Appunti sulla rappresentazione dell'informazione

Roberto Beraldi

DISPENSA PER IL CORSO DI

FONDAMENTI DI INFORMATICA

CORSI DI LAUREA IN INGEGNERIA
CHIMICA, DEI MATERIALI,NUCLEARE (vecchi ordinamenti)
Anno Accademico 2001-2002

<Versione 1.0>

1 Introduzione

Per convenienza tecnologica i sistemi di calcolo sono realizzati mediante circuiti elettronici di tipo digitale, in grado cioe' di riconoscere e lavorare su grandezze che possono assumere due soli valori, per convenzione indicati mediante 0 ed 1. Il termine *bit*, contrazione di binary digit - cifra digitale, e' usato per indicare tale tipo di grandezza. Qualunque tipo di informazione (valori numerici, caratteri, programmi, audio, etc.) deve pertanto essere rappresentata in bit prima che essa possa essere elaborata da un sistema di calcolo.

Mediante un solo bit possono chiaramente essere rappresentati due soli oggetti differenti. Per aumentare la capacita' di rappresentazione si utilizzano insiemi ordinati di bit (sequenze di bit), di dimensione N fissa, trattati dall'elaboratore come un unico oggetto.

Indichiamo con E l' insieme composto dagli M elementi che si desidera rappresentare mediante sequenze binarie. Indichiamo con $\mathcal{A} = \{0,1\}$ l'insieme composto dai valori 0 ed 1, e con $\mathcal{A}^N$ l'insieme delle N-plue $\sigma =< b_{N-1}b_{N-2}\dots b_0 >$ con $b_i \in \mathcal{A}$.

Si chiama *codifica* la funzione $f : E \to \mathcal{A}^N$ che associa una sequenza $\sigma =< b_{N-1}b_{N-2}\dots b_0 >$ di N bit ad un elemento $e \in E$. La funzione inversa di f si chiama *decodifica*. Essa consente di risalire, a partire da una sequenza di bit, all'oggetto rappresentato. Esempi di E sono, valori di verita', caratteri, sottoinsiemi dei numeri interi e reali. In particolare nel seguito verranno considerate le seguenti codifiche:

- numeri interi positivi: codifica mediante Sistema di Numerazione Posizionale
- numeri interi con segno: codifica in Modulo e Segno; Complemento a 2
- numeri reali: codifica in virgola mobile
- caratteri: codifica ASCII
- valori di verita'

E' bene anzitutto stabilire il numero di elementi di $\mathcal{A}^N$, indicato con $\mid \mathcal{A}^N \mid$. Esso vale 2^N. Infatti, e' immediato verificare che $\mid \mathcal{A} \mid = 2$ (si hanno le sole sequenze 0 ed 1), e che $\mid \mathcal{A} \mid = 2 \mid \mathcal{A}^{N-1} \mid$ per $N > 1$, poiche' una sequenza composta da N elementi si puo' ricavare dalla concatenazione di una cifra binaria (0 oppure 1) con una sequenza di $N-1$ cifre binarie.
Da cio', $\mid \mathcal{A}^N \mid = 2 \mid \mathcal{A}^{N-1} \mid = 2^2 \mid \mathcal{A}^{N-2} \mid = \dots = 2^k \mid \mathcal{A}^{N-k} \mid = \dots = 2^{N-1} \mid \mathcal{A} \mid = 2^N$.

Alcuni termini usati per indicare sequenze di dimensione ricorrente sono:

- nibble, 4 bit;
- byte, 8 bit;
- word, 2 byte;
- doppia word, 4 byte.

Inoltre il termine MSB (Most Significant Bit) indica il bit piu' a snistra di una sequenza; il termine LSB (Least Significant Bit) quello piu' a destra.

2 Rappresentazione di numeri interi

Per la rappresentazione di valori numerici interi positivi e' possibile sfruttare le proprieta' dei Sistemi di Numerazione Posizionali, richiamati di seguito. In questo caso si segue un'interpretazione ñaturale ëi simboli: 1 rappresenta il valore *uno* , 0 il valore *zero* . Una sequenza di cifre binarie viene interpretata seguendo lo stesso metodo adoperato per interpretare una sequenza di cifre decimali.

Per la rappresentazione dei relativi sono possibili varie alternative, fra le quali: (a) la codifica in modulo e segno, secondo la quale un bit - normalmente MSB - rappresenta il segno del numero, i rimanenti il modulo; (b) la codifica in complemento a 2, basata su un sistema posizionale ibrido e che, per le sue proprieta', consente di ottimizzare la realizzazione di circuiti elettronici.

2.1 Richiami sui sistemi posizionali

Un sistema di numerazione posizionale si fonda sull'uso di un intero $B > 1$ (detto base) ed un insieme $\mathcal{A}$ di simboli (le cifre) avente cardinalita' $\mid \mathcal{A} \mid = B$. In tale sistema, un qualunque valore (intero o frazionario) puo' essere rappresentato mediante una sequenza di cifre, alle quali e' associato un peso B^i con i posizione della cifra nella sequenza.

Si adopera poi un simbolo (+/-) per specificare il segno del numero ed un ulteriore simbolo (punto o virgola) per indicare l'inizio delle posizioni negative, ossia con pesi B^{-i}. Per marcare la differenza con il concetto di numero, una sequenza di cifre si chiama numerale. Uno stesso numero puo' allora avere piu' numerali che lo esprimono in basi differenti. Se chiaro dal contesto si usa tuttavia il termine numero sia per denotare il valore, sia la sequenza di cifre che lo esprime.
Nell'ambito dei calcolatori le basi ricorrenti sono:

- $B = 2$ (sistema binario, $\mathcal{A}$ = {0 ed 1) })
- $B = 8$ (sistema ottale, $\mathcal{A}$ = {0,1,2,3,4,5,6,7})
- $B = 16$ (sistema esadecimale, $\mathcal{A}$= { 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F})

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

Tabella 1: *Corrispondenza fra valori nelle varie basi*

In generale, il numero v denotato dalla sequenza di cifre

$d_{N-1}d_{N-1}\ldots d_0.d_{-1}\ldots d_{-M}$

di una base B, vale:

$v = I + F$, con $I > 1$ parte intera e $F < 1$ parte frazionaria

$I = \sum_{i=0}^{N-1} d_i B^i$;

$F = \sum_{i=-1}^{-M} d_i B^i = \sum_{i=1}^{M} d_{-i}/B^i$

ossia

$v = \sum_{i=-M}^{N-1} d_i B^i$

Per qualunque base B valgono gli algoritmi dell'aritmetica (somma, prodotto, etc.), gia' ben noti per il sistema di numerazione decimale ($B = 10$). Quando serve specificare la base del sistema, si usa la notazione $(\ldots)_B$ (con B espresso in base 10!). Nel caso $B = 16$ e' prassi anteporre al numero il prefisso 0x o aggiungere la lettera H al termine del numero.

