



Corso di ***Architettura degli Elaboratori - A*** ***a.a. 2021/2022***

**Codifica dell'informazione:
Numeri a Virgola Mobile**

Numeri in virgola mobile

In un calcolo *astronomico* è necessario esprimere:

la massa dell'elettrone: 9×10^{-28} grammi

0.000000000000000000000000000009

la massa del sole: 2×10^{33} grammi

20000000000000000000000000000000

Quante cifre occorre usare?

62 cifre: 34 alla sinistra della virgola e 28 a destra

00000000000000000000000000000000.000000000000000000000000000009

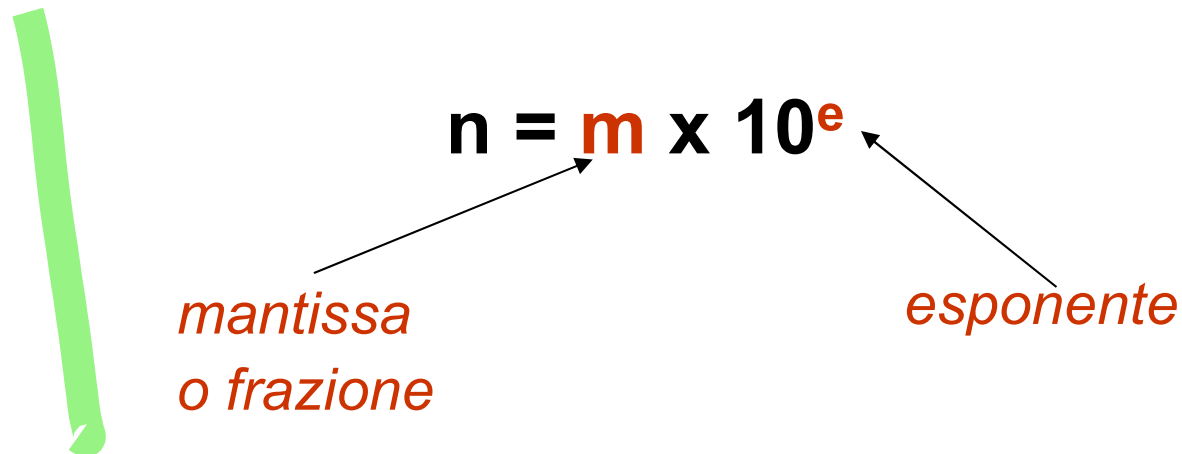
20000000000000000000000000000000.000000000000000000000000000000

Problema: anche se la gamma dei numeri necessari è molto grande, i numeri significativi sono pochi.....

Soluzione: notazione scientifica

Numeri in virgola mobile

- ▶ La notazione scientifica è un tipo di rappresentazione in cui la “**gamma**” dei numeri esprimibili è indipendente dal numero delle cifre significative.



The diagram shows the scientific notation formula $n = m \times 10^e$. A green vertical line is positioned to the left of the formula. An arrow points from the label *mantissa* o *frazione* to the variable m . Another arrow points from the label *esponente* to the superscript e in 10^e .

$$n = m \times 10^e$$

mantissa
o *frazione*

esponente

- la versione informatica di questa notazione è la rappresentazione in **virgola mobile** o **floating point**

Numeri in virgola mobile

Esempi:

$$3,14 = 0,314 \times 10^1 = 3,14 \times 10^0 = 314 \times 10^{-2}$$

$$- 0,0000005 = - 5 \times 10^{-7} = - 0,5 \times 10^{-6}$$

$$127000000 = 127 \times 10^6 = 1,27 \times 10^8$$

La rappresentazione non è unica; esistono convenzioni che permettono di ottenere una rappresentazione unica, ad es. imponendo che la prima cifra significativa della mantissa si trovi immediatamente a destra della virgola; queste forme si dicono **rappresentazioni normalizzate**:

$$3,14 = 0,314 \times 10^1$$

$$- 0,0000005 = - 0,5 \times 10^{-6}$$

$$127000000 = 0,127 \times 10^9$$

Numeri in virgola mobile

Pensando ad una utilizzazione per il calcolatore si possono stabilire ulteriori convenzioni:

- *fissare la lunghezza* della mantissa ad un valore costante
- *limitare l'esponente* ad opportuni intervalli
- utilizzare un *esponente* convenzionale che lo renda *sempre positivo* (notazione in eccesso)
- disporre i tre elementi: <segno, esponente, mantissa> in un ordine stabilito

