



# Carry look-ahead Adder & Firmware Multiplier

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: C.6 & 3.4



## Sommario

### Problemi dei sommatore

Sommatori ad anticipazione di riporto

I problemi del moltiplicatore firmware

Ottimizzazione dei moltiplicatori firmware

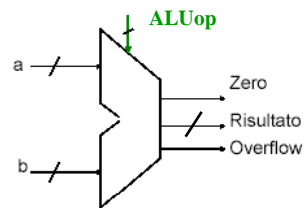


## ALU a 32 bit: struttura finale

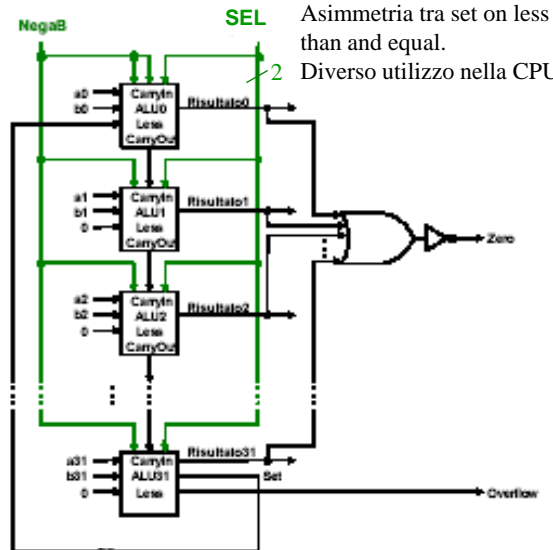


Operazioni possibili:

- 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



$$\begin{array}{r}
 111 \quad \leftarrow \text{Riporto} \\
 1011 + \quad \leftarrow \text{Addendo 1} \\
 110 = \quad \leftarrow \text{Addendo 2} \\
 \hline
 10001
 \end{array}$$

3 Attori: addendo 1, addendo 2, riporto.

Viene eseguita sequenzialmente da dx a sx.

