

**Università di Napoli Federico II – Scuola Politecnica e delle Scienze di Base**  
**Corso di Laurea in Ingegneria Informatica**



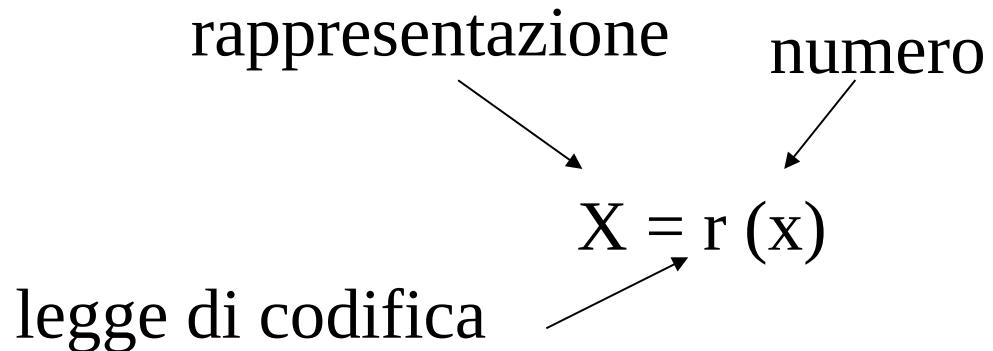
# **Corso di Calcolatori Elettronici I**

Rappresentazione dei numeri



# Introduzione

- Pressoché tutte le applicazioni, per realizzare le loro elaborazioni, utilizzano calcoli numerici, indici, contatori, etc...
  - Così come per qualsiasi altro tipo di dato, anche i numeri, per essere immagazzinati nella memoria di un calcolatore, devono essere tradotti in sequenze di bit
  - Questa operazione è a tutti gli effetti analoga ad una generica operazione di codifica
  - In particolare, l'alfabeto sorgente è costituito da numeri





# Strategie di codifica

- Tipicamente, si adotta una codifica a lunghezza fissa
- Il numero di bit varia a seconda della cardinalità dell'insieme dei numeri che si desidera rappresentare
  - » Nella pratica, resta comunque pari ad un multiplo di 8 bit (tipicamente 8, 16, 32, 64 bit).
- L'associazione di un numero alla parola codice viene
  - » Realizzata differentemente a seconda della tipologia di numeri che si desidera rappresentare
    - ◆ naturali, relativi, razionali, etc...
  - » Influenzata da aspetti che mirano a preservare la facile manipolazione delle rappresentazioni da parte del calcolatore
    - ◆ operazioni aritmetiche, confronti logici, etc...
  - » Le operazioni aritmetiche vengono eseguite sulle rappresentazioni binarie dei numeri



# Overflow - Definizione

- Sia la dimensione che il numero dei registri in un calcolatore sono finiti
- La cardinalità degli insiemi numerici che si rappresentano è, invece, infinita
- È inevitabile dunque che in un insieme di cardinalità infinita solo un sotto-insieme finito di elementi possa essere rappresentato
- Gli operatori aritmetici, pur essendo talvolta chiusi rispetto all'intero insieme, quasi certamente non lo sono rispetto al sotto-insieme di cardinalità finita
- Quando accade che, per effetto di operazioni, si tenta di rappresentare un numero non contenuto nel sotto-insieme si parla di *overflow*

# Overflow - Esempio

- Si assume di rappresentare i numeri naturali mediante un'operazione di codifica sul sottoinsieme da 0 a 127 (7 bit)
  - » La somma  $100 + 100$  genera un overflow, essendo il numero 200 non rappresentabile nel sottoinsieme
  - » La differenza  $5 - 10$  genera un overflow essendo il numero  $-5$  non rappresentabile nel sottoinsieme

