Firmware Multiplier

Prof. Alberto Borghese
Dipartimento di Informatica
borghese@di.unimi.it

Università degli Studi di Milano

Riferimenti sul Patterson 5a ed.: B.6 & 3.4

Sommario

Il moltiplicatore firmware

Ottimizzazione dei moltiplicatori firmware

L’approccio firmware

Nell’approccio firmware, viene inserita nella ALU una micro-architettura costituita da una unità di controllo, dei componenti di calcolo e dei registri.

L’unità di controllo attiva opportunamente le unità di calcolo e il trasferimento da/verso i registri. Approccio *“controllore-datapath”* in piccolo.

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. Inoltre è rigida («hard») e non si può adattare a implementare funzioni diverse.

La soluzione HW viene utilizzata per le operazioni frequenti: la velocizzazione di operazioni complesse che vengono utilizzate raramente non aumenta significativamente le prestazioni (legge di Amdahl) -> Si preferisce un’implementazione Firmware o Software.

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, modificando solamente l’unità di controllo.

Approcci tecnologici alla ALU

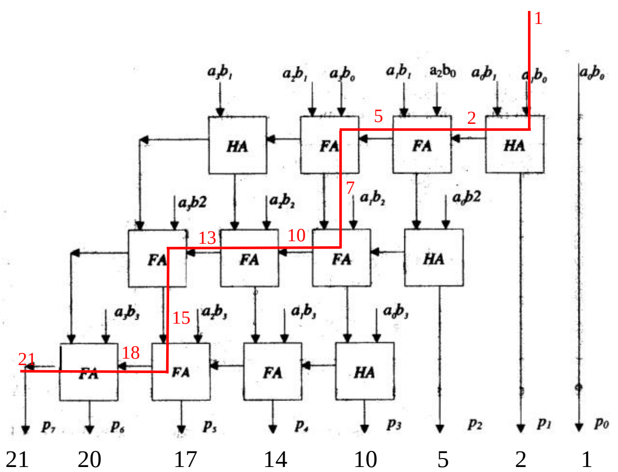
Quattro approcci tecnologici alla costruzione di una ALU (e di una CPU):

– Approccio hardware mediante porte logiche. E’ un approccio esaustivo (tabellare). Per ogni funzione, per ogni ingresso viene memorizzata l’uscita. E’ utilizzabile per funzioni molto particolari (ad esempio di una variabile). Non molto utilizzato.

– Approccio hardware programmabile (e.g. PLA, FPGA). Ad ogni operazione corrisponde un circuito combinatorio specifico.

– 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 collegando opportunamente i componenti, a partire dall’algoritmo che le implementa.

Circuito hardware della moltiplicazione



Se N = 4 cammino critico totale = 21

Come possiamo renderlo più flessibile? Come arrivare a un circuito che serva anche la divisione intera?

Algoritmo firmware per la moltiplicazione

Il razionale degli algoritmi firmware della moltiplicazione è il seguente.

Si analizzano sequenzialmente i bit del moltiplicatore e si creano i **prodotti parziali**:

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

2) Si mette una **copia del moltiplicando** (su n bit) 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 - -

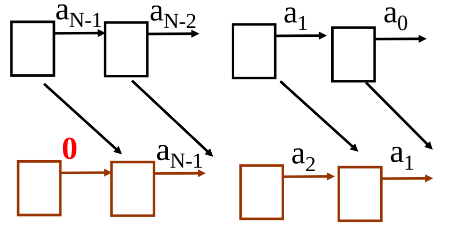
Prodotto 1 0 0 0 0 1 1 1

27 x 5 = 135

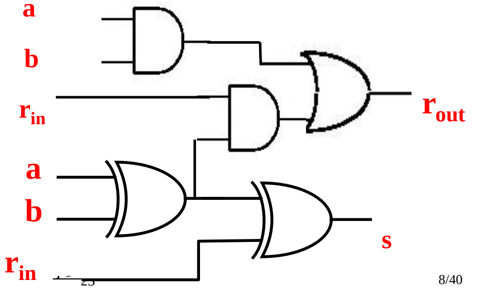
Shift (scalamento) e somma

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

Esempio. shift dx 1 di una parola su N bit:



Il bit a_0 si “perde”.
Il bit a_{N-1} = 0.
$a_k = a_{k+1}$ per k=0,1,2…. N-2



Somma di 2 bit
Utilizzo del riporto in ingresso
Produzione del riporto in uscita

Moltiplicazione utilizzando somma e shift

Analizzo sequenzialmente **ogni bit b_k del moltiplicatore**:

A0) Genero il primo e il secondo prodotto parziale

A1) Sommo il primo prodotto parziale al secondo e ottengo la prima somma parziale,

A2) Sommo il moltiplicando alla somma parziale corrente se $b_k = 1$.

A2) Sommo 0 al prodotto alla somma parziale se $b_k = 0$.

B) Shift a sx di un bit il moltiplicando a ogni passo (A' = A * 2).

27 x 11 = 297
27 + 54 + 0 + 216 = 297

```
          1 1 0 1 1 x        A
          0 1 0 1 1 =        B
----------------------
          1 1 0 1 1 +        A * 2^0
        1 1 0 1 1 - =        A * 2^1
----------------------
      1 0 1 0 0 0 1 +        S_1 ParzSum
      0 0 0 0 0 - - =        0* 2^2
----------------------
      1 0 1 0 0 0 1 +        S_2 ParzSum
    1 1 0 1 1 - - - =        A * 2^3
----------------------
  1 0 0 1 0 1 0 0 1 +        S_3 ParzSum
  0 0 0 0 0 - - - - =        0* 2^4
----------------------
  1 0 0 1 0 1 0 0 1          P  Prodotto
```

$1x2^8 + 1x2^5 + 1x2^3 + 1x2^0 =$

$256 + 32 + 8 + 1 = 297$

Moltiplicazione utilizzando somma e shift

Analizzo sequenzialmente **ogni bit b_k del moltiplicatore** e applico ad ogni passo le stesse operazioni.

