

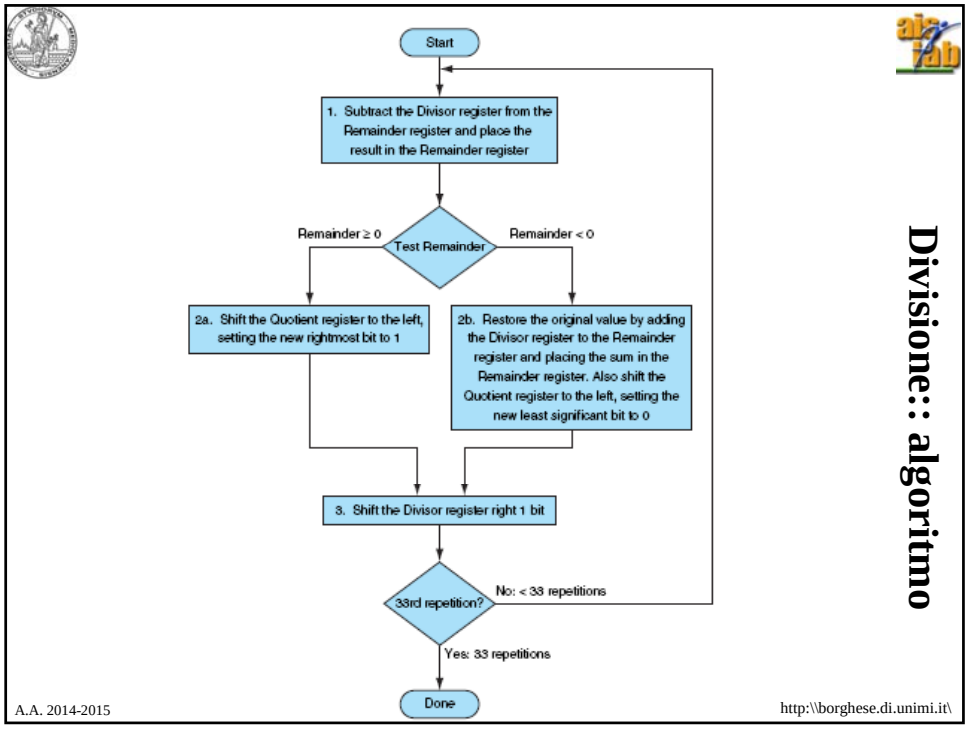

Firmware Division, UC
&
Floating pointer adder

Prof. Alberto Borghese
Dipartimento di Scienze dell'Informazione
borghese@di.unimi.it

Università degli Studi di Milano
Riferimenti sul Patterson, 5a Ed.: 3.4, 3.5


Sommario

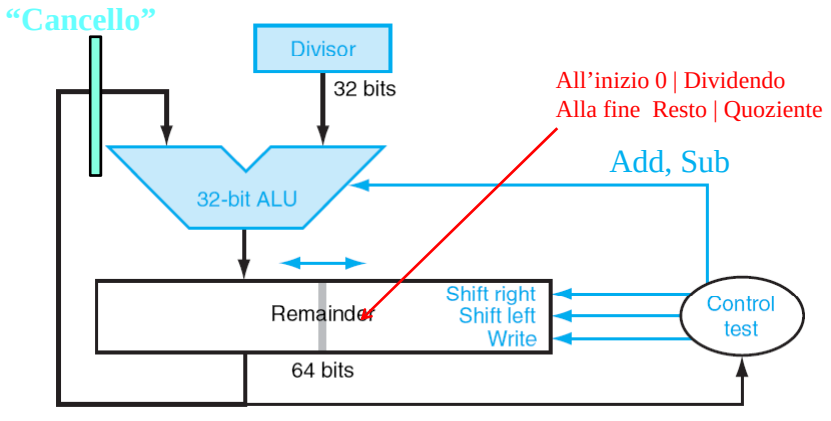
- UC Divisione intera
- Somma in virgola mobile



