

## **RAPPRESENTAZIONE DELLE INFORMAZIONI NEGLI ELABORATORI**

L'entità minima di informazione all'interno di un elaboratore prende il nome di **bit** (**B**inary **digiT**, cifra binaria).  
Mediante un bit possiamo distinguere due informazioni.  
Tutte le informazioni, anche le più complesse, sono rappresentate mediante sequenze di due soli simboli (**0** e **1**), ossia in forma binaria (o **digitale**).  
Si ha perciò bisogno di associare biunivocamente ad ogni informazione "elementare": caratteri, numeri, immagini, ... una sequenza binaria che la rappresenti.



**Codifica dell'informazione**

um.1

Con un bit si possono distinguere due diverse informazioni; per distinguere più informazioni bisogna usare sequenze di bit.  
Le diverse configurazioni di  $n$  bit permettono di individuare  $2^n$  informazioni diverse.

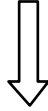
Alcune sequenze di bit assumono nomi particolari:

Byte	8 bit
Kilobit (Kbit o Kb)	$2^{10} = 1.024$ bit
Megabit (Megabit o Mb)	$2^{20} = 1.048.576$ bit
Gigabit (Gigabit o Gb)	$2^{30} = 1.073.741.824$ bit
Terabit (Terabit o Tb)	$2^{40} = 1.099.511.627.776$ bit
Kilobyte (Kbyte o KB)	$2^{10} = 1.024$ byte
Megabyte (Mbyte o MB)	$2^{20} = 2^{10}$ KB = $1.048.576$ byte
Gigabyte (Gbyte o GB)	$2^{30}$ byte
Terabyte (Tbyte o TB)	$2^{40}$ byte

um.2

# CODIFICA DEI NUMERI

Esigenza di rappresentare l'insieme infinito dei numeri mediante un numero limitato di segni grafici.



## Sistemi di numerazione

- **Cifre:** insieme finito di simboli distinti.
- **Codice:** insieme di regole che permette di associare ad una sequenza di cifre uno ed un solo numero.
- **Algoritmi** per l'esecuzione delle operazioni fondamentali.

um. 3

# SISTEMI DI NUMERAZIONE

Il valore di un numero può essere espresso con due rappresentazioni.

- **Non posizionali:** ad esempio, la numerazione romana.
- **Posizionali:** è associato un peso a ciascuna posizione all'interno della rappresentazione del numero. Il valore del numero è ottenuto facendo la somma dei prodotti di ciascuna cifra per il relativo peso (ad esempio, la numerazione decimale).

### Sistemi posizionali: regola generale

Sia  $r$  la base della numerazione. Un numero frazionario assoluto  $N_{(r)}$  è rappresentato con una successione di cifre  $d_i$  che soddisfano le seguenti regole.

L'insieme delle cifre  $d_i$  costituisce un insieme di simboli diversi di cardinalità  $r$ , tali che

$$0 \leq d_i \leq r - 1$$

um. 4

Per cui:

$$N_{(r)} = d_{n-1}d_{n-2} \dots d_1d_0d_{-1}d_{-2} \dots d_{-m}$$

Il valore di tale numero è dato da:

$$V(N_{(r)}) = \sum_{i=-m}^{n-1} d_i * r^i$$

### Vantaggi dei sistemi posizionali

- Esprimono qualunque valore finito: rango illimitato.
- Sono non ridondanti (ad ogni valore corrisponde una sola sequenza di simboli e viceversa).
- Le regole formali per le operazioni aritmetiche sono indipendenti dalla base della rappresentazione.

um. 5

## SISTEMA BINARIO

Base:  $r = 2$

Simboli base:  $d_i \in \{0; 1\}$

ESEMPIO

$$N_{(2)} = (11001)_2$$

$$V(N_{(2)}) = (1*2^4) + (1*2^3) + (0*2^2) + (0*2^1) + (1*2^0) = (25)_{10}$$

Sequenza dei pesi associati alle varie posizioni nei numeri binari.

$$\dots 2^3 \quad 2^2 \quad 2^1 \quad 2^0 \quad . \quad 2^{-1} \quad 2^{-2} \quad 2^{-3} \quad \dots$$

$$\dots (8) \quad (4) \quad (2) \quad (1) \quad (1/2) \quad (1/4) \quad (1/8) \quad \dots$$

**Il numero massimo rappresentabile con  $n$  cifre in base  $r$  risulta:**

$$N_{max} = (r-1)r^{n-1} + (r-1)r^{n-2} + \dots + (r-1)r + (r-1)r^0 =$$

$$= \sum_{i=0}^{n-1} (r-1) * r^i = r^n - 1$$

um. 6

Dato un numero  $N_r$  in base  $r$ , il numero di cifre necessario per la sua rappresentazione può essere ricavato nel seguente modo:

$$r^{n-1} < N_r + 1 \leq r^n$$

$$n-1 < \log_r(N_r + 1) \leq n$$

Da cui:

$$n = \lceil \log_r(N_r + 1) \rceil$$

Dove il simbolo  $\lceil x \rceil$  (detto *ceiling* in inglese) rappresenta l'intero superiore del numero reale  $x$ .

ESEMPIO

Sia dato il numero decimale  $N_{10} = (415)_{10}$

Con base  $r = 2$  si ottiene:

$$n = \lceil \log_2(415 + 1) \rceil = \lceil 8.70 \rceil = 9$$

$$\text{Da cui: } (415)_{10} = (110011111)_2$$

Con base  $r = 7$  si ottiene:

$$n = \lceil \log_7(415 + 1) \rceil = \lceil 3.09 \rceil = 4$$

$$\text{Da cui: } (415)_{10} = (1132)_7$$

um. 7

## CONVERSIONE DI BASE

### Da base $r$ a base 10

Basta esprimere le cifre  $d_i$  e le potenze della base  $r$  in base 10 e applicare la definizione di numero posizionale:

$$N_{(r)} = \sum_{i=m}^{n-1} d_i * r^i$$

ESEMPIO

$$(3246)_7 = 3*7^3 + 2*7^2 + 4*7 + 6 = (1161)_{10}$$

Se si considerano separatamente la parte intera e la parte frazionaria del numero da convertire si possono applicare algoritmi iterativi più agevoli da programmare.

$$N_r = I_r * F_r$$

um. 8

**Algoritmo iterativo per la conversione da base  $r$  a base 10  
(parte intera)**

$$I_r = d_{n-1}r^{n-1} + d_{n-2}r^{n-2} + \dots + d_2r^2 + d_1r^1 + d_0r^0 = \\ = (((\dots((d_{n-1}r + d_{n-2})r + d_{n-3})r + \dots + d_1)r + d_0$$

Esempio:

$$(3246)_7 = (((3 * 7 + 2) * 7 + 4) * 7 + 6) = (1161)_{10}$$

**Algoritmo**

1. Inizio
2. Partire dalla cifra più significativa
3. Se raggiunta la cifra meno significativa andare a 7)
4. Moltiplicare per la base
5. Sommare la cifra successiva;
6. Andare a 3)
7. Fine

um. 9

**Algoritmo iterativo per la conversione da base  $r$  a base 10  
(parte frazionaria)**

$$F_r = .d_{-1}r^{-1} + d_{-2}r^{-2} + \dots + d_{-(m+1)}r^{-(m+1)} + d_{-m}r^{-m} = \\ = .1/r (d_{-1} + 1/r(d_{-2} + 1/r(d_{-3} + \dots + 1/r(d_{-(m+1)} + 1/r d_{-m}))) \dots)$$

ESEMPIO

$$(0.101)_2 = .1/2 * (1 + 1/2 * (0 + 1/2 * 1)) = (0.625)_{10}$$

**Algoritmo**

1. Inizio
2. Partire dalla cifra meno significativa
3. Dividere per la base
4. Se raggiunta cifra più significativa andare a 8)
5. Sommare la cifra precedente
6. Dividere per la base
7. Andare a 4)
8. Fine

um. 10

### Conversione da sistema decimale a sistema in base $r$

Si considerano separatamente la parte intera e la parte frazionaria:

$$N_r = I_r \cdot F_r$$

Conversione parte intera (algoritmo per divisioni)

$$I_r = d_{n-1}r^{n-1} + d_{n-2}r^{n-2} + \dots + d_2r^2 + d_1r^1 + d_0r^0$$

Conversione parte frazionaria (algoritmo per prodotti)

$$F_r = .d_{-1}r^{-1} + d_{-2}r^{-2} + \dots + d_{-(m-1)}r^{-(m-1)} + d_{-m}r^{-m}$$

um. 11

### Algoritmo iterativo per la conversione da base 10 a base $r$ (parte intera)

$$\begin{aligned} I_{10} &= d_{n-1}r^{n-1} + d_{n-2}r^{n-2} + \dots + d_2r^2 + d_1r^1 + d_0r^0 = \\ &= (((\dots((d_{n-1}r + d_{n-2})r + d_{n-3})r + \dots + d_1)r + d_0 \end{aligned}$$

Si chiami  $Q_0$  il quoziente intero  $I_{10}/r$ ,

$d_0$  è il resto intero della divisione  $I_{10}/r$ .

Si può quindi scrivere iterativamente:

$$I_{10} = Q_0 r + d_0$$

$$Q_0 = Q_1 r + d_1$$

$$Q_1 = Q_2 r + d_2$$

.....

$$Q_{n-2} = 0 \cdot r + d_{n-1}$$

L'algoritmo si basa sul calcolo iterativo del quoziente intero diviso per la base. Il resto della divisione fornisce una cifra del numero nella nuova base a partire dalla cifra meno significativa.

um. 12

### Algoritmo

1. Inizio
2. Dividere il numero decimale per la base di arrivo  $r$  e prendere il quoziente intero
3. Il quoziente è il nuovo dividendo
4. Il resto è una cifra nella nuova base a partire dalla cifra meno significativa
5. Se quoziente è diverso da zero torna a 2)
6. Fine

Convertire  $(25)_{10}$  in binario.

$25 : 2 = 12$	resto = 1	↑
$12 : 2 = 6$	resto = 0	
$6 : 2 = 3$	resto = 0	
$3 : 2 = 1$	resto = 1	
$1 : 2 = 0$	resto = 1	

$$(25)_{10} = (11001)_2$$

un. 13

### Algoritmo iterativo per la conversione da base 10 a base $r$ (parte frazionaria)

$$F_{10} = .d_{-1}r^{-1} + d_{-2}r^{-2} + \dots + d_{-(m-1)}r^{-(m-1)} + d_{-m}r^{-m} =$$

$$= .1/r (d_{-1} + 1/r(d_{-2} + 1/r(d_{-3} + \dots + 1/r(d_{-(m-1)} + 1/r d_{-m}))) \dots)$$

Si moltiplica la parte frazionaria per la base di arrivo; la parte intera del risultato fornisce una cifra nella nuova base a partire dalla cifra più significativa.

