Informatica

Rappresentazione dei numeri

Numerazione binaria

Sistemi di numerazione

- Non posizionali: numerazione romana
- Posizionali: viene associato un peso a ciascuna posizione all'interno della rappresentazione del numero. Il valore del numero viene ottenuto con la somma dei prodotti di ciascuna cifra per il relativo peso.

$$(137,2)_b = 1 * b^2 + 3 * b^1 + 7 * b^0 + 2 * b^{-1}$$

osservazioni

Se b è la base della numerazione ogni numero viene rappresentato da un insieme di cifre che ha cardinalità b.

Indicando con c_i ogni singola cifra allora

$$0 <= c_i <= b-1$$

Base 10: notazione posizionale

I numeri sono rappresentati da una stringa di cifre 0..9 ed eventualmente da una virgola.

Ogni cifra ha un peso diverso dovuto alla posizione occupata: 234,56 è diverso da 342,65

Il peso è dato dalle potenze del 10:

$234{,}56 = 2*10^2 + 3*10^1 + 4*10^0 + 5*10^{-1} + 6*10^{-2}$

$342{,}65 = 3*10^2 + 4*10^1 + 2*10^0 + 6*10^{-1} + 5*10^{-2}$

Base 2: notazione posizionale

I numeri sono rappresentati da una stringa di cifre 0, 1 ed eventualmente da una virgola.

Ogni cifra ha un peso diverso dovuto alla posizione occupata: 101,1 è diverso da 110,01

Il peso è dato dalle potenze del 2:

$101,1 = 1*2^2 + 0*2^1 + 1*2^0 + 1*2^{-1} + 0*2^{-2}$

$110,01 = 1*2^2 + 1*2^1 + 0*2^0 + 0*2^{-1} + 1*2^{-2}$

Base 8 e base 16

- I numeri in base 8 sono rappresentati da una stringa di cifre 0..7 ed eventualmente da una virgola.

  $506 = 5*8^2 + 0*8^1 + 6*8^0$

- I numeri in base 16 sono rappresentati da una stringa di cifre 0..9, A..F ed eventualmente da una virgola.

  $BF3 = 11*16^2 + 15*16^1 + 3*16^0$

Vantaggi sistemi posizionali

- Esprimono qualunque valore finito
- Ad ogni valore numerico corrisponde una sola sequenza
- Le regole formali per le operazioni aritmetiche sono indipendenti dalla base della rappresentazione

Quante cifre occorrono?

Con n cifre in base b il numero massimo rappresentabile è:

$N_{max} = (b-1)b^{n-1} + (b-1)b^{n-2} + \ldots + (b-1)b^0$

$= b^n - 1$

Per ogni numero N_b con n cifre abbiamo che

$b^{n-1}-1 < N_b <= b^n-1$ da cui

$b^{n-1} < N_b+1 <= b^n$ e quindi

$n-1 < \log_b (N_b+1) <= n$

Numero di cifre: esempio

Sia N_{10}= 318

Con base 2 occorrono

$n = \log_2(318+1) = 8.32 = 9$

Con base 8 occorrono

$n = \log_8(318+1) = 2.77 = 3$

Con base 16 occorrono

$n = \log_{16}(318+1) = 2.08 = 3$

Conversione da base 2 a base 10

Sia N_2= 100101

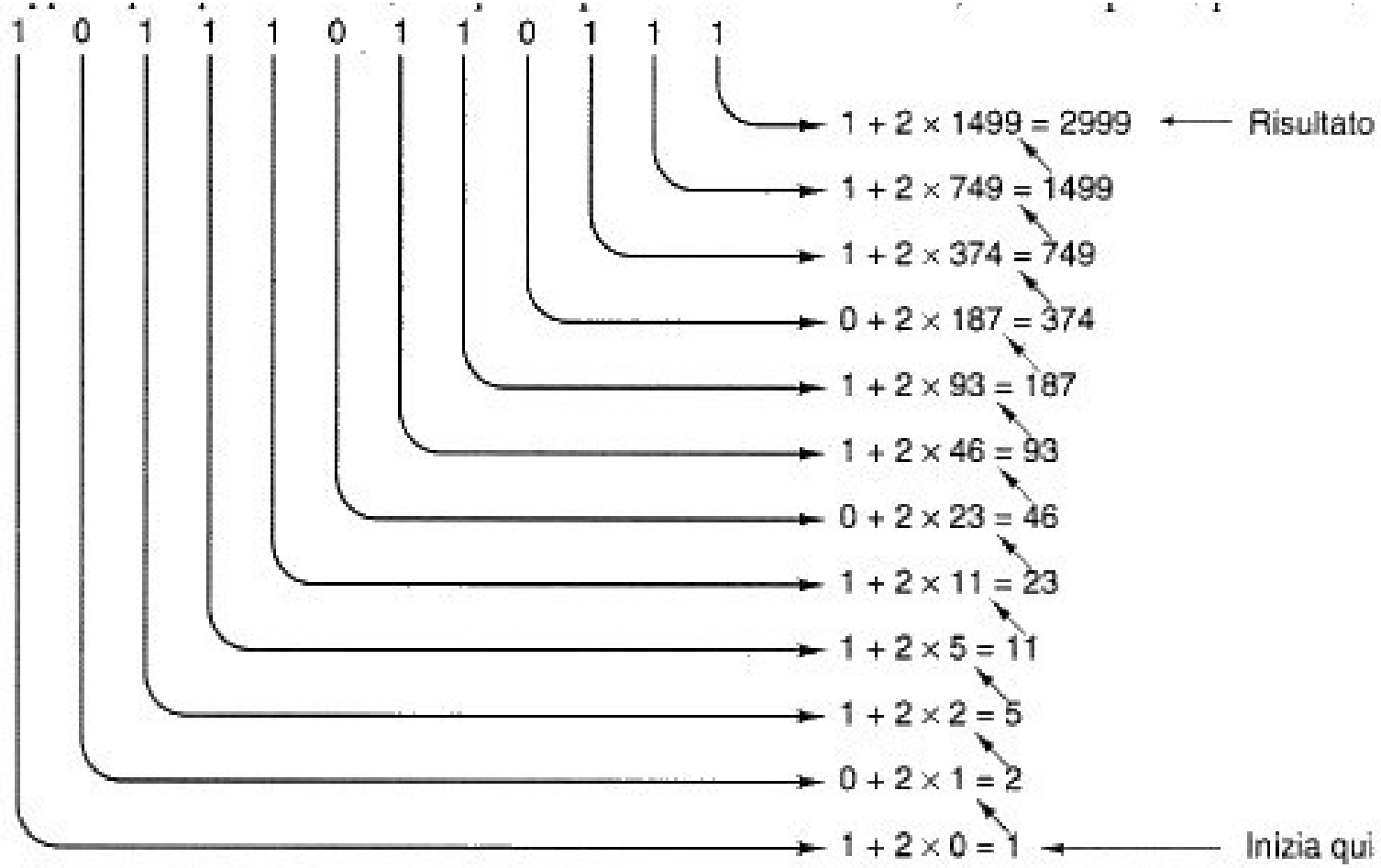
Applicando la definizione di numero posizionale

$(100101)_2 = 1*2^5+0*2^4+0*2^3+1*2^2+0*2^1+1*2^0$

$= 32 + 0 + 0 + 4 + 0 + 1$

$= 37$

Conversione: da 2 a 10



Conversione: da 2 a 10

Il metodo visto nella diapositiva precedente fornisce utili suggerimenti per la realizzazione di un algoritmo di conversione:

numero ← 0
FINCHÈ ci sono cifre da convertire
    numero ← 2 * numero + cifra

numero è il risultato della conversione

Conversione: da 2 a 10

Analogamente per la parte frazionaria:

$(0.0101)_2 = ½(0 + ½(1 + ½(0 + ½ * 1)))$

numero ← 0

Partendo dall'ultima cifra

while ci sono cifre

    numero ← numero / 2 + cifra

numero è il risultato

Conversione: da 2 a 8

Poiché $8 = 2^3$ si raggruppano i bit a blocchi di 3 partendo da destra, e si convertono.

1 101 011 001 diventa

1 5 3 1

Il procedimento si può invertire: per trasformare da base 8 a base 2 si sostituisce ogni cifra ottale nel corrispondente binario utilizzando 3 bit.

Conversione: da 2 a 16

Poiché 16 = 2^4 si raggruppano i bit a blocchi di 4 partendo da destra, e si convertono.

11 0101 1101 diventa

3 5 D

Il procedimento si può invertire: per trasformare da base 16 a base 2 si sostituisce ogni cifra esadecimale nel corrispondente binario utilizzando 4 bit.

