



UC firmware moltiplicazione Floating pointer adder

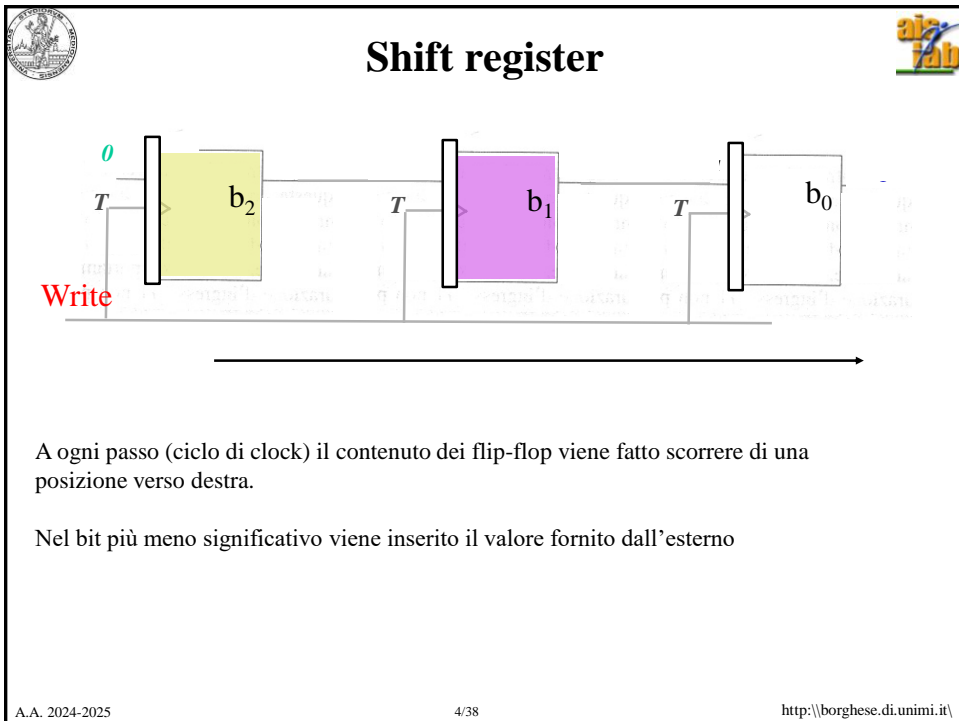
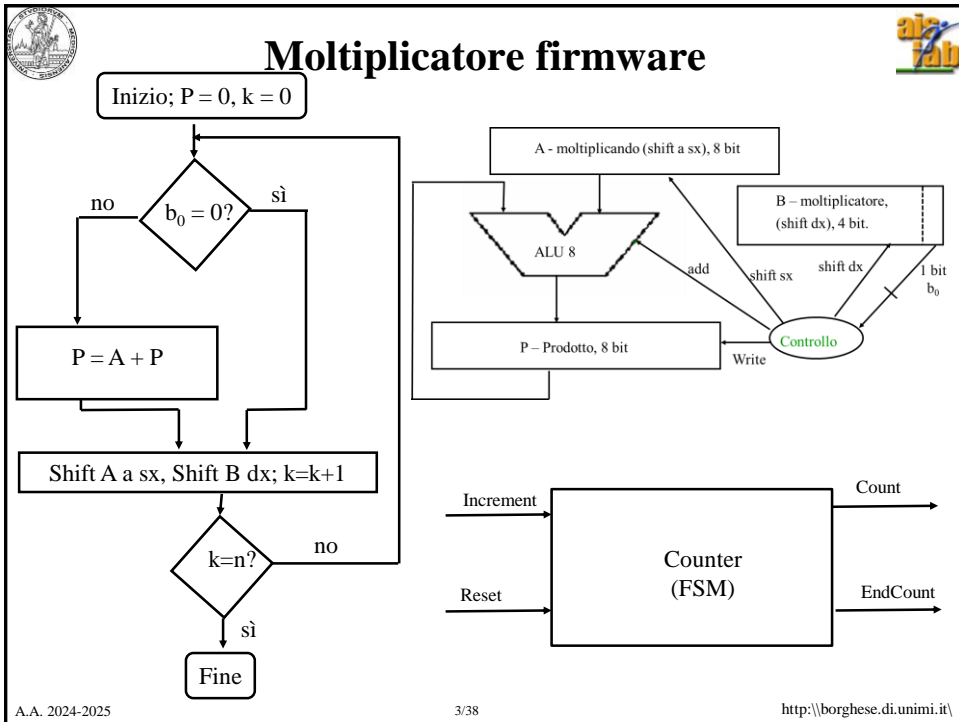
Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

