


Carry look-ahead Adder
&
Firmware Multiplier

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

Università degli Studi di Milano

Riferimenti sul Patterson: B.6 & 3.4


Sommario

Problemi dei sommatori

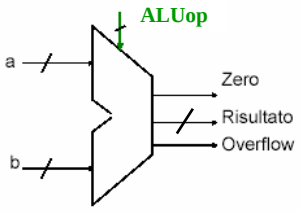
Sommatori ad anticipazione di riporto

I problemi del moltiplicatore firmware

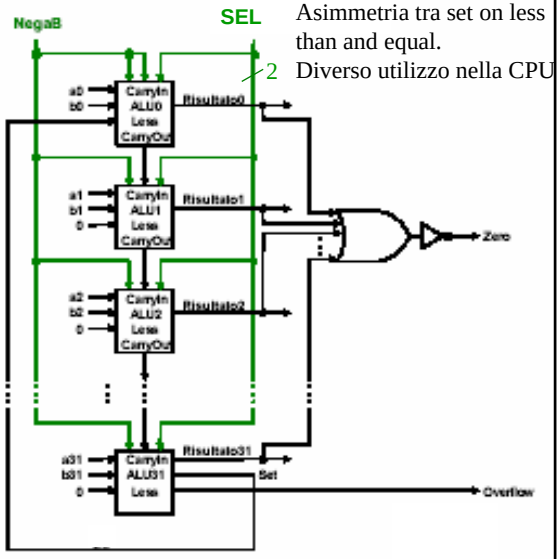
Ottimizzazione dei moltiplicatori firmware

ALU a 32 bit: struttura finale

Operazioni possibili:
- AND
- OR
- Somma / Sottrazione
- Comparazione
- Test di uguaglianza



Sono evidenziate solamente le variabili visibili all'esterno.



Asimmetria tra set on less than and equal.
Diverso utilizzo nella CPU

Operazione di somma

```
111          ← Riporto
 1011 +      ← Addendo 1
  110 =      ← Addendo 2
-------
10001
```

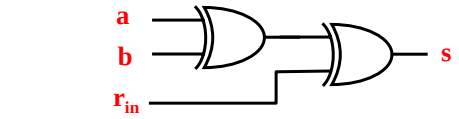
3 Attori: addendo 1, addendo 2, riporto.

Viene eseguita sequenzialmente da dx a sx.

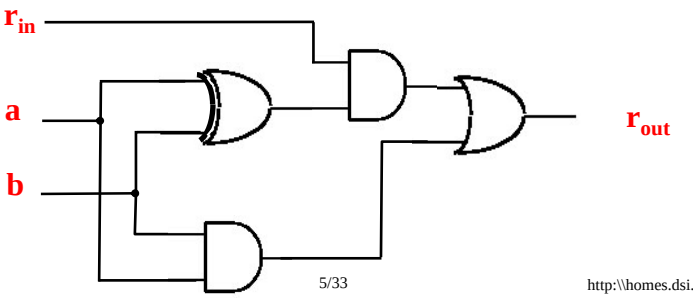
Full adder

$$s = (a \oplus b)\overline{r_{in}} + \overline{(a \oplus b)} r_{in} = (a \oplus b) \oplus r_{in}$$

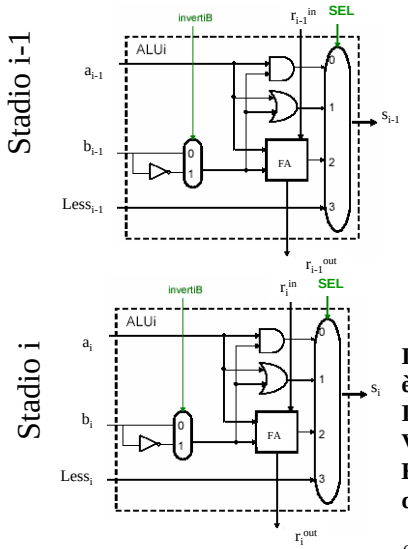
$$r_{out} = ab + (a \oplus b) r_{in} = ab + (a + b) r_{in}$$