Conversione: da 10 a 2

Esempio:

1836	Resto
918	0
459	0
229	1
114	1
57	0
28	1
14	0
7	0
3	1
1	1
0	1

Si riporta la seconda colonna partendo dal fondo: 11100101100

Conversione: da 10 a 2

Algoritmo

Sia n_{10} il valore da convertire e b la nuova base

While n diverso da 0

$resto_i \leftarrow n \text{ mod } b$

$n \leftarrow n \text{ div } b$

Dove

mod è l'0perazione modulo

div è la divisione intera

Conversione: da 10 a 2

Algoritmo:

si sottrae al numero da convertire la potenza del 2 più grande minore del numero stesso. Si usa la differenza ottenuta per ripetere il processo. Il numero binario si ottiene segnando 1 alle potenze del 2 utilizzate, 0 alle altre.

		2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
413	$256= 2^8$									
157	$128= 2^7$	1	1	0	0	1	1	1	0	1
29	$16= 2^4$									
13	$8= 2^3$									
5	$4= 2^2$									
1	$1= 2^0$									

Conversione: da 10 a 2

Esempio:

0.234	
0.468	0
0.936	0
0.872	1
0.742	1
0.482	1
0.964	0
0.928	1
0.856	1
0.712	1
0.424	1
0.848	0
0.696	1
0.392	1

il numero in binario è: 0.0011101111011

Il processo non è ancora terminato. Si può continuare per ottenere un valore più preciso

Conversione: da 10 a 2

Esempio: 0.3

0.6	0
0.2	1
0.4	0
0.8	0
0.6	1

I numeri periodici cambiano cambiando la base con cui si rappresentano

Da questo punto in avanti la sequenza si ripete. Il numero in base 2 è periodico mentre non lo è in base 10.

Approssimazione numeri frazionari

Quando si converte un numero decimale frazionario in una nuova base può capitare che la rappresentazione del numero nella nuova base abbia infinite cifre.

In questo caso occorre approssimarlo con una precisione prestabilita.

Approssimazione numeri frazionari

Esempio: trasformare $(0.4)_{10}$ in base 2 a meno di 1/100.

Per l'approssimazione richiesta sono necessarie 7 cifre binarie

0.4		
0.8	0	
0.6	1	
0.2	1	
0.4	0	
0.8	0	
0.6	1	
0.2	1	0.0110011

Addizione binaria

```
 0 1  1 1 1 0
 101101 +   45
 100111 =   39
-------------
1010100     84
```

```
   1 1 1  1 1 0
 111111 +  63
      1 =   1
-------------
1000000     64
```

Aritmetica dei calcolatori

L'aritmetica dei calcolatori è diversa dall'aritmetica:

- I calcolatori eseguono operazioni su numeri la cui precisione è finita e fissa.
- La rappresentazione dei numeri avviene utilizzando il sistema binario.

Precisione finita

La quantità di memoria utilizzabile per la rappresentazione di un numero è fissata al momento in cui viene progettata la macchina.

Questo determina l'insieme dei numeri rappresentabili.

Precisione finita: problemi

- I numeri interi relativi sono un insieme chiuso rispetto ad addizione, sottrazione e moltiplicazione.
- i numeri a precisione finita **non** sono chiusi rispetto alle operazioni di addizione, sottrazione, moltiplicazione.

Alcune importanti proprietà dell'addizione e della moltiplicazione non sono più valide.

Precisione finita

- Non vale la proprietà associativa:

  a + (b + c) potrebbe essere diverso da (a + b) + c

- Non vale la proprietà distributiva:

  a * (b + c) potrebbe essere diverso da a*b + a*c

Potrebbero esistere anche problemi sull'unicità dell'elemento neutro.

Numeri binari relativi

Rappresentazione in ***complemento a due***.

Il bit più significativo rappresenta il segno

Numeri positivi: il segno è 0. La parte rimanente rappresenta il numero in binario puro.

Numeri negativi: il bit di segno è 1. La parte rimanente rappresenta il complemento a due del numero positivo.

Da decimale a complemento a 2

Se numero >= 0 allora

    0| binario puro

Se numero < 0 allora

    scrivere numero in binario puro

    complementare bit a bit, segno compreso

    aggiungere +1

Da complemento a 2 a decimale

Se bit segno = 0

binario puro da convertire in decimale

Se bit segno = 1

complementare bit a bit segno compreso

sommare +1

convertire in decimale

aggiungere il segno -

Complemento a 2: aritmetica

Rappresentare il numero -23 con 8 bit.