<i>segno</i>	<i>esponente</i>	<i>mantissa</i>
--------------	------------------	-----------------

Numeri in virgola mobile

Esempio

- lunghezza mantissa: 8 cifre
- valore effettivo esponente: da -50 a +49
- in notazione eccesso 50 l'esponente e' sempre positivo (0 - 99)

i numeri $-0,00000005$ e 127000000 si scrivono nel seguente modo:

	<i>segno</i>	<i>esponente</i>	<i>mantissa</i>
$-0,5 \times 10^{-6}$	-	44	50 000 000
$0,127 \times 10^9$	+	59	12 700 000

Numeri in virgola mobile

Esempio di rappresentazione floating point binaria su 32 bit

- il primo bit rappresenta il segno della mantissa (0 per +, 1 per -)
- 7 bit successivi rappresentano l'esponente (espresso in base 2) in notazione eccesso 64 (esponente effettivo tra -64 e +63) $2^6 = 2^h - 1$
- gli ultimi 24 bit rappresentano la mantissa normalizzata

Esempio:

204,17437

rappresentazione binaria:

11001100,00101100100001110 × 2⁰

rappresentazione normalizzata

0,110011000010110010000111 × 2⁸

• bit di segno:

0

• esponente eccesso 64:

1001000

• mantissa:

5 bit di segno
non scritto

110011000010110010000111

Spostando la virgola a sinistra (dividere per la base) si aumenta di 1 l'esponente (si moltiplica per la base) mantenendo l'uguaglianza

0	1001000	110011000010110010000111
---	---------	--------------------------

Numeri in virgola mobile

- La *gamma (range)* è determinata dal numero di cifre dell'esponente e la *precisione* dal numero di cifre della mantissa.

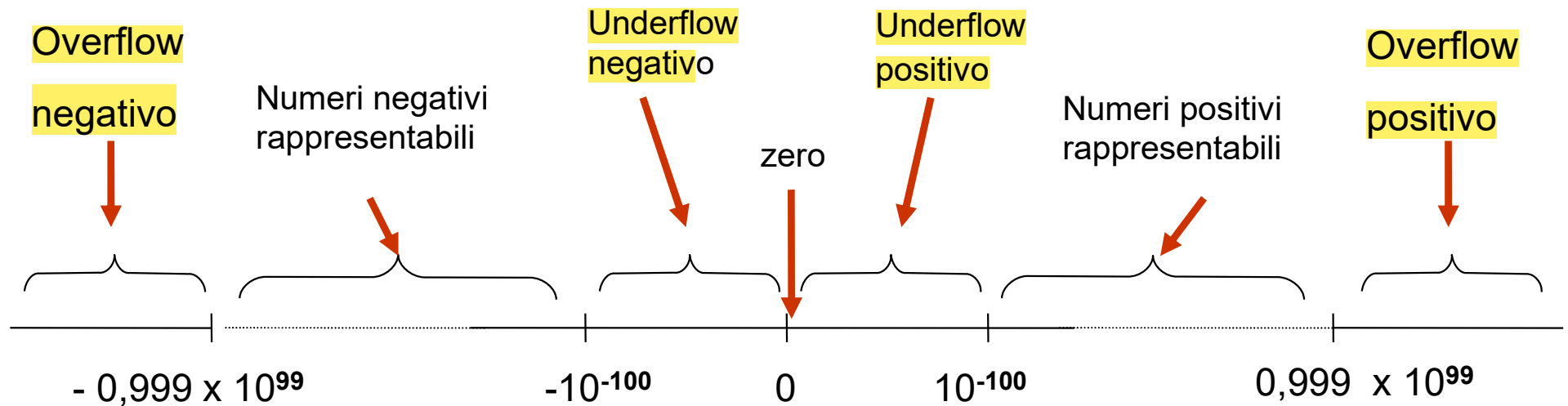
ATTENZIONE!!

- Con i numeri floating-point si può “simulare” il sistema dei numeri reali, pur con grandi differenze:
i numeri reali hanno la potenza del continuo
i numeri floating point sono in numero finito

Numeri in virgola mobile

- Per esempio, consideriamo rappresentazioni (esprese in base 10) con una ***mantissa di tre cifre*** con segno nella ***gamma*** $0,1 \leq |m| < 1$ piu zero ed esponente di due cifre (con segno).
 - minimo numero negativo: $-0,999 \times 10^{99}$
 - massimo numero negativo: $-0,100 \times 10^{-99}$
 - minimo numero positivo: $+0,100 \times 10^{-99}$
 - massimo numero positivo: $+0,999 \times 10^{99}$
- Si rappresentano un numero finito di numeri negativi e numeri positivi, oltre allo zero, che ha tante rappresentazioni.