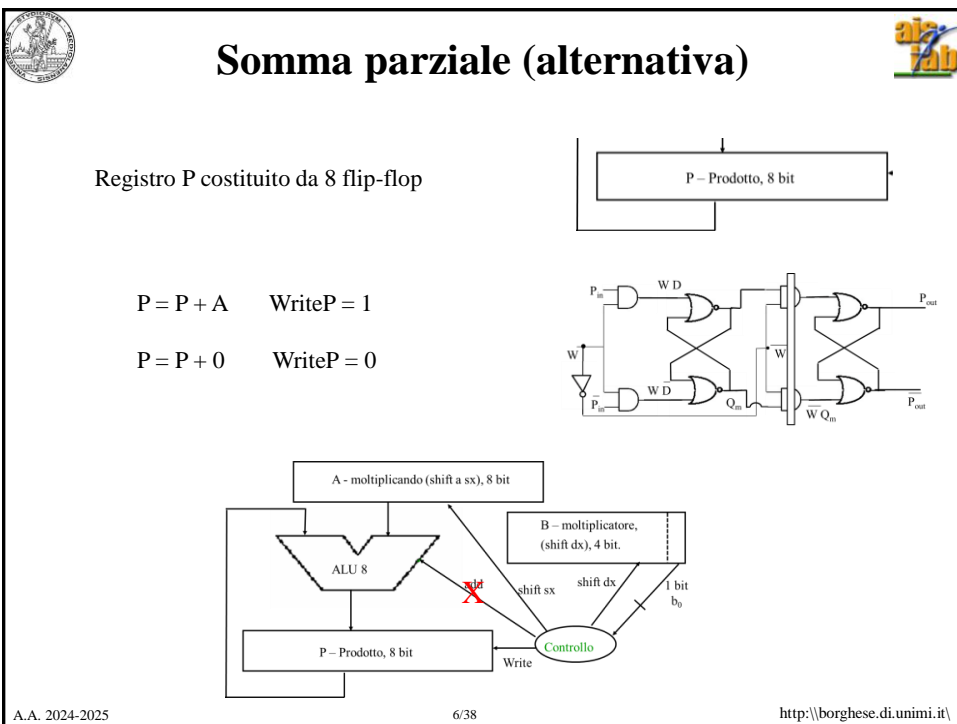
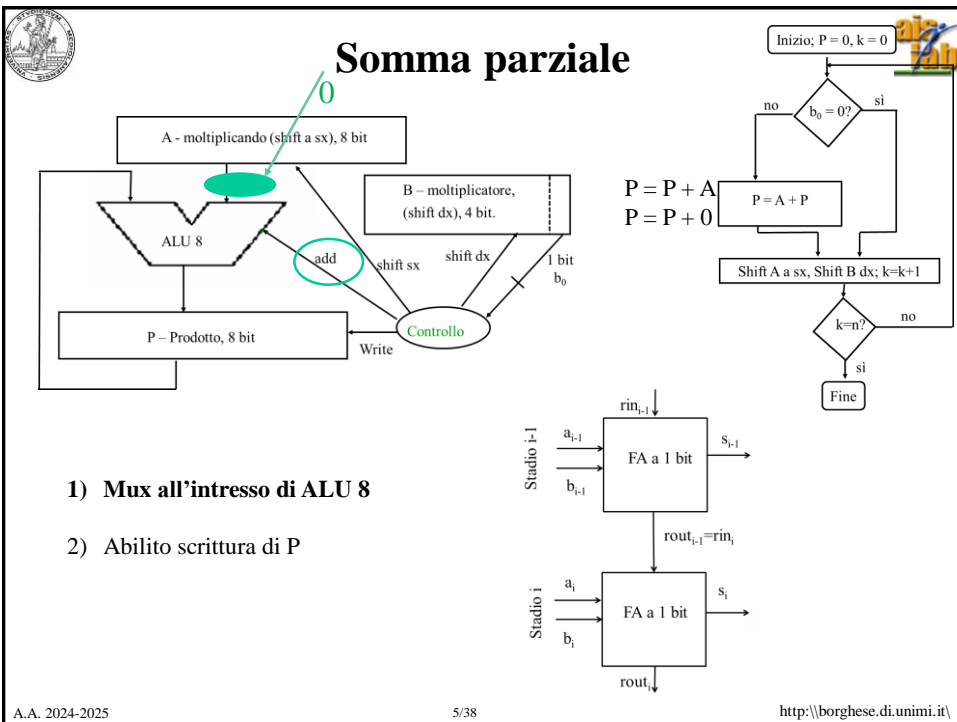
Università degli Studi di Milano
Riferimenti sul Patterson, 6a Ed.: 3.4, 3.5, 4.2



Sommario

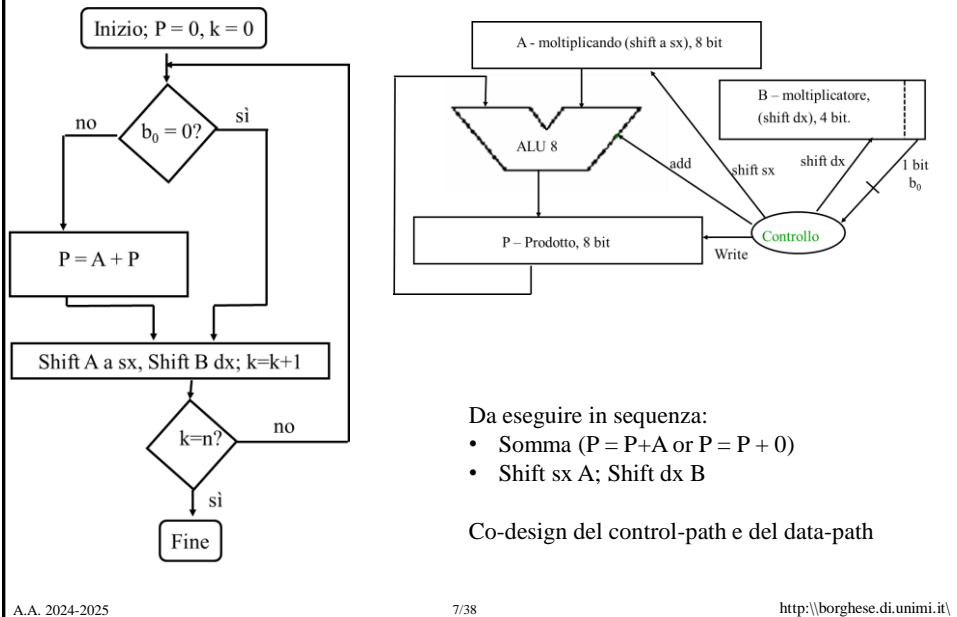
- Unità di controllo del firmware
- Somma in virgola mobile







