


Firmware, moltiplicatori ed addizionatori floating point

Prof. Alberto Borghese
Dipartimento di Scienze dell'Informazione
borghese@dsi.unimi.it

Università degli Studi di Milano


Sommario

Approcci tecnologici alla ALU.

Tre approcci tecnologici alla costruzione di una ALU (e di una CPU):

- Approccio hardware. Ad ogni operazione corrisponde un circuito combinatorio specifico.
- Approccio ROM. E' un approccio esaustivo (tabellare). Per ogni funzione, per ogni ingresso viene memorizzata l'uscita. E' utilizzabili per funzioni molto particolari (ad esempio di una variabile). Non molto utilizzato.
- Approccio firmware (firm = stabile), o microprogrammato. Si dispone di circuiti specifici solamente per alcune operazioni elementari (tipicamente addizione e sottrazione). Le operazioni più complesse vengono sintetizzate a partire dall'algoritmo che le implementa.

L'approccio firmware

Nell'approccio firmware, viene inserita nella ALU una unità di controllo e dei registri. L'unità di controllo attiva opportunamente le unità aritmetiche ed il trasferimento da/verso i registri. Approccio *"controllore-datapath"*.

Viene inserito un microcalcolatore dentro la ALU.

Il primo microprogramma era presente nell'IBM 360 (1964).

L'approccio firmware vs hardware

La soluzione HW è più veloce ma più costosa per numero di porte e complessità dei circuiti.

La soluzione firmware risolve l'operazione complessa mediante una sequenza di operazioni semplici. E' meno veloce, ma più flessibile e, potenzialmente, adatta ad inserire nuove procedure.

La soluzione HW è percorsa per le operazioni frequenti: la velocizzazione di operazioni complesse che vengono utilizzate raramente non aumenta significativamente le prestazioni (legge di Amdahl).

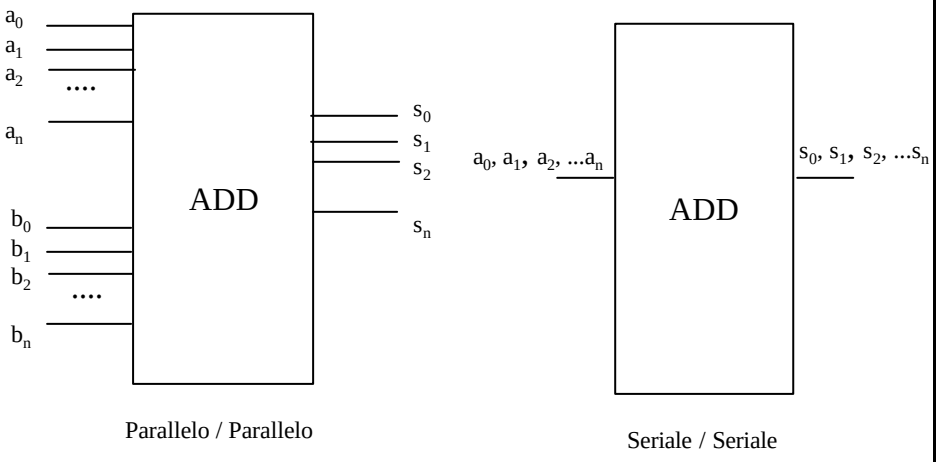
Presentazione seriale / parallela dei dati

Presentazione parallela. Tutti i bit vengono presentati in parallelo (simultaneamente) e la ALU produce in parallelo tutti i bit del risultato. Tipico dell'approccio HW.

Presentazione seriale. Dei 2 operandi viene presentato 1 bit alla volta, del quale viene eseguita l'operazione e fornito il risultato. Viene utilizzata un'unica copia del circuito aritmetico ma il tempo per ottenere l'operazione cresce linearmente con il numero di bit (non si possono utilizzare tecniche di carry-lookahead ad esempio).

Presentazione seriale a byte. In questa modalità i bit vengono presentati a gruppi. Una scelta ragionevole potrebbe essere presentare gruppi di k bit a circuiti capaci di elaborare k bit (cf. addizionatori).