Si itera il procedimento.

$$\begin{aligned} F_{10} * r &= d_{-1} + X_{-1} \\ X_{-1} * r &= d_{-2} + X_{-2} \\ &\dots \\ X_{-(m-1)} * r &= d_{-m} + 0 \end{aligned}$$

un. 14

### Algoritmo

1. Inizio
2. Moltiplicare la parte frazionaria del numero decimale per la base di arrivo
3. Separare la parte intera e la parte frazionaria
4. La parte intera dà una cifra nella nuova base a partire dalla cifra più significativa
5. Se non ottengo zero o non raggiungo la precisione richiesta torna a 2)
6. Fine

L'algoritmo termina naturalmente se si raggiunge un'iterazione in cui  $X_i = 0$ , oppure deve essere arrestato o quando si è raggiunta la precisione richiesta o quando si è raggiunto il numero di cifre prestabilito.

un. 15

Convertire il numero  $(0.3125)_{10}$  in base 2

$0.3125 * 2$	$= 0.625$	bit 0
$0.625 * 2$	$= 1.25$	bit 1
$0.25 * 2$	$= 0.5$	bit 0
$0.5 * 2$	$= 1.0$	bit 1
$0 * 2$	$= 0$	

$$(0.3125)_{10} = (0.0101)_2$$

Convertire il numero  $(0.2)_{10}$  in base 2

$0.2 * 2 = 0.4$	bit 0
$0.4 * 2 = 0.8$	bit 0
$0.8 * 2 = 1.6$	bit 1
$0.6 * 2 = 1.2$	bit 1
$0.2 * 2 = \dots$	

$$(0.2)_{10} = (0.\overline{0011})_2$$

un. 16

## APPROSSIMAZIONE NUMERI FRAZIONARI

In generale, un numero frazionario decimale con un limitato numero di cifre può corrispondere ad un numero frazionario in una base  $r$  qualunque con un numero infinito di cifre.

In questo caso, si deve arrestare l'algoritmo di conversione da base 10 a base  $r$  o quando si è raggiunta una precisione prestabilita o quando si è raggiunto il numero di cifre prestabilito (registro di conversione finito).

Convertire  $(0.3)_{10}$  in binario con una precisione inferiore a  $10^{-2}$

Si applichi iterativamente l'algoritmo di conversione

$0.3 * 2 = 0.6$	bit 0	[peso 0.5]
$0.6 * 2 = 1.2$	bit 1	[peso 0.25]
$0.2 * 2 = 0.4$	bit 0	[peso 0.125]
$0.4 * 2 = 0.8$	bit 0	[peso 0.0625]
$0.8 * 2 = 1.6$	bit 1	[peso 0.03125]
$0.6 * 2 = 1.2$	bit 1	[peso 0.015625]
$0.2 * 2 = 0.4$	bit 0	[peso 0.007815]

un. 17

Per raggiungere un'approssimazione inferiore al limite prescritto si devono calcolare sette cifre binarie, in quanto solo la settima cifra corrisponde a un peso inferiore a  $10^{-2}$ .

$$(0.3)_{10} = (0.0100110)_2$$

Conversione numero frazionario completo

$(12.5)_{10}$

$$(12)_{10} = (1100)_2 \quad (0.5)_{10} = (0.1)_2 \\ (1100.1)_2$$

$(1101.01)_2$

$$(1101)_2 = (13)_{10} \quad (0.01)_2 = (0.25)_{10} \\ (13.25)_{10}$$

un. 18

# SISTEMA OTTALE

Base:  $r = 8$

Simboli base:  $d_i \in \{0; 1; 2; 3; 4; 5; 6; 7\}$

ESEMPIO

$$N_{(8)} = (32.4)_8$$

$$V(N_{(8)}) = (3 \cdot 8^1) + (2 \cdot 8^0) + (4 \cdot 8^{-1}) = (26.5)_{10}$$

## Conversione da binario a ottale

Un numero in base  $r$  può essere scritto nella seguente forma simbolica generale:

$$N = \dots d_8 r^8 + d_7 r^7 + d_6 r^6 + d_5 r^5 + d_4 r^4 + d_3 r^3 + d_2 r^2 + d_1 r^1 + d_0 r^0$$

Si raggruppino i termini a tre a tre:

$$N = \dots (d_8 r^2 + d_7 r + d_6) r^6 + (d_5 r^2 + d_4 r + d_3) r^3 + (d_2 r^2 + d_1 r + d_0)$$

Si operi un cambiamento di base:  $R = r^3$

Ne consegue che:  $D = d'' r^2 + d' r + d$

è una cifra nella nuova base  $R$ , poiché vale la relazione:

$$0 \leq D \leq R - 1$$

un. 19

Si raccolgono i bit a gruppi di tre, partendo dalla virgola:

- procedendo verso sinistra per la parte intera, completando eventualmente l'ultima tripletta con zero non significativi;
- procedendo verso destra per la parte frazionaria, completando eventualmente l'ultima tripletta con zero non significativi.

## Conversione da ottale a binario

Si rappresenta ciascuna cifra ottale in base due su tre bit.

un. 20

Binario  $\Longrightarrow$  Ottale

$\underbrace{0\ 1\ 1}_3\ \underbrace{0\ 1\ 0}_2\ .\ \underbrace{1\ 0\ 0}_4$

Ottale  $\Longrightarrow$  Binario

$\overbrace{0\ 1\ 0}^2\ \overbrace{1\ 0\ 1}^5\ \overbrace{0\ 1\ 1}^3\ .\ \overbrace{0\ 1\ 0}^2$

um. 21

## **SISTEMA ESADECIMALE**

Base:  $r = 16$

Simboli base:  $d_i \in \{0; 1; 2; 3; 4; 5; 6; 7; 8; 9; A; B; C; D; E; F\}$

ESEMPIO

$$(E7A.C)_{16} = (E \cdot 16^2) + (7 \cdot 16) + (A \cdot 16^0) + (C \cdot 16^{-1}) = (3706.75)_{10}$$

### **Conversione da binario a esadecimale**

Si raccolgono i bit a gruppi di quattro, partendo dalla virgola:

- procedendo verso sinistra per la parte intera, completando eventualmente l'ultima quadriplettta con zero non significativi;
- procedendo verso destra per la parte frazionaria, completando eventualmente l'ultima quadriplettta con zero non significativi.

### **Conversione da esadecimale a binario**

Si rappresenta ciascuna cifra esadecimale in base due su quattro bit.

Binario  $\Rightarrow$  Esadecimale

$\underbrace{0001}_1 \underbrace{1010}_A . \underbrace{1000}_8$

Esadecimale  $\Rightarrow$  Binario

$\underbrace{1010}_A \underbrace{1011}_B . \underbrace{0010}_2$

un. 23

## IL NUMERO "MILLE" ESPRESSO IN:

Binario:

1 1 1 1 1 0 1 0 0 0  
 $1 \cdot 2^9 + 1 \cdot 2^8 + 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$   
512 + 256 + 128 + 64 + 32 + 0 + 8 + 0 + 0 + 0

Ottale: 1 7 5 0  
 $1 \cdot 8^3 + 7 \cdot 8^2 + 5 \cdot 8^1 + 0 \cdot 8^0$   
512 + 448 + 40 + 0

Decimale: 1 0 0 0  
 $1 \cdot 10^3 + 0 \cdot 10^2 + 0 \cdot 10^1 + 0 \cdot 10^0$   
1000 + 0 + 0 + 0

Esadecimale: 3 E 8  
 $3 \cdot 16^2 + 14 \cdot 16^1 + 8 \cdot 16^0$   
768 + 224 + 8

un. 24

# ADDIZIONE BINARIA

Siano  $x_i$  e  $y_i$  due variabili binarie, e sia  $s_i$  la loro somma e  $c_i$  il riporto.

Vale la seguente regola:

$x_i$	$y_i$	$s_i$	$c_i$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

ESEMPIO

$$\begin{array}{r}
 \phantom{1}1 \\
 1001 + \\
 0011 = \\
 \hline
 1100
 \end{array}$$

un. 25

Se si deve tener conto anche dei possibili riporti che arrivano dalle cifre precedenti si adotta la seguente tabella:

$x_i$	$y_i$	$c_{i-1}$	$s_i$	$c_i$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

ESEMPIO

$$\begin{array}{r}
 \phantom{1}111 \\
 1011 + \\
 0111 = \\
 \hline
 10010
 \end{array}$$

un. 26

# SOTTRAZIONE BINARIA

$x_i$	$y_i$	$d_i$	$b_i$
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

ESEMPIO

$$\begin{array}{r}
 0 \\
 \cancel{1}011 - \\
 0111 = \\
 \hline
 0100
 \end{array}$$

un. 27

Se si deve tener conto anche dei possibili prestiti concessi alle cifre precedenti si adotta la seguente tabella:

$x_i$	$y_i$	$b_{i-1}$	$d_i$	$b_i$
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

ESEMPIO

$$\begin{array}{r}
 110010 - \\
 011101 = \\
 \hline
 010101
 \end{array}$$

un. 28

## **SOMME E SOTTRAZIONI IN BASE QUALUNQUE**

Operazioni in base  $r = 16$ .

ESEMPIO

DFAA +	DFAA -
BBBB =	BBBB =
-----	-----
19B65	23EF

Operazioni in base  $r = 8$ .

ESEMPIO

5030 +	5030 -
2776 =	2776 =
-----	-----
10026	2032

un. 29

## **NUMERI A RAPPRESENTAZIONE FINITA**

I numeri trattati dagli elaboratori hanno una lunghezza finita. Questo fatto comporta alcune conseguenze non presenti nell'aritmetica ordinaria.

- L'insieme dei numeri è limitato.
- La rappresentazione dei numeri è a precisione finita.
- Un insieme di numeri a precisione finita non forma un insieme chiuso rispetto alle ordinarie operazioni aritmetiche  $+$ ,  $-$ ,  $*$ ,  $/$ .

In un insieme finito di rappresentazione, possono verificarsi tre forme di violazione.

1. Un numero è troppo grande (**overflow**).
2. Un numero è troppo piccolo (**underflow**).
3. Un numero non appartiene all'insieme dei numeri rappresentabili.

un. 30

## IL RIPORTO (CARRY)

Data una somma su numeri finiti a  $n$  bit, assume notevole importanza il riporto di peso  $2^n$  (riporto finale), che andrebbe a collocarsi sulla posizione di peso  $(n + 1)$ .

È normalmente indicato con  $C$  (*carry*) e memorizzato in una opportuna posizione di un registro di stato.

Nel caso di numeri assoluti, un riporto sull'ultima cifra  $C = 1$  indica una condizione di *tracimazione* (*overflow*); in tal caso il risultato della somma non è più esprimibile su  $n$  bit, in quanto non appartiene all'intervallo  $[0; 2^n-1]$

**Esempio:** numeri interi con tre cifre decimali.

Si possono rappresentare i numeri compresi tra 000 e 999; il numero successivo (1.000) richiede una quarta cifra.

