10^1 =$$

Riportando tutto all'esponente maggiore si perdono le ultime 3 cifre del secondo addendo (cioè la quantità  $0.00999 \times 10^{-2}$ ).

## Errore di cancellazione

Si manifesta nel sottrarre numeri molto simili fra loro.

### Esempio:

Supponiamo di lavorare in base 10 con 5 bit per la mantissa normalizzata.

Si consideri la seguente espressione:

$$0.33402 - 0.33247$$

che produce il valore:  $0.15464 \times 10^{-2}$

Nel calcolatore viene valutata:

$$0.33402 \times 10^0 = X_1$$

$$0.33247 \times 10^0 = X_2$$

$$X_1 - X_2 = 0.00155 \times 10^0$$

Normalizzando:

$$X_1 - X_2 = 0.155?? \times 10^{-2}$$

L'elaboratore introduce 00 al posto di ?? producendo  $0.15500 \times 10^{-2}$ . Le ultime due cifre non hanno però significato (il numero è affetto da errore).

☞ Questi errori fanno sì che la somma non goda sempre della proprietà associativa.

## Esempio

$$A = -0.12344 \times 10^0$$

$$B = 0.12345 \times 10^0$$

$$C = 0.32741 \times 10^{-4}$$

- Valutiamo  $A + (B + C)$

$$B + C =$$

$$\begin{array}{r} B \quad 0.12345 \times 10^0 + \\ C \quad 0.00003 \times 10^0 \text{ errore di incolonnamento} \\ \hline 0.12348 \times 10^0 + \\ A \quad -0.12344 \times 10^0 \\ \hline 0.00004 \times 10^0 \end{array}$$

### Normalizzazione:

$$0.40000 \times 10^{-4} \text{ (errore di cancellazione)}$$

- Valutiamo  $(A + B) + C =$

$$A + B =$$

$$\begin{array}{r} A \quad -0.12344 \times 10^0 + \\ B \quad 0.12345 \times 10^0 \text{ errore di incolonnamento} \\ \hline 0.00001 \times 10^0 + \end{array}$$

### Normalizzazione:

$$0.10000 \times 10^{-4} \text{ (errore di cancellazione)}$$

$$\begin{array}{r} 0.10000 \times 10^{-4} + \\ C \quad 0.32741 \times 10^0 \\ \hline 0.42741 \times 10^{-4} \end{array}$$

### Esercizio

Effettuare la somma:

$$4.6 + 0.5$$

con numeri rappresentati in floating point su 16 bit (di cui 8 di esponente in complemento a due, uno per il segno e 7 di mantissa), verificando i risultati ottenuti.

### Soluzione

$$\begin{aligned} 4.6 &= 0100.10011001\dots = 0.1001001 \times 2^3 \\ 0.5 &= 0000.10000000\dots = 0.1000000 \times 2^0 \end{aligned}$$

Allineando i numeri in modo che abbiano identico esponente:

$$\begin{array}{r} 4.6 \quad 0.100 \ 1001 \times 2^3 + \\ 0.5 \quad 0.000 \ 1000 \times 2^3 \\ \hline 0.101 \ 0001 \times 2^3 \end{array} \quad 101.0001 = 5 + 1/16 = 5.0625$$

Valore atteso per il risultato: 5.1

### Esercizio

Effettuare la seguente moltiplicazione fra numeri rappresentati in floating point su 16 bit (di cui 8 di esponente in complemento a due, uno per il segno e 7 di mantissa), verificando il risultato ottenuto:

$$2.15 \times (-4.8)$$

### Soluzione

Si ha:

$$\begin{aligned} 2.15 &= 0010.00100110\dots = 0.100 \ 0100 \times 2^2 \\ -4.8 &= -0100.1100110\dots = -0.100 \ 1100 \times 2^3 \end{aligned}$$

Per moltiplicare i due valori è sufficiente moltiplicare le mantisse e sommare gli esponenti.

