L'insieme R di appartenenza dei termini a_k e b_k viene chiamato base del sistema di numerazione ed è costituito da r simboli ordinati tra i quali compaiono lo zero è l'unità.

Ovviamente nel caso del sistema decimale avremo $r=10$, $R=\{0,1,2,3,4,5,6,7,8,9\}$ e

(9) $$x=\sum_{h=0}^{n-1} a_h 10^h + \sum_{k=0}^{m-1} b_k 10^{-k}$$

Poichè gli elaboratori lavorano con dispositivi comunemente a stati binari risulta molto più semplice utilizzare numerazioni in base binaria oppure ottale oppure esadecimale.

Così avremo che in base binaria:

(10) $$x=\sum_{h=0}^{n-1} a_h 2^h + \sum_{k=0}^{m-1} b_k 2^{-k}$$

mentre in base ottale (poco usata):

(11) $$x=\sum_{h=0}^{n-1} a_h 8^h + \sum_{k=0}^{m-1} b_k 8^{-k}$$

e in base esadecimale:

(12) $$x=\sum_{h=0}^{n-1} a_h 16^h + \sum_{k=0}^{m-1} b_k 16^{-k}$$

Si osservi che la base esadecimale è costituita dagli stessi simboli della base decimale integrata da ulteriori 6 simboli presi tra le prime lettere dell'alfabeto. Dunque:

(13) $$R_{hex}=\{0,1,2,3,4,5,6,7,8,9,A,B,C,D,\mathrm{E},F\}$$

Infine, in una generica base costituita da r simboli, avremo:

(14) $$x=\sum_{h=0}^{n-1} a_h r^h + \sum_{k=0}^{m-1} b_k r^{-k}$$

E' opportuno osservare che, per uno stesso numero x, i termini della successione in (8) dipendono strettamente dal sistema di numerazione scelto.

Nella tabella seguente poniamo a confronto la rappresentazione binaria, ottale, esadecimale e decimale dei primi diciassette numeri interi:

Binario	Ottale	Esadecimale	Decimale
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	A	10
1011	13	B	11
1100	14	C	12
1101	15	D	13
1110	16	E	14
1111	17	F	15
10000	20	10	16

Un semplice programma in C++ per convertire numeri da decimale a ottale e esadecimale è il seguente:

```
#include <iostreamh>
int a[] = {45,245,567,1014, –45, –1,256}

void main()
{
int i;
for( i = 0; i <sizeof( a )/sizeof(int); i ++)
        {
        cout<<endl<<endl<<"In decimale: "<<dec<< a[i];
        cout<<endl<<"In ottale: "<<oct<< a[i];
        cout<<endl<<"In esadecimale: "<<hex<< a[i];
        }
}
```

In questo esempio ci sono un paio di costrutti propri del linguaggio C++.
L'istruzione *#include <iostream.h>* contenuta nell'intestazione (header) abilita l'uso della funzione *cout* utilizzata dal C++ per gestire l'output.

Tale funzione è una versione più efficiente dell'analoga istruzione *printf* del linguaggio C. Le parole chiave *dec*, *hex*, and *oct*, il cui valore di default è decimale, impostano il formato dell'output.

L'operatore *sizeof*, utilizzato nell'intestazione del ciclo for per contare da quanti elementi è costituito il vettore $a[]$, restituisce la dimensione in bytes di un dato. In questo caso *sizeof*(a)=28 e *sizeof*(int)=4.

§.10 Rappresentazione dei numeri interi senza segno

La notazione "unsigned" è utilizzata per rappresentare interi non negativi. Tale notazione non è in grado quindi di rappresentare i numeri negativi e i numeri frazionari.

In un sistema che dedica n bit per la notazione "unsigned" un numero a avrà la seguente rappresentazione binaria:

(15) $$a \equiv a_{n-1} \cdots a_0$$

dove, ovviamente, $a_k \in \{0,1\}$ ed assumerà il seguente valore:

(16) $$a = \sum_{k=0}^{n-1} a_k 2^k$$

L'insieme dei numeri rappresentabili con questa notazione e con n bit disponibili è:

(17) $$A = \left\{ a \in \mathbb{N} : 0 \le a \le 2^n - 1 \right\}$$

Osserviamo che in questa notazione lo zero ammette una rappresentazione unica.
In C++ gli interi senza segno possono essere rappresentati dai seguenti tipi (in funzione del numero di bit da dedicare):

unsigned char (8 bit)
unsigned short (16 bit)
unsigned int (generalmente 32 bit)
unsigned long (generalmente 64 bit)

§.11 Rappresentazione dei numeri interi con segno

Esistono due metodi per rappresentare gli interi con segno. Entrambi i metodi non sono in grado di rappresentare numeri frazionari.