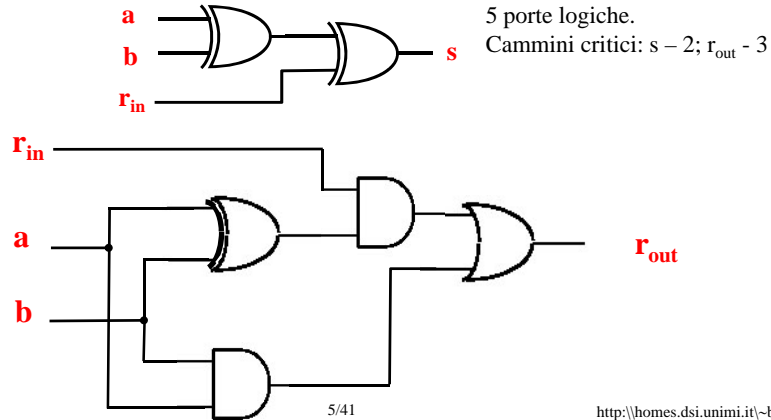


# Full adder



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

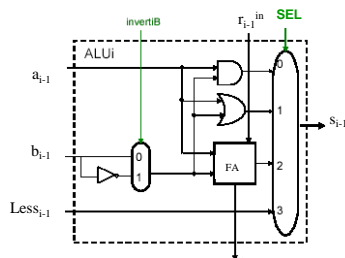
$$r_{out} = ab + (a \oplus b)r_{in}$$



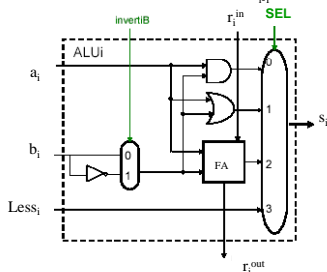
# Circuito della somma



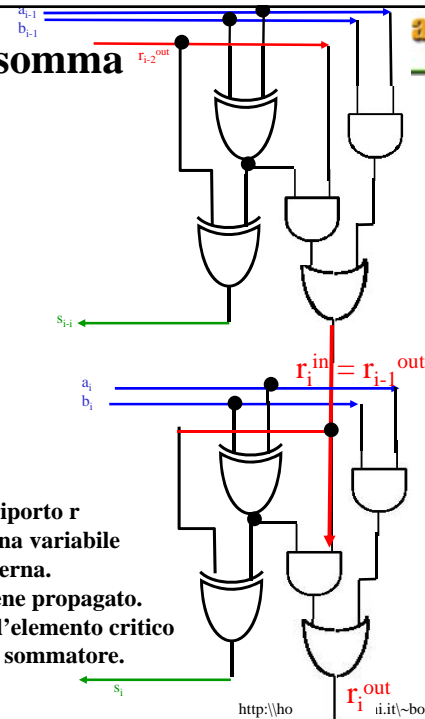
Stadio i-1



Stadio i



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



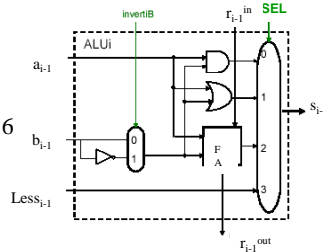


## Cammini critici

Per ogni stadio:  
Somma: 2  
Riporto: 3

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

Riporto =  $3 * N$

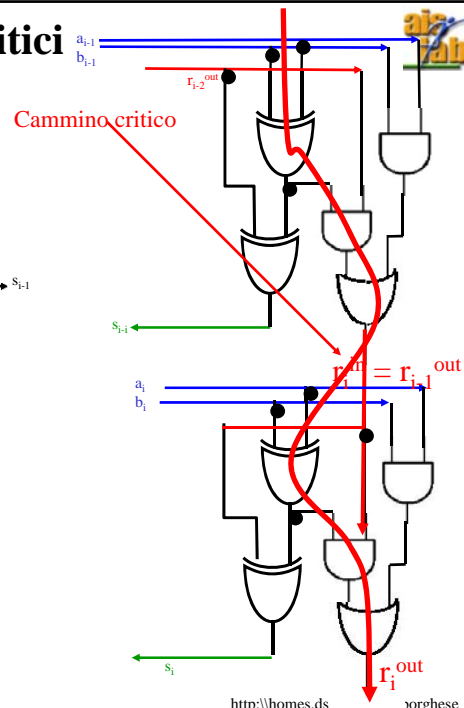


```

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 dispositivi
- Soluzioni  
modificare l'algoritmo  
modificare i dispositivi



## Sommario



Problemi dei sommatore

**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?

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

Scrivo la tabella della verità dove in uscita ho gli N riporti ed  
In ingresso  $2 * N$  valori (gli N bit dei 2 addendi).

La tabella della verità ha  $2^{2N}$  righe (per  $N=32, \dots$ )



## Carry look-ahead (anticipazione di riporto)

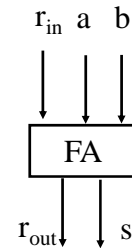


Approccio strutturato per diminuire la latenza della somma.

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

### 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



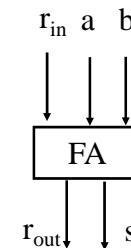
## Propagazione e generazione



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

### Osservazioni:

- Viene generato un riporto dallo stadio  $i$ , qualsiasi sia il riporto in ingresso se  $a = b = 1 \Rightarrow g_i = a_i b_i$ .
