

## Numeri reali

- Numeri con frazioni
- Possono essere rappresentati anche in binario
  - Es.:  $1001.1010 = 2^3 + 2^0 + 2^{-1} + 2^{-3} = 9.625$
- Quante cifre dopo la virgola?
- Se numero fisso, molto limitato
- Se mobile, dobbiamo saper specificare dove si trova la virgola

## Notazione scientifica (decimale)

- 976.000.000.000.000 viene rappresentato come  $9,76 \times 10^{14}$
- 0,00000000000000976 viene rappresentato come  $9,76 \times 10^{-14}$
- Vantaggio: numeri molto grandi e molto piccoli con poche cifre
- Lo stesso per numeri binari:  $\pm S \times B^{\pm E}$ 
  - S = significando o mantissa (come 976)
  - Si assume la virgola dopo una cifra della mantissa
  - B = base

# Floating Point

Bit di segno	Esponente Polarizzato	Significando o Mantissa
--------------	-----------------------	-------------------------

- Numero rappresentato:  
 $\pm 1.\text{mantissa} \times 2^{\text{esponente}}$
- Esponente polarizzato: un valore fisso viene sottratto per ottenere il vero esponente  
k bit per esponente polarizzato  $\rightarrow 2^{k-1} - 1$   
 $e = e_p - (2^{k-1} - 1)$   
Es.: 8 bit  $\rightarrow$  valori tra 0 e 255  $\rightarrow 2^7 - 1 = 127 \rightarrow$   
esponente da -127 a +128

## Normalizzazione

- I numeri in virgola mobile di solito sono normalizzati
- L'esponente è aggiustato in modo che il bit più significativo della mantissa sia 1
- Dato che è sempre 1 non c'è bisogno di specificarlo
- Numero:  $\pm 1.\text{mantissa} \times 2^{\text{esponente}}$
- L'1 non viene rappresentato nei bit a disposizione  
 $\rightarrow$  se 23 bit per la mantissa, posso rappresentare numeri in  $[1, 2)$
- Se non normalizzato, aggiusto l'esponente
  - Es.:  $0,1 \times 2^0 = 1,0 \times 2^{-1}$

# Esempi



(a) Format

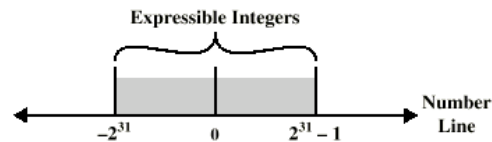
$$\begin{aligned}
 1.1010001 \times 2^{10100} &= 0 \ 10010011 \ 101000100000000000000000 = 1.638125 \times 2^{20} \\
 -1.1010001 \times 2^{10100} &= 1 \ 10010011 \ 101000100000000000000000 = -1.638125 \times 2^{20} \\
 1.1010001 \times 2^{-10100} &= 0 \ 01101011 \ 101000100000000000000000 = 1.638125 \times 2^{-20} \\
 -1.1010001 \times 2^{-10100} &= 1 \ 01101011 \ 101000100000000000000000 = -1.638125 \times 2^{-20}
 \end{aligned}$$

(b) Examples

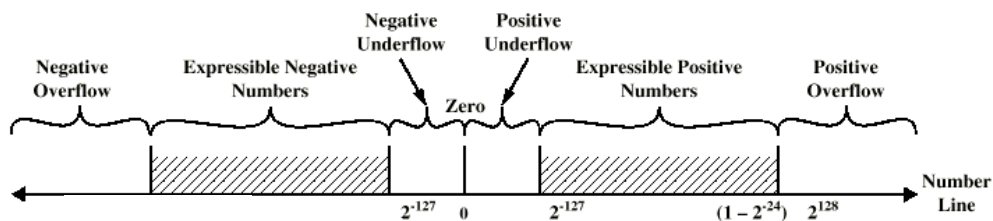
## Numeri rappresentabili (32 bit)

- Complemento a due: da  $-2^{31}$  a  $+2^{31} - 1$
- Virgola mobile (con 8 bit per esponente):
  - Esponente: da -127 (tutti 0) a 128 (tutti 1)
  - Mantissa: 1.0 (tutti 0) a  $2 - 2^{-23}$  (tutti 1, cioè 1.1...1, cioè  $1 + 2^{-1} + 2^{-2} + \dots + 2^{-23} = 2 - 2^{-23}$ )
  - Negativi: Da  $-2^{128} \times (2 - 2^{-23})$  a  $-2^{-127}$
  - Positivi: da  $2^{-127}$  a  $2^{128} \times (2 - 2^{-23})$

