

Sistemi numerici

Il sistema che utilizza il pc per elaborare le istruzioni è il **sistema binario**, esso è composto da due sole cifre 0 e 1.

Utilizzando il sistema binario e avendo n cifre a disposizione possiamo rappresentare 2^n numeri, questo per via del calcolo combinatorio:

$$2 \cdot 2_1 \dots 2_n$$

• Numeri decimali

$$5374_{10} = 5 \cdot 10^3 + 3 \cdot 10^2 + 7 \cdot 10^1 + 4 \cdot 10^0$$

• Numeri binari

$$1101_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 13_{10}$$

Potenze di 2

• $2^0 = 1$	• $2^8 = 256$
• $2^1 = 2$	• $2^9 = 512$
• $2^2 = 4$	• $2^{10} = 1024$
• $2^3 = 8$	• $2^{11} = 2048$
• $2^4 = 16$	• $2^{12} = 4096$
• $2^5 = 32$	• $2^{13} = 8192$
• $2^6 = 64$	• $2^{14} = 16384$
• $2^7 = 128$	• $2^{15} = 32768$

Conversione numerica

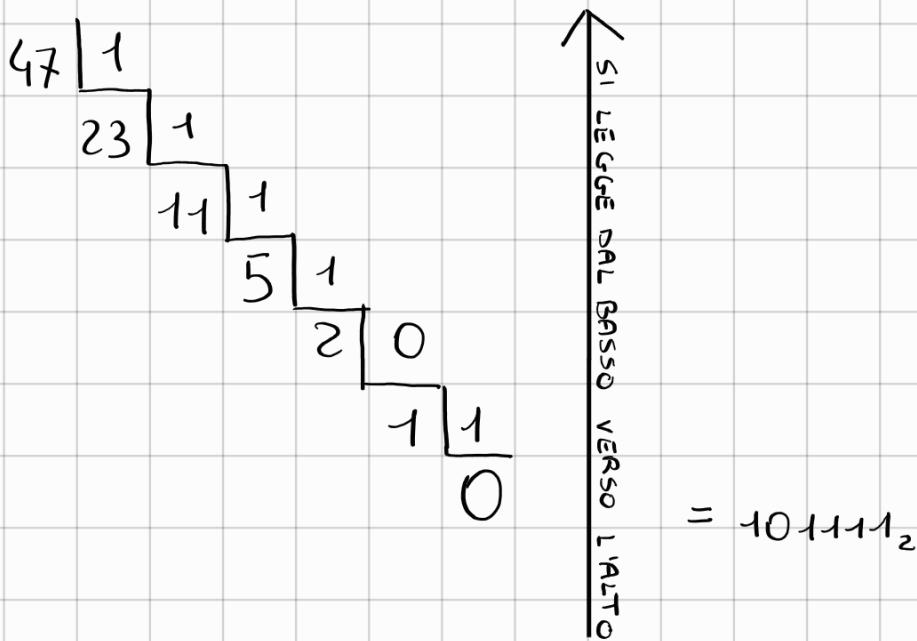
• Conversione da binario a decimale:

Un esempio di conversione in decimale può essere 10011₂ che in decimale diventa:

$$\begin{aligned} 10011_2 &= 2^4 \cdot 1 + 2^3 \cdot 0 + 2^2 \cdot 0 + 2^1 \cdot 1 + 2^0 \cdot 1 = \\ &= 16 \cdot 1 + 8 \cdot 0 + 4 \cdot 0 + 2 \cdot 1 + 1 \cdot 1 = \boxed{19_{10}} \end{aligned}$$

• Conversione da decimale a binario:

Un esempio di conversione in decimale può essere 47_{10} che in decimale diventa:



Per la conversione da decimale a binario ci sono 2 metodi:

• Metodo 1:

- Trova la più grande potenza di 2 più adatta;
- sottrai;
- se non esce 0 ripeti.

Esempio:

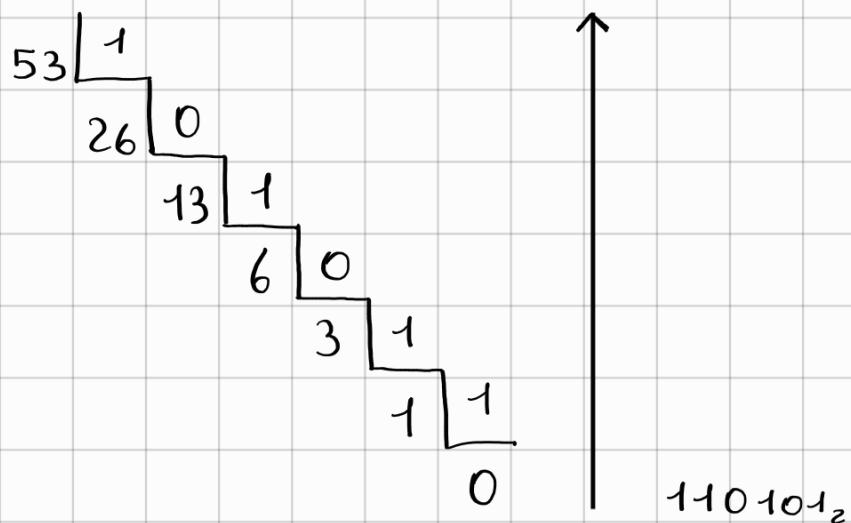
53_{10}	$32 \cdot 1$
► $53 - 32 = 21$	$16 \cdot 1$
► $21 - 16 = 5$	$4 \cdot 1$
► $5 - 4 = 1$	$1 \cdot 1$

$\rightarrow 110101_2$

• Metodo 2:

- dividi per 2;
- metti il resto a sinistra della rappresentazione binaria.

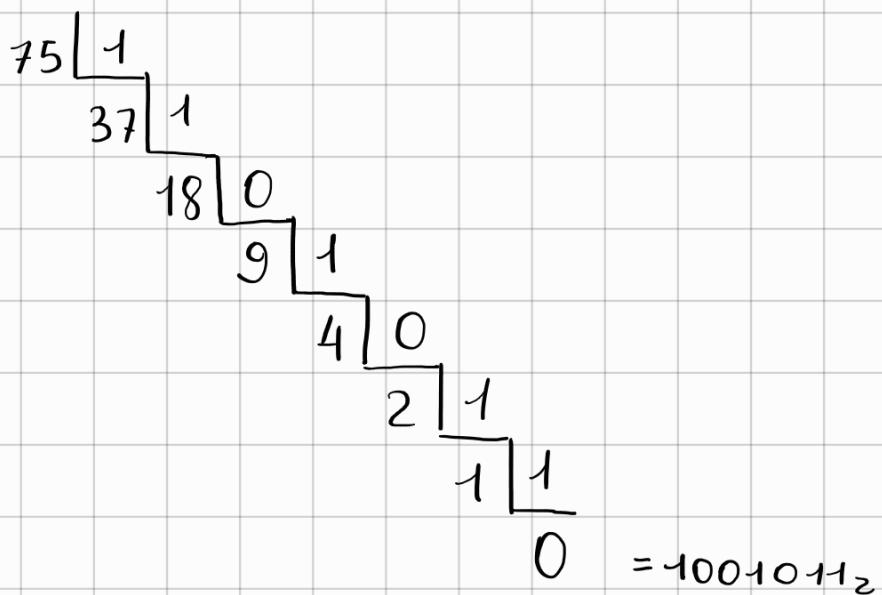
Esempio:



Altri esempi dei due metodi possono essere:

$$75_{10} = 64 + 8 + 2 + 1 = 1001011_2$$

oppure



Da x a y

Quando invece abbiamo una base x e vogliamo convertirla ad una base y prima convertiamo da x a decimale e poi da decimale a y .

Esempio:

Abbiamo 25_{20} e vogliamo convertirlo in base 4:

1) Convertiamo prima in base 10 quindi:

$$25_{20} = 2 \cdot 20^1 + 5 \cdot 20^0 = 40 + 5 = 45$$

2) convertiamo 45 in base 4 e per fare questo basterà dividere 45 finché non esce 0 e ad ogni divisione metteremo da parte i resti:

RESTI DI SOTTRAZIONI

$$\begin{array}{r} 45 \\ | \\ 11 \quad 3 \\ | \\ 2 \quad 2 \\ | \\ 0 \end{array}$$

$= 231_4$

Valori binari e intervalli

- Numero decimale a N cifre

- Quanti valori? 10^n

- Range? $[0, 10^n - 1]$

Esempio con un numero decimale a 3 cifre:

- $10^3 = 1000$ possibili valori;

- Range: $[0, 999]$

- Numero binario a n cifre

- Quanti valori? 2^n

- Range $[0, 2^n - 1]$

Esempio con un numero binario a 3 cifre:

- $2^3 = 8$ possibili valori

- Range: $[0, 7] = [000_2, 111_2]$

Numeri esadecimales

Hex Digit	Decimal Equivalent	Binary Equivalent
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Le caratteristiche principali dei numeri esadecimales sono 2:

- ha come base 16
- abbreviazione binaria

Conversione da esadecimale a binario

Proviamo per esempio a convertire 4AF (anche scritto 0x4AF) in binario:

0 1 0 0 | 1 0 1 0 | 1 1 1 1
 4 A F 2

Conversione da esadecimale a decimale

Per la conversione da esadecimale a decimale invece proviamo a convertire sempre 4AF₁₆ in decimale:

$$16^2 \cdot 4 + 16^1 \cdot 10 + 16^0 \cdot 15 = 1199_{10}$$

Bits, Bytes, Nibble...

