

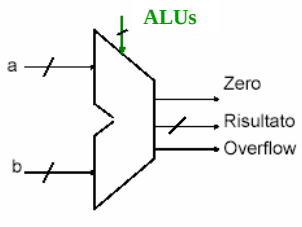

Carry look-ahead Adder
&
Firmware Multiplier

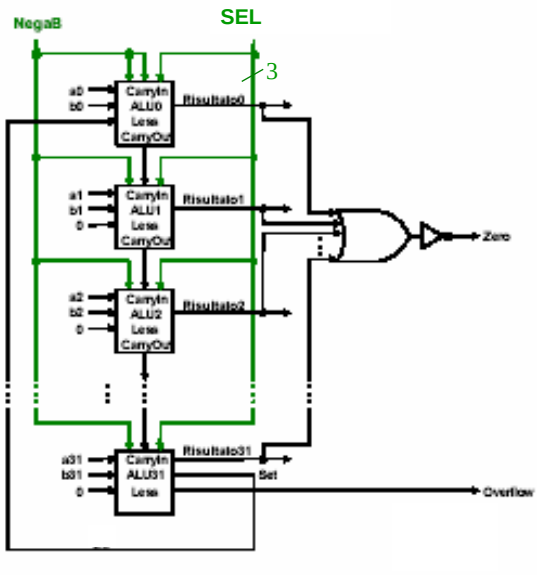
Prof. Alberto Borghese
Dipartimento di Scienze dell'Informazione
borghese@dsi.unimi.it

Università degli Studi di Milano


Sommario

ALU a 32 bit: struttura finale





Qual'è il problema?

Operazione di somma

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

3 Attori: addendo 1, addendo 2, riporto.

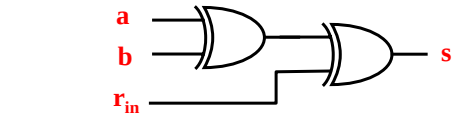
Viene eseguita sequenzialmente da dx a sx.