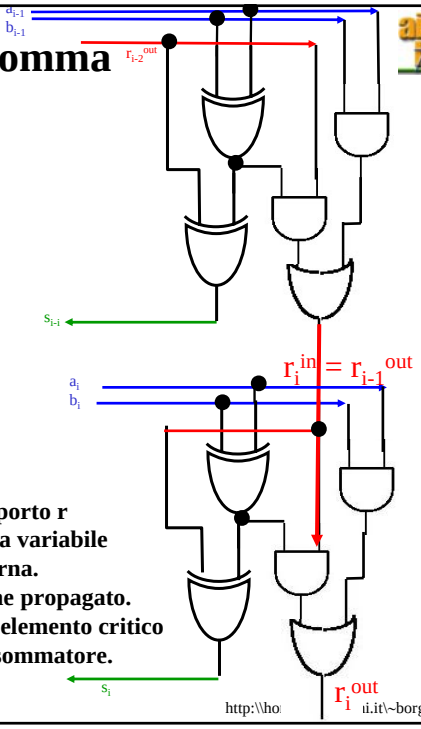


5 porte logiche.
Cammini critici: s – 2; r_{out} - 3



Circuito della somma





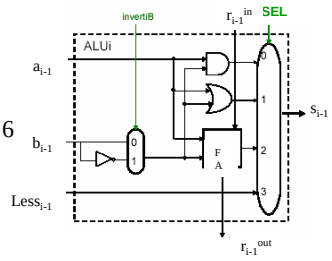
Il riporto r è una variabile Interna. Viene propagato. E' l'elemento critico del sommatore.

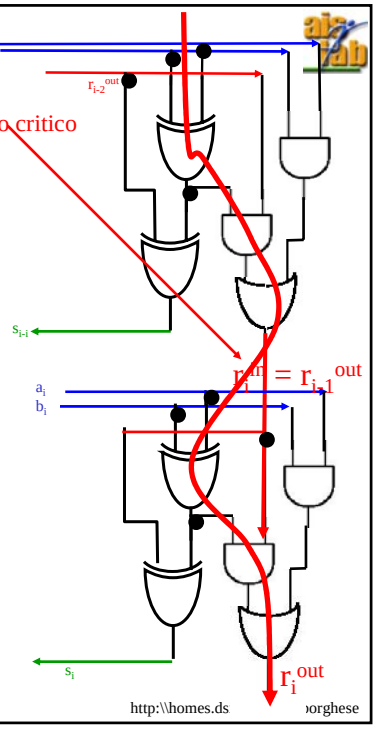
Cammini critici

Per ogni stadio:
Somma: 2
Riporto: 3

Per due stadi:
Somma: 2
Riporto: 3 + 3 = 6

Riporto = 3 * N





111
1011 +
110 =

10001

Funzionamento sequenziale

I problemi del full-adder

Il full adder con propagazione del riporto è lento:

- Il riporto si propaga sequenzialmente
  - caratteristica dell'algoritmo di calcolo
- la commutazione dei circuiti non è istantanea (tempo di commutazione)
  - caratteristica fisica dei dispositvi
- Soluzioni
  - modificare l'algortimo
  - modificare i dispositivi

Sommario

Problemi dei sommatori

Sommatori ad anticipazione di riporto

I problemi del moltiplicatore firmware

Ottimizzazione dei moltiplicatori firmware

Prima possibilità: forma tabellare

Riscrivo le equazioni del riporto in modo non sequenziale. Come?

$r_0 = a_0b_0 + (a_0 + b_0)r_0 = a_0b_0 + a_0r_0 + b_0r_0$

$r_1 = a_1b_1 + (a_1 + b_1)r_0 =$
$a_1b_1 + a_1(a_0b_0 + a_0r_0 + b_0r_0) +$
$+ b_1(a_0b_0 + a_0r_0 + b_0r_0) =$ …

$r_2 = a_2b_2 + (a_2r_1 + b_2)r_1 =$
$a_2b_2 + a_2(a_1b_1 + a_1r_1 + ...$

Molto complesso per n grande.

$s = (a \oplus b) \oplus r_{in}$

$r_{out} = ab + (a + b)\, r_{in}$

Riporto indipendente da r_{in} Riporto funzione di r_{in}

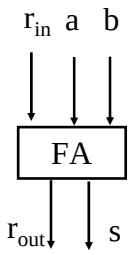
$r_{out} = f(a_0, b_0, a_1, b_1, a_2, b_2, a_3, b_3, ...)$

Funzione a 2 * N ingressi ed 1 uscita.

