I sistemi numerici

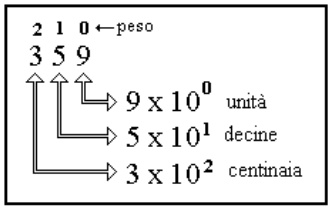
Il sistema Decimale è costituito da 10 **simboli**. I dieci simboli che tutti conosciamo sono lo zero (0), l'uno (1), il due (2), ecc. ecc., indicati come l'insieme Dec={0,1,2,3,4,5,6,7,8,9}. Il numero di simboli che costituiscono un sistema numerico definisce la **Base**. Un qualunque numero che appartiene a questo sistema deve essere espresso solo con i simboli che fanno parte del sistema.

I concetti appena esposti per il sistema decimale si estendono ad altri sistemi numerici. La tabella che segue sintetizza i quattro sistemi numerici che sono maggiormente usati.

Sistema	Base	Simboli	Es. Validi	Es. Errati
Decimale	10	0 1 2 3 4 5 6 7 8 9	12, 49, 121	3A, 2F
Binario	2	0 1	10, 0110, 1100	35, 210
Ottale	8	0 1 2 3 4 5 6 7	12, 72, 731	94, 1A3
Esadecimale	16	0 1 2 3 4 5 6 7 8 9 A B C D E F	3A, 51, F3	4G9

In genere ci si riferisce ai simboli che formano un numero con il termine generico di **cifra**, fatta eccezione per i numeri decimali, che vengono indicati come **digit**, ed i numeri binari indicati come **bit**.
Uno stesso numero potrebbe essere interpretato in diversi sistemi numerici, come il numero 101 che può essere inteso come numero decimale (centouno) oppure binario (uno-zero-uno). Per indicare in modo univoco l'appartenenza di un numero ad un sistema si scrive: $(101)_{\mathbf{2}}$ per indicare che 101 è un numero binario; $(101)_{\mathbf{16}}$ per indicare che è esadecimale oppure $(101)_{\mathbf{10}}$ se deve essere interpretato come numero decimale.

Formato di un numero decimale.

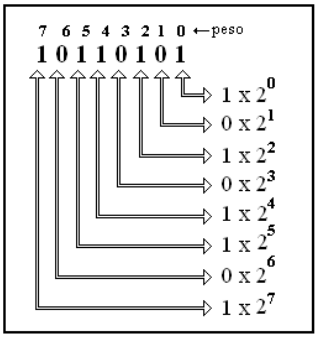


Un numero decimale è strutturato in termini di unità, decine, centinai, ecc. ecc. Consideriamo ad esempio il numero 359. Sappiamo che ogni cifra ha un suo ben preciso '**peso**'. Nell'esempio indicato il 9 rappresentano le unità, cioè quante volte $10^{\mathbf{0}}$. Il 5 rappresentano le decine, ossia quante volte $10^{\mathbf{1}}$. La cifra 3 rappresentano le centinaia $10^{\mathbf{2}}$.
Quindi possiamo dire che ciascuna cifra ha un peso che parte da 0 e cresce spostandosi verso sinistra in base alla posizione occupata dal numero. La cifra a destra è quella meno significativa, quella a sinistra è la cifra più significativa. Il **valore numerico** che dobbiamo assegnare al numero 359 (questa è una sequenza di simboli!) si calcola nel modo seguente:

$$359 = 3\text{x}\mathbf{10}^2 + 5\text{x}\mathbf{10}^1 + 9\text{x}\mathbf{10}^0 = \text{valore } 359$$

L'analisi appena fatta deve valere anche per gli atri sistemi numerici, vediamo come.