# Rappresentazione dei numeri naturali

- Si voglia rappresentare attraverso sequenze di bit di lunghezza costante pari ad  $n$  l'insieme dei numeri naturali
  - » Il numero degli elementi rappresentabili è pari a  $2^n$
  - » Tipicamente, volendo rappresentare sempre anche lo zero, si rappresentano i numeri compresi tra 0 e  $2^n - 1$
- L'associazione tra ogni numero e la propria rappresentazione avviene, nei casi pratici, nella maniera più intuitiva
  - » Ad ogni numero viene associata la stringa di bit che lo rappresenta in un sistema di numerazione binario posizionale.
- L'overflow avviene quando si tenta di rappresentare un numero esterno all'intervallo  $[0; 2^n - 1]$

# Esempio

Rappresentazione dei numeri naturali su 4 bit

$n=4$

Intervallo: [0;15]

Codifica:  $X=x$

$x$	$X_2$	$X_{10}$
15	1111	15
14	1110	14
13	1101	13
12	1100	12
11	1011	11
10	1010	10
9	1001	9
8	1000	8
7	0111	7
6	0110	6
5	0101	5
4	0100	4
3	0011	3
2	0010	2
1	0001	1
0	0000	0

# Operazioni sui numeri naturali su 4 bit

- Per realizzare le operazioni, il calcolatore può lavorare direttamente sulle rappresentazioni
- La correttezza dei calcoli è garantita dalle leggi dell'aritmetica binaria posizionale (analoghe a quelle della classica aritmetica decimale)
- L'overflow può essere facilmente rilevato attraverso la valutazione del riporto (o del prestito) sull'ultima cifra.



# Esempi

$6+$ $8=$ ----- <b>14</b>	$0110+$ $1000=$ ----- <b>1110</b>	$11 -$ $5=$ ----- <b>6</b>	$1011-$ $0101=$ ----- <b>0110</b>	$0101\times$ $0011=$ ----- <b>0101</b>
$14+$ $3=$ ----- <b>17</b>	$1110+$ $0011=$ ----- <b>10001</b>	$9-$ $7=$ ----- <b>2</b>	$1001-$ $0111=$ ----- <b>0010</b>	$0101$ $0101=$ $0000==$ $0000==$ ----- <b>0001111</b>

overflow

# Rappresentazione dei numeri relativi

- Esistono diverse tecniche
  - Segno e modulo
    - Corrispondente a quella comunemente utilizzata per i calcoli “a mano”
    - Poco utilizzata in macchina per le difficoltà di implementazione degli algoritmi, basati sul confronto dei valori assoluti degli operandi e gestione separata del segno
  - Complementi
    - Complementi alla base
    - Complementi diminuiti
  - Per eccessi

# Rappresentazione in segno e modulo

- Utilizza un singolo bit (quello più significativo) per codificare il segno
- Utilizza i restanti bit per rappresentare il modulo (che è un numero naturale)
- Se si utilizzano n bit:
  - » La legge di codifica  $X=r(x)$  è:  $X = |x| + 2^{n-1} * \text{sign}(x)$
  - » Si possono rappresentare i numeri relativi compresi nell'intervallo  $[-(2^{n-1} - 1); 2^{n-1} - 1]$
  - » In totale i numeri rappresentati sono  $2^n - 1$  (e non  $2^n$ )
    - ◆ Ciò dipende dal fatto che, a causa della natura di questa codifica, lo zero ammette due possibili rappresentazioni dette *zero positivo* e *zero negativo*.

# Esempio

Rappresentazione in segno e modulo su 4 bit

$n=4$

Intervallo:  $[-7;7]$

Codifica:

$$X = |x| + 8 * \text{sign}(x)$$

$x$	$X_2$	$X_{10}$
7	<b>0111</b>	7
6	<b>0110</b>	6
5	<b>0101</b>	5
4	<b>0100</b>	4
3	<b>0011</b>	3
2	<b>0010</b>	2
1	<b>0001</b>	1
0	<b>0000;1000</b>	<b>0;8</b>
-1	<b>1001</b>	<b>9</b>
-2	<b>1010</b>	<b>10</b>
-3	<b>1011</b>	<b>11</b>
-4	<b>1100</b>	<b>12</b>
-5	<b>1101</b>	<b>13</b>
-6	<b>1110</b>	<b>14</b>
-7	<b>1111</b>	<b>15</b>



