INFORMATICA PER BIOTECNOLOGIE

Rappresentazione dell'informazione negli elaboratori

Il sistema di numerazione comunemente impiegato è quello *posizionale in base 10* (decimale).
Il termine ***posizionale*** indica che il valore assunto da ciascuna cifra dipende dalla sua posizione nella sequenza.
A ciascuna cifra è assegnato un valore che dipende, oltre che dalla posizione, anche dalla base del sistema di numerazione.
La ***base*** di un sistema di numerazione posizionale corrisponde al numero di simboli usati per scrivere i numeri ed indica quante unità di un certo ordine sono necessarie per formare un'unità di ordine immediatamente superiore.

Il sistema decimale si basa su 10 simboli diversi (chiamati "numerali"), corrispondenti a:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

I numeri si possono rappresentare anche con altri sistemi di numerazione posizionali; in generale se B è la base di un sistema di numerazione, il sistema posizionale ad esso associato avrà B cifre di valore:

$$0, 1, 2, \ldots\ldots B_{-2}, B_{-1}$$

In generale il numero in base B:

$$(c_n\ c_{n-1}\ c_{n-2}\ \ldots\ldots c_1\ c_0)_B$$

corrisponde al valore

$$c_n*B^n + c_{n-1}*B^{n-1} + c_{n-2}*B^{n-2} + \ldots\ldots c_1*B^1 + c_0*B^0$$

dove $0<=c_i<B$ $(i=0..n)$

Se il numero non è intero si devono introdurre, per le cifre frazionarie, le potenze negative.

Un sistema di numerazione molto importante è quello in base 2 comunemente detto *binario*. La sua importanza deriva dal fatto che viene utilizzato dai circuiti del calcolatore che eseguono le operazioni aritmetiche in quanto nei circuiti elettronici si preferisce distinguere soltanto due tipi di segnali elettrici.
Le cifre del sistema binario, dette anche *bit*, sono:

0, 1

Il fatto di poter utilizzare due sole cifre facilmente traducibili in segnali elettrici per il calcolatore permette di eseguire rapidamente operazioni anche molto lunghe.
L'utilizzo del sistema binario riguardo solo i circuiti del calcolatore che traduce automaticamente per noi i numeri in ingresso ed in uscita, in maniera da poter lavorare con il sistema di numerazione in base 10.

Ex.

La sequenza 111001 rappresenta in base 2 il numero:

$11001 = 1*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0$

che in base decimale risulta pari a 25

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

Altre basi frequentemente usate sono la base 8, o *ottale*, che usa le cifre:

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

e la base 16, o *esadecimale*, che usa le cifre:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Al simbolo A si associa il valore decimale pari a 10, a B quello pari a 11… a F il valore decimale pari a 15.

Ex.

La sequenza 3D5 rappresenta in base 16 il numero:

$3A5 = 3*16^{2^*} + 10*16^1 + 5*16^0$

che in base decimale risulta pari a 981

$(3A5)_{16} = (933)_{10}$

Negli elaboratori i numeri sono rappresentabili con un numero finito di cifre e quindi non tutti i numeri sono rappresentabili.
In generale nel sistema in base m con n cifre si possono rappresentare i numeri compresi nell'intervallo $[0, m^n-1]$.

Conversione di base

In generale per effettuare la conversione di un numero da sistema decimale ad un qualunque sistema in base B basta dividere il numero per la base B fino a che il quoziente non diventa nullo. I resti di tali divisioni, scritti in ordine inverso, rappresentano le cifre del numero in base B; tali resti avranno valori compresi tra 0 e B–1.

La conversione di un numero da base decimale a binaria si effettua nel seguente modo:

- si divide il numero per 2 e si considera il quoziente ed il resto
- il resto, che può valere 0 o 1, rappresentala cifra binaria di peso minore (bit meno significativo)
- se il quoziente non è nullo si ripete il procedimento, mentre se è nullo il calcolo è terminato ed il numero in sistema binario è dato dalla sequenza dei resti delle divisioni presi in ordine inverso a quello di svolgimento delle divisioni

La cifra meno significativa si indica con *LSB* (Less Significant Bit), mentre quelle più significativa si indica con *MSB* (More Significant Bit).

Ex.

Il numero decimale 157 si può convertire in base 2 effettuando la divisione per 2 fino a che il quoziente diventa 0:

157 : 2 resto 1 (LSB)
78 : 2 resto 0
39 : 2 resto 1
19 : 2 resto 1
9 : 2 resto 1
4 : 2 resto 0
2 : 2 resto 0
1 : 2 resto 1 (MSB)

0

$(157)_{10} = (10011101)_2$

Come verifica si effettua la conversione del numero 10011101 da base binaria a base decimale:

$$10011101 = 1*2^7 + 0*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = (157)_{10}$$

$$(10011101)_2 = (157)_{10}$$

Aritmetica degli elaboratori

Le operazioni nel sistema binario si eseguono con le stesse modalità del sistema decimale.

Somma
Si esegue la somma secondo le seguenti regole:

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 0 con riporto (*carry*) di 1

Ex.

```
  1 1     1
    1 1 0 0 1 +
      1 1 0 1 =
  ---------------
  1 0 0 1 1 0
```

Quando si rappresentano i numeri con una quantità fissa *n* di cifre può succedere che il risultato di una somma non sia rappresentabile su n cifre; questa condizione è detta di *overflow*.

Sottrazione
Si esegue la sottrazione secondo le seguenti regole:

- 0 - 0 = 0
- 1 - 0 = 1

- 1 - 1 = 0
- 0 - 1 = 1 con prestito (*borrow*) di 1

Ex.

```
  1 1 1 0 0 1 -
    1 0 1 0 1 =
 ---------------
  1 0 0 1 0 0
```

Moltiplicazione
La moltiplicazione tra numeri binari si effettua con lo stesso algoritmo utilizzato per i numeri decimali, detto di somma e scorrimento.
Si esegue la moltiplicazione secondo le seguenti regole:

- 0 x 0 = 0
- 1 x 0 = 0
- 0 x 1 = 0
- 1 x 1 = 1

Ex.

```
    1 1 1 0 0 1 x
          1 0 1 =
 -----------------
    1 1 1 0 0 1
  0 0 0 0 0 0
1 1 1 0 0 1
-------------------
1 0 0 0 1 1 1 0 1
```

Rappresentazione dei numeri relativi

Se il calcolatore pensa di aver bisogno sia dei numeri positivi che negativi, li rappresenterà con il segno: 0 per i numeri positivi e 1 per quelli negativi.