- Viene generato un riporto allo stadio  $i$ , se il riporto in ingresso è  $= 1$  ed una delle due variabili in ingresso è  $= 1 \Rightarrow$  se  $p_i = (a_i \oplus b_i) \Rightarrow$  viene generato riporto se  $p_i r_i^{in} = 1$  ( $p_i$  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  $r_0^{in} = 0$ .

$r_{in}$	$0000000$	$0111000$	$0111000$
$a$	$10101101+$	$10101101+$	$10111101+$
$b$	$10000-$	$11010-$	$11000=$
	$10111111$	$11100111$	$11010101$

$$r_5^{in} = r_4^{out} = 0$$

$$r_5^{in} = r_4^{out} = 1$$

$$r_5^{in} = r_4^{out} = 1$$

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



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

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

$$r_0^{out} = g_0 + p_0 r_0^{in}$$

$$r_1^{out} = g_1 + p_1 r_1^{in} = g_1 + p_1 g_0 + p_1 p_0 r_0^{in}$$

$$\begin{array}{r}
 r_1^{out} \swarrow 111 \\
 1001 + \\
 10 = \\
 \hline
 1100
 \end{array}$$

$$g_0 = 0$$

$$p_0 = p_1 = 1$$

$$\begin{array}{r}
 r_1^{in} \swarrow 110 \\
 1001 + \\
 11 = \\
 \hline
 1100
 \end{array}$$

$$g_0 = 1$$

$$p_1 = 1$$

$$\begin{array}{r}
 r_1^{out} \swarrow 10 \\
 1010 + \\
 11 = \\
 \hline
 1100
 \end{array}$$

$$g_1 = 1$$



## Sviluppo della funzione logica riporto



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

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

$$r_0 = g_0 + p_0 r_0$$

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

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

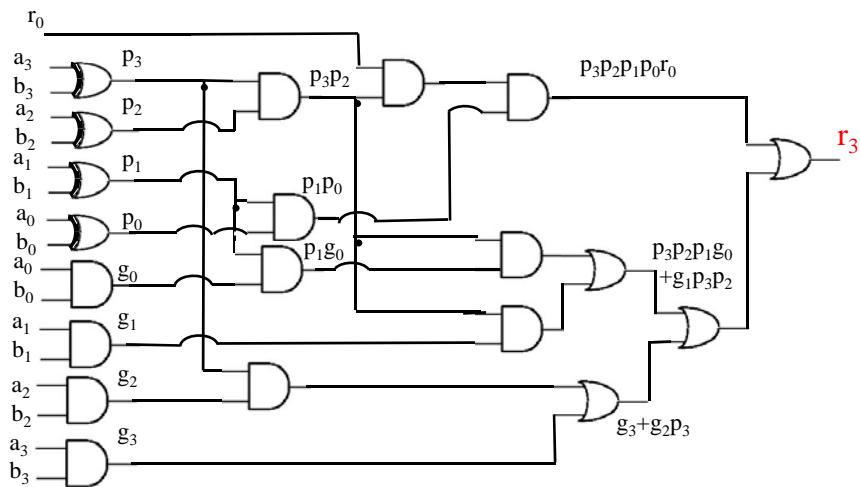
$$r_3 = g_3 + p_3 r_2 = g_3 + p_3 (g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 r_0) = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 r_0$$