# Operazioni in segno e modulo

- Diversamente dalla rappresentazione dei numeri naturali, questa volta non è possibile lavorare direttamente sulle rappresentazioni dei numeri per realizzare le operazioni aritmetiche
- È necessario, invece, lavorare separatamente sul segno e sul modulo
- Quando, ad esempio, si sommano due numeri di segno discorde, bisogna determinare quello con modulo maggiore e sottrarre ad esso il modulo dell'altro. Il segno del risultato sarà quello dell'addendo maggiore in modulo
- Tale caratteristica, insieme con il problema della doppia rappresentazione dello zero, rende i calcoli particolarmente laboriosi e, per questo motivo, non è molto utilizzata nella pratica

# Rappresentazione in complementi alla base



- Una seconda tecnica per la rappresentazione dei numeri relativi consiste nell'associare a ciascun numero il suo resto modulo  $M=2^n$ , definito come:

$$X = |x|_M = x - [x/M] * M$$

- Questo tipo di codifica, su  $n$  bit, è equivalente ad associare:
  - » il numero stesso ( $X=x$ ), ai numeri positivi compresi tra 0 e  $2^{n-1} - 1$ ;
    - ◆  $0 \leq x \leq 2^{n-1} - 1 \Rightarrow [x/M] = 0 \Rightarrow |x|_M = x$
  - » il numero  $X = 2^n - |x|$ , ai numeri negativi compresi tra  $-2^{n-1}$  e -1;
    - ◆  $-2^{n-1} \leq x \leq -1 \Rightarrow [x/M] = -1 \Rightarrow |x|_M = x + 2^n$
- I numeri rappresentati sono quelli compresi nell'intervallo

$$[-2^{n-1}; 2^{n-1} - 1]$$

# Esempio

Rappresentazione in complementi alla base su 4 bit

$n=4$

Intervallo:  $[-8; 7]$

Codifica:

$$X=x; \quad 0 \leq x \leq 7$$

$$X=2^n - |x|; \quad -8 \leq x \leq -1$$

$x$	$X_2$	$X_{10}$
7	0111	7
6	0110	6
5	0101	5
4	0100	4
3	0011	3
2	0010	2
1	0001	1
0	0000	0
-1	1111	15
-2	1110	14
-3	1101	13
-4	1100	12
-5	1011	11
-6	1010	10
-7	1001	9
-8	1000	8



# Complementi alla base: proprietà

- Questa rappresentazione ha il fondamentale vantaggio di permettere, nell'ambito di operazioni aritmetiche, di lavorare direttamente sulle rappresentazioni.
- La proprietà sulla quale questa affermazione si basa è la seguente:  
*la rappresentazione della somma (algebrica) di  $x$  ed  $y$  si ottiene come somma (modulo- $M$ ) delle rappresentazioni di  $x$  e  $y$ ; analoghe sono le proprietà della differenza e del prodotto.*

$$|x+y|_M = ||x|_M + |y|_M|_M$$

- Questo tipo di codifica conserva, inoltre, la proprietà delle rappresentazioni di avere il primo bit alto se (e solo se) il corrispondente numero è negativo.

# Esempi di addizioni in complementi alla base

$2+$ $- 6 =$ ----- $-4$	$0010 +$ $1010 =$ ----- $1100$	$- 2 +$ $- 3 =$ ----- $-5$	$1110 +$ $1101 =$ ----- $11011$
----------------------------------	---	-------------------------------------	--

si ignora

somma  
modulo-16

È possibile effettuare la somma direttamente tra le rappresentazioni modulo-M: il risultato ottenuto in questo modo, è proprio la rappresentazione (modulo-M) del risultato corretto.