Divisione:: algoritmo

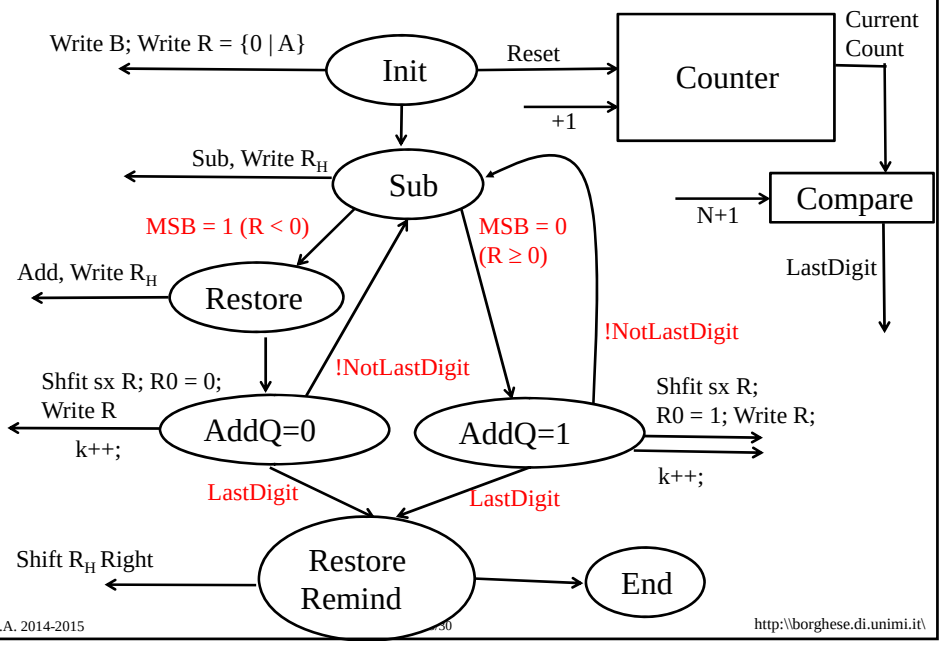
Il circuito firmware ottimizzato della divisione

Inizializzazione: Resto = 0 | Dividendo



In questo circuito ottimizzato il resto viene shiftato a sinistra una volta di troppo. Occorre alla fine spostare il Resto (solo il resto, i 32 bit più significativi del registro R) a destra.

STG dell'Unità di Controllo per la divisione



Definizione della struttura della FSM

<S0, S, I, Y, f(S,I), g(S)>

S = {Init, Sub, Restore, AddQ=1, AddQ=0, Restore R_H, End}
S0 = Init

I = {MSB, LastDigit}

Y = {Write B, Write R = {0 | A}, Sub, Add, Write R_H, R0 = 0; R0 = 1; k++; Shift R left, Shift R_H right}

f(S,I) = ?

g(S) = ?

Definizione del funzionamento della FSM: la STT

S \ I	MSB=0 LastDigit	MSB = 1 LastDigit	MSB=0 !LastDigit	MSB = 1 !LastDigit
Init	Sub	Sub	Sub	Sub
Sub	AddQ=1	Restore	AddQ=1	Restore
Restore	AddQ=0	AddQ=0	AddQ=0	AddQ=0
Add Q=0	Restore R_H	Restore R_H	Sub	Sub
Add Q=1	Restore R_H	Restore R_H	Sub	Sub
Restore R_H	End	End	End	End
End	End	End	End	End

