



UC firmware moltiplicazione Floating pointer adder

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

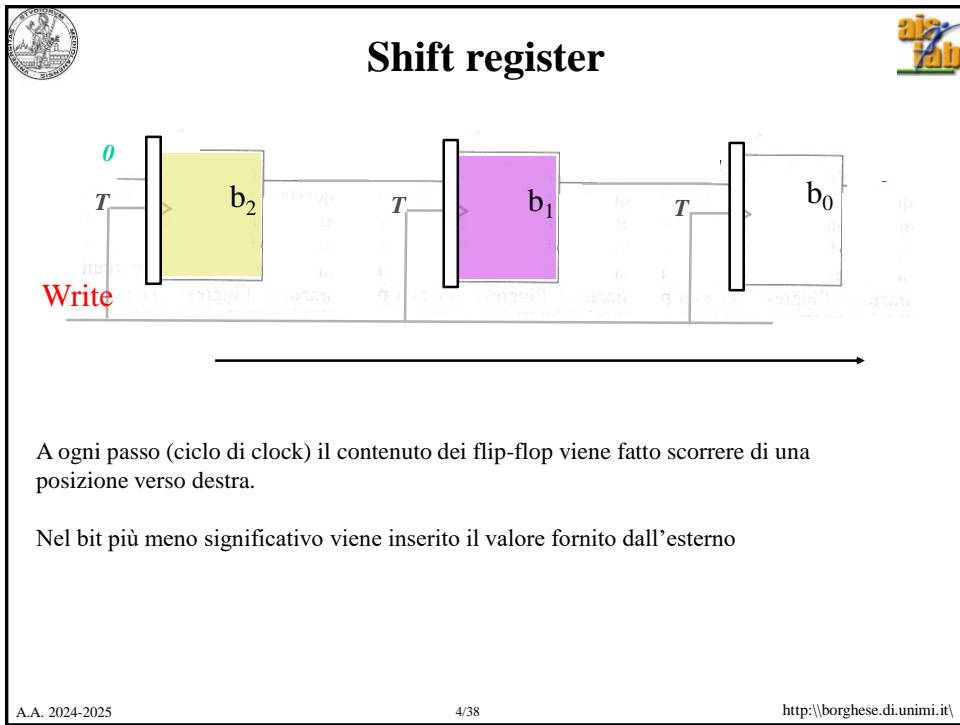
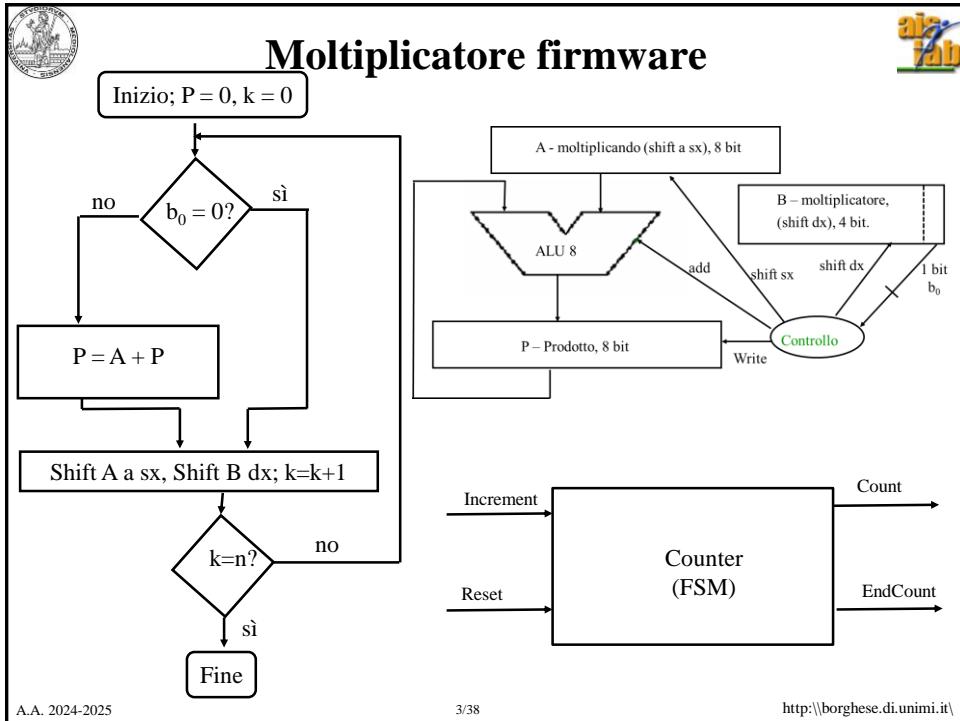
Università degli Studi di Milano
Riferimenti sul Patterson, 6a Ed.: 3.4, 3.5, 4.2