Per ogni bit della parola (5 bit):

A1) Sommo il moltiplicando alla somma paziale corrente, P, se $b_k = 1$.

A2) Sommo 0 alla somma parziale corrente se $b_k = 0$.

B) Shift a sx di un bit il moltiplicando (A' = A * 2).

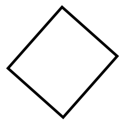
27 x 11 = 297

P contiene le somme parziali, al termine conterrà la somma totale, cioè il risultato del prodotto.

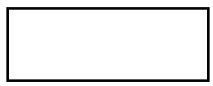
```
          1 1 0 1 1 x        A
          0 1 0 1 1 =        B
----------------------
0 0 0 0 0 0 0 0 0 0 +        Initial P=0
          1 1 0 1 1 =        A * 2^0
----------------------
0 0 0 0 0 1 1 0 1 1 +        S_1=P+A*2^1
        1 1 0 1 1 - =        A * 2^1
----------------------
0 0 0 1 0 1 0 0 0 1 +        S_2=S_1+A*2^2
      0 0 0 0 0 - - =        0* 2^2
----------------------
0 0 1 0 1 0 0 0 1 +          S_3=S_2
    1 1 0 1 1 - - - =        A * 2^3
----------------------
  1 0 0 1 0 1 0 0 1 +        S_4 =S_3+A*2^4
  0 0 0 0 0 - - - - =        0* 2^4
----------------------
  1 0 0 1 0 1 0 0 1          Final P=S_4+0
```

Diagrammi di flusso (flow chart)

Inizio / terminazione

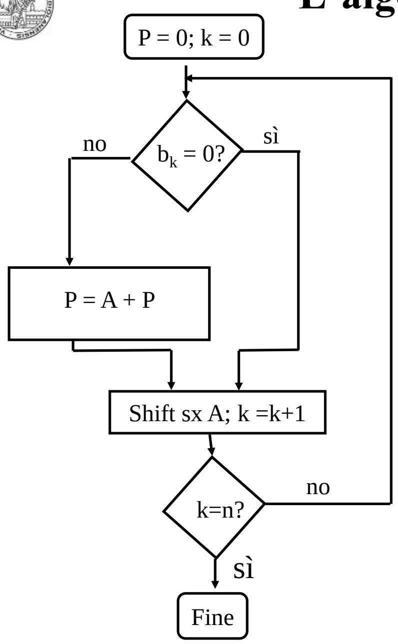


Test



Processo (esecuzione)