Esempio. Prodotto per la base.
Consideriamo il sistema di numerazione binario. Il prodotto per 2 si ottiene spostando a sinistra le cifre binarie, aggiungendo eventualmente uno 0
$1_{10} = 1_2$
2_{10} x $1_{10} = 2_{10} = 10_2$
2_{10} x $2_{10} = 4_{10} = 100_2$
2_{10} x $4_{10} = 8_{10} = 100_2$
In generale, la stringa di $N+1$ cifre binarie avente il primo bit pari ad 1 seguito da N bit a 0, denota il valore 2^N.
□
Alcuni valori di N sono usati per i multipli:

- $2^{10} = K$ (Kilo);
- $2^{20} = M$ (Mega);
- $2^{30} = G$ (Giga);
- $2^{40} = T$ (Tera);
- $2^{50} = P$ (Peta)

Si presti attenzione al fatto che 1K = 1024 e non a 1000; come del resto per gli altri simboli di multiplo.

Esempio.Massimo valore rappresentabile con N bit.
Nel sistema binario valgono le stesse regole aritmetiche note per $B = 10$. In particolare per la somma di due numeri: 0+0=0; 0+1=1+0=1; 1+1=10; pertanto, il numero $2^N - 1$ e' rappresentato dalla sequenza di N bit pari ad 1, che e' anche il massimo valore che puo' essere rappresentano con N bit mediante la codifica posizionale pura. Ad esempio: $1000_2 = 111_2 + 1_2$, da cui $1000_2 - 1_2 = 111_2 = 2^3 - 1 = 7$
□

2.2 Codifica mediante Sistema di Numerazione Posizionale

Per la rappresentazione di valori numerici interi positivi mediante cifre binarie e' possibile usare direttamente il sistema di numerazione posizionale. La codifica in questo consiste nel ricavare la sequenza di cifre binarie a partite dalla sequenza di cifre decimali con la quale si denota il valore da codificare. Le seguenti sezioni affrontano il problema generale della conversione fra basi.

2.2.1 Conversione fra basi

Consideriamo il problema generale di derivare la stringa di cifre $(S_1)_{B_1}$, con la quale indicare il valore v in un sistema con base B_1, a partire dalla stringa $(S_2)_{B_2}$, che indica lo stesso valore v in un sistema con base B_2 diversa da B_1 (problema della conversione fra basi). Sono possibili due alternative, a seconda che si vogliono effettuare le operazioni di conversione nella base di arrivo o in quella di partenza.

2.2.2 Uso della base di partenza

Consideriamo prima la conversione di numeri interi. Si usa la seguente proprieta': per qualunque numero intero positivo a, dato un intero $b > 1$ esistono due valori interi ed unici q ($q < a$) ed r ($0 \leq r < b$) tali che:

$a = qb + r$

$q = a$ div b e' detto *quoziente* della divisione intera fra a e b, $r = a$ mod b *resto*.

Si parte dal polinomio:

$v = d_{N-1}B^{N-1} + d_{N-2}B^{N-2} + \ldots + d_0B^0$

e si mette B in evidenza:

$v = B(d_{N-1}B^{N-2} + d_{N-2}B^{N-3} + \ldots + d_1) + d_0$, da cui $a = Bq_1 + r_0$, dove:

$r_0 = d_0 = v$ mod B;
$q_1 = d_{n-1}B^{n-2} + d_{n-2}B^{n-3} + \ldots + d_1 = v$ div B;

Lo stesso metodo si applica a q_1 per isolare d_1 e cosi via. Il procedimento termina quando si ottiene il quoziente nullo, condizione sicuramente raggiunta poiche' a div $b < a$. Se q_i indica l'i-simo quoziente ($i \geq 0$, ponendo $q_0 = v$) la cifra i-sima vale pertanto: $r_i = d_i = q_i$ mod B. Continuando eventualmente il procedimento, si otterrebbero sempre cifre pari a 0. Il metodo descritto si chiama metodo delle divisioni successive.

Esempio.
Applichiamo l'algoritmo per $n = 15$:
b_0 = 15 mod 2 = 1; q_1 = 15 div 2 = 7;
b_1 = 7 mod 2 = 1; q_0 = 7 div 2 = 3;
b_2 = 3 mod 2 = 1; q_2 = 3 div 2 = 1;
b_3 = 1 mod 2 = 1; q_3 = 1 div 2 = 0.
□
Poiche' n mod 2 = 1 se e solo se n e' dispari, per la conversione e' sufficiente procedere con le divisioni intere fino ad ottenere il quoziente nullo;si pone poi il bit $b_i = 1$ se il quoziente q_i e' dispari, altrimenti $b_i = 0$.

Esempio.
Convertire 13 in base 2.
$13 \to 6 \to 3 \to 1 \to 0$; 1101 ($q_0 = 13, q_1 = 6, ...$)
□

Si consideri adesso il problema della conversione di un numero $0 < F < 1$. Si vogliono trovare le cifre d_{-i} che esprimono il numero nella base di arrivo B, ma si desidera eseguire i calcoli nella base di partenza. Scriviamo il valore F come:

$F = d_{-1}B^{-1} + d_{-2}B^{-2} + \ldots + d_{-m}B^{-m} + \ldots$

Si noti che non si conosce a priori il numero m delle cifre. Infatti il numero da convertire puo' essere periodico nella base di arrivo B, ma non in quella di partenza. Eseguiamo ora il prodotto $p = fB$:

$p = fB = d_{-1} + d_{-2}B^{-1} + \ldots + d_{-m}B^{-m-1} + \ldots$

La cifra d_{-1} e' l'unica con peso intero (pari ad 1) e quindi rappresenta la parte intera di FB, ossia - indicando con trunc(x) la parte intera di x:

d_{-1} = trunc(FB)

Il metodo puo' essere adesso applicato al valore frazionario FB - trunc(FB) per isolare la cifra d_{-2}. Si procede poi in modo analogo. Il procedimento termina quando si ottiene un prodotto nullo o la precisione desiderata. Il metodo si chiama metodo delle moltiplicazioni successive

Esempio.
Si desidera convertire 0.8125 in base 2:
0.8125 x 2 = 1.625; d_{-1} = 1;
0.625 x 2 = 1.25; d_{-2} = 1;
0.25 x 2 = 0.50; d_{-3} = 0;
0.50 x 2 = 1.0; d_{-4} = 1;
$0.8125 = 0.1101_2$
□