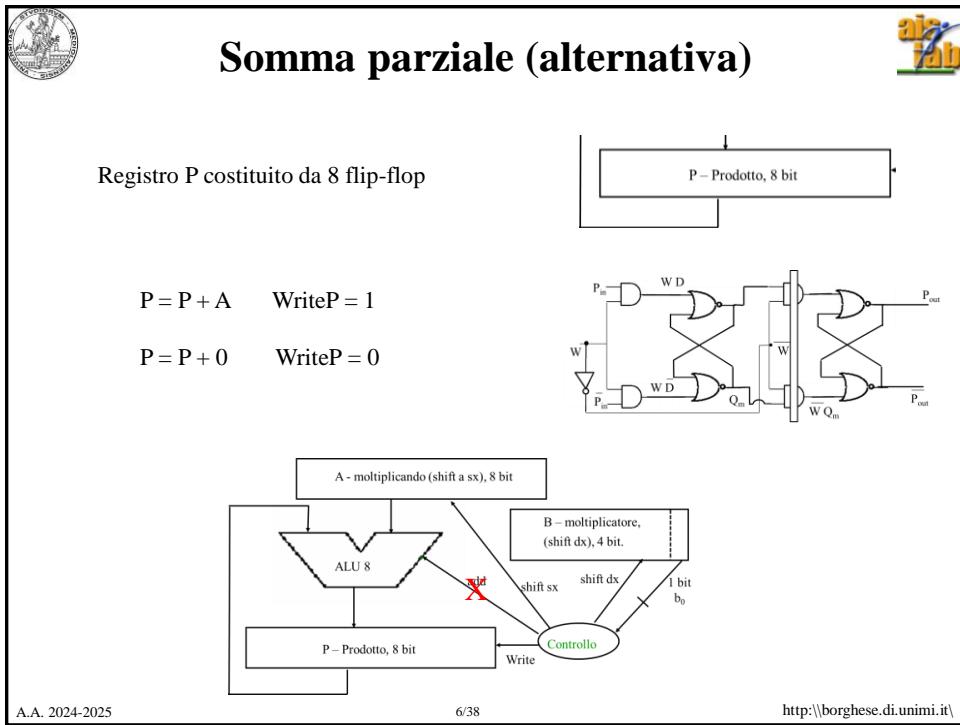
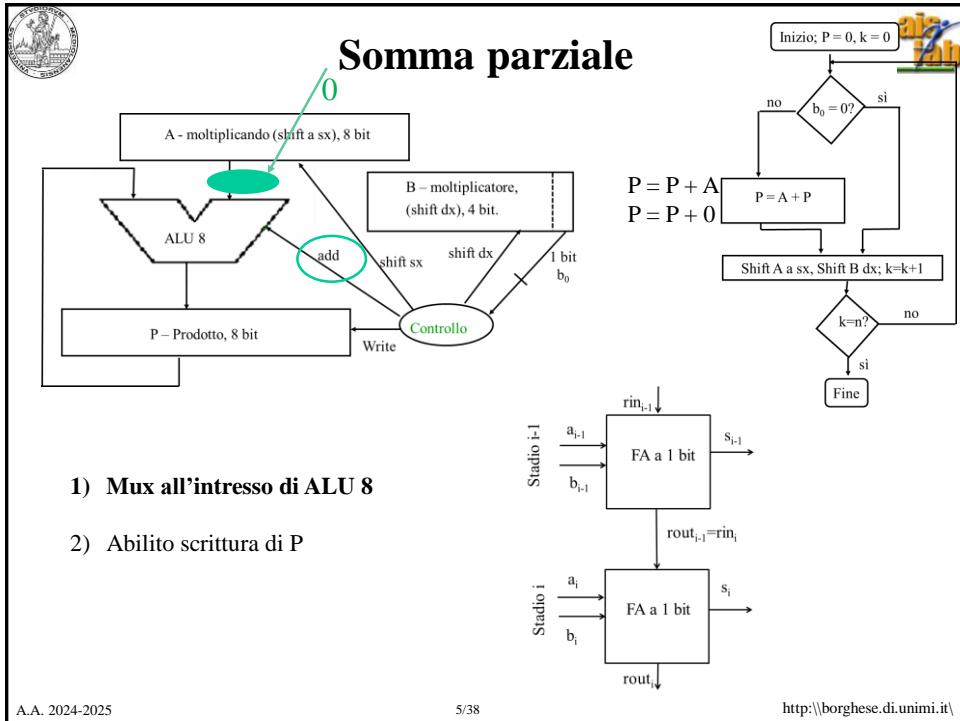


Sommario



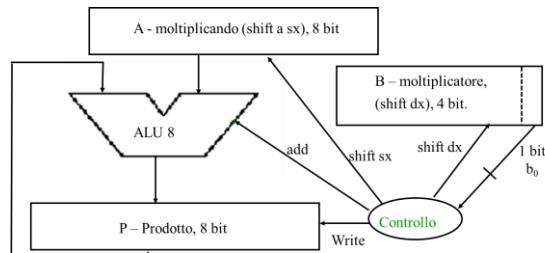
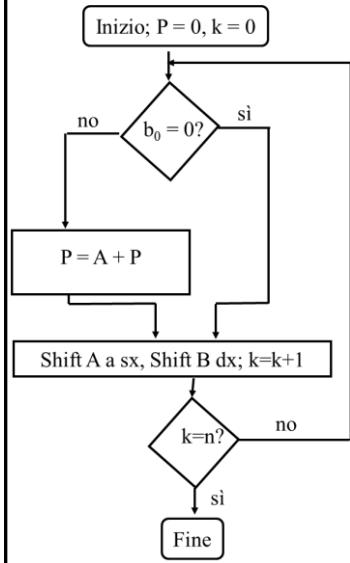
- Unità di controllo del firmware
- Somma in virgola mobile







Operazioni elementari



Da eseguire in sequenza:

- Somma ($P = P+A$ or $P = P + 0$)
- Shift sx A; Shift dx B

Co-design del control-path e del data-path



STG – S_0

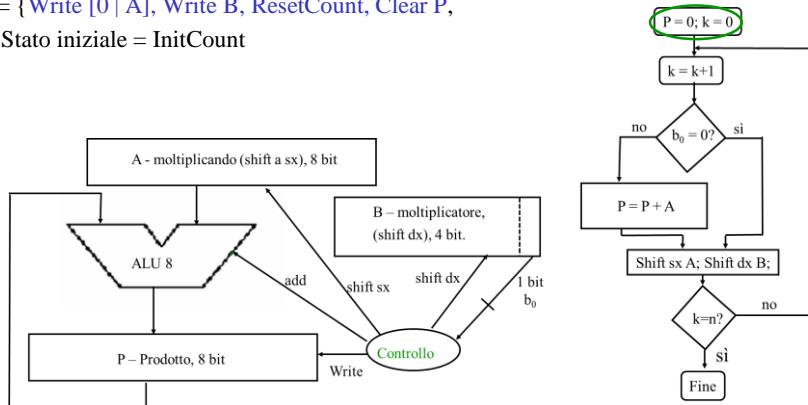


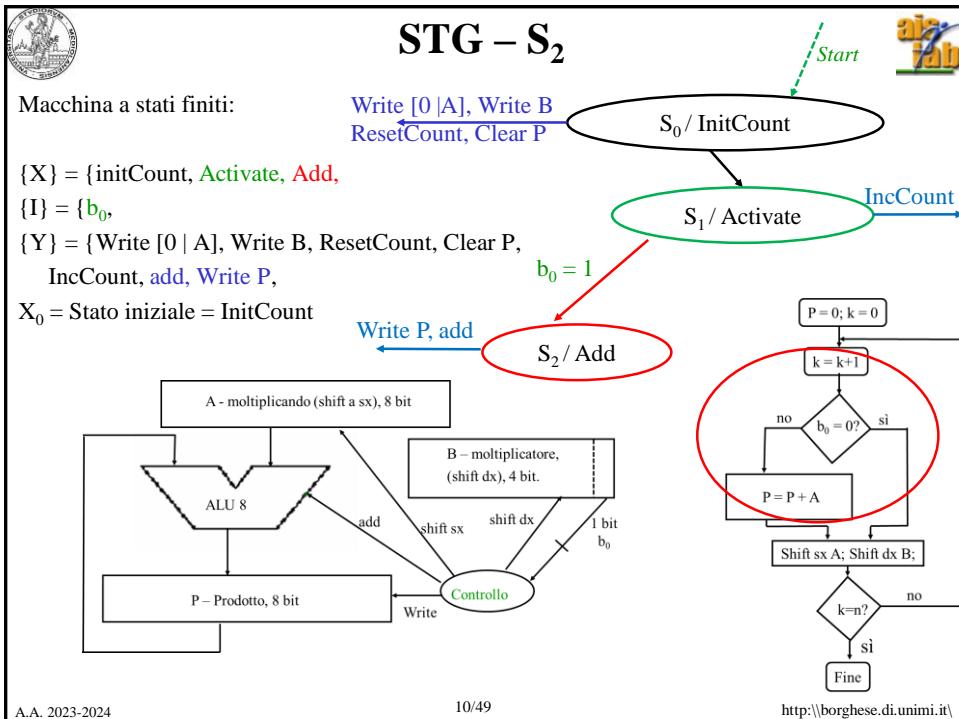
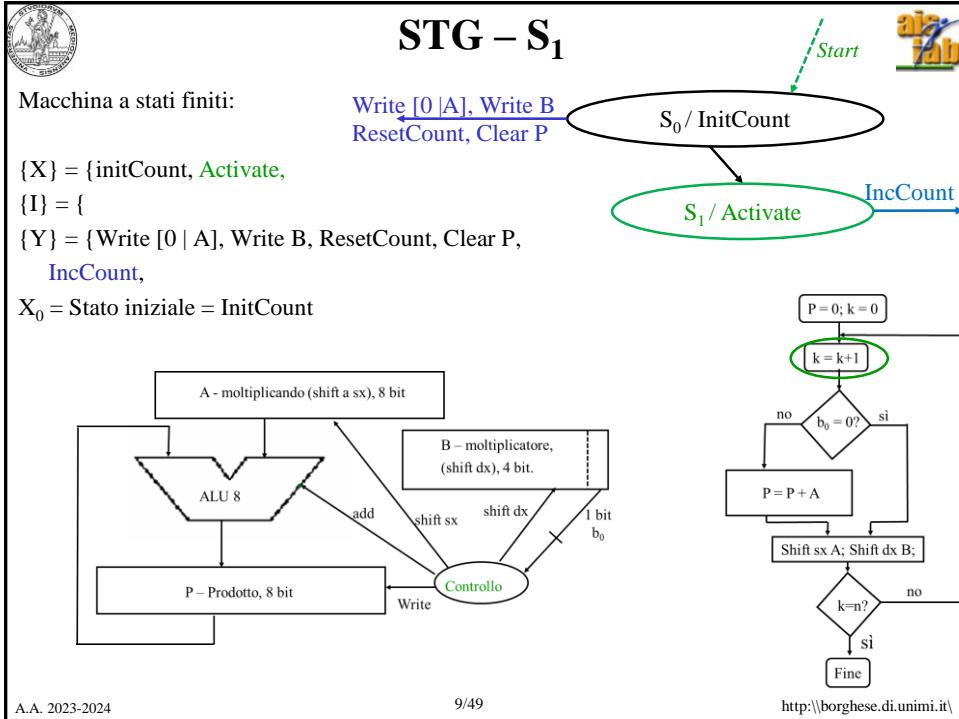
Macchina a stati finiti:

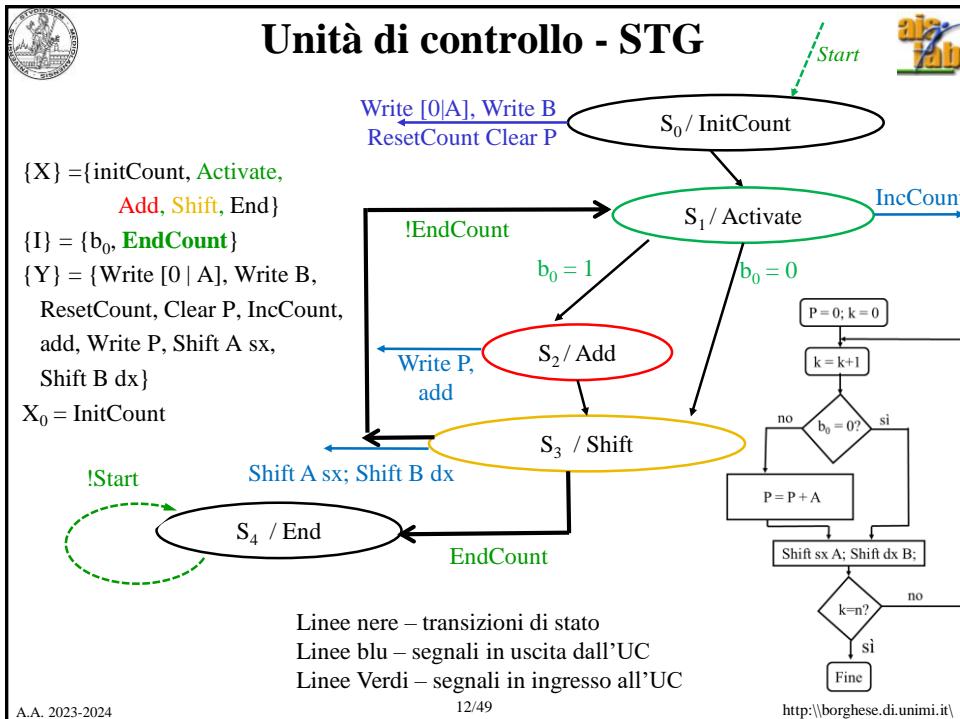
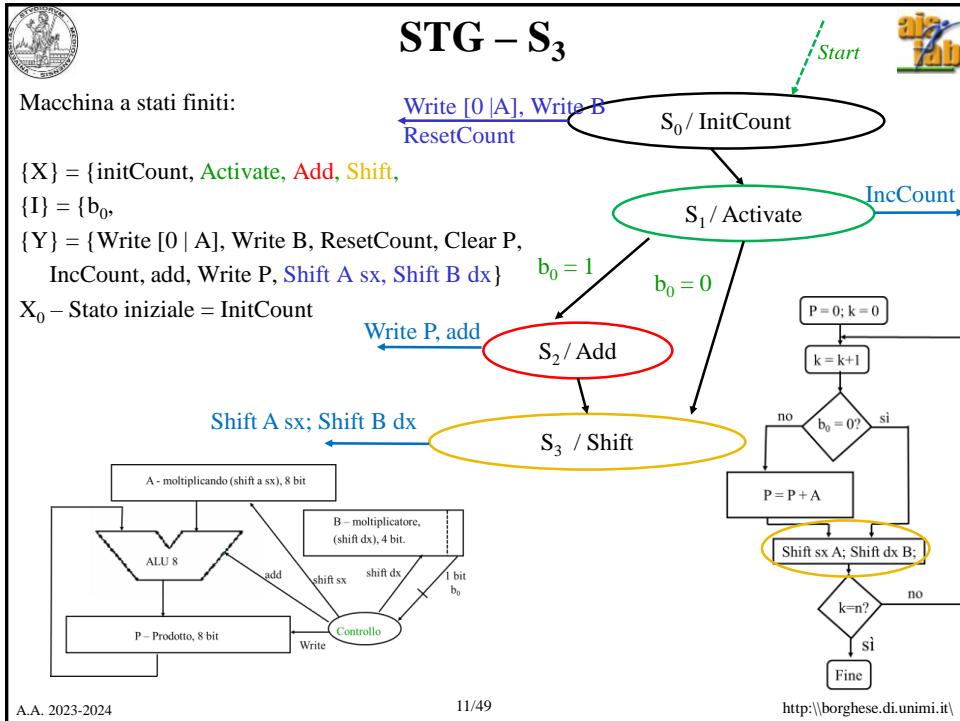
Write [0 | A], Write B
ResetCount, Clear P

$S_0 / \text{InitCount}$

$\{X\} = \{\text{initCount},$
 $\{I\} = \{$
 $\{Y\} = \{\text{Write } [0 | A], \text{ Write } B, \text{ ResetCount, Clear } P,$
 $X_0 = \text{Stato iniziale} = \text{InitCount}$









Macchina di Huffman

$\{X\} = \{\text{initCount, Activate, }$

$\text{Add, Shift, End}\}$

$\{I\} = \{b_0, \text{EndCount}\}$

$\{Y\} = \{\text{Write } [0 | A], \text{Write B, }$
 $\text{ResetCount, Clear P, IncCount, }$
 $\text{add, Write P, Shift A sx, Shift B dx}\}$

$X_0 = \text{InitCount}$

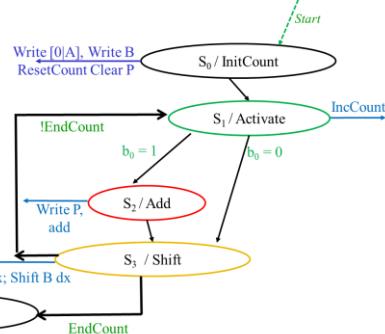
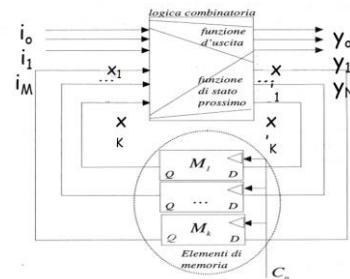
$M = 2$ - 2 input binary (yes/no)

$N = 9$ - 9 uscite binarie

(yes/no - write / no write)

$K = 3$ - 5 stati

NB Sono riportati solo i segnali in uscita (Y) quando vengono asseriti (=1)