L'algoritmo



A → 1 1 0 1 1 x

B → 0 1 0 1 1 =

----

0 0 0 0 0 0 0 0 0 0 + Initial P=0

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

----

0 0 0 0 0 1 1 0 1 1 + S_1=P+A

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

----

0 0 0 1 0 1 0 0 0 1 + $S_2=S_1+A$

0 0 0 0 0 - - = $0 * 2^2$

----

0 0 1 0 1 0 0 0 1 + $S_3=S_2+0$

1 1 0 1 1 - - - = $A * 2^3$

----

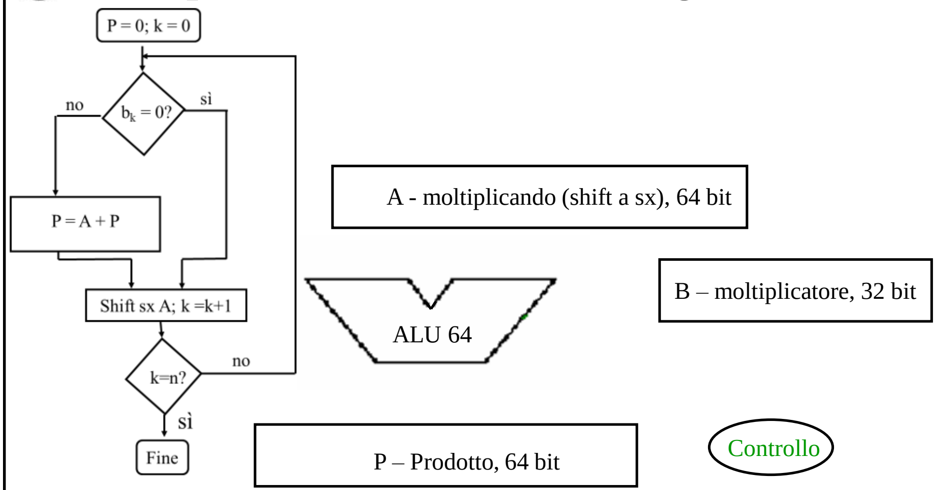
1 0 0 1 0 1 0 0 1 + $S_4 = S_3+A$

0 0 0 0 0 - - - - = $0 * 2^4$

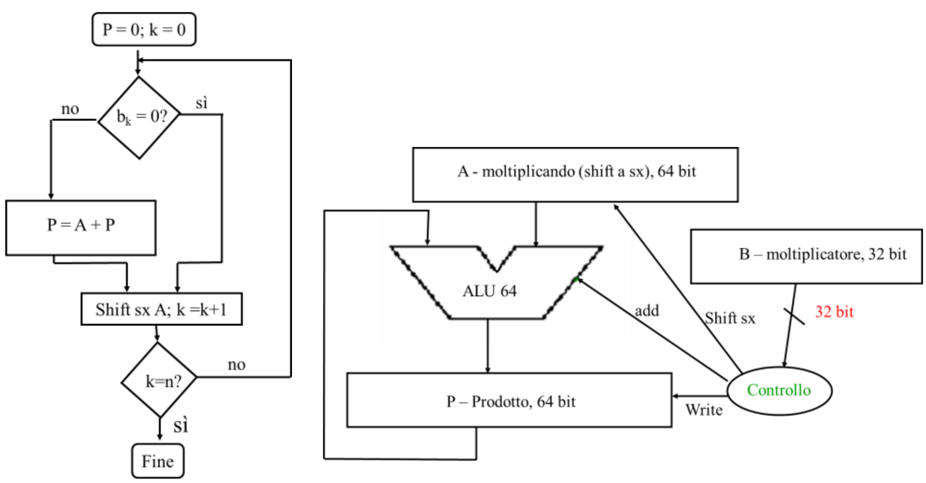
----

1 0 0 1 0 1 0 0 1 Final $P=S_4+0$

Implementazione circuitale – gli attori



Implementazione circuitale



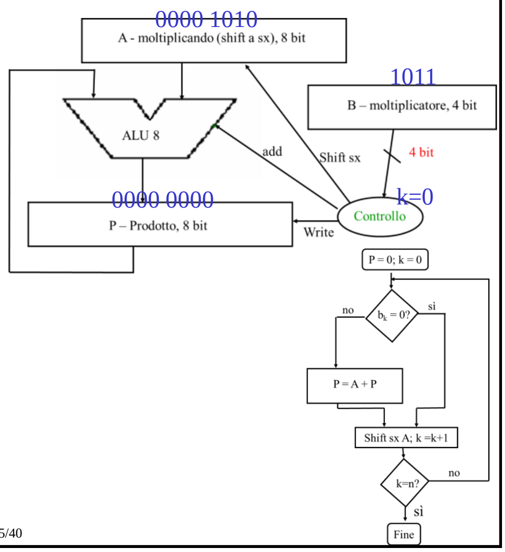
Qual'è il problema?

Esempio su 4 bit

Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000

A → 1 0 1 0 x 10_{10} x
B → 1 0 1 1 = 11_{10} =
----
0 0 0 0 0 0 0 0 +   P
$A = A * 2^0$ 1 0 1 0 = A^0
----
0 0 0 0 1 0 1 0 + S_1
$A^1 = A * 2^1$ 1 0 1 0 - A^1
----
0 0 0 1 1 1 1 0 + S_2
$A^2 = A * 2^2$ 0 0 0 0 - - A^2
----
0 0 0 1 1 1 1 0 + S_3
$A^3 = A * 2^2$ 1 0 1 0 - - - A^3

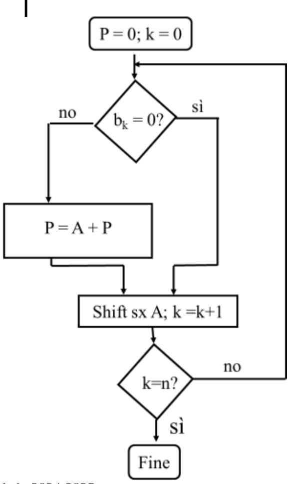
P → 0 1 1 0 1 1 1 0 110_{10}

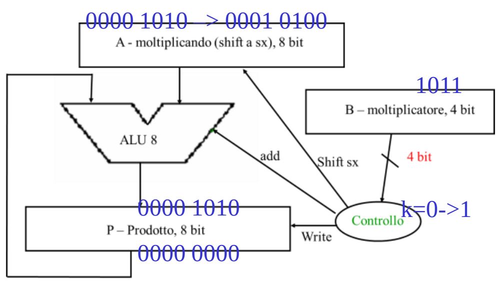


Esempio – passo 1

Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	b0=1->P=P+A	1011	0000 1010	0000 1010
	Moltiplicando << 1	1011	0001 0100	0000 1010

10*1+ 0
=10

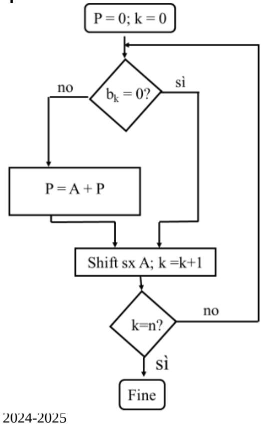


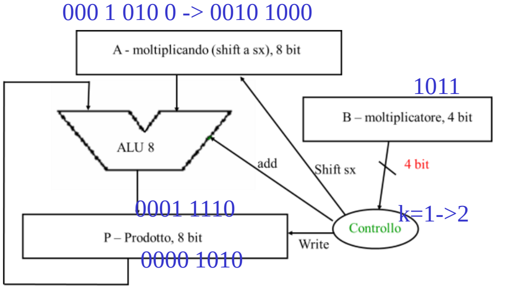


Esempio – passo 2

Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	b0=1->P=P+A	1011	0000 1010	0000 1010
	Moltiplicando << 1	1011	0001 0100	0000 1010
2	b1=1->P=P+A	1011	0001 0100	0001 1110
	Moltiplicando << 1	1011	0010 1000	0001 1110

10*2+10
= 30

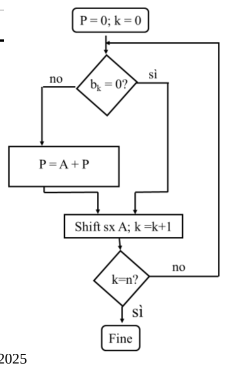


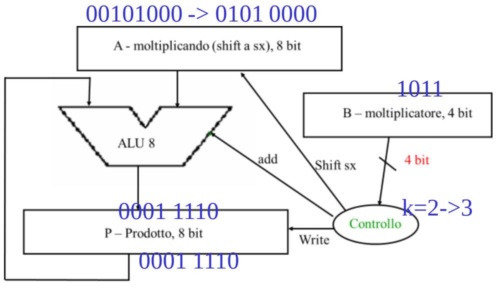


Esempio – passo 3

Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	b0=1->P=P+A	1011	0000 1010	0000 1010
	Moltiplicando << 1	1011	0001 0100	0000 1010
2	b1=1->P=P+A	1011	0001 0100	0001 1110
	Moltiplicando << 1	1011	0010 1000	0001 1110
3	b2=0->Nulla	1011	0010 1000	0001 1110
	Moltiplicando << 1	1011	0101 0000	0001 1110

