


UC firmware moltiplicazione
Floating pointer adder

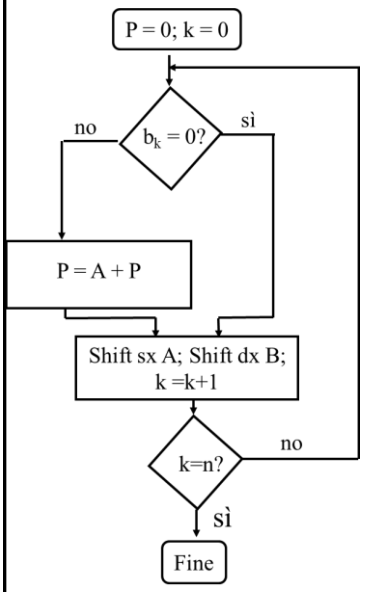
Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

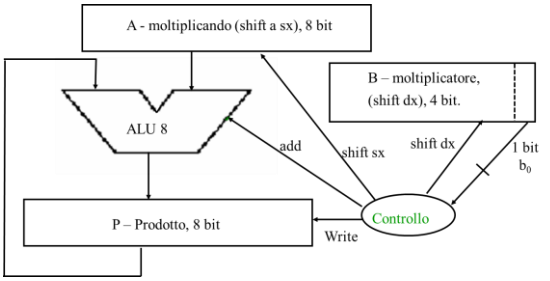
Università degli Studi di Milano
Riferimenti sul Patterson, 6a Ed.: 3.4, 3.5, 4.2

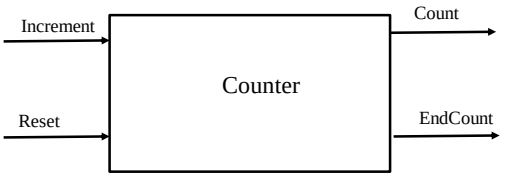

Sommario

- Unità di controllo del firmware
- Somma in virgola mobile

Moltiplicatore firmware







Unità di controllo - STG

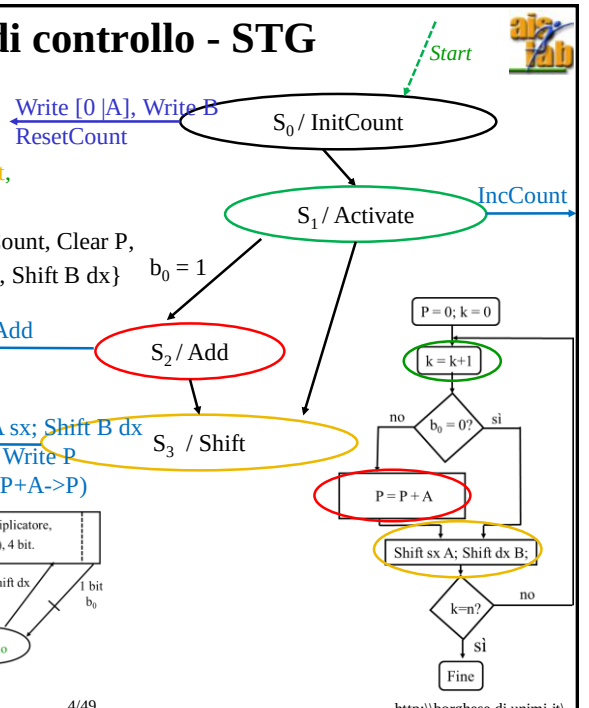
Macchina a stati finiti:

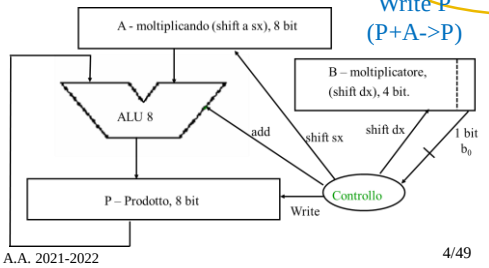
{X} = {initCount, Activate, Add, Shift,

{I} = {b_0,

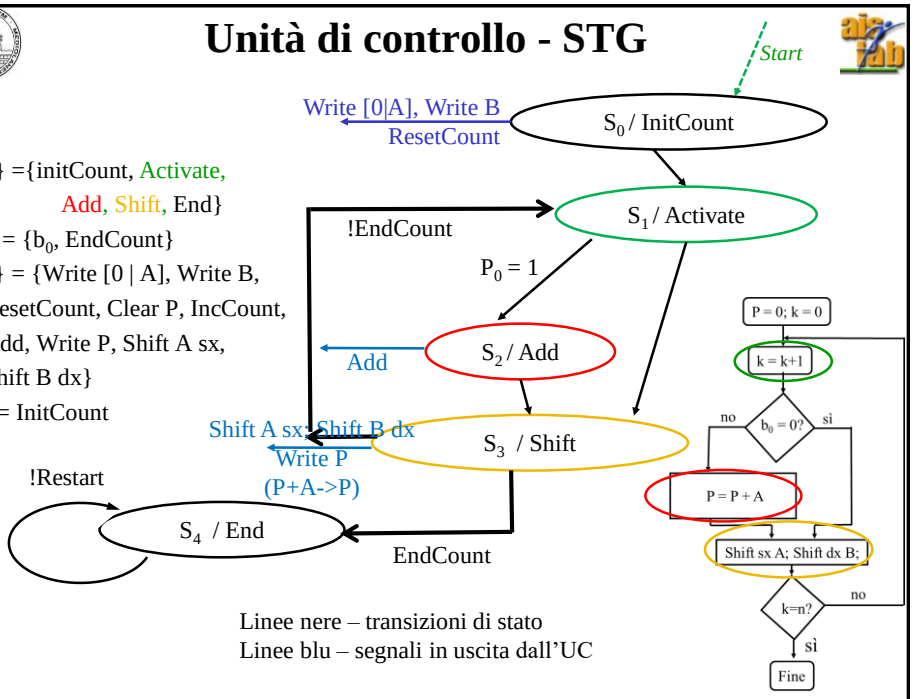
{Y} = {Write [0 | A], Write B, ResetCount, Clear P,
IncCount, Add, Write P, Shift A sx, Shift B dx}

X_0 – Stato iniziale = InitCount





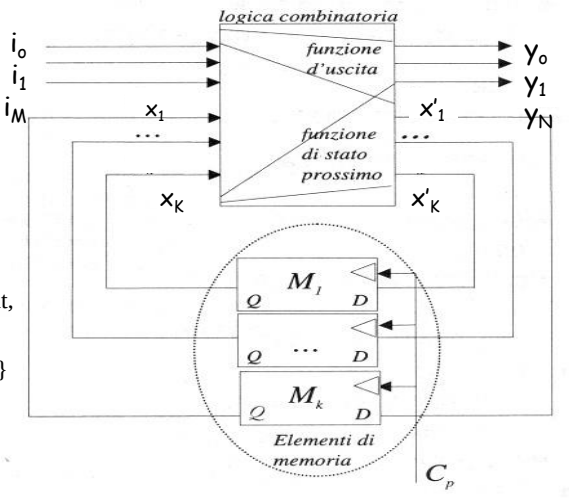
Unità di controllo - STG



{X} ={initCount, Activate,
Add, Shift, End}
{I} = {b_0, EndCount}
{Y} = {Write [0 | A], Write B,
ResetCount, Clear P, IncCount,
Add, Write P, Shift A sx,
Shift B dx}
X_0 = InitCount

Linee nere – transizioni di stato
Linee blu – segnali in uscita dall'UC

Dimensionamento

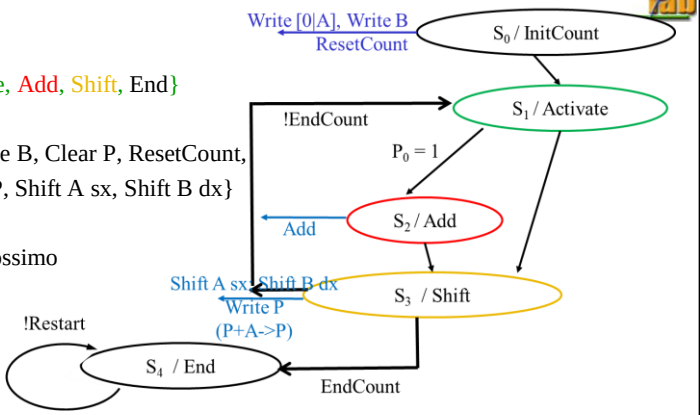


{X} ={initCount, Activate,
Add, Shift, End}
{I} = {b_0, EndCount}
{Y} = {Write [0 | A], Write B,
ResetCount, Clear P, IncCount,
Add, Write P,
Shift A sx, Shift B dx}
X_0 = InitCount

