



# Floating pointer adder & Firmware Division

Prof. Alberto Borghese  
Dipartimento di Scienze dell'Informazione  
[borgnese@dsi.unimi.it](mailto:borgnese@dsi.unimi.it)

Università degli Studi di Milano

Riferimenti sul Patterson: 3.4, 3.5



## Sommario

- **Divisione intera**
- Somma in virgola mobile



## La divisione decimale



$$\begin{array}{r}
 \text{Dividendo} \quad \text{Divisore} \\
 \begin{array}{r}
 \text{---} \\
 2516 : 12 = 209 \\
 \underline{116} \\
 8 \\
 \text{Resto}
 \end{array}
 \end{array}$$

Quoziente

$$\text{Dividendo} = \text{Divisore} * \text{Quoziente (quoto)} + \text{Resto}$$



## La divisione decimale::algoritmo



$$\text{Passo 1)} \quad \begin{array}{r} \text{---} \\ 2516 : 0012 = 0209 \end{array}$$

Il 12 nel 2 ci sta 0 volte.

$$\begin{array}{r} -0 \\ \text{---} \\ 2 \end{array}$$

$$12 \times 0 = 0$$

2

### Resto parziale - I

$$\text{Passo 2)} \quad 25$$

Considero il 5 del divisore e lo aggiungo al resto parziale. Il

12 nel 25 ci sta 2 volte.

$$\begin{array}{r} -24 \\ \text{----} \\ 1 \end{array}$$

$$12 \times 2 = 24$$

1

### Resto parziale - II

$$\text{Passo 3)} \quad 11$$

Considero l'1 del dividendo e lo aggiungo al resto parziale.

Il 12 nell'11 ci sta 0 volte.

$$\begin{array}{r} -0 \\ \text{----} \\ 11 \end{array}$$

$$12 \times 0 = 0$$

11

### Resto parziale - III

$$\text{Passo 4)} \quad 116$$

Considero il 6 del dividendo e lo aggiungo al resto parziale.

Il 12 nel 16 ci sta 1 volta.

$$\begin{array}{r} -108 \\ \text{-----} \\ 8 \end{array}$$

$$12 \times 9 = 108$$

8

Resto parziale - IV = RESTO



## La divisione tra numeri binari



Divisione decimale fra i numeri  $a = 1.001.010$  e  $b = 1.000$   $a : b = ?$

1001010	:	1000	=	1	Dividendo : Divisore	74	:	8	=	9	resto	2
1000-												
----												
1												

1001010	:	1000	=	1001	Quoziente
1000-					
----					
1010					Resto parziale
1000-					
----					
10					Resto

$$\text{Dividendo} = \text{Quoziente} \times \text{Divisore} + \text{Resto}$$



## Confronto tra divisione tra numeri binari e decimali



Definisco il resto parziale della divisione. Questo è coincidente all'inizio con il dividendo.

In DECIMALE:

Ad ogni passo devo verificare QUANTE VOLTE il resto parziale contiene il divisore. Il risultato è un numero che va da 0 a 9 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

In BINARIO:

Ad ogni passo devo verificare SE il resto parziale contiene il divisore. Ovverosia se lo contiene 0 o 1 volta. Il risultato è un numero che può valere {0, 1}.

In DECIMALE:

Il numero che viene sottratto al resto parziale è ottenuto moltiplicando il divisore per una delle cifre da 0 a 9.

In BINARIO:

Il numero che viene sottratto al resto parziale può essere solamente 0 o il DIVISORE stesso.



## Peculiarità della divisione



- Come rappresento la condizione “il divisore è contenuto nel resto parziale”?

Esempi:

10 (resto parziale) non è contenuto in 1000 (divisore)

1001 (resto parziale è contenuto in 1000 (divisore)

Per tentativo.

Eseguo la sottrazione.

Risultato  $\geq 0 \rightarrow$  il resto parziale è contenuto nel divisore.

Risultato  $< 0 \rightarrow$  il resto parziale non è contenuto nel divisore (è più piccolo del divisore).

*Osserviamo che nel secondo caso abbiamo fatto in realtà una sottrazione che non avremmo dovuto effettuare.*

NB il calcolatore non può sapere se il resto parziale contiene il divisore fino a quando non ha effettuato la sottrazione.



## Note e strategia di implementazione



I bit del didendo vengono analizzati da sx a dx. Il divisore viene spostato verso dx di 1 bit ad ogni passo.

Il quoziente cresce dal bit più significativo verso il bit meno significativo. Cresce verso dx.

All'inizio il divisore è allineato alla sinistra del dividendo: le cifre del dividendo sono allineate agli 0 del divisore e gli 0 del dividendo sono allineati alle cifre del divisore.

Ci sono  $N+1$  passi di divisione, il primo darà sempre 0 e si potrebbe omettere.

Occorre quindi effettuare:

Shift quoziente a sx ad ogni passo.

Scrittura di 1 o 0 nel registro quoziente.

Shift del divisore verso dx ad ogni passo..

Utilizzo un unico registro per dividendo e resto, considerando che il primo resto parziale è uguale al dividendo.