Presentazione seriale / parallela dei dati



Ha senso un input parallelo ed un output seriale o vice-versa?

Sommario

Approccio firmware

Circuiti della moltiplicazione

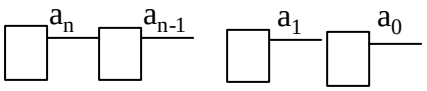
Circuiti della somma in virgola mobile

Shift (scalamento)

Dato A su 32 bit: $a_j = a_{j-k}$ k shift amount (>, =, < 0).

Effettuato al di fuori delle operazioni selezionate dal Mux della ALU, da un circuito denominato *Barrel shifter*.

Tempo comparabile con quello della somma.



Shift dx 1

Il bit a_0 si "perde".
Il bit a_n = 0.

La moltiplicazione ed il firmware

Il razionale degli algoritmi firmware della moltiplicazione è il seguente.

Si analizzano sequenzialmente i bit del moltiplicatore e:

1) Si mette 0 nella posizione opportuna (se il bit analizzato del moltiplicatore = 0).

2) Si mette una copia del moltiplicando nella posizione opportuna (se il bit analizzato del moltiplicatore è = 1).

Moltiplicando 1 1 0 1 1 x

Moltiplicatore 1 0 1 =

---

1 1 0 1 1 +

0 0 0 0 0 -

1 1 0 1 1 - -

Prodotto 1 0 0 0 0 1 1 1

Moltiplicazione utilizzando somma e shift

Utilizzo un registro prodotto da 64 bit, inizializzato a 0.

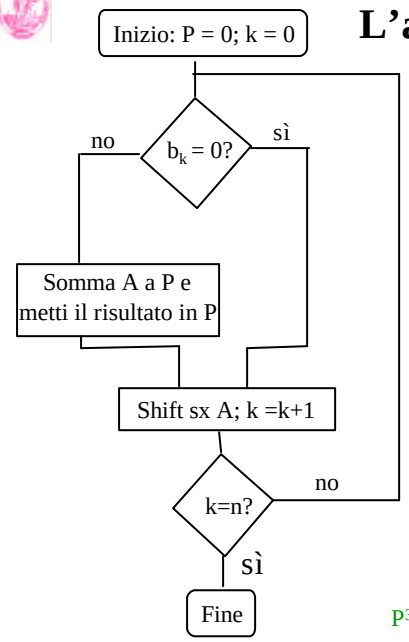
Itero per ogni bit del moltiplicatore:

A) Sommo il moltiplicando al prodotto se il bit = 1.

B) Shift a sx di un bit il moltiplicando (B' = B * base).

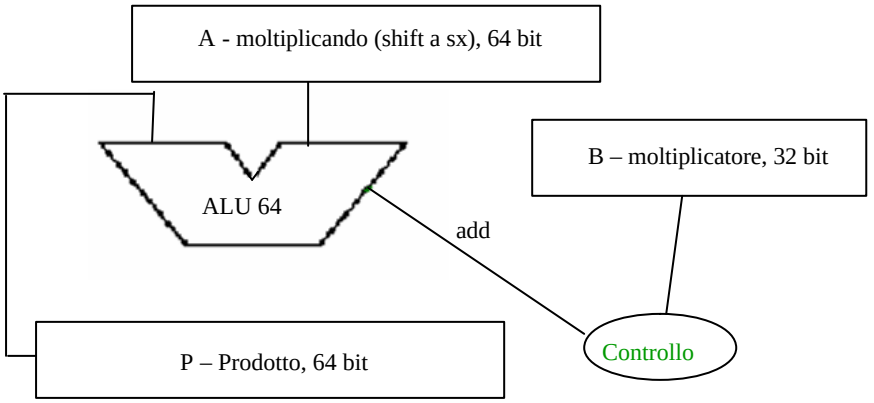
```
A        1 1 0 1 1 x
B          1 1 1 =
--------------------
         1 1 1 1 1
           1 1 0 1 1 +
         1 1 0 1 1 -
--------------------
       1
       1 0 1 0 0 0 1 +
         1 1 0 1 1 - -
--------------------
     1 0 1 1 1 1 0 1
```


