



UC firmware moltiplicazione Floating pointer adder

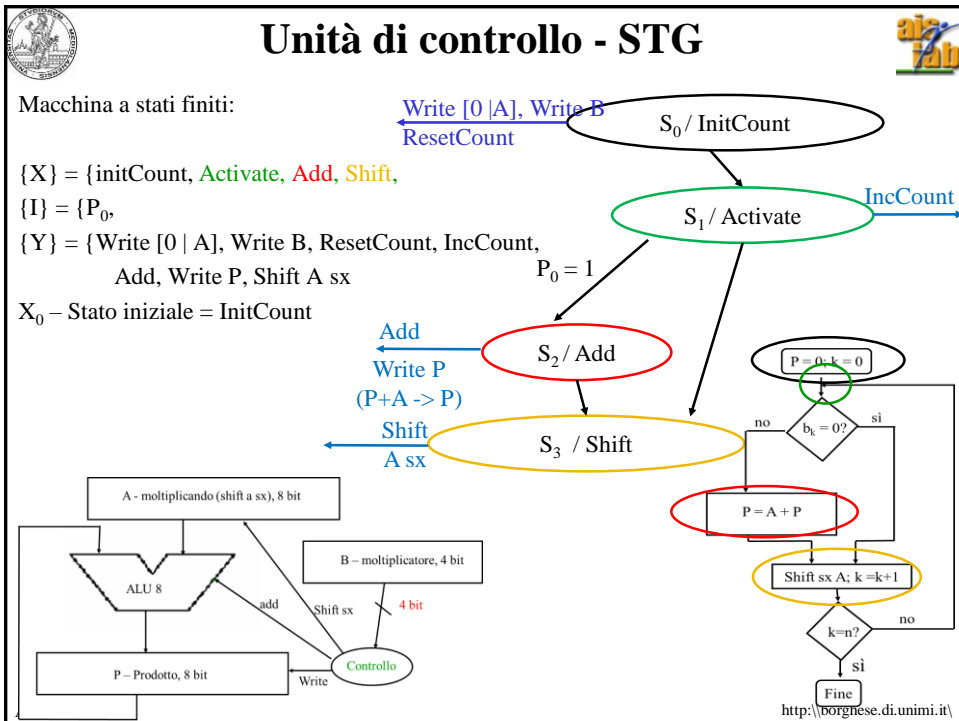
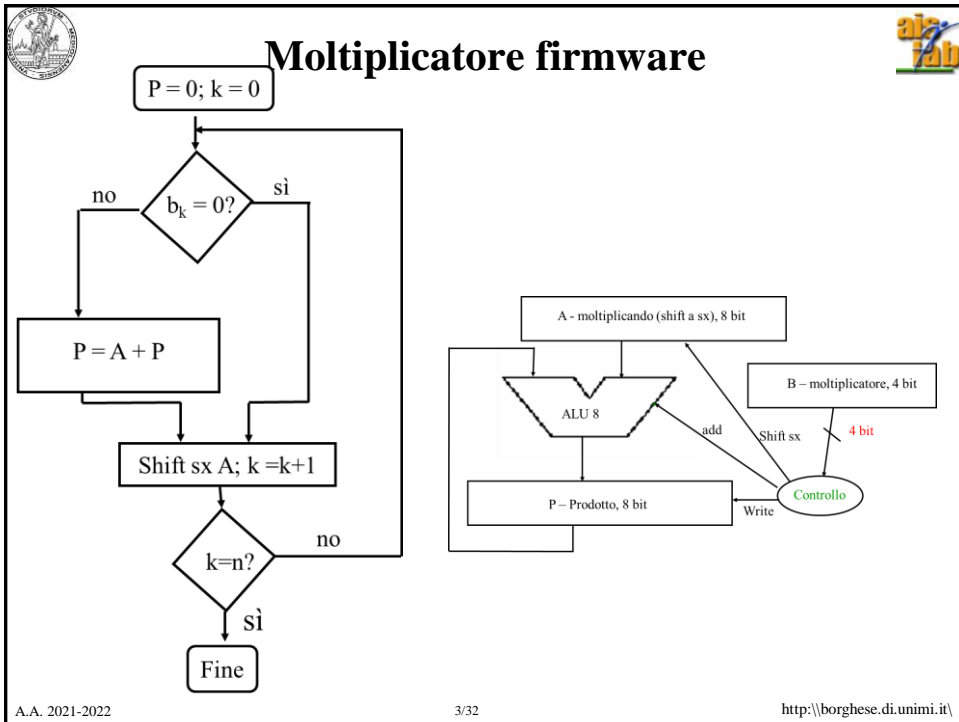
Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