Unità di controllo - STT



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

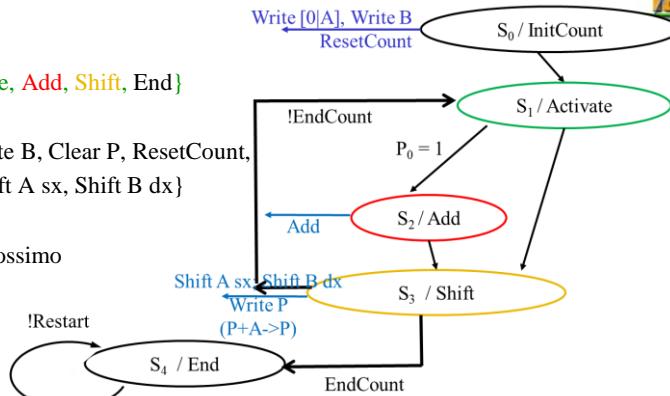
$\{I\} = \{b_0, \text{EndCount}\}$

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

$X_0 = \text{InitCount}$

$f(X, I)$ – Funzione stato prossimo

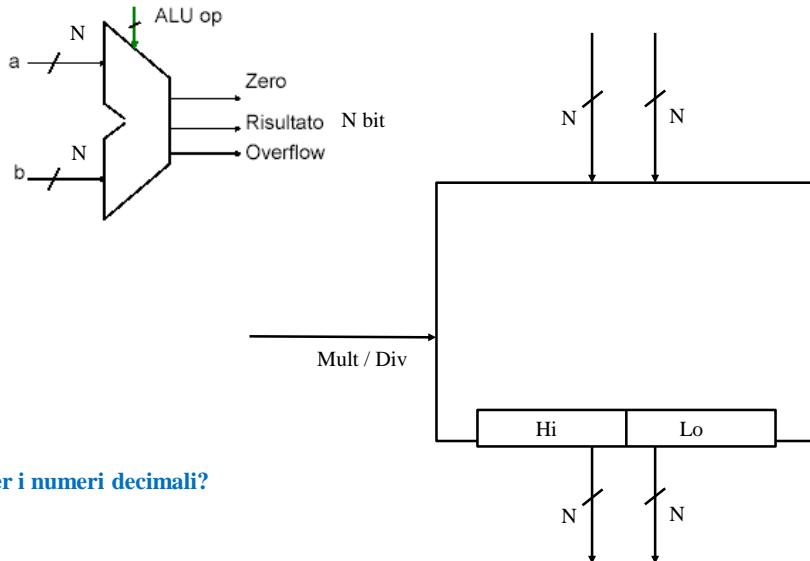
$g(X)$ – Funzione di uscita



	!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, Write P
Shift	Activate	End	Activate	End	Shift A sx, Shift B dx
End	End	End	End	End	



Circuiti operazioni tra numeri interi



Sommario



- Unità di controllo del firmware
- Somma in virgola mobile



Codifica in virgola mobile Standard IEEE 754 (1980)

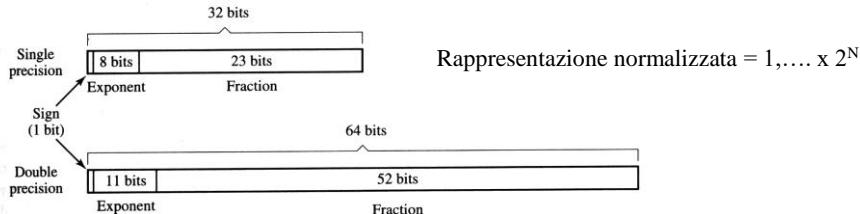


Figure 2-10 Single-precision and double-precision IEEE 754 floating point formats.

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:

$$\begin{array}{r}
 79,99 + \\
 0,161 = \\
 \hline
 80,151 \times 10^0 = 80,151
 \end{array}
 \Rightarrow \text{Allineo all'unità}$$

Altre possibilità sono:

$$\begin{array}{r}
 799,9 + \\
 1,61 = \\
 \hline
 801,51 \times 10^{-1} = 8,0151 \times 10^1
 \end{array}
 \Rightarrow \text{Allineo alla potenza maggiore}
 \qquad
 \begin{array}{r}
 7,999 + \\
 0,0161 = \\
 \hline
 8,0151 \times 10^1
 \end{array}
 \Rightarrow \text{Allineo alla potenza minore (forma normalizzata)}$$



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:

$$\begin{array}{r} 79,99 \\ + \\ 0,161 \\ \hline 80,151 \times 10^0 \end{array}$$

$$\begin{array}{r} 799,9 \\ + \\ 1,61 \\ \hline 801,51 \times 10^{-1} \end{array}$$

$$\begin{array}{r} 7,999 \\ + \\ 0,0161 \\ \hline 8,0154 \times 10^1 \end{array}$$

La rappresentazione **migliore** è:

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

Risultato normalizzato

$$8,0154 \times 10^1$$

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 -> **Perdo 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).

A.A. 2024-2025

19/38

<http://borghese.di.unimi.it/>



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.

A.A. 2024-2025

20/38

<http://borghese.di.unimi.it/>



Esempio: aritmetica in floating point accurata



$$a = 2,34 \quad b = 2,56 \cdot 10^{-2} = 0,0256$$

prima e 2

$$a + b = ?$$

Codifica su 3 cifre decimali totali (1 dopo la virgola).

Approssimazione mediante **rounding**.

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

$$\begin{array}{r} 2,34 \\ + \\ 0,0256 = \\ \hline 2,36 \end{array}$$

ho troncato il secondo addendo per rimanere nella capacità

Con le cifre di guardia e di arrotondamento posso scrivere:

$$\begin{array}{r} 2,3400 \\ + \\ 0,0256 = \\ \hline 2,3656 \end{array}$$

L'arrotondamento finale (round) viene effettuato **sul risultato** per rientrare in 3 cifre decimali fornisce: **$2,37 \neq 2,36$**



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:

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

$\Rightarrow C > A?$ Yes! E' corretto.