Operazioni elementari



STG – S₀



Macchina a stati finiti:

{X} = {initCount,

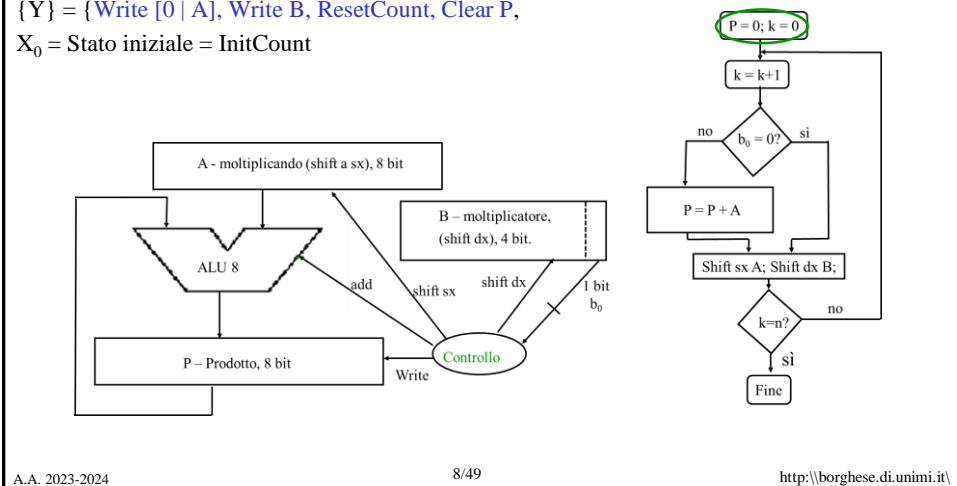
{I} = {

{Y} = {Write [0 | A], Write B, ResetCount, Clear P,

X₀ = Stato iniziale = InitCount

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

S₀ / InitCount





STG – S₁

Macchina a stati finiti:

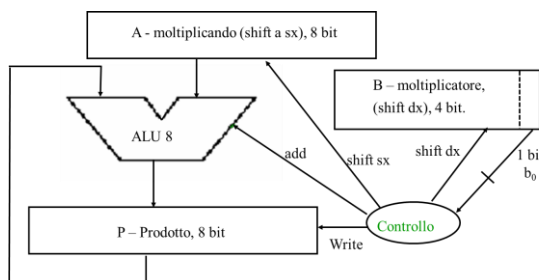
Write [0 | A], Write B
ResetCount, Clear P

{X} = {initCount, **Activate**,

{I} = {

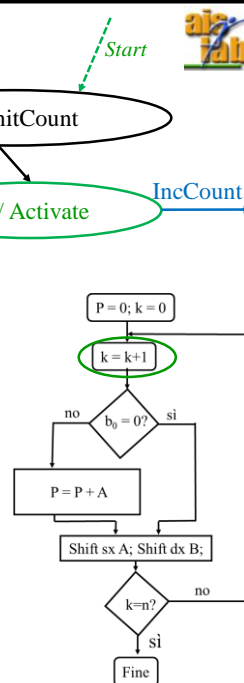
{Y} = {Write [0 | A], Write B, ResetCount, Clear P,
IncCount,

X₀ = Stato iniziale = InitCount



A.A. 2023-2024

9/49



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



STG – S₂

Macchina a stati finiti:

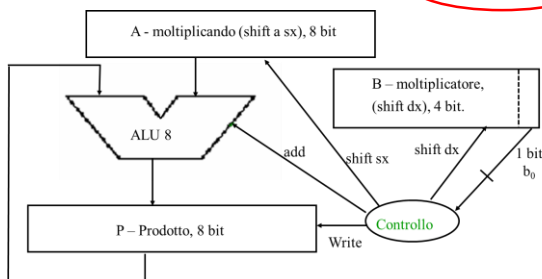
Write [0 | A], Write B
ResetCount, Clear P