# Complementi alla base: overflow

- Quando si sommano due numeri si può avere overflow solo se essi sono entrambi positivi o entrambi negativi (i bit più significativi degli addendi sono uguali)
- Se gli addendi sono di segno concorde, si ha overflow se il segno del risultato è discorda
  - es. addendi positivi (MSB=0) e risultato negativo (MSB=1)
- Il riporto (carry) NON è un'indicazione di overflow



# Complementi alla base: complementazione

- In complementi alla base, a partire dalla rappresentazione di un numero, è anche particolarmente semplice ottenere la rappresentazione del suo opposto
- È infatti sufficiente *complementare tutti i bit a partire da sinistra, tranne l'uno più a destra ed eventuali zero successivi.*
- Questa ulteriore caratteristica consente di realizzare le sottrazioni attraverso la composizione di una complementazione (nel senso suddetto) ed un'addizione.
- Nell'aritmetica in complementi alla base, di conseguenza, l'addizionatore e il complementatore rappresentano i componenti fondamentali per la realizzazione di tutte le operazioni.



# Esempi di complementazione su 4 bit

- La rappresentazione di  $6_{10}$  su 4 bit è  $0110_2$ .
- Complementando tutti i bit tranne l'uno più a destra e gli zero successivi si ottiene:  $1010_2$ .
- $1010_2$  è la rappresentazione di -6 in complementi alla base.
  
- La rappresentazione di  $5_{10}$  su 4 bit è  $0101_2$ .
- Complementando tutti i bit tranne l'uno più a destra e gli zero successivi si ottiene:  $1011_2$ .
- $1011_2$  è la rappresentazione di -5 in complementi alla base.
  
- La rappresentazione di  $1_{10}$  su 4 bit è  $0001_2$ .
- Complementando tutti i bit tranne l'uno più a destra e gli zero successivi si ottiene:  $1111_2$ .
- $1111_2$  è la rappresentazione di -1 in complementi alla base.

# Esempio di moltiplicazione in complementi alla base

$$\begin{array}{r} 2* \\ - 3= \\ \hline \end{array}$$

$$- 6$$

0010×

1101=

-----

0010

0000=

0010==

0010====

-----

0011010

si ignora



prodotto

# Estensione del segno



- Problema:
  - Sia dato un intero  $N$ , rappresentato in complemento mediante  $n$  bit
  - Rappresentare  $N$  usando  $n+q$  bit ( $q>0$ )
- Soluzione:
  - Fare  $q$  copie del bit più significativo
- Esempio:  $-2 = (110)_2$  su 3 bit diventa  $(111110)_2$  su 6 bit



# Complementi diminuiti

- La rappresentazione in complementi diminuiti costituisce un’ulteriore alternativa per la codifica dei numeri relativi.
- Concettualmente è analoga alla rappresentazione in complementi alla base.
- La differenza rispetto ad essa è che la legge di codifica dei numeri negativi è leggermente differente:
  - »  $X=2^n - |x|$ ; (complementi alla base)
  - »  $X=2^n - 1 - |x|$ ; (complementi diminuiti)
- I numeri rappresentabili, se si utilizzano n bit, sono quelli compresi nell’intervallo  $[-(2^{n-1} - 1); 2^{n-1} - 1]$ .
- Anche in questo caso, quindi, le cifre rappresentabili sono  $2^n - 1$  e ancora una volta è lo zero ad avere una doppia rappresentazione.

# Esempio

Rappresentazione in complementi diminuiti su 4 bit

$n=4$

Intervallo:  $[-7; 7]$

Codifica:

$$X=x; \quad 0 \leq x \leq 7$$

$$X = 2^n - 1 - |x|; \quad -7 \leq x \leq -1$$

$x$	$X_2$	$X_{10}$
7	<b>0111</b>	7
6	<b>0110</b>	6
5	<b>0101</b>	5
4	<b>0100</b>	4
3	<b>0011</b>	3
2	<b>0010</b>	2
1	<b>0001</b>	1
0	<b>0000;1111</b>	0;15
-1	<b>1110</b>	14
-2	<b>1101</b>	13
-3	<b>1100</b>	12
-4	<b>1011</b>	11
-5	<b>1010</b>	10
-6	<b>1001</b>	9
-7	<b>1000</b>	8