23 → 00010111 → 11101000 + 1 → 11101001

Calcolare 12 – 23

12 → 00001100 +

-23 → 11101001 =

11110101 → 00001010 + 1 →00001011 → 11

Complemento a 2: aritmetica

Calcolare 125 + 3 utilizzando 8 bit in complemento a 2.

125 → 64 + 32 + 16 + 8 + 4 + 1 → 01111101 +

3 → 2 + 1 → 00000011 =

10000000

Il numero ottenuto ha il segno uguale ad 1. E' negativo. Il valore rappresentato è –128.

E' un caso di overflow. Con 8 bit 125 + 3 non si può fare.

Regola generale: quando sommando due valori con uguale segno si ottiene un numero con segno diverso si ha *overflow*.

Rappresentazione numeri razionali

I numeri razionali vengono espressi attraverso la notazione scientifica :

$$n = f \times 10^{e}$$

in cui

- **f** si chiama frazione, o mantissa,
- **e** è un numero intero positivo o negativo chiamato esponente.

La versione "informatica" di questa notazione si chiama **floating point**

Floating point: esempi

$3,14 = 0,314 \times 10^{1} = 3,14 \times 10^{0} = 0,00314 \times 10^{3} = \ldots\ldots$

$1,4142 = 0,14142 \times 10^{1} = 0,14142 \times 10^{2} = 14142 \times 10^{-4}$

$234 = 0,234 \times 10^{3} = 234 \times 10^{0} = \ldots\ldots\ldots\ldots$

$0,00001 = 0,1 \times 10^{-4} = 1 \times 10^{-5} = \ldots\ldots\ldots\ldots\ldots$

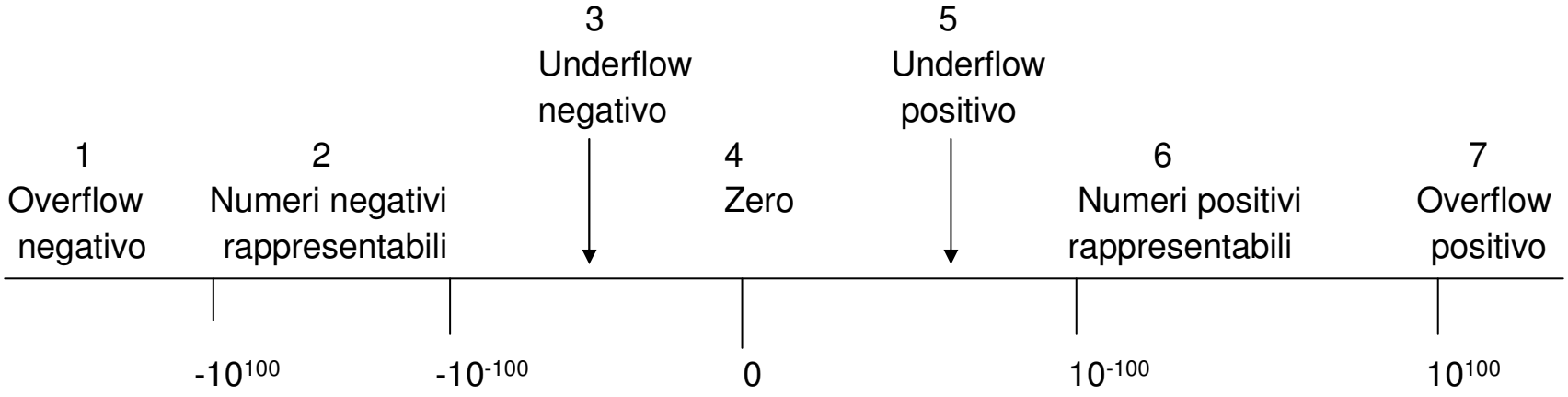
- Ogni numero può essere rappresentato in diversi modi.
- L'ordine di grandezza dei numeri rappresentabili è dato dal numero di cifre a disposizione per l'esponente.
- La precisione è data dal numero di cifre a disposizione per la frazione.