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

{I} = {b₀,

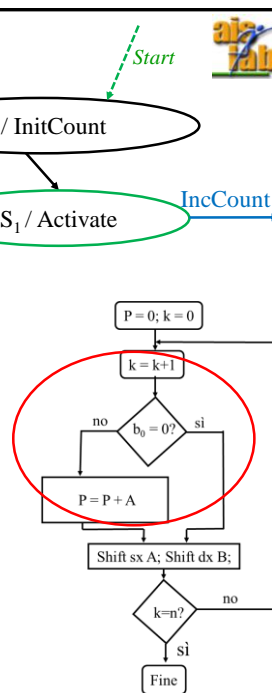
{Y} = {Write [0 | A], Write B, ResetCount, Clear P,
IncCount, **add**, **Write P**,

X₀ = Stato iniziale = InitCount

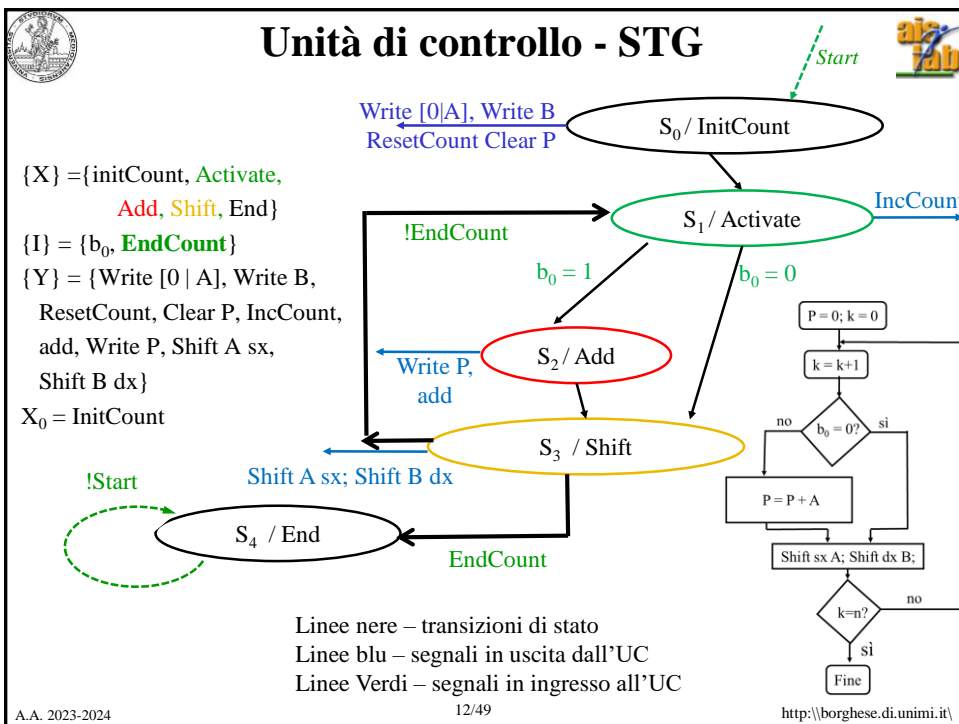
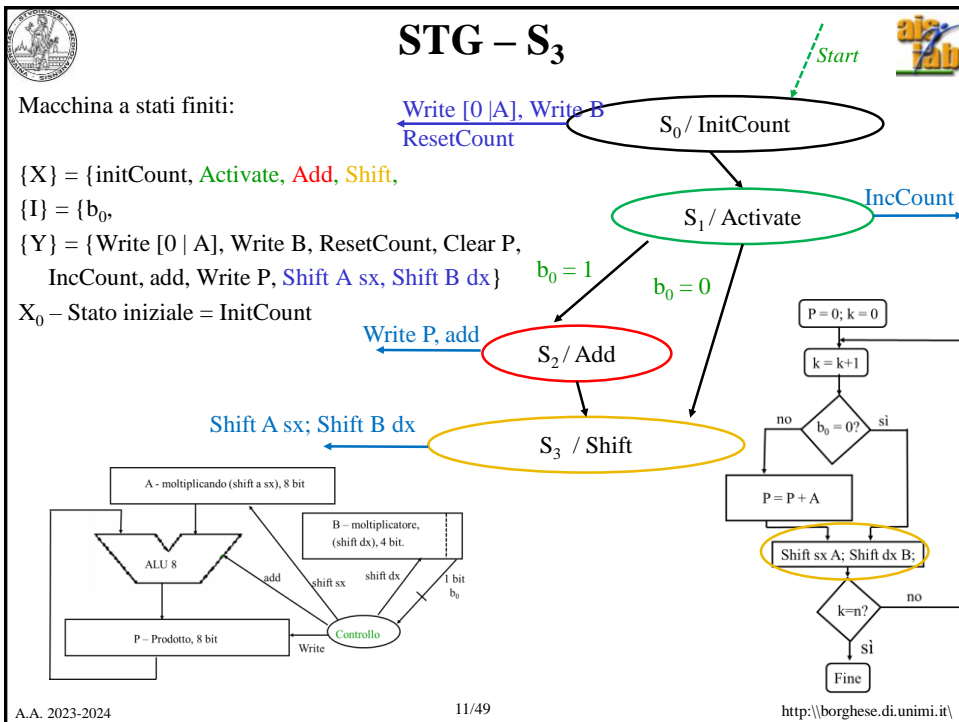