Non si può rappresentare il risultato della somma ( $600 + 700$ ) perché il numero di cifre destinato alla rappresentazione è insufficiente: si ha un **overflow**.

um. 31

Poiché il numero 999 può essere scritto come  $10^3-1$  ( $1000-1$ ), possiamo enunciare la seguente regola:

**con  $n$  cifre decimali si possono rappresentare i numeri interi da 0 a  $10^n-1$**

I calcolatori usano basi o radici diverse da 10, di solito 2, 8 e 16. In accordo con l'osservazione precedente si ha che:

- con  $n$  cifre binarie si possono rappresentare i numeri interi da zero a  $2^n-1$
- con  $n$  cifre ottali si possono rappresentare i numeri interi da zero a  $8^n-1$
- con  $n$  cifre esadecimali si possono rappresentare i numeri interi da zero a  $16^n-1$

um. 32

Consideriamo la base due: con tre cifre binarie si possono rappresentare i numeri compresi tra 0 e  $2^3-1$  (ossia  $8 - 1 = 7$ ).  
**ESEMPIO**

numero	rappresentazione
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

um. 33

Si debbano eseguire le seguenti somme su numeri binari a 6 bit.  
**ESEMPIO**

$$\begin{array}{r}
 010101+ \\
 011010= \\
 \hline
 101111
 \end{array}$$

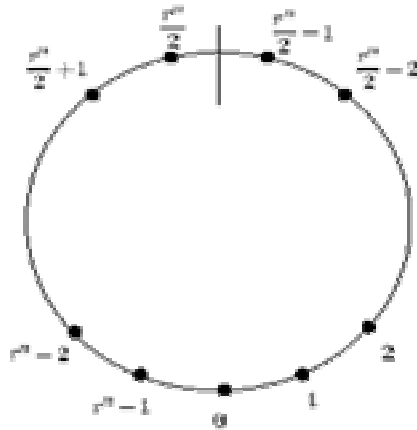
C = 0 il risultato è corretto.

$$\begin{array}{r}
 110101+ \\
 100111= \\
 \hline
 1|011100
 \end{array}$$

C = 1 condizione di trascinamento: risultato errato.

um. 34

I numeri trattati dagli elaboratori hanno una lunghezza finita. Per tener conto della lunghezza finita dei registri, si preferisce sostituire la rappresentazione lineare con una a cerchio.



un. 35

## NUMERI RELATIVI

Nella rappresentazione usuale si contraddistingue un numero positivo da uno negativo anteponendo al numero stesso un simbolo specifico.

$$\begin{array}{ll} + 3 & + 4328 \\ - 3 & - 4328 \end{array}$$

I simboli + e - non vanno confusi con gli operatori di somma e sottrazione.

In un sistema di rappresentazione strettamente binario non possono essere introdotti altri simboli rispetto a 0 e 1.

Si può rappresentare un numero positivo o negativo in base  $r$  senza aggiungere nuovi simboli agli  $r$  simboli usuali della rappresentazione.

Sia dato il numero:

$$N_r = d_{n-1} d_{n-2} \dots d_1 d_0$$

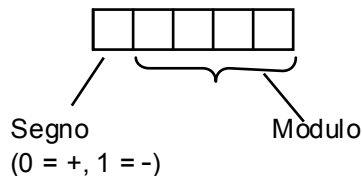
un. 36

Si può associare l'informazione del segno alla cifra più significativa, con la seguente convenzione:

$$d_{n-1} = \begin{cases} 0 & \text{per i numeri } \textit{positivi} \\ r - 1 & \text{per i numeri } \textit{negativi} \end{cases}$$

Esistono tre modi fondamentali di rappresentazione di numeri relativi.

1. *In Modulo e Segno (M&S)*
2. *In complemento alla base ( $\overline{\overline{N}}$ )*
3. *In complemento alla base diminuita ( $\overline{N}$ )*



um. 37

## **MODULO E SEGNO (M&S)**

Nella rappresentazione in *Modulo e Segno*, si associa il segno alla cifra più significativa e si riservano le restanti ( $n-1$ ) cifre per rappresentare il modulo del numero.

Sia dato il numero:

$$N_r = d_{n-1} d_{n-2} \dots d_1 d_0$$

Il valore della cifra di segno è dato da:

$$d_{n-1} = \begin{cases} 0 & \text{per i numeri } \textit{positivi} \\ r - 1 & \text{per i numeri } \textit{negativi} \end{cases}$$

Le restanti ( $n-1$ ) cifre rappresentano il *modulo* del numero.

Si adotta la seguente convenzione.

**Numeri positivi:**  $+ N_r = 0 \mid N \mid = 0 d_{n-2} \dots d_1 d_0$

**Numeri negativi:**  $- N_r = (r-1) \mid N \mid = (r-1) d_{n-2} \dots d_1 d_0$

um. 38

Per i numeri binari in *Modulo e Segno*, bisogna prefissare la lunghezza in bit della rappresentazione, e per il segno consegue la seguente convenzione.

bit di segno =  $\begin{cases} 0 & \text{per i numeri } \textit{positivi} \\ 1 & \text{per i numeri } \textit{negativi} \end{cases}$

ESEMPIO

$(6)_{10} = (110)_2$       *Binario puro*  
 $(+6)_{10} = (0110)_2$     *Binario positivo M&S*  
 $(-6)_{10} = (1110)_2$     *Binario negativo M&S*

Per calcolare il valore vero di un numero bisogna separare il bit più significativo di segno dagli altri:

$N = 1 | 0011 = -(0011)_2 = (-3)_{10}$

Numeri binari a sei cifre:     $(+12)_{10} = 0 | 01100$   
     $(-12)_{10} = 1 | 01100$

un. 39

In una rappresentazione in *M&S* su  $n$  cifre sono rappresentabili i numeri compresi nell'intervallo:

$$-(r^{n-1} - 1) \leq N \leq +(r^{n-1} - 1)$$

Infatti, non considerando la cifra di segno, rimangono  $(n - 1)$  cifre per il modulo, per cui il modulo massimo è:

$$N = r^{n-1} - 1$$

Lo zero assume due diverse rappresentazioni:

$$\begin{array}{ll} + 0 & 000\dots 00 \\ - 0 & 100\dots 00 \end{array}$$

Un numero binario su  $n$  cifre rappresenta i numeri compresi nell'intervallo:

$$-(2^{n-1} - 1) \leq N \leq +(2^{n-1} - 1)$$

Sono pertanto rappresentabili:  $2 * (2^{n-1} - 1) + 1 = 2^n - 1$  numeri diversi.

Poiché su  $n$  bit sono rappresentabili  $2^n$  combinazioni si spreca la combinazione legata alla doppia rappresentazione dello zero.

un. 40

<b>Corrispondenza decimale binario in M&amp;S</b>		
Valori rappresentati su cinque bit.		
Decimale Rappresentato	Binario in M&S	Valore Rappresentazione
+ 15	01111	15
+ 14	01110	14
+ 13	01101	13
....	....	....
+ 1	00001	1
+ 0	00000	0
- 0	10000	16
- 1	10001	17
....	....	....
- 13	11101	29
- 14	11110	30
- 15	11111	31

um. 41

<b>Calcolo valore vero in M&amp;S</b>
Il valore decimale di un numero binario su $n$ cifre è dato da
$V(N) = (-1)^{dn-1} * \sum_{i=0}^{n-2} d^i * r^i$
<b>ESEMPIO</b>
Numeri binari a sei cifre
$V(011001) = (-1)^0 * (1*2^4 + 1*2^3 + 1*2^0) = + 25_{10}$
$V(111001) = (-1)^1 * (1*2^4 + 1*2^3 + 1*2^0) = - 25_{10}$

um. 42

### Negazione di un numero in M&S

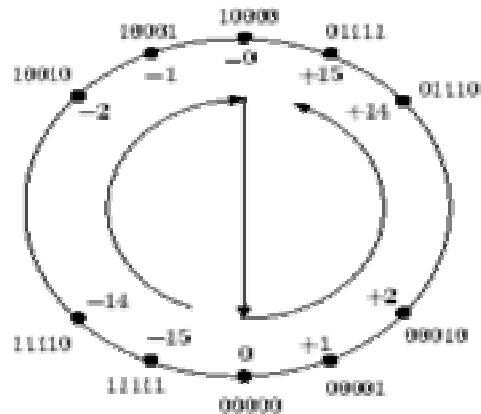
L'opposto di un numero si calcola

1. Lasciando invariato il modulo.
2. Invertendo la codifica del segno (nel sistema binario, invertendo il primo bit).

0 11001 = (+ 25)<sub>10</sub>

1 11001 = (- 25)<sub>10</sub>

0 11001 = (+ 25)<sub>10</sub>



um. 43

Dalla rappresentazione sul cerchio dei numeri binari in *M&S* su cinque bit si possono trarre le seguenti osservazioni.

- Esistono due zeri distanti fra loro una semicirconferenza.
- Andando da 00000 (0<sub>10</sub>) a 01111 (15<sub>10</sub>) sono rappresentati concordemente i numeri da + 0 a + 15.
- Andando da 10000 (16<sub>10</sub>) a 11111 (31<sub>10</sub>) sono rappresentati discordemente (in ordine decrescente) i numeri da - 0 a - 15.

**La rappresentazione in *M&S*, all'interno degli elaboratori, semplifica le operazioni di moltiplicazione e divisione, ma complica notevolmente quelle di addizione e sottrazione.**

um. 44

## COMPLEMENTO ALLA BASE

Il concetto di complemento alla base è intimamente connesso alla rappresentazione di un numero (in qualunque base  $r$ ) su un numero limitato di cifre.

Per poter definire il complemento alla base di un numero si deve prefissare il numero di cifre  $k$  della sua rappresentazione.

Data una base  $r$ , in un registro a  $k$  posizioni si può rappresentare qualunque numero MOD ( $r^k$ ).

ESEMPIO: contatore decimale a  $k$  cifre.

Se il contatore segna il numero  $N$ , può essere pensato come rappresentativo di qualunque numero  $N'$

$$N' = M * r^k + N$$

con  $M$  intero.

un. 45

ESEMPIO: contatore decimale a quattro cifre.

Per fare l'operazione:

$$\begin{array}{r} 8725 - \\ 3512 = \\ \hline \end{array}$$

5213

Si può calcolare:

$$\begin{array}{r} 10000 - \\ 3512 = \\ \hline \end{array}$$

6488

E successivamente calcolare:

$$\begin{array}{r} 8725 + \\ 6488 = \\ \hline \end{array}$$

15213

Che MOD( $10^4$ ) fornisce il valore richiesto.

un. 46

Dato un numero  $N$  di  $k$  cifre in base  $r$ , si definisce complemento alla base di  $N$  il numero:

$$\overline{N} = r^k - N$$

ESEMPIO  
 Con:  $r = 10 ; k = 2 ; N = 64$

$$\overline{N} = 10^2 - 64 = 36$$

Con:  $r = 2 ; k = 5 ; N = 01011$

$$\begin{array}{r} 10000 - \\ 01011 = \\ \hline \overline{N} = 10101 \end{array}$$

un. 47

**COMPLEMENTO ALLA BASE DIMINUITA**

Dato un numero  $N$  di  $k$  cifre in base  $r$ , si definisce complemento alla base diminuita di  $N$  il numero:

$$\overline{N} = (r^k - 1) - N$$

ESEMPIO  
 Con:  $r = 10 ; k = 2 ; N = 64$

$$\overline{N} = 99 - 64 = 35$$

Con:  $r = 2 ; k = 5 ; N = 01011$

$$\begin{array}{r} 11111 - \\ 01011 = \\ \hline \overline{N} = 10100 \end{array}$$

Il complemento alla base diminuita di un numero in base  $r$  si ottiene calcolando cifra per cifra il complemento a  $(r - 1)$ .  
 Il complemento alla base diminuita di un numero binario è detto complemento a uno e si ottiene complementando il numero bit a bit.

un. 48

### Relazione fra numeri in complemento

Dato un numero  $N_{(r)}$  di  $k$  cifre in base  $r$ , esiste la seguente relazione fra il complemento alla base  $\overline{\overline{N}}_{(r)}$  e il complemento alla base diminuita  $\overline{N}_{(r)}$ .

$$\overline{\overline{N}}_{(r)} = r^k - N_{(r)}$$

$$\overline{\overline{N}}_{(r)} = (r^k - 1) - N_{(r)} = r^k - N_{(r)} - 1$$

$$\overline{N}_{(r)} = \overline{\overline{N}}_{(r)} + 1$$

Vale la relazione

$$\overline{\overline{\overline{N}_{(r)}}} = r^k - \overline{\overline{N}}_{(r)} = r^k - (r^k - N_{(r)}) = N_{(r)}$$

$$\overline{\overline{\overline{N}_{(r)}}} = N_{(r)}$$

un. 49

### RAPPRESENTAZIONE DEI NUMERI INTERI RELATIVI IN COMPLEMENTO ALLA BASE

Utilizzando il concetto di *complemento alla base*, è possibile introdurre una rappresentazione dei numeri interi relativi, detta **rappresentazione in complemento alla base**.