Il segno rappresenta la cifra più a sinistra, mentre le altre cifre indicano il modulo, questa è la *rappresentazione in modulo e segno*.

Ex.

0 1 1 = +3
1 1 1 = -3

Per eliminare la presenza di due zeri (con la rappresentazione in modulo e segno esiste uno zero negativo = 1 0 0 ed uno zero positivo = 0 0 0) e per effettuare le operazioni secondo le regole conosciute anche con i numeri negativi, si usa la *rappresentazione in complemento a due*.

In tale rappresentazione i numeri positivi sono gli stessi della rappresentazione in modulo e segno, mentre per quelli negativi si procede nel seguente modo:

- si inverte ogni cifra del numero dato
- si somma 1

Ex.

+3 = 0 1 1
-3 = 1 0 0 + 1 = 1 0 1

Molti calcolatori usano la rappresentazione in complemento a due in quanto consente di effettuare somma e sottrazione con gli stessi procedimenti ed utilizzando gli stessi circuiti.

In generale, dati *n* bit, si possono rappresentare:

- i numeri assoluti compresi tra 0 e $2^n - 1$
- in modulo e segno i numeri compresi tra $-2^{n-1} - 1$ e $2^{n-1} - 1$
- in complemento a due i numeri compresi tra -2^{n-1} e $2^{n-1} - 1$

Nel caso in cui la somma di due numeri positivi generi overflow, per segnalare l'errore ai circuiti di controllo il risultato è un numero negativo (ovviamente errato !).
I dettagli sulle rappresentazioni con segno sono sviluppati piu' avanti.

Rappresentazione dei numeri frazionari

Le cifre che compongono la parte frazionaria hanno un peso pari ad una potenza negativa.

In generale, un numero in base B:

$$c_n\ c_{n-1}\ c_{n-2}\ \ldots..\ c_1\ c_0\ .\ c_{-1}\ c_{-2}\ c_{-m}$$

è pari a

$$c_n*B^n + c_{n-1}*B^{n-1} + \ldots.. + c_1*B^1 + c_0*B^0\ .\ c_{-1}*B^{-1} + c_{-2}*B^{-2} + \ldots.. + c_{-m}*B^{-m}$$

Poiché gli elaboratori codificano ogni dato unicamente tramite “0” e “1”, e non possono utilizzare esplicitamente il simbolo “.”, per separare la parte intera da quella frazionaria di un numero si utilizza:

- rappresentazione in *virgola fissa* (fixed-point)
- rappresentazione in *virgola mobile* (floating-point)

Rappresentazione in virgola fissa

Si assume che la posizione della virgola sia fissata in un preciso punto all’interno della sequenza.

Ex.

Si supponga di utilizzare numeri con 8 bit e che la virgola sia posizionata in modo da utilizzare 6 bit per la parte intera e 2 per quella frazionaria.

1 1 0 0 0 1 1 0

rappresenta il numero in base binaria

1 1 0 0 0 1.1 0

Per convertire da sistema decimale a sistema binario in virgola fissa in cui si impiegano m bit per la parte intera ed n per la parte frazionaria si segue il seguente algoritmo:

- la parte intera viene convertita in un numero binario a m bit con l'algoritmo adottato per i numeri assoluti
- la parte frazionaria viene convertita in un numero a n bit nel seguente modo:
    - la si moltiplica per 2 e si considera la parte intera che costituisce la rappresentazione binaria a partire dal bit più significativo; si itera questo procedimento fino ad ottenere parte frazionaria nulla oppure si ricavano gli n bit disponibili

Ex.

Convertire nella notazione a virgola fissa con 5 bit per la parte intera e 5 per la parte frazionaria il numero decimale 22.412

per la parte intera:

22 : 2 resto 0 (LSB)
11 : 2 resto 1
5 : 2 resto 1
2 : 2 resto 0
1 : 2 resto 1 (MSB)
0

$(22)_{10} = (10110)_2$

per la parte frazionaria:

0.412 x 2 = 0.824 0
0.824 x 2 = 1.648 1
1.648 x 2 = 1.296 1
1.296 x 2 = 0.592 0
0.592 x 2 = 1.184 1

$(22.412)_{10} = 1011001101$

ESERCIZI

1) convertire il numero binario 100110001 in base 10, 4 8 e 16.

N.B. essendo $4 = 2^2$ si analizzano le cifre a due a due facendo corrispondere ad esse una cifra in base 4

2) convertire il numero $(201102)_3$ in base 10 e 9.

3) convertire il numero $(341)_5$ nel sistema binario.

Soluz: (341:2 = 143 resto = 0 } numeratore in base 5
143:2 = 44 resto = 0
44:2 = 22 resto = 0
22:2 = 11 resto = 0
11:2 = 3 resto = 0
3:2 = 1 resto = 1
1:2 = 0 resto = 1

RISULTATO $(341)_5 = (1100000)_2$

Infatti $(341)_5 = (96)_{10}$ e 96 vale appunto $(1100000)_2$

4) eseguire le seguenti operazioni:

11001 – 110
100110 – 10001
1101 x 101

5) rappresentare in complemento a due su 6 bit i seguenti numeri:
+14, -2, +9, -9

6) rappresentare in modulo e segno su 6 bit i seguenti numeri:
+14, -2, +9, -9

DETTAGLI sul cambio di base

UN numero V in base b:

(1) $a_n\ a_{n-1}\ a_{n-2}\ \ldots..\ a_1\ a_0\ .\ a_{-1}\ a_{-2}\ a_{-m}$

con $0 <= a_i <= (b-1)$

dove a_n è il coefficiente di b^n

ovvero

$$V = a_n*b^n + a_{n-1}*b^{n-1} + .. + a_1*b^1 + a_0*b^0 + a_{-1}*b^{-1} + a_{-2}*b^{-2} + \ldots + a_{-m}*b^{-m}$$

Parte intera — Parte frazionaria

Separiamo la parte intera e la parte frazionaria di V:

$$V = I_b + F_b$$

(2) $I_b = a_n*b^n + a_{n-1}*b^{n-1} + .. + a_1*b^1 + a_0$ (Parte INT)

(3) $F_b = a_{-1}*b^{-1} + a_{-2}*b^{-2} + \ldots + a_{-m}*b^{-m}$ (PARTE FRAZ.)