A.A. 2023-2024

10/49



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





Macchina di Huffman

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

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

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

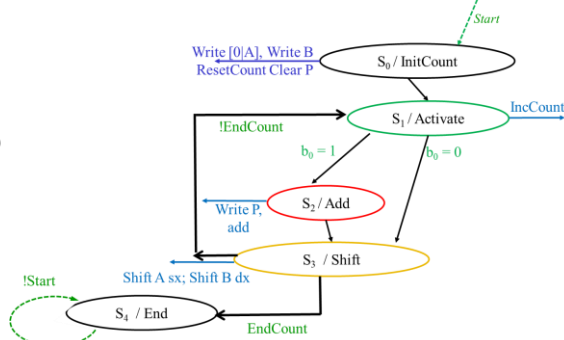
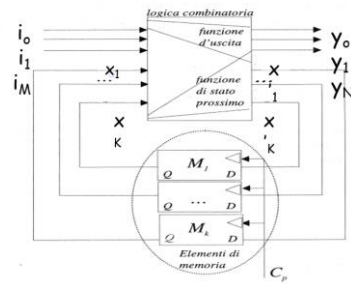
$X_0 = \text{InitCount}$

$M = 2$ - 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



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

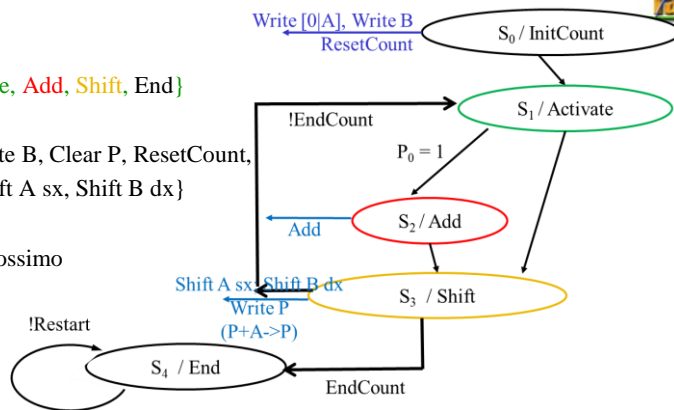
$\{I\} = \{b_0, \text{EndCount}\}$

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

$X_0 = \text{InitCount}$

$f(X, I)$ - Funzione stato prossimo

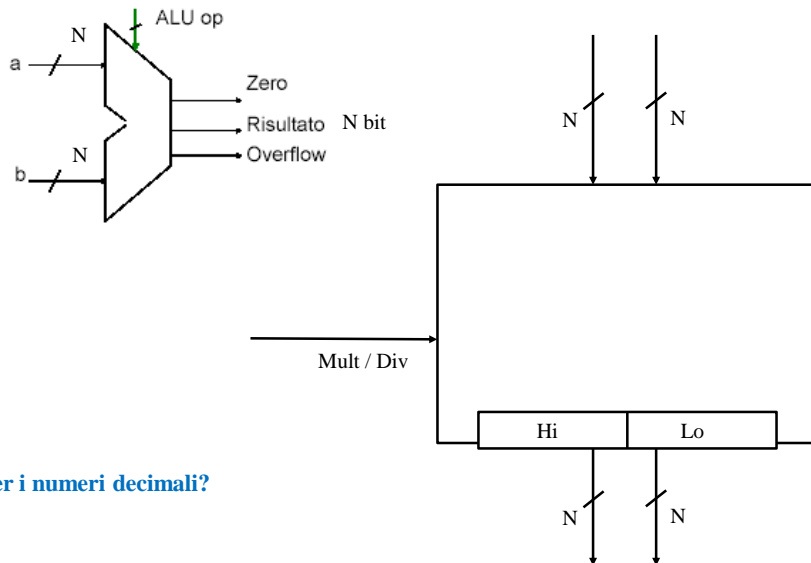
$g(X)$ - Funzione di uscita



	!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



E per i numeri decimali?



Sommario



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



Codifica in virgola mobile Standard IEEE 754 (1980)

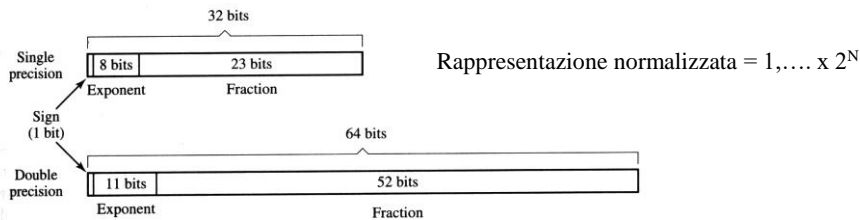


Figure 2-10 Single-precision and double-precision IEEE 754 floating point formats.

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:

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

$$\text{-----}$$

$$80,151 \times 10^0 = 80,151 \Rightarrow \text{Allineo all'unità}$$

Altre possibilità sono:

$$\begin{array}{rcl} 799,9 & + & 7,999 \\ 1,61 & = & 0,0161 \end{array}$$

$$\text{-----}$$

$$801,51 \times 10^{-1} = 8,0151 \times 10^1 \Rightarrow \text{Allineo alla potenza maggiore}$$

$$\text{-----}$$

$$8,0151 \times 10^1 \Rightarrow \text{Allineo alla potenza minore (forma normalizzata)}$$



Quale forma conviene utilizzare?