## Numeri rappresentabili (32 bit)



(a) Twos Complement Integers

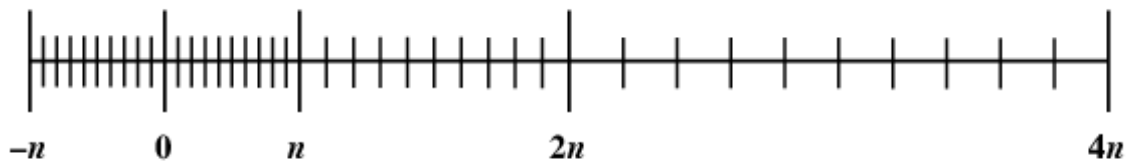


(b) Floating-Point Numbers

## Numeri non rappresentabili (32 bit)

- Negativi minori di  $-2^{128} \times (2 - 2^{-23})$  [overflow negativo]
- Negativi maggiori di  $-2^{-127}$  [underflow negativo]
- Positivi minori di  $2^{-127}$  [underflow positivo]
- Positivi maggiori di  $2^{128} \times (2 - 2^{-23})$  [overflow positivo]
- Non c'è una rappresentazione per lo 0
- I numeri positivi e negativi molto piccoli (valore assoluto minore di  $2^{-127}$ ) possono essere approssimati con lo 0
- Non rappresentiamo più numeri di  $2^{32}$ , ma li abbiamo divisi in modo diverso tra positivi e negativi
- I numeri rappresentati non sono equidistanti tra loro: più densi vicino allo 0 (errori di arrotondamento)

## Densità dei numeri in virgola mobile



## Precisione e densità

- Nell'esempio, 8 bit per esponente e 23 bit per mantissa
- Se più bit per l'esponente (e meno per la mantissa), espandiamo l'intervallo rappresentabile, ma i numeri sono più distanti tra loro → minore precisione
- La precisione aumenta solo aumentando il numero dei bit
- Di solito, precisione singola (32 bit) o doppia (64 bit)

# Densità

- Numero (positivo) =  $1, m \times 2^e$
- Fissato  $e$ , i numeri rappresentabili sono tra  $2^e$  (mantissa tutta a 0) e  $2^e \times (2 - 2^{-23})$ 
  - Quanti numeri?  $2^{23}$
- Consideriamo adesso  $e+1 \rightarrow$  i numeri rappresentabili sono tra  $2 \times 2^e$  e  $2 \times 2^e \times (2 - 2^{-23})$ 
  - Intervallo grande il doppio
  - Quanti numeri? Sempre  $2^{23}$

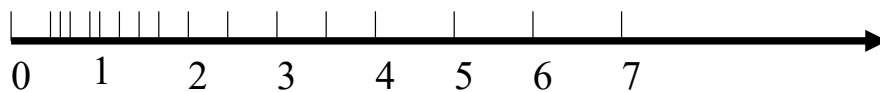
## Esempio con 4 bit (+ segno)

$f, e$  (due bit per esponente),  $a, b$  (due bit per mantissa)

$$\text{Numero} = 2^{f \times 2 + e - 1} \times (1 + a \times 2^{-1} + b \times 2^{-2})$$

f	e	a	b	numero
0	0	0	0	0.5
0	0	0	1	0.625
0	0	1	0	0.75
0	0	1	1	0.875
0	1	0	0	1
0	1	0	1	1.25
0	1	1	0	1.5
0	1	1	1	1.75
1	0	0	0	2
1	0	0	1	2.5
1	0	1	0	3
1	0	0	1	3.5
1	1	0	0	4
1	1	0	1	5
1	1	1	0	6
1	1	1	1	7

- Numeri positivi rappresentati con 2 bit esponente e 2 bit mantissa



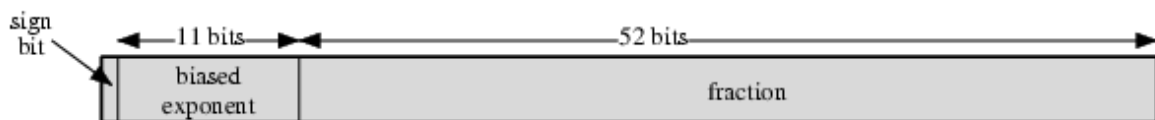
## Standard IEEE 754

- Standard per numeri in virgola mobile
- Formato singolo a 32 bit e doppio a 64 bit
- Esponente con 8 e 11 bit
- 1 implicito a sinistra della virgola
- Formati estesi (più bit per mantissa ed esponente) per risultati intermedi
  - Più precisi → minore possibilità di risultato finale con eccessivo arrotondamento