La rappresentazione in complemento alla base adotta la seguente convenzione.

**Numeri Positivi:**  $+N \rightarrow 0 \mid N \mid$

**Numeri Negativi:**  $-N \rightarrow (r-1) \mid \overline{\overline{N}} \mid$

Dove  $\mid \overline{\overline{N}} \mid$  rappresenta il complemento alla base del modulo del numero.

Dalla relazione:  $\overline{\overline{N}}_{(r)} = \overline{N}_{(r)} + 1$

deriva la seguente *regola pratica*.

Sia dato il numero positivo su  $n$  cifre.

$$+N_r = 0 \, d_{n-2} \dots d_1 d_0$$

Il suo complemento alla base è dato da:

$$\overline{\overline{N}}_{(r)} = (r-1) \overline{d}_{n-2} \overline{d}_{n-3} \dots \overline{d}_1 \overline{d}_0 + 1 \quad \text{dove } \overline{d}_i = (r-1) - d_i$$

un. 50

## **RAPPRESENTAZIONE DEI NUMERI INTERI RELATIVI IN COMPLEMENTO A DUE**

Vale la seguente assunzione.

### **Numeri positivi**

Sono rappresentati dal loro modulo e hanno il bit più significativo di segno a 0 (e quindi coincidono con la rappresentazione in M&S)

$$+N \rightarrow 0 \mid N \mid$$

### **Numeri negativi**

Sono rappresentati dal complemento a due del corrispondente numero positivo, segno compreso.

I numeri negativi hanno il bit più significativo di segno sempre a 1.

$$-N \rightarrow 1 \mid \overline{N} \mid = \mid \overline{\overline{N}} \mid$$

um. 51

### **Regole per la complementazione a due**

Da decimale a complemento a due.

Se  $N \geq 0 \rightarrow 0 \mid$  binario puro

Se  $N < 0 \rightarrow$  1) Prendere binario puro segno compreso

2) Complementare bit a bit

3) Sommare +1

Da complemento a due a decimale.

Se  $N \geq 0 \rightarrow 0 \mid$  binario puro

Se  $N < 0 \rightarrow$  1) Prendere binario segno compreso

2) Complementare bit a bit

3) Sommare +1

4) Convertire numero binario

5) Aggiungere il segno -

um. 52



### Negazione binario in complemento a due

La negazione di un numero binario  $N$  si ottiene facendo il complemento a due del numero segno compreso.

ESEMPIO

$$011101 = +29$$

inversione

$$100010 +$$

$$1 =$$

-----

$$100011 = -29$$

$$110110 = -10$$

inversione

$$001001 +$$

$$1 =$$

-----

$$001010 = +10$$

un. 55

### Regola per la complementazione a due

Ulteriore regola pratica per il calcolo del complemento a due.

*Partendo dalla cifra meno significativa e procedendo verso sinistra, lasciare inalterati tutti gli zeri a destra del primo uno, lasciare inalterato tale uno, e complementare (in senso logico) i restanti bit fino al più significativo.*

ESEMPIO

01100

11010

10100

00110

### Intervallo di rappresentazione in complemento a due

Nel caso di numeri binari su  $n$  bit, si ha:

$$-2^{n-1} \leq N \leq 2^{n-1} - 1$$

Sono utilizzate tutte le possibili  $2^n$  combinazioni, con una sola rappresentazione per lo zero, e un numero negativo in più rispetto ai positivi.

un. 56

**Calcolo valore vero in complemento a due**  
 Nel caso di numeri binari su  $n$  bit, il valore vero di un numero  $N = d_{n-1}d_{n-2} \dots d_0$  rappresentato in complemento a due può essere calcolato con la seguente espressione

$$V(N) = -d_{n-1}2^{n-1} + \sum_{i=0}^{n-2} d^i * 2^i$$

Al bit di segno è attribuito un peso pari a  $2^{n-1}$   
 Dimostrazione  
 Se  $N \geq 0$  allora  $d_{n-1} = 0$  da cui  $V(N) = \sum_{i=0}^{n-2} d^i * 2^i$   
 Se  $N < 0$  allora  $d_{n-1} = 1$  da cui  $V(N) = -2^n + \overline{N}$   
 $= -2^n + 2^{n-1} + \sum_{i=0}^{n-2} d^i * 2^i = -2^{n-1} + \sum_{i=0}^{n-2} d^i * 2^i$

$V(0011) = -0*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = (+3)_{10}$   
 $V(1011) = -1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = (-5)_{10}$   
 $V(10000) = -1*2^4 + 0*2^3 + 0*2^2 + 0*2^1 + 0*2^0 = (-16)_{10}$

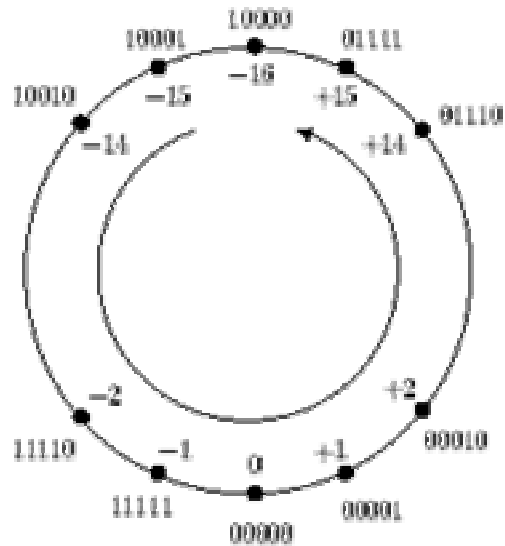
un. 57

**Corrispondenza decimale binario in complemento a due**  
 Valori rappresentati su cinque bit in complemento a due.

Decimale Rappresentato	Binario in Complemento a due	Valore Rappresentazione
+ 15	0 1 1 1 1	15
+ 14	0 1 1 1 0	14
+ 13	0 1 1 0 1	13
...	...	...
+ 2	0 0 0 1 0	2
+ 1	0 0 0 0 1	1
0	0 0 0 0 0	0
- 1	1 1 1 1 1	31
...	...	...
- 14	1 0 0 1 0	18
- 15	1 0 0 0 1	17
- 16	1 0 0 0 0	16

un. 58

### Cerchio delle rappresentazioni in complemento a due



un. 59

### Vantaggi della rappresentazione in complemento a due

Tenendo presente che si opera su registri di dimensione finita  $k$ , e quindi con una rappresentazione  $MOD(r^k)$ , le operazioni di sottrazione possono ridursi a operazioni di somma complementando a due il sottraendo.

Dalla definizione di complemento alla base si deduce che:

$$-M_{(r)} = \overline{\overline{M_{(r)}}} - r^k$$

Da cui:

$$N_{(r)} - M_{(r)} = N_{(r)} + \overline{\overline{M_{(r)}}} - r^k$$

Permette di fare somme e sottrazioni (somme algebriche) utilizzando lo stesso algoritmo (e quindi gli stessi componenti hardware o software) utilizzato per la somma tra numeri positivi. La moltiplicazione non è altrettanto semplice e a volte si ricorre ad un preventivo cambiamento di rappresentazione in modulo e segno.

un. 60

## RAPPRESENTAZIONE DEI NUMERI INTERI RELATIVI IN COMPLEMENTO A UNO

Vale la seguente assunzione.

### Numeri positivi

Sono rappresentati dal loro modulo e hanno il bit più significativo di segno a zero (e quindi coincidono con la rappresentazione in M&S)

$$+N \rightarrow 0 \mid N \mid$$

### Numeri negativi

Sono rappresentati dal complemento a uno del corrispondente numero positivo, segno compreso.

I numeri negativi hanno il bit più significativo di segno sempre a uno.

$$-N \rightarrow 1 \mid \bar{N} \mid = \mid \overline{+N} \mid$$

un. 61

Sia dato il numero positivo su  $n$  cifre:

$$+N_r = 0 d_{n-2} \dots d_1 d_0$$

Vale la seguente *regola pratica*.

Il complemento alla base diminuita è dato da:

$$\bar{N}_{(r)} = (r-1)\bar{d}_{n-2}\bar{d}_{n-3}\dots\bar{d}_1\bar{d}_0 \quad \text{dove } \bar{d}_i = (r-1) - d_i$$

### Regola per la complementazione a uno

La complementazione a uno di un numero binario si ottiene complementando singolarmente ogni bit del numero.

ESEMPIO	1 0 1 1 0 0 1
	inversione
	0 1 0 0 1 1 0
	0 1 1 0 1 0 0
	inversione
	1 0 0 1 0 1 1

un. 62

### Intervallo di rappresentazione in complemento a uno

Coincide con l'intervallo di rappresentazione dei numeri in *M&S*.

Nel caso di numeri binari su  $n$  bit, si ha:

$$-(2^{n-1} - 1) \leq N \leq 2^{n-1} - 1$$

Anche in questo caso si ha una doppia rappresentazione per lo zero.

$$+ 0 = 000 \dots 0$$

$$- 0 = 111 \dots 1$$

un. 63

### Calcolo valore vero in complemento a uno

Nel caso di numeri binari su  $n$  bit, il valore vero di un numero

$N = d_{n-1}d_{n-2} \dots d_0$  rappresentato in complemento a uno può essere calcolato con la seguente espressione

$$V(N) = -d_{n-1}(2^{n-1} - 1) + \sum_{i=0}^{n-2} d^i * 2^i$$

Al bit di segno è attribuito un peso pari a  $(2^{n-1} - 1)$

Dimostrazione

Se  $N \geq 0$  allora  $d_{n-1} = 0$  da cui  $V(N) = \sum_{i=0}^{n-2} d^i * 2^i$

Se  $N < 0$  allora  $d_{n-1} = 1$  da cui  $V(N) = -(2^{n-1} - 1) + \overline{N}$

$$= -2^{n-1} + 1 + \sum_{i=0}^{n-2} d^i * 2^i = -2^{n-1} + 1 + \sum_{i=0}^{n-2} d^i * 2^i$$

$$V(0101) = -0 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = (+5)_{10}$$

$$V(1101) = -1 * (2^3 - 1) + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = (-2)_{10}$$

$$V(10000) = -1 * (2^4 - 1) + 0 * 2^3 + 0 * 2^2 + 0 * 2^1 + 0 * 2^0 = (-15)_{10}$$

un. 64

<b>Corrispondenza decimale binario in complemento a uno</b>		
Valori rappresentati su cinque bit in complemento a uno.		
Decimale	Binario in	Valore
Rappresentato	Complemento a uno	Rappresentazione
+ 15	0 1 1 1 1	15
+ 14	0 1 1 1 0	14
+ 13	0 1 1 0 1	13
...	...	...
+ 2	0 0 0 1 0	2
+ 1	0 0 0 0 1	1
0	0 0 0 0 0	0
- 0	1 1 1 1 1	31
- 1	1 1 1 1 0	30
...	...	...
- 14	1 0 0 0 1	17
- 15	1 0 0 0 0	16

un. 65



un. 66

**Confronto rappresentazioni**  
 Confronto fra i valori rappresentati su cinque bit nelle tre rappresentazioni introdotte.