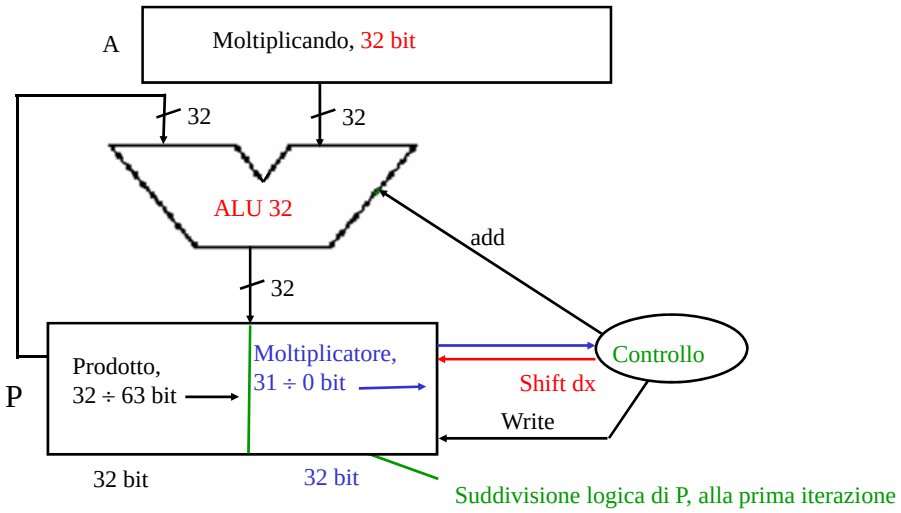
Next State $S_{t+1} = f(S_t, I_t)$

Definizione del funzionamento della FSM: la STT

S \ I	Write B	Write R={0 | A}	Reset count	Sub	Add	Shift R left	Write R0 =0	Write R0=1	Write R_H	Shift R_H right	k++
Init	1	1	1	0	0	0	0	0	0	0	0
Sub	0	0	0	1	0	0	0	0	1	0	0
Restore	0	0	0	0	1	0	0	0	1	0	0
Add Q=0	0	0	0	0	0	1	1	0	0	0	1
Add Q=1	0	0	0	0	0	1	0	1	0	0	1
Restore R_H	0	0	0	0	0	0	0	0	0	1	1
End	0	0	0	0	0	0	0	0	0	0	0

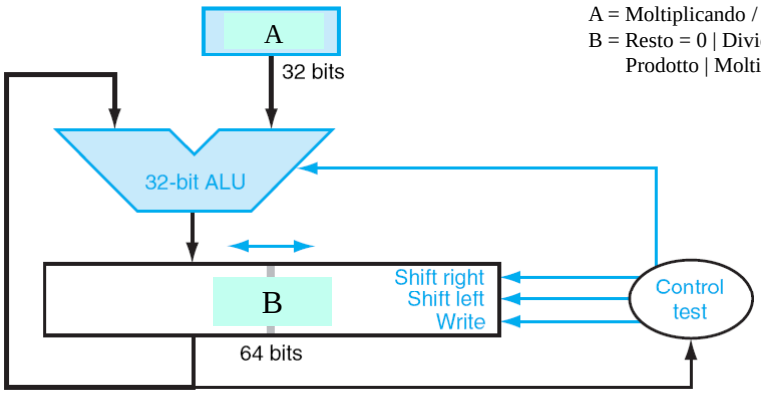
Uscita Y=g(S)

Circuito ottimizzato della moltiplicazione



Un unico circuito per moltiplicazione e divisione

Inizializzazione:
A = Moltiplicando / Divisore
B = Resto = 0 | Dividendo / Prodotto | Moltiplicatore



Definizione della struttura della FSM

<S0, S, I, Y, f(S,I), g(S)>

S = {Init, Sub, Restore, AddQ=1, AddQ=0, Restore R_H, End}
S0 = Init

I = {MSB, LastDigit}

Y = {Write B, Write R = {0 | A}, Sub, Add, Write R_H, R0 = 0; R0 = 1; k++; Shift R left, Shift R_H right}

f(S,I) = ?

g(S) = ?

Unità di controllo per divisione / moltiplicazione

<S0, S, I, Y, f(S,I), g(S)>

S = {Init, Sub, Restore, AddQ=1, AddQ=0, Restore R_H, **Shift dx**, End}
S0 = Init

I = {MSB, LastDigit, **Multiply/Divide**}

Y = {Write B, **Write A**, Write R = {0 | A}, **Write R = {0 | B}**, Sub, Add, **None**, Write R_H, R0 = 0; R0 = 1; Shift R left, Shift R_H right, k++}

f(S,I) = ?

g(S) = ?

