Codifica delle informazioni nella memoria

I computer possono utilizzare parole di memoria di 2 byte (16 bit), 4 byte (32 bit), 8 byte (64 bit). Poiché il tempo per elaborare una parola è costante, la sua lunghezza è uno dei modi per misurare la potenza e la velocità di calcolo di un computer.
La memorizzazione delle informazioni avviene in modo diverso se si tratta di numeri o di caratteri (lettera dell'alfabeto o un qualsiasi simbolo della tastiera).

Numeri interi

La memorizzazione di un numero **intero con segno** si serve di 16 bit, di cui il primo viene utilizzato per la memorizzazione del segno:

segno	**numero**														
0 = + 1 = –															

I numeri interi positivi rappresentabili con un computer che usa una parola di 16 bit possono variare da 0 a +32767, che corrisponde a $2^{15} - 1$.

Numero decimale **0**

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Numero decimale **+32767**

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

Per rappresentare i numeri negativi non si può usare la stessa codifica dei numeri positivi impostando semplicemente a 1 il primo bit; se così facessimo infatti la somma di un numero con il suo opposto non sarebbe zero.
Si osservi l'esempio seguente in cui, per semplicità, ci serviamo di una parola di 8 bit, scrivendo separatamente e in neretto il primo bit del segno:

numero decimale +7 codifica binaria 111 impostazione degli 8 bit **0** 0000111
numero decimale –7 codifica binaria 111 impostazione degli 8 bit **1** 0000111

Se sommiamo i numeri binari dei due byte così ottenuti abbiamo:

00000111 +
10000111 =
———————
10001110

Le regole della **somma binaria** sono:
0 + 0 =0,
0 + 1 =1,
1 + 0 =1,
1 + 1 = 0 con riporto di 1,
1 + 1 + 1 = 1 con riporto di 1.

Il numero ottenuto non corrisponde a 0. Dobbiamo allora trovare una codifica per i numeri negativi tale che la somma di un numero con il suo opposto produca 0 come risultato.

Il tipo di codifica scelta si basa sulla **regola del complemento a 2**:

> Dato un numero in forma binaria, per ottenere il complemento a 2 si sostituisce 0 dove c'è 1 e 1 dove c'è 0 e al numero ottenuto si somma 1.

ESEMPIO

Complemento a 2 del numero binario 1101001.

```
Numero binario              1101001
scambiamo i valori 1 e 0    0010110
sommiamo 1                  0010110 +
                                  1 =
                            ———————
                              10111
```

Con questa regola il numero dato si trasforma nel suo opposto se si conviene di trascurare l'eventuale riporto che eccede i 16 bit dedicati alla memorizzazione del numero.

ESEMPIO

Calcolo dell'opposto del numero binario 100110110000110.

```
bit del segno   bit del numero      numero decimale corrispondente
0               100110110000110     19846

calcoliamo il suo complemento a 2:
consideriamo il numero con tutte le 16 cifre   0100110110000110
scambiamo 0 e 1                                1011001001111001 +
sommiamo 1                                                   1 =
                                               ————————————————
                                               1011001001111010
```

Sommiamo adesso il numero dato e il suo complemento a 2:

```
  0100110110000110 +
  1011001001111010 =
————————————————————
10000000000000000
```

Allora, se trascuriamo il primo bit che è quello che eccede i 16 dedicati alla rappresentazione, la somma è 0. Dunque il complemento a 2 di un numero rappresenta il suo opposto.

In definitiva, poiché il massimo numero positivo che si può rappresentare con 15 bit è, come abbiamo visto

0111111111111111 valore decimale 32767 $(2^{15} - 1)$

il minimo valore negativo è il suo complemento a 2 che è

1000000000000000 valore decimale –32768 (-2^{15})

Il più piccolo valore intero rappresentabile in un computer si indica comunemente con il termine MININT, il più grande con MAXINT; si ha quindi, con 16 bit, che:

MININT = –32768 MAXINT = +32767

Numeri reali

La rappresentazione dei numeri **reali** (non interi) può essere fatta usando

• la virgola fissa (**fixed point**):

1.5 0.00123 12.564

oppure

• la virgola mobile (**floating point**):

3E–4 12E+18 1.47E–3

(Si noti che per scrivere i numeri decimali viene usato sempre il punto al posto della virgola, secondo la notazione anglosassone).