# Formati IEEE 754



(a) Single format



(b) Double format

## Numeri rappresentati (formato singolo)

- Alcune combinazioni (es.: valori estremi dell'esponente) sono interpretate in modo speciale
- Esponente polarizzato da 1 a 254 (cioè esponente da -126 a +127): numeri normalizzati non nulli in virgola mobile  $\rightarrow \pm 2^{e-127} \times 1.f$
- Esponente 0, mantissa (frazione) 0: rappresenta 0 positivo e negativo
- Esponente con tutti 1, mantissa 0: infinito positivo e negativo
  - Overflow può essere errore o dare il valore infinito come risultato
- Esponente 0, mantissa non nulla: numero denormalizzato
  - Bit a sinistra della virgola: 0, vero esponente: -126
  - Positivo o negativo
  - numero:  $2^{-126} \times 0.f$
- Esponente tutti 1, mantissa non nulla: errore (Not A Number)





## Aritmetica in virgola mobile

- Allineare gli operandi aggiustando gli esponenti (per somma e sottrazione)
- Possibili eccezioni del risultato:
  - Overflow dell'esponente: esponente positivo che è più grande del massimo
  - Underflow dell'esponente: esponente negativo minore del minimo valore (numero troppo piccolo)
  - Underflow della mantissa: mantissa 0 (allineando, gli 1 sono usciti fuori)
  - Overflow della mantissa: riporto del bit più significativo



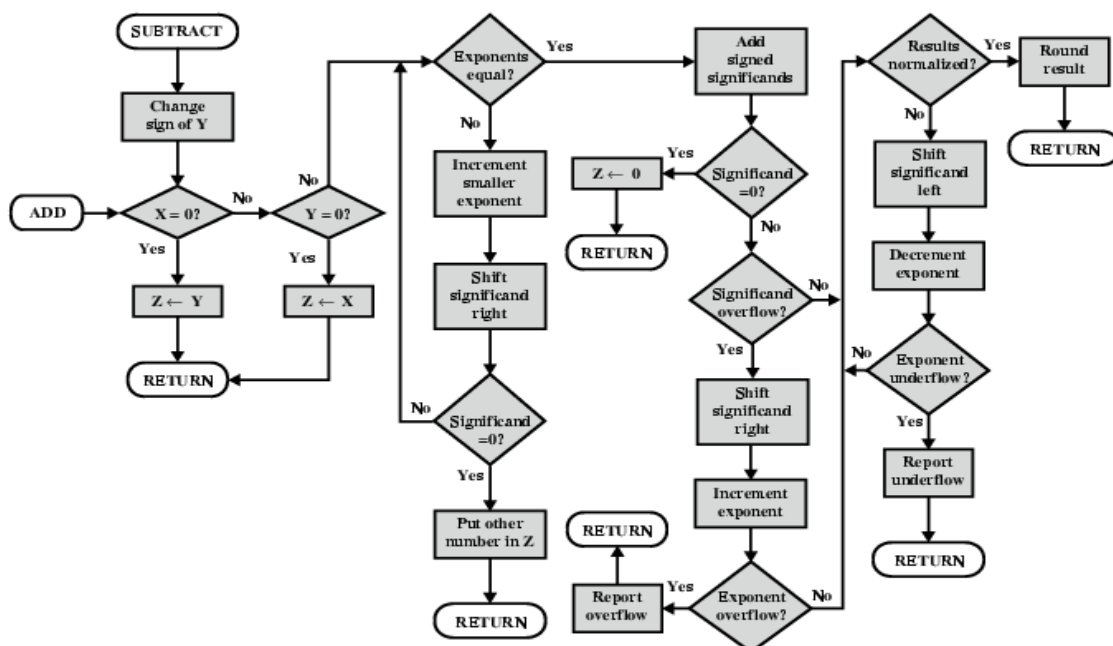
## Somma e sottrazione

- Quattro fasi:
  - Controllo dello zero
    - Se uno dei due è 0, il risultato è l'altro numero
  - Allineamento delle mantisse
    - Rendere uguali gli esponenti
  - Somma o sottrazione delle mantisse
  - Normalizzazione del risultato
    - Traslare a sinistra finché la cifra più significativa è diversa da 0

# Allineamento delle mantisse

- Esempio (in base 10):
- $(123 \times 10^0) + (456 \times 10^{-2})$
- $123 + 4,56 \rightarrow$  Non possiamo semplicemente sommare 123 a 456: il 4 deve essere allineato sotto il 3
- Nuova rappresentazione:  $(123 \times 10^0) + (4,56 \times 10^0)$
- Adesso posso sommare le mantisse  
( $123 + 4,56 = 127,56$ )
- Risultato:  $127,56 \times 10^0$