## 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  $3 * 4 = 12$





## Quanto si guadagna con l'anticipazione del riporto?



Cammino critico per le variabili interne:

$$r_2^{\text{out}} \Rightarrow 5$$

$$r_1^{\text{out}} \Rightarrow 4$$

$$r_0^{\text{out}} \Rightarrow 3$$

Cammino critico per le variabili esterne:

$$r_3^{\text{out}} \Rightarrow 6$$

$$s_3 \Rightarrow 6 \text{ NB la prima porta XOR è in comune con } r_2^{\text{out}}$$

$$s_2 \Rightarrow 5 \text{ NB la prima porta XOR è in comune con } r_1^{\text{out}}$$

$$s_2 \Rightarrow 4 \text{ NB la prima porta XOR è in comune con } r_0^{\text{out}}$$

$$s_0 \Rightarrow 2$$



## Addizionatori modulari



La complessità del circuito è tollerata per piccoli n.

Circuiti sommatore indipendenti si hanno per 4 bit.

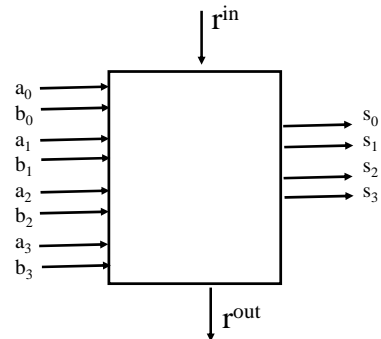
Moduli elementari.

Come si ottiene la somma?

Collegando in cascata i moduli (sommatore elementari).

Cammino critico =  $6 * N/4$ . Per 32 bit, 48.

Per confronto, senza parallelizzazione, per 32 bit,  $N * 3 = 96$ .



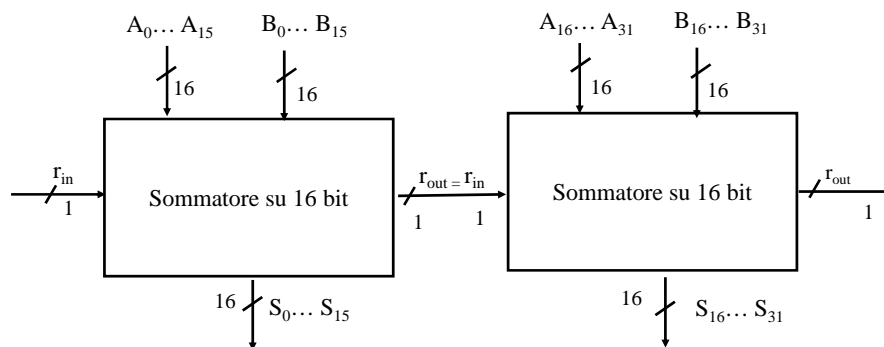


## Addizionatori modulari::esempio



Occorre sommare 2 variabili, A e B, su  $N = 32$  bit  
Ho a disposizione due sommatore su 16 bit.

Come si ottiene la somma?



## Sommario



Problemi dei sommatore

Sommatori ad anticipazione di riporto

**I problemi del moltiplicatore firmware**

Ottimizzazione dei moltiplicatori firmware



## 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).



## 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
	1 1 0 1 1 - -
	-----
Prodotto	1 0 0 0 0 1 1 1



## Shift (scalamento)

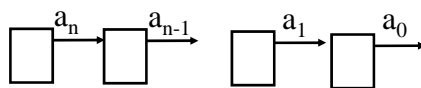