Esempio.
Si desidera convertire 0.3 in base 2:
0.3 x 2 = 0.6; d_{-1} = 0;
0.6 x 2 = 1.2; d_{-2} = 1;
0.2 x 2 = 0.4; d_{-3} = 0;
0.4 x 2 = 0.8; d_{-4} = 0;
0.8 x 2 = 1.6; d_{-5} = 1;
0.6 x 2 = 1.2; d_{-6} = 1;
...
$0.3 = 0.0\overline{(1001)}_2$ Il numero 0.3 e' periodico in base 2.
□

Esempio.
Esprimere il valore $x = 1/3 = 0.3333\ldots$ in base 3:
E' immediato vedere che $x = 0.1_3$, e cioe' esso non e' periodico in base 3, mentre lo e' in base 10.
□

2.2.3 Uso della base di arrivo

E' sufficiente esprimere il polinomio con cui si calcola il valore v direttamente nella base di arrivo.

Esempio.
Convertire 0xFF in base 10.
Ricordiamo che $F_{16} = 15_{10}$; pertanto: $v = 15_{10}16^1_{10} + 15_{10}16^0_{10} = 240_{10} + 15_{10} = 255_{10}$
□
Esempio.
Convertire 128 da base 10 in base 16.
Si ha: $v = 1_{16}A^2_{16} + 2_{16}A^1_{16} + 8_{16}A^0_{16}$ =0x64 + 0x14 + 0x8 = 0x80.
□

Benche' il metodo sia lo stesso, il primo esempio sembra piu' naturale rispetto al secondo, poiche' le operazioni sono condotte in base 10, con la quale si ha familiarita'. Questo metodo e' infatti preferito quando la base di arrivo e' 10. L'algoritmo e' identico nel caso di conversione di valori frazionari.

Esempio.
Esprimere il valore rappresentato da $.11_2$ in base 10.
Risp. Per comodita' ammettiamo il pedice 10, $v = 2^{-1} + 2^{-2} = 0.5 + 0.25 = 0.75$
□

2.2.4 Relazione fra rappresentazioni nelle basi B e B^k Esempio

Si vuole rappresentare in base 2 un valore v espresso in base 16. Siano $e_{N-1}e_{N-2}\dots e_0$ le cifre relative al valore espresso in base 16 (e_i cifra esadecimale), e $b_{M-1}b_{M-2}\dots b_0$ quelle che esprimono v in base 2 (b_i cifra binaria).
Lavorando in base 10:

$v = e_{N-1}16^{N-1} + e_{N-2}16^{N-2} + \ldots + e_0 16^0$

Esprimiamo i coefficienti e_i in base 2. Sono necessari 4 bit (e quindi $M = 4N$):

$e_0 = 2^3 b_3 + 2^2 b_2 + 2b_1 + b_0$
$e1 = 2^3 b_7 + 2^2 b_6 + 2b_5 + b_4$
...
$e_i = 2^3 b_{4i+3} + 2^2 b_{4i+2} + 2b_{4i+1} + b_{4i} = \sum_{j=0}^{3} 2^j b_{4i+j}, (i = 0..N-1)$

da cui, sfruttando $16 = 2^4$:

$v = \sum_{i=0}^{N-1} 16^i e_i = \sum_{i=0}^{N-1} 2^{4i} e_i = \sum_{i=0}^{N-1} 2^{4i} (\sum_{j=0}^{3} 2^j b_{4i+j}) = \sum_{i=0}^{N-1} \sum_{j=0}^{3} 2^{4i+j} b_{4i+j}$

Ponendo $k = 4i + j$ si ottiene:

$v = \sum_{k=0}^{M-1} 2^k b_k$

La rappresentazione di un valore v in base 2, partendo dalla rappresentazione di v in base 16, si ottiene sostituendo ad ogni cifra esadecimale la corrispondente rappresentazione in base 2, facendo attenzione ad usare sempre 4 bit. Viceversa, per trovare la rappresentazione di un numero in base 16, partendo dalla sua rappresentazione in base 2, e' sufficiente raggruppare le cifre binarie in gruppi di 4 cifre e sostituire ogni gruppo con la cifra esadecimale che esso codifica. Il metodo vale per qualunque coppia di basi B_1,B_2, tali che $B_1 = B_2^k$ (k intero).

Esempio.
Esprimere il valore 1001101_2 in base 16.
Risp. Aggiungendo 0 a sinistra, in modo da formare due gruppi da 4 cifre: 0100 1101, da cui 0x4D.
□
Esempio.
Si vuole convertire 0xA0 in base 2. Si osservi che A rappresenta il valore, espresso in decimale 10=8+2, che equivale al binario 1010, mentre ovviamente 0 e' rappresentato da 0000. Quindi:
0xA0 = 10100000_2
□
Esempio.
Si vuole convertire 0xFF in base 2. Sappiamo che 0xF=1111, pertanto
0xFF= $11111111_2 = 255_{10}$
□
Le basi 8 e 16 possono essere usate come strumento per scrivere in forma breve lunghe sequenze di bit.

Esempio.
10111000111011101111_2= 0xB8EEF
□

2.3 Rappresentazione in complemento alla base

Fissato il numero N di cifre, si definisce complemento in base 2, o semplicemente complemento a 2, di un numero intero x, il valore $C_{2^N}(x)$, tale che:

$$C_{2^N}(x) = \begin{cases} x & 0 \leq x < 2^{N-1} \\ 2^N - |x| & -2^{N-1} \leq x < 0 \end{cases}$$

I valori che possono essere espressi in complemento a 2 mediante N bit sono quindi compresi nell'intervallo $[-2^{N-1} \ldots 2^{N-1} - 1]$. Il complemento alla base di numero positivo coincide con il numero stesso, mentre il complemento di un numero negativo e' un valore positivo nell'intervallo chiuso $[2^{N-1} \ldots 2^N - 1]$.

Per ricavare x noto il suo complemento, si esegue la differenza $C_{2^N}(x) - 2^N$, e questo ovviamente solo nel caso in cui $2^{N-1} \leq C_{2^N}(x) \leq 2^N - 1$:

$$x = \begin{cases} C_{2^N}(x) & 0 \leq C_{2^N}(x) < 2^{N-1} \\ C_{2^N}(x) - 2^N & 2^{N-1} \leq C_{2^N}(x) \leq 2^N - 1 \end{cases}$$

che puo' essere riscritta, conoscendo la stringa $b_{N-1}b_{N-1} \ldots b_0$ che rappresenta il valore in complemento, come:

$x = -b_{N-1}2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i$

Per verificare tale espressione notiamo infatti che, se v e' un valore negativo, il suo complemento e' maggiore o uguale a 2^{N-1} e pertanto $b_{N-1} = 1$; per cui si puo' scrivere:

$x = C_{2^N}(x) - 2^N = (2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i) - 2^N = -2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i = -b_{N-1}2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i$

D'altra parte, se x e' positivo esso deve essere minore di 2^{N-1} e quindi $b_{N-1} = 0$. Inoltre il complemento di x coincide con il valore x stesso, denotato da $b_{N-2}b_{N-2} \ldots b_0$.