0 + 30
= 30

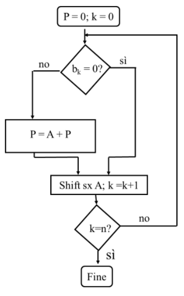


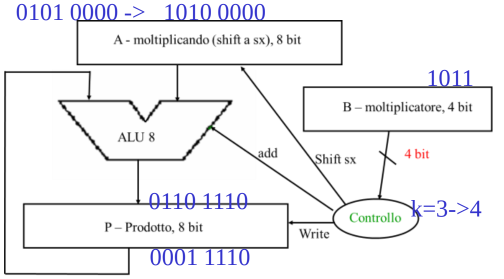


Esempio – passo 4

Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	b0=1->P=P+A	1011	0000 1010	0000 1010
	Moltiplicando << 1	1011	0001 0100	0000 1010
2	b1=1->P=P+A	1011	0001 0100	0001 1110
	Moltiplicando << 1	1011	0010 1000	0001 1110
3	b2=0->Nulla	1011	0010 1000	0001 1110
	Moltiplicando << 1	1011	0101 0000	0001 1110
4	b3=1->P=P+A	1011	0101 0000	0110 1110
	Moltiplicando << 1	1011	1010 0000	0110 1110

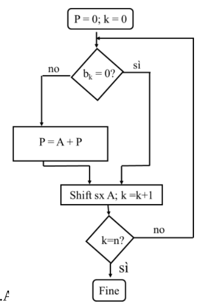
10*8 + 30
= 110

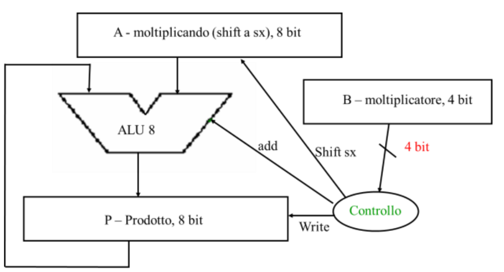




Esempio – riassunto

Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	b0=1->P=P+A	1011	0000 1010	0000 1010
	Moltiplicando << 1	1011	0001 0100	0000 1010
2	b1=1->P=P+A	1011	0001 0100	0001 1110
	Moltiplicando << 1	1011	0010 1000	0001 1110
3	b2=0->Nulla	1011	0010 1000	0001 1110
	Moltiplicando << 1	1011	0101 0000	0001 1110
4	b3=1->P=P+A	1011	0101 0000	0110 1110
	Moltiplicando << 1	1011	1010 0000	0110 1110





Esercizi

Costruire il circuito HW che esegui la moltiplicazione 7 x 9 in base 2 su 4 bit.

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

Sommario

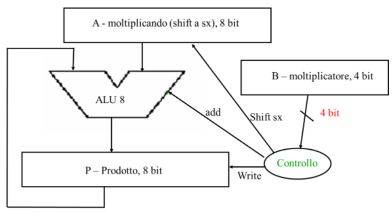
I moltiplicatori firmware

Ottimizzazione dei moltiplicatori firmware

Ottimizzazione

Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	b0=1->P=P+A	1011	0000 1010	0000 1010
	Moltiplicando << 1	1011	0001 0100	0000 1010
2	b1=1->P=P+A	1011	0001 0100	0001 1110
	Moltiplicando << 1	1011	0010 1000	0001 1110
3	b2=0->Nulla	1011	0010 1000	0001 1110
	Moltiplicando << 1	1011	0101 0000	0001 1110
4	b3=1->P=P+A	1011	0101 0000	0110 1110
	Moltiplicando << 1	1011	1010 0000	0110 1110

- Inizializzo B (ho tutti i bit di B)
- A ogni passo leggo B, ma utilizzo solo 1 bit, b_k
- Utilizzo b_k a ogni iterazione, **poi non serve più**.
- Non è necessario conservare tutti i bit di B per tutta la durata dell'operazione.

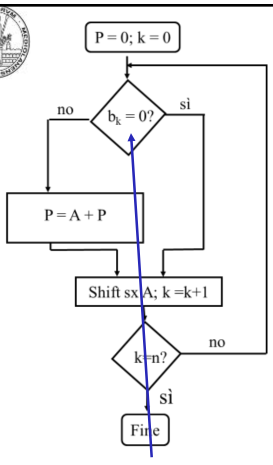


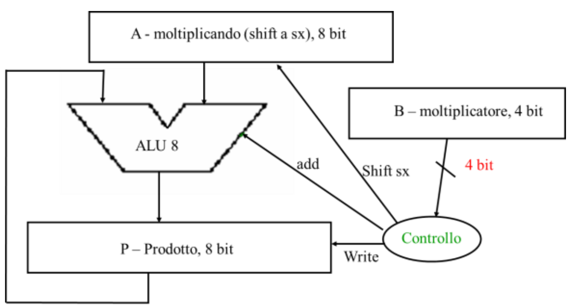
Situazione del tipo: **produttore-consumatore**.

Produco dei dati: la parola B.

Consumo i dati: 1 bit della parola B a ogni iterazione (consume = utilizzo una tantum e non riutilizzo in seguito.