Formato di un numero binario



Il sistema binario è costituto dalle sole 2 cifre del sistema Bin={0,1}.
Un numero binario deve essere scritto in pacchetti di 8 bit, o multipli di 8. Se il numero di bit è inferiore a 8, si aggiungono tanti zeri a sinistra del numero fino a completare il pacchetto di 8. Consideriamo il numero binario 10110101. Il bit meno significativo, quello a destra, rappresenta le 'unità' nel sistema binario.
In analogia con il sistema decimale in cui sono pari a 10^0, nel sistema binario, con base uguale a 2, le unità sono 2^0. Al bit meno significativo segue un bit che rappresenta le così dette 'decine' e quindi quante volte 2^1. Il bit che segue ancora rappresenta le così dette "centinaia" e quindi quante volte 2^2. In questo modo ad ogni bit, in funzione della sua posizione, è assegnato un peso che ne determina un preciso valore. Vediamo come si calcola il valore da assegnare al numero binario dell'esempio citato.

$$10110101 = 1\text{x}\mathbf{2}^7 + 0\text{x}\mathbf{2}^6 + 1\text{x}\mathbf{2}^5 + 1\text{x}\mathbf{2}^4 + 0\text{x}\mathbf{2}^3 + 1\text{x}\mathbf{2}^2 + 0\text{x}\mathbf{2}^1 + 1\text{x}\mathbf{2}^0 = 181$$

Formato di un numero esadecimale

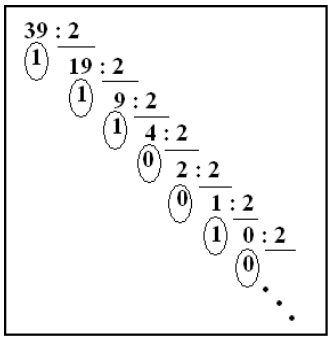
Quanto detto in precedenza per il sistema decimale e binario deve essere applicabile anche al sistema esadecimale. Il sistema esadecimale è costituito dalle 16 cifre del sistema Hex={0,1,2...9,A,B,C,D,E,F}.
Ogni cifra del numero ha un suo peso che partendo da zero, per la cifra meno significativa, aumenta via via che si considera la cifra più significativa verso sinistra. Ad esempio, consideriamo il numero 3C5. La cifra **5** ha peso 0 e vale 16^0 volte 5; la cifra **C** ha peso 1 e vale 16^1 volte il valore di C, cioè 12; la cifra **3** ha un peso 2 e vale $2*16^2$. Il valore da assegnare al numero deve essere calcolato come:

$$3C5 = 3x\mathbf{16}^2 + 12x\mathbf{16}^1 + 5x\mathbf{16}^0 = 965$$

E' del tutto intuitivo l'estensione al caso del sistema numerico ottale.

Conversione da Decimale a Binario

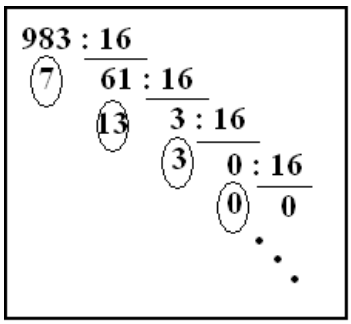
Per ottenere la conversione di un numero decimale in binario si devono eseguire una serie di divisioni per 2, fino a quando il resto non si annulla. Vediamo la figura riportata qui a lato. Come si vede sono stati cerchiati i resti di ciascuna divisione. Questi resti costituiscono il numero binario. Continuando sempre la divisione, da un certo punto in poi i resti sono sempre nulli, come pure i quozienti. Questo fatto ci fa capire che gli zeri che si ottengono, e che fanno parte del numero, devono collocarsi a sinistra del numero binario in una posizione che non alterano il valore del numero. Pertanto leggendo i resti ottenuti dal basso verso l'alto e scrivendoli da sinistra a destra otteniamo la conversione binaria. Quindi:



$$(\ 39\ )_{10} = (\ 00100111\ )_2$$

Conversione Decimale a Esadecimale

La conversione da decimale ad esadecimale si ottiene dividendo il numero decimale per 16 tante volte fino a quando il quoziente non è nullo. Si osservi la figura qui a lato. Il numero decimale 983 viene diviso tante volte per 16 fino ad ottenere resto e quoziente nulli. Il numero esadecimale deve essere letto, come per il caso binario, dal basso verso l'alto e scritto da destra a sinistra. Bisogna fare attenzione a convertire i resti superiore a 9 in una cifra esadecimale. Nel caso in figura il resto **13** deve essere scritto come simbolo esadecimale **D**, poiché 10=A, 11=B, ecc. ecc. Quindi la conversione in esadecimale del numero 983 è 3D7 (si deve leggere: tre-di-sette).