Il bit piu' significativo di un numero che rappresenti il complemento in base 2 di x consente di capire se si tratta di un numero positivo, in qual caso esso vale 0, o negativo se esso vale 1. Il bit tuttavia non rappresenta il segno; infatti per trovare il valore negativo di un numero a partire dal suo valore positivo non e' sufficiente invertire solamente il bit in posizione piu' significativa, come accade invece nella rappresentazione in modulo e segno.

Esempio.
Calcolare la rappresentazione di $x = -3$ in complemento a 2 usando 4 bit. Esprimere il complemento in base 10.
Risp. Indichiamo con x' il complemento di x; $x' = 2^4 - 3 = 16 - 3 = 13$. □

Per calcolare il complemento non e' necessario effettuare operazioni di sottrazione. Infatti, lavorando in base 2 e sfruttando il fatto che 0+1=1+0=1 si ha: $\overline{b} + b = 1$, dove $\overline{b}$ indica l'operazione di inversione del valore del bit b: $\overline{0} = 1$ e $\overline{1} = 0$. Indicando con $\overline{a}$ il numero binario ottenuto da a invertendo tutti i bit (chiamato complemento ad 1 di a) si ottiene $a + \overline{a} = 11 \ldots 1 = 2^N - 1$, ossia $\overline{a} = 2^N - 1 - a$; da cui, per $x < 0$:

$C_{2^N}(x) = 2^N - |x| = 2^N - 1 + 1 - |x| = \overline{|x|} + 1$

Per calcolare il complemento di un numero intero negativo x si puo' quindi:

- Calcolare il complemento ad 1 del modulo di x;
- Sommare 1.

Esempio.
Calcolare la rappresentazione di $x = -3$ in complemento a 2 usando 4 bit.

$$\begin{array}{lcll} |x| & = & 0011_2 & \\ & & & \\ \overline{|x|} & = & 1100_2 & + \\ & & 0001_2 & = \\ x' & = & \overline{1101_2} & \end{array}$$

2.3.1 Proprieta' della rappresentazione in complemento alla base

Il motivo per il quale si preferisce usare la rappresentazione in complemento a 2 e' dovuto principalmente al fatto che le operazioni di sottrazione si trasformano, in questa notazione, in somme. Nell'eseguire la somma un eventuale riporto in posizione N deve essere ignorato, senza che cio' generi alcun errore (chiaramente solo nei casi in cui il risultato appartiene all'intervallo di valori rappresentabili con N bit secondo questo metodo). Verifichiamo tale proprieta' con due esempi, nei quali sono indicati con x ed y due valori positivi, scelti in modo tale che possano essere espressi mediante il rispettivo complemeto, $0 \leq x < 2^{N-1}$, $0 \leq y \leq 2^{N-1}$. Per comodita' indichiamo i valori complemento con x' e y'. Sia $d = x - y$. Notiamo anzitutto che $-2^{N-1} \leq d \leq 2^{N-1} - 1$, e quindi la differenza puo' essere espressa mediante complemento.

Esempio. Calcolo di $d = x - y$, nel caso in cui $x - y > 0$
In questo caso, $d' = d$. Sostituiamo x ed y con i rispettivi complementi. Per calcolare il complemento della differenza, d', si esegue la somma dei complementi di x e $-y$: $s = x + (2^N - y) = x - y + 2^N$; la somma produce riporto in posizione N poiche' per ipotesi $x - y > 0$ e quindi $s > 2^N$. Il riporto deve essere ignorato, e cio' equivale a sottrarre 2^N da s, si ha quindi: $d' = s - 2^N = x - y$
Come esempio numerico, consideriamo $N = 3, x = 3, y = -2$;
$x' = 011_2$
$y' = 2^3 - 2 = 8 - 2 = 6 = 110_2$.
Da cui: $x' + y' = 011_2 + 110_2 = (1)001_2; d' = d = 001_2 = 1$.
□

Esempio. Calcolo di $d = x - y$, con $x - y < 0$
In questo caso, $d' = 2^N - (|x - y|) = 2^N - (y - x)$. Sostituiamo i valori x e y con i rispettivi complementi ed eseguiamone la somma:
$s = x + (2^N - y) = 2^N - (y - x)$;
per ipotesi $y - x > 0$ e quindi s coincide con la definizione del complemento di $x - y$: $d' = s = 2^N - (y - x)$
Come esempio numerico, consideriamo $N = 3, x = 2, y = -3$. Si ha:
$x' = 010_2$
$y' = 2^3 - 3 = 8 - 3 = 5 = 101_2$.
Da cui: $d' = x' + y' = 010_2 + 101_2 = 111_2; d = -2^2 + 3 = -4 + 3 = -1$
□

Esempio. Verifica della proprieta' $a - a = 0$
Il complemento di a e' pari ad $a' = 2^N - a$, e quindi $a - a = a + a' = a - a + 2^N = 2^N$. Ignorando il bit di riporto ($b_N = 1$) si ottiene 0. Ad esempio, con $N = 4$, e $a = 3$ si ha, lavorando in base 2, $a = 0011$; $a' = 1101$ e quindi $a + a'$=(1)0000.
□

Nei casi in cui x ed y hanno lo stesso segno, si puo' dimostrare che la somma dei complementi indica il complemento del risultato nei casi in cui il bit in posizione piu' significativa coincida con quello degli addendi. Cio' avviene sempre nel caso in cui il risultato appartiene all'intervallo di valori rappresentabili con N bit. Negli altri casi si verifica un errore di *overflow*, ossia di superamento dei limitidi rappresentazione del metodo. Questo accade se $x + y > 2^{N-1} - 1$ o se $-x - y < -2^{N-1}$