Razionale - I





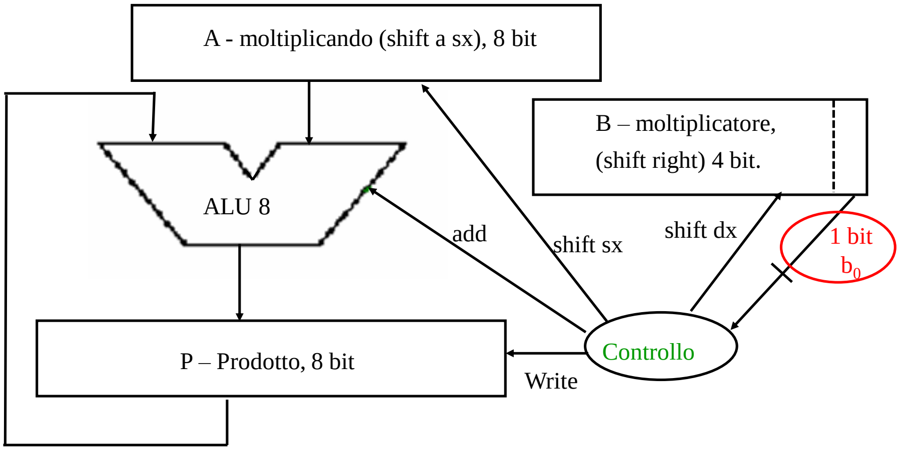
Per scegliere, **b_k**, a ogni passo k, serve un mux all'interno dell'unità di controllo.

Posso ottenere lo stesso effetto in altro modo:

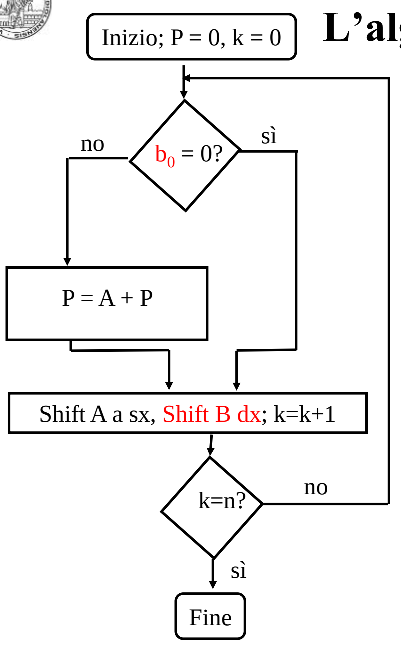
- Leggo b_0
- Shift a dx di una posizione di B

Espongo così all'unità di controllo b_k a ogni iterazione.

Implementazione circuitale ottimizzata - I



L’algoritmo - I

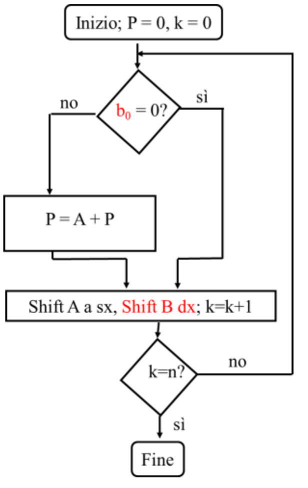


A → 1 0 1 0 x
B → 1 0 1 1 =
----
0 0 0 0 0 0 0 0 + P
$A = A * 2^0$ 1 0 1 0 = A^0
----
0 0 0 0 1 0 1 0 + P_1
$A^1 = A * 2^1$ 1 0 1 0 - A^1
----
0 0 0 1 1 1 1 0 + P_2
$A^2 = A * 2^2$ 0 0 0 0 - - A^2
----
0 0 0 1 1 1 1 0 + P_3
$A^3 = A * 2^2$ 1 0 1 0 - - - A^3

P → 0 1 1 0 1 1 1 0

Esecuzione - I

Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	b0=1->P=P+A	1011	0000 1010	0000 1010
	Moltiplicando << 1	1011	000 1010 0	0000 1010
	Moltiplicatore >> 1	0 101 1	000 1010 0	0000 1010
2	b0=1->P=P+A	0 101	000 1010 0	0001 1110
	Moltiplicando << 1	0 101	00 1010 00	0001 1110
	Moltiplicatore >> 1	00 10	00 1010 00	0001 1110
3	b0=0->Nulla	00 10 11	00 1010 00	0001 1110
	Moltiplicando << 1	00 10	0 1010 000	0001 1110
	Moltiplicatore >> 1	000 1	0 1010 000	0001 1110
4	b0=1->P=P+A	0001 011	0 1010 000	0110 1110
	Moltiplicando << 1	0000	1010 0000	0110 1110
	Moltiplicatore >> 1	0000	1010 0000	0110 1110



Razionale per una seconda implementazione

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

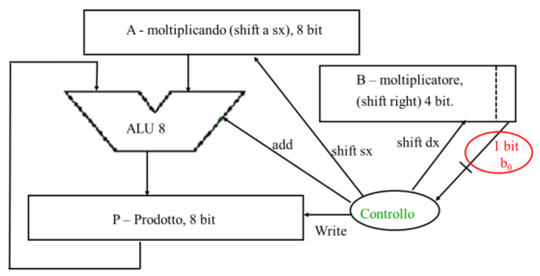
Gli N bit del moltiplicando sommati al registro prodotto vengono incolonnati di una posizione più a sinistra a ogni iterazione. Occupano N bit.

Ad ogni iterazione 1 bit del registro prodotto viene calcolato definitivamente.