Binario	M&S	Complemento a due	Complemento a uno
00000	0	0	0
00001	1	1	1
00010	2	2	2
...	...	...	...
01110	14	14	14
01111	15	15	15
10000	- 0	- 16	- 15
10001	- 1	- 15	- 14
...	...	...	...
11101	- 13	- 3	- 2
11110	- 14	- 2	- 1
11111	- 15	- 1	- 0

un. 67

**ADDIZIONE DI NUMERI BINARI IN M&S**

Si calcola separatamente il modulo e il segno del risultato secondo la seguente tabella.

S(X)	S(Y)	S(R)	R
0	0	0	$ X  +  Y $
1	1	1	$ X  +  Y $
0	1	0 se $ X  \geq  Y $	$ X  -  Y $
		1 se $ X  <  Y $	$ Y  -  X $
1	0	0 se $ X  \leq  Y $	$ Y  -  X $
		1 se $ X  >  Y $	$ X  -  Y $

un. 68

Eeguire la seguente addizione con  $n = 6$  bit.

ESEMPIO

$$X = 110100 \quad (-20)_{10}$$

$$Y = 001011 \quad (+11)_{10}$$

$$S(X) = 1; S(Y) = 0$$

$$|X| > |Y|$$



$$S(R) = 1$$

$$|R| = |X| - |Y|$$

$$\begin{array}{r} 10100 - \\ 01011 = \\ \hline \end{array} \quad \begin{array}{l} |X| \\ |Y| \end{array}$$

$$\begin{array}{r} 01001 \\ \hline \end{array} \quad |R|$$

$$R = 101001 \quad (-9)_{10}$$

un. 69

Eeguire la seguente addizione con  $n = 6$  bit.

ESEMPIO

$$X = 100111 \quad (-7)_{10}$$

$$Y = 110010 \quad (-18)_{10}$$

$$S(X) = 1; S(Y) = 1$$



$$S(R) = 1$$

$$|R| = |X| + |Y|$$

$$\begin{array}{r} 00111 + \\ 10010 = \\ \hline \end{array} \quad \begin{array}{l} |X| \\ |Y| \end{array}$$

$$\begin{array}{r} 11001 \\ \hline \end{array} \quad |R|$$

$$R = 111001 \quad (-25)_{10}$$

un. 70

Occorre prestare attenzione al fatto che nelle operazioni sui moduli degli addendi non si verifichino condizioni di tracimazione (**overflow**).

La tracimazione si individua dal fatto che, nella somma fra i soli moduli, si ottiene un bit di riporto dalla cifra più significativa.

ESEMPIO (con  $n = 6$  bit)

$$X = 010111 \quad (+23)_{10}$$

$$Y = 001110 \quad (+14)_{10}$$

$$S(X) = 0; S(Y) = 0 \Rightarrow \begin{cases} S(R) = 0 \\ |R| = |X| + |Y| \end{cases}$$

$$\begin{array}{r} 10111 + \\ 01110 = \\ \hline 100101 \end{array} \quad \begin{array}{l} |X| \\ |Y| \\ |R| \end{array} \quad C = 1 \text{ overflow}$$

um. 71

## SOTTRAZIONE DI NUMERI BINARI IN M&S

Si calcola separatamente il modulo e il segno del risultato secondo la seguente tabella.

S(X)	S(Y)	S(R)	R
0	0	0 se $ X  \geq  Y $	$ X  -  Y $
		1 se $ X  <  Y $	$ Y  -  X $
0	1	0	$ X  +  Y $
1	0	1	$ X  +  Y $
1	1	0 se $ X  \leq  Y $	$ Y  -  X $
		1 se $ X  >  Y $	$ X  -  Y $

um. 72

Eeguire la seguente sottrazione  $X - Y$  con  $n = 6$  bit.

ESEMPIO

$$\begin{array}{l} X = 110100 \quad (-20)_{10} \\ Y = 111011 \quad (-27)_{10} \end{array}$$
$$\left. \begin{array}{l} S(X) = 1; S(Y) = 1 \\ |X| \leq |Y| \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} S(R) = 0 \\ |R| = |Y| - |X| \end{array} \right.$$

$$\begin{array}{r} 11011 - \quad |Y| \\ 10100 = \quad |X| \\ \hline 00111 \quad |R| \end{array}$$

$$R = 000111 \quad (+7)_{10}$$

un. 73

### **ADDIZIONE DI NUMERI BINARI IN COMPLEMENTO A DUE**

Le operazioni di somma e sottrazione nella rappresentazione in complemento a due sono particolarmente agevoli, per le seguenti condizioni.

- Sia la somma che la sottrazione si riportano ad una operazione di somma (eventualmente complementando un operando).
- Il bit di segno si tratta con le stesse regole con cui si trattano tutti gli altri bit del numero.

Si applicano a tutti i bit le regole per i binari puri.

Contrariamente a quanto avveniva per i binari puri il bit di carry non fornisce alcuna informazione sulla correttezza del risultato.

un. 74

Eseguire la seguente addizione con  $n = 6$  bit.  
**ESEMPIO**

011101 +	X	$(+29)_{10}$
101100 =	Y	$(-20)_{10}$
-----		
<b>1</b> 001001	R	$(+9)_{10}$

Il risultato è corretto anche se il riporto dalla cifra più significativa vale  $C_n = 1$ .

011001 +	X	$(+25)_{10}$
010100 =	Y	$(+20)_{10}$
-----		
<b>0</b> 101101	R	$(-19)_{10}$

Il risultato è **non** corretto anche se il riporto dalla cifra più significativa vale  $C_n = 0$ .

un. 75

**Condizione di trascinamento (overflow)**  
 Si verifica una condizione di trascinamento quando il risultato dell'addizione non appartiene all'intervallo dei numeri ammessi nella rappresentazione a  $n$  bit.

$$[- 2^{n-1}, + (2^{n-1} - 1)]$$

*Regola pratica*  
 Si può riconoscere la condizione di trascinamento attraverso due possibili regole.

1. I riporti generati in corrispondenza dei bit di peso  $n$  e di peso  $(n-1)$  sono diversi.
2. Gli addendi sono concordi e il bit di segno del risultato è diverso dal bit di segno degli addendi.

un. 76

Eseguire la seguente addizione con  $n = 6$  bit.

ESEMPIO

011101 +	X	(+29) <sub>10</sub>
101100 =	Y	(-20) <sub>10</sub>
-----		
<b>1</b> 001001	R	(+9) <sub>10</sub>

Poiché  $C_n = 1$  e  $C_{n-1} = 0$  il risultato è corretto (l'operazione non può essere in overflow perché il segno degli addendi è discorde).

011001 +	X	(+25) <sub>10</sub>
010100 =	Y	(+20) <sub>10</sub>
-----		
<b>0</b> 101101	R	- 19 <sub>(10)</sub>

Poiché  $C_n = 0$  e  $C_{n-1} = 1$  il risultato è errato (esistono potenziali condizioni di overflow perché il segno degli addendi è concorde).

un. 77

**SOTTRAZIONE DI NUMERI BINARI IN COMPLEMENTO A 2**

È ricondotta ad un'operazione di somma ( $MOD(2^n)$ ) aggiungendo al minuendo il complemento a due del sottraendo.

$$R = X - Y = X + (-Y) = X + \overline{\overline{Y}} - 2^n$$

Eseguire la seguente sottrazione con  $n = 6$  bit.

ESEMPIO

(X - Y)	010010 +	$\underline{\underline{X}}$	(+18) <sub>10</sub>
	110101 =	$\overline{\overline{Y}}$	(+11) <sub>10</sub> = (001011) <sub>2</sub>
	-----		
	<b>1</b> 000111	R	(+7) <sub>10</sub> Risultato corretto

(-X + Y)	101110 +	$\overline{\overline{X}}$	(+18) <sub>10</sub> (010010) <sub>2</sub>
	001011 =	Y	(+11) <sub>10</sub>
	-----		
	111001	R	(-7) <sub>10</sub> Risultato corretto

un. 78

### Osservazioni conclusive

- Sommando due numeri positivi si ha sempre  $carry = 0$  e si può avere o meno *overflow*.
- Sommando due numeri negativi si ha sempre  $carry = 1$  e si può avere o meno *overflow*.
- Sommando due numeri di segno discorde non si ha mai *overflow* e si può avere ogni valore per il *carry*.

um. 79

## NUMERI FLOATING POINT

Cosa si può fare per rappresentare numeri molto grandi o molto piccoli?

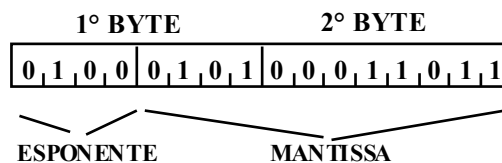
O si usano più byte o si sacrifica la precisione, adottando una rappresentazione in “virgola mobile”.

ESEMPIO

Supponiamo di avere a disposizione solo **due byte** e di dover memorizzare 13.077.130.

$$13.077.130 = (1.307 \times 10.000) + 7.130 = (1.307 \times 10^4) + 7.130$$

Se siamo disposti a trascurare l'ultimo addendo (7.130), possiamo memorizzare il numero dedicando i primi quattro bit *all'esponente* (il 4 di  $10^4$ ) e i restanti 12 bit al moltiplicatore 1.307 (*mantissa*)



um. 80

Otteniamo un sistema di rappresentazione in cui la “**gamma**” dei numeri esprimibile è indipendente dal numero delle cifre significative.

Sistema basato sulla notazione scientifica:

$$n = m \times 10^e$$

Mantissa  
o frazione

esponente



Rappresentazione in virgola mobile o **floating point**

um. 81

La gamma è determinata dal numero di cifre dell'esponente e la precisione dal numero di cifre della mantissa.

Poiché un numero si può rappresentare in tanti modi:

$$1.350 = 13,50 * 10^2 = 0,1350 * 10^4$$

è scelta una forma come standard.

Con i numeri floating point si può “simulare” il sistema dei numeri reali, pur con grandi differenze.

**I numeri reali hanno la potenza del continuo.**

**I numeri floating point sono in numero finito.**

Due limitazioni

**1. Ai bordi:** qualsiasi strumento di calcolo ha a disposizione solo un numero finito di cifre, non è possibile implementare numeri troppo grandi o numeri troppo piccoli.

**1. All'interno:** non è possibile rappresentare in un calcolatore nessun numero che abbia più cifre di quelle che la macchina consente.

um. 82

I numeri reali del calcolatore, quindi, non sono i reali di Weierstrass e Dedekind, ma sono numeri macchina, in pratica aggregati di un numero finito di numeri assegnati con un numero finito di cifre significative: sotto insieme finito di  $\mathbb{Q}$ .