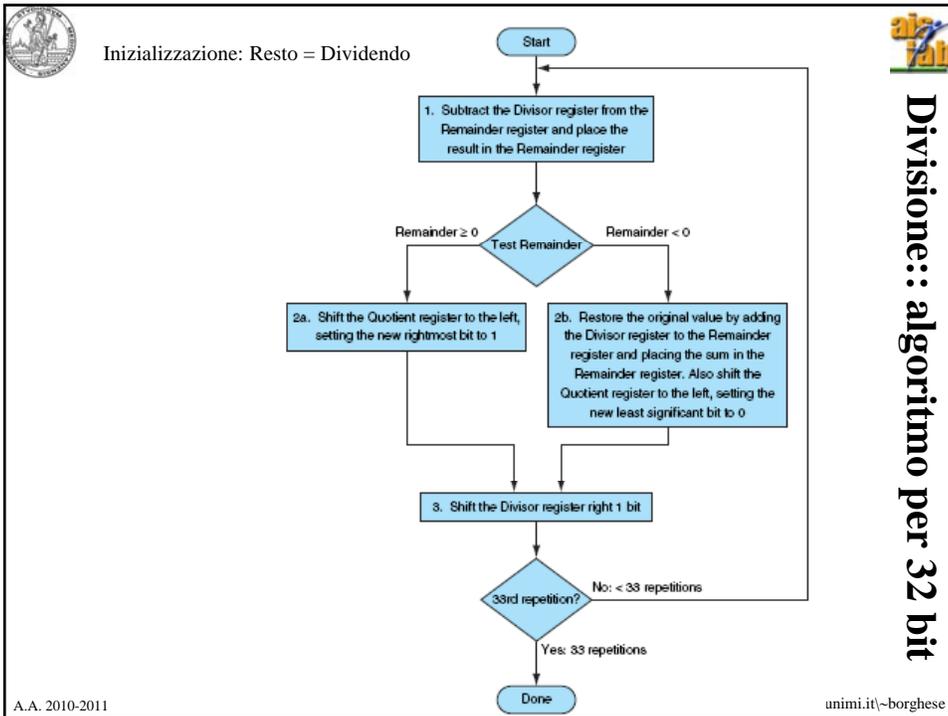
Inizializzazione: Resto = Dividendo



# Divisione: algoritmo per 32 bit

A.A. 2010-2011

unimi.it/~borghese



## Il circuito firmware della divisione

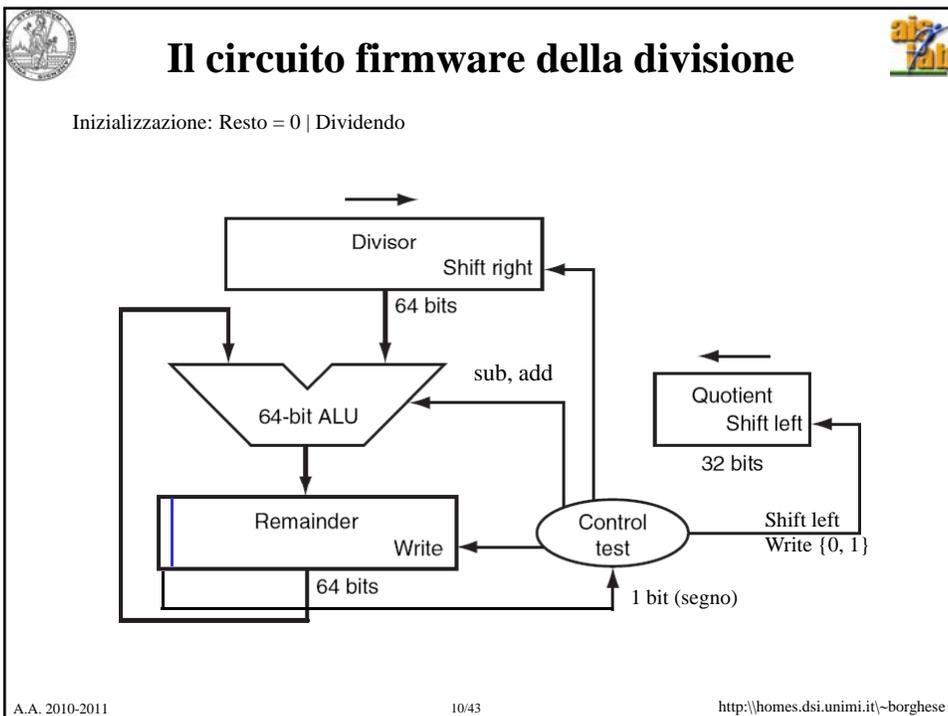


Inizializzazione: Resto = 0 | Dividendo

A.A. 2010-2011

10/43

http://homes.dsi.unimi.it/~borghese





## Esempio



Divisione decimale fra i numeri  $a = 7$  e  $b = 2$   $a : b = ?$

Inizializzo il divisore alla sinistra delle quattro cifre significative. La prima cifra del quoziente sarà sempre 0.

Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem - Div	0000	0010 0000	0110 0111
	2b: Rem < 0 $\Rightarrow$ +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem - Div	0000	0001 0000	0111 0111
	2b: Rem < 0 $\Rightarrow$ +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem - Div	0000	0000 1000	0111 1111
	2b: Rem < 0 $\Rightarrow$ +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem - Div	0000	0000 0100	0000 0011
	2a: Rem $\geq$ 0 $\Rightarrow$ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem - Div	0001	0000 0010	0000 0001
	2a: Rem $\geq$ 0 $\Rightarrow$ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001



## Esempio - step 1



Il resto parziale = dividendo: 0000 0111

```

—
0000 0111 : 0010 =
-0010
-----
<0

```

Il divisore (0010) non è contenuto nel resto parziale (00 0111)

Osservando i registri, in pratica, eseguiamo la differenza tra:

```

0000 0111 -      7      - Resto parziale
0010 0000 =      32 = 2*2^4 - Divisore allineato al di fuori e alla sx dei 4 bit del dividendo.
-----
1110 0111 -2510 < 0

```

Quoziente al passo 1: 0



## Esempio – step 2



Il resto parziale è ancora uguale al dividendo: 0000 0111

$$\begin{array}{r}
 \text{---} \\
 0000 \text{ 0111} : 0010 = \\
 -001 0 \\
 \text{-----} \\
 <0
 \end{array}$$

Il divisore (0010) non è contenuto nel resto parziale (0 0111)

Osservando i registri, in pratica, eseguiamo la differenza tra:

$$\begin{array}{r}
 0000 \text{ 0111} - \quad 7 \quad - \text{ Resto parziale} \\
 0001 \text{ 0000} = \quad 16 = 2 * 2^3 - \text{ Divisore allineato al MSB dei 4 bit del dividendo.} \\
 \text{-----} \\
 1111 \text{ 0111} \quad -9_{10} < 0
 \end{array}$$

Quoziente al passo 2: 00



## Esempio – step 3



Il resto parziale è ancora uguale al dividendo: 0000 0111

$$\begin{array}{r}
 \text{---} \\
 0000 \text{ 0111} : 0010 = \\
 -00 10 \\
 \text{-----} \\
 <0
 \end{array}$$

Il divisore (0010) non è contenuto nel resto parziale (0111)

Osservando i registri, in pratica, eseguiamo la differenza tra:

$$\begin{array}{r}
 0000 \text{ 0111} - \quad 7 \quad - \text{ Resto parziale} \\
 0000 \text{ 1000} = \quad 8 = 2 * 2^2 - \text{ Divisore allineato al terzo dei 4 bit del dividendo.} \\
 \text{-----} \\
 1111 \text{ 1111} \quad -1_{10} < 0
 \end{array}$$

Quoziente al passo 3: 000



## Esempio – step 4



Il resto parziale è ancora uguale al dividendo: 0000 0111

$$\begin{array}{r}
 \text{---} \\
 0000 \text{ 0111} : 0010 = \\
 -0 \text{ 010} \\
 \text{-----} \\
 0 \text{ 001} \quad > 0
 \end{array}$$

Il divisore (0010) è contenuto nel resto parziale (111)

Osservando i registri, in pratica, eseguiamo la differenza tra:

$$\begin{array}{r}
 0000 \text{ 0111} - \quad \text{Resto parziale} \\
 0000 \text{ 0100} = \quad \text{Divisore allineato al secondo dei 4 bit del dividendo.} \\
 \text{-----} \\
 0000 \text{ 0011} \quad 3_{10} \quad \text{E' il nuovo resto parziale } 7 - 1 * 2^2 = 3
 \end{array}$$

Quoziente al passo 4: 0001



## Esempio – step 5



Il resto parziale = 0000 0011

$$\begin{array}{r}
 \text{---} \\
 0000 \text{ 0111} : 0010 = \\
 -0010 \\
 \text{-----} \\
 0001 \quad > 0
 \end{array}$$

Il divisore (0010) è contenuto nel resto parziale (11)

Osservando i registri, in pratica, eseguiamo la differenza tra:

$$\begin{array}{r}
 0000 \text{ 0011} - \quad \text{Resto parziale} \\
 0000 \text{ 0010} = \quad \text{Divisore allineato LSB dei 4 bit del dividendo.} \\
 \text{-----} \\
 0000 \text{ 0001} \quad 1_{10} \quad \text{E' il nuovo resto parziale } 3 - 1 * 2^1 = 1 \quad \text{RESTO FINALE}
 \end{array}$$