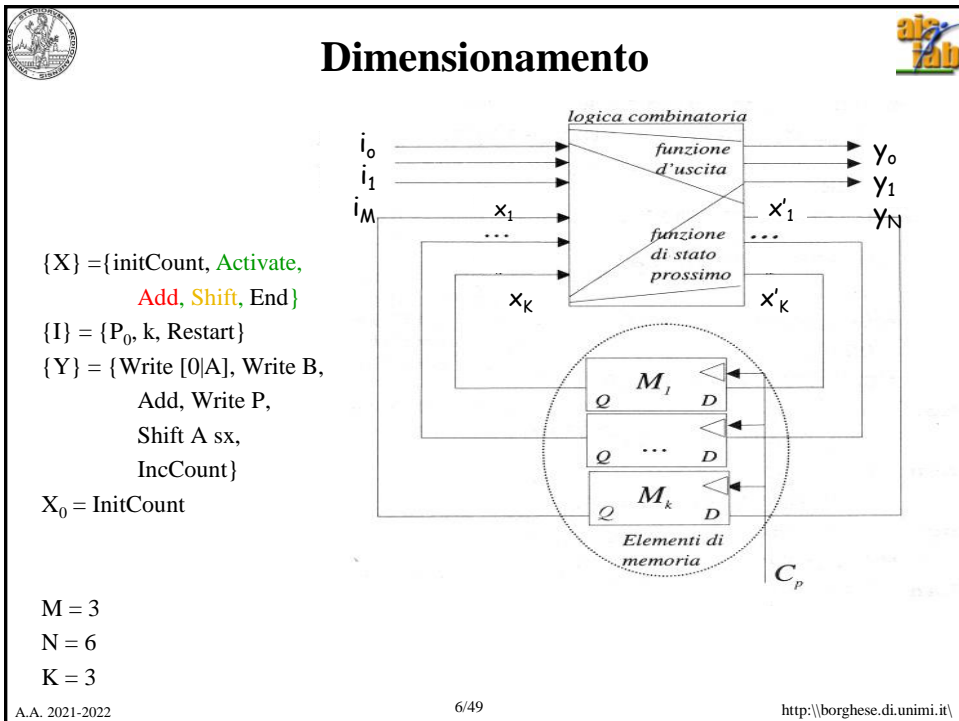
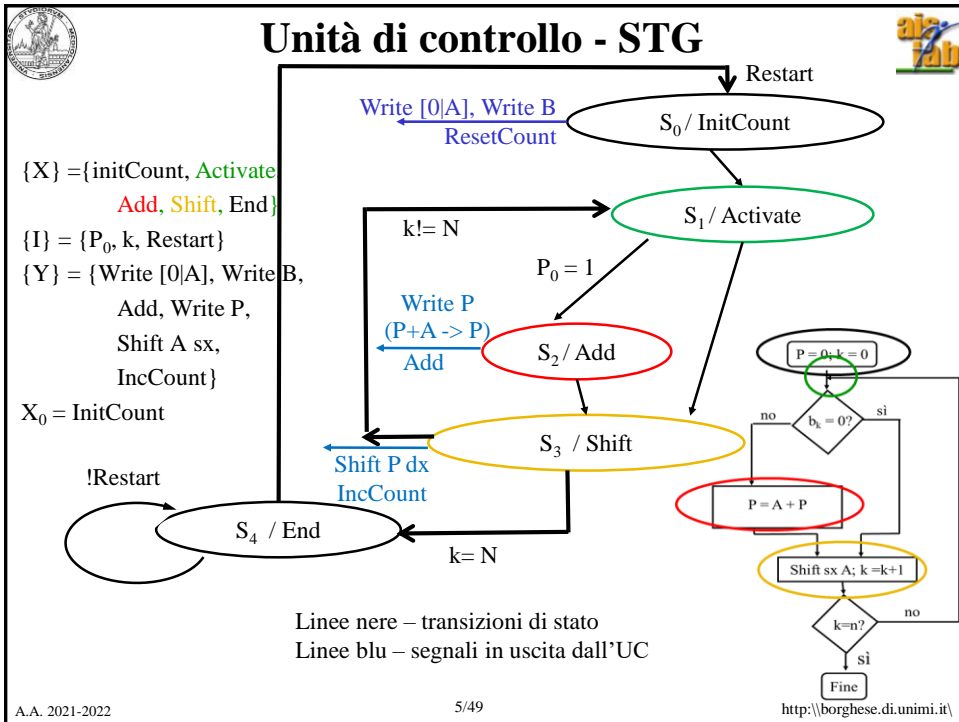
Università degli Studi di Milano
Riferimenti sul Patterson, 6a Ed.: 3.4, 3.5, 4.2