M = 2 - 2 input binary (yes/no)
N = 8 - 8 uscite binarie (yes/no)
K = 3 - 5 stati

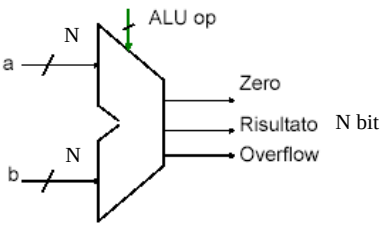
Unità di controllo - STT

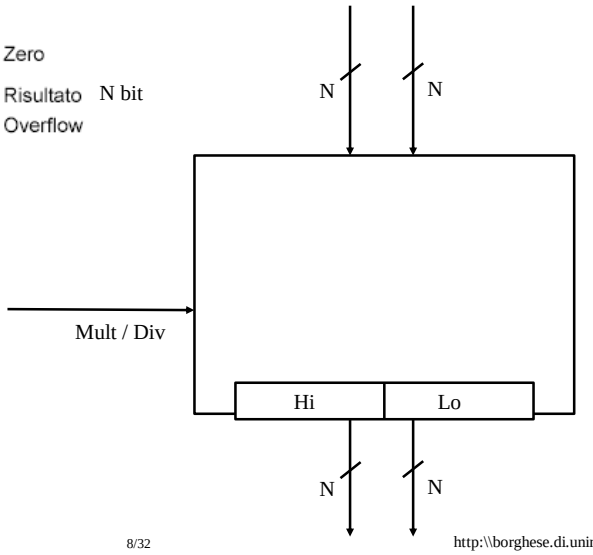
{X} = {initCount, Activate, Add, Shift, End}
{I} = {b_0, EndCount}
{Y} = {Write [0 | A], Write B, Clear P, ResetCount, IncCount, Add, Write P, Shift A sx, Shift B dx}
X_0 = InitCount
f(X,I) – Funzione stato prossimo
g(X) – Funzione di uscita



	!EndCount $b_0 = 0$	EndCount $b_0 = 0$	!EndCount $b_0 = 1$	EndCount $b_0 = 1$	Uscita
InitCount	Activate	Activate	Activate	Activate	Clear P, Write A, Write B, Reset counter, Clear P
Activate	Shift	Shift	Add	Add	Inc counter
Add	Shift	Shift	Shift	Shift	Add
Shift	Activate	End	Activate	End	Shift A sx, Shift B dx, Write P
End	End	End	End	End	

Circuiti operazioni tra numeri interi





E per i numeri decimali?

Sommario

- Unità di controllo del firmware
- Somma in virgola mobile

Codifica in virgola mobile Standard IEEE 754 (1980)

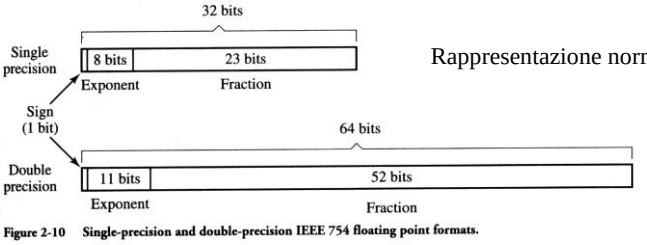


Figure 2-10 **Single-precision and double-precision IEEE 754 floating point formats.**

Rappresentazione normalizzata = 1,…. x 2^N

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 = $7{,}999 \times 10^{1}$ b = $1{,}61 \times 10^{-1}$ a + b = ?

NB I numeri decimali sono normalizzati -> vanno riportati alla stessa base (incolonnati correttamente):

Una possibilità è:

79,99 + a = $7{,}999 \times 10^{1} = 79{,}99 \times 10^{0}$
0,161 = b = $1{,}61 \times 10^{-1} = 0{,}161 \times 10^{0}$

$80{,}151 \times 10^{0}$ = 80,151 = $8{,}0151 \times 10^{1}$ in forma normalizzata

Altre possibilità sono:

799,9 +
1,61 =

$801{,}51 \times 10^{-1} = 8{,}0151 \times 10^{1}$
in forma normalizzata =

7,999 +
0,0161 =

$8{,}0151 \times 10^{1}$

Quale forma conviene utilizzare?

a = $7{,}999 \times 10^{1}$ b = $1{,}61 \times 10^{-1}$ a + b = ?