Numeri in virgola mobile



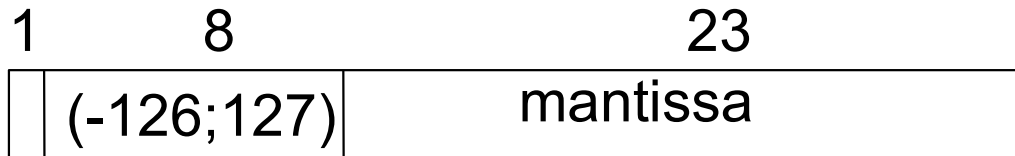
- “Spazio” tra numeri adiacenti non costante
- Arrotondamento
- Il numero di cifre della mantissa determina la densità dei punti, cioè la *precisione* delle approssimazioni
- Il numero di cifre dell'esponente determina la *dimensione degli intervalli* dei numeri rappresentabili

Standard IEEE 754

- Ogni produttore aveva un suo formato floating-point
- Fine anni '70 la **IEEE** costituisce un comitato al fine di *standardizzare* l'aritmetica floating-point
- Diversi Formati:
 - «*half precision*» (16 bit), «*single precision*» (32 bit), «*double precision*» (64 bit), «*quad precision*» (128 bit)
- *Base 2* per mantissa, *notazione in eccesso* per esponente
- Mantissa *normalizzata*: la parte intera è sempre 1, questo bit è nascosto quindi la mantissa si compone della sola parte frazionaria

Standard IEEE 754

Semplice precisione: 32 bit



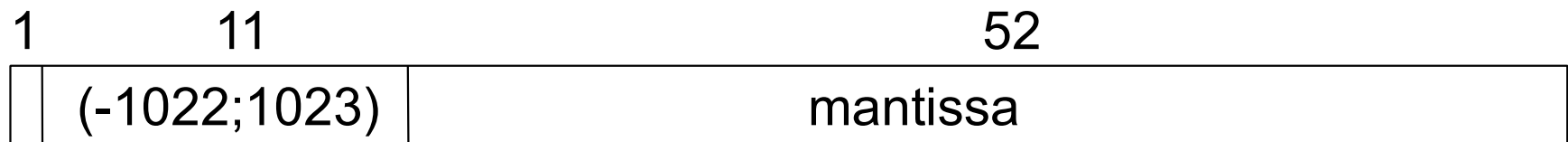
segno

esponente
eccesso 127

Min numero rappresentabile > 0 : 2^{-126} (10^{-38})

Max numero rappresentabile: appr. 2^{128} (10^{38})