Sommario

- **Unità di controllo del firmware**
- Somma in virgola mobile







Unità di controllo - STT



$\{X\} = \{\text{initCount, Activate, Add, Shift, End}\}$

$\{I\} = \{P_0, k, \text{Restart}\}$

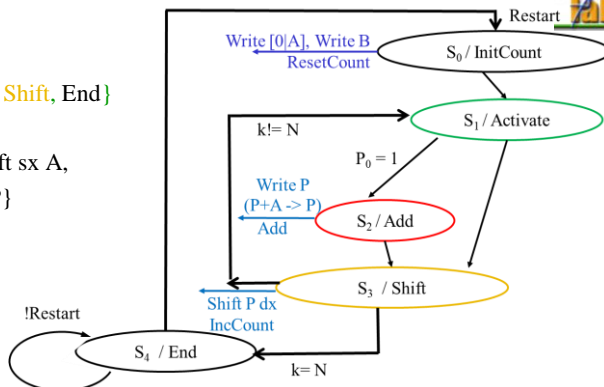
$\{Y\} = \{\text{Write } [0|A], \text{Write B, Shift } s_x \text{ A,}$

$\text{IncCount, Add, Write P}\}$

$X_0 = \text{InitCount}$

$f(X,I)$ – Funzione stato prossimo

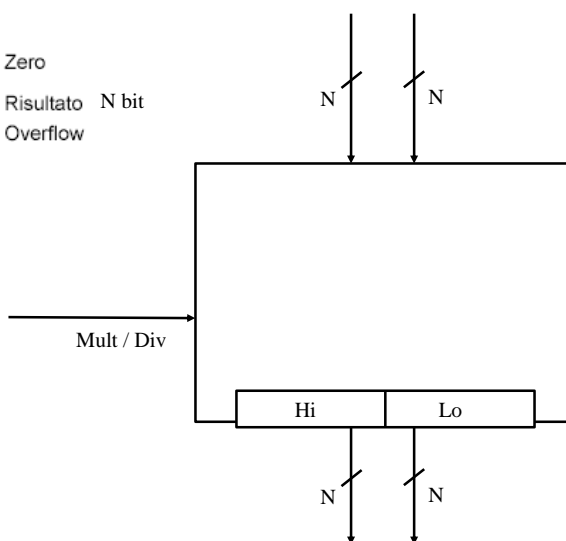
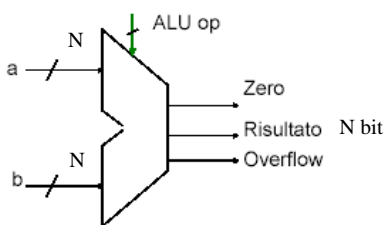
$g(X)$ – Funzione di uscita



	$k \neq N$ $b_k = 0$	$k = N$ $b_k = 0$	$k \neq N$ $b_k = 1$	$k = N$ $b_k = 1$	$k \neq N$ $b_k = 0$	$k = N$ $b_k = 0$	$k \neq N$ $b_k = 1$	$k = N$ $b_k = 1$	Uscita
InitCount	Restart	Restart	Restart	Restart	Restart	Restart	Restart	Restart	Clear P, Clear counter
Activate	Activate	Activate	Activate	Activate	Activate	Activate	Activate	Activate	Inc counter
Add	Shift	Shift	Add	Add	Shift	Shift	Add	Add	Add, Write P
Shift	Shift	Shift	Shift	Shift	Shift	Shift	Shift	Shift	Shift A sx
End	End	End	End	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)

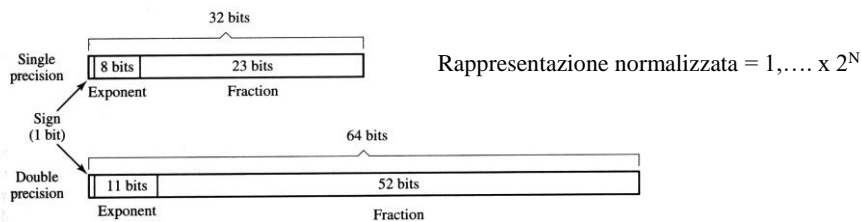


Figure 2-10 Single-precision and double-precision IEEE 754 floating point formats.