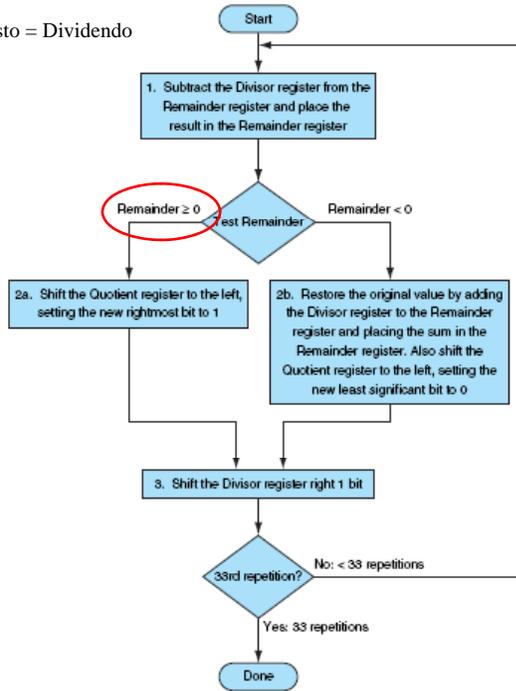
Quoziente al passo 5: 00011



Inizializzazione: Resto = Dividendo



1001010 : 1000 = 1  
 -1000  
 -----  
 1010



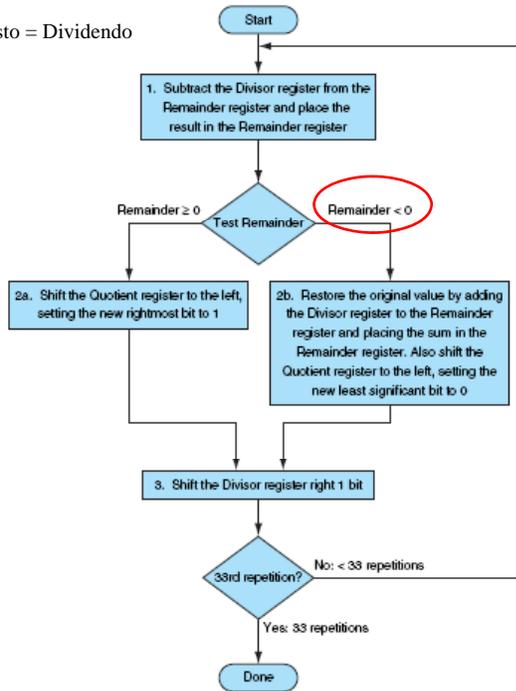
Divisione:: passo 1



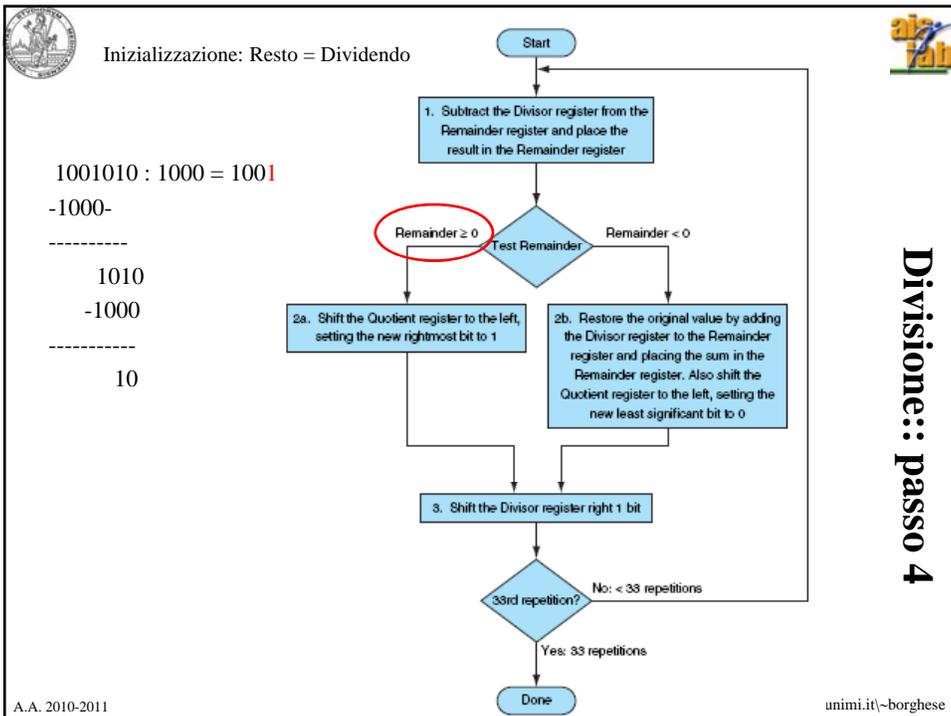
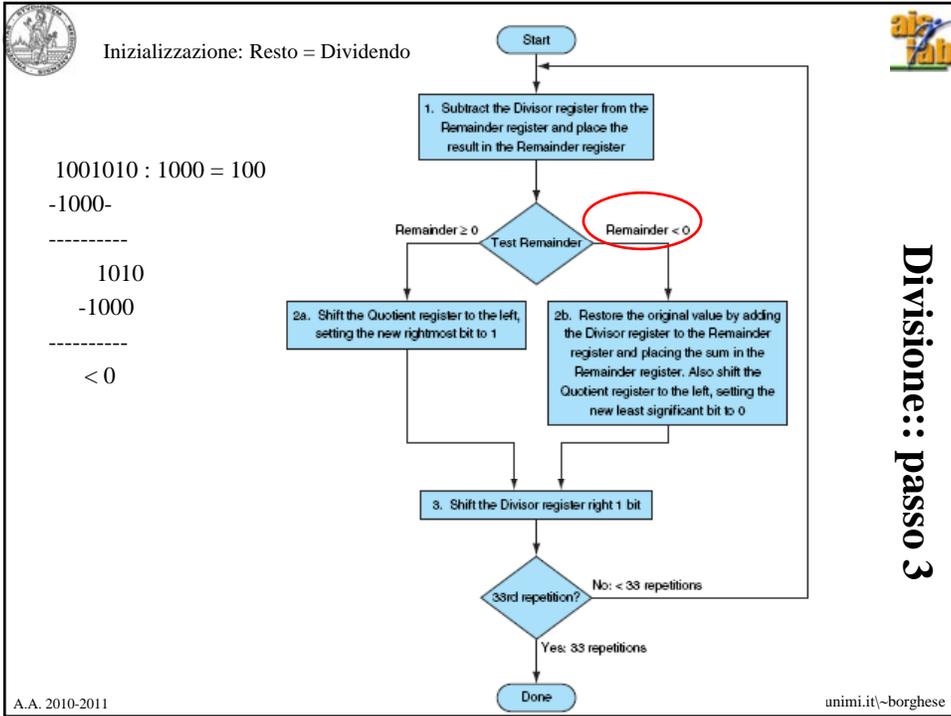
Inizializzazione: Resto = Dividendo