Dato A su 32 bit:  $a_j = a_{j-k}$  k shift amount ( $>$ ,  $=$ ,  $<$  0).

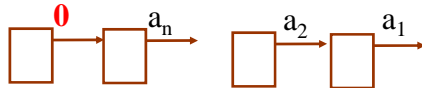
Effettuato al di fuori delle operazioni selezionate dal Mux della ALU, da un circuito denominato *Barrel shifter*.

Tempo comparabile con quello della somma.

Operazioni codificate in modo specifico nell'ISA.



Shift dx 1



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



## Moltiplicazione utilizzando somma e shift



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

$$\begin{array}{r} 11011 \times A \\ 111 = B \end{array}$$

$$\begin{array}{r} \text{-----} \\ 00000+ \\ 11011 \\ \text{-----} \end{array}$$

Itero per ogni bit del moltiplicatore:

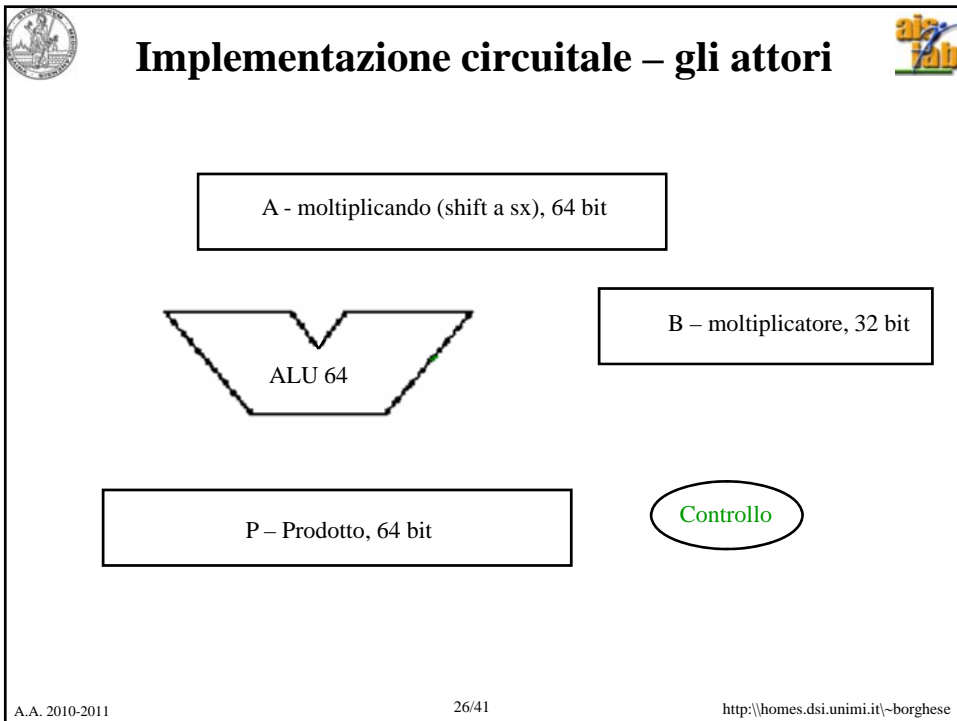
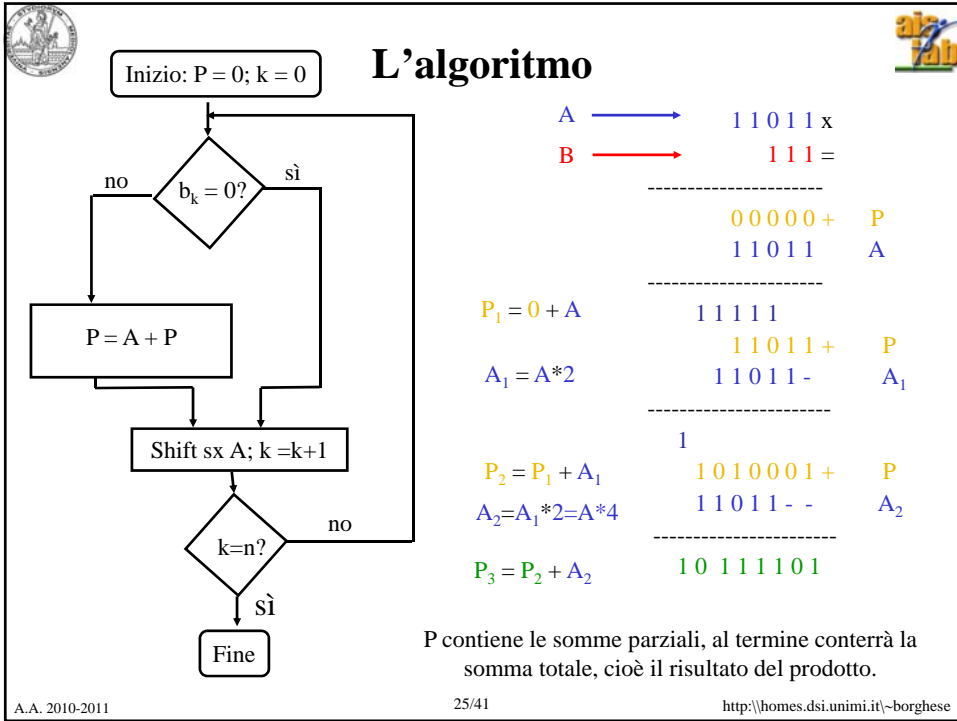
A) Sommo il moltiplicando al prodotto se il bit = 1.

$$\begin{array}{r} 11111 \\ 11011+ \quad P \\ 11011- \quad A \end{array}$$

B) Shift a sx di un bit il moltiplicando

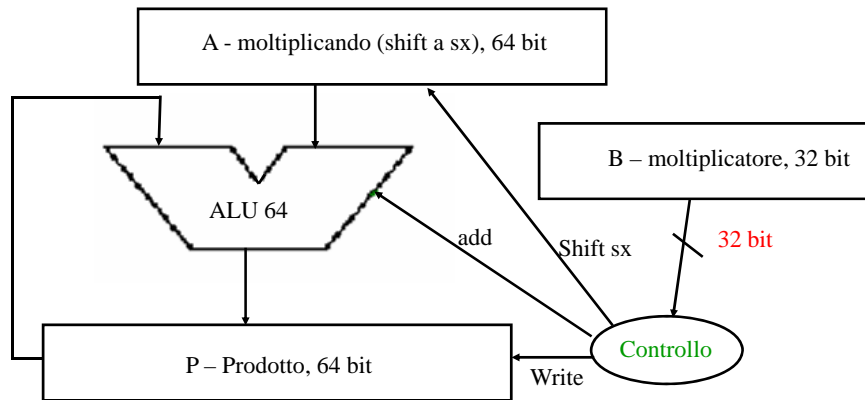
( $A' = A * \text{base}$ ).

$$\begin{array}{r} 1 \\ 1010001+ \\ 11011- - \\ \text{-----} \\ 10111101 \end{array}$$





## Implementazione circuitale



Qual'è il problema?



## Esercizi



Costruire il circuito HW che esegui la moltiplicazione  $7 \times 9$  in base 2.

Eseguire la stessa moltiplicazione secondo l'algoritmo visto, indicando passo per passo il contenuto dei 3 componenti: A che contiene il moltiplicando, B che contiene il moltiplicatore e P che contiene somme parziali ed il risultato finale.



## Sommario



Problemi dei sommatori

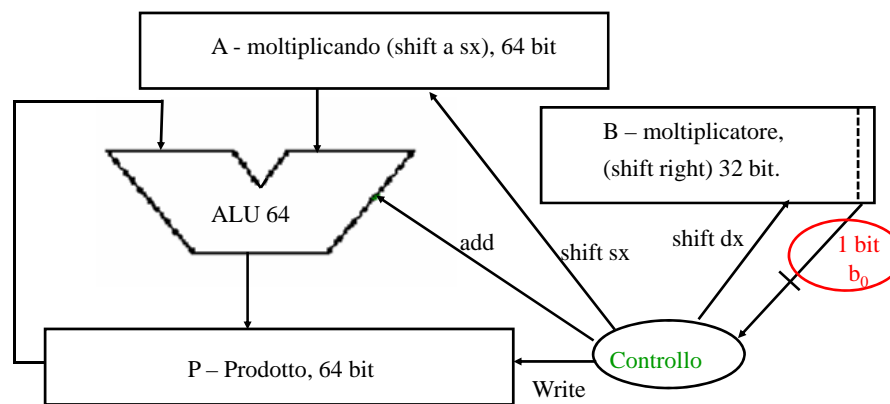
Sommatori ad anticipazione di riporto

I problemi del moltiplicatore firmware

Ottimizzazione dei moltiplicatori firmware