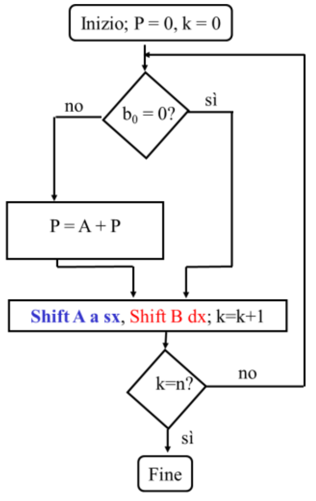
Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	b0=1->P=P+A	1011	0000 1010	0000 1010
	Moltiplicando << 1	1011	000 1010 0	0000 1010
	Moltiplicatore >> 1	0 101	000 1010 0	0000 1010
2	b0=1->P=P+A	0 101	000 1010 0	0001 1110
	Moltiplicando << 1	0 101	00 1010 00	0001 1110
	Moltiplicatore >> 1	00 10	00 1010 00	0001 1110
3	b0=0->Nulla	00 10	00 1010 00	0001 1110
	Moltiplicando << 1	00 10	0 1010 000	0001 1110
	Moltiplicatore >> 1	000 1	0 1010 000	0001 1110
4	b0=1->P=P+A	0001	0 1010 000	0110 1110
	Moltiplicando << 1	0000	1010 0000	0110 1110
	Moltiplicatore >> 1	0000	1010 0000	0110 1110

Analisi dello shift

A scorre rispetto al prodotto P che resta fermo



P scorre rispetto ad A che resta fermo



Razionale per una seconda implementazione

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

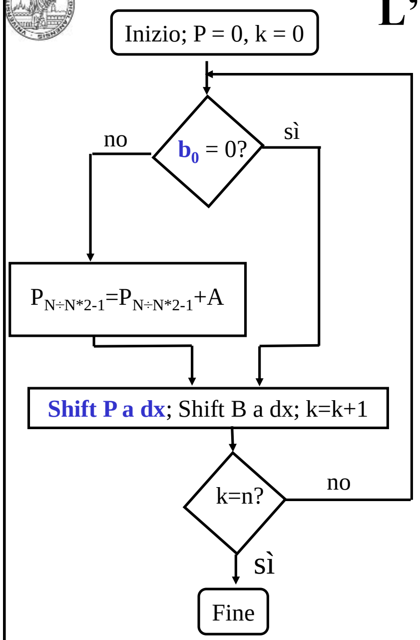
Spostamento di A a sx rispetto al registro prodotto, P.

Spostamento di P a dx rispetto al registro moltiplicando, A

Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	b0=1->P=P+A	1011	0000 1010	0000 1010
	Moltiplicando << 1	1011	000 1010 0	0000 1010
	Moltiplicatore >> 1	0 101	000 1010 0	0000 1010
2	b0=1->P=P+A	0 101	000 1010 0	0001 1110
	Moltiplicando << 1	0 101	00 1010 00	0001 1110
	Moltiplicatore >> 1	00 10	00 1010 00	0001 1110
3	b0=0->Nulla	00 10	00 1010 00	0001 1110
	Moltiplicando << 1	00 10	0 1010 000	0001 1110
	Moltiplicatore >> 1	000 1	0 1010 000	0001 1110
4	b0=1->P=P+A	000 1	0 1010 000	0110 1110
	Moltiplicando << 1	0000	1010 0000	0110 1110
	Moltiplicatore >> 1	0000	1010 0000	0110 1110

```
        1 0 1 0 x
        1 0 1 1 =
-----------------------
0 0 0 0 0 0 0 0 +      P
        1 0 1 0 =      A^0
-----------------------
0 0 0 0 1 0 1 0 +      P_1
      1 0 1 0 -        A^1
-----------------------
0 0 0 1 1 1 1 0 +      P_2
    0 0 0 0 - -        A^2
-----------------------
0 0 0 1 1 1 1 0 +      P_3
  1 0 1 0 - - -        A^3
-----------------------
0 1 1 0 1 1 1 0
```

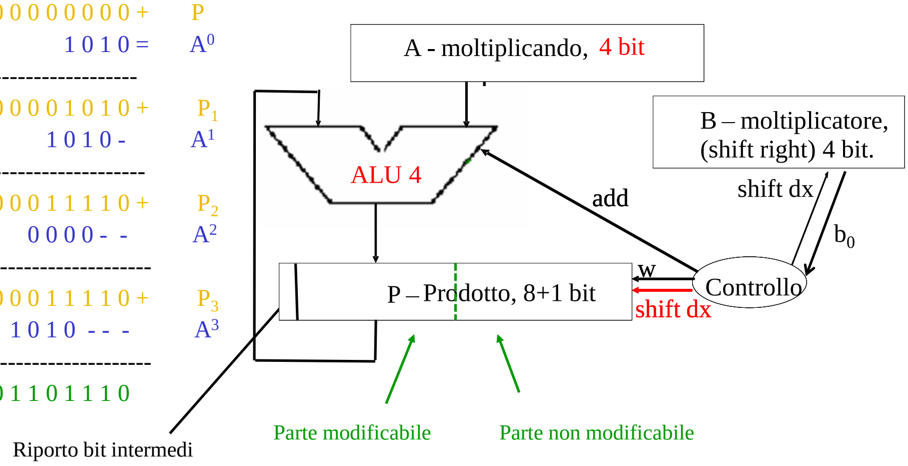
L’algoritmo ottimizzato - II




```
A →          1 0 1 0 x
B →          1 0 1 1 =
      ----------------------
        0 0 0 0 0 0 0 0 +    P
                1 0 1 0 =    A0
      ----------------------
        0 0 0 0 1 0 1 0 +    P1
              1 0 1 0 -      A1
      -----------------------
        0 0 0 1 1 1 1 0 +    P2
            0 0 0 0 - -      A2
      -----------------------
        0 0 0 1 1 1 1 0 +    P3
          1 0 1 0 - - -      A3
      -----------------------
P →     0 1 1 0 1 1 1 0
```