Doppia precisione: 64 bit



segno

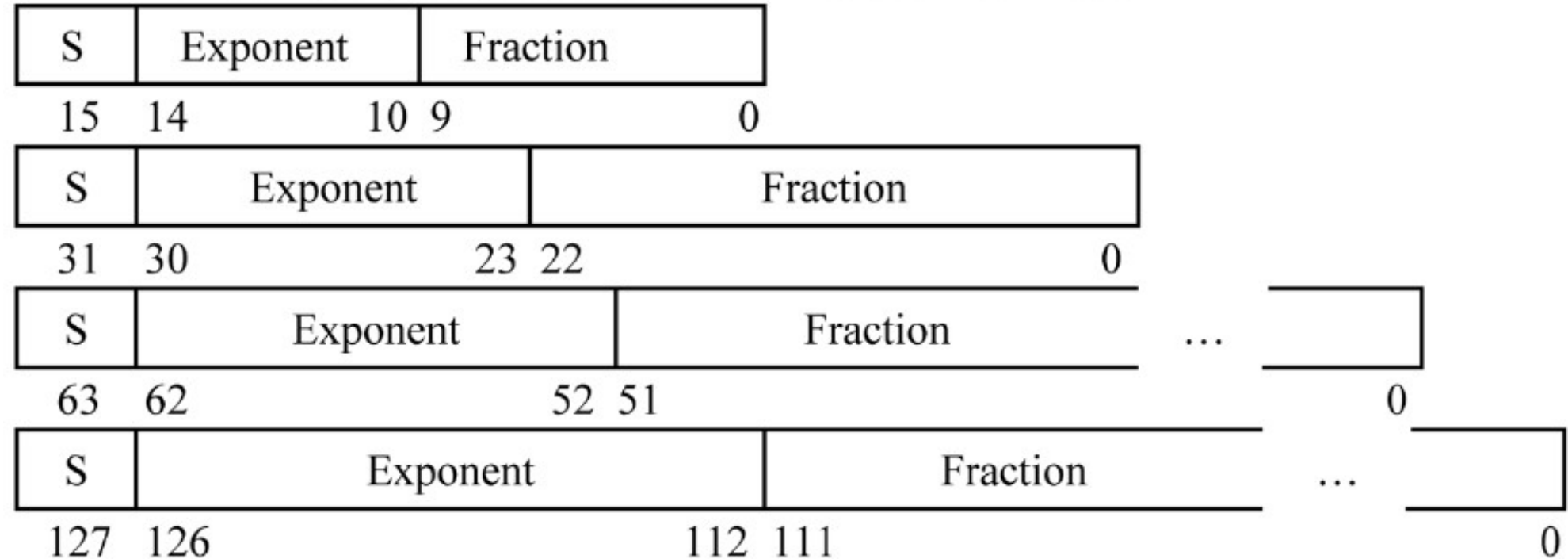
esponente
eccesso 1023

Min numero rappresentabile > 0 : 2^{-1022} (10^{-308})

Max numero rappresentabile: appr. 2^{1024} (10^{308})