Esempio.
Calcolare il valore $x' + y'$ con x'=0xAF ed y'=0x7A, interpretando i numeri esadecimali come rappresentazioni in complemento a 2 su 8 bit. Si esprima il risultato r in base 10
Risp. Come primo metodo, scegliamo di eseguire le operazioni in base 10. 0xAF = 10 x 16 + 15 = 175; 0x7A = 7 x 16 + 10 = 122. Poiche' $N = 8$, valori complemento minori di $2^7 = 128$ denotano numeri positivi, gli altri valori negativi; pertanto $x = x' - 2^8 = 175 - 256 = -81$ e $y = y' = 122$, da cui $r = 41$.
Secondo metodo. Usiamo la base 2. 0xAF = 10101111_2 e 0x7A = 01111010_2. Da cui:

	111111100_2	$\leftarrow$ riporti
$(175)_{10}$	10101111_2	+
$(122)_{10}$	01111010_2	=
$(297)_{10}$	100101001_2	

Il bit in posizione 8 viene trascurato. Cio' equivale ad effettuare la sottrazione 297-256=41=00101001_2. □

Esempio (domanda appello del 11 gennaio 2002).
Assumendo di rappresentare interi relativi con 8 bit, quali fra le seguenti somme provocano overflow? (I pedici indicano la base di numerazione e i valori riportati sono il risultato della somma $2^8 + N$, in accordo con la definizione della rappresentazione in complemento a due.)

a. $(01001101)_2 + (11101101)_2$
b. $(162)_8 + (145)_8$
c. $(5E)_{16}$ + $(7A)_{16}$
d. $(FA)_{16}$ + $(AF)_{16}$
Risp. Osserviamo anzitutto la definizione di complemento. Essa assume implicitamente che il bit in posizione 8 vada ignorato. Pertanto, per i valori positivi, la somma $2^8 + N$ coincide con N (infatti se $N = b_7b_6b_5b_4b_3b_2b_1b_0$, allora $N + 2^8 = 1b_7b_6b_5b_4b_3b_2b_1b_0$). Nel seguito per semplicita', il pedice 10 e' omesso.
a. Gli addendi hanno segno discorde e quindi non puo' esserci overflow.

b. Convertiamo i valori in binario: $(162)_8 = (001110010)_2$; $(145)_8 = (001100101)_2$. Eseguiamo la somma dei numeri binari:

11000000	$\leftarrow$ riporti
01110010	+
01100101	=
11010111	

Si riconosce la condizione di overflow poiche' la somma ha MSB=1, mentre per gli addendi MSB=0.
Come metodo alternativo operiamo in base 10. Poiche' $b_7 = 0$ i valori sono positivi e valgono rispettivamente: $8^2 + 6(8^1) + 2 = 114$ e $8^2 + 4(8^1) + 5 = 101$; la somma vale $114 + 101 = 215$. Si e' pertanto nella condizione di overflow poiche' con 8 bit valori compresi fra $2^7 = 128$ e $2^8 - 1 = 255$ rappresentano valori negativi.

c. Convertiamo i valori in base 10.$(5E)_{16} = 5(16) + 14 = 94_{10}$ e $(7A)_{16} = 7(16) + 10 = 122_{10}$. La somma vale $94 + 122 = 216$ e quindi si ha un errore di overflow, per lo stesso motivo del caso precedente.

d. Operiamo in base 10. $(FA)_{16} = 15(16) + 10 = 250$ (complemento di un valore negativo) e $(AF)_{16} = 10\,16 + 15 = 175$ (complemento di un valore negativo), con somma 425. Poiche' e' verificata la condizione $425 \geq 256$ si deve sottrarre 256, e quindi si ottiene 425-256=169 che esprime il complemento di un valore negativo (poiche' compreso nell'intervallo [128..255]) e quindi non si ha overflow.
Come metodo alternativo, determiniamo i valori di partenza degli addendi. $(FA)_{16} = 250$ e' il complemento di $250 - 256 = -6$ e $(AF)_{16} = 175$ il complemento di $175 - 256 = -81$. La somma vale: $-6 - 81 = -87$

che puo' essere espresso in complemento mediante 8 bit. Si noti che il complemento di -87 vale ovviamente 256-87=169.

2.4 Rappresentazione in Modulo e Segno

Questo tipo di rappresentazione e' molto semplice. Un bit della stringa, di prassi quello piu' significativo, esprime il segno, mentre i rimanenti il modulo. Indichiamo con $s = b_{N-1}$ il bit di segno ($s = 1$ per numeri negativi). Il valore denotato dalla stringa $b_{N-1}b_{N-2}\dots b_0$ e' pertanto:

$v = (-1)^s \sum_{i=0}^{N-2} b_i 2^i$

Il range di valori rappresentabili e' $[-(2^{N-1}-1)...2^{N-1}-1]$

Esempio. Indicare in base 10 il valore x espresso in modulo e segno da 10000101_2.
Risp. Il bit di segno vale 1; il modulo vale $0000101_2 = 5$. Da cui $x = -5$.
□

La rappresentazione in modulo e segno presenta diversi svantaggi. Ad esempio lo 0 ha due rappresentazioni, -0 e +0 (00_2 e 10_2); inoltre non vale la proprieta' $a + (-a)$=0.

2.5 Riassunto dei metodi di codifica

La tabella seguente riporta una sintesi dei metodi di codifica dei numeri interi e dei relativi range dei valori che possono essere rappresentati.

CODIFICA	RANGE DI VALORI
Posizionale	$[0...2^N-1]$
Modulo e segno	$[-(2^{N-1}-1)...2^{N-1}-1]$
Complemento a 2	$[-2^{N-1}...2^{N-1}-1]$

Come esempio di applicazione la seguente tabella riporta i valori espressi in base 10 ottenuti interpretando una sequenza compsta da tre cifre binarie seguendo i metodi illustrati.

Sequenza	Posizionale	Modulo e Segno	Complemento a 2
000	0	0	0
001	1	1	1
010	2	2	2
011	3	3	3
100	4	-0	-4
101	5	-1	-3
110	6	-2	-2
111	7	-3	-1

3 Rappresentazione di numeri reali

3.1 Virgola mobile

La rappresentazione in virgola mobile deriva dalla notazione scientifica. Un numero reale puo' sempre essere espresso nella forma scientifica $m10^{exp}$, dove m e' un reale (la mantissa), ed exp un intero (l'esponente). Ad esempio, $(12.3)\ 10^{-5}$ e' una forma scientifica. Fra le diverse coppie di valori (m, exp) con cui rappresentare lo stesso numero se ne individua una, detta rappresentazione normalizzata, imponendo $0.1 \leq m < 1$.

Ad esempio, la forma normalizzata del valore precedente e' $(0.123)\ 10^{-3}$.

