



Università degli Studi di Milano
CdL in Informatica per la comunicazione digitale
AA 2025/2026

Architettura degli elaboratori
Lez 3:
**Rappresentazione di numeri
frazionari – Parte B**

Marco Tarini
marco.tarini@unimi.it

31



**Normalizzare un numero significa avere
la virgola che “galleggia” -> “floating” point**

0 decine di migliaia	0 decine di migliaia	0 decine di migliaia
0 migliaia	0 migliaia	0 migliaia
0 centinaia	2 centinaia	0 centinaia
0 decine	3 decine	0 decine
3 unità	4 unità	0 unità
1 decimi	7 decimi	1 decimi
4 centesimi	5 centesimi	2 centesimi
1 millesimi	0 millesimi	7 millesimi

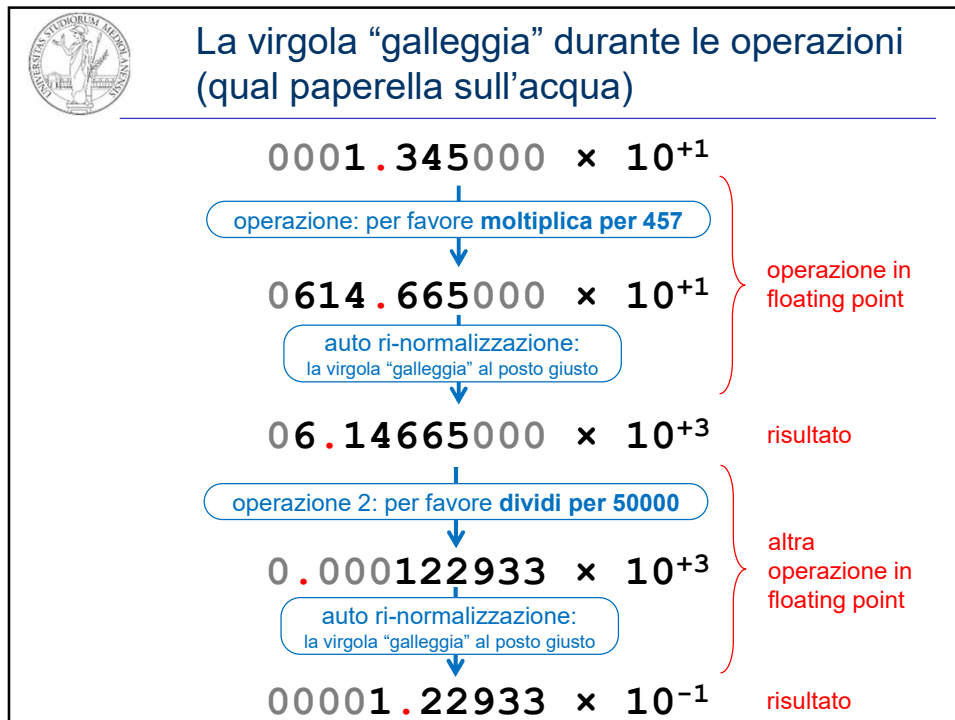
3.141
=
 3.141×10^0

234.75
=
 $2.3475 \times 10^{+2}$

0.127
=
 1.27×10^{-1}

Architettura degli elaboratori - 32 - Aritmetica binaria

32



33

Limiti di rappresentazione in virgola mobile
Overflow / underflow. Esempio (in base 10)

- Consideriamo una rappresentazione *float* in base 10 provvista di:
 - segno, **esponente** di due cifre con offset 50 (valori da -50 a +49),
 - e **mantissa** di tre cifre
- Possiamo quindi rappresentare numeri come 1.467×10^{-19}
- Limitazioni: non possiamo rappresentare numeri che siano...
 - Troppo grandi in valore assoluto, cioè:
 - se positivi: $> +1.999 \times 10^{49}$ --- se negativi: $< -1.999 \times 10^{49}$
 - Troppo piccoli in valore assoluto, cioè:
 - se positivi: $< +1.000 \times 10^{-50}$ --- se negativi: $> -1.000 \times 10^{-50}$
- Se un'operazione tenta di generare un numero non rappresentabile...
 - Se tenta di produrre un numero troppo **lontano dallo zero** (grande in valore assoluto): genera un errore di **OVERFLOW**
L'esponente è un positivo troppo sopra allo zero! (è $> +49$)
 - Se tenta di produrre un numero troppo **vicino allo zero** (piccolo in valore assoluto): genera un errore di **UNDERFLOW**
L'esponente è un negativo troppo sotto allo zero! (è < -50)