1001010 : 1000 = 10  
 -1000-  
 -----  
 1010  
 -1000  
 -----  
 < 0



Divisione:: passo 2





## Come ottimizzare il circuito della divisione



Il resto si sposta di 1 bit alla volta verso dx ma rimane pari al numero di bit della parola.

Possiamo evitare di spostare il resto.

Il divisore si sposta verso dx di un bit ad ogni passo e divide il resto parziale. Otteniamo lo stesso risultato se **spostiamo il resto parziale a sx di un bit ad ogni passo.**

**Il quoziente si sposta verso sx ad ogni passo.**

Inizializziamo il resto come  $\text{RESTO} = 0 \mid \text{DIVIDENDO}$

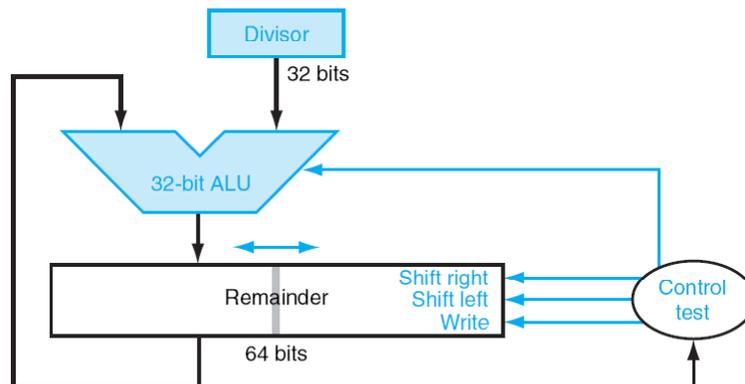
**Ad ogni passo sposto il dividendo di una posizione a sx ed inserisco un bit del quoziente.**

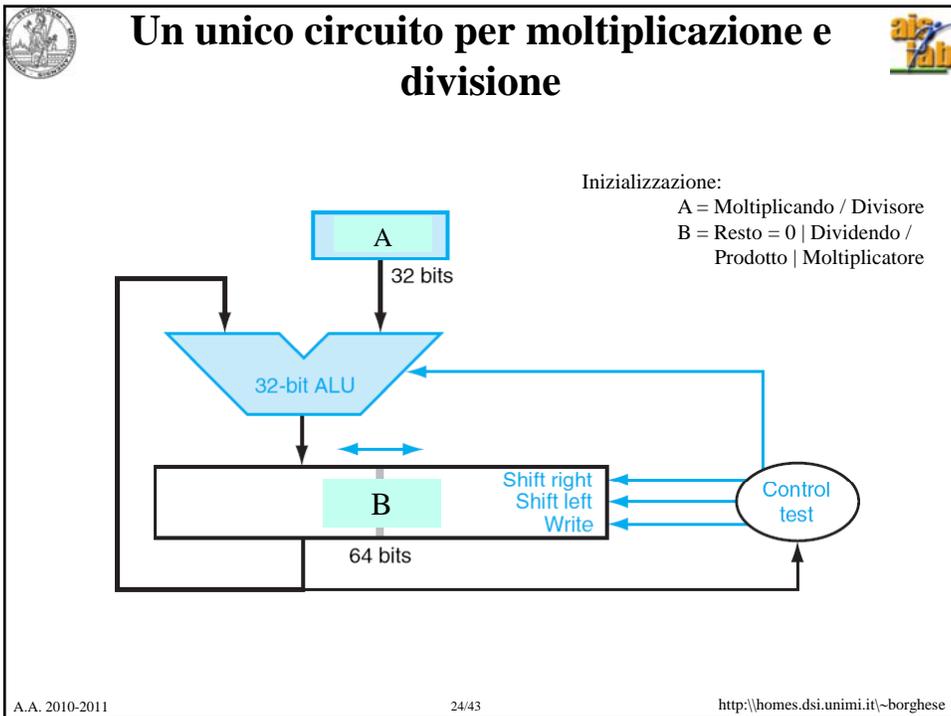
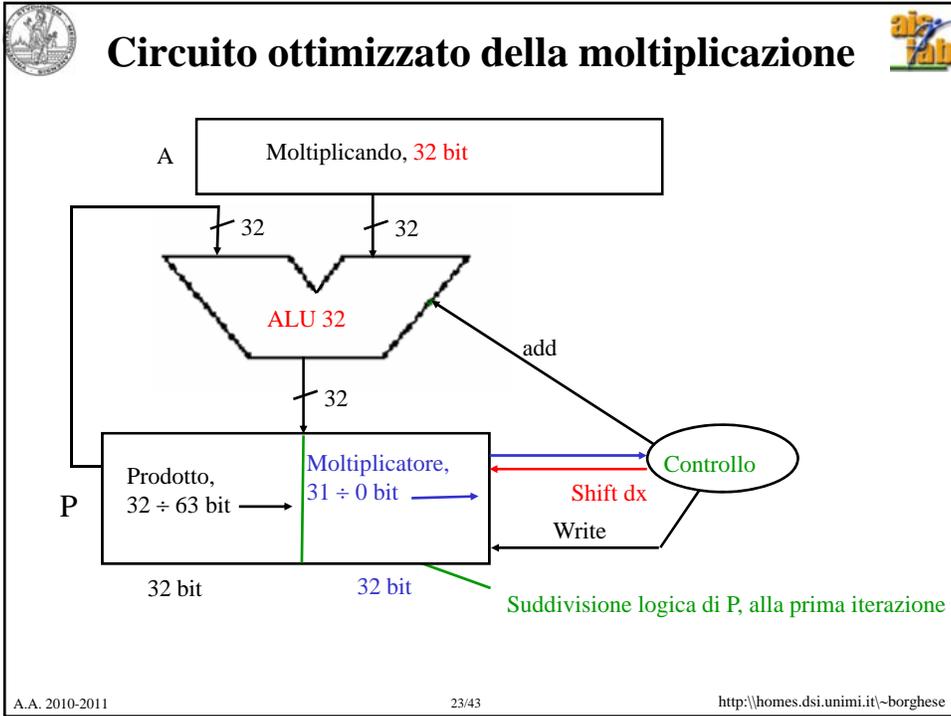