# Complementi diminuiti: perché?

- Maggiore semplicità con cui è possibile calcolare la rappresentazione dell'opposto di un numero, a partire dalla rappresentazione del numero stesso: basta semplicemente complementare tutti i bit della rappresentazione indistintamente.
- Esempio:
  - » la rappresentazione in complementi diminuiti su 4 bit di 4 è 0100;
  - » complementando tutti i bit si ottiene 1011;
  - » 1011 è la rappresentazione in complementi diminuiti su 4 bit di -4.
  - » la rappresentazione in complementi diminuiti su 4 bit di - 6 è 1001;
  - » complementando tutti i bit si ottiene 0110;
  - » 0110 è la rappresentazione in complementi diminuiti su 4 bit di 6.

# Aritmetica in complementi diminuiti



- Componenti:
  - » Ancora l'addizionatore
  - » Un complementatore
- Il risultato però deve essere opportunamente “corretto” (per renderlo compatibile con l’aritmetica in modulo  $2^n-1$ ).
- In particolare deve essere aggiunta un’unità al risultato nei seguenti casi:
  - » se entrambi gli addendi sono negativi;
  - » se un addendo è positivo, l’altro negativo e la somma è positiva.
- Nei casi suddetti l’aritmetica degli interi positivi darebbe overflow
  - » L’overflow (carry) quindi può essere interpretato come la necessità di effettuare la correzione

# Esempi di somme in complementi diminuiti.



$$\begin{array}{r} -2+ \\ -3= \\ \hline -5 \end{array} \quad \begin{array}{r} 1101+ \\ 1100= \\ \hline 11001+ \\ \quad \quad \quad 1= \\ \hline 1010 \end{array}$$

**overflow**

Somma di due numeri negativi.  
Si è generato overflow nell'aritmetica degli interi positivi.  
**Necessita correzione.**

$$\begin{array}{r} 5+ \\ -2= \\ \hline 3 \end{array} \quad \begin{array}{r} 0101+ \\ 1101= \\ \hline 10010+ \\ \quad \quad \quad 1= \\ \hline 0011 \end{array}$$

**overflow**

Somma di un numero positivo e un numero negativo.  
Il risultato è positivo.  
Si è generato overflow nell'aritmetica degli interi positivi  
**Necessita correzione.**

$$\begin{array}{r} 3+ \\ -4= \\ \hline -1 \end{array} \quad \begin{array}{r} 0011+ \\ 1011= \\ \hline 1110 \end{array}$$

Somma di un numero positivo e un numero negativo.  
Il risultato è negativo.  
Non si è generato overflow nell'aritmetica degli interi positivi  
Non necessita alcuna correzione.

# Rappresentazione eccesso-k

- La rappresentazione in eccesso-k costituisce un metodo diverso da quello dei resti in modulo per ricondurre i numeri negativi a positivi.
- In particolare, tutti i numeri sono traslati “verso l’alto” di k, che viene scelto maggiore o uguale al numero più piccolo da rappresentare:  $X = r(x) = x+k$

# Esempio

Rappresentazione in eccesso-8 su 4 bit

$n=4$

Intervallo:  $[-8; 7]$

Codifica:

$$X = x + k;$$

$x$	$X_2$	$X_{10}$
7	<b>1111</b>	<b>15</b>
6	<b>1110</b>	<b>14</b>
5	<b>1101</b>	<b>13</b>
4	<b>1100</b>	<b>12</b>
3	<b>1011</b>	<b>11</b>
2	<b>1010</b>	<b>10</b>
1	<b>1001</b>	<b>9</b>
0	<b>1000</b>	<b>8</b>
-1	<b>0111</b>	<b>7</b>
-2	<b>0110</b>	<b>6</b>
-3	<b>0101</b>	<b>5</b>
-4	<b>0100</b>	<b>4</b>
-5	<b>0011</b>	<b>3</b>
-6	<b>0010</b>	<b>2</b>
-7	<b>0001</b>	<b>1</b>
-8	<b>0000</b>	<b>0</b>