Architettura degli elaboratori - 35 - Aritmetica binaria

35



Limiti di rappresentazione in virgola mobile: Precisione limitata

- Possono essere rappresentati, ma con limitazioni, i numeri appartenenti agli intervalli citati.
- Per questi numeri, i limiti sono dati dalla precisione con cui possiamo rappresentarli. Esempio: in base 10, con tre cifre di mantissa:
 - ▶ Il valore $1/3$ verrà rappresentato come 0.333×10^0 . Naturalmente 0.333 è solo un arrotondamento di $1/3$ (cioè $0.\bar{3}$).
 - ▶ Il valore π verrà rappresentato come 0.314×10^1 . Di nuovo, è solo un arrotondamento.
- In generale questo genere di arrotondamento non causano grossi problemi (soprattutto se si dispone di abbastanza mantissa).
- Però bisogna tenerne conto:
 - ▶ in teoria, $1/3 \times 3 = 1$,
ma $(0.333 \times 10^0) \times (0.3 \times 10^1) = 0.999 \times 10^0 \neq 1$.

Architettura degli elaboratori

- 36 -

Aritmetica binaria

36



Limiti di rappresentazione in virgola mobile: quanta precisione? Una nota pratica.

- Quante cifre (significative) rappresenta un numero in float, in base 10?
 - ▶ Per esempio, quando dico
 $3.1415000\dots$ (3 virgola 14, 15 «e rotti») o
 $3,141,500,000$ (3 miliardi, 151 milioni, 400 mila «e rotti») o
 0.000031415 (zero virgola 0, 0, 0, 0, 0, 3, 1, 4, 1, 5 «e rotti»)
esprimono cifre di dimensioni diverse di cui conosco lo stesso numero di cifre: cinque (quelle scritte in chiaro non sono note)
- Cifre in base 2 nei formati:
i bit di mantissa, più uno (lo «1.» iniziale della forma normalizzata)
- Come sappiamo: $2^{10} \cong 10^3$
- Ne consegue che 10 cifre in binario sono circa equivalenti a 3 cifre in decimale
- Quindi: nella codifica «single precision» (23 bit di mantissa),
in cui abbiamo $23+1=24$ cifre in binario,
abbiamo circa $24 \times 3/10 = 7$ cifre decimali significative (e un po')
- Usando la codifica «double precision» (52 bit di mantissa),
in cui abbiamo $52+1=53$ cifre in binario,
abbiamo circa $53 \times 3/10 = 16$ cifre decimali significative (quasi)

Architettura degli elaboratori

- 37 -

Aritmetica binaria

37



Lo standard IEEE 754: rappresentazioni speciali

- Un numero «float», secondo lo standard IEEE 754, può assumere queste configurazioni:

Normalizzato	+/-	tutto il resto	qualsiasi combinazione
Subnormale	+/-	000...000	qualsiasi combinazione ≠0
+/- Zero	+/-	000...000	0
+/- Infinito	+/-	111...111	0
Not A Number	+/-	111...111	qualsiasi combinazione ≠0

esponente

mantissa

Architettura degli elaboratori

- 44 -

Aritmetica binaria

44



Configurazioni speciali

- Alcuni valori speciali sono rappresentati da una configurazione apposita.
- Sono:
 - **0+** («zero positivo», o «+0», o semplicemente «0», «ZERO»)
rappresenta: lo 0 esatto
oppure: un valore positivo molto piccolo (non altrimenti rappresentabile)
 - **0-** («zero negativo», o «-0»)
rappresenta: un valore negativo molto piccolo (non altrim. rappresent.)
 - **+INF** («più infinito»)
rappresenta: infinito,
oppure un valore positivo molto grande (idem)
 - **-INF** («meno infinito»)
rappresenta: meno infinito
oppure un valore negativo molto grande in modulo (idem)
 - **NAN** (o «NaN», o «Not-A-Number»)
è risultato di un'operazione illecita o dal risultato non determinabile