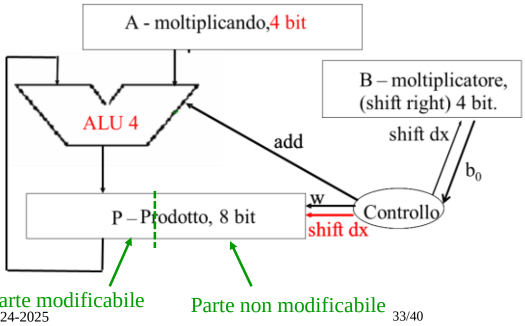

Implementazione ottimizzata - II

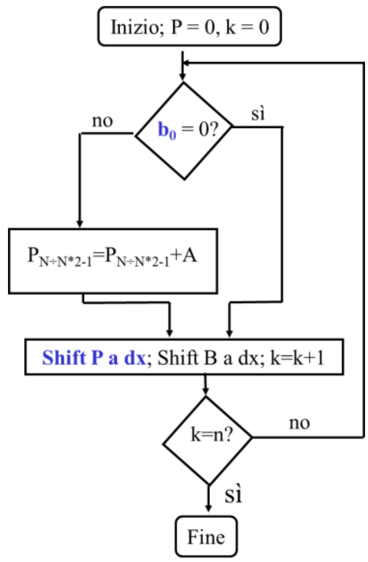
```
             1 0 1 0 x
             1 0 1 1 =
----------------------
  0 0 0 0 0 0 0 0 +    P
          1 0 1 0 =    A0
----------------------
  0 0 0 0 1 0 1 0 +    P1
        1 0 1 0 -      A1
-----------------------
  0 0 0 1 1 1 1 0 +    P2
      0 0 0 0 - -      A2
-----------------------
  0 0 0 1 1 1 1 0 +    P3
    1 0 1 0 - - -      A3
-----------------------
  0 1 1 0 1 1 1 0
```



Esecuzione - II

Iterazione	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	1010	0000 0000
1	b0=1->P=P+A	1011	1010	1010 0000
	Prodotto >> 1	1011	1010	0 1010 000
	Moltiplicatore >> 1	0 101	1010	0 1010 000
2	b0=1->P=P+A	0 101	1010	0 1010 000
	Prodotto >> 1	0 101	1010	0 11110 00
	Moltiplicatore >> 1	00 10	1010	0 11110 00
3	b0=0->Nulla	00 10	1010	0 11110 00
	Prodotto >> 1	00 10	1010	0 011110 0
	Moltiplicatore >> 1	000 1	1010	0 011110 0
4	b0=1->P=P+A	000 1	1010	1 101110 0
	Prodotto >> 1	0000	1010	0110 1110
	Moltiplicatore >> 1	0000	1010	0111 1110





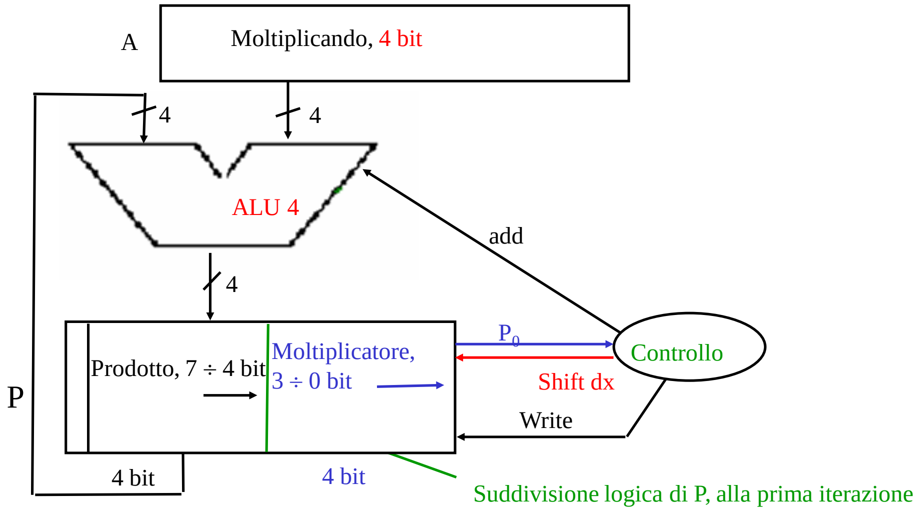
Razionale dell’implementazione - III

Il numero di bit del registro **prodotto** corrente (somma dei prodotti parziali) più il numero di bit da esaminare nel registro **moltiplicatore** rimane costante ad ogni iterazione (pari a 8 bit).

Si può perciò eliminare il registro moltiplicatore.

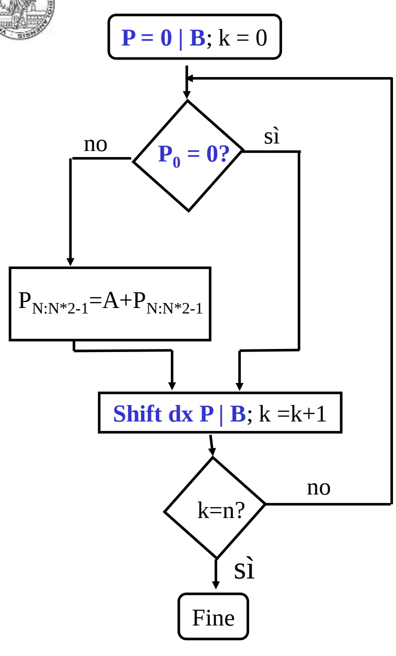
Iterazione	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	1010	0000 0000
1	b0=1->P=P+A	$\boxed{1011}$	1010	$\boxed{1010}$ 0000
	Prodotto >> 1	1011	1010	0 1010 000
	Moltiplicatore >> 1	0 $\boxed{101}$	1010	$\boxed{0\ 1010}$ 000
2	b0=1->P=P+A	0 101	1010	1 1110 000
	Prodotto >> 1	0 101	1010	0 11110 00
	Moltiplicatore >> 1	00 $\boxed{10}$	1010	$\boxed{0\ 11110}$ 00
3	b0=0->Nulla	00 10	1010	0 11110 00
	Prodotto >> 1	00 10	1010	0 011110 0
	Moltiplicatore >> 1	000 $\boxed{1}$	1010	$\boxed{0\ 011110}$ 0
4	b0=1->P=P+A	000 1	1010	1 101110 0
	Prodotto >> 1	0000	1010	0110 1110
	Moltiplicatore >> 1	0000	1010	$\boxed{0111\ 1110}$