Conversioni Binario-Esadecimale e Esadecimale-Binario

Ciascuna cifra del sistema esadecimale richiede 4 bit per poterla rappresentare in binario. Poiché il sistema esadecimale è fatto di 16 simboli, il numero di bit per poter rappresentare 16 oggetti è 4. Infatti 2^4=16. La tabella riportata a lato permette di vedere tutte le possibili combinazioni di 4 bit con l'associazione della cifra esadecimale (Hexadecimal). Quindi, un numero espresso in binario con un numero di bit multiplo di 8 può essere subito convertito in esadecimale dividendo i bit del numero in pacchetti di 4 bit e convertendo ciascun pacchetto nella corrispondente cifra esadecimale. Ad esempio:

BIN	HEX
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

111011 → 00111011 → 0011-1011 → 3-B → 3B

La prima espressione non è un numero formato da 8 bit. La seconda espressione è un numero formato da 8 bit, sono stati aggiunti 2 bit a sinistra. La terza espressione è la divisione del numero in due pacchetti di 4 bit. La quarta espressione associa a ciascun pacchetto di 4 bit la corrispondente cifra esadecimale. L'ultima espressione è la conversione in esadecimale del numero bonario.
Allo stesso modo un numero esadecimale può essere subito convertito in numero binario semplicemente sostituendo ogni cifra esadecimale con la corrispondente sequenza di 4 bit. Ad esempio:

3A9 → 0011-1010-1001 → 0000-0011-1010-1001 → 0000001110101001

La prima espressione è un numero esadecimale. La seconda espressione sostituisce ad ogni cifra esadecimale l'equivalente in binario. La terza espressione complementa il numero di bit a multiplo di 8. L'ultima espressione è la conversione in binario.

Somma binaria

La somma di due bit soddisfa la seguente tabella.

Bit	Somma	Riporto
0 + 0	**0**	**0**
0 + 1	**1**	**0**
1 + 0	**1**	**0**
1 + 1	**0**	**1**

La tabella espone il caso della somma di due soli bit. Se dobbiamo sommare due numeri in binario la tabella deve essere considerata per ciascuna coppia di bit che sommiamo. Ad esempio

	BIN	HEX
Riporto	1 0 0 1 1 0 0 0	
Primo Numero	**0 1 0 0 1 1 0 1**	4D
Sec. Numero	**0 1 1 0 1 1 1 0**	6E
Somma	1 0 1 1 1 0 1 1	BB

Dall'esempio si vede che, partendo dalla coppia dei bit meno significativi, il bit somma è ottenuto sulla base della regola esposta dalla precedente tabella.

Complemento a 1 (C_1) di un numero binario

Per complemento ad 1 di un numero binario si intende il numero binario che si ottiene complementando ciascun bit. Il complemento di 0 è 1, il complemento di 1 è 0. In questo modo:

$$C_1(\ 011011\ ) = 100100$$

Complemento a 2 (C_2) di un numero binario

Il complemento a 2 di un numero binario è il numero binario che si ottiene sommando 1 al complemento ad 1 del numero binario. Ossia:

$$C_2(\ 1011101\ ) = C_1(\ 1011101\ ) + 1$$

Numeri positivi e negativi.

Quando parliamo di un numero binario dobbiamo sempre pensarlo come formato da un numero di bit multiplo di 8. Consideriamo un numero binario x = 01110011. Calcoliamo il complemento a 2 di questo numero, cioè:

$$C_2(\ x\ ) = C_2(\ 01110011\ ) = C_1(\ 01110011\ ) + 1 = 10001100 + 00000001 = 10001101$$

Ossia:

$$C_2(x) = 10001101$$

Dobbiamo chiederci che significato ha questo numero. Facciamo la somma del numero x e del suo complemento a 2, considerando sempre la somma in 8 bit. Ossia:

X	: 0 1 1 1 0 0 1 1	+
$C_2(X)$	: 1 0 0 0 1 1 0 1	=
(1) 0 0 0 0 0 0 0 0		

Dal risultato ottenuto si vede che, **a parte il nono bit** messo tra parentesi, il risultato della somma è nullo. In altre parole $C_2(x)$ è l'**opposto** di x. Il numero x in decimale è 115, allora il complemento a 2 di x è un numero negativo ed è -115. Si assume che il bit 7 è il bit di segno: se questo è 1 il numero è da considerarsi negativo.