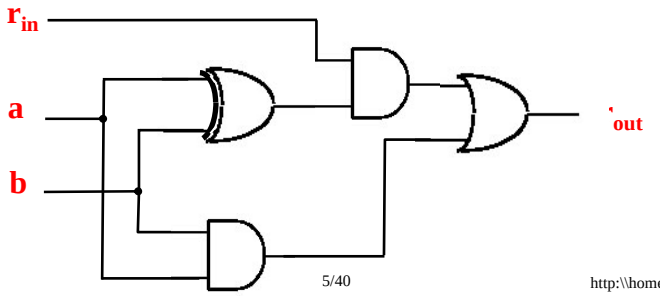
Semplificazione circuitale

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

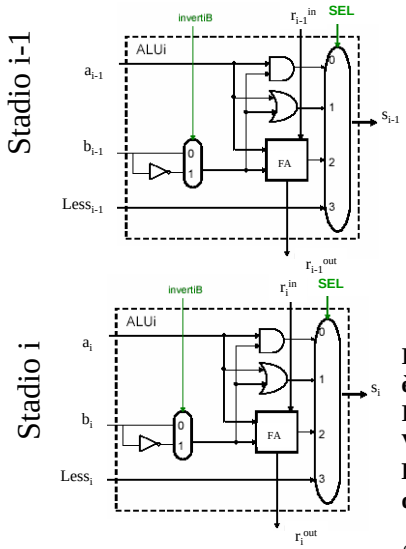
$$r_{out} = ab + (a \oplus b) r_{in}$$

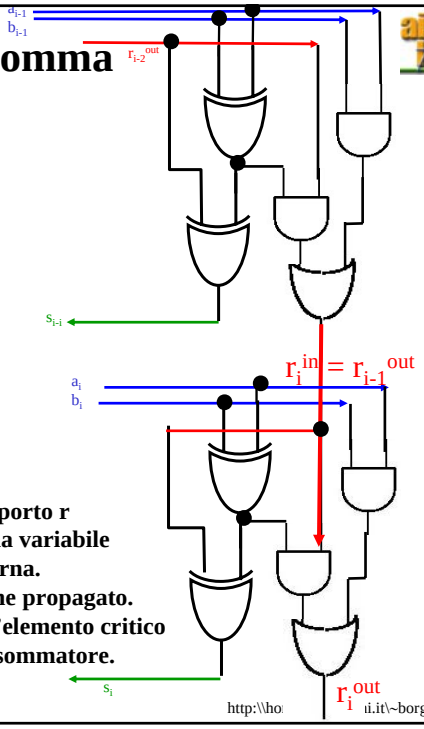
5 porte logiche.
Cammini critici: s – 2; r_{out} - 3





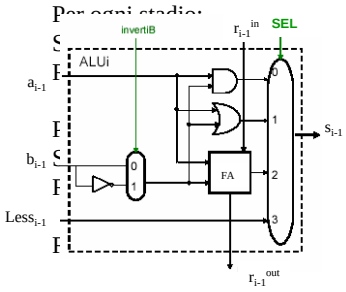
Circuito della somma





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

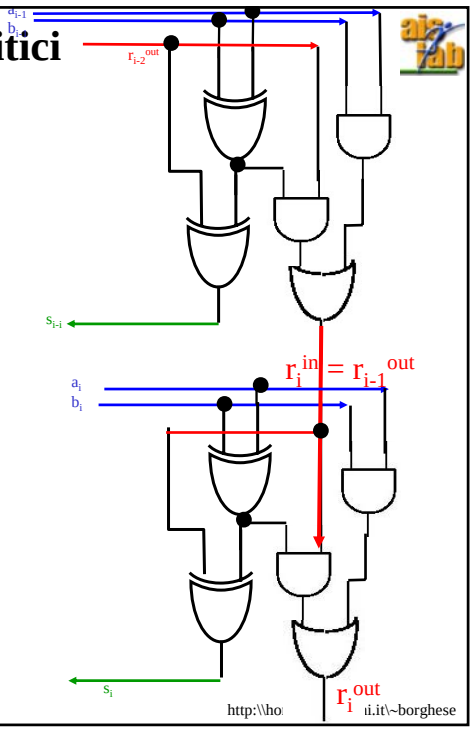
Cammini critici



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

Il firmware

Moltiplicatori 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.

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

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