# Rappresentazione eccesso-k - Proprietà

- Analogamente al caso dei complementi diminuiti, la somma va corretta aggiungendo o sottraendo la costante k, e quindi in maniera sufficientemente semplice
- Le moltiplicazioni e le divisioni risultano invece più complesse.
- Il vantaggio di tale codifica è che viene conservata la proprietà della disuguaglianza sulle rappresentazioni:

$$x_1 > x_2 \Leftrightarrow X_1 > X_2$$

- Questa rappresentazione, perciò, è utilizzata soltanto laddove siano richieste fondamentalmente somme algebriche e confronti logici fra gli operandi.
- Tipicamente si utilizza per rappresentare gli esponenti nella rappresentazione in virgola mobile

# Overview



DIE  
TI.  
UNI  
NA

Rappresentazione di interi con 4 bit

Decimale	Senza segno	Segno e modulo	Complemento a uno	Complemento a due	Eccesso 8
+8	1000	n/d	n/d	n/d	n/d
+7	0111	0111	0111	0111	1111
+6	0110	0110	0110	0110	1110
+5	0101	0101	0101	0101	1101
+4	0100	0100	0100	0100	1100
+3	0011	0011	0011	0011	1011
+2	0010	0010	0010	0010	1010
+1	0001	0001	0001	0001	1001
(+0)	0000	0000	0000	0000	1000
(-0)	n/d	1000	1111	n/d	n/d
-1	n/d	1001	1110	1111	0111
-2	n/d	1010	1101	1110	0110
-3	n/d	1011	1100	1101	0101
-4	n/d	1100	1011	1100	0100
-5	n/d	1101	1010	1011	0011
-6	n/d	1110	1001	1010	0010
-7	n/d	1111	1000	1001	0001
-8	n/d	n/d	n/d	1000	0000

# Rappresentazione dei numeri reali

- Nel caso della rappresentazione di numeri razionali sussiste il problema legato alla caratteristica dei numeri razionali di costituire un insieme *denso*, mentre per la rappresentazione si ha a disposizione un numero finito di bit
- A causa di ciò, anche se si sceglie per la rappresentazione un intervallo limitato, non tutti i numeri all'interno di esso potranno essere rappresentati.
- Quando si tenta di rappresentare un numero per cui non è stata prevista una rappresentazione, spesso si associa a tale numero la rappresentazione del numero “più vicino”, cioè quello che lo *approssima* meglio
- In questo caso si parla di *errore di approssimazione*



# Underflow

- Quando in un'elaborazione si tenta di rappresentare un numero “troppo vicino” allo zero, l'errore di approssimazione può far sì che la rappresentazione scelta sia quella dello zero
- Questa evenienza può condizionare pesantemente i calcoli successivi nel seguito dell'elaborazione a causa delle peculiarità della cifra *zero* (che, ad esempio, non può essere utilizzata al denominatore di una frazione)
- Si parla allora di *underflow*.
- Questa problematica è molto sentita nell'ambito del calcolo numerico, specialmente nelle applicazioni che, per loro natura, tendono a lavorare con quantità molto piccole (p.es. applicazioni di calcolo differenziale)

# Rappresentazione di numeri reali

- Vengono usate due notazioni:

A) Notazione in virgola fissa

- Dedica parte delle cifre alla parte intera e le altre alla parte frazionaria
  - + XXX .YY

B) Notazione in virgola mobile

- Dedica alcune cifre a rappresentare un esponente della base che indica l'ordine di grandezza del numero rappresentato



# Numeri reali in virgola fissa

- Quando di un numero frazionario si rappresentano separatamente la parte intera e la parte frazionaria si parla di rappresentazione in *virgola fissa*.
- La rappresentazione dei due contributi (che sono numeri interi) può essere realizzata secondo una delle tecniche viste in precedenza.
- In questo caso la posizione della virgola è fissa e resta sottintesa.