Tale forma puo' essere applicata in una qualunque base *B*, con rappresentazione normalizzata stabilita dalla condizione $1/B \leq m < 1$. In particolare, per $B = 2$ si ha: $2^{-1} \leq m < 1$. Dal momento che $2^{-1} = 0.1_2$, tale condizione implica che la prima cifra binaria a destra del punto sia sempre pari ad 1, ossia la mantissa assume la forma $m = (0.1\text{xxxx...})_2$ = $(1.\text{xxxx...})_2(0.1)_2$, dove x indica una generica cifra binaria. In base 2 si preferisce definire mantissa il valore *m*= $(1.\text{xxxxx..})_2$ ($1 \leq m < 2$), ossia come somma di 1 con un valore frazionario f (il numero 0.xxxx...): $m = 1 + 0.f$, indicato brevemente $(1.f)$, ed inlcudere $(0.1)_2 = 2^{-1}$ nella parte esponente

Nella rappresentazione in virgola mobile il segno viene rappresentato mediante un bit *s* (*s*=0 per il segno positivo, *s*=1 per segno negativo). Inoltre, per evitare di rappresentare il segno dell'esponente e' possibile seguire una codifica polarizzata. Al posto di *exp* si codifica la somma $e = exp + c$, dove *c* (detto *eccesso*), assume un valore che garantisce $e \geq 0$.

Un valore reale *v*, in forma polarizzata con costante *c*, si puo' quindi esprimere come tripla (s, f, exp), che denota il valore:

$v = (-1)^s(1.f)2^{exp-c}$.

Esempio.
Rappresentare il valore *x*=209.8125 in base 2 in forma normalizzata senza polarizzazione.
Risp. Rappresentiamo *x* in forma binaria, convertendo separatamente la parte intera e quella frazionaria:
$209 \to 104 \to 52 \to 26 \to 13 \to 6 \to 3 \to 1 \to 0$; ossia 11010001;

$0.8125 = 0.1101_2$. Si ottiene $x = 11010001.1101_2$; normalizzato, x=1.10100011101_2 x 2^7. Si ricordi che la divisione per 2 di un numero *x* espresso in base 2, corrisponde ad effettuare uno spostamento a sinistra (shift) delle cifre di *x*. Inoltre, si e' usata la base 2 per esprimere il valore della mantissa *m*, e la base 10 per esprimere la parte esponente). Da cui, $f = 10100011101_2$, $exp = 7 = 111_2$
□

Esempio (domanda appello del 15 febbraio 2002) . Immaginando di rappresentare i numeri razionali in floating point, con locazioni da 32 bit, di cui 24 per la mantissa, fornire la rappresentazione dei seguenti numeri:

1. 64.0625
2. -254
3. 0.078125

1. Convertiamo separatamente le parti intera e decimale.
   $64 = 2^6 = 1000000_2$;
   0.0625 x 2 = 0.125; d_{-1} = 0;
   0.125 x 2 = 0.25; d_{-2} = 0;
   0.25 x 2 = 0.50; d_{-3} = 0;
   0.50 x 2 = 1.0; d_{-4} = 1;
   $0.0625 = 2^{-4} = 0.0001_2$
   quindi, $64.0625 = 1000000.0001_2$ che, normalizzando, diventa:
   $(1.0000000001)_2 2^6$.

   Il primo bit della mantissa puo' non essere memorizzato (bit nascosto o implicito), pertanto sono necessari 23 bit. Dei rimanenti 32-23=9 bit, un bit si sfrutta per il segno della mantissa (bit di segno *s*) mentre i rimanenti per l'esponente *e*, che scegliamo di rappresentare con polarizzazione $c = 127$. Pertanto, $m = 1.00000000010000000000000_2$; $e = 6 + 127 = 133 = 128 + 4 + 1 = 10000101_2$; $s = 0$.

2. $254 \to 127 \to 63 \to 31 \to 15 \to 7 \to 3 \to 1$, da cui: $254 = 11111110$. In virgola mobile, $254 = (1.1111110_2)2^7$, da cui: $m = 1.11111100000000000000000$; $e = 7 + 127 = 134 = 128 + 4 + 2 = 10001010_2$; $s = 1$

3. $x = 0.078125 = 2^{-7} = 0.0000001_2$. Da cui: $x = 1.00\ldots0_2\ 2^{-7}$; $m = 1.0...$, $e = 127 - 7 = 120 = 64 + 32 + 4 = 01100100_2$, $s = 0$. □

3.1.1 Precisione

La precisione della rappresentazione in virgola mobile e' stabilita dal numero di cifre della mantissa. Infatti, l'insieme $\mathcal{F}$ dei valori che possono essere rappresentati mediante un sistema a virgola mobile e' costituito dai soli valori che possono essere messi nella forma $(1.f)2^E$. Sia p il numero di cifre binarie con cui poter esprimere la parte frazionaria f della mantissa. Consideriamo due valori successivi di $\mathcal{F}$: $v_1 = (1.f)2^E$ e $v_2 = (1.f + 2^{-p})2^E$. Qualunque valore $v_1 < x < v_2$ puo' essere rappresentato solo in modo approssimato mediante v_1 o v_2. Indichiamo con $fl(x)$ il valore che approssima x.

Dato un valore $x \neq 0$ si definiscono gli errori

- assoluto: $E = fl(x) - x$, $\mid E \mid \leq 2^{-p}2^E$;
- relativo: $\varepsilon = \frac{fl(x)-x}{x}$, $\mid \varepsilon \mid \leq 2^{-p}$.

Tali errori quantificano la bonta' dell'approssimazione. Dato il valore approssimato di x, $fl(x)$, si ha: $fl(x) = x(1+\varepsilon)$.

Esempio

Supponiamo di usare 7 bit per la mantissa. Qual e' la rappresentazione binaria in virgola mobile di $x = 0.6$?
Risp. Convertiamo il valore in binario:
0.6 x 2 = 1.2; d_{-1} = 1;
0.2 x 2 = 0.4; d_{-2} = 0;
0.4 x 2 = 0.8; d_{-3} = 0;
0.8 x 2 = 1.6; d_{-4} = 1;
0.6 x 2 = 1.2; d_{-5} = 1;
...

Il valore 0.6 e' periodico in base 2: $x = 0.10011001\ldots$; troncando le cifre binarie, la forma normalizzata diventa, $fl(x) = 1.001100_2 2^{-1}$. Con errore $E \leq 2^{-7} = 0.0078125$. Infatti $fl(x) = 2^{-1} + 2^{-4} + 2^{-5} = 0.5 + 0.0625 + 0.03125 = 0.59375$ ed $E = x - fl(x) = 0.0625$.
□
Si ovvervi che la distanza fra valori consecutivi di $\mathcal{F}$ vale $(1.f)2^E - (1.f + 2^{-p})2^E = 2^{-p}2^E$. I valori di $\mathcal{F}$ non sono distanziati uniformemente. Per un fissato valore di E la distanza vale $2^{-p}2^E$, ma raddoppia passando da E ad $E + 1$ (Fig. 1).