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

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

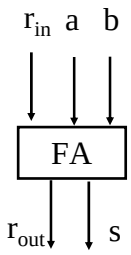
Funzione a 2 * N ingressi ed 1 uscita.

In questo caso posso sfruttare l'uguaglianza: $r_{out} = ab + (a \oplus b)\, r_{in} = ab + (a + b)\, r_{in}$

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?

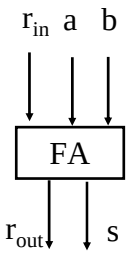
Quando ho almeno due 1, in ingresso; cioè tra r_{in}, a e b.

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

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

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

$$r_0 = g_0 + p_0 r_0$$

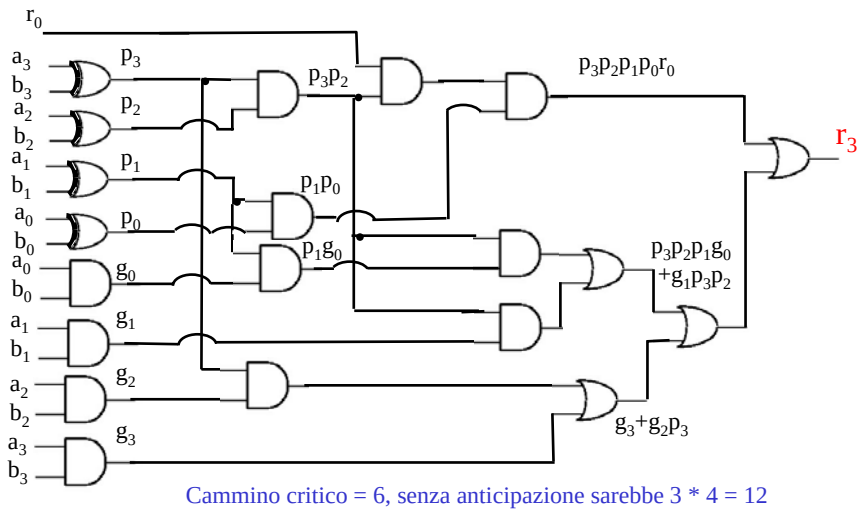
$$r_1 = g_1 + p_1 r_0 = g_1 + p_1 g_0 + p_1 p_0 r_0$$

