Rappresentazione dei numeri (S. 1.4)

- Per rappresentare un numero in un calcolatore si usa una stringa di bit chiamata **numero binario**
- La stringa di bit è un vettore costituito da n bit (con n>0), ciascuno dei quali può avere valore 0 o 1
- Un numero binario B a n bit è rappresentato da $b_{n-1}b_{n-2}\ldots b_1b_0$
- Il valore numerico intero naturale *val(B)* è compreso in $[0,2^n[$
- $val(B) = b_{n-1}\cdot 2^{n-1} + b_{n-2}\cdot 2^{n-2} + \ldots + b_1\cdot 2^1 + b_0\cdot 2^0$
- I numeri interi relativi sono rappresentati tramite una di tre rappresentazioni
  - segno e valore assoluto
  - complemento a 1
  - complemento a 2
- Se il bit b_{n-1} vale 0 allora il valore del numero è positivo o nullo, mentre se vale 1 il valore è negativo. Il bit b_{n-1} viene detto bit di segno

decimale	binario	
0	0000	
1	0001	
2	0010	
3	0011	$1\cdot 2^0 = 1$
4	0100	$1\cdot 2^1 = 2$
5	0101	$1\cdot 2^1 + 1\cdot 2^0 = 3$

Rappresentazioni

- Con **segno e valore assoluto**, il bit di segno cambia da 0 a 1 per passare dal valore positivo al valore negativo.
  Es. 3 = 0011 e -3 = 1011
  - Per convertire da decimale a binario: dividere per 2 e prendere i resti delle divisioni
- **Complemento a 1**, si commuta ciascun bit.
  Es. 3 = 0011 e -3 = 1100
- **Complemento a 2**, si aggiunge 1 al complemento a 1 del numero. Es. 3 = 0011 e -3 = 1101
  - Per convertire da binario (in compl. a 2) a decimale, se il bit più significativo è 0, come per la rappresentazione con segno e valore assoluto, altrimenti complementare e sommare 1
- La tecnica del complemento a 2 è usata per il calcolo di addizione e sottrazione
- Per convertire da binario a decimale è necessario sapere **a priori** quale codifica è stata usata per il numero binario

	segno e valore assoluto	complemento a 2
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
-1	1001	1111
-2	1010	1110
-3	1011	1101

Addizioni e sottrazioni con segno

- La somma di due numeri binari può avere un **riporto in uscita** (carry-out)
- Es. la somma di due numeri binari 1 e 1 (a 1 bit) è il numero binario 10, ovvero somma 0 e riporto 1
- Per sommare numeri a *n* bit si usa la stessa tecnica, si somma ciascun addendo da destra verso sinistra e si propaga l'eventuale riporto. Il riporto in uscita di una coppia di bit più a destra diventa **riporto in entrata** (carry-in) dei bit appena più a sinistra
- Per la sottrazione (A-B), si usa la rappresentazione in complemento a due del sottraendo (B), e si trascura il riporto in uscita dei bit più significativi

```
0 +   1 +   0 +   1 +
0 =   0 =   1 =   1 =
-     -     -     -
0     1     1     1 0
                  ↑ ↑
          riporto   somma
          in uscita
```

```
00010 +   00101 +   01001 +
00100 =   00101 =   00011 =
-----     -----     -----
00110     01010     01100
2+4=6     5+5=10    9+3=12
```

```
00110 +   11010 +   11110 +
11110 =   11110 =   00011 =
-----     -----     -----
00100     11000     00001
6-2=4     -6-2=-8   -2+3=1
```


Estensione e riduzione del segno

- Per aumentare il numero di bit usati per codificare un numero, per es. per adattarlo alla lunghezza della parola di memoria, senza cambiarne il valore
  - Estensione: se positivo aggiungere 0 a sinistra; se negativo aggiungere 1 a sinistra
    - Es. numero positivo +3 = 011 = 0011 = 00011
    - Es. numero negativo -3 = 101 = 1101 = 11101
  - Riduzione: se positivo togliere 0 a sinistra, ma fino a prima che emerga il bit 1; se negativo togliere 1 a sinistra, ma fino a prima che emerga il bit 0

Trabocco

- Con n bit si possono rappresentare numeri relativi nell'intervallo $[-2^{n-1}, 2^{n-1}[$
  - Per es. con n = 4 si possono rappresentare i numeri nell'intervallo [-8,+7[
- Se l'operazione aritmetica produce un risultato fuori dall'intervallo suddetto si dice che si verifica un trabocco (**overflow**)
- L'addizione di due addendi **discordi** (ovvero che hanno segni differenti) è in effetti una sottrazione, allora il trabocco non si può verificare; se i due addendi sono **concordi**, il trabocco si può verificare
- Si verifica trabocco se e solo se i due addendi sono concordi e il bit di segno della somma è diverso da quello degli addendi

Numeri frazionari

- Con 32 bit, per un **intero** con segno in complemento a due, possono essere codificati i numeri nell'intervallo da -2^{31} a $+2^{31}-1$, ovvero in decimale da -10^{10} a $+10^{10}$ circa
- Per **numeri frazionari** si può considerare che la virgola sia presente (fissa) tra il bit b_{31} e il bit b_{30} (appena dopo il bit di segno), si può quindi rappresentare un valore piccolo come 10^{-10}
  - Per numeri piccoli va bene, per numeri grandi no
- Per rappresentare numeri grandi e numeri piccoli, si usa la rappresentazione in **virgola mobile** (floating point)
  - Si usano: un bit di segno, alcuni bit significativi, alcuni bit per l'esponente (base implicita 2) con segno
  - Con 32 bit, si ha un bit per il segno, 23 bit significativi, 8 bit per l'esponente; in decimale si possono rappresentare i numeri da $\pm 10^{-38}$ a $\pm 10^{38}$ circa

10111.1011 =
$2^4 + 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-3} + 2^{-4}$
= 16 + 4 + 2 + 1 + 0,5 + 0,125 + 0,0625
= 23,6875

normalizzando (esponente 4)
1.01111011

S=0 (positivo)
esponente=1000 0011 (è 4+127 bias)

bit significativi=011 1101 1000 ... 0
$2^4 \cdot (1 + 2^{-2} + 2^{-3} + 2^{-4} + 2^{-5} + 2^{-7} + 2^{-8})$

Rappresentazione dei caratteri (S. 1.5)

- Lo schema di codifica dei caratteri più comune è l'ASCII (American Standard Code for Information Interchange)
- I caratteri alfanumerici, i simboli e i caratteri di controllo sono rappresentati da codici a 7 bit, ma si usano 8 bit (quindi un byte) per praticità
- I quattro bit di ordine più basso dei codici ASCII sono le cifre da 0 a 9, e sono i primi 10 valori. Questa codifica è detta codifica binaria dei decimali o BCD (binary-coded decimal)
- Per rappresentare alfabeti diversi da quello inglese si è resa necessaria una codifica che usa più di 8 bit (es. UTF-8, che impiega un numero variabile di byte per la codifica di ciascun simbolo)