# Numeri reali in virgola fissa

- La stringa  $0, b_{-1}b_{-2}\dots b_{-m}$  si interpreta come
$$b_{-1}2^{-1} + b_{-2}2^{-2} + \dots + b_{-m}2^{-m}$$
- Esempio:
  - $0,1011$  si interpreta come
$$2^{-1} + 2^{-3} + 2^{-4} = 1/2 + 1/8 + 1/16 = 0,5 + 0,125 + 0,0625 = 0,6875$$
  - ovvero come
$$6875 / 10000 = 625 * 11 / 16 * 625 = 11 / 16$$
$$11 * 2^{-4} = 0,6875$$
- La stringa  $1011$  è rappresentativa dell'intero  $(11)_{10}$  che va scalato del fattore  $2^{-4}$



# Numeri reali in virgola mobile

- Un numero reale  $x$  può essere rappresentato dalla tripla

$$(s, m, e)$$

tale che:

$$x = (-1)^s \cdot m \cdot b^e$$

dove:

- »  $s$  è il segno ( $s=0$  positivo,  $s=1$  negativo)
- »  $m$  è detta mantissa
- »  $e$  è detto esponente
- »  $b$  è la base di numerazione adottata
- Il numero di bit utilizzati per la rappresentazione di mantissa ed esponente è fisso



# Normalizzazione

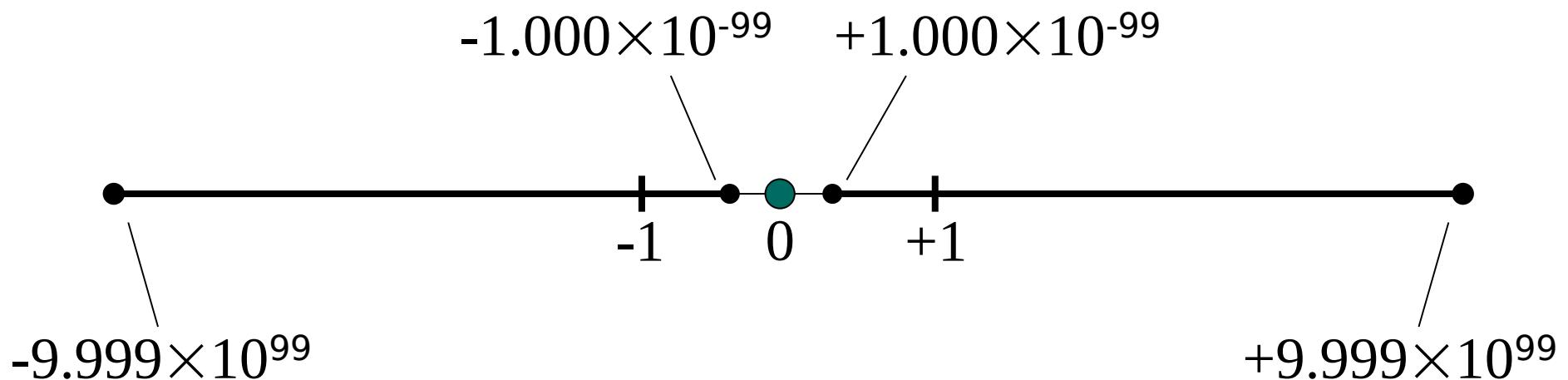
- Per ciascun numero esistono infinite coppie che lo rappresentano.
- Esempio ( $b=10$ ):
  - ◆ 346.09801 è rappresentato da
    - » ( 346.09801, 0 ) oppure
    - » ( 346098.01, -3 ) oppure
    - » ( 0.034609801, 4 ) etc...
- Per rappresentazione normalizzata si intende convenzionalmente quella in cui la mantissa è maggiore o uguale a uno e minore della base :  $1 \leq m < b$   
$$( 3.4609801, 2 )$$