## Il circuito firmware ottimizzato della divisione



Inizializzazione: Resto = 0 | Dividendo







## Il segno della divisione



Dividendo = Quoziente x Divisore + Resto

+	+	+
+	-	-
-	+	-
-	-	+

Qual'è il segno del resto?

Esempio:

$$+7 = +2 \times +3 + 1$$

$$-7 = -2 \times +3 - 1 \quad (\text{sarebbe uguale anche a } -2 \times +4 + 1!!, \text{ ma cambierebbe il quoziente in funzione del segno del resto!})$$

$$+7 = -2 \times -3 + 1$$

$$-7 = +2 \times -3 - 1$$

Osserviamo che il quoziente è positivo se i segni di Divisore e Dividendo sono concordi.  
Il resto ha il segno del Dividendo.



## Sommario



- Divisione intera
- **Somma in virgola mobile**



## Codifica in virgola mobile Standard IEEE 754 (1980)

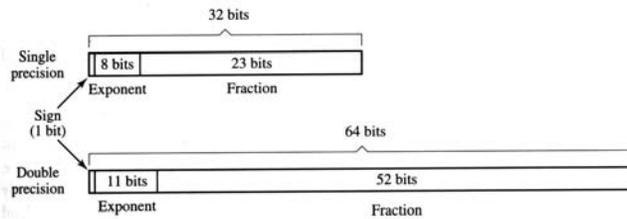


Figure 2-10 Single-precision and double-precision IEEE 754 floating point formats.

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



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

NB I numeri decimali sono normalizzati.

Una possibilità è:

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

100,151

100,51  $\rightarrow$  1,0015  $\times 10^2$  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. Come 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. Quale numero si allinea?
- 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



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

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

Devo trasformare uno dei numeri, una possibilità è:

$$\begin{array}{r} 9,999 \times 10^1 + \\ 0,016 \times 10^1 = \end{array} \quad \text{Perdo un bit perchè non rientra nella capacità della mantissa}$$

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

Normalizzazione per ottenere il risultato finale:

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

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

$$\begin{array}{l} \text{Esempio: } 9,99999 \times 10^2 \Rightarrow \text{Arrotondo alla cifra più vicina} \Rightarrow 10,00 \times 10^3 \\ \qquad \qquad \qquad \Rightarrow \text{Normalizzazione} \qquad \qquad \qquad \Rightarrow 1,0 \times 10^4 \end{array}$$



## 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



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

$$a + b = ?$$

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:

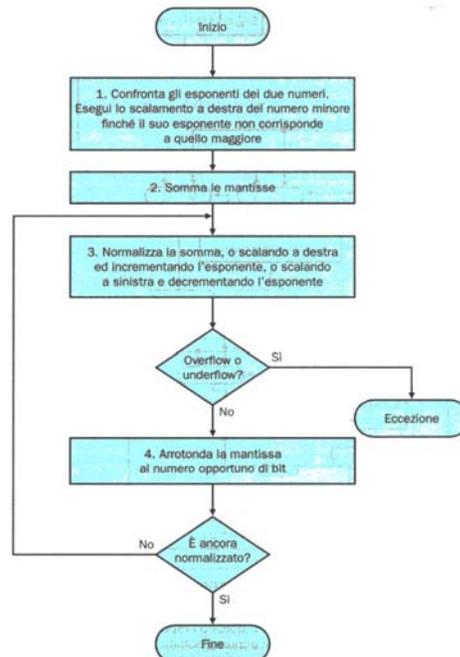
$$2,3400 +$$

$$0,0256 =$$

-----

$$2,3656$$

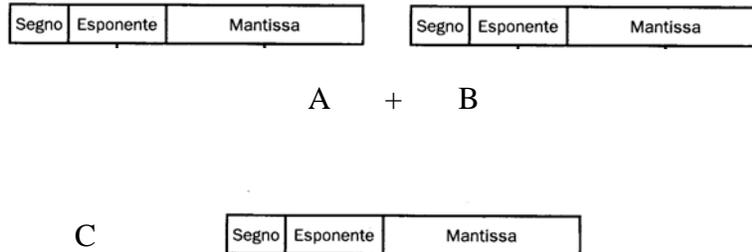
L'arrotondamento finale fornisce per rintrare in 3 cifre decimali fornisce: **2,37**



Algoritmo risultante



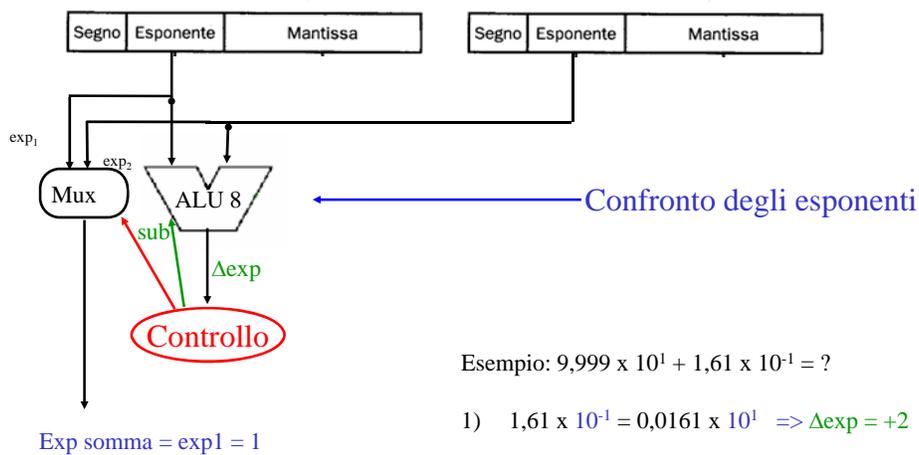
## Il circuito della somma floating point: gli attori

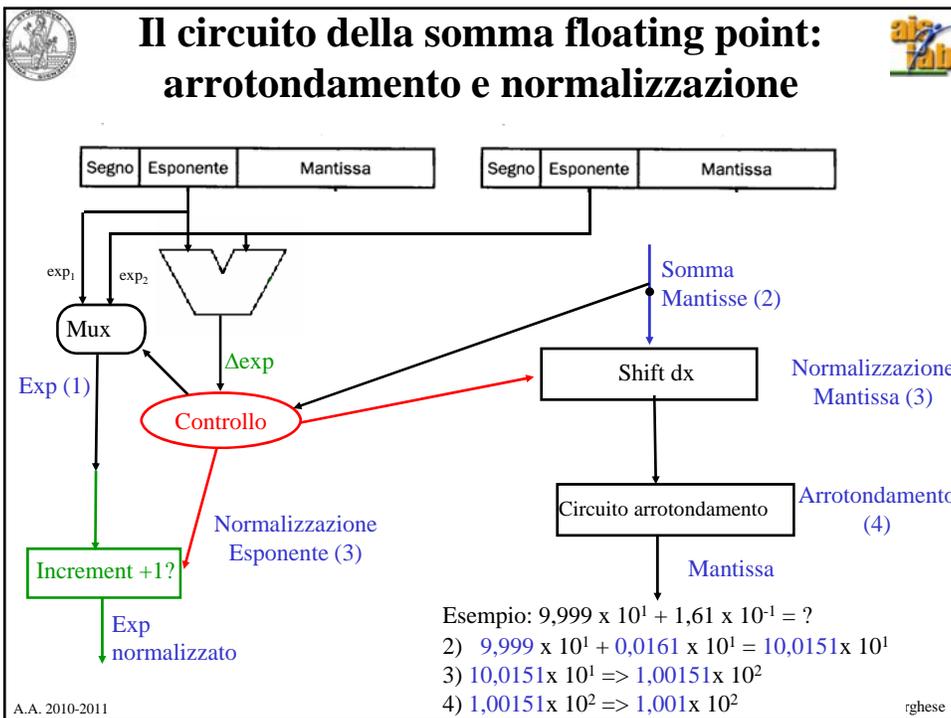
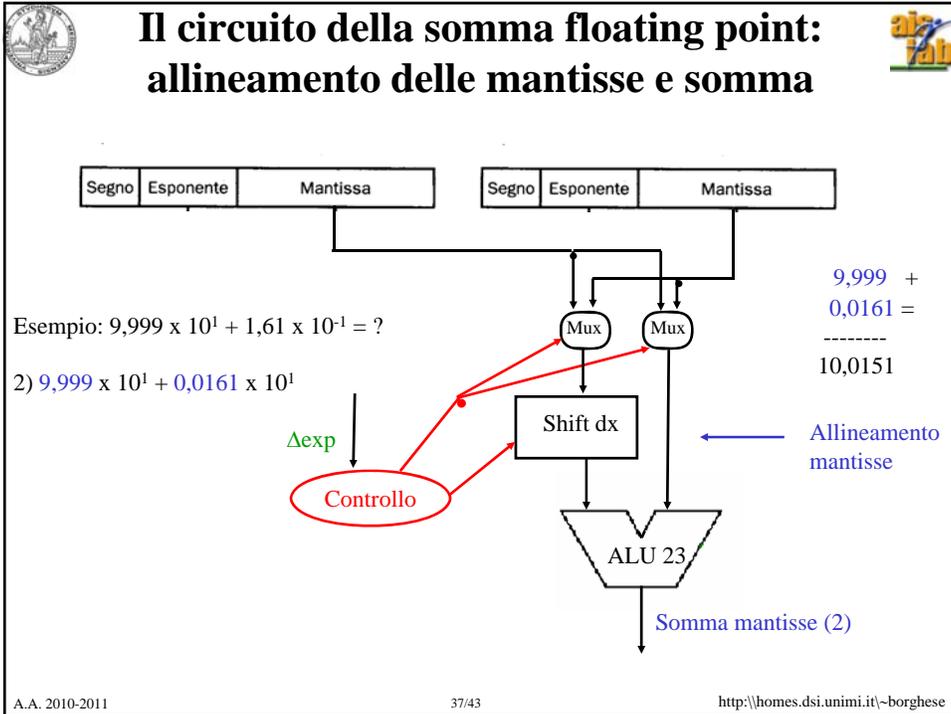