Un **bit** è una singola unità di un numero binario ad esempio 0 o 1.

Invece un **byte** sono 8 bit ed un nibble 4 bit, quindi 2 nibble formano un byte.

- Un esempio di bits può essere:

- Esempio di Bytes & Nibble invece può essere:

byTe

1 0 0 1 0 1 1 0
Nibble

- Infine un esempio di Bytes può essere.

Grandi potenze di 2

Esempi di grandi potenze di z sono:

- $2^{10} = 1 \text{ Kilo} \approx 1000(1024)$
 - $2^{20} = 1 \text{ mega} \approx 1 \text{ milione}(1048576)$
 - $2^{30} = 1 \text{ giga} \approx 1 \text{ miliardo}(1073741824)$

Addizioni

Le addizioni con numeri decimali e binari si fa in due modi diversi:

- ## • decimal

$$\begin{array}{r}
 1 \\
 3734 \\
 + \\
 5168 \\
 \hline
 8902
 \end{array}$$

• binario

RIPORTI

$$\begin{array}{r} 1 \ 0 \ 1 \ 1 + \\ 0 \ 0 \ 1 \ 1 = \\ \hline 1 \ 1 \ 1 \ 0 \end{array}$$

In binario le cose che cambiano nell'addizione sono:

- $1+1$ fa 0 con riporto di 1
- $0+0$ fa 0
- $1+0$ fa 1
- $0+1$ fa 1

Esempi di operazioni esadecimali

Facciamo 2 esempi di operazione con numeri esadecimali:

• Addizione:

$$\begin{array}{r} 3 \ A \ 0 \ 9 + \\ 1 \ B \ 1 \ 7 = \\ \hline 5 \ 5 \ 2 \ 0 \end{array}$$

9
A
B
C
D
E
F

0 com riporto di 1

• Sottrazioni:

$$\begin{array}{r} 3 \ D \ 0 \ 9 - \\ 1 \ B \ 2 \ 7 = \\ \hline 2 \ 1 \ E \ 2 \end{array}$$

Overflow

Poiché i sistemi digitali operano con un numero fisso di Bit, è necessario considerare quei casi in cui, durante un'addizione o una moltiplicazione, il numero di bit sia insufficiente a poter rappresentare il risultato.

Questo problema prende il nome di overflow (ossia strabordare)

Un esempio di overflow può essere $101 \cdot 111$

$$\begin{array}{r} 101 \times \\ 111 = \\ \hline 101 + \\ 101 // + \\ \hline 101111 = \\ 100011 \end{array}$$

In questo caso i due moltiplicandi utilizzano entrambi 3 bit ma il loro prodotto ne richiede almeno 6.

In questi casi i bit in eccesso vengono **scartati** generando un risultato sballato perché togliendo l'eccesso verrebbe $101 \cdot 111 = 011$, dunque $5 \cdot 7 = 3$

Shift di Bit a destra o a sinistra

Un operatore binario aggiuntivo rispetto alla normale aritmetica decimale è l'operazione di **shift** (o spostamento).

Lo shift è un operatore che prevede lo spostamento **a destra o a sinistra** di una certa quantità di caselle di tutti i bit del numero binario sul quale viene applicato

$$000101_2 \ll 2 = 010100_2$$

$$011000_2 \gg 2 = 000110_2$$

Numeri binari negativi

Fino ad ora, abbiamo parlato di numeri binari positivi.

Per comprendere ulteriori operazioni aritmetiche è necessario introdurre il **segno** nei numeri binari.