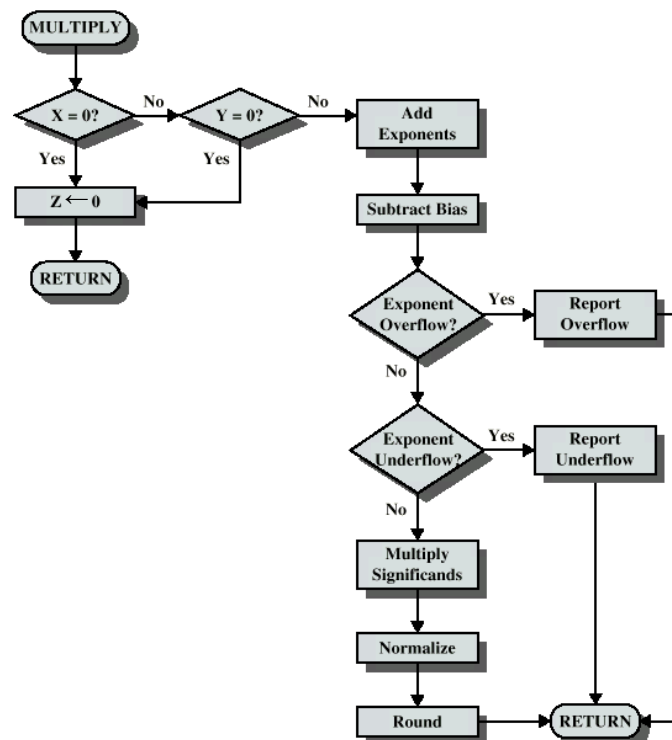
## Somma e sottrazione in virgola mobile



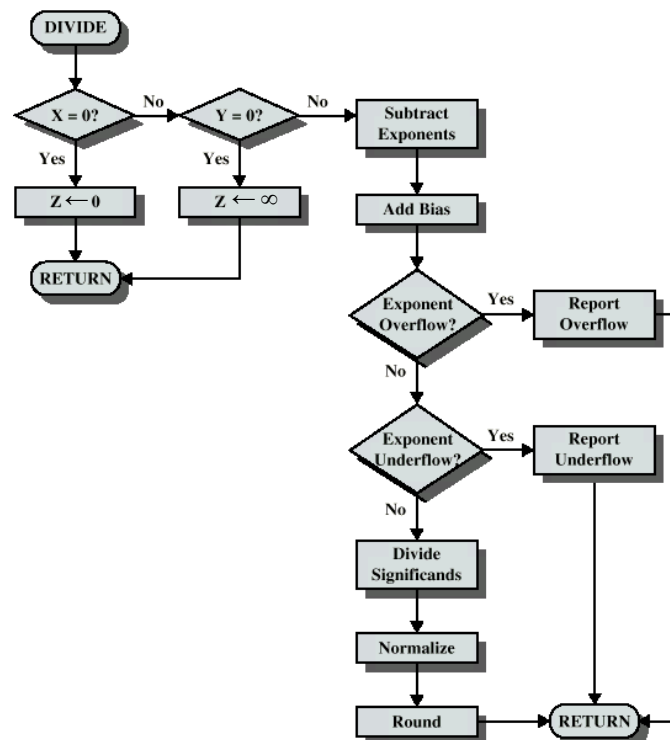
# Moltiplicazione e divisione

- Controllo dello zero
- Somma degli esponenti
- Sottrazione polarizzazione
- Moltiplicazione/divisione operandi
- Normalizzazione
- Arrotondamento

## Moltiplicazione in virgola mobile



# Divisione in virgola mobile



## Precisione del risultato: bit di guardia

- Di solito operandi nei registri della ALU, che hanno più bit di quelli necessari per la mantissa +1 → i bit più a destra sono messi a 0 e permettono di non perdere bit se i numeri vengono shiftati a destra
- Es.:  $X - Y$ , con  $Y = 1,11...11 \times 2^0$  e  $X = 1,00...00 \times 2^1$
- $Y$  va shiftato a destra di un bit, cioè diventa  $0,111...11 \times 2^1$  → un 1 viene perso senza i bit di guardia
- Risultato:
  - Senza bit di guardia:  
 $(1,0...0 - 0,1...1) \times 2 = 0,0...01 \times 2 = 1,0...0 \times 2^{-22}$
  - Con bit di guardia:  
 $(1,0...0 - 0,1...11) \times 2 = 0,0...001 \times 2 = 1,0...0 \times 2^{-23}$



## Precisione del risultato: arrotondamento

- Se il risultato è in un registro più lungo, quando lo si riporta nel formato in virgola mobile, bisogna arrotondarlo
- Quattro approcci:
  - Arrotondamento al più vicino (default)
    - Bit aggiuntivi che iniziano con 1 → sommo 1
      - Bit aggiuntivi 10...0 → sommo 1 se l'ultimo bit è 1, altrimenti 0
    - Bit aggiuntivi che iniziano con 0 → elimino
  - Arrotondamento (per eccesso) a  $+\infty$  e arrotondamento (per difetto) a  $-\infty$ 
    - Usati nell'aritmetica degli intervalli
  - Arrotondamento a 0 (cioè troncamento dei bit in più)