Poiché però il prodotto di due numeri di 7 bit richiede in generale 14 bit per essere esattamente rappresentato, mentre anche la mantissa risultante dovrà essere rappresentata in 7 bit (evidentemente, i più significativi), l'operazione potrà dare luogo ad approssimazioni.

$$\begin{array}{r}
 100\ 0100 \times \\
 100\ 1100 = \\
 \hline
 1\ 0001\ 00\_ + \\
 10\ 0010\ 0\_ + \\
 1\ 0001\ 00\_ + \\
 \hline
 (0)1\ 0100\ 0011\ 0000
 \end{array}$$

Isolando i 7 bit più significativi si ha:

$$R = -0.01010000 \times 2^5 = -0.1010000 \times 2^4 = -10$$

Risultato teorico: -10.32

## Codifica dei caratteri

I caratteri di un testo vengono codificati tramite sequenze di bit, utilizzando un **codice** di traduzione.

Quello più usato è il **codice ASCII** (American Standard Code for Information Interchange).

Utilizza 7 bit. Rappresenta 128 caratteri. Mancano, ad esempio i caratteri accentati, greci etc.

### Esempio

‘0’ è codificato come  $048_{10}$  e  $30_{16}$  (e  $011\ 0000_2$ )

‘9’ è codificato come  $057_{10}$  e  $39_{16}$

‘a’ è codificato come  $097_{10}$  e  $61_{16}$

‘z’ è codificato come  $122_{10}$  e  $7A_{16}$

- Vengono inseriti in un byte, per cui 1 bit viene normalmente ignorato. In trasmissione funziona come **bit di parità** per individuare errori.
- Il valore (0 o 1) del bit di parità è scelto in modo che il numero di 1 sia pari.

**Ad esempio:**  $0 \Rightarrow 30_{16} \Rightarrow 0\ 011\ 0000_2$

Il codice **ASCII esteso** utilizza invece 8 bit (256 caratteri)

## Tabella dei codici ASCII

0	NUL	42	*	84	T	126	~	168	®	210	“	252
1	SOH	43	+	85	U	127	˘	169	©	211	”	253
2	STX	44	,	86	V	128	Ä	170	™	212	‘	254
3	ETX	45	-	87	W	129	Å	171	’	213	‚	255
4	EOT	46	.	88	X	130	Ç	172	¨	214	÷	
5	ENQ	47	/	89	Y	131	È	173	≠	215	◊	
6	ACK	48	0	90	Z	132	Ñ	174	Æ	216	ÿ	
7	BEL	49	1	91	[	133	Ö	175	Ø	217	ÿ	
8	BS	50	2	92	\	134	Ü	176	∞	218	/	
9	HT	51	3	93	]	135	á	177	±	219	¤	
10	LF	52	4	94	^	136	à	178	≤	220	<	
11	VT	53	5	95	_	137	â	179	≥	221	>	
12	FF	54	6	96	`	138	ã	180	¥	222	fi	
13	CR	55	7	97	a	139	ä	181	μ	223	fl	
14	SO	56	8	98	b	140	å	182	∂	224	‡	
15	SI	57	9	99	c	141	ç	183	∑	225	·	
16	DLE	58	:	100	d	142	ë	184	∏	226	,	
17	DC1	59	;	101	e	143	è	185	π	227	„	
18	DC2	60	<	102	f	144	ê	186	∫	228	‰	
19	DC3	61	=	103	g	145	ë	187	ª	229	Â	
20	DC4	62	>	104	h	146	ì	188	º	230	Ê	
21	Nak	63	?	105	i	147	ì	189	Ω	231	Á	
22	SYN	64	@	106	j	148	î	190	æ	232	È	
23	ETB	65	A	107	k	149	ï	191	ø	233	Ë	
24	Can	66	B	108	l	150	ñ	192	¿	234	Í	
25	EM	67	C	109	m	151	ó	193	¡	235	Î	
26	SUB	68	D	110	n	152	ò	194		236	Ì	
27	ESC	69	E	111	o	153	ô	195	√	237	ì	
28	FS	70	F	112	p	154	ö	196	f	238	Ó	
29	GS	71	G	113	q	155	õ	197	≈	239	Ô	
30	RS	72	H	114	r	156	ú	198	Δ	240	□	
31	US	73	I	115	s	157	ù	199	«	241	Ò	
32		74	J	116	t	158	û	200	»	242	Ú	
33	!	75	K	117	u	159	ü	201	…	243	Û	
34	"	76	L	118	v	160	ÿ	202		244	Ü	
35	#	77	M	119	w	161	°	203	À	245	ı	
36	\$	78	N	120	x	162	¢	204	Ã	246	^	
37	%	79	O	121	y	163	£	205	Ö	247	˘	
38	&	80	P	122	z	164	§	206	Œ	248	˚	
39	'	81	Q	123	{	165	•	207	œ	249	˛	
40	(	82	R	124		166	◊	208	–	250	˜	
41	)	83	S	125	}	167	ß	209	—	251	°	