Rappresentazione normalizzata IEEE754



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







## Circuito della somma floating point

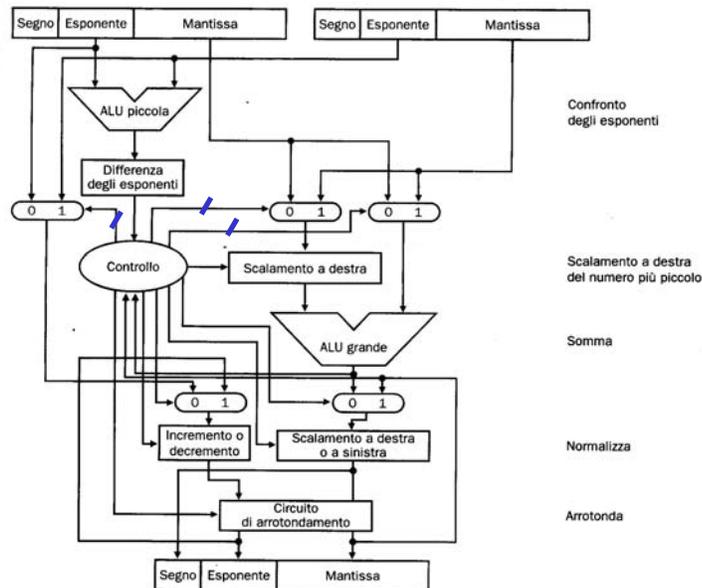


Le tre linee in blu contengono lo stesso segnale di controllo.

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

Gestisce anche i numeri negativi.

Problemi?



A.A. 2010-2011



## 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.

A.A. 2010-2011

40/43

<http://homes.dsi.unimi.it/~borghese>



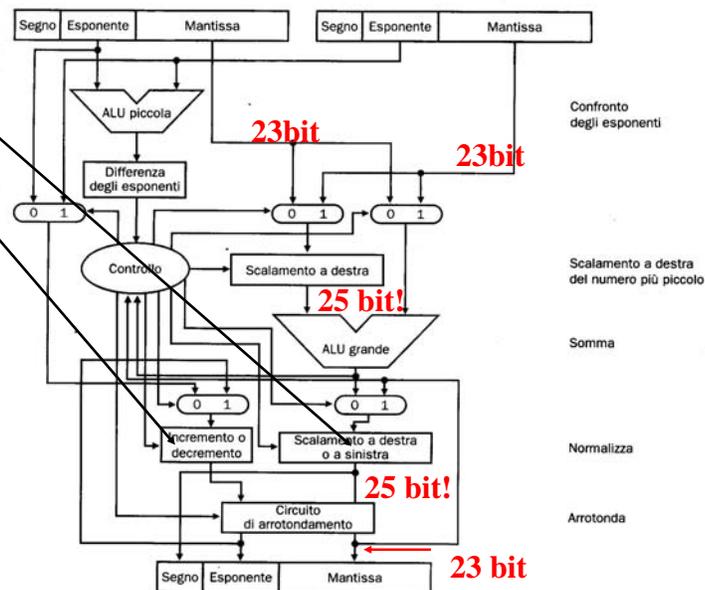
## Circuito della somma floating point con bit di arrotondamento



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**.



A.A. 2010-2011



## 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.

A.A. 2010-2011

42/43

<http://homes.dsi.unimi.it/~borghese>



## Sommario



- Divisione intera
- Somma in virgola mobile