Supponiamo di avere 4 cifre in tutto per il risultato del prodotto: 1 per la parte intera e 3 per la parte decimale:

79,99 +
0,161 =

~~8~~0,151 x 10^{0}

799,9 +
1,61 =

~~80~~1,51 x 10^{-1}

7,999 +
0,0161 =

8,015~~1~~ x 10^{1}

La rappresentazione **migliore** è:

7,999 +
0,0161 =

Risultato normalizzato
8,015~~1~~ x 10^{1}

Con la quale posso scrivere: 1 cifra prima della virgola (8) e 3 cifre dopo la virgola (015), 1 va perso, ma è la cifra che pesa di meno.

Con la rappresentazione più a sinistra, perdo le decine, con quella in mezzo decine e centinaia commettendo un errore grande sulla rappresentazione.

Allineo al numero con esponente maggiore (perdo cifre di peso minore).

Approssimazione

Interi -> risultato esatto (o overflow)

Numeri decimali -> Spesso occorrono delle approssimazioni
- Troncamento (floor): 8,0151 -> 8,015
- Arrotondamento alla cifra superiore (ceil): 8,0151 -> 8,016
- Arrotondamento alla cifra più vicina: (round) 8,0151 -> 8,015

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

Invece di approssimare gli operandi, I bit di guardia e arrotondamento consentono di approssimare il risultato finale.

Esempio: aritmetica in floating point accurata

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

Senza cifre di arrotondamento e utilizzando il troncamento, devo scrivere:

2,34 +
0,02 = ho troncato il secondo addendo per rimanere nella capacità

2,36

Con il bit di guardia e di arrotondamento posso scrivere:

2,3400 +
0,0256 =

2,3656

L'arrotondamento finale (round) viene effettuato **sul risultato** per rientrare in 3 cifre decimali fornisce: **2,37**

L'effetto perverso del troncamento

C = A + B
if (**C > A**) then
  (a)…
else
  (b)….

A = 7,999 x 10^{1} B = 1,61 x 10^{-1} C = A + B = (7,999 + 0,0161) x 10^{1} = 8,0151 x 10^{1}
*Passando alla codifica su 4 bit con **troncamento degli operandi** ottengo:*
A = 7,999 x 10^{1} B = 1,61 x 10^{-1} C = A + B = (7,999 + 0,0161) x 10^{1} = 8,015
=> C > A correttamente

A = 7,999 x 10^{1} B = 1,61 **x 10^{-4}** C = A + B = 7,999161
*Passando alla codifica su 4 bit con **troncamento degli operandi** ottengo:*
A = 7,999 x 10^{1} B = 1,61 x 10^{-4} C = A + B = (7,999 + 0,0000161) x 10^{1} = 7,999
=> C = A errore sull'istruzione di test!!!

Questo è un errore molto comune quando si considera l'aritmetica con i numeri decimali

Non vale la proprietà associativa della somma

Z = A + (B + C)

A = -10^{38} B = 10^{38} C = 1

(B + C) = 10^{38} =➔ Z = 0 Risultato sbagliato

Z = (A + B) + C

(A + B) = 0 =➔ Z = 1 Risultato corretto

Risultati molto diversi.

Problemi di arrotondamento – IEEE 754

A = 4
B = 1,000000357627868652343 8 x 10^0

In IEEE754:
A = 1 x 2^2 = 4
B = 1, 00000 00000 00000 00000 011 x 2^0 parte frazionaria su 23 bit

A = 0 1000 0001 00000 00000 00000 00000 000 codifica IEEE754 su 32 bit
B = 0 1111 1111 00000 00000 00000 00000 011 codifica IEEE754 su 32 bit

Allineo B ad A: => B = 0, 01000 00000 00000 00000 000~~11~~ x 2^2 parte frazionaria su 23 bit
Segue che: => C = A + B = 1,01000 00000 00000 00000 000 x 2^2 su 23 bit

0,01000 00000 00000 00000 000~~11~~ +	su 23 bit	Base = 2^2
1,00000 00000 00000 00000 000	=	su 23 bit	