L'algoritmo

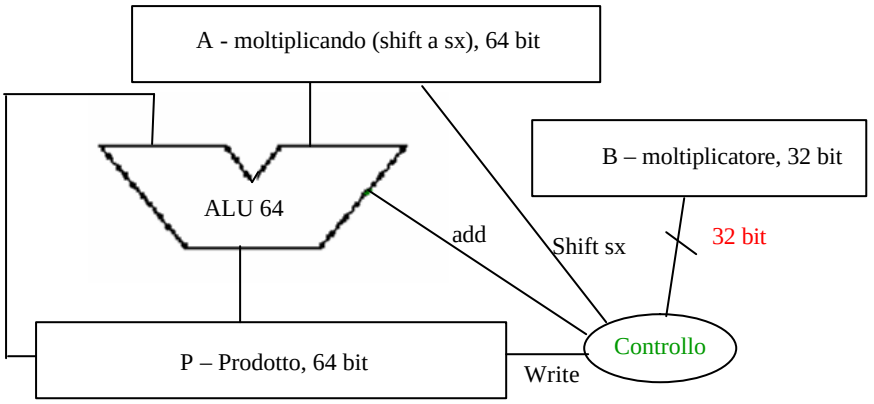



```
A        1 1 0 1 1 x
B          1 1 1 =
--------------------
         1 1 1 1 1
P1 = 0 + A
           1 1 0 1 1 +
A1 = A*2
         1 1 0 1 1 -
--------------------
       1
P2 = P1 + A1
       1 0 1 0 0 0 1 +
A2=A1*2=A*4
         1 1 0 1 1 - - =
--------------------
P3 = P2 + A2
     1 0 1 1 1 1 0 1
```

Implementazione circuitale – prima parte

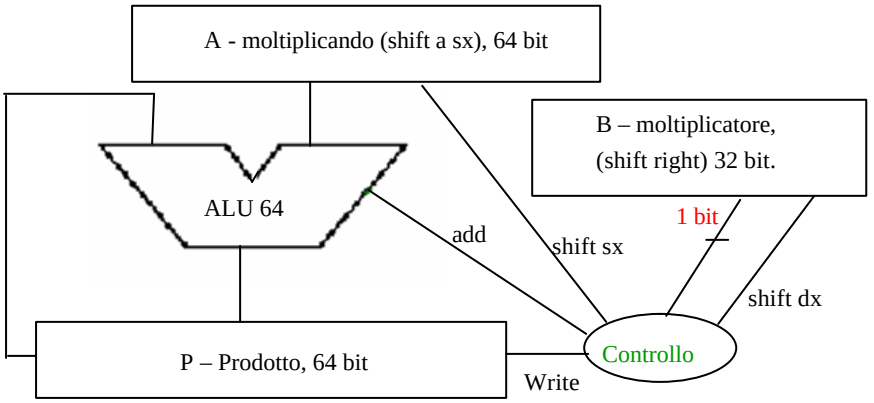


Implementazione circuitale – seconda parte



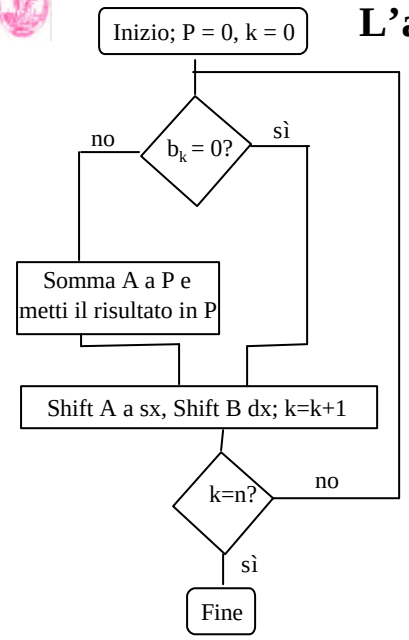
Qual'è il problema?