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

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

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

$\Rightarrow C = A$ **Errore!!! Il test dà risultato errato!!! E il codice seguente è errato.**

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



Non vale la proprietà associativa della somma



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

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

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

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

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

Risultati molto diversi.

Non vale la proprietà associativa!



Problemi di arrotondamento – IEEE 754



$$A = 4$$

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

$$Z = B + A = 5,0000003576278686523438$$

In IEEE754:

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

$$B = 1,00000\ 00000\ 00000\ 00000\ 011 \times 2^0 \quad \text{parte frazionaria su 23 bit}$$

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

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

Allineo B ad A (shift B a dx di 2 posizioni): \Rightarrow

$$B = 0, 01000\ 00000\ 00000\ 00000\ 000 \times 2^2 \quad \text{parte frazionaria su 23 bit}$$

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

$$\begin{array}{rcl} B & 0,01000\ 00000\ 00000\ 00000\ 000 & \text{su 23 bit} \\ A & 1,00000\ 00000\ 00000\ 00000\ 000 & = \text{su 23 bit} \end{array} \quad \text{Base} = 2^2$$

$$\begin{array}{rcl} \hline Z=B+A & 1,01000\ 00000\ 00000\ 000 & \text{su 23 bit} \\ C = B+A & = 5 & \Rightarrow \text{errore!} \quad \text{Può essere pericoloso!} \end{array} \quad \text{Base} = 2^2$$



Problemi di troncamento – IEEE 754



A = 4

B = 1,0000003576278686523438 x 10⁰

Z = B + A = 5,0000003576278686523438

In IEEE754:

A = 1 x 2²

B = 1, 00000 00000 00000 00000 011 x 2⁰ 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 : 2²):

$$\begin{array}{rcl} B & 0,01000 00000 00000 00000 00011 + & \text{su 25 bit} & \text{Base} = 2^2 \\ A & 1,00000 00000 00000 00000 00000 = & \text{su 25 bit} & \\ \hline Z & 1,01000 00000 00000 00000 0001\textcircled{1} & \text{su 25 bit} & \text{Base} = 2^2 \end{array}$$

Per rounding, Z = 1,01000 00000 00000 0001\textcircled{0} su 23 bit Base = 2²

$$C = A+B = 5 + 2^{-23}2^2 = 5,000000476837158203125$$

molto più vicino al valore vero di quanto era 5,000000000000000000000!

<http://borgheze.di.unimi.it/>

A.A. 2024-2025

25/38



Algoritmo di somma in virgola mobile - I



- 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).

$$\begin{array}{ll} a = 9,12 \times 10^0 & b = 8,99 \times 10^{-1} \\ \downarrow & \downarrow \\ a = 9,12 \times 10^0 & b = 0,899 \times 10^0 \end{array}$$

- Effettuare la somma delle mantisse.

$$9,12 +$$

$$0,899 =$$

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

Se il numero risultante è normalizzato termino qui. Altrimenti:

- Normalizzare il risultato.

$$10,019 \times 10^0 \rightarrow 1,0019 \times 10^1$$

A.A. 2024-2025

26/38

<http://borgheze.di.unimi.it/>



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

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

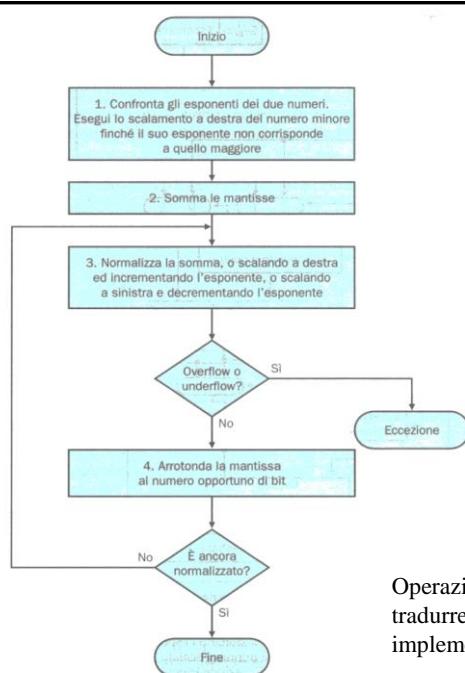
2) Somma delle mantisse:

$$\begin{array}{r} 9,999 \\ 0,0161 \\ \hline 10,015 \end{array} \quad \begin{array}{l} + \\ \text{Perdo una cifra perchè non rientra nella capacità della mantissa (troncamento)} \\ \hline 10,015 \times 10^1 \end{array}$$

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):

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

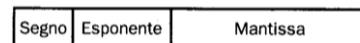
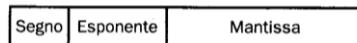


Algoritmo risultante

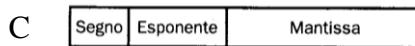
Operazioni complesse da
tradurre in operazioni
implementabili dall'hardware



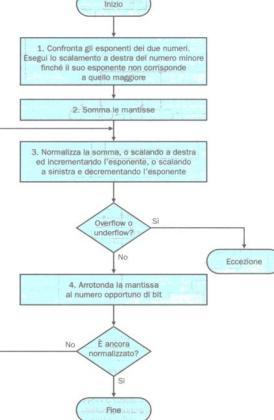
Il circuito della somma floating point: gli attori