1,01000 00000 00000 00000 000	su 23 bit	Base = 2^2

C = A+B = 5

Problemi di troncamento – IEEE 754

A = 4
B = 1,000000357627868652343 8 x 10^0

In IEEE754:
A = 1 x 2^2
B = 1, 00000 00000 00000 00000 011 x 2^0 parte frazionaria su 23 bit

A = 0 1000 0001 00000 00000 00000 00000 000 codifica IEEE754 su 32 bit
B = 0 1111 1111 00000 00000 00000 00000 011 codifica IEEE754 su 32 bit

Senza aggiungere i bit di guardia e arrotondamento quando allineo B ad A (Potenza : 2^2):

	0,01000 00000 00000 00000 000 +	su 23 bit	Base = 2^2
	1,00000 00000 00000 00000 000 =	su 23 bit	
Somma	---		
	1,01000 00000 00000 00000 000	su 23 bit	Base = 2^2

Per rounding, C = 1,01000 00000 00000 00000 000 su 23 bit Base = 2^2

C = A+B = A+1 = 5 può essere pericoloso.

Problemi di troncamento – IEEE 754

A = 4

B = 1,000000357627868652343 8 x 10^0

In IEEE754:

A = 1 x 2^2

B = 1, 00000 00000 00000 00000 011 x 2^0 parte frazionaria su 23 bit

A = 0 1000 0001 00000 00000 00000 00000 000 codifica IEEE754 su 32 bit

B = 0 1111 1111 00000 00000 00000 00000 011 codifica IEEE754 su 32 bit

Se aggiungo i bit di guardia e arrotondamento quando allineo B ad A (Potenza : 2^2):

	0,01000 00000 00000 00000 00011 +	su 25 bit	Base = 2^2
	1,00000 00000 00000 00000 00000 =	su 25 bit	

Somma	1,01000 00000 00000 00000 00011	su 25 bit	Base = 2^2
Per rounding, C =	1,01000 00000 00000 00000 001	su 23 bit	Base = 2^2

C = A+B = 5 + $2^{-23}2^2$ = 5+0.00000047683715820312 5

molto più vicino al valore vero!

Algoritmo di somma in virgola mobile - I

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

a = 9,12 x10^0 ⇓ a = 9,12 x10^0

b= 8,99 x 10^{-1} ⇓ b= 0,899 x 10^0

2) Effettuare la somma delle mantisse.

9,12 +
0,899 =

10,019 x 10^0

Se il numero risultante è normalizzato termino qui. Altrimenti:

3) Normalizzare il risultato.

10,019 x 10^0 ➔ 1,0019 x 10^1

Esempio di somma in virgola mobile - II

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

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

1) Esprimo entrambi i numeri con la base 10^{1}
1,61 x 10^{-1} = 0,0161 x 10^{1}

2) Somma delle mantisse:

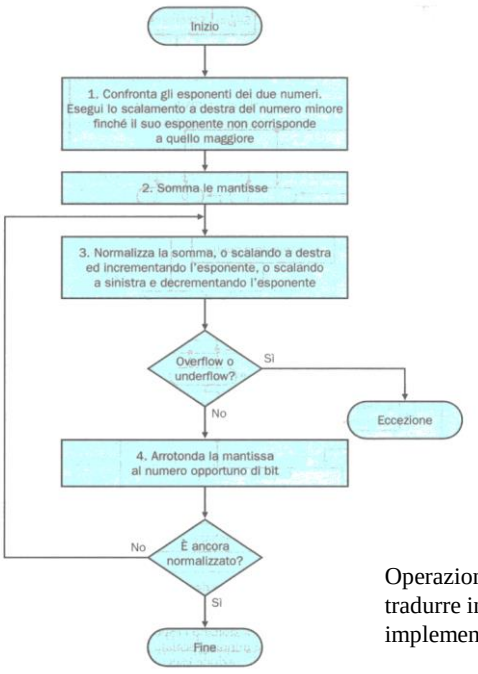
9,999 +
0,016~~(1)~~ = Perdo una cifra perchè non rientra nella capacità della mantissa (troncamento)

10,015 x 10^{1} Il risultato non è più normalizzato, anche se i due addendi sono normalizzati.