Rappresentazione 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



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

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

Una possibilità è:

$$\begin{array}{r} 79,99 + \\ 0,161 = \\ \hline \end{array} \quad \begin{array}{l} a = 7,999 \times 10^1 = 79,99 \times 10^0 \\ b = 1,61 \times 10^{-1} = 0,161 \times 10^0 \end{array}$$

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

Altre possibilità sono:

$$\begin{array}{r} 799,9 + \\ 1,61 = \\ \hline \end{array} \quad \begin{array}{r} 7,999 + \\ 0,0161 = \\ \hline \end{array}$$

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



Quale forma conviene utilizzare?



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

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

$$\begin{array}{r} 79,99 + \\ 0,161 = \\ \hline \end{array} \quad \begin{array}{r} 799,9 + \\ 1,61 = \\ \hline \end{array} \quad \begin{array}{r} 7,999 + \\ 0,0161 = \\ \hline \end{array}$$

$$80,151 \times 10^0 \quad 801,51 \times 10^{-1} \quad 8,0151 \times 10^1$$

La rappresentazione **migliore** è:

$$\begin{array}{r} 7,999 + \\ 0,0161 = \\ \hline \end{array}$$

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

Risultato normalizzato

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



$$a = 2,34 \quad b = 0,0256$$

$$a + b = ?$$

Codifica su 3 cifre decimali totali.

Approssimazione mediante **rounding**.

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

$$2,34 +$$

$$0,02 =$$

2,36

ho troncato il secondo addendo per rimanere nella capacità

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)....

$$A = 7,999 \times 10^1 \quad B = 1,61 \times 10^{-1} \quad C = A + B = (7,999 + 0,0161) \times 10^1 = 8,0151 \times 10^1$$

Passando alla codifica su 4 bit con **troncamento degli operandi** ottengo:

$$A = 7,999 \times 10^1 \quad B = 1,61 \times 10^{-1} \quad C = A + B = (7,999 + 0,0161) \times 10^1 = 8,015$$

=> C > A correttamente

$$A = 7,999 \times 10^1 \quad B = 1,61 \times 10^{-4} \quad C = A + B = 7,999161$$

Passando alla codifica su 4 bit con **troncamento degli operandi** ottengo:

$$A = 7,999 \times 10^1 \quad B = 1,61 \times 10^{-4} \quad C = A + B = (7,999 + 0,0000161) \times 10^1 = 7,999$$

=> **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



$$Z = A + (B + C)$$

$$A = -10^{38} \quad B = 10^{38} \quad C = 1$$

$$(B + C) = 10^{38} \quad \Rightarrow \quad Z = 0 \quad \text{Risultato sbagliato}$$

$$Z = (A + B) + C$$

$$(A + B) = 0 \quad \Rightarrow \quad Z = 1 \quad \text{Risultato corretto}$$

Risultati molto diversi.



Problemi di arrotondamento – IEEE 754



$$A = 4$$

$$B = 1,0000003576278686523438 \times 10^0$$

In IEEE754:

$$A = 1 \times 2^2 = 4$$

$$B = 1,00000000000000000000011 \times 2^0 \quad \text{parte frazionaria su 23 bit}$$

$$A = 0 \ 1000 \ 0001 \ 00000 \ 00000 \ 00000 \ 00000 \ 000 \quad \text{codifica IEEE754 su 32 bit}$$

$$B = 0 \ 1111 \ 1111 \ 00000 \ 00000 \ 00000 \ 00000 \ 011 \quad \text{codifica IEEE754 su 32 bit}$$

$$\text{Allineo B ad A: } \Rightarrow B = 0,01000 \ 00000 \ 00000 \ 00000 \ 00000 \ 00001 \times 2^2 \text{ parte frazionaria su 23 bit}$$

$$\text{Segue che: } \Rightarrow C = A + B = 1,01000 \ 00000 \ 00000 \ 00000 \ 000 \times 2^2 \text{ su 23 bit}$$

$$0,01000 \ 00000 \ 00000 \ 00000 \ 00001 \ + \quad \text{su 23 bit} \quad \text{Base} = 2^2$$

$$1,00000 \ 00000 \ 00000 \ 00000 \ 000 \ = \quad \text{su 23 bit}$$

$$1,01000 \ 00000 \ 00000 \ 00000 \ 000 \quad \text{su 23 bit} \quad \text{Base} = 2^2$$

$$C = A+B = 5$$



Problemi di troncamento – IEEE 754



$$A = 4$$

$$B = 1,0000003576278686523438 \times 10^0$$

In IEEE754:

$$A = 1 \times 2^2$$

$$B = 1,00000000000000000000011 \times 2^0 \quad \text{parte frazionaria su 23 bit}$$

$$A = 0 \ 1000 \ 0001 \ 00000 \ 00000 \ 00000 \ 00000 \ 000 \quad \text{codifica IEEE754 su 32 bit}$$

$$B = 0 \ 1111 \ 1111 \ 00000 \ 00000 \ 00000 \ 00000 \ 011 \quad \text{codifica IEEE754 su 32 bit}$$