Per esempio, consideriamo rappresentazioni con una mantissa di tre cifre con segno nella gamma  $0,1 \leq |m| < 1$  ed esponente di due cifre con segno.

minimo numero negativo:  $-0,999 \times 10^{99}$

massimo numero negativo:  $-0,100 \times 10^{-99}$

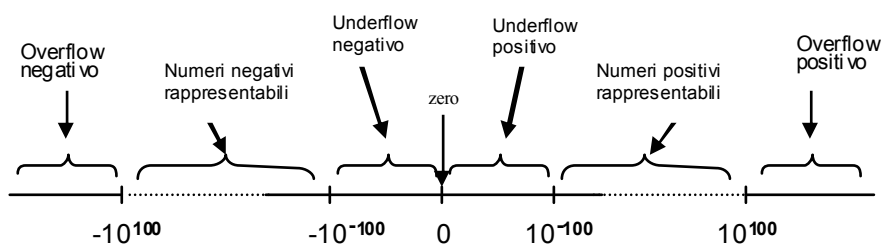
minimo numero positivo:  $0,100 \times 10^{-99}$

massimo numero positivo:  $0,999 \times 10^{99}$

Si rappresentano 179.100 numeri negativi e 179.100

numeri positivi, oltre allo zero, che ha tante rappresentazioni.

um. 83



**Overflow:** il risultato dell'operazione ha un esponente maggiore del massimo consentito, il valore assunto dal numero è "troppo" grande per essere rappresentato.

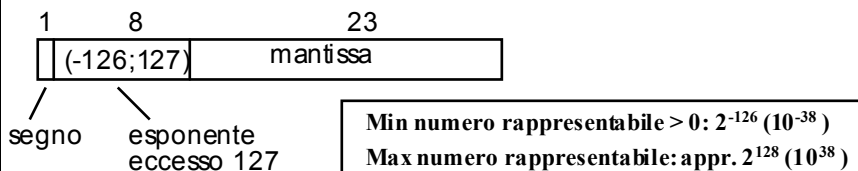
**Underflow:** il risultato dell'operazione ha un esponente minore del minimo consentito, il valore assunto dal numero è "troppo" piccolo per essere rappresentato.

Gli errori di underflow (lo zero è un'approssimazione) sono meno seri di quelli di overflow.

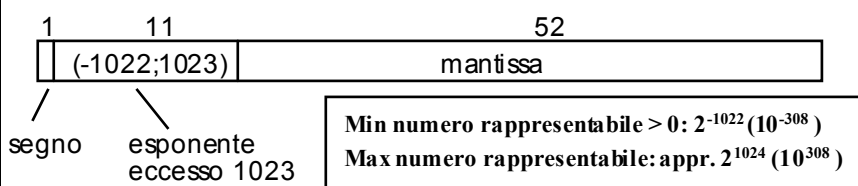
um. 84

# STANDARD IEEE 754

Semplice precisione: 32 bit



Doppia precisione: 64 bit



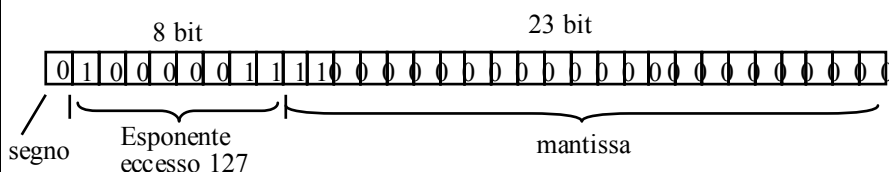
Precisione estesa: 80 bit (1 segno – 15 esponente – 64 mantissa)

Institute of Electrical and Electronic Engineers

un. 85

$$(+12)_{10} = (1100)_2 = 0,1100 * 2^4$$

$$127+4 = (131)_{10} = (10000011)_2$$



In questo modo il valore del numero rappresentato si ottiene come

$$(\text{segno}) * 2^{(\text{esponente} - 127)} * (+ b_{22} * 2^{-1} + b_{21} * 2^{-2} + \dots + b_0 * 2^{-23})$$

dove segno = +1 se  $b_{31} = 0$ , segno = -1 se  $b_{31} = 1$

L'esponente è ottenuto interpretando gli otto bit del campo

esponente come un numero senza segno, ovvero

$$\text{Esponente} = b_{23} * 2^0 + b_{24} * 2^1 + \dots + b_{30} * 2^7$$

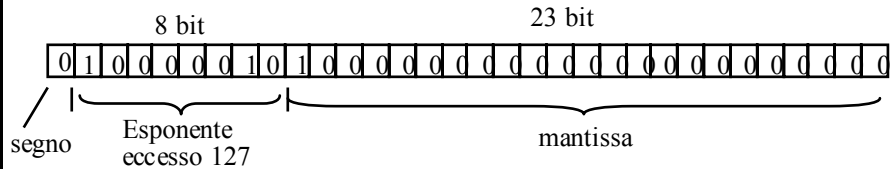
Segno = 0 = +, Esponente = 131

Mantissa =  $2^{-1} + 2^{-2} = 3/4$

Il numero vale quindi:  $2^{(131-127)} * (3/4) = 2^4 * 3/4 = 12$

un. 86

Rendendo implicito il bit a uno più significativo della mantissa si ottiene la rappresentazione secondo lo standard IEEE 754



Segno = 0 = +, Esponente = 130

$$(\text{segno}) * 2^{(\text{esponente} - 127)} * (1 + b_{22} * 2^{-1} + b_{21} * 2^{-2} + \dots + b_0 * 2^{-23})$$

Mantissa =  $2^{-1} = 1/2$

Il numero vale quindi:  $2^{(130-127)} * (1 + 1/2) = 2^3 * 3/2 = 12$

$(41400000)_{16}$

un. 87

## LA CODIFICA DELL'INFORMAZIONE

I sistemi di elaborazione operano al loro interno soltanto con segnali a due valori (binari).

I sistemi di elaborazione devono tuttavia scambiare informazioni con il mondo esterno in ingresso e in uscita.

Tali informazioni devono essere comprensibili ad un operatore umano, e assumono prevalentemente la forma di caratteri alfanumerici (numeri decimali, lettere dell'alfabeto, simboli di punteggiatura, simboli matematici).

Tale apparente incomunicabilità fra i due linguaggi, è ricomposta mediante l'adozione di opportune convenzioni (**codici**), mediante le quali è possibile rappresentare in modo univoco un certo numero di simboli con configurazioni di bit prestabilite.

un. 88

Si dice **alfabeto** un insieme non vuoto e finito di simboli detti caratteri.

Ad esempio:

➤  $A = \{ 0; 1 \}$  è l'alfabeto binario.

➤  $B = \{ A; B; C; \dots; X; Y; Z \}$  è l'alfabeto anglosassone.

Una **stringa** o **parola** in un dato alfabeto è una successione di simboli (anche ripetuti) di quello stesso alfabeto.

Dato un insieme  $I$  di oggetti, si dice codifica dell'insieme  $I$  nell'alfabeto  $L$ , un procedimento che permetta di stabilire una corrispondenza biunivoca fra gli elementi di  $I$  e un sottoinsieme di parole di  $L$ .

Per rappresentare  $M$  simboli diversi (**parole**), mutuamente esclusivi, con un codice binario, la lunghezza  $m$  della sequenza di bit del codice deve soddisfare la seguente relazione:

$$m \geq \lceil \log_2 M \rceil$$

un. 89

Siano dati due alfabeti qualunque,  $A$  di  $k$  simboli e  $B$  di  $m$  simboli:

$$A = \{ a_{k-1}; a_{k-2} \dots; a_0 \}$$

$$B = \{ b_{m-1}; b_{m-2} \dots; b_0 \}$$

è sempre possibile codificare l'insieme delle parole di  $A$  nell'alfabeto  $B$  e viceversa.

#### **Procedimento**

Si possono interpretare i  $k$  simboli di  $A$  come un sistema di numerazione posizionale in base  $k$ , e quindi ogni parola di  $A$  può essere interpretata come un numero intero nel sistema posizionale di base  $k$ .

Analogamente, ogni parola di  $B$  può essere interpretata come un numero intero nel sistema posizionale di base  $m$ .

Codificare una parola di  $A$  in  $B$  equivale ad un cambiamento di base, da base  $k$  a base  $m$ .

un. 90

Sia  $A = \{ A; B; C \}$  un alfabeto di tre caratteri e  
 $B = \{ \text{Rosso}; \text{Verde}; \text{Giallo}; \text{Blu} \}$  un alfabeto di quattro caratteri.  
Si vuole codificare la seguente stringa di A in B:

B C C A B

Per interpretare i simboli di A come un sistema di numerazione  
posizionale in base tre, attribuisco ai simboli di A i seguenti valori:

$$A = 2; B = 1; C = 0$$

Per interpretare i simboli di B come un sistema di numerazione  
posizionale in base quattro, attribuisco ai simboli di B i seguenti  
valori:

$$\text{Rosso} = 3; \text{Verde} = 2; \text{Giallo} = 1; \text{Blu} = 0$$

Da cui:

$$B C C A B = 1 \cdot 3^4 + 0 \cdot 3^3 + 0 \cdot 3^2 + 2 \cdot 3^1 + 1 \cdot 3^0 = (88)_{10}$$

Vale la conversione  $(88)_{10} = (1120)_4$

Per cui, infine:

$$(B C C A B)_A = (\text{Giallo Giallo Verde Blu})_B$$

un. 91

## ***CODICI DECIMALI PESATI***

Per rappresentare, per esempio, le dieci cifre distinte dei numeri  
decimali, occorrono codici binari a quattro bit.

Un codice decimale si dice **pesato** se la relazione che intercorre  
tra la cifra decimale D da rappresentare e l'insieme delle cifre  
binarie ad esso associato è del tipo:

$$D = \sum_{i=0}^{m-1} b_i \cdot p_i$$

dove:

$m$  = numero di bit del codice

$b_i$  = generico bit del codice

$p_i$  = peso corrispondente

un. 92

## **CODICE BCD (BINARY CODE DECIMAL)**

Il codice decimale BCD è un codice pesato con pesi (8421). Con questa codifica, ogni cifra decimale è rappresentata dal binario puro corrispondente, secondo la seguente tabella:

Decimale	Codice BCD
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1

Con questa codifica, le posizioni del codice da (1010) a (1111) non sono utilizzate.

un. 93

### **Conversione decimale – BCD**

Si converte ogni cifra decimale separatamente nel corrispondente codice BCD su quattro bit.

ESEMPIO

Convertire il numero decimale  $(5902)_{10}$  in codice binario BCD.

5	9	0	2
0 1 0 1	1 0 0 1	0 0 0 0	0 0 1 0

Si raggruppano le cifre binarie a quattro a quattro e si converte ciascun raggruppamento nella corrispondente cifra decimale secondo il codice BCD.

Ricavare il valore decimale della sequenza di bit

0 0 0 1	1 0 0 0	0 1 1 1	0 0 1 1
1	8	7	3

sapendo che corrisponde a una codifica BCD.

un. 94

### Somme in BCD

L'operazione di somma aritmetica in codice binario BCD si esegue sommando, in binario, le cifre di ugual peso.

Se la somma fornisce una configurazione non permessa, oppure si ha riporto, occorre ancora sommare il valore  $(0110)_{BCD} = 6_{10}$

L'eventuale riporto si somma alla cifra di peso superiore.