NB: In questa fase si può generare la necessità di rinormalizzare il numero (passo 3):

10,015 x 10^{1} = 1,001 x 10^{2} in forma normalizzata (per una cifra per effetto del troncamento)

Algoritmo risultante

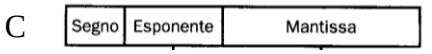


Operazioni complesse da tradurre in operazioni implementabili dall'hardware

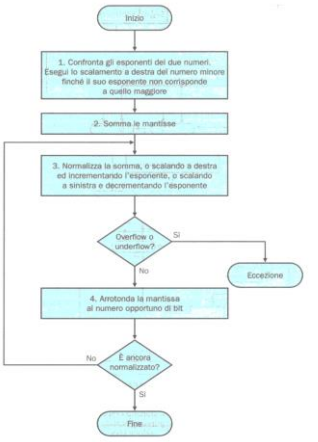
Il circuito della somma floating point: gli attori

A + B

C



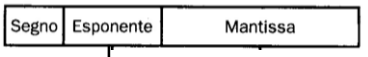
Rappresentazione normalizzata IEEE754

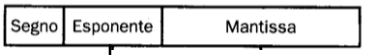


Determinazione dell'esponente

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

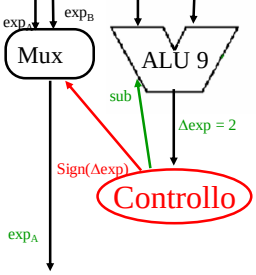
A





B

1 −1



Confronto degli esponenti
(sottrazione: 8 bit di campo + 1 segno)

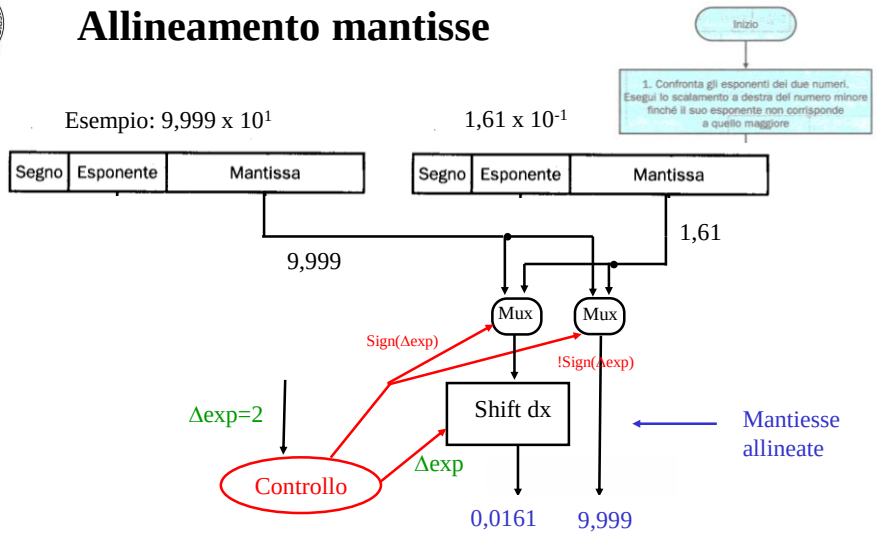
If ($\Delta exp \geq 0$) Exp somma = exp_A
If ($\Delta exp < 0$) Exp somma = exp_B

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

1a) $1{,}61 \times 10^{-1} \Rightarrow 0{,}0161 \times 10^{1}$ $\Rightarrow \Delta exp = +2$