Carry look-ahead (anticipazione di riporto)

Approccio strutturato per diminuire la latenza della somma.

$r_{out} = ab + (a \oplus b)\, r_{in}$



Analisi del singolo stadio.
Quando si genera un riporto in uscita?

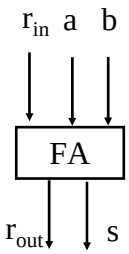
Quando ho almeno due 1, in ingresso; cioè tra r_{in}, a e b.

11000 riporto
1101 +
100 =

10001

Propagazione e generazione

Ho riporto quando ho almeno due 1, in ingresso; cioè tra r_{in}, a e b.



Osservazioni:

1) Viene generato un riporto dallo stadio i, qualsiasi sia il riporto in ingresso se a = b = 1 => $g_i = a_i b_i$.

2) Viene generato un riporto allo stadio i, se il riporto in ingresso è = 1 ed una delle due variabili in ingresso è = 1 => $p_i = (a_i \oplus b_i) r_i^{in}$.(pi propaga il segnale di riporto r_i^{in}).

Quando sia la condizione 1) che la condizione 2) è verificata?
Cosa succede se entrambe le condizioni sono verificate?

Esempio

Sono interessato ad r_4^{out}. Supponiamo $r_0^{in} = 0$.

r_{in}	0 0 0 0 0 0 0	0 1 1 1 0 0 0	0 1 1 1 0 0 0
a	1 0 1 0 1 1 0 1 +	1 0 1 0 1 1 0 1 +	1 0 1 1 1 1 0 1 +
b	1 0 0 0 0 =	1 1 0 1 0 =	1 1 0 0 0 =
	---	---	---
	1 0 1 1 1 1 1 1	1 1 1 0 0 1 1 1	1 1 0 1 0 1 0 1
	$r_5^{in} = r_4^{out} = 0$	$r_5^{in} = r_4^{out} = 1$	$r_5^{in} = r_4^{out} = 1$
		Per propagazione	Per generazione
		$p_4 = (a_4 \oplus b_4) r_4^{in}$.	$g_4 = a_4 b_4$

Sviluppo della funzione logica riporto

$$r_i^{out} = ab + (a \oplus b)\, r_i^{in}$$

$$r_i^{out} = g_i + p_i\, r_i^{in}$$

$$r_0 = g_0 + p_0 r_0$$

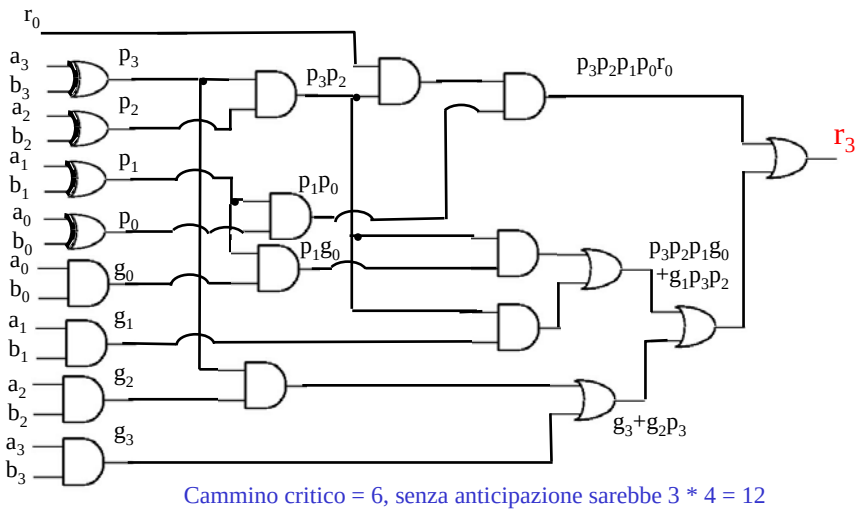
$$r_1 = g_1 + p_1 r_0 = g_1 + p_1 g_0 + p_1 p_0 r_0$$

$$r_2 = g_2 + p_2 r_1 = g_2 + p_2(g_1 + p_1 g_0 + p_1 p_0 r_0) = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 r_0.$$

$$r_3 = g_3 + p_3 r_2 = g_3 + p_3(g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 r_0) = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 r_0.$$

Determinazione del cammino critico.