Sommario

- UC Divisione intera
- Somma in virgola mobile

Codifica in virgola mobile
Standard IEEE 754 (1980)

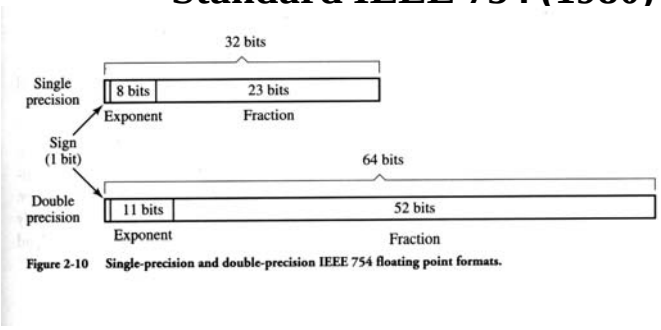


Figure 2-10 **Single-precision and double-precision IEEE 754 floating point formats.**

Rappresentazione polarizzata dell'esponente:

Polarizzazione pari a 127 per singola precisione =>
1 viene codificato come 1000 0000.

Polarizzazione pari a 1023 in doppia precisione.
1 viene codificato come 1000 0000 000.

Esempio di somma in virgola mobile

a = 9,999 x 10^{1} b = 1,61 x 10^{-1} a + b = ?

NB I numeri decimali sono normalizzati.

Una possibilità è:

```
 99,99   +
  0,161 =
---------

100,151
```

100,151 → 1,0015 x 10^{2} in forma normalizzata

Algoritmo di somma in virgola mobile - I

1) Trasformare i due numeri in modo che le due rappresentazioni abbiano la stessa base: allineamento della virgola. Quale si allinea?

2) Effettuare la somma delle mantisse.

Se il numero risultante è normalizzato termino qui. Altrimenti:

3) Normalizzare il risultato.

Algoritmo di somma in virgola mobile - II

1) Trasformare **uno dei due numeri** in modo che le due rappresentazioni abbiano la stessa base: allineamento della virgola. Si allinea all'esponente più alto (denormalizzo il numero più piccolo).

2) Effettuare la somma delle mantisse.

Se il numero risultante è normalizzato termino qui. Altrimenti:

3) Normalizzare il risultato.

Esempio di somma in virgola mobile - II

a = $9{,}999 \times 10^{1}$ b = $1{,}61 \times 10^{-1}$ a + b = ?

Supponiamo di avere a disposizione 4 cifre per la mantissa e due per l'esponente.

Devo trasformare uno dei numeri, una possibilità è:

$9{,}999 \times 10^{1}$ +
$0{,}016 \times 10^{1}$ = Perdo un bit perchè non rientra nella capacità della mantissa
---
$10{,}015 \times 10^{1}$ Il risultato non è più normalizzato, anche se i due addendi sono normalizzati.

Normalizzazione per ottenere il risultato finale:

$10{,}015 \times 10^{1} = 1{,}001 \times 10^{2}$ in forma normalizzata.

NB: In questa fase si può generare la necessità di rinormalizzare il numero (passo 3):
Esempio: $9{,}99999 \times 10^{2}$ => Arrotondo alla cifra più vicina => $10{,}00 \times 10^{3}$
=> Normalizzazione => $1{,}0 \times 10^{4}$

Approssimazione

Interi -> risultato esatto (o overflow)

Numeri decimali -> Spesso occorrono delle approssimazioni

- Troncamento (floor): $1{,}001 \times 10^2$ (cf. Slide precedente)
- Arrotondamento alla cifra superiore (ceil): $1{,}002 \times 10^2$
- Arrotondamento alla cifra più vicina: $1{,}002 \times 10^2$

IEEE754 prevede 2 bit aggiuntivi nei calcoli per mantenere l'accuratezza.
bit di guardia (guard)
bit di arrotondamento (round)

Esempio: aritmetica in floating point accurata

a = 2,34 b = 0,0256 a + b = ? Codifica su 3 cifre decimali totali.
Approssimazione mediante arrotondamento.