Il primo metodo, chiamato notazione "signed", non consente una rappresentazione univoca dello zero e, inoltre, complica eccessivamente l'algoritmo di somma pertanto non viene generalmente utilizzato sugli elaboratori.

La rappresentazione binaria di un numero $a \equiv a_{n-1} \cdots a_0$ costituito da n bit assumerà il seguente valore:

(18) $$(-1)^{a_{n-1}} \cdot \sum_{k=0}^{n-2} a_k 2^k$$

da cui è facile desumere che $a \ge 0$ se $a_{n-1} = 0$ e $a \le 0$ se $a_{n-1} = 1$.

L'insieme dei numeri rappresentabili con questa notazione e con n bit disponibili è:

(19) $$A=\left\{a\in\mathbb{Z}:-(2^{n-1}-1)\le a\le 2^{n-1}-1\right\}$$

L'intervallo è simmetrico rispetto all'origine ma, come già anticipato, lo zero non ha una rappresentazione univoca.

§.12 Complemento a 2

La seconda notazione per la rappresentazione dei numeri interi con segno, detta "complemento a 2" è pressocchè adottata da tutti gli elaboratori.
Il nostro solito numero a binario e costituito da n bit assume, in tale notazione, il valore:

(20) $$a=\sum_{k=0}^{n-2}a_k 2^k - a_{n-1}2^{n-1}$$

Rispetto alla notazione signed, in questo caso un numero è negativo se e solo se $a_{n-1}=1$ quindi, con questa notazione, lo zero è rappresentato in maniera univoca e l'intervallo, non più simmetrico, dei numeri rappresentabili è

(21) $$A=\left\{a\in\mathbb{Z}:-2^{n-1}\le a\le 2^{2-1}-1\right\}$$

Una proprietà fondamentale di questa rappresentazione è la seguente:

(22) $$-a=1+\overline{a}$$

$\overline{a}$ indica il complemento a 1 secondo la semplice regola

(23) $$\overline{a}\equiv\overline{a_{n-1}\cdots a_0}=\overline{a}_{n-1}\cdots\overline{a}_0$$

in cui, su un singolo bit,

(24) $$\overline{x}=\neg x=\begin{cases}1 & x=0\\ 0 & x=1\end{cases}$$

Per verificare la (22) osserviamo che

(25) $$\overline{a}=\sum_{k=0}^{n-2}\overline{a}_k 2^k-\overline{a}_{n-1}2^{n-1}$$

e che

(26) $$\sum_{k=0}^{n-2}a_k 2^k+\sum_{k=0}^{n-2}\overline{a}_k 2^k=\sum_{k=0}^{n-2}(a_k+\overline{a}_k)2^k=\sum_{k=0}^{n-2}2^k=2^{n-1}-1$$

da cui

(27) $$\sum_{k=0}^{n-2}\overline{a}_k 2^k=2^{n-1}-1-\sum_{k=0}^{n-2}a_k 2^k$$

Questa espressione, sostituita nella (25), offre

(28) $$\overline{a}=-\sum_{k=0}^{n-2}a_k 2^k+2^{n-1}-\overline{a}_{n-1}2^{n-1}-1$$

ovvero

(29) $$\overline{a}+1=-\sum_{k=0}^{n-2}a_k 2^k+(1-\overline{a}_{n-1})2^{n-1}$$

e poichè $1-\overline{a}_{n-1}=a_{n-1}$ ricaviamo infine

(30) $$\overline{a}+1=\sum_{k=0}^{n-2}-a_k 2^k+a_{n-1}2^{n-1}=-\left(\sum_{k=0}^{n-2}a_k 2^k-a_{n-1}2^{n-1}\right)=-a$$

La proprietà (22) consente di calcolare rapidamente l'opposto di un numero calcolando il suo complemento a 1 e aggiungendo a questi 1.

Ad esempio il numero $a=1$, la cui rappresentazione binaria a 8 bit è 00000001, ha complemento a 1 pari a 11111110 che, addizionato a 1, offre il valore binario 11111111 corrispondente, appunto a $-a$.

Usando la notazione esadecimale anzichè quella binaria non cambia nulla; infatti in questo caso $a\equiv 01$ e $-a=1+\overline{a}\equiv 01+\overline{01}=01+F\ \mathrm{E}=FF$.