Implementazione circuitale finale



Qual'è il problema?

L'algoritmo



A ——— 1 1 0 1 1 x

B ——— 1 1 1 =

---

1 1 1 1 1

$A = A * 2^0$ 1 1 0 1 1 +

$A = A * 2^1$ 1 1 0 1 1 -

1

1 0 1 0 0 0 1 +

$A = A * 2^2$ 1 1 0 1 1 - -

P ——— 1 0 1 1 1 1 0 1

Esempio

Iterazione	Passo	Moltiplicatore	Moltiplicando	Prodotto
0	Valori iniziali	0011	0000 0010	0000 0000
1	1a: 1 ⇒ Prod = Prod + Mcando	0011	0000 0010	0000 0010
	2: Scala a sinistra Moltiplicando	0011	0000 0100	0000 0010
	3: Scala a destra Moltiplicatore	0001	0000 0100	0000 0110
2	1a: 1 ⇒ Prod = Prod + Mcando	0001	0000 1000	0000 0110
	2: Scala a sinistra Moltiplicando	0001	0000 1000	0000 0110
	3: Scala a destra Moltiplicatore	0000	0000 1000	0000 0110
3	1: 0 ⇒ Nessuna operazione	0000	0000 1000	0000 0110
	2: Scala a sinistra Moltiplicando	0000	0001 0000	0000 0110
	3: Scala a destra Moltiplicatore	0000	0001 0000	0000 0110
4	1: 0 ⇒ Nessuna operazione	0000	0001 0000	0000 0110
	2: Scala a sinistra Moltiplicando	0000	0010 0000	0000 0110
	3: Scala a destra Moltiplicatore	0000	0010 0000	0000 0110

Razionale per una seconda implementazione

Meta' dei bit del registro moltiplicando vengono utilizzati ad ogni iterazione.

Ad ogni iterazione si aggiunge 1 bit al registro prodotto.

Si sposta la somma dei prodotti parziali verso dx di 1 bit ad ogni iterazione.

```
          1 1 0 1 1 x
            1 1 1 =
---------------------
        1 1 1 1 1
          1 1 0 1 1 +
        1 1 0 1 1 -
---------------------
    1
      1 0 1 0 0 0 1 +
      1 1 0 1 1 - -
---------------------
    1 0 1 1 1 1 0 1
```

Seconda implementazione

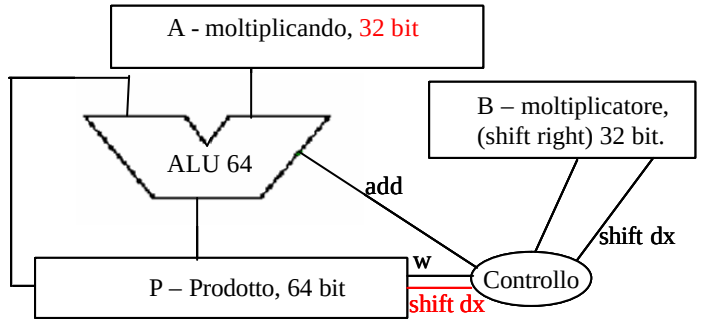
1ª implementazione

1 1 0 1 1
1 1 0 1 1

Sposto a sx il moltiplicando | Sposto a dx il prodotto
P[1] primo prodotto parziale

2ª implementazione

1 1 0 1 1
1 1 0 1 1




```
   1 1 0 1 1 x
     1 1 1 =
----------------------
   1 1 1 1 1
     1 1 0 1 1 +
   1 1 0 1 1 -
------------------------
   1
  1 0 1 0 0 0 1 +
  1 1 0 1 1 - -
-----------------------
  1 0  1 1 1 1 0 1
```

Qual'è il problema?