$r_3 = g_3 + p_3 r_2 = g_3 + p_3(g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 r_0) =$
$g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 r_0.$



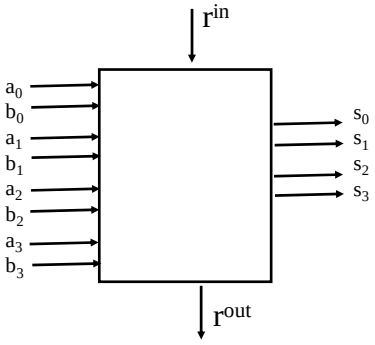
Cammino critico = 6, senza anticipazione sarebbe 3 * 4 = 12

Addizionatori modulari

La complessità del circuito è tollerata per piccoli n.

Circuiti sommatori indipendenti si hanno per 4 bit.

Moduli elementari.



Come si ottiene la somma?

Collegando in cascata i moduli (sommatori elementari).
Cammino critico = 6 * N/4. Per 32 bit, 48.
Per confronto, senza parallelizzazione, per 32 bit, N * 3 = 96.

Sommario

Problemi dei sommatori

Sommatori ad anticipazione di riporto

I problemi del moltiplicatore firmware

Ottimizzazione dei moltiplicatori firmware

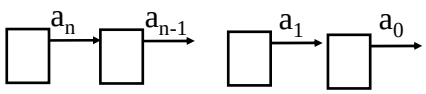
Shift (scalamento)

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

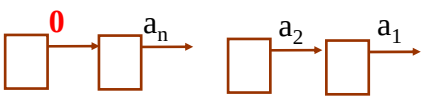
Effettuato al di fuori delle operazioni selezionate dal Mux della ALU, da un circuito denominato *Barrel shifter*.

Tempo comparabile con quello della somma.

Operazioni codificate in modo specifico nell'ISA.



Shift dx 1



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

Algoritmi per la moltiplicazione

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.

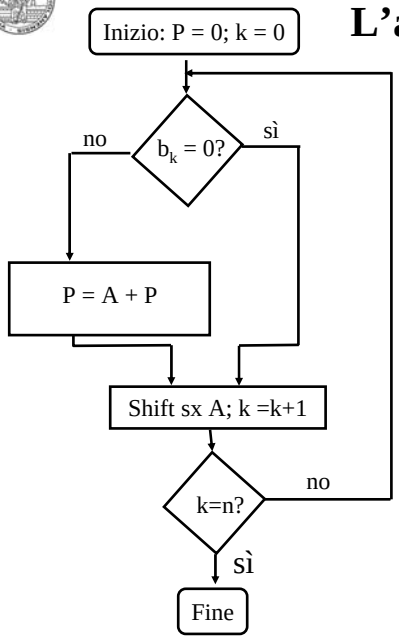
Itero per ogni bit del moltiplicatore:

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

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

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

---

0 0 0 0 0 + P
1 1 0 1 1 A

$P_1 = 0 + A$ → 1 1 1 1 1

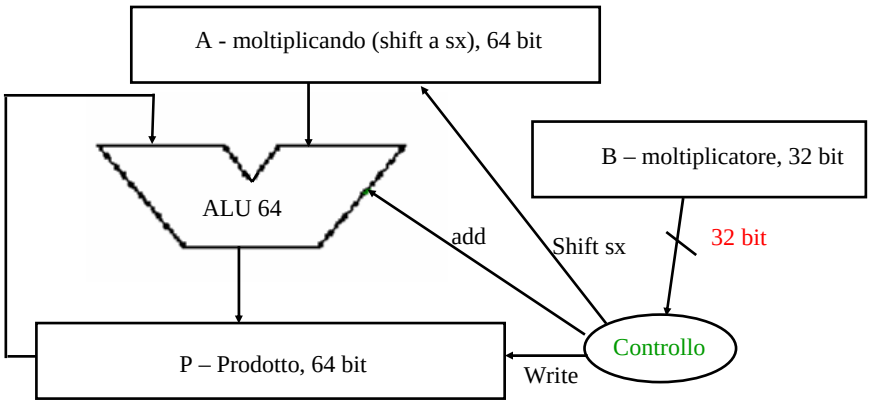
$A_1 = A*2$ → 1 1 0 1 1 + P
1 1 0 1 1 - A_1