Riportiamo nella seguente tabella un esempio della rappresentazione di alcuni numeri nelle tre notazioni appena viste con 8 bit:

Rappresentazione a 8 bit			
Numero	**Unsigned**	**Signed**	**Complemento a 2**
-128	NR	NR	10000000
-127	NR	11111111	10000001
-2	NR	10000010	11111110
-1	NR	10000001	11111111
0	00000000	00000000 10000000	00000000
1	00000001	00000001	00000001
127	01111111	01111111	01111111
128	10000000	NR	NR

(NR= non rappresentabile)

§.13 Estensione

Dovendo passare da una rappresentazione a n bit (in cui $a=a_{n-1}\cdots a_0$) ad una rappresentazione a $m>n$ bit (in cui $a=b_{m-1}\cdots b_0$), è necessario seguire precise regole di conversione che differiscono a seconda della notazione adottata.

Se si sta facendo uso della notazione unsigned avremo che

(31) $$b_k = \begin{cases} a_k & 0 \le k \le n-1 \\ 0 & n \le k \le m-1 \end{cases}$$

Se si sta facendo uso della notazione signed:

(32) $$b_k = \begin{cases} a_k & 0 \le k \le n-2 \\ 0 & n-1 \le k \le m-2 \\ a_{n-1} & k = m-1 \end{cases}$$

Infine, per la notazione complemento a 2:

(33) $$b_k = \begin{cases} a_k & 0 \le k \le n-2 \\ a_{n-1} & n-1 \le k \le m-1 \end{cases}$$

Nella seguente tabella riportiamo qualche esempio di conversione in notazione complemento a 2

8-Bit	**32-Bit**
0xFF	0xFFFFFFFF
0x0F	0x0000000F
0x01	0x00000001
0x80	0xFFFFFF80
0xB0	0xFFFFFFB0

(il prefisso 0x indica che il numero è espresso in esadecimale)

§.14 Notazione in virgola mobile (IEEE 754 Standard)

La notazione in virgola mobile è l'equivalente binario della notazione scientifica.
Un numero in virgola mobile è costituito da tre componenti:

- il segno s
- la mantissa f
- l'esponente e

Tale notazione consente, facendo uso di linguaggi di programmazione di alto livello, di effettuare calcoli approssimati con i numeri reali.
Nello standard IEEE 754 sono disponibili tre formati: a 32 bit, a 64 bit e a 80 bit.

Lo standard a 32 bit, comunemente chiamato formato a singola precisione, dedica 23 bit per la mantissa, 8 bit per l'esponente polarizzato (biased) e 1 bit per il segno.

La polarizzazione altro non è che una traslazione adottata per evitare esponenti con segno (che consumerebbe un bit per la sua memorizzazione dimezzando così l'intervallo degli esponenti rappresentabili).

Dopo ogni operazione il risultato viene inoltre normalizzato (anche in questo caso per risparmiare bit).
La normalizzazione consiste nell'aggiustamento dell'esponente in modo che la prima cifra a sinistra sia sempre 1 (e così, essendo fissa, non viene memorizzata). Tale cifra viene chiamata bit implicito.

Con queste convenzioni un numero viene pertanto rappresentato nel seguente modo:

(34) $$(-1)^s(1.f)2^{e-127}$$

Lo standard a 64 bit, comunemente chiamato formato a doppia precisione, dedica 52 bit per la mantissa, 11 bit per l'esponente polarizzato (biased) e 1 bit per il segno.

In questo formato un numero verrà rappresentato nel seguente modo:
(35) $$(-1)^s(1.f)2^{e-1023}$$

Poichè lo zero non può essere normalizzato, per convenzione, esso verrà rappresentato da una string di 32 bit (64 bit) tutti posti a zero.

§.15 Rappresentazione dei caratteri da tastiera - standard ASCII

Per rappresentare i caratteri alfanumerici (in cui i numeri non vengono considerati come tali ma semplicemente come simboli al pari delle lettere dell'alfabeto o dei segni di interpunzione) è stato sviluppato uno standard per garantire la compatibilità tra i differenti elaboratori.

Lo standard internazionale ormai adottato da tutti (seppur con qualche differenza) è noto come set di caratteri ASCII (American Standard Code for Information Interchange).

In tale standard ogni carattere viene rappresentato con un byte e dunque consente di rappresentare fino a 256 caratteri distinti dei quali i primi 128 sono specificati direttamente dallo standard.