Exp somma = $exp_A = +1$

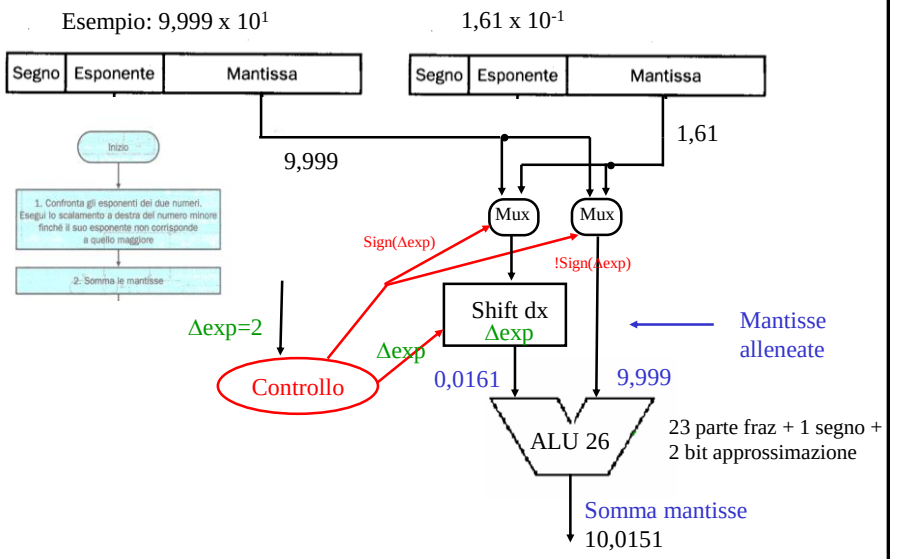
Allineamento mantisse



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

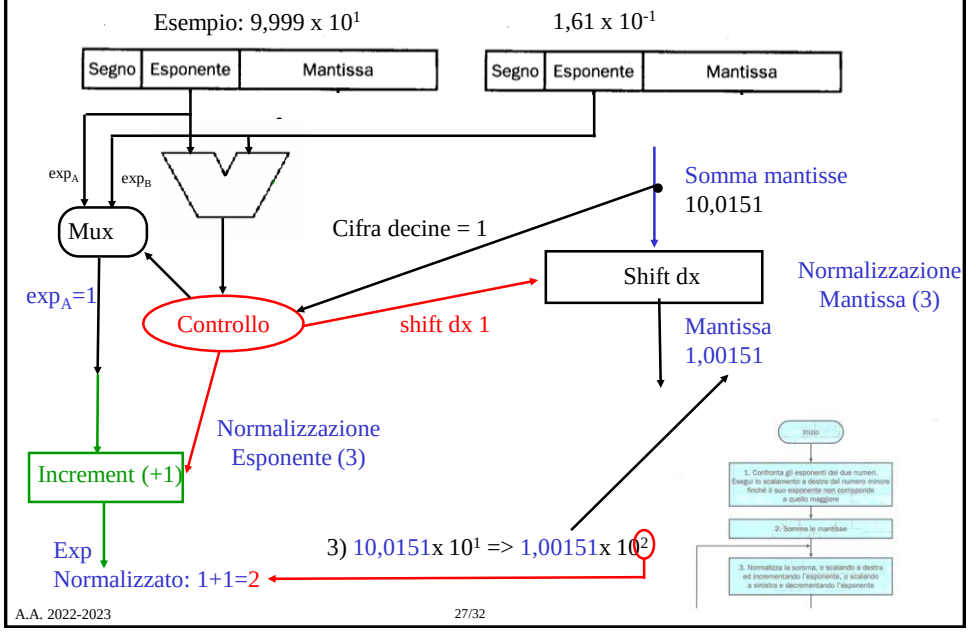
1b) 1,61 x 10^{-1} => 0,0161 x 10^1 => Δexp = +2