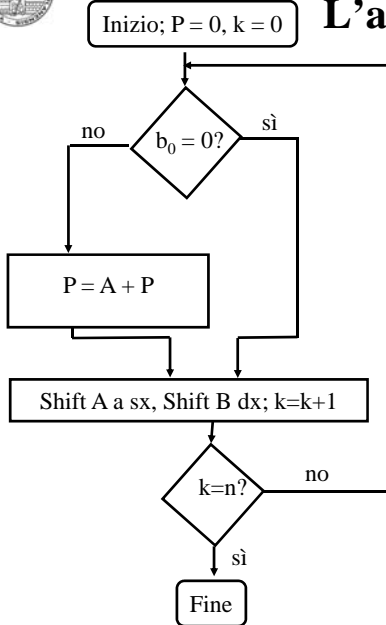


## Implementazione circuitale ottimizzata - I





## L'algoritmo - I



$$\begin{array}{r}
 A \xrightarrow{\text{blue}} 11011x \\
 B \xrightarrow{\text{red}} 111 = \\
 \hline
 00000+ \quad P \\
 11011 \quad A \\
 \hline
 11111 \\
 11011+ \quad P \\
 11011- \quad A^1 \\
 \hline
 1 \\
 1010001+ \quad P \\
 11011- - \quad A^2 \\
 \hline
 P \xrightarrow{\text{green}} 10111101
 \end{array}$$



## Esempio - I



Iterazione	Passo	Moltiplicatore	Moltiplicando	Prodotto
0	Valori iniziali	0010	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	0000	0000 0100	0000 0010
2	1a: 1 ⇒ Prod = Prod + Mcando	0001	0000 0100	0000 0100
	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

$$\begin{array}{r}
 0010 \times \\
 0011 = \\
 \hline
 \end{array}$$

Moltiplicazione su 4 bit.





## 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.

$$\begin{array}{r}
 11011x \\
 111= \\
 \hline
 00000+ \quad P \\
 11011 \quad A \\
 \hline
 11111 \\
 11011+ \quad P \\
 11011- \quad A^1 \\
 \hline
 1 \\
 1010001+ \quad P \\
 11011- - \quad A^2 \\
 \hline
 10111101
 \end{array}$$



## Implementazione ottimizzata - II



1<sup>a</sup> implementazione

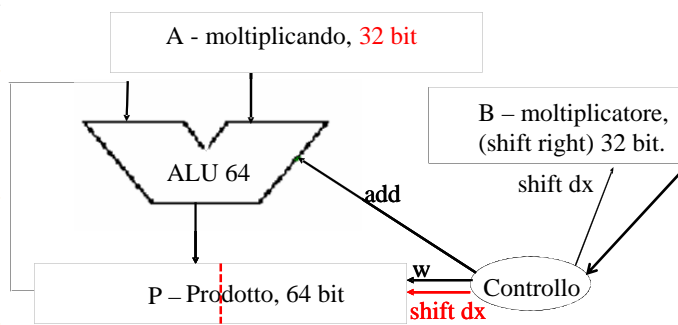
2<sup>a</sup> implementazione

11011  
11011

Sposto a sx il moltiplicando | Sposto a dx il prodotto  
P<sup>1</sup> primo prodotto parziale

11011  
11011

$$\begin{array}{r}
 11011x \\
 111- \\
 \hline
 00000+ \\
 11011 \\
 \hline
 11111 \\
 11011+ \\
 11011- \\
 \hline
 1 \\
 1010001+ \\
 11011- - \\
 \hline
 - \\
 10111101
 \end{array}$$



Qual'è il problema?

## Implementazione ottimizzata - II

1<sup>a</sup> implementazione