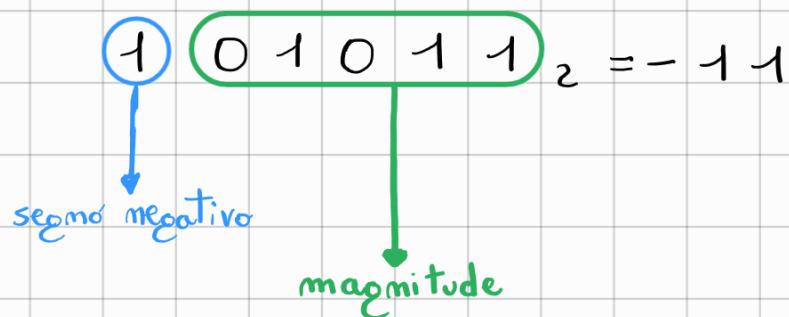
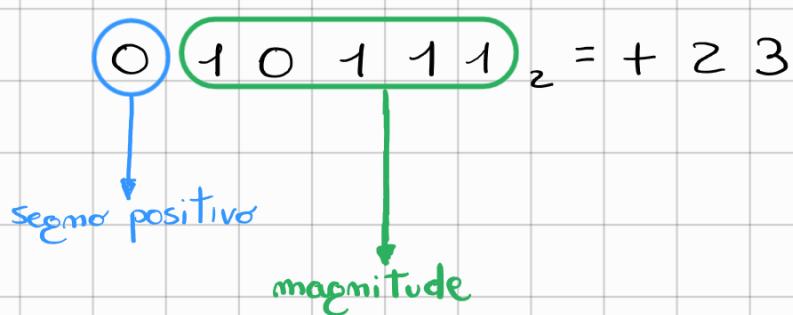
Esistono 2 diversi metodi di rappresentare i numeri negativi e sono:

- 1) i numeri in **Sign / Magnitude**
- 2) i numeri in **Complemento a 2**

Numeri in Sign / Magnitude

Nel primo metodo di rappresentazione la differenza è poca: il most significant Bit del numero rappresenta il segno, mentre tutti gli altri rappresentano il suo valore (o **magnitude**).

Nel caso in cui il bit del segno assume valore 0 allora il numero sarà positivo mentre se è 1 allora il numero sarà negativo:



Quindi dati n bit nei 2 metodi di rappresentazione abbiamo i seguenti intervalli di valori disponibili:

$$[0, 2^n - 1]$$

senza segno

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

sign/magnitude

Nonostante ciò, anche questo formato ha delle problematiche:

- le addizioni non funzionano;
- abbiamo 2 modi per rappresentare lo 0 (1000 e 0000, corrispondenti a -0 e +0).

$$\begin{array}{r} 1\ 1\ 1\ 0 \\ 0\ 1\ 1\ 0 \\ \hline 1\ 0\ 1\ 0\ 0 \end{array}$$

In questa addizione per esempio c'è un errore perché anche se scartassimo l'overflow avremmo comunque $-6 + 6 = +4$

Numeri in complemento a 2

Per risolvere questo problema è stato introdotto il sistema binario basato sul **complemento a 2**, che va ad aggiungere un ulteriore significato al most significant Bit.

In questo sistema infatti assume il suo valore effettivo ma a differenza del sistema senza segno, il valore assoluto sarà negativo.

Dati n bit, il most significant Bit assumerà il valore $-(2^{n-1})$, mentre tutti gli altri bit manterranno il valore positivo:

$$1\ 0\ 1\ 1\ 1_2 = 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + (-1 \cdot 2^4) = -9_{10}$$

Per poter calcolare il complemento a 2 di un numero decimale negativo, invece, è prima necessario calcolare il **Complemento a 1** del suo corrispettivo numero binario positivo.

Com complemento a 1 si intende semplicemente invertire tutti i bit, trasformando quindi tutti i suoi 0 in 1 e viceversa:

$$-25_{10} \rightarrow +25_{10} = 011001_2 \rightarrow 100110_2$$

Una volta calcolato il prossimo passo sarà semplicemente sommare 1 al risultato del complemento a 1:

$$100110 + 1 = 100\ 111_2$$

Per confermare che la conversione sia avvenuta correttamente, possiamo calcolare il valore del complemento a 2 quindi:

$$\begin{array}{r} \text{sg 3 2 1 0} \\ 100111_2 = 1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4 + (-1 \cdot 2^5) = -25_{10} \end{array}$$

L'utilità del complemento a 2 è evidente, perché grazie al modo in cui viene rappresentato è possibile svolgere operazioni tra numeri di segno opposto senza problematiche.

Viene anche risolto il problema dei due zeri, perché 0000 corrisponde a 0 mentre 1000 sarà -8.

$$+6 = 0110_2$$

$$-6 = 1010_2$$

$$\begin{array}{r} 1010 + \\ 0110 = \\ \hline 10000 \end{array}$$

Scartando l'overflow otteniamo $+6 - 6 = 0$, quindi il risultato è corretto.

Il range di valori negativi rappresentabili in complemento a 2 sarà:

$$\underbrace{[-(2^{m-1}), 2^{m-1}-1]}_{\text{complemento a 2}}$$