I e F in base c (la nuova base) :

(4) $I_C = a_s * c^s + a_{s-1} * c^{s-1} + .. + a_1 * c^1 + a_0$ **(Parte INT)**

(5) $F_c = a_{-1} * c^{-1} + a_{-2} * c^{-2} + \ldots + a_{-k} * c^{-k}$ …….
(PARTE FRAZ.)

Dall'algebra

(6) $I = R_0 + c * Q_0$
(esempio 15 = 1 + 2*7)

dove Q_0 e' il quoziente e R_0 e' il resto ($0 <= R_0 < c$)

Ovvero $I/c = R_0/c + Q_0$
(esempio 15/2 = 1/2 + 7)

dove = I =15 **in base 10**
c=2 la nuova base
R_0=1 Resto
Q_0=7 Quoziente intero

La formula (4) si puo' riscrivere cosi':

$$I_c = a_0 + c*(a_1+a_2*c^1 + a_3*c^2 +\ldots. a_s*c^{s-1}) \qquad (7)$$

Osservando la (6) e la (7) si nota che il secondo membro di entrambe le formule e' simile alla (6) se poniamo

$$R_0 = a_0 \qquad (8)$$

$$Q_0 = a_1+a_2*c^1 + a_3*c^2 +\ldots. a_s*c^{s-1}$$

Ora basta ripetere il procedimento su eseguito considerando Q_0 come nuovo numero da scomporre

$$Q_0 = a_1+a_2*c^1 + a_3*c^2 +\ldots. a_s*c^{s-1}$$

$$Q_0 = R_1 + c * Q_1$$

$$Q_0 = a_1 + c*(a_2 + a_3*c^1 +\ldots. a_s*c^{s-2})$$

$$R_1 = a_1$$

$$Q_1 = a_2 + a_3*c^1 +\ldots. a_s*c^{s-2} \qquad (9)$$

… continuando ancora si ottiene una sequenza

$$(R_h \; R_{h-1} \; \ldots \; R_1 \; R_0)$$

che rappresenta i coefficienti di I_c nella (4).

Esempio:

325(base 10) ----------→ (???????) base 2

325 = 162*2 + 1 R_0=1
162 = 81*2 + 0 R_1=0
81 = 40*2 + 1 R_2=1
40 = 20*2 + 0 R_3=0
20 = 10*2 + 0 R_4=0
10 = 5*2 + 0 R_5=0
5 = 2*2 + 1 R_6=1
2 = 1*2 + 0 R_7=0
1 = 0*2 + 1 R_8=1
==========

325(base 10) ----------→ 101000101 (base 2)

R_0 e' il meno significativo.
R_8 e' quello piu' significativo

PARTE FRAZIONARIA

Per la parte frazionaria

$$F_c = a_{-1} * c^{-1} + a_{-2} * c^{-2} + \ldots + a_{-n} * c^{-n} + \ldots \quad \text{(PARTE FRAZ.)} \quad (10)$$

In maniera analoga si puo' considerare la seguente relazione

$$F^*c = I' + F' \qquad (11)$$

esempio $(0{,}74 * 2) = 1 + 0{,}48$ (in base 10)

I' = eventuale parte intera >=0

F'= eventuale parte frazionaria

Dalla (10) osserviamo che :

$$F^*c = a_{-1} + a_{-2} * c^{-1} + \ldots + a_{-n} * c^{-n+1} + \ldots \qquad (12)$$

poniamo $a_{-1} = I'$

e $a_{-2} * c^{-1} + \ldots + a_{-n} * c^{-n+1} + \ldots = F'$

e continuando..

$F'' * c = I'' + F_c''$

$F' * c = a_{-2} + a_{-3} * c^{-1} + \ldots + a_{-n} * c^{-n+2} + \ldots.$

poniamo $a_{-2} = I''$

$a_{-3} * c^{-1} + \ldots + a_{-n} * c^{-n+2} + \ldots. = F_c''$

Quindi otterremo una sequenza di

$a_{-1}, a_{-2}, a_{-3}, \ldots, a_{-h}, \ldots$

che rappresentano i coefficienti della parte frazionaria del numero nella nuova base c.

esempio:

0,875(base 10) ----------→ (???????) base 2

0,875*2 = 1,750	$a_{-1}=1$	F=0,750
0,750*2 = 1,5	$a_{-2}=1$	F=0,5
0,5*2 = 1,0	$a_{-3}=1$	F=0

quindi

$0,875_{(base\ 10)}$ → (0,111) base 2

esempio:

0,076(base 10) ----------→ (???????) base 2

0,076*2 = 0,152	**a_{-1}=0**	**F=0,152**
0,152*2 = 0,304	**a_{-2}=0**	**F=0,304**
0,304*2 = 0,608	**a_{-3}=0**	**F=0,608**
0,608*2 = 1,216	**a_{-4}=1**	**F=0,216**
0,216*2 = 0,432	**a_{-5}=0**	**F=0,432**
0,432*2 = 0,864	**a_{-6}=0**	**F=0,864**
0,864*2 = 1,728	**a_{-7}=1**	**F=0,728**
0,728*2 = 1,456	**a_{-7}=1**	**F=0,456**

……
……

0,076(base 10) ----------→ (0,00010011..) base 2

Notare che non e' preciso (ovvero ad un numero finito di cifre dopo la virgola in base b non corrisponde necessariamente un numero finito di cifre dopo la virgola in un'altra base!)

esempio

	per 2		
0,65625	**1,3125**	**1**	fraz=0,3125
0,3125	0,625	**0**	fraz=0,625
0,625	1,25	**1**	fraz=0,25
0,25	0,5	**0**	fraz=0,5
0,5	1	**1**	fraz=0
0	0	**0**	

quindi **0,65625 (base 10) = 0,101010 (base 2)**

Esercizio:

TRASFORMARE 352 da base 10 a base 2

352 = 176*2 + 0 R_0=0
176 = 88*2 + 0 R_1=0
88 = 44*2 + 0 R_2=0
44 = 22*2 + 0 R_3=0
22 = 11*2 + 0 R_4=0
11 = 5*2 + 1 R_5=1
5 = 2*2 + 1 R_6=1
2 = 1*2 + 1 R_7=0
1 = 0*2 + 1 R_8=1
==========

$(352)_{10}$ ----------→ $(101100000)_2$

R_0 e' il bit meno significativo.
R_8 e' quello piu' significativo

`

RAPPRESENTAZIONE DI NUMERI IN BASE 2,4,8,16

In base a quanto detto finora, e' importante conoscere la seguente regola per passare da una delle seguenti basi: 2,4,8,16 ad altre dello stesso gruppo:

1) data la rappresentazione in esadecimale di un numero intero (esempio F2AB), per ottenere la rappresentazione in base 2 o 4 o 8 non e' necessario calcolarne il valore in base 10.