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

$$a + b = ?$$

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

$$\begin{array}{r} 79,99 \quad + \\ 0,161 = \\ \hline \end{array}$$

$$80,151 \times 10^0$$

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

$$801,51 \times 10^{-1}$$

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

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

La rappresentazione **migliore** è:

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

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): $8,0151 \rightarrow 8,015$
- Arrotondamento alla cifra superiore (ceil): $8,0151 \rightarrow 8,016$
- Arrotondamento alla cifra più vicina: (round) $8,0151 \rightarrow 8,015$

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



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

2,34 +

0,0256 = ho troncato il secondo addendo per rimanere nella capacità

2,36

Con le cifre 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 \neq 2,36**



L'effetto perverso del troncamento



$C = A + B$

if (**C > A**) then

(a)...

else

(b)....

$A = 7,999 \times 10^1$ $B = 1,61 \times 10^{-1}$

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

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

$A = 7,999 \times 10^1$ $B = 1,61 \times 10^{-1}$
 $\Rightarrow C > A$? Yes! E' corretto.

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

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

$C = A + B = 7,999161$

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

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

$C = A + B = (7,999 + 0,0000161) \times 10^1 = 7,999$

$\Rightarrow C = A$ 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:

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

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

$$A = 0 \quad 1000000100000000000000000000000 \quad \text{codifica IEEE754 su 32 bit}$$

$$B = 0 \quad 1111111100000000000000000000011 \quad \text{codifica IEEE754 su 32 bit}$$

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

$$B = 0,01000000000000000000000011 \times 2^2 \quad \text{parte frazionaria su 23 bit}$$

$$\text{Segue che:} \quad \Rightarrow \quad C = A + B = 1,01000000000000000000000011 \times 2^2 \quad \text{su 23 bit}$$

$$B \quad 0,01000000000000000000000011 \quad \text{su 23 bit} \quad \text{Base} = 2^2$$

$$A \quad 1,000000000000000000000000 \quad \text{su 23 bit}$$

$$Z=B+A \quad 1,010000000000000000000000 \quad \text{su 23 bit} \quad \text{Base} = 2^2$$

$$C = B+A = 5 \Rightarrow \text{errore!} \quad \text{Può essere pericoloso!}$$



Problemi di troncamento – IEEE 754



$$A = 4$$

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

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

In IEEE754:

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

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

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

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

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

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

$$\begin{array}{rcl} Z & 1,01000\ 00000\ 00000\ 00000\ 00011 & \text{su 25 bit} \quad \text{Base} = 2^2 \end{array}$$

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

$$C = A+B = 5 + 2^{-23}2^2 = 5,0000000476837158203125$$

molto più vicino al valore vero di quanto era 5,00000000000000000000000!



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).

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

- 2) Effettuare la somma delle mantisse.

$$\begin{array}{r} 9,12 + \\ 0,899 = \\ \hline 10,019 \times 10^0 \end{array}$$

Se il numero risultante è normalizzato termino qui. Altrimenti:

- 3) Normalizzare il risultato.

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



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:

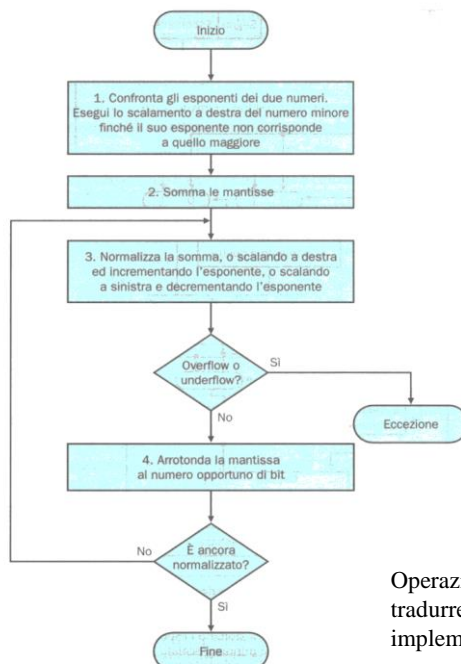
$$9,999 \quad +$$

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

$$10,015 \times 10^1 \quad \text{Il risultato non è più normalizzato, anche se i due addendi sono normalizzati.}$$

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

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

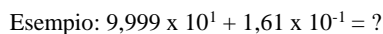
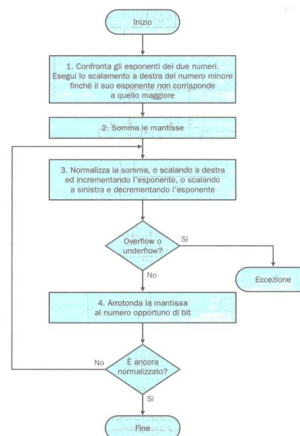