## Esempio: intervallo di rappresentazione

- Con  $b=10$ , usando 4 cifre per  $m$  e 2 per  $e$  (più due bit per i relativi segni), l'insieme rappresentabile (utilizzando solo rappresentazioni normalizzate) è:

$$[-9.999 \times 10^{99}, -1.000 \times 10^{-99}] \cup \{0\} \cup [+1.000 \times 10^{-99}, +9.999 \times 10^{99}]$$





# Approssimazione

- Come è facile verificare, in questo tipo di rappresentazione l'approssimazione non è costante.
- In particolare la precisione assoluta è molto spinta in prossimità dello zero e va diminuendo progressivamente a mano a mano che il numero aumenta (in valore assoluto).
- Ad esempio:
  - » in prossimità dello zero l'errore massimo che può essere commesso è pari a  $1.001 \cdot 10^{-99} - 1.000 \cdot 10^{-99} = 0.001 \cdot 10^{-99}$ ;
  - » in prossimità dell'estremo superiore dell'intervallo di rappresentazione, invece, l'errore massimo che si può commettere è  $9.999 \cdot 10^{99} - 9.998 \cdot 10^{99} = 0.001 \cdot 10^{99}$ .
- Si commettono quindi “errori piccoli” su “numeri piccoli” ed “errori grandi” su “numeri grandi”.
- Quello che resta inalterato è invece l'errore relativo, costante su tutto l'asse di rappresentabilità.

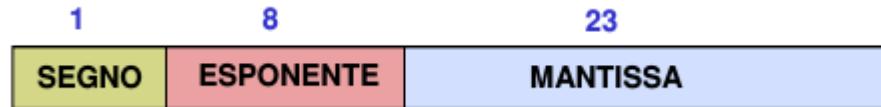
# Overflow e Underflow

- L'errore relativo dipende dal numero di cifre della mantissa.
- Gli estremi dell'intervallo di rappresentazione dipendono dal numero di cifre dell'esponente.
- Nel caso precedente di 2 cifre per l'esponente, si ha overflow per numeri maggiori (in modulo) di  $10^{99}$  e si ha underflow per numeri minori (in modulo) di  $10^{-99}$ .

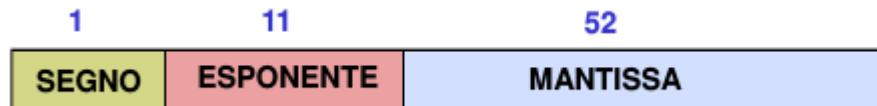
# Standard IEEE 754 (1985)



- Formato standard indipendente dall'architettura
- Precisione semplice a 32 bit:

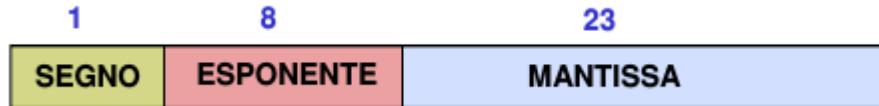


- Precisione doppia a 64 bit



- Notazioni in modulo e segno
- Alcune configurazioni dell'esponente sono riservate

# IEEE 754 a 32 bit



- $x = (-1)^s \times 1.F \times 2^{\text{exp-bias}}$
- **ESPONENTE**
  - Rappresentato in eccesso 127
  - L'intervallo è [-126, +127]
  - Il valore -127 è riservato per rappresentazioni speciali
- **MANTISSA**
  - Se ne rappresenta solo la parte frazionaria
  - Nella rappresentazione normalizzata binaria, la parte intera è sempre pari a 1, per cui il bit corrispondente alla parte intera non viene memorizzato

# Esempio

- 8.5
- Segno +
- 8.5 in binario:  $1000.1 = 1.0001 * 2^3$ 
  - Mantissa: 000100000000000000000000000000
  - Esponente:  $3 + 127 = 130 = 10000010$
  - **Numero: 0 10000010 000100000000000000000000**