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

$$0,01000 \ 00000 \ 00000 \ 00000 \ 000 \ + \quad \text{su 23 bit} \quad \text{Base} = 2^2$$

$$1,00000 \ 00000 \ 00000 \ 00000 \ 000 \ = \quad \text{su 23 bit}$$

Somma

$$1,01000 \ 00000 \ 00000 \ 00000 \ 000 \quad \text{su 23 bit} \quad \text{Base} = 2^2$$

$$\text{Per rounding, } C = 1,01000 \ 00000 \ 00000 \ 00000 \ 000 \quad \text{su 23 bit} \quad \text{Base} = 2^2$$

$$C = A+B = A+1 = 5 \quad \text{può essere pericoloso.}$$



Problemi di troncamento – IEEE 754



$$A = 4$$

$$B = 1,0000003576278686523438 \times 10^0$$

In IEEE754:

$$A = 1 \times 2^2$$

$$B = 1,000000000000000000000011 \times 2^0 \quad \text{parte frazionaria su 23 bit}$$

$$A = 0 \quad 1000 \quad 0001 \quad 000000000000000000000000 \quad \text{codifica IEEE754 su 32 bit}$$

$$B = 0 \quad 1111 \quad 1111 \quad 000000000000000000000011 \quad \text{codifica IEEE754 su 32 bit}$$

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

$$0,01000 \quad 00000 \quad 00000 \quad 00000 \quad 00011 \quad + \quad \text{su 25 bit} \quad \text{Base} = 2^2$$

$$1,00000 \quad 00000 \quad 00000 \quad 00000 \quad 00000 \quad = \quad \text{su 25 bit}$$

Somma

$$1,01000 \quad 00000 \quad 00000 \quad 00000 \quad 00011 \quad \text{su 25 bit} \quad \text{Base} = 2^2$$

$$\text{Per rounding, } C = 1,01000 \quad 00000 \quad 00000 \quad 00000 \quad 001 \quad \text{su 23 bit} \quad \text{Base} = 2^2$$

$$C = A+B = 5 + 2^{-23}2^2 = 5+0.000000476837158203125$$

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).

$$a = 9,12 \times 10^0$$

↓

$$a = 9,12 \times 10^0$$

$$b = 8,99 \times 10^{-1}$$

↓

$$b = 0,899 \times 10^0$$

- 2) Effettuare la somma delle mantisse.

$$9,12 +$$

$$0,899 =$$

$$10,019 \times 10^0$$

Se il numero risultante è normalizzato termino qui. Altrimenti:

- 3) Normalizzare il risultato.

$$10,019 \times 10^0 \rightarrow 10019 \times 10^1$$



Esempio di somma in virgola mobile - II



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

$$a + b = ?$$

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

$$1,61 \times 10^{-1} = 0,0161 \times 10^1$$

2) Somma delle mantisse:

$$9,999 \quad +$$

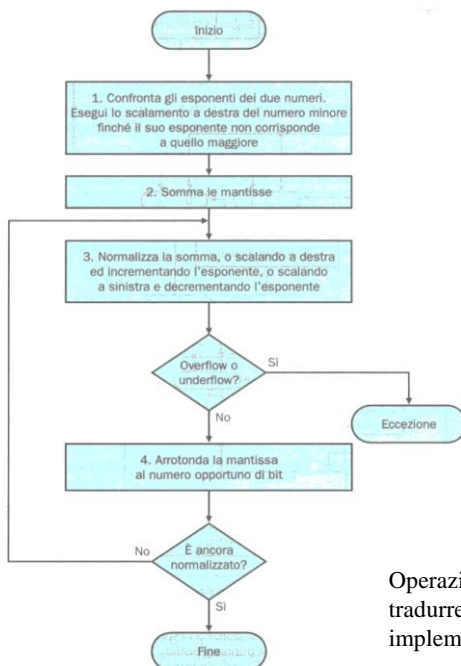
$$0,0161 \quad = \quad \text{Perdo una cifra perchè non rientra nella capacità della mantissa (troncamento)}$$

$$\text{-----}$$
$$10,015 \times 10^1$$

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):