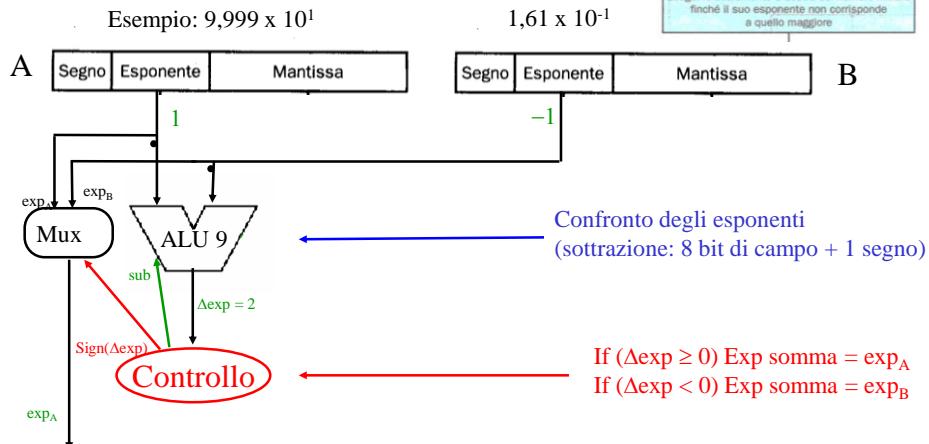
A + B



Rappresentazione normalizzata IEEE754



Determinazione dell'esponente



Occorre determinare la mantissa

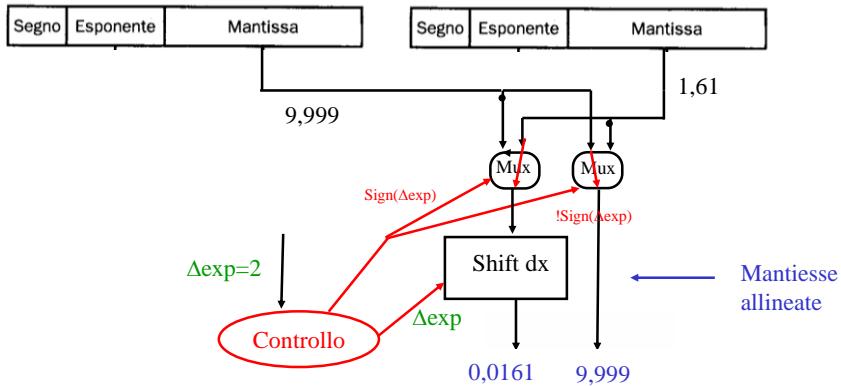


Allineamento mantisse

Esempio: $9,999 \times 10^1$

$1,61 \times 10^{-1}$

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



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

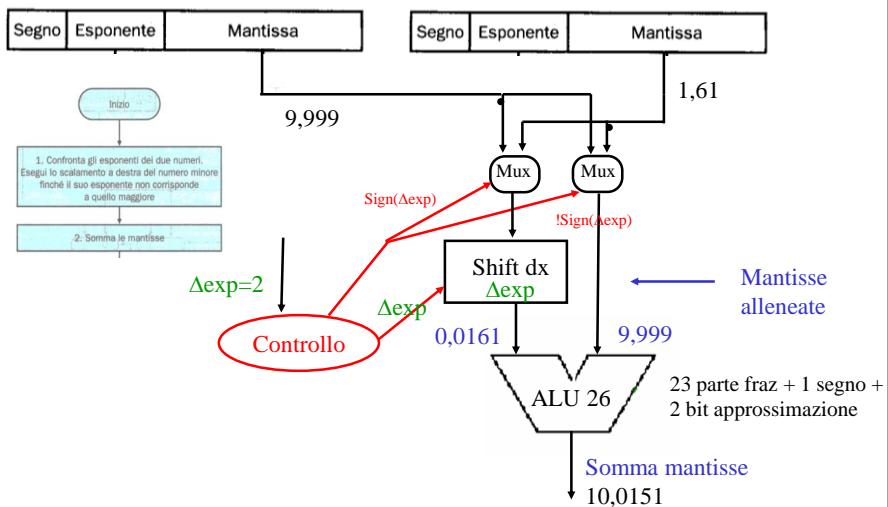
1b) $1,61 \times 10^{-1} \Rightarrow 0,0161 \times 10^1 \Rightarrow \Delta exp = +2$