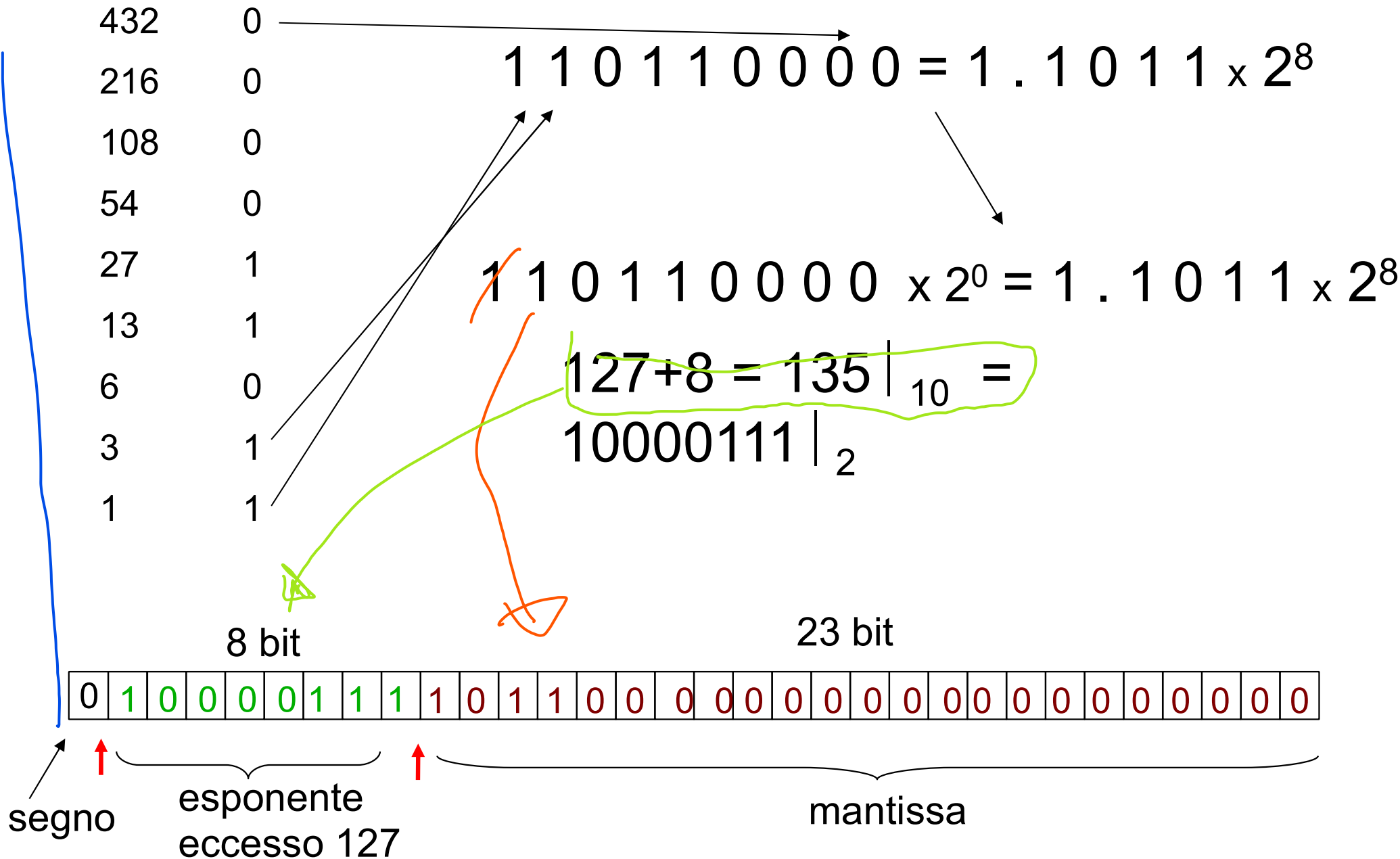
Standard IEEE 754

IEEE Half-, Single-, Double-, and Quad-Precision Formats:



es

Standard IEEE 754



Standard IEEE 754

- Numeri normalizzati e denormalizzati
- Formati speciali per identificare infinito e NaN (Not a Number, esempio se dividiamo infinito per infinito)

	<i>esp</i>	<i>mantissa M</i>	<i>valore v</i>
Numero normalizzato	$0 < esp < 255$	qualunque	$v = (-1)^s(1,M)2^{esp-127}$
Numero denormalizzato	$esp = 0$	$M \neq 0$	$v = (-1)^s(0,M)2^{-126}$
Zero	$esp = 0$	$M = 0$	$v = (-1)^s 0$
Infinito	$esp = 255$	$M = 0$	$v = (-1)^s \infty$
NaN	$esp = 255$	$M \neq 0$	$v = \text{NaN}$

Virgola mobile nel RISC-V

Nome	Esempio	Commenti
32 registri virgola mobile	<code>f0, f1, f2, ..., f31</code>	I registri in virgola mobile del RISC-V possono contenere un numero in virgola mobile in precisione singola o in doppia precisione
2 ⁶¹ parole doppie di memoria	Memoria[0], Memoria[8], ..., Memoria[18 446 744 073 709 551 608]	Si accede solo tramite le istruzioni di trasferimento dati. Il RISC-V utilizza l'indirizzamento al byte, per cui gli indirizzi sequenziali delle parole doppie differiscono di 8. La memoria mantiene le strutture dati, come i vettori, e il contenuto scaricato dai registri

Convenzioni di chiamata

<code>f0-f7</code>	<code>ft0-ft7</code>	FP Temporaries	Caller
<code>f8-f9</code>	<code>fs0-fs1</code>	FP Saved registers	Callee
<code>f10-f11</code>	<code>fa0-fa1</code>	FP Function arguments/Return values	Caller
<code>f12-f17</code>	<code>fa2-fa7</code>	FP Function arguments	Caller
<code>f18-f27</code>	<code>fs2-fs11</code>	FP Saved registers	Callee
<code>f28-f31</code>	<code>ft8-ft11</code>	$R[rd] = R[rs1] + R[rs2]$	Caller

Virgola mobile nel RISC-V

Categoria	Istruzione	Esempio	Significato	Commenti
Aritmetica	Somma VM singola prec.	<code>fadd.s f0,f1,f2</code>	$f0 = f1 + f2$	somma VM (singola precisione)
	Sottrazione VM singola prec.	<code>fsub.s f0,f1,f2</code>	$f0 = f1 - f2$	sottraz VM (singola precisione)
	Moltiplicaz VM singola prec.	<code>fmul.s f0,f1,f2</code>	$f0 = f1 \times f2$	moltiplicaz VM (singola precisione)
	Divisione VM singola prec.	<code>fdiv.s f0,f1,f2</code>	$f0 = f1 / f2$	divisione VM (singola precisione)
	Radice quadrata VM singola prec.	<code>fsqrt.s f0,f1</code>	$f0 = \sqrt{f1}$	Radice quadrata VM (singola precisione)
	Somma VM doppia prec.	<code>fadd.d f0,f1,f2</code>	$f0 = f1 + f2$	somma VM (doppia precisione)
	Sottrazione VM doppia prec.	<code>fsub.d f0,f1,f2</code>	$f0 = f1 - f2$	sottraz VM (doppia precisione)
	Moltiplicaz VM doppia prec.	<code>fmul.d f0,f1,f2</code>	$f0 = f1 \times f2$	moltiplicaz VM (doppia precisione)
	Divisione VM doppia prec.	<code>fdiv.d f0,f1,f2</code>	$f0 = f1 / f2$	divisione VM (doppia precisione)
	Radice quadrata VM doppia prec.	<code>fsqrt.d f0,f1</code>	$f0 = \sqrt{f1}$	radice quadrata VM (doppia precisione)