Algoritmo risultante

Operazioni complesse da
tradurre in operazioni
implementabili dall'hardware

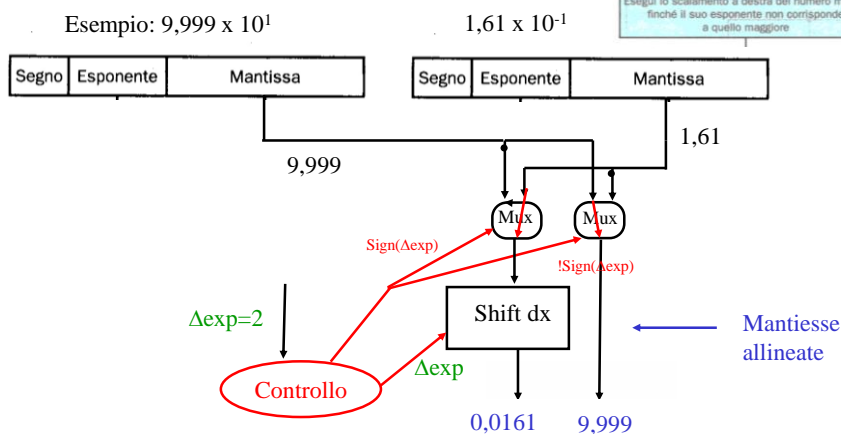


Rappresentazione normalizzata IEEE754





Allineamento mantisse

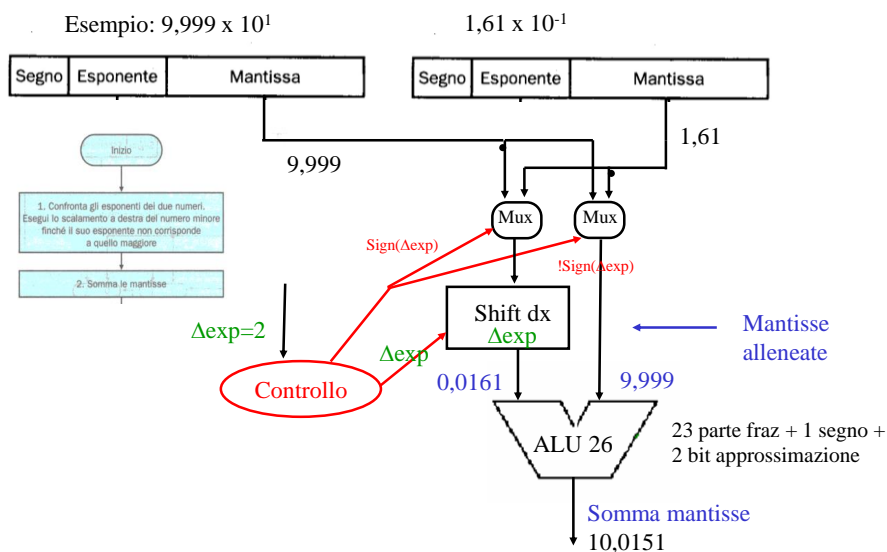


Esempio: $9,999 \times 10^1 + 1,61 \times 10^{-1} = ?$

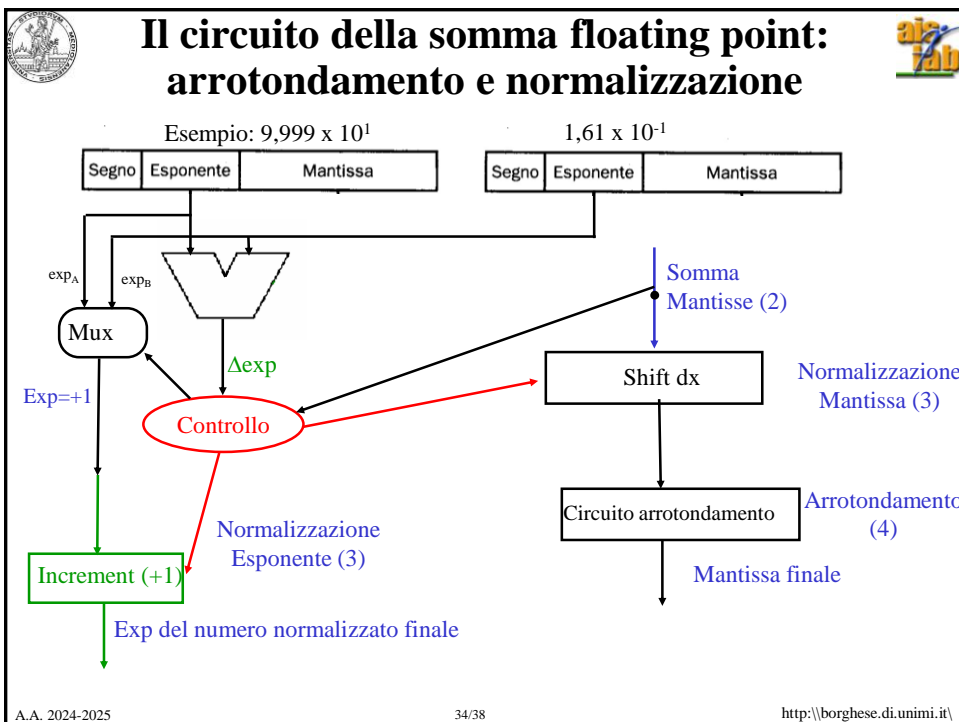
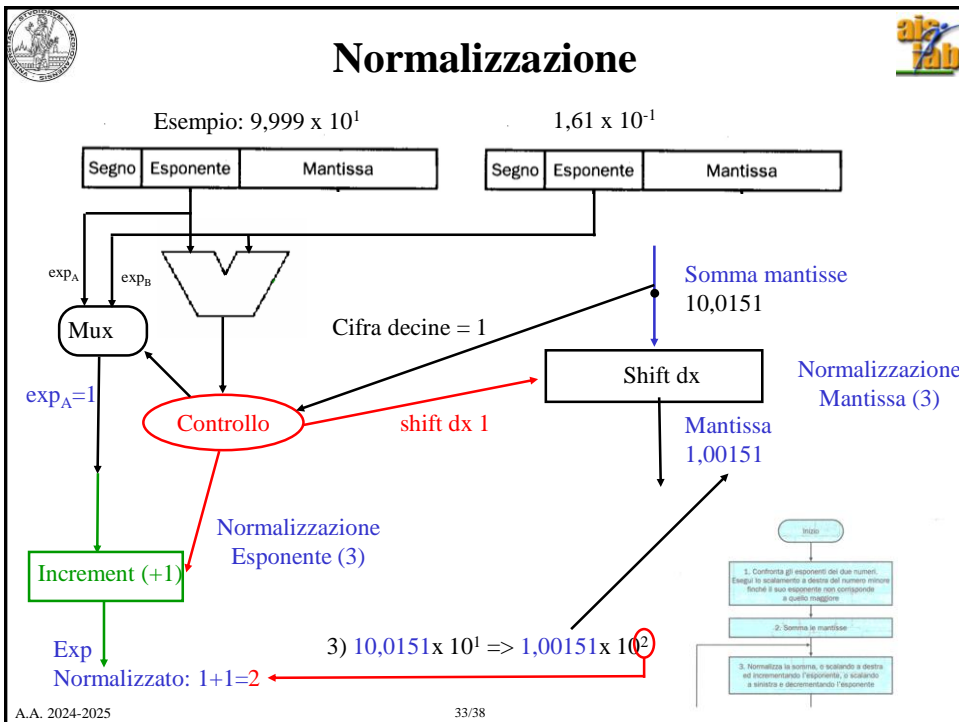
1b) $1,61 \times 10^{-1} \Rightarrow 0,0161 \times 10^1 \Rightarrow \Delta\text{exp} = +2$