1

$P_2 = P_1 + A_1$ → 1 0 1 0 0 0 1 + P
$A_2=A_1*2=A*4$ → 1 1 0 1 1 - - A_2

$P_3 = P_2 + A_2$ → 1 0 1 1 1 1 0 1

Implementazione circuitale



Qual'è il problema?

Sommario

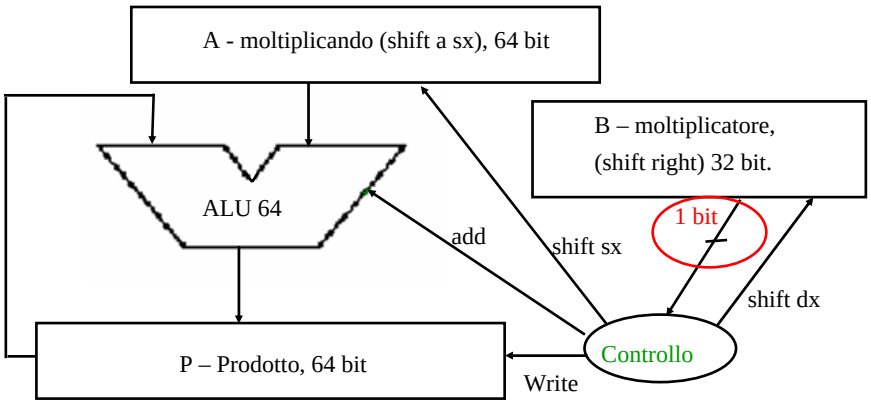
Problemi dei sommatori

Sommatori ad anticipazione di riporto

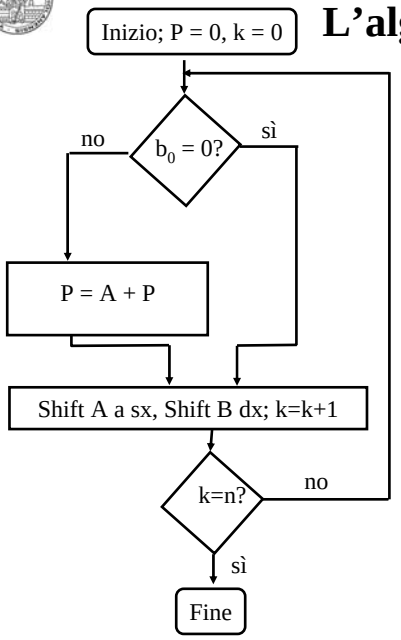
I problemi del moltiplicatore firmware

Ottimizzazione dei moltiplicatori firmware

Implementazione circuitale ottimizzata - I



L'algoritmo - I



A → 1 1 0 1 1 x
B → 1 1 1 =

---

0 0 0 0 0 + P

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

---

1 1 1 1 1

1 1 0 1 1 + P

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

1

1 0 1 0 0 0 1 + P

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

P → 1 0 1 1 1 1 0 1

Esempio - I

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.

Ad ogni iterazione sommo N cifre (pari al numero di cifre del moltiplicando).

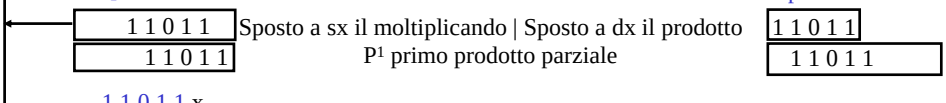
Spostamento della ALU sul registro prodotto.
Oppure
Si sposta la somma dei prodotti parziali verso dx di 1 bit ad ogni iterazione.

```
      1 1 0 1 1 x
        1 1 1 =
-----------------------
      0 0 0 0 0 +    P
      1 1 0 1 1      A
-----------------------
    1 1 1 1 1
      1 1 0 1 1 +    P
    1 1 0 1 1 -      A1
-----------------------
  1
    1 0 1 0 0 0 1 +  P
    1 1 0 1 1 - -    A2
-----------------------
  1 0 1 1 1 1 0 1
```