Esempio

Stabilire il numero di cifre binarie richieste per garantire una rappresentazione con errore relativo $\varepsilon < 10^{-6}$
Risp. E' necessario esprimere l'errore come potenza di 2: $10^{-6} = 2^{-p}$; da cui $p = 6/Log2 \approx 6(3.22) \approx 19.93$ e, approssimando all'intero superiore, $p = 20$. Quindi con 20 bit per la parte frazionaria, l'errore che si commette e' inferiore a 0.000001%.□

Nei casi i cui $\mid x \mid$ e' maggiore del massimo valore di $\mathcal{F}$ (pari a $(1 + (1 - 2^{-p}))2^{E_{max}}$) si incorre in un errore di *overflow*. In modo simile, se x e' minore del piu' piccolo valore diverso da 0 di $\mathcal{F}$ si determina una condizione di *underflow*.

E' il caso di accennare allo standard IEEE 754 per la rappresentazione dei numeri in virgola mobile, che prevede quattro gradi di precisione ($\mid m \mid$ indica il numero di bit della mantissa, incluso il bit nascosto).

- singola ($\mid m \mid$ = 24 bit; $E_{max} = 127; E_{min} = -126$);

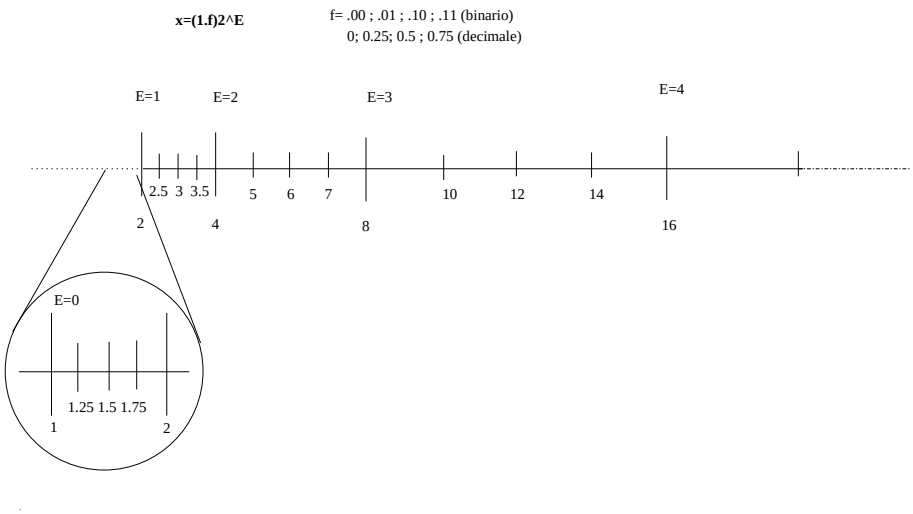

Figura 1: Distribuzione dei valori con $p = 2$

- singola estesa ($|\ m\ |$ = $\geq$ 32 bit; $E_{max} \geq 1023; E_{max} \leq -1022$);
- doppia ($|\ m\ |$ = 53 bit; $E_{max} = 1023; E_{min} = -1022$);
- doppia estesa ($|\ m\ |$ = $\geq$ 64 bit; $E_{max} \geq 16383; E_{min} \leq -16382$);

I problemi derivanti dalla limitazione sul numero delle cifre ed i conseguenti errori prodotti nei calcoli, sono trattati in dettaglio nei corsi di Analisi Numerica.

Esempio: Rappresentazione del tipo real in Turbo Pascal.
Il tipo real usato in Turbo Pascal, usa 6 byte (48 bit): $b_{47}b_{46}\ldots b_1b_0$. Il bit piu' a sinistra (bit pi significativo), b_{47}, rappresenta il bit di segno ($s = b_{47}$; s=0 per il segno e' +, s=1 per il segno -). La mantissa *m* e' lunga 40 bit; essendo il bit piu' significativo sempre pari ad 1 esso non viene memorizzato; sono memorizzati invece i rimanenti 39 bit, in corrispondenza di $b_{46}\ldots b_9b_8$, che rappresentano la parte frazionaria f. I rimanenti 8 bit, $b_7\ldots b_1b_0$, denotano il valore $e = exp + 129$ per ottenere sempre un valore positivo come esponente.

Il valore *v* associato ai valori f, s, e vale:

$v = (-1)^s(1.f)2^{e-129}$;

dove:
s = b_{47} (bit di segno)
f = parte frazionaria, memorizzata in $b_{46}\ldots b_8$
e = esponente polarizzato, memorizzato in $b_7\ldots b_0$.

Il valore nullo e', per convenzione, rappresentato da *e*=0.

Esempio
Qual e', in Turbo Pascal, la rappresentazione binaria del reale 8?
Risp. $8 = (1.0)2^3$. Il bit di segno vale *s*=0; f=0 (e quindi 39 bit pari a 0); *exp*=3. e quindi l'esponente polarizzato vale 129+3 = 132 = 10000100_2 = 0x84. Pertanto, il reale 8 e' rappresentato dalla stringa di bit 0000 0000 0000 0000 0000 1000 0100, espressa in base 16 da 0x00 00 00 00 00 84. □

Esempio.
Verificare che 0x800000000084 rappresenta il numero -8 secondo la codifica del tipo *real* del Turbo Pascal.

Esempio.
Esprimere in base 10 il numero rappresentato da 0x400000000080 secondo la codifica del tipo *real* del Turbo Pascal.
Risp. Espandiamo il numero esadecimale in modo da avere la stringa di bit: 0100 0000 0000 0000 0000 1000 0000, da cui si ricavano le seguenti informazioni: s=0; f=100 0000 0000 0000 0000; e = 1000 0000, che in base 10 diventano: s=0; e=128; f=0.5, da cui: $v=(-1)^0(1.5)2^{128-129} = (1.5)2^{-1} = 0.75\square$