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



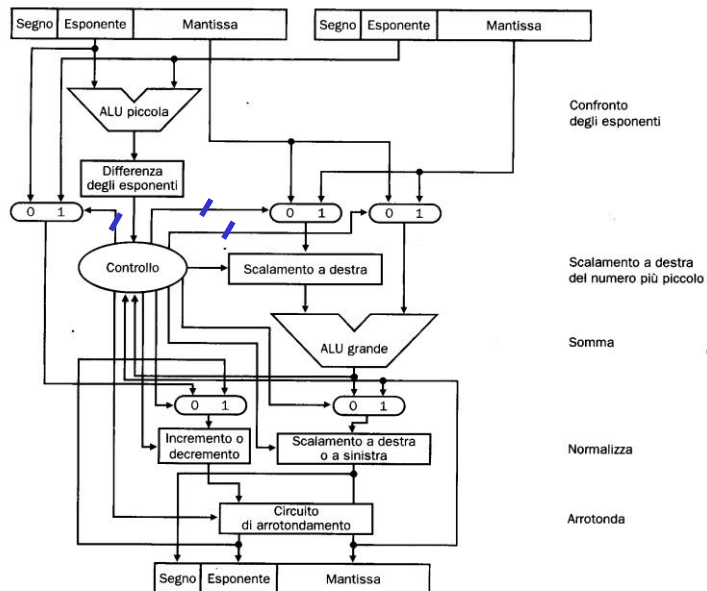
Le tre linee in blu contengono lo stesso segnale di controllo (funzione di Δexp).

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

Gestisce anche i numeri negativi.

Problemi?

A.A. 2024-2025



Circuito della somma floating point con bit di arrotondamento



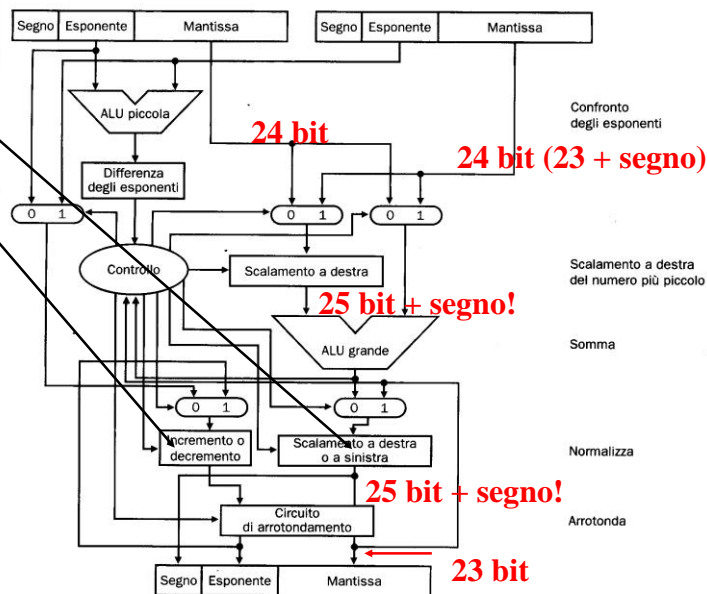
In quale caso la mantissa viene scalata a sx?

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