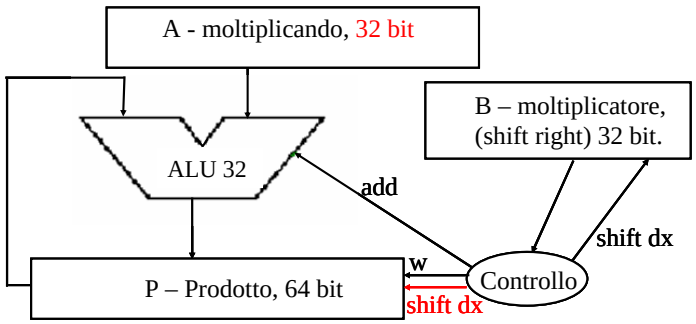

Implementazione ottimizzata - II

1ª implementazione

2ª implementazione




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

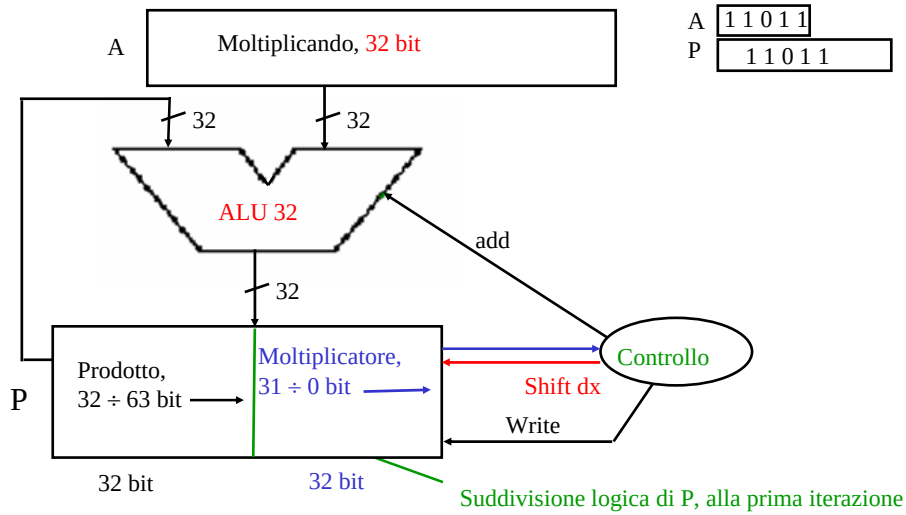
Razionale dell'implementazione - III

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.

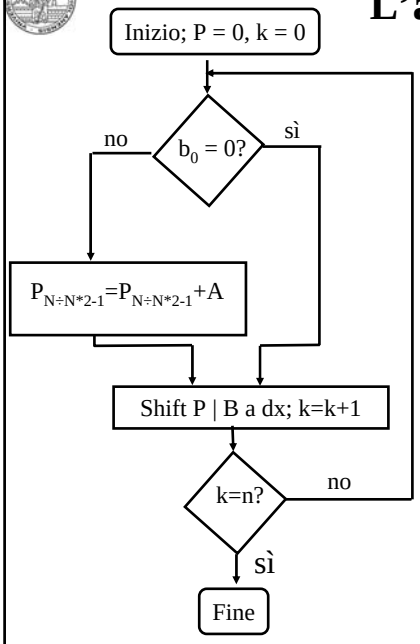
```
        1 1 0 1 1 x
          1 1 1 =
----------------------
        0 0 0 0 0 +
        1 1 0 1 1
----------------------
      1 1 1 1 1
        1 1 0 1 1 +
      1 1 0 1 1 -
-----------------------
    1
      1 0 1 0 0 0 1 +
      1 1 0 1 1 - -
 ----------------------
-
    1 0  1 1 1 1 0 1
```


Circuito ottimizzato - III



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 - III




```
A ──►      1 1 0 1 1 x
B ──►        1 1 1 =
     ----------------------
           0 0 0 0 0 +
           1 1 0 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 - III

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

Problemi dei sommatori

Sommatori ad anticipazione di riporto

I problemi del moltiplicatore firmware

Ottimizzazione dei moltiplicatori firmware