Floating point

Supponiamo di utilizzare:

- 3 cifre con segno per la frazione
- 2 cifre con segno per l'esponente
- La frazione f è $0{,}1 \leq |f| < 1$

Con questa rappresentazione i numeri vanno da $0{,}100 \times 10^{-99}$ a $0{,}999 \times 10^{99}$



Floating point

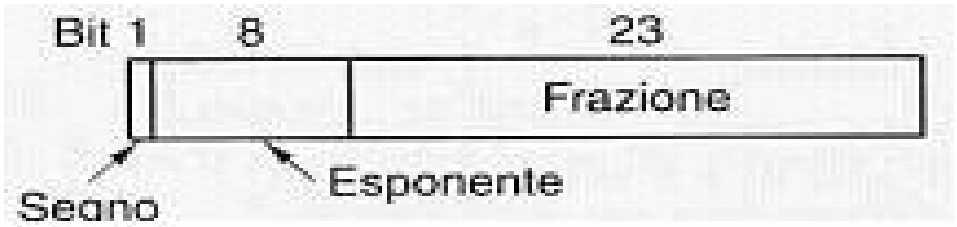
Verso la fine degli anni '70 la IEEE creò un comitato per la standardizzazione dell'aritmetica floating-point.

Obiettivo:

- rendere possibile lo scambio di dati floating-point fra calcolatori diversi.
- fornire ai progettisti hardware un modello corretto.

Il risultato del lavoro portò alla creazione dello Standard 754 (IEEE, 1985), frutto del lavoro svolto in massima parte da una persona sola, il professore di matematica di Berkeley William Kahan.

Floating point

singola precisione (32 bit) esponente ad eccesso 127	
doppia precisione (64 bit) esponente ad eccesso 1023	
precisione estesa (80 bit), per il calcolo interno	

Floating point

- Il bit 1 a sinistra della frazione non deve essere memorizzato, poiché è sempre presente.
- Ogni rappresentazione consiste in un 1 bit implicito, una virgola binaria implicita e poi 23 o 52 bit arbitrari.
- Se tutti i 23 o 52 bit della frazione sono 0, la frazione ha il valore numerico 1,0;
- se sono tutti 1, la frazione è numericamente leggermente inferiore a 2,0.
- Per evitare confusioni con una frazione tradizionale, la combinazione dell' 1 implicito, della virgola binaria implicita e dei 23 o 52 bit espliciti si chiama significando, invece di chiamarsi frazione o mantissa.
- Tutti i numeri normalizzati hanno un significando s per $1 <= s < 2$.

Floating point

Oltre ai numeri normalizzati, lo standard comprende anche altri quattro tipi numerici:

- i numeri denormalizzati per rappresentare underflow. Hanno un esponente che vale 0 e una frazione rappresentata dai 23 o 52 bit.
- due zeri uno positivo e l'altro negativo, determinati dal bit di segno. Entrambi hanno un esponente 0 e una frazione 0.
- il valore "infinito" rappresentato da un esponente con tutti 1 (non permesso per i numeri normalizzati) e una frazione di 0.
- un formato speciale, chiamato NaN (Not a Number), con esponente 1 e qualunque sequenza di bit diversa da 0.

Da decimale a floating point

13.25 → 1101.01

Si normalizza: 1.10101 con esponente 3

Il nuovo esponente sarà 127 + 3 = 130

127 in base 2 → 01111111 +

11 =

10000010

Il segno 0

0 10000010 10101000000000000000000

Da decimale a floating point

0.8125 → 0.1101

Normalizzando: 1.101 con esponente -1

Il nuovo esponente sarà: 01111111 –

1

01111110

Il segno: 0

0 01111110 10100000000000000000000

Da floating point a decimale

Supponiamo di avere in esadecimale

40A40000

In binario:

0100 0000 1010 0100 0000 0000 0000 0000

Segno 0 → positivo

Esponente: 10000001 – 01111111 = 10 = 2

Mantissa: 1.01001 → 101.001

Il numero è: 5.125

Da floating point a decimale

Sia: C0A60000

1100 0000 1010 0110 0000 0000 0000 0000

Segno negativo

Esponente: 100000001 – 011111111 = 10

Mantissa: 1.010011 → 101.0011 → 5.1875

Da floating point a decimale

Sia BE000000

1011 1110 0000 0000 0000 0000 0000 0000

Segno negativo

Esponente negativo 01111111 -

$$\frac{01111100 =}{11}$$

Mantissa: 1.0 $\rightarrow$ 0.001 $\rightarrow$ - 0.125