Somma delle mantisse

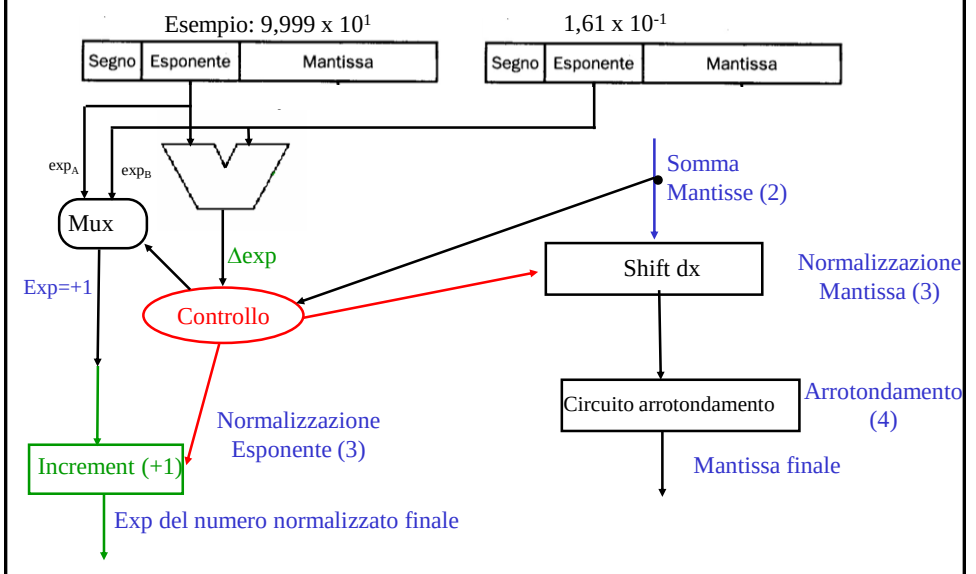


2) Somma delle mantisse: 0,0161 x 10^1 + 9,999 x 10^1 = 10,0151 x 10^1

Normalizzazione



Il circuito della somma floating point: arrotondamento e normalizzazione



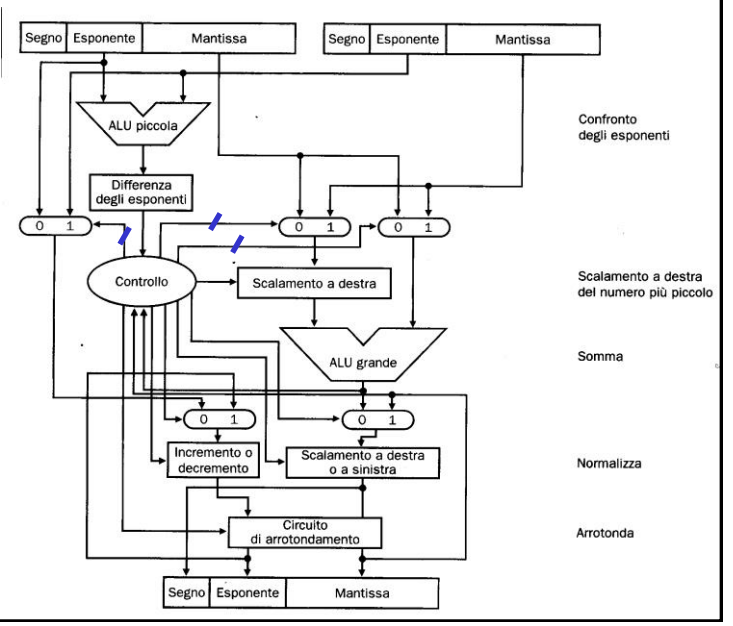
Circuito della somma floating point

Le tre linee in blu contengono lo stesso segnale di controllo (funzione di Δexp).

Gestisce anche la rinormalizzazione: $9{,}99999 \times 10^2 = 10{,}00 \times 10^1$

Gestisce anche i numeri negativi.

Problemi?



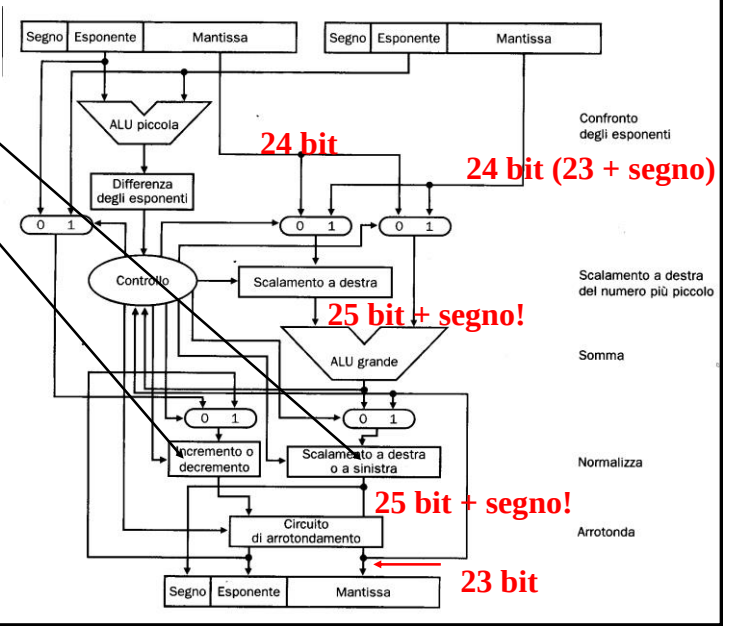
Circuito della somma floating point con bit di arrotondamento

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.

Mantissa 1,…



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 del firmware
- Somma in virgola mobile