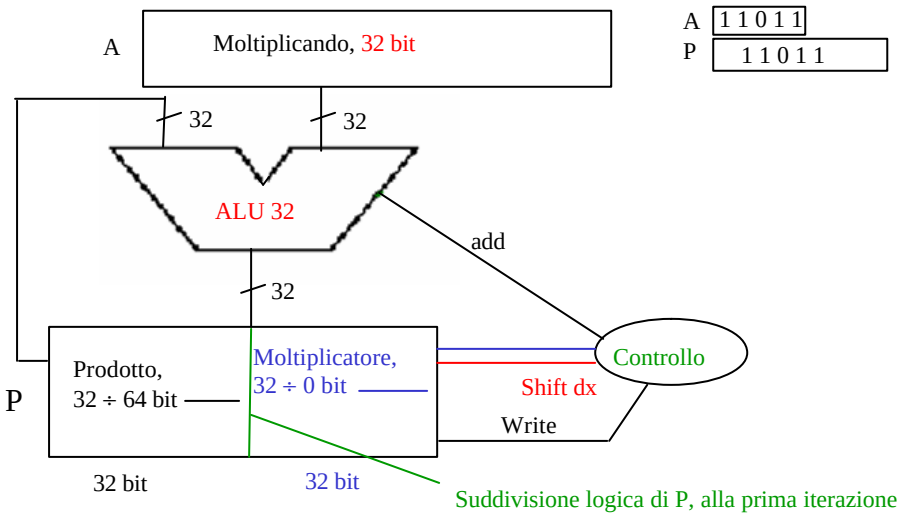
Razionale dell'implementazione ottimizzata

Il numero di bit del registro prodotto corrente (somma dei prodotti parziali) più il numero di bit da esaminare nel registro moltiplicando rimane costante ad ogni iterazione (pari a 64 bit).

Si può perciò eliminare il registro moltiplicando.

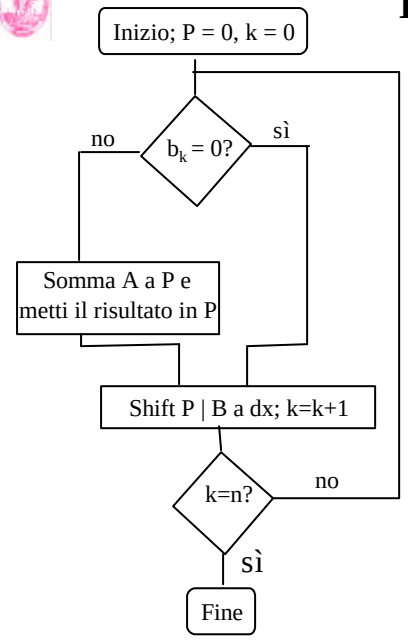
```
   1 1 0 1 1 x
     1 1 1 =
----------------------
   1 1 1 1 1
     1 1 0 1 1 +
   1 1 0 1 1 -
------------------------
   1
  1 0 1 0 0 0 1 +
  1 1 0 1 1 - -
-----------------------
  1 0  1 1 1 1 0 1
```

Circuito ottimizzato



Il moltiplicando è allineato sempre ai 32 bit più significativi del prodotto.
Ad ogni iterazione, il prodotto si allarga, il moltiplicatore si restringe.

L'algoritmo ottimizzato



A —— 1 1 0 1 1 x
B —— 1 1 1 =

---

1 1 1 1 1
1 1 0 1 1 +
1 1 0 1 1 -

1
1 0 1 0 0 0 1 +
1 1 0 1 1 - -

P —— 1 0 1 1 1 1 0 1

Esempio di esecuzione dell'algoritmo ottimizzato

Iterazione	Passo	Moltiplicando	Prodotto
0	Valori iniziali	0010	0000 0011
1	1a: 1 ⇒ Prod = Prod + Mcando	0010	0010 0011
	2: Scala a destra Prodotto	0010	0001 0001
2	1a: 1 ⇒ Prod = Prod + Mcando	0010	0011 0001
	2: Scala a destra Prodotto	0010	0001 1000
3	1: 0 ⇒ Nessuna operazione	0010	0001 1000
	2: Scala a destra Prodotto	0010	0000 1100
4	1: 0 ⇒ Nessuna operazione	0010	0000 1100
	2: Scala a destra Prodotto	0010	0000 0110

Il moltiplicando è allineato (e sommato) ai bit più significativi del prodotto.