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



Algoritmo risultante

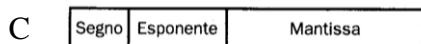
Operazioni complesse da tradurre in operazioni implementabili dall'hardware



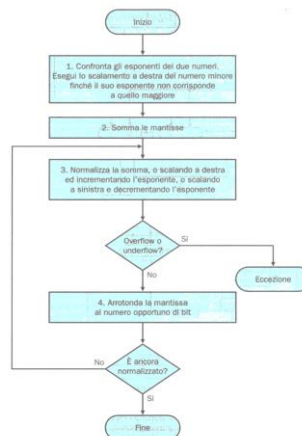
Il circuito della somma floating point: gli attori



A + B



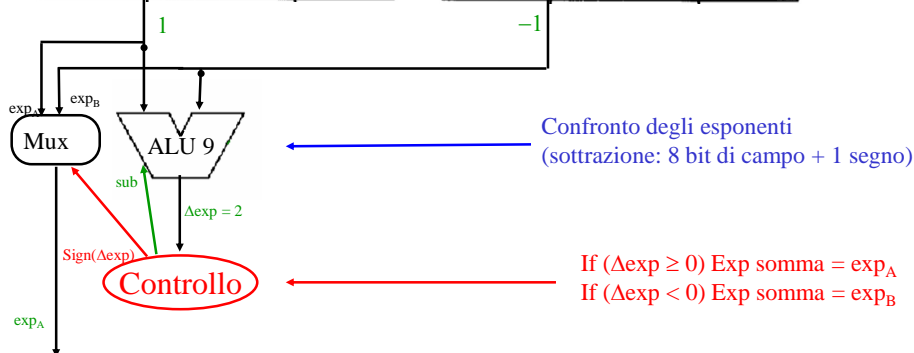
Rappresentazione normalizzata IEEE754



Determinazione dell'esponente



1. Confronta gli esponenti dei due numeri. Eseguì lo scalamiento a destra del numero minore finché il suo esponente non corrisponde a quello maggiore



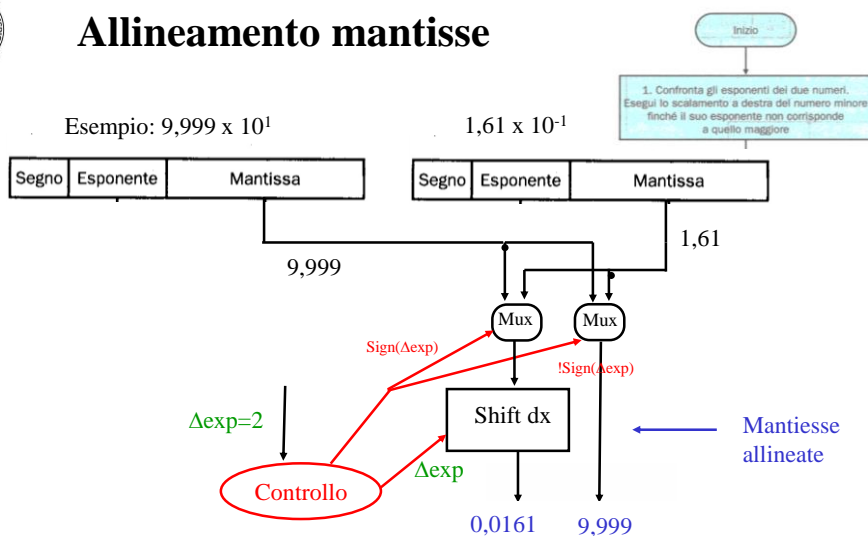
Esempio: $9,999 \times 10^1 + 1,61 \times 10^{-1} = ?$

