



Firmware Multiplier

Prof. Alberto Borghese
Dipartimento di Informatica
borgnese@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 complessità dei circuiti (numero di porte logiche). Inoltre è rigida («hard») e non si può adattare a implementare funzioni diverse.

La soluzione HW viene utilizzata per le operazioni frequenti (e.g. nei componenti per elaborazione di segnali veloci: radar, sonar...) 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):

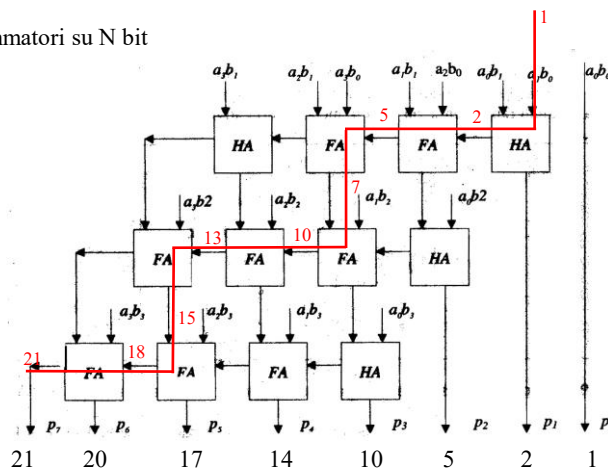
- **Approccio hardware mediante porte logiche.** E' un approccio esaustivo (tabellare). Per ogni funzione, per ogni ingresso viene memorizzata l'uscita. E' utilizzabile per funzioni molto particolari (ad esempio di una variabile). Non molto utilizzato.
- **Approccio hardware programmabile** (e.g. PLA, 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



Abbiamo tanti sommatore su N bit



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.

Moltiplicando

1 1 0 1 1 X

Moltiplicatore

1 0 1 =

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 = 1).

Prodotto

1 0 0 0 0 1 1 1

27 x 5 = 135

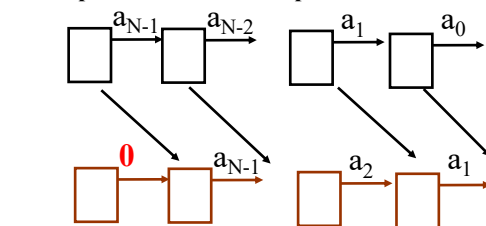


Shift (scalamento) e somma



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

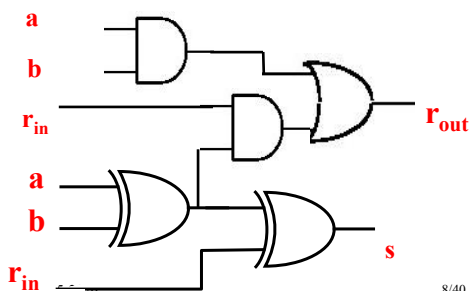
Esempio. shift dx 1 di una parola su N bit:



Il bit a_0 si "perde".

Il bit $a_{N-1} = 0$.

$a_k = a_{k+1}$ per $k=0,1,2,\dots,N-2$



Somma su 1 bit

Utilizzo del riporto in ingresso

Produzione del riporto in uscita



Moltiplicazione utilizzando somma e shift



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

1 1 0 1 1 x A
0 1 0 1 1 = B

A0) Genero il primo e il secondo prodotto parziale

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

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

1 0 1 0 0 0 1 + S_1 ParzSum
0 0 0 0 0 - - = $0 * 2^2$

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

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

1 0 1 0 0 0 1 + S_2 ParzSum
1 1 0 1 1 - - - = $A * 2^3$

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

1 0 0 1 0 1 0 0 1 + S_3 ParzSum
0 0 0 0 0 - - - - = $0 * 2^4$

27 x 11 = 297

27 + 54 + 0 + 216 = 297

1 0 0 1 0 1 0 0 1 P Prodotto
 $1x2^8 + 1x2^5 + 1x2^3 + 1x2^0 =$
 $256 + 32 + 8 + 1 = 297$

A.A. 2025-2026

9/40

<http://borghese.di.unimi.it/>

9



Moltiplicazione utilizzando somma e shift



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

1 1 0 1 1 x A
0 1 0 1 1 = B

Per ogni bit della parola (5 bit):