Sommario

Approccio firmware

Circuiti della moltiplicazione

Circuiti della somma in virgola mobile

La somma in virgola mobile

Esempio:

- Supponiamo di avere 4 cifre per la mantissa e 2 per l'esponente.
- Supponiamo esponente maggiore > 1.
- Operazione esemplificativa: $9{,}999 \times 10^{1} + 1{,}61 \times 10^{-1} = 1{,}00151 \times 10^{2}$

Algoritmo per la somma in virgola mobile:

1) Trasformare uno dei due numeri in modo che le due rappresentazioni abbiano la stessa base: allineamento della virgola. Quale 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

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

1) Trasformare uno dei due numeri in modo che le due rappresentazioni abbiano la stessa base: allineamento della virgola.
   $1{,}61 \times 10^{-1} = 0{,}0161 \times 10^{1} = 0{,}016 \times 10^{1}$

2) Effettuare la somma delle mantisse:

$$\begin{array}{r} 9{,}999 + \\ 0{,}016 = \\ \hline 10{,}015 \end{array}$$

   Il risultato non è normalizzato.

3) Normalizzare il risultato.

$10{,}015 \times 10^{1} = 1{,}0015 \times 10^{2}$

NB occorre controllare l'overflow o l'underflow dell'esponente, ogni volta che l'esponente viene aggiornato!

Rinormalizzazione per approssimazione

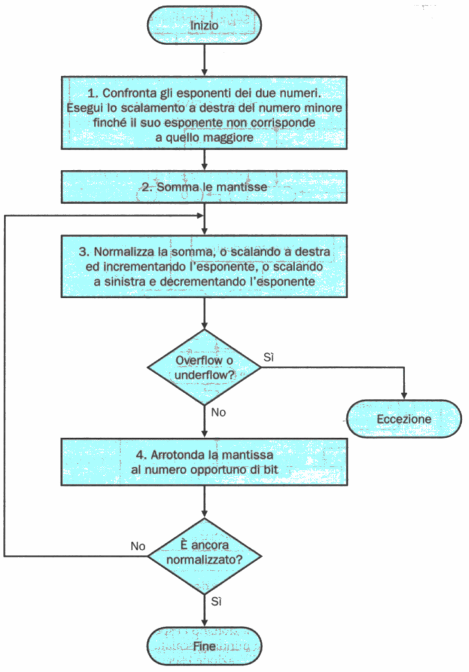
NB: Supponiamo di avere 4 cifre per la mantissa e 2 per l'esponente.

4) Troncamento o arrotondamento di $1{,}0015 \times 10^2$
Troncamento (floor): $1{,}001 \times 10^2$
Arrotondamento alla cifra superiore (ceil): $1{,}002 \times 10^2$
Arrotondamento alla cifra più vicina: $1{,}002 \times 10^2$

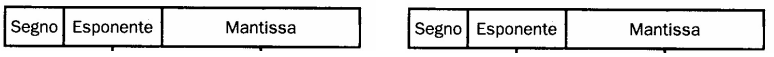
NB: In questa fase si può generare la necessità di rinormalizzare il numero (passo 3):

Esempio: $9{,}99999 \times 10^2$ => Arrotondo alla cifra più vicina => $10{,}00 \times 10^3$
=> Normalizzazione => $1{,}0 \times 10^4$