Sottrazione binaria

Per eseguire una sottrazione di due numeri binari si osservi che la differenza 25 – 18 può essere scritta come 25 + (-18), ossia come una **somma** tra un numero positivo ed uno negativo. Quindi basta ricavare il numero binario del sottraendo e sommarlo ad minuendo. Facciamo un esempio, supponiamo di voler fare 25-18. Calcoliamo l'espressione binaria di -18.

$$-18 = C_2(18) = C_1(18) + 1 = C_1(00010010) + 1 = 11101101 + 00000001 = 11101110$$

Quindi facciamo la somma:

```
 25 =  0 0 0 1 1 0 0 1   +
-18 =  1 1 1 0 1 1 1 0   =
  (1)  0 0 0 0 0 1 1 1   =   7
```

Osserviamo che facendo la somma c'è il riporto di un 9° bit, messo tra parentesi, che deve essere tralasciato. Questo bit è indicato come bit di **Overflow** e sta ad indicare che la somma non può essere espressa in 8 bit. Se l'ottavo bit fosse stato 1 significherebbe che il risultato è negativo. Ad esempio eseguiamo 18-25:

```
 18 = 0 0 0 1 0 0 1 0  +
-25 = 1 1 1 0 0 1 1 1  =
      1 1 1 1 1 0 0 1 =  -7
```

Come per la somma binaria possiamo ricavare le regole che permettono la sottrazione di numeri in formato binario. Osserviamo la seguente tabella della sottrazione binaria.

Bit	Sottrazione	Riporto
0 – 0	**0**	**0**
0 – 1	**1**	**1**
1 – 0	**1**	**0**
1 - 1	**0**	**0**

Rispetta alla tabella della somma, il riporto esiste solo nel caso di 0-1. Nella sottrazione, a differenza della somma, bisogna prima eseguire la sottrazione tra minuendo e sottraendo e poi considerare il riporto. Osserviamo l'esempio che segue.

Sottraendo	**1 1 0 0 0 1 0 1**
Minuendo	**0 1 1 0 1 1 1 0**
Riporto	0 1 1 1 1 1 1 0
Sottrazione	**0 1 0 1 0 1 1 1**

Si osservi che i riporti sono sfasati verso sinistra di 1 bit. Inoltre si osservi che rispetto alla somma in cui la riga dei riporti è la prima riga, nella sottrazione la riga dei riporti è la terza riga, il riporto è l'ultimo valore da sottrarre.

Funzioni logiche su numeri binari

Le funzioni logiche di base sono le seguenti.

Funzione	Descizione	Esempio
NOT	Complementa i singoli bit, ossia gli 1→0 e gli 0→1.	Se X=0110, cioè X=6, allora Y=NOT(X)=1001, ossia Y=9. Si osservi che la somma X+Y=**X+NOT(X)** = 2^4-1 =15, con 4 il numero dei bit che formano il numero.
OR	La funzione OR agisce al minimo su 2 variabili ed esegue la funzione OR su ciascuna coppia di bit.	Se X=00110101 ed Y=10000111 allora la funzione OR fornisce il valore Z: Z = X OR Y = OR(00**1**10101 , 10**0**00111) = 10**1**10111
AND	La funzione AND agisce al minimo su due variabili e fornisce un risultato che la funzione AND applicata a ciascuna coppia di bit delle variabili X e Y.	Supponiamo X=00110101 ed Y=10000111 allora la funzione AND fornisce il valore Z: Z = X AND Y) = AND(00110101 , 10000111) = 00000101
XOR	La funzione XOR, indicata anche con ExOR, è la funzione OR escusiva.	Supponiamo X=00110101 ed Y=10000111 allora la funzione XOR fornisce il valore Z: Z = X XOR Y = XOR(00110101 , 10000111) = 10110010

In base alla tabella precedente, possiamo valutare funzioni più complesse.
Ad esempio valutiamo l'espressione **Z = NOT(X AND Y).** Questo significa che dobbiamo prima valutare il risultato dell'operazione più interna X AND Y e poi su questo risultato dobbiamo applicare la funzione NOT.
Il nome delle funzioni logiche devono essere sempre scritte in maiuscolo e preceduto da uno o due spazi bianchi. Lo stesso anche dopo il nome.