Senza cifre di arrotondamento devo scrivere:

2,34 +
0,02 =

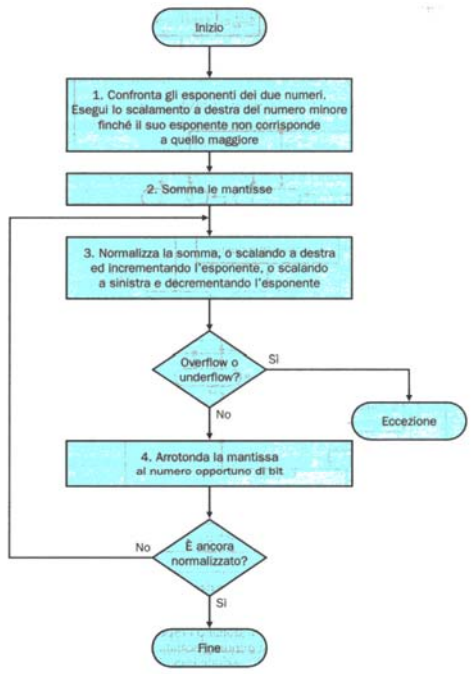
2,36

Con il bit di guardia e di arrotondamento posso scrivere:

2,3400 +
0,0256 =

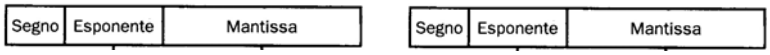
2,3656

L'arrotondamento finale fornisce per rintrare in 3 cifre decimali fornisce: **2,37**