Procedimento:

base 10	base 2	base 4	base 8	base 16
0	0000	00	00	0
1	0001	01	01	1
2	0010	02	02	2
3	0011	03	03	3
4	0100	10	04	4
5	0101	11	05	5
6	0110	12	06	6
7	0111	13	07	7
8	1000	20	10	8
9	1001	21	11	9
10	1010	22	12	A
11	1011	23	13	B
12	1100	30	14	C
13	1101	31	15	D
14	1110	32	16	E
15	1111	33	17	F

Usando la tabella a sinistra basta trasformare **F2AB** in **1111 0010 1010 1011**

e in questo modo abbiamo ottenuto la trasformazione da base 16 a base 2.
Per passare alla base 4 , basta prendere la rappresentazione in base 2 e raggruppare, partendo da **destra** i simboli a coppie di due : 11 11 00 10 10 10 10 11 e per ogni coppia vedere il corrispondente valore in base 4 (es 11 --> 3 , 10 --> 2 etc . In tal modo otteniamo **33022223** che corrisponde alla rappresentazione in base 4 di F2AB.
Per la base 8 , analogamente si puo' partire dalla rappresentazione binaria raggruppando a 3 a 3 i simboli binari , partendo da **destra**
1 111 001 010 101 011 e per ogni terna si calcola facilmente il valore ottale
011 --> 3
101 --> 5 etc.
otterremo quindi **171253 (in base 8)**

RICORDATE CHE QUESTA REGOLA VALE SOLO PERCHE' LE BASI 2,4,8,16 SONO TUTTE POTENZE DI DUE

NUMERI BINARI CON SEGNO

METODO MODULO E SEGNO

Il metodo *modulo e segno* utilizza il bit piu' significativo per rappresentare il segno, codificando con uno **0** il segno "+" e con un 1 il segno "-" .

Esempio:

2_{10} = 0000 0010

-2_{10}= 1000 0010

Così facendo si ottiene la rappresentazione del numero 0 come

1000 0000 = -0

0000 0000 = +0

Inoltre, eseguendo somme tra numeri con segno diverso, si ottengono risultati diversi, come nell'esempio che segue:

0000 0001 +
1000 1001 =

1000 1010

} ovvero 1 + (-9) = -10 errato!!!

Per evitare l'errore i numeri negativi dovrebbero essere considerati come positivi ma l'operazione da compiere dovrebbe essere una differenza e non una somma.

L'intervallo ottenibile su n bit in modulo e segno e':

$$-(2^{n-1} - 1) <= N <= (2^{n-1} - 1)$$

ex: per n=8 -127 <= N <= 127
per n=16 -32767 <= N <= 32767
per n=32 - 2147483647 <= N <= 2147483647

COMPLEMENTO A 1

Il complemento a 1 di un numero binario N su n bit e' pari a

N	se	$0 <= N <= (2^{n-1} -1)$
$2^n - \|N\| - 1$	se	$-(2^{n-1} -1) <= N < 0$

Si ottiene lo stesso risultato invertendo tutti gli n bit di N (da 0 a 1 e viceversa).

Ex: $7_{10} = 0000\ 0111_2$

e $-7_{10} = 1111\ 1000_2$

(che corrisponde a 256 -7 -1=248 11111000 senza segno)

Se un numero e' fornito in complemento a 1, la decodifica si ottiene effettuando nuovamente il complemento a 1 (ottenendo il valore positivo corrispondente).

Questo metodo non risolve il problema del doppio zero (rappresentato come 0000 0000 e come 1111 1111
ma sommando numeri con segno si ottengono risultati corretti a patto di compensare l'attraversamento della doppia rappresentazione dello zero che si verifica quando c'e' un riporto oltre il bit piu' significativo(end around carry); in tal caso bisogna sommare 1 al risultato

C_n

1111 1100 + -3
1111 1101 = -2

ovvero -3+ (-2) = -5 corretto!!!

1 1111 1001 +
1 =

1111 1010

In assenza di riporto, invece:

0000 0101 + 5 +
1111 1000 = -7 =

1111 1101 -2 corretto

L'intervallo dei valori rappresentabili su n bit in complemento a 1 e' lo stesso del modulo e segno:

$$-(2^{n-1} - 1) <= N <= (2^{n-1} - 1)$$

IN SINTESI: I numeri positivi hanno la stessa rappresentazione di quella con modulo e segno, la rappresentazione dei numeri negativi si ottiene dalla rappresentazione del numero positivo invertendo i bit (oppure rappresentando in binario puro il numero $2^n - |N| - 1$.

COMPLEMENTO A 2

Il complemento a 2 di un numero binario N su n bit e' pari a :

N	se	$0<=N<=(2^{n-1}-1)$
$2^n - \|N\|$	se	$-(2^{n-1})<=N<0$

Questo significa che solo i valori negativi subiscono una modifica binaria se l'elaboratore rappresenta i valori numerici in complemento a 2. Per i valori positivi la rappresentazione è uguale alla rappresentazione in binario puro , mentre per i numeri negativi bisogna rappresentarli come il binario puro di $2^n - |N|$.

Si ottiene lo stesso risultato invertendo tutti gli n bit di N e sommando 1 (ovvero effettuando il complemento a 1 e sommando 1)

Ex: $+7_{10} = 0000\ 0111_2$

e $-7_{10} = 1111\ 1000_2 + 1 = 1111\ 1001_2$

Se un numero e' fornito in complemento a 2, per ottenere il valore corrispondente ,cambiato di segno, basta (ri)complementarlo a 2! .

Questo metodo risolve il problema del doppio zero (rappresentato come 0000 0000 . .sommando numeri con segno si ottengono risultati corretti.

```
  1111 1101  +        -3   ( in compl a 2 )
  1111 1110 =         -2   ( in compl a 2 )
 ------------------------
                ovvero   -3+ (-2) = -5 corretto!!!
1 1111 1011     ( l'ultimo bit a sinistra non si considera!)
```

il binario 1111 1011 è la rappresentazione del numero -5 in compl. a 2 . Se lo ricomplementiamo a 2 otterremo 0000 0100 +1 = 00000101 che corrisponde a 5 , ovvero a - (-5).