Virgola mobile nel RISC-V

Categoria	Istruzione	Esempio	Significato	Commenti
Confronto	Uguaglianza VM singola prec.	<code>feq.s x5,f0,f1</code>	<code>x5 = 1 if (f0 == f1), else x5 = 0</code>	Confronto in VM (precisione singola)
	Test di minoranza VM singola prec.	<code>flt.s x5,f0,f1</code>	<code>x5 = 1 if (f0 < f1), else x5 = 0</code>	Confronto in VM (precisione singola)
	Test di minore uguale VM singola prec.	<code>fle.s x5,f0,f1</code>	<code>x5 = 1 if (f0 <= f1), else x5 = 0</code>	Confronto in VM (precisione singola)
	Uguaglianza VM doppia prec.	<code>feq.d x5,f0,f1</code>	<code>x5 = 1 if (f0 == f1), else x5 = 0</code>	Confronto in VM (doppia precisione)
	Test di minoranza VM doppia prec.	<code>flt.d x5,f0,f1</code>	<code>x5 = 1 if (f0 < f1), else x5 = 0</code>	Confronto in VM (doppia precisione)
	Test di minore uguale VM doppia prec.	<code>fle.d x5,f0,f1</code>	<code>x5 = 1 if (f0 <= f1), else x5 = 0</code>	Confronto in VM (doppia precisione)
Trasferimento dati	Lettura di una parola in VM	<code>flw f0,4(x5)</code>	<code>f0 = Memoria[x5+4]</code>	Leggi da memoria dato in singola precisione
	Lettura di una parola doppia in VM	<code>fld f0,8(x5)</code>	<code>f0 = Memoria[x5+8]</code>	Leggi da memoria dato in doppia precisione
	Scrittura di una parola in VM	<code>fsw f0,4(x5)</code>	<code>Memoria[x5+4] = f0</code>	Trasferisci in memoria dato in singola precisione
	Scrittura di una parola doppia in VM	<code>fsd f0,8(x5)</code>	<code>Memoria[x5+8] = f0</code>	Trasferisci in memoria dato in doppia precisione

Ricapitolando

- Una sequenza di N bit da sola non rappresenta nulla: si deve sempre specificare il contesto (la codifica) nella quale deve avvenire la sua interpretazione (decodifica)

- Per cui, cosa rappresenta la sequenza che segue?

11100000101011110101011

- il carattere ® in codifica UTF-8 (è il 5 della lingua Tamil)
- Il colore (224,239,171) in RGB
- 375 µs di voce umana
- il numero 14725035 se in binario puro (divisioni successive per 2)
- il numero -2052181 in complemento a 2 su 24 bit
- Il numero -6336427 in modulo e segno su 24 bit
- continuate voi

Ricapitolando

- Un informatico deve saper:
 - Cos'è la notazione scientifica per un numero
 - Cos'è una rappresentazione normalizzata e perché se ne ha bisogno
 - Essere consapevole dei limiti di rappresentazione rispetto all'insieme dei numeri reali
 - Conoscere i vari formati dello standard IEEE 754
 - Saper codificare un numero reale in uno degli standard IEEE 754
 - Saper decodificare una sequenza di bit in uno degli standard IEEE 754 nel corrispondente numero reale

problem)

Qualche problema

- x **Problema 1:** Ipotezziamo l'esistenza del formato IEEE 754 in precisione **scarsa** su 8 bit con 1 bit di segno, 4 bit di esponente e 3 bit di mantissa (nell'ordine). Esso condivide tutte le caratteristiche dei formati a precisione singola e doppia dello standard IEEE 754, ovvero, la codifica dei numeri normalizzati, denormalizzati, dello zero, dell'infinito, del NaN, con le stesse convenzioni di codifica. Dire:
 - ✓ In eccesso a quale valore sarebbe codificato l'esponente?
 - ✓ Che numero sarebbe rappresentato dalla sequenza 11011100?
- x **Problema 2:** Descrivere la notazione standard IEEE 754 in semplice precisione, specificando i campi che la compongono e descrivendo i passi da eseguire per codificare il numero 13,75.
- x **Problema 3:** Decodificare la sequenza
1 10000011 111000000000000000000000
assumendo che rappresenti un numero in formato IEEE 754 in precisione singola