Algoritmo risultante

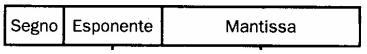


Il circuito della somma floating point: gli attori



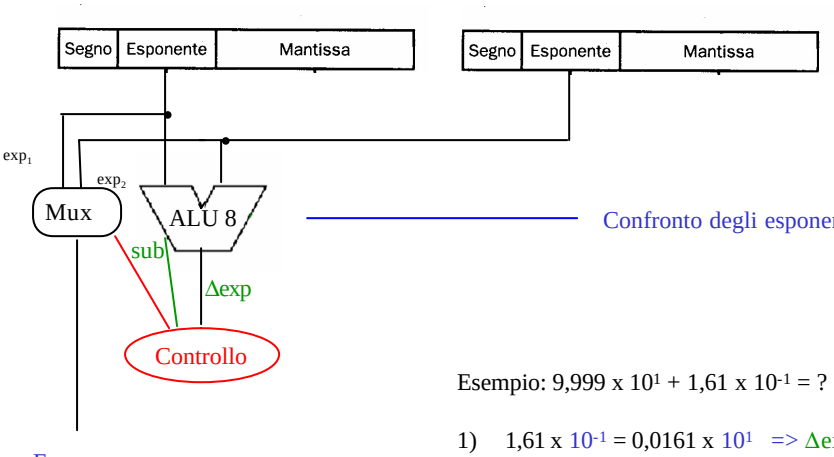
A + B

C



Rappresentazione normalizzata IEEE754

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

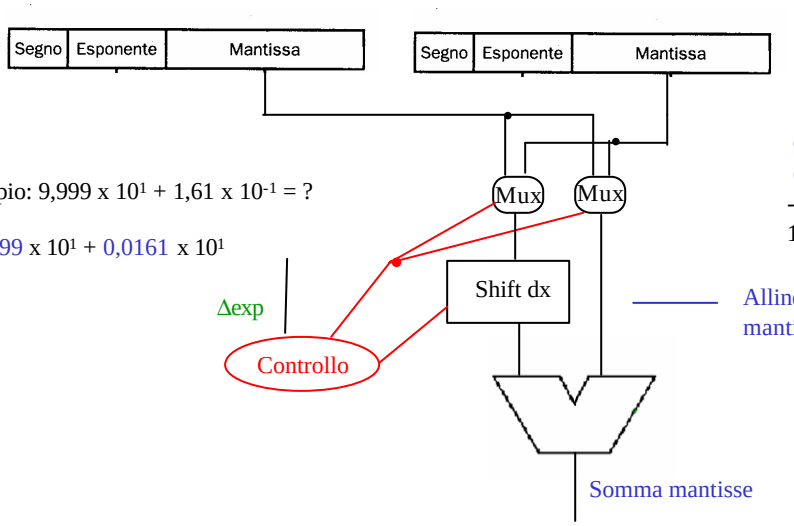


Confronto degli esponenti

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

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

Il circuito della somma floating point: allineamento delle mantisse e somma



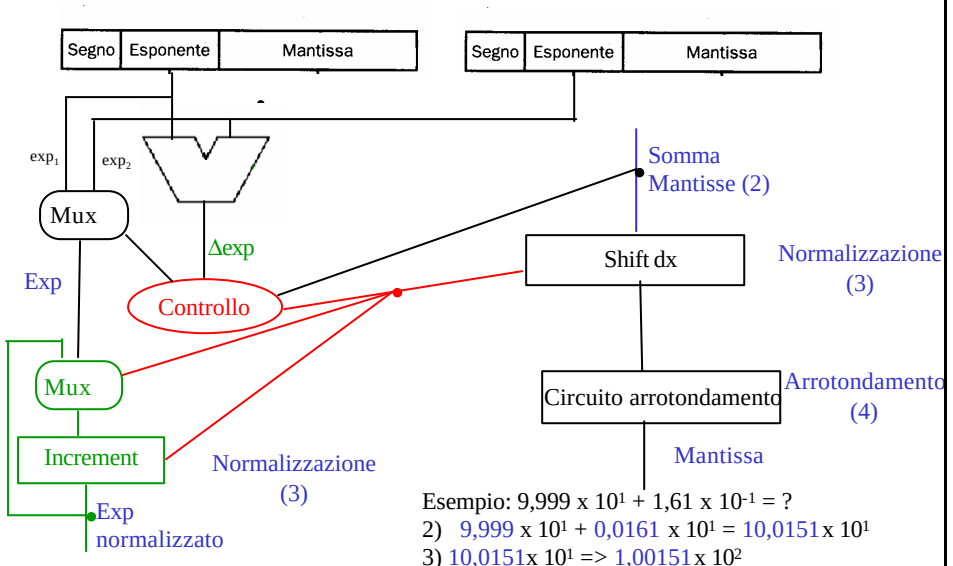
Esempio: 9,999 x 10^1 + 1,61 x 10^{-1} = ?