ESEMPIO

0000 0101 +	5 +
1111 1001 =	-7 =
1111 1110	-2 corretto

L'intervallo dei valori rappresentabili su n bit in complemento a 2 e':

$$-(2^{n-1}) <= N <= (2^{n-1} - 1)$$

Ex: per n=8 bit i valori vanno da –128 a + 127
per n=16 bit i valori vanno da –32768 a + 32767

-128 = 1000 0000
+127 = 0111 1111

Per le somme in compl. a 2 è possibile definire una regola che permette di individuare la condizione di overflow (trabocco) osservando gli ultimi due bit di riporto c_{n-1} e c_n ; se in uno di essi si ha riporto e nell'altro no, si ha overflow.

ESEMPIO CON n = 4 (valori rappresentabili –8 +7)

c_n c_{n-1}
↓ ↓
1 1

1011 +	-5 +
1110 =	-2=
1001	-7 corretto!!

c_{n-1} e c_n concordi no overflow

Ricordate che il risultato deve contenere solo 4 bit .

Esempio

1 0 1 1

1011 + -5 +
1011 = -5=

0110 6 è errato!! c_{n-1} e c_n discordi **overflow!!!!**

(c_n =1 c_{n-1} =0 , si e' verificato overflow (infatti –10 non e' rappresentabile in 4 bit !).

Suggerimento per i calcoli :
Avrete notato che tutti i numeri negativi rappresentati in complemento a 2 hanno il bit piu' significativo a 1.

EX: **1**000 0011 rappresenta un numero negativo in 8 bit in complem. a 2.
Potete dare al bit piu' significativo il valore (-128) mentre gli altri 1 hanno il normale peso positivo. In tal caso il numero negativo rappresentato vale (-128 +2 +1) = -125.
Questo procedimento evita di ricorrere al complemento a 1 e poi sommare 1.

IN SINTESI:

•Se il numero è positivo basta assegnare al bit del segno il valore 0 e codificare sui rimanenti k-1 bit il numero.
•Se il numero è negativo basta rappresentare il suo complemento a 2 su k bit.

Per definizione si chiama complemento a 2 su k bit di un numero negativo quel valore che sommato al valore assoluto del numero da come risultato 2^n.

Esempio con n = 8 bit

1 1 1 1

	1	0	1	1	1	0	0	1
	1	1	0	1	0	0	1	0
	1	0	0	0	1	0	1	1

-71	**+**
-46	**=**
-117	

infatti il compl. a 2 di 1000 1011 e' 0111 0101 che vale 117 in base 10

IL METODO ECCESSO M

DETTO ANCHE **biased.**

Rappresenta un numero negativo/positivo memorizzandolo come somma di se stesso con un valore M normalmente scelto pari a

2^{n-1} oppure $2^{n-1}-1$.

Quindi all'intervallo dei valori del numero originale viene fatto corrispondere un intervallo di numeri positivi (piu' lo zero).

Ad esempio con n=8 e M=128 si ha la rappresentazione "eccesso 128" e i numeri –5 e +5 vengono scritti come

1000 0101 = $(133)_{10}$ = +5 eccesso 128
0111 1011 = $(123)_{10}$ = -5 eccesso 128

In tal modo i numeri da –128 a +127 vengono rappresentati nell'intervallo 0÷255.

Con n=8 e M=127 si ha la rappresentazione "eccesso 127" e i numeri –5 e +5 vengono scritti come

1000 0100 = $(132)_{10}$ = +5 eccesso 127
0111 1010 = $(122)_{10}$ = -5 eccesso 127

In tal modo i numeri da –127 a +128 vengono rappresentati nell'intervallo 0÷255.

In generale data la base b, il valore M puo ' essere tale che

$$0 <= M <= b^n-1$$

Il range rappresentato sara' **$[-M \quad b^n-1-M]$**

se b=2 e M = 128 il range sara' [-128 +127]

NUMERI FLOATING POINT

Il problema del minimo numero di bit necessari a rappresentare un dato valore o ad eseguire determinati calcoli riguarda sia i calcoli finanziari che quelli scientifici.
Un calcolo astronomico potrebbe richiedere sia la massa in grammi di un elettrone (circa $9* 10^{-28}$) sia la massa del sole $2* 10^{33}$ grammi ,una gamma che supera 10^{60}. Si potrebbero rappresentare questi numeri con:
00000000000000000000000000000000,00000000000000000000000009
20000000000000000000000000000000,00000000000000000000000000

e si potrebbero effettuare tutti i calcoli tenendo 34 cifre alla sinistra della virgola decimale e 28 cifre alla sua destra. Questo permetterebbe di avere 62 cifre significative . Su un calcolatore binario si potrebbe usare l'aritmetica a precisione multipla per ottenere piu' cifre significative. La massa del sole pero' non si conosce bene nemmeno con cinque cifre significative per non parlare di 62. In realta' poche sono le misure che si possono (o si devono) effettuare con 62 numeri significativi. Anche se si potessero tenere tutti i risultati intermedi di 62 cifre e poi eliminarne 50 o 60 prima di stampare i risultati finali, si tratterebbe di uno spreco si di tempo di CPU che di memoria.
Quello che server e' un sistema per la rappresentazione dei numeri in cui la gamma di numeri esprimibile sia indipendente dal numero delle cifre significative.

NOTAZIONE SCIENTIFICA

Un modo per separare la gamma dalla precisione consiste nell'esprimere i numeri in notazione scientifica:

$$n = f*10^{e}$$

in cui **f** **si chiama frazione, o mantissa mentre**
e **(intero con segno) e' detto esponente .**

La versione informatica di questa notazione si chiama floating point .