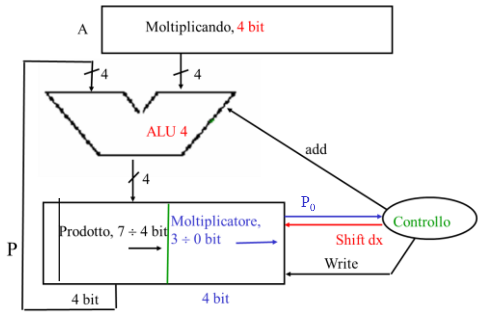
Circuito ottimizzato - III



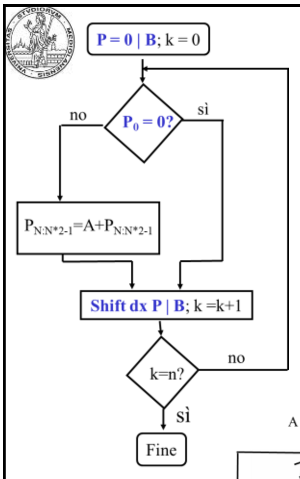
Il moltiplicando è allineato sempre ai 4 bit più significativi del prodotto.
Ad ogni iterazione, il prodotto si allarga, il moltiplicatore si restringe.

Algoritmo ottimizzato





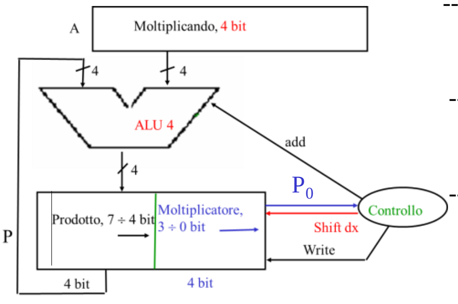
L'algoritmo ottimizzato - III



A → 1 0 1 0 x
B → 1 0 1 1 =

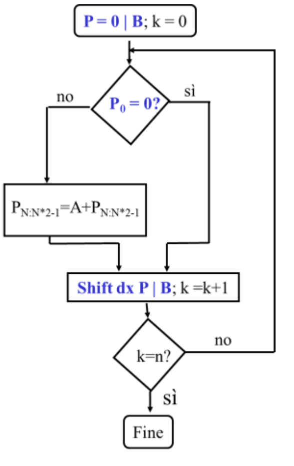
----
0 0 0 0 0 0 0 0 + P
1 0 1 0 = A^0
----
0 0 0 0 1 0 1 0 + P_1
1 0 1 0 - A^1
----
0 0 0 1 1 1 1 0 + P_2
0 0 0 0 - - A^2
----
0 0 0 1 1 1 1 0 + P_3
1 0 1 0 - - - A^3
----
0 1 1 0 1 1 1 0

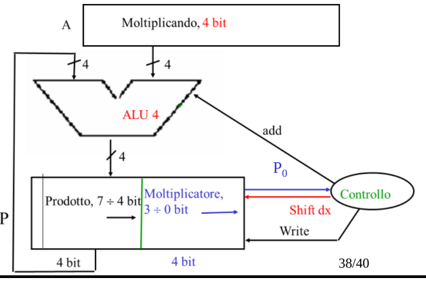
P



Esempio di esecuzione dell'algoritmo ottimizzato - III

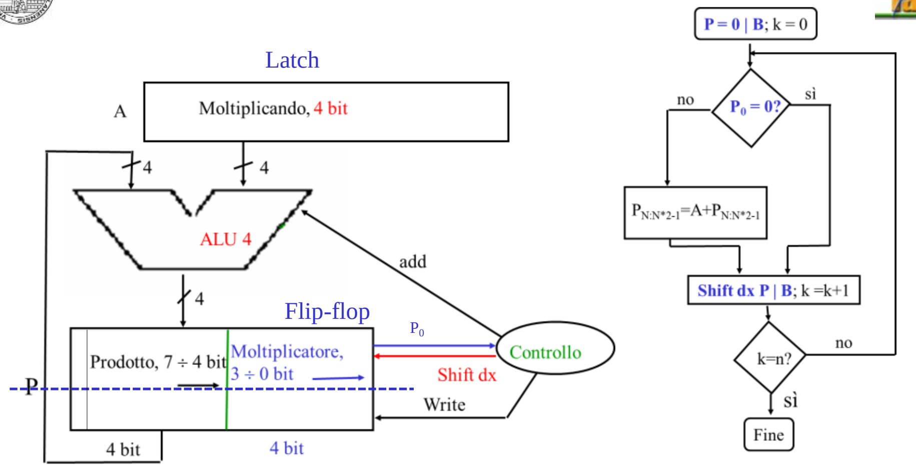
Iterazione	Passo	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1010	0000 1011
1	**p0=1**->P=P+A	1010	1010 1011
	Prodotto >> 1	1010	0 1010 101
2	**p0=1**->P=P+A	1010	1 1110 101
	Prodotto >> 1	1010	0 11110 10
3	**p0=0**->Nulla	1010	0 11110 10
	Prodotto >> 1	1010	0 011110 1
4	**p0=1**->P=P+A	1010	1 101110 0
	Prodotto >> 1	1010	0111 1110





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

Complessità



- Registro moltiplicando (4*4 = 16) – latch. Viene solo letto.
- Registro Prodotto ((8+1)*8 = 72) – Flip flop perchè registro a scorrimento e perchè il suo contenuto viene letto e scritto.
- ALU4 (5*4 = 20)
- UC ?

(Moltiplicatore HW aveva complessità 65 porte logiche), ma questo circuito può essere utilizzato anche per la divisione…

Sommario

I moltiplicatori firmware

Ottimizzazione dei moltiplicatori firmware