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

Exp somma = $exp_A + 1$



Allineamento mantisse

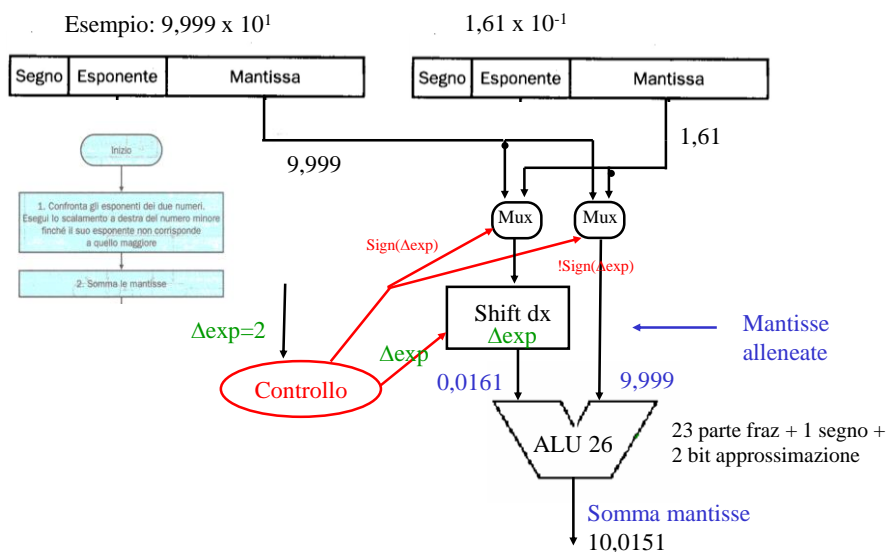


Esempio: $9,999 \times 10^1 + 1,61 \times 10^{-1} = ?$

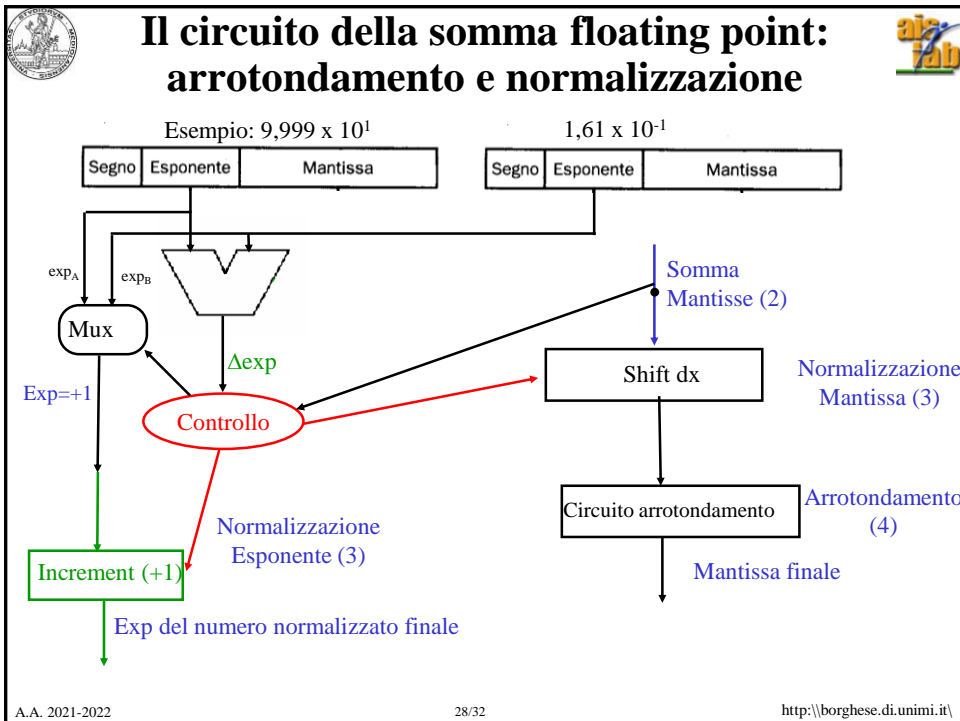
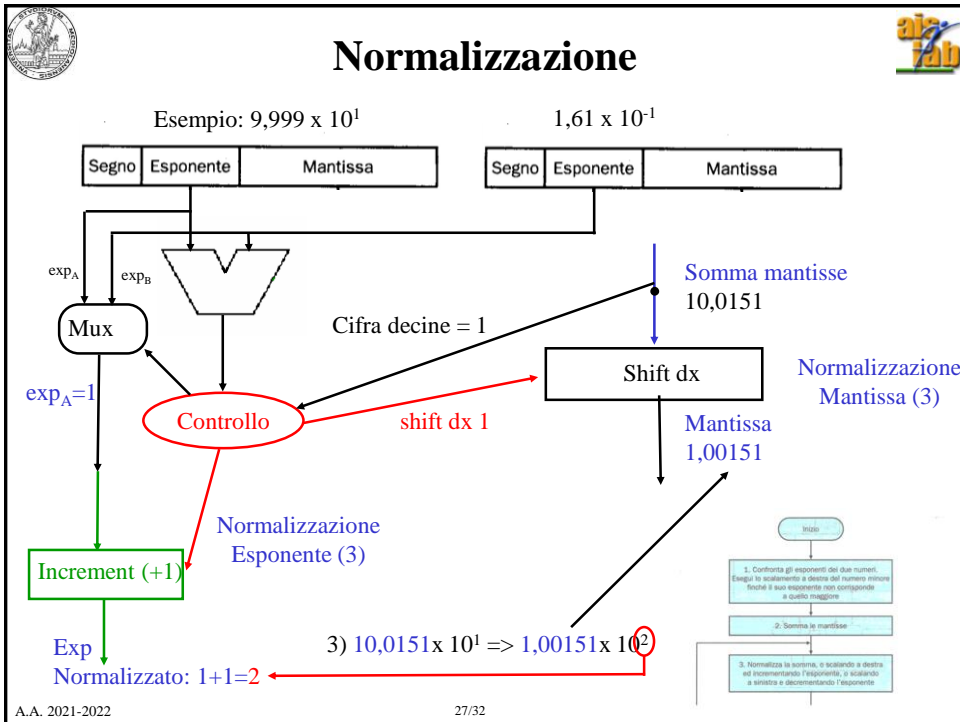
1b) $1,61 \times 10^{-1} \Rightarrow 0,0161 \times 10^1 \Rightarrow \Delta exp = +2$