Il secondo modo di rappresentare i numeri si chiama anche **notazione scientifica** o **rappresentazione esponenziale**.
La lettera E sta al posto di **10 elevato a** ed è seguita dall'esponente a cui elevare la base 10: la potenza di 10 va moltiplicata per il numero (**mantissa**) che precede la lettera E.

ESEMPIO

Il numero in forma esponenziale

7.2342 E–5

significa

$7.2342 \times 10^{-5} = 0.000072342$

Questo tipo di rappresentazione viene usato solitamente in calcoli scientifici, quando si devono trattare numeri molto grandi o molto piccoli scrivendoli in una forma compatta e facilmente leggibile.

ESEMPI

Il numero reale

0.000000000000003

può essere scritto, molto più semplicemente, nella forma

3 E–15

Il numero reale

123000000000000

può essere scritto

1.23 E+14

Un numero in virgola mobile può essere scritto rappresentando la mantissa con un numero, in valore assoluto, maggiore di 0 e minore di 10.

ESEMPIO

Il numero reale

125.74 E+14

si può scrivere

1.2574 E+16

Questa rappresentazione, che si chiama notazione esponenziale **normalizzata**, mette in evidenza la mantissa (la parte prima di E) e l'esponente (la parte dopo E).

In realtà occorre ricordare che i numeri all'interno del computer sono rappresentati con sequenze di bit.
Nel caso di valori numerici in base 2 la rappresentazione in virgola mobile normalizzata prevede che il numero sia rappresentato con mantissa maggiore di 0 e minore di 2.

ESEMPIO

Il numero reale binario

$$1001.011_2 \times 2^{-5}$$

nella rappresentazione in virgola mobile normalizzata si scrive:

$$1.001011_2 \times 2^{-2}$$

Questo significa, in pratica, che per normalizzare un dato in base 2 bisogna spostare il punto di separazione tra la parte intera e la parte frazionaria, e modificare in corrispondenza l'esponente di 2, in modo che la parte intera valga sempre 1:

$$1.bbbbb_2 \times 2^n$$

Inoltre esiste la necessità di una rappresentazione normalizzata standard che consenta la *portabilità* del software, cioè la possibilità che i programmi funzionino su macchine di produttori diversi. Lo standard per l'aritmetica in virgola mobile si chiama **IEEE 754** (*Standard for Floating-Point Arithmetic*, 2008).
Il formato IEEE ha la seguente struttura:

$$s1.bbbbbbbb...b \times 2^{esponente}$$

dove ogni *b* è un bit 0 o 1 e *s* rappresenta il bit del segno.

La rappresentazione interna dei numeri nella memoria dell'elaboratore subisce una limitazione dovuta alle dimensioni fisiche della cella di memoria, che si traduce in un limite per il numero di cifre significative associate al numero.
Tutto questo viene descritto con il termine **precisione** della rappresentazione interna dei numeri.

Per i numeri binari, lo standard definisce tre formati principali:
- Singola precisione (*binary32*)
- Doppia precisione (*binary64*)
- Quadrupla precisione (*binary128*)

che codificano i numeri in sequenze, rispettivamente, di 32, 64 e 128 bit.
Per esempio, per la singola precisione si ha il seguente schema di rappresentazione:

Si osservi che nel **campo mantissa** è memorizzata solo la parte frazionaria, mentre la parte intera, essendo sempre 1, non è memorizzata e diventa un bit nascosto, che consente di avere a disposizione un bit aggiuntivo per la precisione nella rappresentazione del numero.