```

1 1 0 1 1 x
  1 1 0 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

```

2<sup>a</sup> implementazione

```

1 1 0 1 1
1 1 0 1 1
-----
1 1 0 1 1

```

Sposto a sx il moltiplicando | Sposto a dx il prodotto  
P<sup>1</sup> primo prodotto parziale

A - moltiplicando, 32 bit  
B - moltiplicatore, (shift right) 32 bit.  
ALU 32  
P - Prodotto, 64 bit  
Controllo

Parte modificabile      Parte non modificabile

A.A. 2010-2011      35/41      <http://homes.dsi.unimi.it/~borghese>

## 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).

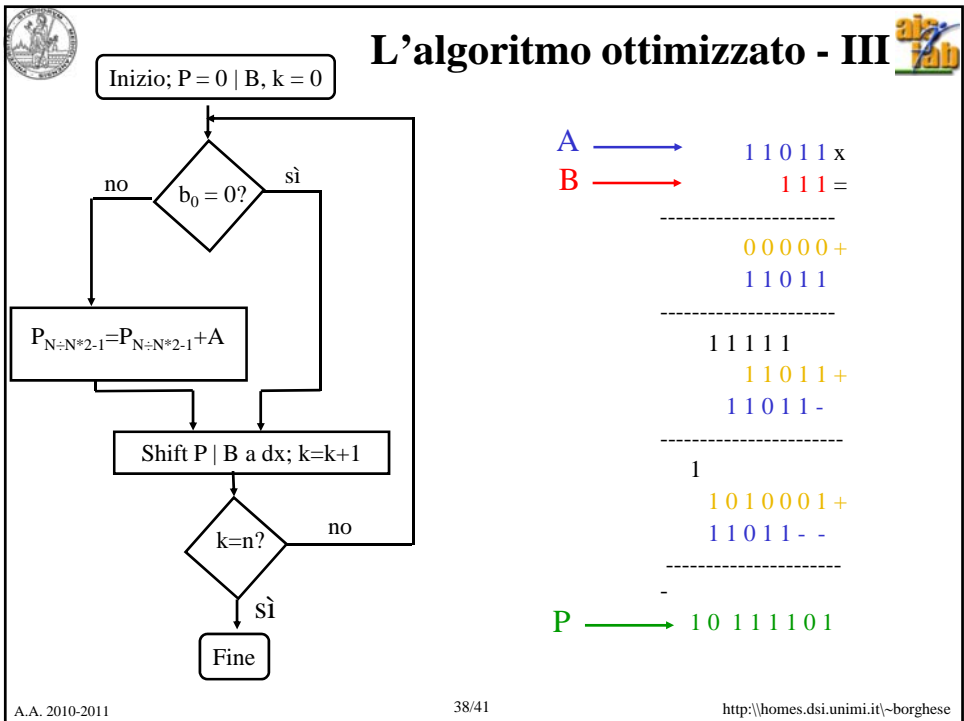
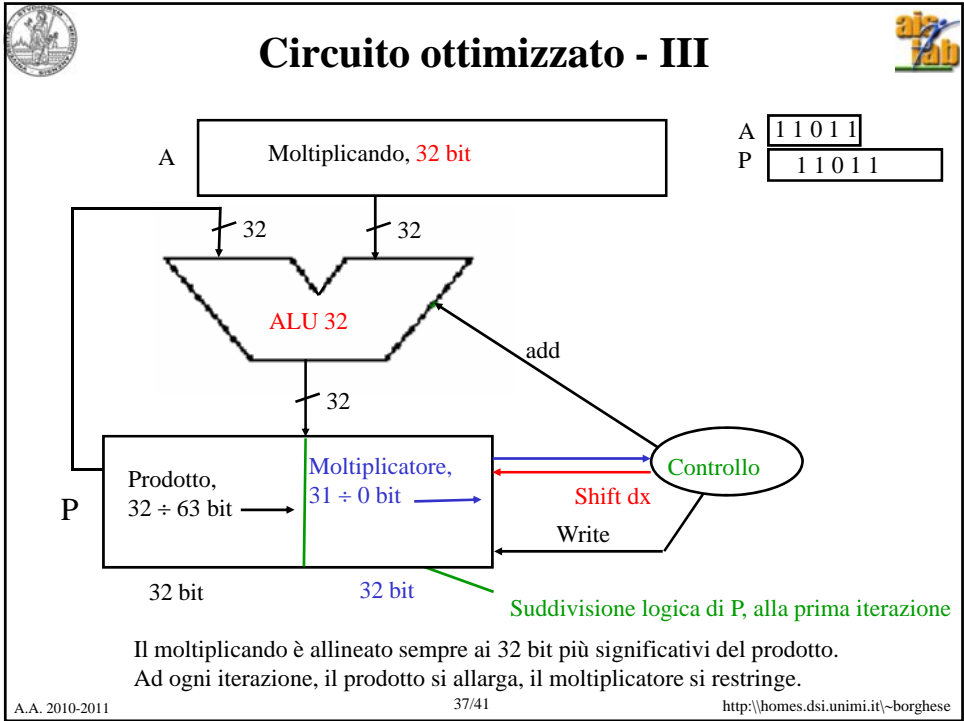
Si può perciò eliminare il registro moltiplicatore.