Estremi dell'intervallo dei valori rappresentabili per il tipo real Turbo Pascal Il valore v e' legato alla coppia (f,e) dalla relazione $|v| = (1.f)2^{e-129}$. L'intervallo e' simmetrico rispetto allo 0. Consideriamo quindi solo valori positivi (s=0). I valori limite si trovano calcolando v con i valori limite di f ed e:
$f_{min} = 0$;
$f_{max} = 2^{-1} + 2^{-2} + \ldots + 2^{-39} = 1 - 2^{-39}$;
$e_{min} = 1$ (e = 0 e' riservato per la rappresentazione di 0);
$e_{max} = 2^8 - 1 = 255$;

da cui, considerando i soli valori positivi, si ha come valore minimo maggiore di 0, $v_{min} = (1.0)2^{1-129} = 2^{-128} \approx (2.9)10^{-39}$; mentre il valore massimo vale $v_{max} = (1 + (1 - 2^{-39}))2^{255-129} = (2 - 2^{-39})2^{126} \approx (1.7)10^{38}$;$\square$

Visualizzazione del contenuto di memoria in Turbo Pascal. Il TP consente di esaminare il contenuto della memoria (dump), visualizzando in forma esadecimale. Cio' puo' essere adoperato, ad esempio, come esercizio per verificare la rappresentazione interna di valori. A tal fine, in modalita' debug-watch, si deve usare l'opzione ,m dopo il nome della variabile da esaminare. Prestare attenzione al fatto che, nel caso di dump di piu' di un byte, i byte sono visualizzati in ordine inverso, dal meno significativo al piu' significativo. Per la corretta interpretazione vanno quindi letti al contrario.

4 Rappresentazione di caratteri (codifica ASCII)

Il termine carattere indica un tipo di dato non-numerico. Sono caratteri tutti i simboli che possono essere immessi da tastiera. Sono esempi di carattere i simboli di punteggiatura, le cifre decimali, le lettere dell'alfabeto, lo spazio, etc. Per rappresentare i caratteri all'interno dell'elaboratore e' necessario associare ad ognuno di essi un numero binario. Una codifica molto usata e' la codifica ASCII (American Standard Code for Information Exchange) che adotta 7 bit per rappresentare 128 caratteri differenti. I caratteri ASCII sono suddivisi in 4 gruppi di 32 caratteri.

I primi 32 caratteri, da 0x00 a 0x1F (31), formano un insieme speciali di caratteri (caratteri di controllo), cui va aggiunto l'ultimo carattere 0x7F (127) - il carattere per la cancellazione, utilizzati nella trasmissione di dati o come controllo di dispositivi di stampa.

Il secondo gruppo di 32 caratteri, da 0x20 (32) a 0x3F (63), comprende vari simboli di punteggiatura, caratteri speciali e cifre decimali. Il codice ASCII di una cifra decimale c e' 0x3c. Ad esempio il codice della cifra 0 e' 0x30, di 1 0x31 etc., ossia, in decimale, 48, 49, etc.

Il terzo gruppo (da 0x40 a 0x5F) contiene i caratteri alfabetici maiuscoli; il quarto ed ultimo (da 0x60 a 0x7F) i caratteri alfabetici minuscoli. Le stringhe di bit dei due codici ASCII relativi ad uno stesso carattere, minuscolo e maiuscolo, differiscono solamente per il bit in posizione 5. Ad esempio, il codice della lettera E vale 0100 0101 (0x45) ed il codice di e vale 0110 0101 (0x65). In base 10, tale differenza dei codici vale 32 (=00100000_2).

Decimale	Simbolo	Decimale	Carattere	Decimale	Carattere	Decimale	Carattere
0	NUL	32		64	@	96	‘
1	SOH	33	!	65	A	97	a
2	STX	34	″	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	’	71	G	103	g
8	BS	40	(	72	H	104	h
9	TAB	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	□

5 Rappresentazione di valori di verita’

Questa sezione pone l’attenzione sulle proprieta’ di un sistema algebrico (l’algebra di Boole) per la manipolazione di elementi logici, definito dal matematico inglese George Boole nel 1854, e che costituisce la base per la manipolazione di espressioni logiche nei linguaggi di programmazione, nonche’ per la progettazione di sistemi digitali in genere. Il valore 1 rappresenta il valore vero, mentre lo 0 rappresenta falso.

L’algebra di Boole e’ caratterizzata da:

- un insieme composto da due valori distinti, $\mathcal{A} = \{1, 0\}$;
- operatore unario $\neg$ (NOT logico), definito da: $\neg 0 = 1; \neg 1 = 0$;
- operatore binaro $+$ (OR logico), definito da: 0 + 0 = 0; 0 + 1 =1; 1 + 0 = 1; 1 + 1 =1
- operatore binaro $\cdot$ (AND logico), definito da: $0 \cdot 0 = 0$; $0 \cdot 1 = 0$; $1 \cdot 0 = 0$; $1 \cdot 1 = 1$

Una importante caratteristica dell’algebra di Boole e’ che le proprieta’ sono soddisfatte a coppie: se una proprieta’ P e’ vera, allora e’ vera anche la proprieta’ duale P', ottenuta a partire da P e sostituendo:

1 con 0 e viceversa
+ con · e viceversa;

Ecco un elenco delle principali equivalenze dell'algebra; A,B,C sono elementi di $\mathcal{A}$:

Idempotenza
$A+A = A$
$A \cdot A = A$

Commutativita'
$A+B = B+A$
$A \cdot B = B \cdot A$

Associativita'
$A+(B + C) = (A + B) + C$
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$

Distributivita'
$A+(B \cdot C) = (A+B) \cdot (A+C)$
$A \cdot (B+C) = (A \cdot B) + (A \cdot C)$

Doppia negazione
$\neg\neg A = A$

Identita'
$A \cdot 1 = A$
$A + 0 = A$

Terzo escluso
$A + (\neg A) = 1$

Contraddizione
$A \cdot (\neg A) = 0$

Leggi di De Morgan
$\neg (A \cdot B) = (\neg A) + (\neg B)$
$\neg (A + B) = (\neg A) \cdot (\neg B)$

Indice

1 Introduzione **2**

2 Rappresentazione di numeri interi **2**
2.1 Richiami sui sistemi posizionali . . . 3
2.2 Codifica mediante Sistema di Numerazione Posizionale . . . 4
2.2.1 Conversione fra basi . . . 4
2.2.2 Uso della base di partenza . . . 4
2.2.3 Uso della base di arrivo . . . 6
2.2.4 Relazione fra rappresentazioni nelle basi B e B^k Esempio . . . 7
2.3 Rappresentazione in complemento alla base . . . 7
2.3.1 Proprieta' della rappresentazione in complemento alla base . . . 9
2.4 Rappresentazione in Modulo e Segno . . . 11
2.5 Riassunto dei metodi di codifica . . . 11

3 Rappresentazione di numeri reali **11**
3.1 Virgola mobile . . . 11
3.1.1 Precisione . . . 13

4 Rappresentazione di caratteri (codifica ASCII) **15**

5 Rappresentazione di valori di verita' **16**