Somma delle mantisse

Esempio: $9,999 \times 10^1$

$1,61 \times 10^{-1}$



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

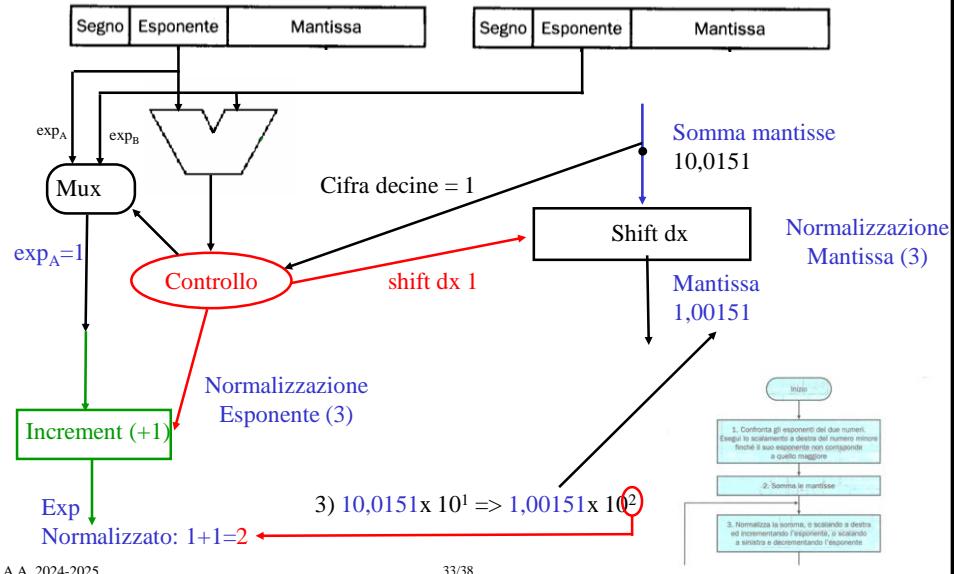


Normalizzazione



Esempio: $9,999 \times 10^1$

$1,61 \times 10^{-1}$



A.A. 2024-2025

33/38

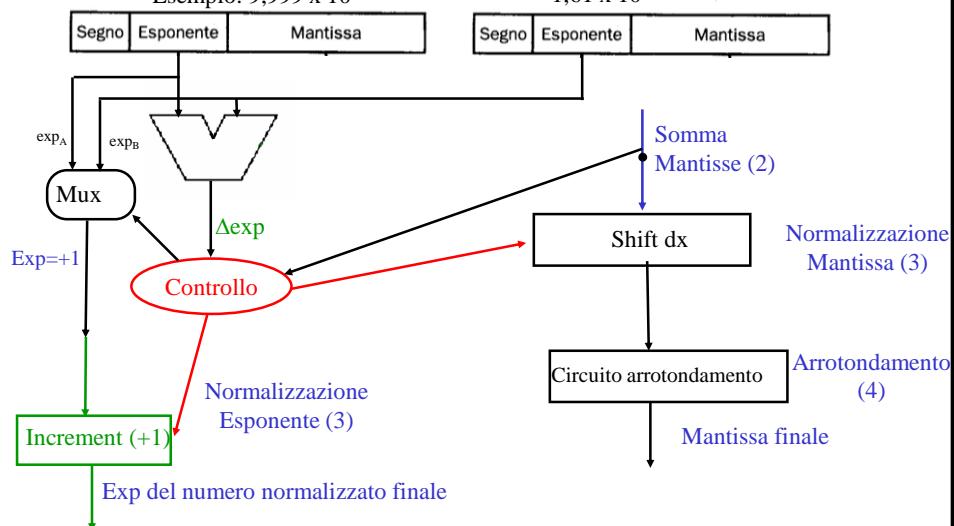


Il circuito della somma floating point: arrotondamento e normalizzazione



Esempio: $9,999 \times 10^1$

$1,61 \times 10^{-1}$



A.A. 2024-2025

34/38

<http://borgheze.di.unimi.it/>



Circuito della somma floating point



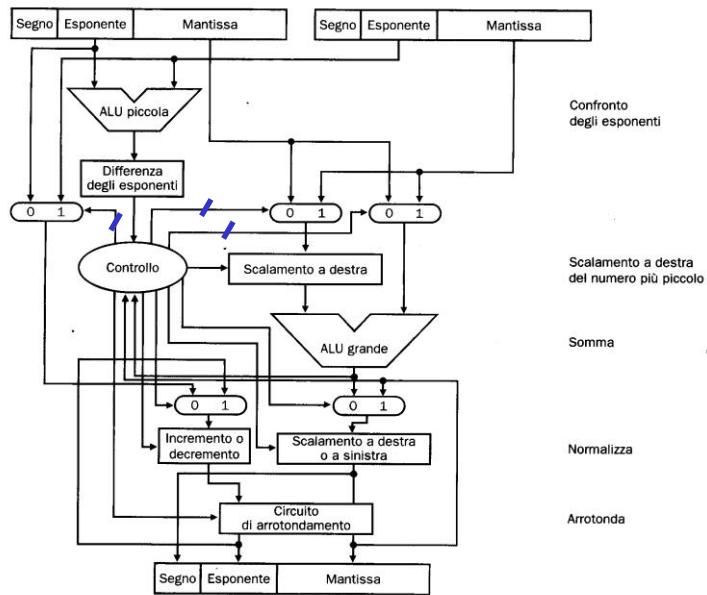
Le tre linee in blu contengono lo stesso segnale di controllo (funzione di Δ_{exp}).

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

Gestisce anche i numeri negativi.

Problemi?

A.A. 2024-2025



Circuito della somma floating point con bit di arrotondamento



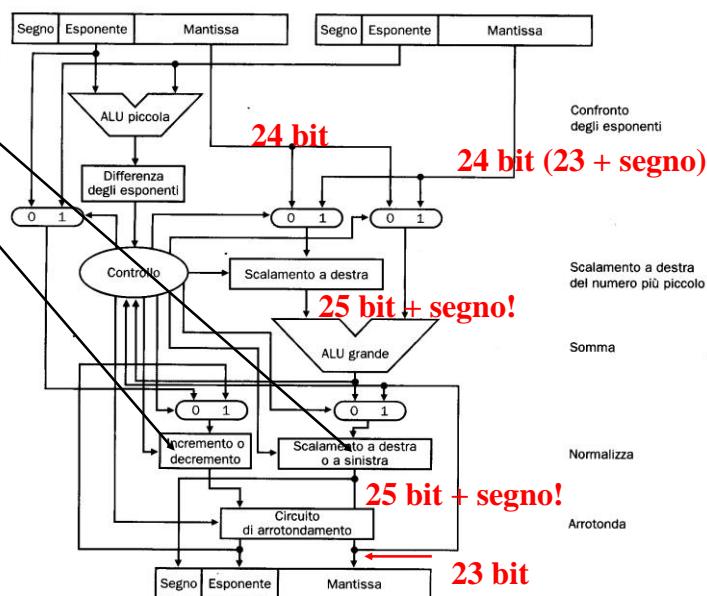
In quale caso la mantissa viene scalata a sx?

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. 2024-2025





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