**Altri codici:** ad esempio, sono EBCDIC (Extended Binary Coded Decimal Interchange Code) (8 bit) per elaboratori IBM, etc.

## Rappresentazioni binarie

## Riassumendo

- **100** come intero con  $n = 32$  bit (4 byte)

100: 2      0

50: 2 0

25: 2      1

12: 2 0

6: 2 0

3: 2 1

1: 2 1

[illegible]

- **-100**

11111111111111111111111110011100

- **100** come reale in virgola mobile
  - 1 byte per l'esponente
  - 3 byte per la mantissa

$$(100)_{10} = (1100100)_{10}$$

Normalizzato:

$$0.1100100 \times 2^7$$

**Esponente:** 7 = 00000111

0 00000111 1001000000000000000000

Il primo bit è il segno. Il primo bit della parte frazionaria è omissso poiché, essendo normalizzata, è sempre 1.

- **-100** come reale in virgola mobile

1 00000111 1001000000000000000000

- **'100'** come caratteri con codifica ASCII

00110001	31 <sub>16</sub>
00110000	30 <sub>16</sub>
00110000	

- **'-100'** come caratteri con codifica ASCII

00101101 ...

## Codifica delle immagini

Codificate come sequenze di 0 ed 1 (*bit map*).

*Digitalizzazione*, passaggio da una immagine ad una sequenza binaria.

L'immagine è vista come una matrice di **punti** (*pixel*).

Ad ogni punto è associato un numero che corrisponde ad un particolare **colore** o, per immagini in bianco e nero, ad un **livello di grigio** (numero potenza di 2).

Una maggiore qualità di immagine implica una maggiore occupazione di memoria:

- numero maggiore di punti per pollice (*dpi*, dot per inch; 1 pollice = 2.54 cm);
- numero maggiore di colori (*palette*).

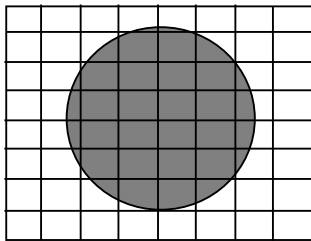
Per interpretare le sequenze di bit occorre conoscere:

- dimensioni dell'immagine;
- risoluzione (misurata in dpi);
- numero di colori o sfumature.

## Codifica delle immagini

**Standard di codifica:** TIFF (Tagged Image File Format) utilizza tecniche di compressione, GIF, etc.

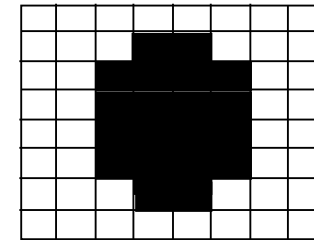
### Esempio:



64 pixel, immagine a due colori (bianco e nero).

Serve un solo bit per codificare la tonalità di colore, per ciascun pixel.

In totale 64 bit (8 byte).



```
00000000
00011000
00111100
00111100
00111100
00111100
00011000
00000000
```