Alcuni esempi

$$3{,}14 = 0{,}314 \times 10^{1}$$

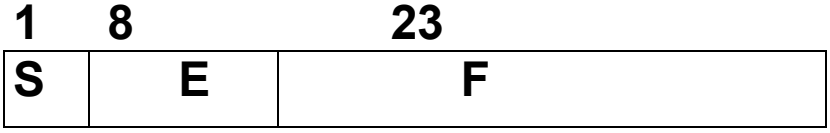
$0{,}000001 = 0{,}1 * 10^{-5} = 1{,}0 * 10^{-6}$
$1941 = 0{,}1941 * 10^{4} = 1{,}941 * 10^{3}$

La gamma viene effettivamente determinata dal numero di cifre dell'esponente e la precisione viene determinata dal numero di cifre della frazione.
Poiche' vi sono piu' modi di rappresentare un dato numero, solitamente viene scelta una forma come standard:

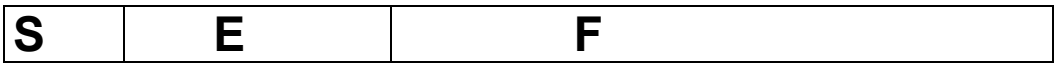
STANDARD FLOATING-POINT IEEE 754 (1985)
(Institute of Electrical and Electronics Engineers)

Fino al 1980 circa ogni produttore aveva il suo formato floating point . Lo standard (accettato dalla maggior parte dei produttori) prevede che un numero X sia espresso in virgola mobile nella forma:

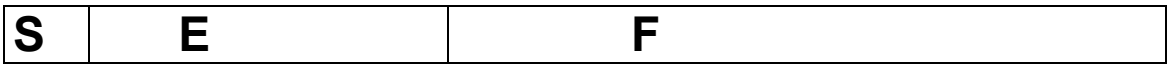
$$X = -1^{S} * 2^{E-M} * 1.F$$



singola precisione (bias =127, E da 0 a 255)



1 11 52 tot=64 bit
doppia precisione(bias=1023, E da 0 a 2047)



1 15 112 tot=128 bit
lunghezza campi in bit

quadrupla precisione
quadrupla precisione(bias=16383, E da 0 a 32767)

dove

S= segno per la mantissa (0 positiva, 1 negativa)

E= esponente espresso come eccesso M (detto bias)

F= mantissa frazionaria, normalizzata in modo tale che la mantissa sia compresa tra 1.00000000…
e 1.111111111…..
in tal modo il primo "1" (quello prima del punto decimale) sara' sottinteso (hidden bit) recuperando così un bit di precisione. Il punto binario e' sottinteso.

ESEMPIO 1
Rappresentare in singola precisione il numero N = -423

S= 1 segno negativo bit segno = 1
423 in binario e' 110100111
Mantissa 1,10100111 * (2 elevato alla 8)
BIAS = 127
ESPONENTE = 127 + 8 = 135 = 1000 0111
F=10100111
Rappresentazione nel calcolatore
1 10000111 10100111000000000000000 (con hidden bit)
S E F

ESEMPIO 2
Rappresentare in singola precisione
il numero x = 42,6875

S= 0 segno POSITIVO bit segno = 0
42,6875 in binario e' 101010,1011
OVVERO 1,010101011 * (2 elevato a 5)
Mantissa 1,010101011
BIAS = 127
ESPONENTE = 127 + 5 = 132 = 1000 0100
F=010101011

Rappresentazione nel calcolatore

0 10000100 **01010101100000000000000** **(con hidden bit)**

S E M

Per quanto riguarda l'underflow e l'overflow nella rappresentazione floating point, si deve tenere presente che esistono delle configurazioni dell'esponente e della mantissa considerate "riservate" . Per esempio, in singola precisione, valori dell'esponente con tutti "0" o con tutti "1" , e con mantissa 0 rappresentano lo zero e l'infinito (tali rappresentazioni non risultano essere normalizzate).
In corrispondenza di una mantissa con una sequenza di bit non tutti nulli e un esponente pieno di "1" ,si intende rappresentare il risultato di una operazione che non non ha senso (per esempio infinito/infinito), queto formato speciale non normalizzato viene chiamato NAN (*Not A Number*) utilizzato per convogliare informazioni circa risultati non validi.

TABELLA RIASSUNTIVA

NORMALIZZATO

±	0<Exp<Max	**Qualsiasi sequenza di 0 e 1**

Denormalizzato

±	0	**Qualsiasi sequenza di 0 e 1**

ZERO

±	0	**00000000000000000000000000000000**

INFINITO

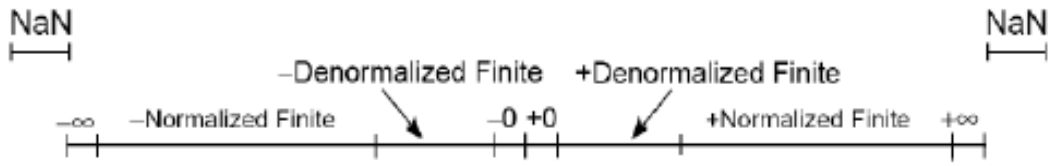
±	111111111111	**00000000000000000000000000000000**

NAN (Not a Number)

±	111111111111	**Qualsiasi sequenza di 0 e 1**

Sintesi dello standard IEEE754

Lo standard IEEE754



Real Number and NaN Encodings For 32-Bit Floating-Point Format

S	E	F			S	E	F
1	0	0	–0	+0	0	0	0
1	0	$0.XXX^2$	–Denormalized Finite	+Denormalized Finite	0	0	$0.XXX^2$
1	1...254	Any Value	–Normalized Finite	+Normalized Finite	0	1...254	Any Value
1	255	0	–∞	+∞	0	255	0
X^1	255	$1.0XX^2$	–SNaN	+SNaN	X^1	255	$1.0XX^2$
X^1	255	1.1XX	–QNaN	+QNaN	X^1	255	1.1XX

NOTES:
1. Sign bit ignored.
2. Fractions must be non-zero.

	Range denormalizzato	Range normalizzato	Decimale
32 bit	Min: 2^{-149} Max:$(1-2^{-23})\times2^{-126}$	Min: 2^{-126} Max: $(2-2^{-23})\times2^{127}$	1.4×10^{-45} 3.4×10^{38}

Ricordiamo che per rappresentare valori normalizzati in Floating Point ieee754 in singola precisione (32 bit), l'esponente E (che comprende l'eccesso M) non può assumere i valori 0 e 255, poiché questi sono riservati alla rappresentazione dei numeri **denormalizzati** che hanno la forma : $X = -1^S * 2^{E-M} * 0.F$ (l'hidden bit vale ora zero). In tal modo si ottiene un underflow verso lo zero più graduale.