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

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

Determinazione del cammino critico.

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



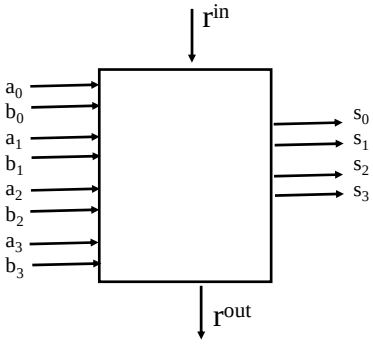
Cammino critico = 6, senza anticipazione sarebbe 3 * 4 = 12

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

Il firmware

Moltiplicatori firmware

Ottimizzazione dei moltiplicatori firmware

Approcci tecnologici alla ALU.

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

- Approccio hardware programmabile (e.g. PAL). 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.

Approccio ROM alla ALU

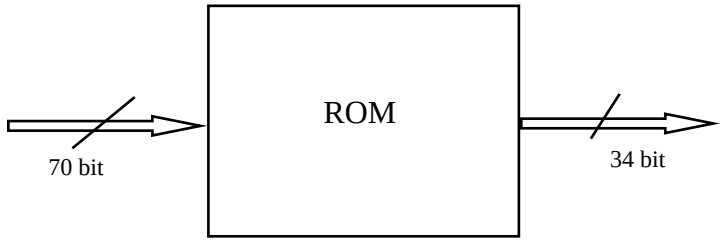
Input:
- A n bit
- B n bit
- SEL k bit
- InvertiB: 1 bit

Totale: 2*n + k + 1 bit

Output:
- S n bit
- zero 1 bit
- overflow 1 bit

Totale: n + 2 bit

Per dati su 32 bit, 5 operazioni:



Capacità della ROM: 2^{68} = 2.95.. x 10^{20} parole di 34 bit!

(Capacità della ROM per dati su 4 bit, 5 operazioni: 2^{12} = 4,098 parole di 6 bit)

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).

Sommario

Problemi dei sommatori

Sommatori ad anticipazione di riporto

Il firmware

Moltiplicatori firmware

Ottimizzazione dei moltiplicatori firmware

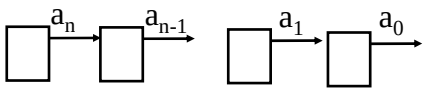
Shift (scalamento)

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

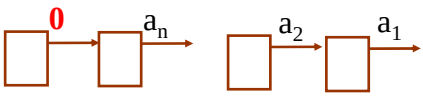
Effettuato al di fuori delle operazioni selezionate dal Mux della ALU, da un circuito denominato *Barrel shifter*.

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.

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.