```

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

```

A.A. 2010-2011      36/41      <http://homes.dsi.unimi.it/~borghese>





## Esempio di esecuzione dell'algoritmo ottimizzato - III

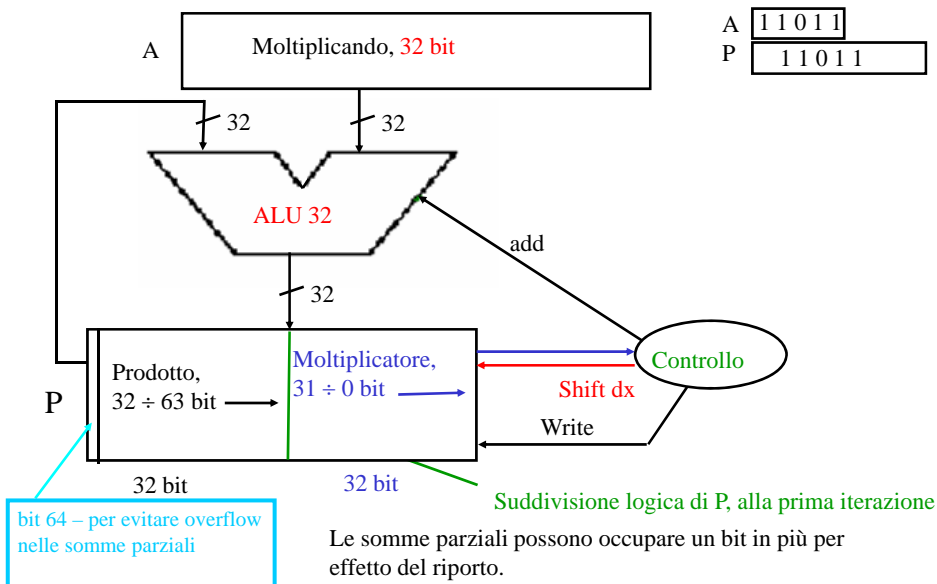


Iterazione	Passo	Moltiplicando	Prodotto
0	Valori iniziali	0010	0000 0010
1	1a: 1 $\Rightarrow$ Prod = Prod + Mcando	0010	0010 0011
	2: Scala a destra Prodotto	0010	0001 0001
2	1a: 1 $\Rightarrow$ Prod = Prod + Mcando	0010	0011 0001
	2: Scala a destra Prodotto	0010	0001 1001
3	1: 0 $\Rightarrow$ Nessuna operazione	0010	0001 1000
	2: Scala a destra Prodotto	0010	0000 1100
4	1: 0 $\Rightarrow$ 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.



## Circuito finale – moltiplicatore firmware





## Sommario



Problemi dei sommatori

Sommatori ad anticipazione di riporto

I problemi del moltiplicatore firmware

Ottimizzazione dei moltiplicatori firmware