2) 9,999 x 10^1 + 0,0161 x 10^1

9,999 +
0,0161 =

10,051

Il circuito della somma floating point: arrotondamento e normalizzazione



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

2) 9,999 x 10^1 + 0,0161 x 10^1 = 10,0151 x 10^1

3) 10,0151 x 10^1 => 1,00151 x 10^2

4) 1,00151 x 10^2 => 1,001 x 10^2

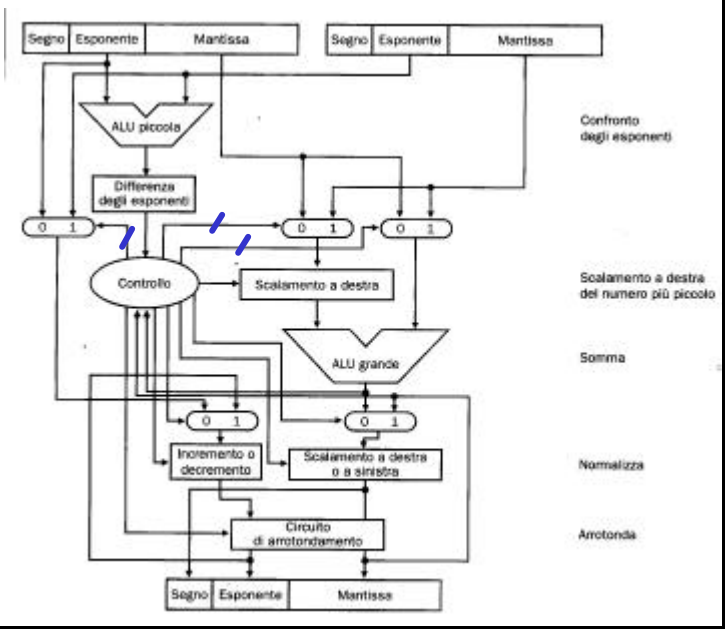
Circuito della somma floating point

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?



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.

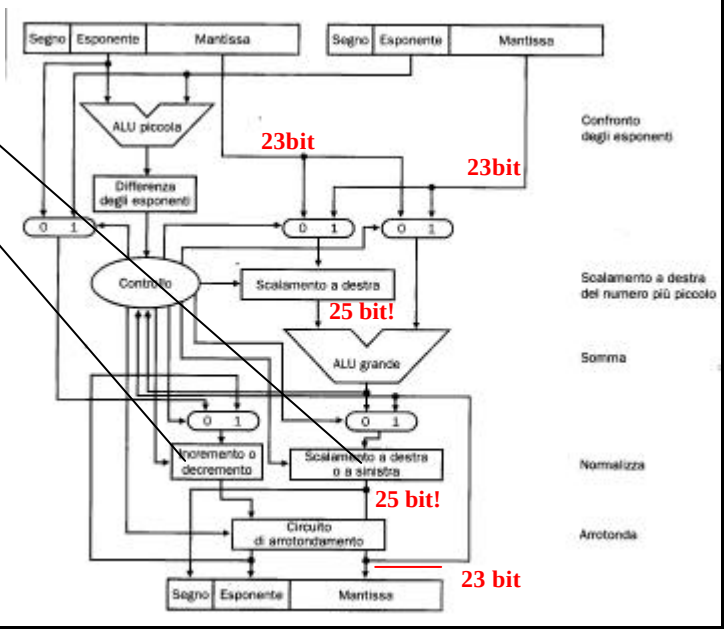
Circuito della somma floating point

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.

Sono utilizzati per gestire al meglio l'arrotondamento.



Sommario

Approccio firmware

Circuiti della moltiplicazione

Circuiti della somma in virgola mobile