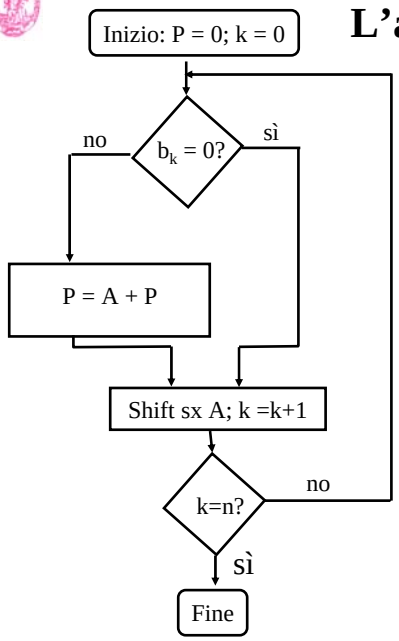
Itero per ogni bit del moltiplicatore:

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

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

```
  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
----------------------
      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
```

P1 = 0 + A

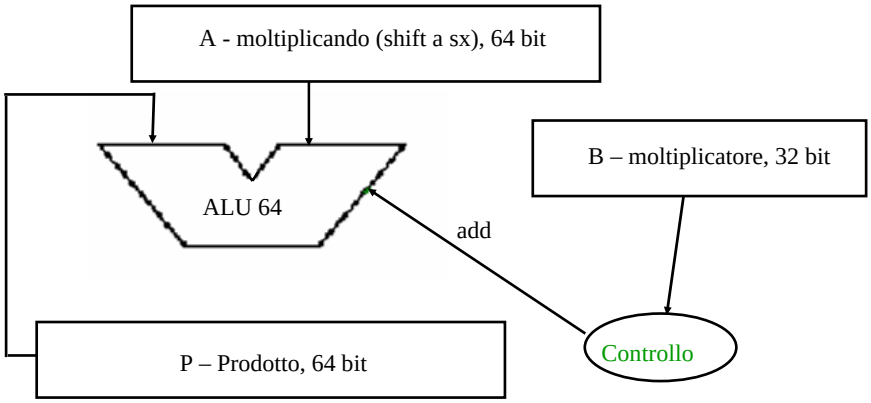
A1 = A*2

P2 = P1 + A1

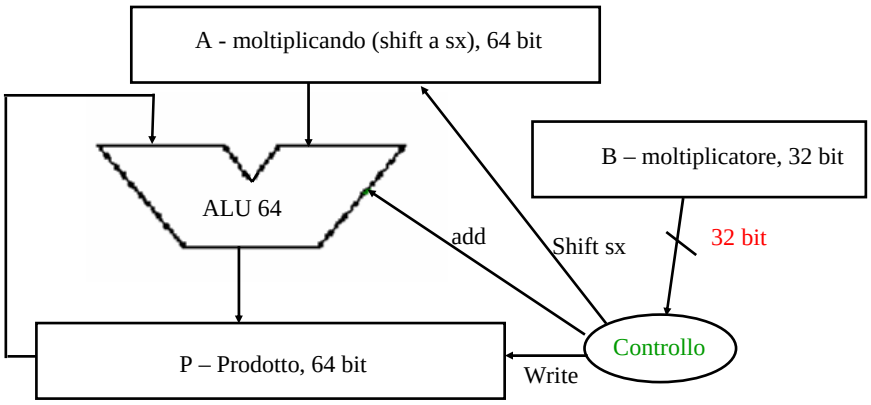
A2=A1*2=A*4

P3 = P2 + A2

Implementazione circuitale – prima parte

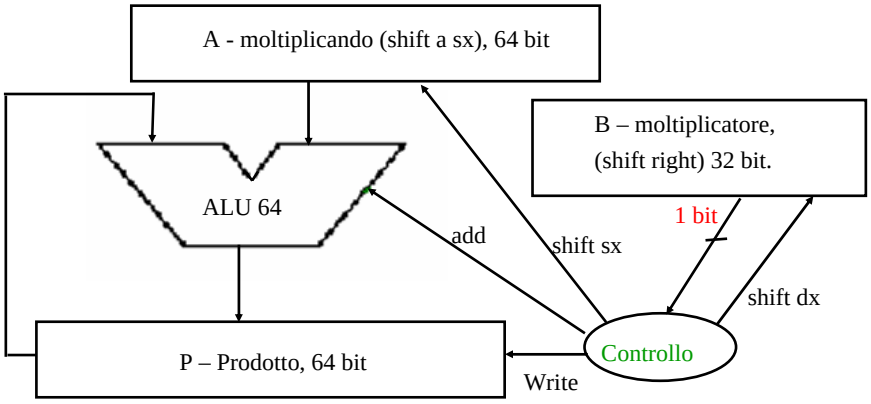


Implementazione circuitale – seconda parte



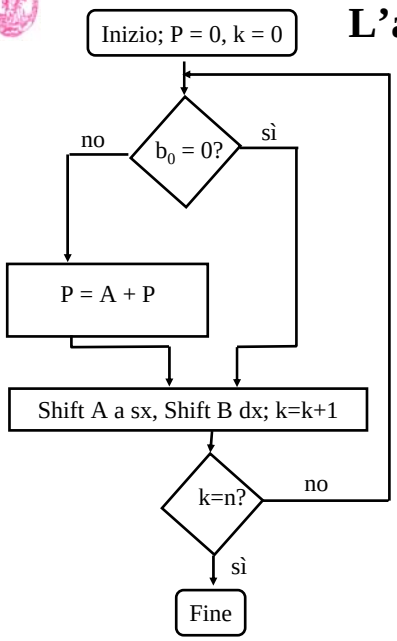
Qual'è il problema?

Implementazione circuitale finale



Qual'è il problema?