ESEMPIO

$$\begin{array}{r} 0010\ 0011 + \quad 23_{10} + \\ 0001\ 0010 = \quad 12_{10} = \\ \hline 0011\ 0101 \quad 35_{10} \end{array}$$

Il risultato è corretto perché non c'è riporto su nessuna cifra.

um. 95

ESEMPIO

$$\begin{array}{r} 0010\ 1001 + \quad 29_{10} + \\ 0001\ 0010 = \quad 12_{10} = \\ \hline 0011\ 1011 \quad ?_{10} \end{array}$$

La cifra calcolata 1011 non fa parte delle cifre ammesse dal codice BCD. Il codice BCD salta dalla cifra 1001 alla cifra 0000 con un salto di sei unità. Allora alla cifra calcolata 1011, che non fa parte delle cifre ammesse dal codice BCD, si devono aggiungere le sei unità mancanti, e sommare l'eventuale riporto alla cifra di peso superiore.

$$\begin{array}{r} 0011\ 1011 + \quad ?_{10} + \\ 0000\ 0110 = \quad 6_{10} = \\ \hline 0100\ 0001 \quad 41_{10} \end{array}$$

um. 96

ESEMPIO	
$\begin{array}{r} \overset{11}{0010\ 1001} + \\ 0001\ 1000 = \\ \hline 0100\ 0001 \end{array}$	$\begin{array}{r} 29_{10} + \\ 18_{10} = \\ \hline ?_{10} \end{array}$
<p>La cifra calcolata 0001, fa parte delle cifre ammesse dal codice BCD, ma il risultato è errato. Avendo ottenuto un riporto si devono aggiungere le sei unità mancanti, e sommare l'eventuale riporto alla cifra di peso superiore.</p>	
$\begin{array}{r} 0100\ 0001 + \\ 0000\ 0110 = \\ \hline 0100\ 0111 \end{array}$	$\begin{array}{r} ?_{10} + \\ 6_{10} = \\ \hline 47_{10} \end{array}$
<small>un. 97</small>	

<h2>CODICE GRAY</h2>
<p>I codici riflessi sono codici in cui le rappresentazioni di due numeri consecutivi differiscono per un solo bit.</p> <p>Trovano applicazione nelle conversioni analogico digitale, in quanto, nel passaggio da un numero rappresentato al successivo, si evita di passare attraverso possibili configurazioni spurie, dovute al non simultaneo cambiamento di tutti i bit del codice.</p> <p>Il codice Gray è ottenuto seguendo questo procedimento.</p> <ol style="list-style-type: none"> <li>1. Si scrivono i primi due bit: zero e uno.</li> <li>2. Si riflettono rispetto ad un ipotetico asse.</li> <li>3. Si aggiunge uno zero nella parte alta ed un uno nella parte bassa.</li> <li>1. Si torna al punto 2 fino al raggiungimento della dimensione prefissata del codice.</li> </ol>
<small>un. 98</small>

0		0		00
1	Riflessione	1	Aggiungo 0	01
---		---		---
		1	Aggiungo 1	11
		0		10
Codice Gray		Valore decimale		
	0000			0
	0001			1
	0011			2
	0010			3
	0110			4
	0111			5
	0101			6
	0100			7
	...			...

un. 99

## ***CODICI ALFANUMERICI***

Quando, oltre alle cifre decimali, si vogliono anche codificare caratteri (maiuscoli e minuscoli), simboli di punteggiatura, parentesi e operatori aritmetici, occorre estendere il numero di bit  $m$  del codice.

Se  $M$  è il numero totale di caratteri per cui si vuole definire un codice binario, occorre prevedere una lunghezza di codice  $m$  data dalla relazione:

$$m \geq \lceil \log_2 M \rceil$$

ZONATURA

DIGIT (8421)

un. 100

# IL CODICE ASCII

## American Standard Code for Information Interchange

È di gran lunga il codice alfanumerico più diffuso per lo scambio di informazioni fra sistemi di elaborazione.

Costituisce di fatto uno standard per la codifica dell'informazione nei sistemi di elaborazione.

Il codice ASCII standard è codificato su 7 bit, e quindi può rappresentare al massimo  $2^7 = 128$  simboli diversi.

Esiste una versione del codice ASCII che usa un ottavo bit (un byte). Tale codice è detto Codice ANSI-ASCII esteso (0..255).

Bit di controllo: parità pari.

ANSI (American National Standard Institute)

un. 10.1

<i>ASCII</i>	<i>Simbolo</i>	<i>ASCII</i>	<i>Simbolo</i>	<i>ASCII</i>	<i>Simbolo</i>
0101010	*	0111001	9	1000111	G
0101011	+	0111010	:	1001000	H
0101100	,	0111011	;	1001001	I
0101101	-	0111100	<	1001010	J
0101110	.	0111101	=	1001011	K
0101111	/	0111110	>	1001100	L
0110000	0	0111111	?	1001101	M
0110001	1	1000000	@	1001110	N
0110010	2	1000001	A	1001111	O
0110011	3	1000010	B	1010000	P
0110100	4	1000011	C	1010001	Q
0110101	5	1000100	D	1010010	R
0110110	6	1000101	E	1010011	S
0111000	8	1000110	F	1010100	T

un. 10.2

Codifica della parola **cane**

**01100011 01100001 01101110 01100101**  
**c a n e**

Problema inverso: quale testo è codificato da una data sequenza?

**011010010110110000100000010100000110111100101110**

- si divide la sequenza in gruppi di otto bit (un byte);
- si determina il carattere corrispondente ad ogni byte

**01101001-01101100-00100000-01010000-01101111-00101110**  
**i l blank P o .**

un. 10.3

I numeri possono essere codificati in due modi:

ASCII: **37** 00110011 00110111 (2 byte)  
3 7

BINARIO: **37** 00100101 (1 byte)

Il primo modo è usato per le comunicazioni con l'esterno (input/output).

Il secondo modo è usato all'interno; per fare i calcoli non è possibile usare direttamente le codifiche ASCII:

Esempio:

Numero	ASCII
3 +	00110011
2 =	00110010
----	-----
e	01100101

Esistono dei programmi di conversione che trasformano i numeri da una codifica all'altra.

un. 10.4

## ***CODICE UNICODE***

Lo standard Unicode è stato introdotto per rappresentare caratteri di testo in sistemi informatici, ed è stato assunto come standard internazionale con la sigla ISO (International Standard Organization) 10646.

Propone uno standard per rappresentare i caratteri e simboli di tutti i linguaggi scritti.

Unicode usa un codice a 16 bit, con cui è possibile codificare:  
 $2^{16} = 65.536$  caratteri distinti.

un. 105

## ***CODICI RIDONDANTI***

Un codice si dice ridondante, quando codifica gli  $M$  simboli distinti con  $n = m + r$  bit, cioè usando  $r$  bit aggiuntivi rispetto agli  $m$  bit strettamente richiesti dalla codifica binaria.

L'aggiunta di bit di ridondanza permette di costruire codici che consentono di controllare eventuali errori di trasmissione.

Si hanno due tipi di codici ridondanti:

1. **Codici a rivelazione di errore:** consentono di individuare la presenza di un errore;
2. **Codici a correzione di errore:** consentono non solo di individuare la presenza di un errore, ma anche di identificarne la posizione in modo da poterlo correggere.

un. 106

## ***CODICE BCD CON PARITÀ***

Per costruire un codice BCD a rivelazione di errore, si può aggiungere ad ogni parola codice un bit ridondante ( $r = 1$ ) detto di **parità**. Il bit di parità è posto a zero o a uno, in modo tale che la somma degli uni nella parola codice sia pari.

Decimale	Codice BCD	Bit di parità
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	1
3	0 0 1 1	0
4	0 1 0 0	1
5	0 1 0 1	0
6	0 1 1 0	0
7	0 1 1 1	1
8	1 0 0 0	1
9	1 0 0 1	0

un. 107

## ***CODICE DI HAMMING***

La **distanza di Hamming** fra due parole codice, si ottiene contando il numero di bit diversi in posizioni corrispondenti:

1 0 1 0 0 1

Distanza di Hamming = 3

0 0 1 1 1 1

In un codice a rivelazione di errore la distanza di Hamming fra due parole codice deve essere  $\geq 2$ , in quanto un errore singolo deve produrre una sequenza di bit che non appartiene a nessuna parola codice.

In un codice a correzione di errore singolo la distanza di Hamming fra due parole codice deve essere  $\geq 3$ , in quanto un errore singolo deve produrre una sequenza di bit che ha distanza di Hamming 1 dalla parola originaria e distanza di Hamming almeno 2 rispetto a ogni altra parola codice, in modo da identificare univocamente la parola originaria.

un. 108

Il codice di Hamming fornisce una procedura sistematica per generare codici ridondanti correttivi, tali che sia palese l'indicazione degli eventuali bit errati nella parola codice. Sarà considerato solo il caso di codice di Hamming autocorrettivo per bit singolo (in grado cioè di correggere un eventuale errore su un solo bit). Siano:

$m$	bit di parola
$r$	bit di ridondanza
$n = m + r$	bit di parola codice

Ognuna delle  $2^m$  parole legali, ha  $n$  parole codice errate a distanza di Hamming 1, ottenute cambiando un bit alla volta nella parola originaria. Ognuna delle  $2^m$  parole legali richiede  $(n + 1)$  configurazioni di bit dedicate. Per cui deve essere:

$$2^n \geq 2^m (n + 1)$$

$$2^m * 2^r \geq 2^m (m + r + 1)$$

$$2^r \geq (m + r + 1)$$

un. 10.9

Con  $m = 4$  si devono avere  $r = 3$  bit di ridondanza. Si dispongono gli  $m$  bit della parola e gli  $r$  bit di ridondanza (o di controllo) nel seguente modo:

$m = 4$ bit parola						
1	2	3	4	5	6	7
		↑		↑	↑	↑
↓	↓		↓			
$r = 3$ bit controllo						

Si dispongono i bit di controllo nelle posizioni corrispondenti a potenze di 2 (1, 2, 4, 8, ...) a partire dalla più significativa. Ad ogni bit di controllo è assegnato un valore di parità sulle sequenze di bit individuate.

1	2	3	4	5	6	7
X		X		X		X
	X	X			X	X
			X	X	X	X

un. 11.0

Ogni bit di controllo deve garantire la parità sulle sequenze di bit della tabella:

r1	r2	m1	r3	m2	m3	m4
1	2	3	4	5	6	7
X		X		X		X
	X	X			X	X
			X	X	X	X

bit 1: controlla la parità sui bit 1, 3, 5 e 7  
bit 2: controlla la parità sui bit 2, 3, 6 e 7  
bit 4: controlla la parità sui bit 4, 5, 6 e 7  
Il generico bit b è controllato dai bit r<sub>i</sub> secondo la seguente tavola della verità:

r3	r2	r1	b
		X	1
	X		2
	X	X	3
X			4
X		X	5
X	X		6
X	X	X	7

um. 11.1

La rilevazione e correzione di un eventuale bit errato avviene controllando il valore di parità dei bit di controllo.

Se il valore di parità del bit di controllo è corretto si pone a zero il valore nella tabella precedente; se non è corretto si pone a 1 il valore.

**ESEMPIO**

Se si riscontra errata la parità nei bit r<sub>1</sub> e r<sub>3</sub>, il bit errato è b = 5.

Infatti:

r3	r2	r1	b
1	0	1	5

Se si riscontra errata la parità nei bit r<sub>1</sub> e r<sub>2</sub>, il bit errato è b = 3.