CODIFICA DEI CARATTERI

Il calcolatore , nato per manipolare solo numeri, ha trovato ben presto impiego anche nei campi nei quali c'e' l'esigenza di trattare anche caratteri, parole, frasi, etc.
Nasce l'esigenza di ideare dei meccanismi per realizzare una corrispondenza tra la rappresentazione dell'informazione, consona all'uomo e quella propria degli elaboratori digitali.
A differenza di quanto accade con i numeri, per codificare i quali si utilizzano tecniche basate sul loro "valore", per i caratteri le codifiche non possono che essere convenzionali, basate su tabelle di conversione che mettono in relazione i caratteri con sequenze prestabilite di bit. Tali sequenze, in seguito, potranno essere lette come numeri binari puri. In tal modo le tabelle di conversione assumono anche il ruolo di *codifica numerica* dei caratteri.
Con queste tabelle, ciascun elaboratore puo' codificare ,in binario, le informazioni ricevute sotto forma di caratteri, manipolarle e renderle nuovamente nel formato piu' consono all'uomo.
Per evitare il proliferarsi di troppe tabelle di codifica adottate da varie case costruttrici, con conseguente difficolta' di scambio di informazioni, un comitato statunitense che si occupa della standardizzazione (ANSI American National Standard Institute) ha realizzato un codice, denominato ASCII (American Standard Code for Information Interchange). Tale codice codifica 128 simboli utilizzando 7 bit , numerati in decimale da 0 a 127.
I primi 32 caratteri non sono stampabili o visualizzabili, e servono per il controllo della visualizzazione a video, della stampa , trasmissione dati. (1 =SOH = start of header, 4=EOT = End of Text 10=LF=line feed)
I rimanenti caratteri sono visualizzabili e comprendono: le cifre del sistema decimale, i caratteri dell'alfabeto latino maiuscoli e minuscoli, i segni di punteggiatura e altri simboli particolari usati negli USA e nei paesi anglosassoni.

!
34 codice ascii --> "
35 codice ascii --> #
36 codice ascii --> $
37 codice ascii --> %
38 codice ascii --> &
39 codice ascii --> '
40 codice ascii --> (
41 codice ascii -->)
42 codice ascii --> *

44 codice ascii --> ,
88 codice ascii --> X
89 codice ascii --> Y
90 codice ascii --> Z
91 codice ascii --> [
**92 codice ascii --> **
93 codice ascii -->]
94 codice ascii --> ^
45 codice ascii --> -
46 codice ascii --> .
47 codice ascii --> /
55 codice ascii --> 7

43 codice ascii --> +
44 codice ascii --> ,
89 codice ascii --> Y
91 codice ascii --> [
93 codice ascii -->]
45 codice ascii --> -
47 codice ascii --> /
49 codice ascii --> 1
51 codice ascii --> 3
53 codice ascii --> 5
55 codice ascii --> 7
57 codice ascii --> 9

88 codice ascii --> X
90 codice ascii --> Z
**92 codice ascii --> **
94 codice ascii --> ^
46 codice ascii --> .
48 codice ascii --> 0
50 codice ascii --> 2
52 codice ascii --> 4
54 codice ascii --> 6
56 codice ascii --> 8
90 codice ascii --> Z

126 codice ascii --> ~

UN carattere assente sulla tastiera si ottiene con la sequenza
ALT num

esempio ALT 123 fara' comparire {
ALT 125 fara' comparire }

Il codice ASCII presenta alcune proprieta', in particolare codifica le lettere e numeri con valori consecutivi rispettando l'ordine alfabetico. Il valore delle minuscole e' maggiore del valore delle maiuscole.

VAL(A) = 65 VAL(a) = 97
VAL(B) = 66 VAL(b) = 98

……
VAL(Z) = 90 VAL(z) = 122

PROPRIETA' IMPORTANTE

VAL(z) – VAL(Z) = VAL(a) – VAL(A)

La distanza numerica tra una lettera maiuscola e la sua corrispondente minuscola e' costante.

Poiche' l'unita' di memorizzazione dei dati in un calcolatore, e' il byte, utilizzando la codifica ASCII avanza un bit. Questo permette di codificare altri 128 simboli.
Il codice ASCII dal 128 al 255 (HEX 80 FF) vengono sfruttati dai costruttori per altri alfabeti di lingue europee che richiedono simboli aggiuntivi (*ä å Ñ úüÕ*) . Per tali caratteri non e' prevista standardizzazione.
ASCII STANDARD (su 7 bit)
ASCII ESTESO (su 8 bit tutti e 256 caratteri)

CODICE EBCDIC

Extended Binary Coded Decimal Interchange Code
Usato da Calcolatori IBM (mini e Mainframe)
Usa 8 bit (2 nibbles 1 nibble=4bit)

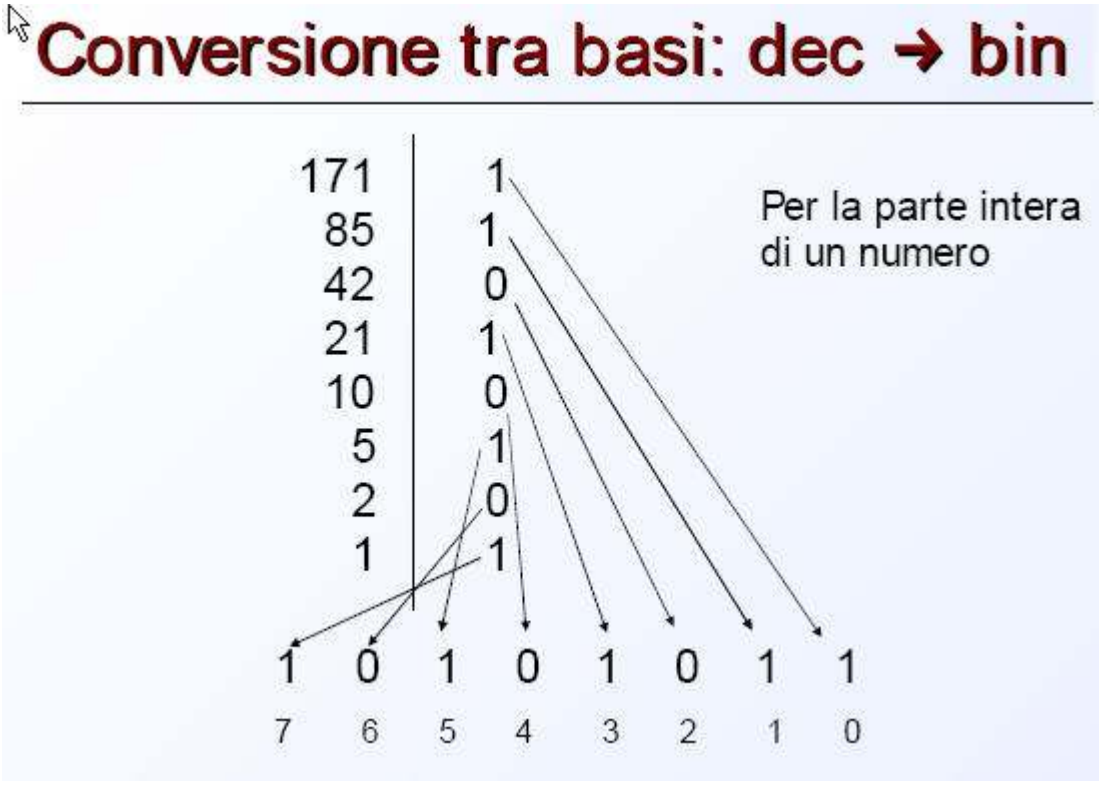
EX	**la lettera "D"**	**0100**	**0101**
		zone	**digit**
	numero 0	**1111**	**0000**
	numero 1	**1111**	**0001**
	....		
	....		
	numero 9	**1111**	**1001**