L'algoritmo



A → 1 1 0 1 1 x
B → 1 1 1 =

---

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

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 =
-----------------------
      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
```

Seconda implementazione

1ª implementazione

2ª implementazione

1 1 0 1 1

1 1 0 1 1

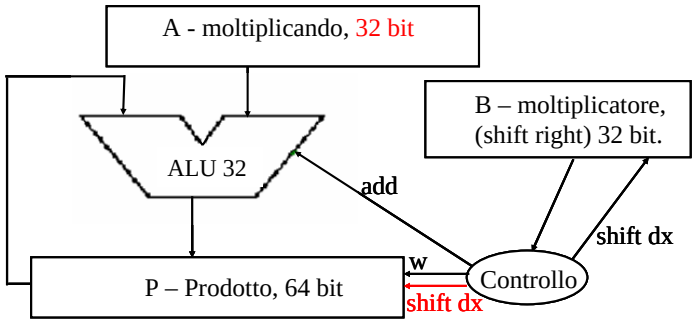
Sposto a sx il moltiplicando | Sposto a dx il prodotto

P[1] primo prodotto parziale

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 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 =
---
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

Sommario

Problemi dei sommatori

Sommatori ad anticipazione di riporto

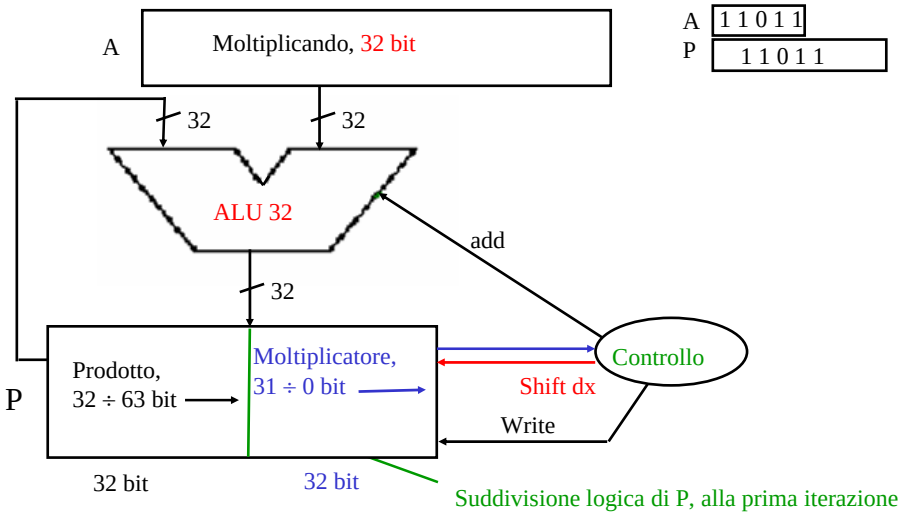
Addizionatori modulari

Il firmware

Moltiplicatori firmware

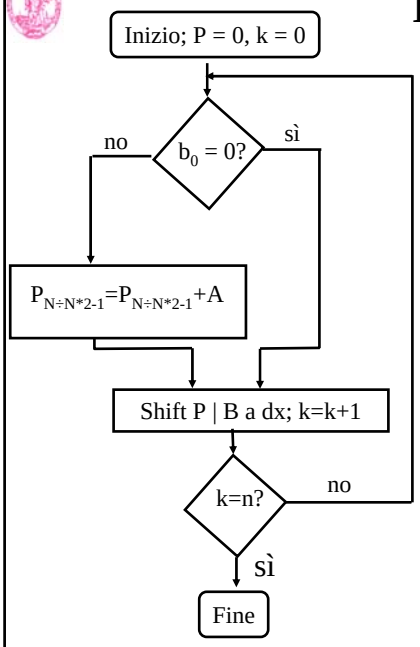
Ottimizzazione dei moltiplicatori firmware

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



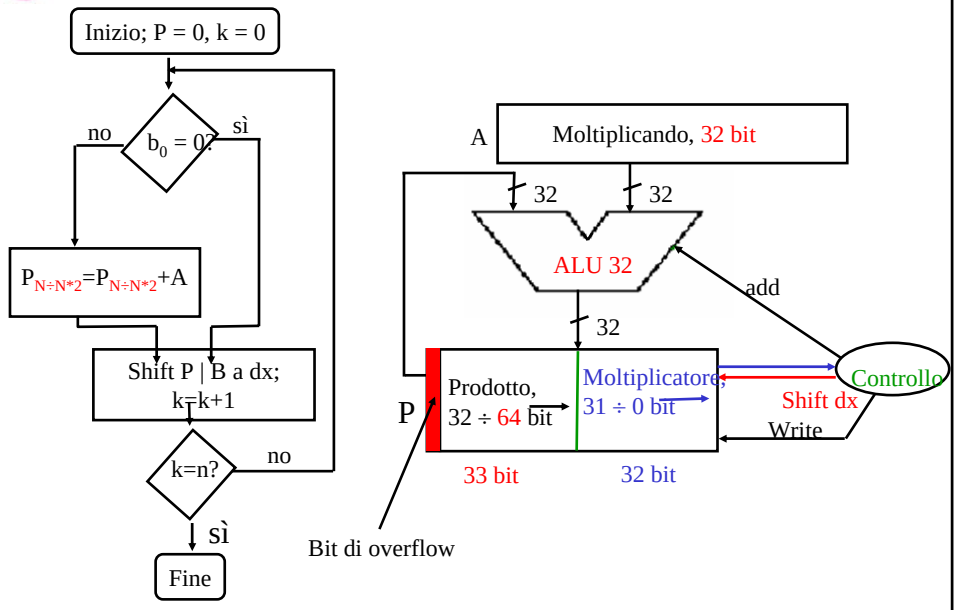

```
A ──►      1 1 0 1 1 x
B ──►        1 1 1 =
     ---------------------
           0 0 0 0 0 +
           1 1 0 1 1
     ---------------------
         1 1 1 1 1
           1 1 0 1 1 +
         1 1 0 1 1 -
     ---------------------
       1
         1 0 1 0 0 0 1 +
         1 1 0 1 1 - -
      ----------------------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


Esempio di esecuzione dell'algoritmo ottimizzato

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.

L'overflow



Sommario

Problemi dei sommatori

Sommatori ad anticipazione di riporto

Addizionatori modulari

Il firmware

Moltiplicatori firmware

Ottimizzazione dei moltiplicatori firmware