Ad esempio, valutiamo la seguente espressione algebrica: F = NOT[(X AND Y) XOR (NOT(X))]
Supponiamo che X=3A, Y = B5. Per vedere il valore di F si devono valutare prima le espressioni più interne e, via via, quelle più esterne.

NOT(X) = NOT(00111010) = 11000101 = C5
X AND Y = 3A AND B5 = $(30)_{16}$
F = NOT(30 XOR C5) = F5

Rappresentazione binaria dei numeri reali

Possiamo avere due modi per convertire un numero reale in un formato binario. In particolar modo possiamo avere una conversione indicata come ***Fixed Point*** ed una indicata come ***Floating Point***.

Fixed Point. La conversione Fixed Point riserva un numero N_1 di bit per la conversione della parte intera del numero reale ed un numero N_2 di bit per la conversione della parte frazionaria. Ad esempio si può pensare di riservare 1 byte (8 bit) per la conversione della parte intera ed 1 byte per la conversione della parte frazionaria, in questo caso un numero reale in binario potrebbe essere 00110110.10110011

La regola da applicare per la conversione dei numeri reali è la stessa di quella esistente per i numeri decimali. Consideriamo il numero 3.459, nel consueto sistema decimale possiamo scriverlo come:

$$3.459 = 3*\mathbf{10^0} \,.\, 4*\mathbf{10^{-1}}+5*\mathbf{10^{-2}}+9*\mathbf{10^{-2}}$$

In questa espressione si è messo in evidenza la potenza del 10 che moltiplicare ogni digit per ottenere il corretto numero decimale.

Questa regola **deve valere** anche nel il sistema binario.
Ad esempio, il numero 1101.10011 viene trasformato in decimale nel seguente modo:

$$1101.10011 = 1*\mathbf{2^3}+1*\mathbf{2^2}+0*\mathbf{2^1}+1*\mathbf{2^0} \,.\, 1*\mathbf{2^{-1}}+0*\mathbf{2^{-2}}+0*\mathbf{2^{-3}}+1*\mathbf{2^{-4}}+1*\mathbf{2^{-5}}=13.59375$$

In questa espressione il colore rosso evidenza la parte intera, quello verde la parte frazionaria.

Per ricavare la conversione in binario di un numero reale si opera nel modo seguente.
La parte intera viene convertita in binario nel modo già descritto. La parte frazionata si moltiplica per 2 e ad ogni passaggio si isola la parte intera, che può essere o uno ZERO (0) oppure un UNO (1) , e che rappresenta il bit di conversione. Lo schema che segue illustra tale concetto.

Decimale moltiplicato per 2	Bit
0.59375 * 2 = 1.1875 → **1** + 0.1875	1
0.1875 *2 = 0.375 → **0**+ 0.375	0
0.375 * 2 = 0.75 → **0** + 0.75	0
0.75 * 2 = 1.5 → **1** + 0.5	1
0.5 * 2 = 1.0 → **1** + 0.0	1

L'ultima colonna rappresenta la conversione in binario del numero decimale 0.59374

Bisogna fare una piccola considerazione. Il numero decimale ha una parte intera ed una frazionata. Entrambe devono essere scritte in pacchetti di 8 bit, o multipli di 8.

Supponiamo di effettuare una codifica in un formato di 8 bit per la parte intera e di 8 bit per la parte decimale, diciamo una conversione in 2 Byte. In questo caso il nostro numero non potrà essere più grande di 127.996 e non può essere più piccolo di -127.996, avendo considerato il bit 7 come bit di segno. Inoltre con il formato appena detto non riusciamo a rappresentare tutti i numeri veramente esistenti tra i limiti indicati.