UNICODE

L'industria dei calcolatori nacque negli stati uniti e porto' cosi' alla creazione dei caratteri ASCII (pronunciato *aschi*) . ASCII va bene per l'inglese ma e' meno adatto per le altre lingue. Il francese, per esempio, ha accenti (per esempio système), il tedesco ha altre punteggiature (per esempio für) e cosi' via . Alcune lingue hanno alfabeti completamente diversi (per esempio il russo e l'arabo). Altre lingue hanno lettere che non si trovano in ASCII come la β tedesca . C'e' il cinese che non ha neanche un alfabeto. La rapida diffusione dei calcolatori in tutto il mondo (i commerciali li vogliono vendere a tutti i costi e in tutti i continenti), ha reso necessario set di caratteri diversi. Dopo vari tentativi di estendere l'ASCII , introducendo le code page (o pagine di codice), alcune aziende del settore hanno fondato un consorzio per creare un nuovo sistema, chiamato UNICODE, e lo ha fatto riconoscere come standard (IS 10646). UNICODE viene ora supportato da vari sistemi operativi (W2000 e XP) e da alcuni linguaggi di programmazione (es: JAVA) e da molte applicazioni.
UNICODE assegna a un simbolo o carattere un valore a 16 bit permanente e unico, chiamato code point . Non vengono usati caratteri a piu' byte ne' sequenze di escape.

Dato che UNICODE usa 16 bit si avranno 65536 code point . Poiche' le lingue mondiali insieme usano circa 200.000 simboli, i code point sono risorse scarse, da assegnare con molta attenzione).
Per facilitare la conversione da ASCII a UNICODE il consorzio UNICODE ha adottato Latin-1 come code point da 0 a 255 .
Ora invece lo standard Unicode, che tendenzialmente e' perfettamente allineato con la norma ISO/IEC 10646, prevede una codifica fino a 21 bit e supporta un repertorio di codici numerici che possono rappresentare circa un milione di caratteri. Ciò appare sufficiente a coprire anche i fabbisogni di codifica di scritti del patrimonio storico dell'umanità, nelle diverse lingue e negli svariati sistemi di segni utilizzati.

RIEPILOGO:


Conversione tra basi: dec → bin
171 1
85 1
42 0
21 1
10 0
5 1
2 0
1 1
Per la parte intera di un numero
1 0 1 0 1 0 1 1
7 6 5 4 3 2 1 0


Conversione tra basi: bin ↔ ott
Binario → Ottale
0 1 1 0 1 0 . 1 0 (26.5)
3 2 . 4 (26.5)
Ottale → Binario
2 5 3 . 2 (171.25)
0 1 0 1 0 1 0 1 1 . 0 1 0 (171.25)

Conversione tra basi: bin ↔ esa

Tabella ASCII – in esadecimale

American Standard Code for Information Interchage

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
20		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
80			‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹	Œ			
90		‘	’	“	”	•	–	—	˜	™	š	›	œ			Ÿ
A0		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B0	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F0	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Per esempio La lettera “A” ha una rappresentazione esadecimale ASCII sommando il valore $(40)_{16}$ e il valore $(1)_{16}$ ovvero $(40)_{16} + (1)_{16} = (41)_{16} \rightarrow (65)_{10}$.

ESERCIZI

1) Codificare su 4 bit il numero -5
secondo le rappresentazioni:
   a) modulo e segno
   b) compl. a 1
   c) compl a 2

2) Dati n = 8 bit, trasformare in esadecimale i seguenti valori

   a) 200 in binario puro
   b) -20 in complemento a 1
   c) -19 in complemento a 2
   d) -80 in modulo e segno

3) Supponendo di avere solo 8 bit a disposizione rappresentare la somma algebrica

AF + F6

in binario e in decimale tenendo presente che l'elaboratore usa la rappresentazione in complemento a due.

4) Trovare la base per la quale e' valida la seguente uguaglianza:

$$(3001)_? = (141)_{12}$$

5) Un computer utilizza 4 byte per memorizzare numeri con la virgola secondo la seguente notazione standard con :

$$X = -1^{S} * 2^{E-M} * 1.F$$

S prende 1 bit per il segno
E esponente con eccesso M (8 bit)
M rappresenta l'eccesso pari a 128
F parte decimale nei restanti bit

(NB: il numero 1 prima della virgola e' sottinteso)

Rappresentare il numero x = -1024,03125

6) Trovare il numero minimo di bit per rappresentare il valore
+2048 in caso di rappresentazione in complemento a 2.

7) Eseguire con le regole del complemento a 1 (su 8 bit) le seguenti somme algebriche:

- 35 - 100
- -100 -11
- 11-11
- 63-64
- 100 +28
- -99 -27

Eseguire con le regole del complemento a 2 (su 8 bit) le stesse somme algebriche sopra elencate:

Calcolare le basi che rendono valide le seguenti uguaglianze:

- $(111)_b = (1013)_5$
- $(101)_b = (52)_{16}$
- $(123)_b = (212)_4$

Trovare la formula che permette di calcolare piu' facilmente i seguenti valori:

1) $(0,11111)_2$
2) $(0,777)_8$
3) $(0,4444)_5$
4) $(0,33333)_4$
5) $(0,555)_6$