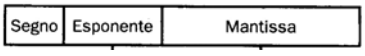
Algoritmo risultante

Il circuito della somma floating point: gli attori



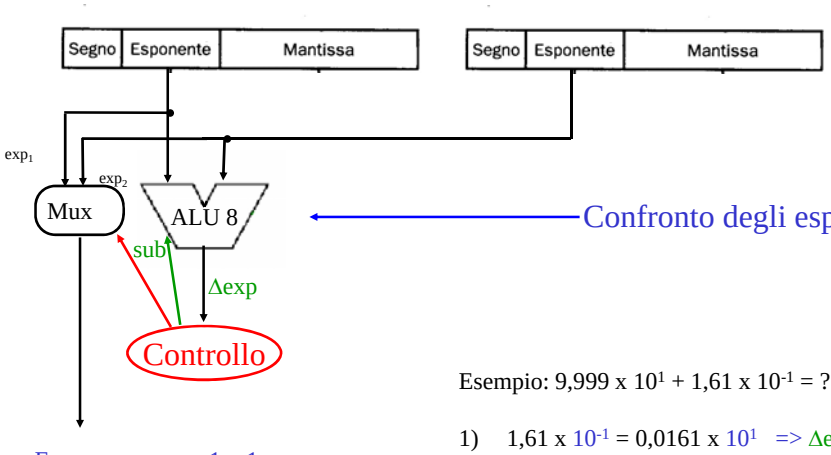
A + B

C



Rappresentazione normalizzata IEEE754

Il circuito della somma floating point: determinazione dell'esponente comune

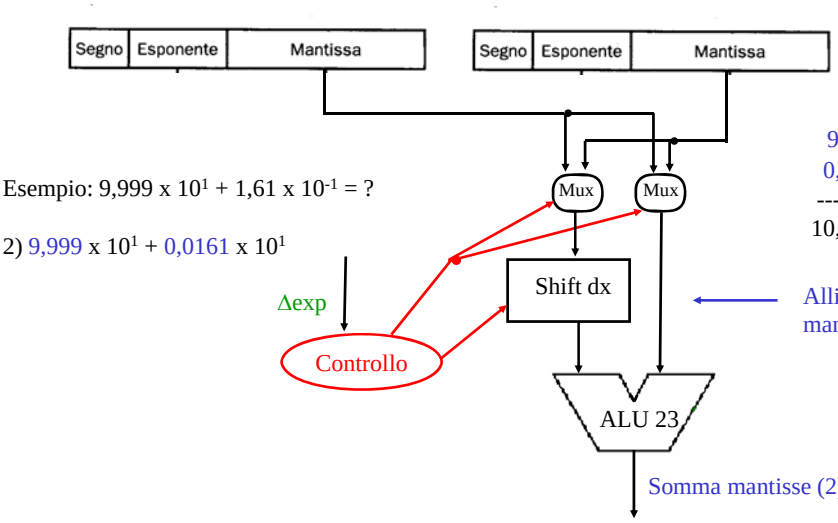


Esempio: 9,999 x 10^{1} + 1,61 x 10^{-1} = ?

1) 1,61 x 10^{-1} = 0,0161 x 10^{1} => Δexp = +2

Exp somma = exp1 = 1

Il circuito della somma floating point: allineamento delle mantisse e somma



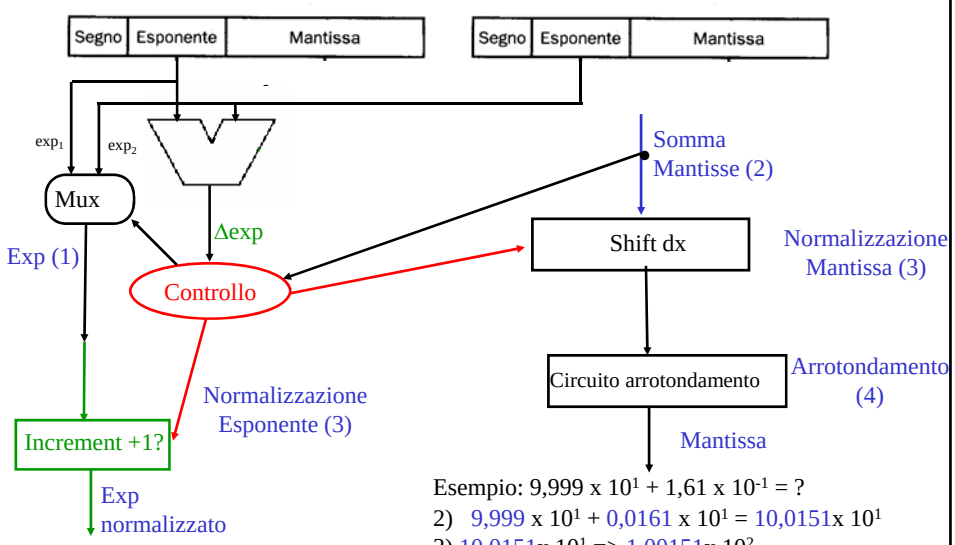
Esempio: 9,999 x 10^{1} + 1,61 x 10^{-1} = ?