Architettura degli elaboratori

- 45 -

Aritmetica binaria

45



Configurazioni speciali: ZERO+, ZERO-, +INF, -INF,

- +INF e -INF sono il risultato di operazioni che generano overflow
- 0+ e 0- possono essere il risultato di operazioni che generano underflow
- (nota: il segno viene sempre calcolato correttamente!)
- Quando vengono coinvolti in operazioni matematiche o booleane, questi valori si comportano nel modo sensato, per es:

$0+ + 5.3 \rightarrow 5.3$	$+INF + +INF \rightarrow +INF$
$+INF == +INF \rightarrow \text{VERO}$	$+INF * 2.0 \rightarrow +INF$
$3.1 / 0+ \rightarrow +INF$	$-3.1 / 0+ \rightarrow -INF$
$3.1 / 0- \rightarrow -INF$	$-3.1 / 0- \rightarrow +INF$
$3.2 / +INF \rightarrow 0+$	$3.2 - 3.2 \rightarrow 0+$
$+INF + 5.3 \rightarrow +INF$	$4.2 + 0- \rightarrow 4.2$
$0+ == 0- \rightarrow \text{VERO (nota!)}$	$300.0 / -INF \rightarrow 0-$
$400.2 < +INF \rightarrow \text{VERO}$	$-INF < +INF \rightarrow \text{VERO}$
$0- < 0+ \rightarrow \text{VERO}$	$0.000001 < 0+ \rightarrow \text{FALSO}$

Architettura degli elaboratori
- 46 -
Aritmetica binaria

46



Configurazioni speciali: NAN (Not-A-Number)

- Esempi di operazioni illecite / indeterminate che producono NAN:
 - $\text{sqrt}(-5.0) \rightarrow \text{NAN}$ (radice quadrata di meno 5)
 - $+INF + -INF \rightarrow \text{NAN}$
 - $0+ / 0+ \rightarrow \text{NAN}$
 - $0+ / 0- \rightarrow \text{NAN}$
 - $+INF - +INF \rightarrow \text{NAN}$
 - $+INF / +INF \rightarrow \text{NAN}$
- Le operazioni numeriche che coinvolgono NAN fanno sempre NAN
 - $\text{NAN} + 5.3 \rightarrow \text{NAN}$
 - $\text{NAN} \times 0 \rightarrow \text{NAN}$
 - $3.1 / \text{NAN} \rightarrow \text{NAN}$
 - $\text{NAN} - \text{NAN} \rightarrow \text{NAN}$

“qualunque espressione che coinvolga un NAN fa NAN”
“NAN si propaga senza fine in tutti i conti”
- Le operazioni booleane ($\rightarrow \{\text{vero}, \text{falso}\}$) che coinvolgono NAN fanno FALSO
 - $\text{NAN} < 5.3 \rightarrow \text{FALSO}$
 - $\text{NAN} \geq 0 \rightarrow \text{FALSO}$
 - $3.1 == \text{NAN} \rightarrow \text{FALSO}$
 - $\text{NAN} == \text{NAN} \rightarrow \text{FALSO}$ (nota: vale solo per $x=\text{NAN}$ che x non sia uguale a sé stesso)

Architettura degli elaboratori
- 47 -
Aritmetica binaria

47



Paragonare numeri in virgola mobile fra loro – note pratiche sull'uso

- Disequazioni: funziona come atteso
 - ▶ maggiore $a > b$
 - ▶ minore $a < b$
 - ▶ maggiore o uguale $a \geq b$
 - ▶ minore o uguale $a \leq b$
- Ugualianze “secche”: $a == b$
 - ▶ non è saggio, non basta paragonare le rappresentazioni bit a bit
 - ▶ due numeri in floating-point possono essere appena diversi, solo come conseguenza di minuscole approssimazioni, ed essere rappresentati da sequenze di bit diverse
 - ▶ E' molto meglio testare se la differenza fra i due sia *QUASI* zero:
 $-\text{epsilon} < (a-b) < \text{epsilon}$, con un *epsilon* piccolo
 - ▶ es:
 $(-1e-10 < a-b) \text{ AND } (a-b < 1e-10)$