Infatti:

r3	r2	r1	b
0	1	1	3

um. 11.2

Si determini il codice di Hamming per la parola su  $m = 4$  bit:  
 1 0 1 1

Si mettano i bit assegnati nelle posizioni corrette:

r1	r2	m1	r3	m2	m3	m4
1	2	3	4	5	6	7
		1		0	1	1

Deve essere:  
 $r1 = 0$  perché sia pari la sequenza 1, 3, 5, 7.  
 $r2 = 1$  perché sia pari la sequenza 2, 3, 6, 7.  
 $r3 = 0$  perché sia pari la sequenza 4, 5, 6, 7.

Il codice di Hamming completo per la parola data risulta:

r1	r2	m1	r3	m2	m3	m4
1	2	3	4	5	6	7
0	1	1	0	0	1	1

un. 11.3

Si supponga di avere la seguente parola codice, e di voler controllare se è corretta:

r1	r2	m1	r3	m2	m3	m4
1	2	3	4	5	6	7
0	1	0	0	0	1	1

Si verifica la correttezza di parità dei bit di controllo, e si individua univocamente l'eventuale bit errato.  
 La sequenza 1, 3, 5, 7 è dispari ( $r1$  errato).  
 La sequenza 2, 3, 6, 7 è dispari ( $r2$  errato).  
 La sequenza 4, 5, 6, 7 è pari ( $r3$  corretto).  
 La parola codice data contiene un bit errato che è il bit  $b = 3$ .

Infatti:

r3	r2	r1	b
0	1	1	3

La parola codice corretta risulta quindi essere:

1	2	3	4	5	6	7
0	1	1	0	0	1	1

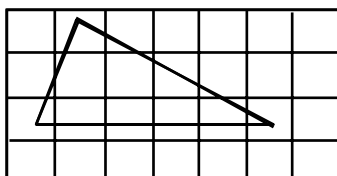
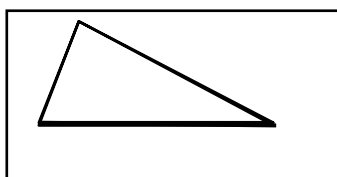
un. 11.4

Lettere e numeri non costituiscono le uniche informazioni utilizzate dagli elaboratori, ma si stanno diffondendo sempre di più applicazioni che utilizzano ed elaborano anche altri tipi di informazione: **diagrammi, immagini, suoni**.  
Spesso in questi casi si parla di applicazioni di tipo

**Multimediale.**

un. 115

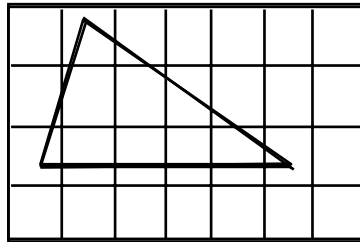
## ***LA CODIFICA DELLE IMMAGINI***



Ogni quadratino è chiamato **“pixel”**

un. 116

Poiché una sequenza di bit è lineare, è necessario definire delle convenzioni per ordinare la griglia dei pixel in una sequenza. Assumiamo che i pixel siano ordinati da sinistra verso destra e dal basso verso l'alto.



0	1	0	0	0	0	0
22	23	24	25	26	27	28
0	1	1	0	0	0	0
15	16	17	18	19	20	21
0	1	1	1	1	0	0
8	9	10	11	12	13	14
0	0	0	0	0	0	0
1	2	3	4	5	6	7

Con questa convenzione la rappresentazione della figura sarà data dalla stringa binaria

**0000000 0111100 0110000 0100000**

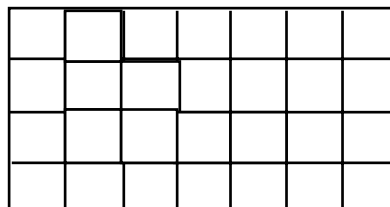
un. 117

Non sempre il contorno della figura coincide con le linee della griglia. Quella che si ottiene nella codifica è un'approssimazione della figura originaria.

Se riconvertiamo in immagine la stringa

**0000000011110001100000100000**

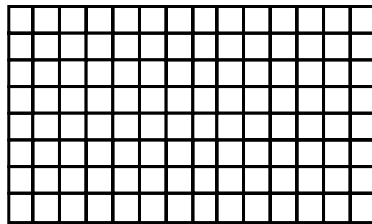
otteniamo:



un. 118

La rappresentazione sarà più fedele all'aumentare del numero di pixel, ossia al diminuire delle dimensioni dei quadratini della griglia in cui è suddivisa l'immagine

**“risoluzione dell’immagine”**



um. 119

Assegnando un bit ad ogni pixel è possibile codificare solo immagini senza livelli di chiaroscuro.  
Le immagini in bianco e nero hanno delle sfumature (diversi livelli di intensità di grigio).  
Per codificare le immagini con diversi livelli di grigio si stabilisce per ogni pixel il livello medio di grigio, cui è assegnata convenzionalmente una rappresentazione binaria.



Per memorizzare un pixel non è più sufficiente un solo bit.

um. 120

Ad esempio, se utilizziamo quattro bit possiamo rappresentare  $2^4 = 16$  livelli di grigio, mentre con otto bit ne possiamo distinguere  $2^8 = 256$ .

Analogamente possiamo codificare le immagini a colori.

In questo caso si tratta di individuare un certo numero di sfumature di colore differenti e di codificare ogni sfumatura mediante un'opportuna sequenza di bit.

La rappresentazione di un'immagine mediante la codifica dei pixel, è chiamata codifica **bitmap** e l'immagine è detta "**discretizzata**".

Il numero di byte richiesti dipende dalla risoluzione e dal numero di colori che ogni pixel può assumere.

Ad esempio, i monitor utilizzano *risoluzioni* di  $640 \times 480$ ,  $1024 \times 768$ , oppure  $1280 \times 1024$  ed un numero di colori per pixel che va da 256 fino a sedici milioni di colori.

un. 12.1

Per distinguere 256 colori sono necessari otto bit per la codifica di ciascun pixel: la codifica di un'immagine formata da  $640 \times 480$  pixel richiederà 2.457.600 bit (307.200 byte).

Esistono delle tecniche di compressione delle informazioni che consentono di ridurre drasticamente lo spazio occupato dalle immagini.

Immagini in movimento: memorizzazione mediante sequenze di fotogrammi (sono necessarie delle tecniche per ottimizzare tale memorizzazione).

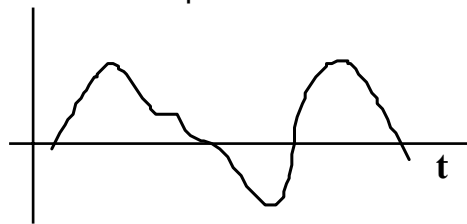
un. 12.2

## CODIFICA DEI SUONI

Anche i suoni possono essere rappresentati in forma digitale. Dal punto di vista fisico un suono è un'alterazione della pressione dell'aria che, quando rilevata dall'orecchio umano, è trasformata in un particolare stimolo al cervello.

La durata, l'intensità e la variazione nel tempo della pressione dell'aria sono le quantità fisiche che rendono un suono diverso da ogni altro.

Un suono può essere descritto mediante l'onda di pressione che descrive la variazione della pressione dell'aria nel tempo.

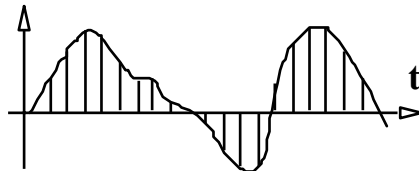


un. 12.3

Sull'asse delle ascisse è rappresentato il tempo e sull'asse delle ordinate è rappresentata la pressione corrispondente al suono stesso.

La conversione di un segnale continuo in una successione di numeri è eseguita con due successive operazioni elementari.

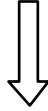
**1. Campionamento** del segnale (cioè si preleva una successione di campioni a intervalli costanti di tempo).



2. Ogni campione è **quantizzato** ossia convertito in un numero (si codificano in forma digitale le informazioni estratte dai campionamenti).

un. 12.4

Quanto più frequentemente il valore d'intensità dell'onda è campionato, tanto più precisa sarà la sua rappresentazione.



Il segnale può essere riprodotto perfettamente sulla base dei valori campione se la frequenza di campionamento è superiore al doppio della componente del segnale di frequenza più elevata. Un errore è comunque introdotto quando si converte il valore analogico di un campione in un numero con un numero limitato di cifre.

um. 125

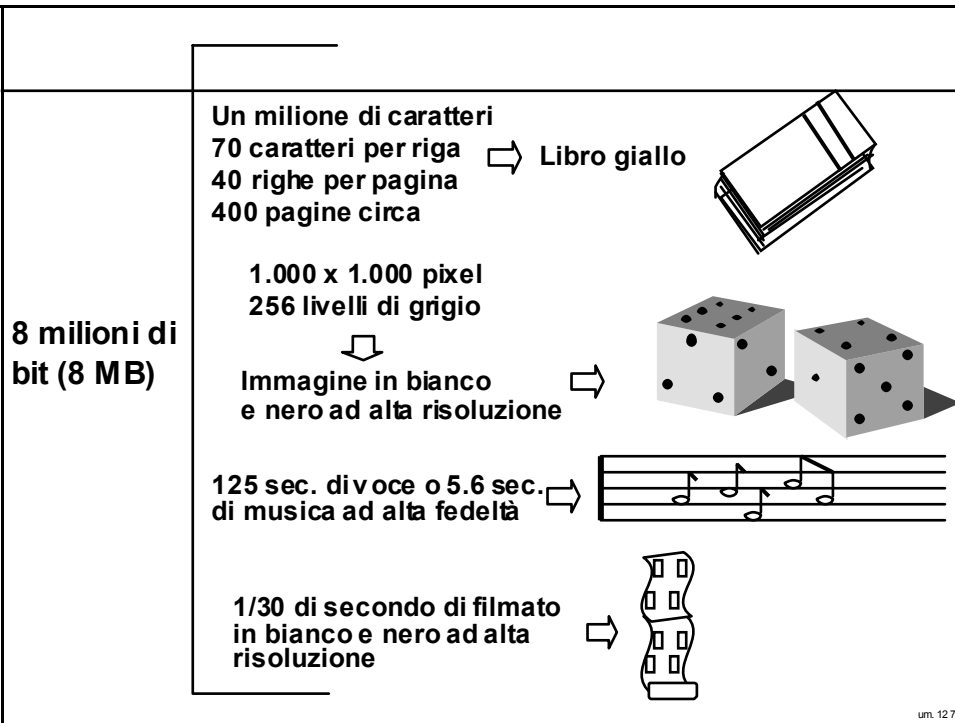
### **Trasmissione della voce sulla rete digitale ISDN**

Segnale vocale campionato ogni 125 milionesimi di secondo (8.000 campioni al secondo); di solito sono usati otto bit per campione. Sono trasmesse solo le componenti della voce di frequenza più bassa, come nella trasmissione analogica.

### **Compact disk musicale**

Si mescolano due registrazioni (stereofonia); 44.100 campioni al secondo per ogni registrazione; 16 bit per campione. Servono pertanto 1.411.200 bit per ogni secondo di registrazione.

um. 126



UBERTINI MASSIMO

<http://www.ubertini.it>

[massimo@ubertini.it](mailto:massimo@ubertini.it)

Dip. Informatica Industriale

I.T.I.S. "Giacomo Fauser"

Via Ricci, 14

28100 Novara Italy

tel. +39 0321482411

fax +39 0321482444

<http://www.fauser.edu>

<http://www.fauser.edu/fau/sistem.htm>

[massimo@fauser.edu](mailto:massimo@fauser.edu)

*Massimo Ubertini*