2) 9,999 x 10^{1} + 0,0161 x 10^{1}

9,999 +
0,0161 =

10,0151

Il circuito della somma floating point: arrotondamento e normalizzazione



Esempio: $9,999 \times 10^1 + 1,61 \times 10^{-1} = ?$

2) $9,999 \times 10^1 + 0,0161 \times 10^1 = 10,0151 \times 10^1$

3) $10,0151 \times 10^1 \Rightarrow 1,00151 \times 10^2$

4) $1,00151 \times 10^2 \Rightarrow 1,001 \times 10^2$

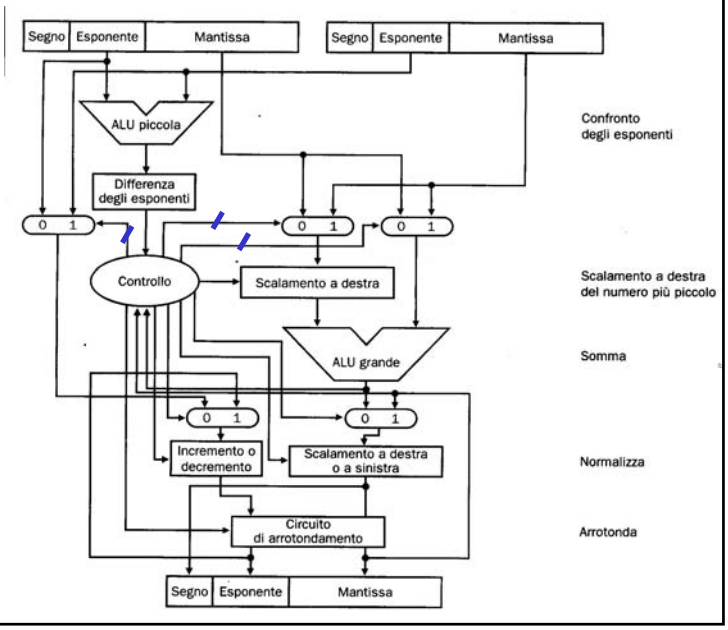
Circuito della somma floating point

Le tre linee in blu contengono lo stesso segnale di controllo.

Gestisce anche la rinormalizzazione: $9,99999 \times 10^2 = 10,00 \times 10^3$

Gestisce anche i numeri negativi.

Problemi?



Esempio

P = 0,5 – 0,4375 somma binaria con precisione di 4 bit.

$A = 1{,}000 \times 2^{-1}$
$B = -1{,}110 \times 2^{-2}$ Espressione in forma normalizzata.

1) Allineamento di A e B. Trasformo B: $-1{,}110 \times 2^{-2} = -0{,}1110 \times 2^{-1}$
2) Somma delle mantisse: $1{,}000 + (-0{,}111) = 0{,}001 \times 2^{-1}$
3) Normalizzazione della somma: $0{,}001 \times 2^{-1} = 1{,}000 \times 2^{-4}$
   Controllo di overflow o underflow: $-126 \leq -4 \leq 127$.
4) Arrotondamento della somma: 1,000 non lo richiede.

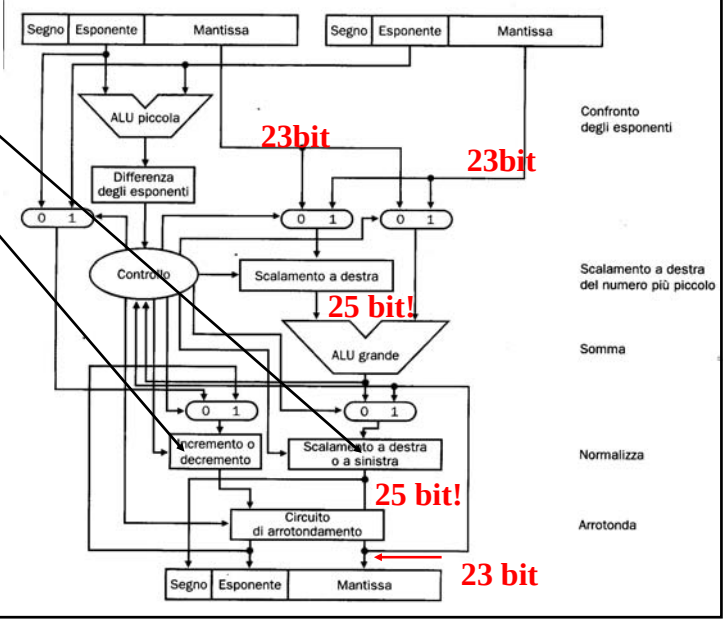
$1{,}000 \times 2^{-4} = 1/2^4 = 1/16 = 0{,}0625$ c.v.d.

Circuito della somma floating point con bit di arrotondamento

In quale caso la mantissa viene scalata a sx?

In quale caso l'esponente viene decrementato?

La rappresentazione interna, secondo IEEE 754, prevede 2 bit aggiuntivi: bit di guardia e bit di arrotondamento.



Prodotto e divisione in virgola mobile

- Prodotto delle mantisse
- Somma degli esponenti
- Normalizzazione

- Divisione in virgola mobile = Prodotto di un numero per il suo inverso.

Sommario

- UC Divisione intera
- Somma in virgola mobile