Architettura degli elaboratori

- 50 -

Aritmetica binaria

50



Operazioni in virgola mobile nel computing

- Operazioni in virgola mobile sono
 - ▶ più complesse da implementare in HW delle op sui numeri interi
 - ▶ in molti contesti, sono le operazioni più utili e comuni dell'elaborazione
 - ▶ quindi, tipicamente, anche quelle più ottimizzate
- La potenza di calcolo è spesso espressa in **FLOPS**
Floating-point **O**peration per **S**econd
(operazioni in virgola mobile al secondo), e loro multipli, come...
 - ▶ K-FLOPS (kilo-flops): migliaia di op al sec (anni 50)
 - ▶ M-FLOPS (mega-flops): milioni di op al sec (anni 60-70)
 - ▶ G-FLOPS (giga-flops): miliardi di op al sec (anni 80)
 - ▶ T-FLOPS (tera-flops): migliaia di miliardi di op al sec
(Per es: performance di picco di CPU moderne = decine di TeraFlops)
performance di picco di GPU moderne = centinaia di TeraFlops)
 - ▶ P-FLOPS (peta-flops): milioni di miliardi di op al sec (supercomputer)
 - ▶ Exa-flops («*Exascale computing*») (solo i più grandi mainframes di oggi)

Architettura degli elaboratori

- 52 -

Aritmetica binaria

52



Esercizi di riepilogo

- Dire quale numero è codificato, usando lo standard a precisione singola, dal float costituito dalla sequenza di 32 bit: 1111 0010 1101 0000 0000 0000 0000
- Capire quali di questi quattro numeri (nota: sono in forma normale in base 2)
 - (A) $+1.0101001_2 \times 2^{+43}$
 - (B) $+1.0101001_2 \times 2^{-43}$
 - (C) $-1.0101001_2 \times 2^{+43}$
 - (D) $-1.0101001_2 \times 2^{-43}$
- corrisponda a:
 - (1) un numero molto grande, positivo
(come per es un miliardo, o un 1 seguito da molti molti zeri)
 - (2) un numero dal modulo molto grande, ma negativo
(come *meno* un miliardo, o un -1 seguito da molti zeri)
 - (3) un numero molto vicino allo zero, ma positivo
(come un miliardesimo, o $+0.00...00$ *qualcosa*)
 - (4) un numero molto vicino allo zero, ma negativo
(come MENO un miliardesimo o $-0.00...00$ *qualcosa*)
- Esprimi come sequenza di bit il più grande float "non speciale" possibile con la codifica standard a singola precisione (attento a non incorrere in configurazioni speciali).
- Stima quale numero hai codificato, in decimale (approssimando molto)
- Ripeti, questa volta per esprimere il numero negativo maggiore possibile (quindi, il più vicino allo zero)
- Stima quale numero hai codificato, in decimale (approssimando molto)

- 53 -

53



Numeri denormalizzati (o subnormali) [slide opzionale]

- Un'altra configurazione speciale è costituita dai numeri «subnormali»
- Un numero subnormale ... non è in forma normale:
 - Forma normale: $\pm 1.\text{mantissa} \times 2^{\text{esponente}}$
 - Forma subnormale: $\pm 0.\text{mantissa} \times 2^{\text{esponente minimo}}$
- E' codificato usando esponente «tutti zeri», ma la mantissa non tutti zeri (altrimenti è la codifica del ± 0)
- E' un numero che sarebbe troppo vicino allo zero per essere rappresentato in forma normale, perché l'esponente sarebbe minore del minor numero negativo esprimibile in quella codifica
- Per es, usando 32 bit, che prevede 8 bit di esponente con offset a 127 (in cui quindi l'esponente varia quindi da -126 a +127), (nota: l'esponente minimo è quello codificato con 0000 0001, non tutti 0) se vogliamo rappresentare il numero
che in forma normale sarebbe: 1.01100×2^{-127}
usiamo invece la forma
subnormale equivalente 0.10110×2^{-126}

Architettura degli elaboratori

- 54 -

Aritmetica binaria

54