Come esempio facciamo la conversione del numero 7.31. La parte intera sarà rappresentata dal numero binario 00000111, assumendo una conversione in 8 bit, la parte decimale sarà rappresentata dal numero 01001111 anch'essa convertita in 8 bit. Quindi:

7.31 = 00000111.01001111

Se proviamo a fare l'operazione inversa, conversione da binario a decimale, otteniamo il numero 7.3085. Questo ci fa capire che nel passaggio da decimale a binario la conversione, fermandosi ad un numero finito di bit, ha inevitabilmente introdotto un'approssimazione nella codifica del numero e quindi un errore da tener presente nei calcoli.

Errore della rappresentazione binaria

Supponiamo di avere una rappresentazione di numeri reali nel formato di 2 Byte, uno per la parte intera ed uno per la parte frazionaria. Tralasciamo la conversione della parte intera supponendola nulla. La conversione della parte frazionaria diventa sempre più precisa man mano che aumentano i bit di conversione. Infatti, per fissare le idee consideriamo il numero x=0.137.
Una prima conversione a 4 bit fornisce il numero binario x=0.0010, mentre a 7 bit x=0.0010001. Nel primo caso se riconvertiamo in decimale otteniamo x=0.125 e nel secondo caso x=0.132. In altre parole, numeri diversi da quello iniziale. Quindi la conversione in binario introduce un'approssimazione con un'inevitabile errore. Nel primo caso otteniamo un errore relativo del -8.7% e nel secondo caso del -3.6%. Quindi si vede che con l'aumentare del numero dei bit si abbassa sempre di più l'errore che commettiamo nella conversione. Un errore ineliminabile a meno che non considerassimo un numero infinito di bit per la conversione.

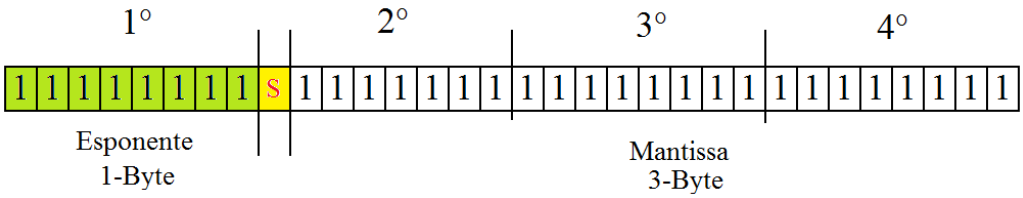
Floating Point(*). Il metodo Floating Point, più sofisticato del precedente, nasce dall'esigenza di aumentare l'intervallo di rappresentazione dei numeri reali e, quindi, di abbassare l'errore introdotto dal numero finito di bit usati per la conversione.
In analogia con il sistema decimale, i numeri possono essere espressi in notazione scientifica: un numero frazionario per una potenza del 10. Ad esempio, il numero 234.187 può essere scritto come 2.34187×10^2.
Allo stesso modo, consideriamo la conversione binaria del numero 7.31 dell'esempio fatto pocanzi. Nella notazione Fixed-Point abbiamo scritto: 7.31= 00000111.01001111 che può essere scritto nel seguente modo:

$$\mathbf{7.31 = 1.1101001111 \times 2^2 = 1.1101001\text{-}11100000 \times 2^2}$$

Nell'ultima espressione si sono aggiunti degli zeri per formare due pacchetti da 8 bit per la parte decimale. La struttura di un numero reale in formato binario Floating-Point deriva da queste considerazioni:

1) Occorre un byte per memorizzare l'esponente del 2, incluso il segno dell'esponente
2) Occorre un bit per memorizzare il segno del numero
3) Occorrono un certo numero di bit per memorizzare il numero stesso o mantissa.

In genere si assume il seguente formato per memorizzare un numero reale con un'approssimazione definita a "singola precisione":



In questo schema a 32 bit (4 Byte), 1 Byte è riservato per l'esponente, 1 bit per memorizzare il segno del numero, i restanti 23 bit per la memorizzazione della mantissa.

(*) Formato di codifica binaria IEEE 754
IEEE = Institute of Electrical and Electronics Engineers