Somma delle mantisse



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





Circuito della somma floating point

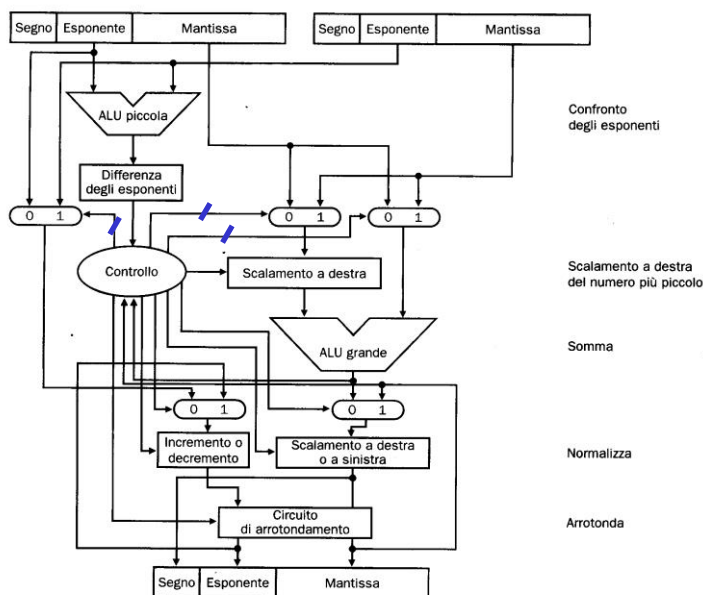


Le tre linee in blu contengono lo stesso segnale di controllo (funzione di Δexp).

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

Gestisce anche i numeri negativi.

Problemi?



A.A. 2021-2022



Circuito della somma floating point con bit di arrotondamento

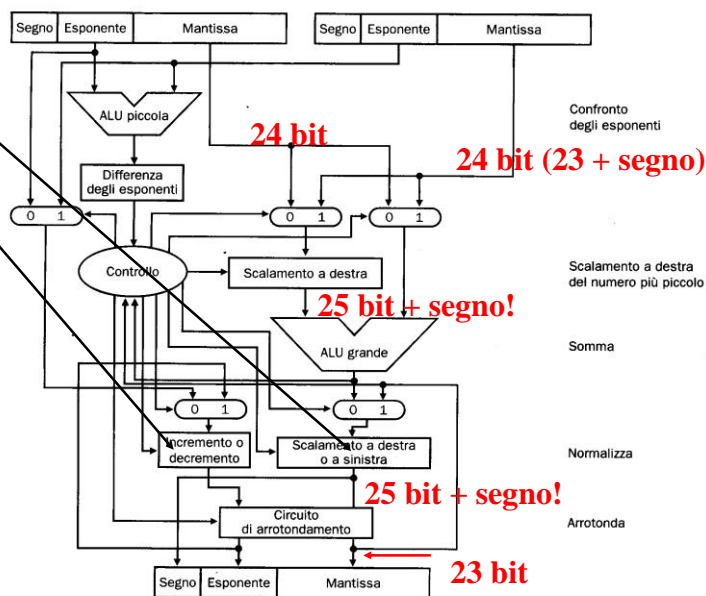


In quale caso la mantissa viene scalata a sx?

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,...



A.A. 2021-2022



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