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

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

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

0 0 0 0 0 1 1 0 1 1 + $S_1 = P + A * 2^1$
1 1 0 1 1 - = $A * 2^1$

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

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

27 x 11 = 297

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

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

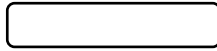
A.A. 2025-2026

10/40

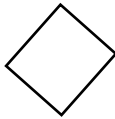
10



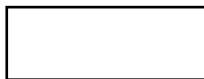
Diagrammi di flusso (flow chart)



Inizio / terminazione



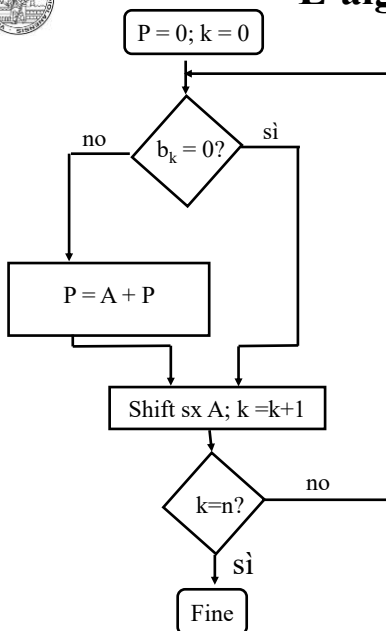
Test



Processo (esecuzione)



L'algoritmo



A $\longrightarrow$
B $\longrightarrow$

1 1 0 1 1 x
0 1 0 1 1 =

0 0 0 0 0 0 0 0 0 + Initial P=0
1 1 0 1 1 = A * 2⁰

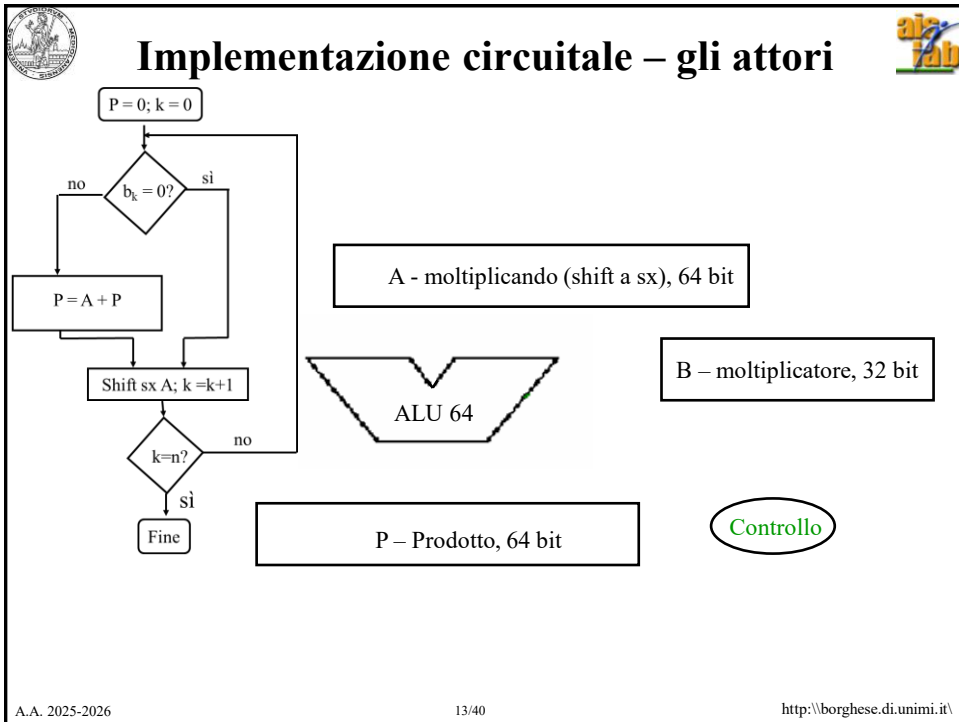
0 0 0 0 0 1 1 0 1 1 + S₁ = P + A
1 1 0 1 1 - = A * 2¹

0 0 0 1 0 1 0 0 0 1 + S₂ = S₁ + A
0 0 0 0 0 - - = 0 * 2²

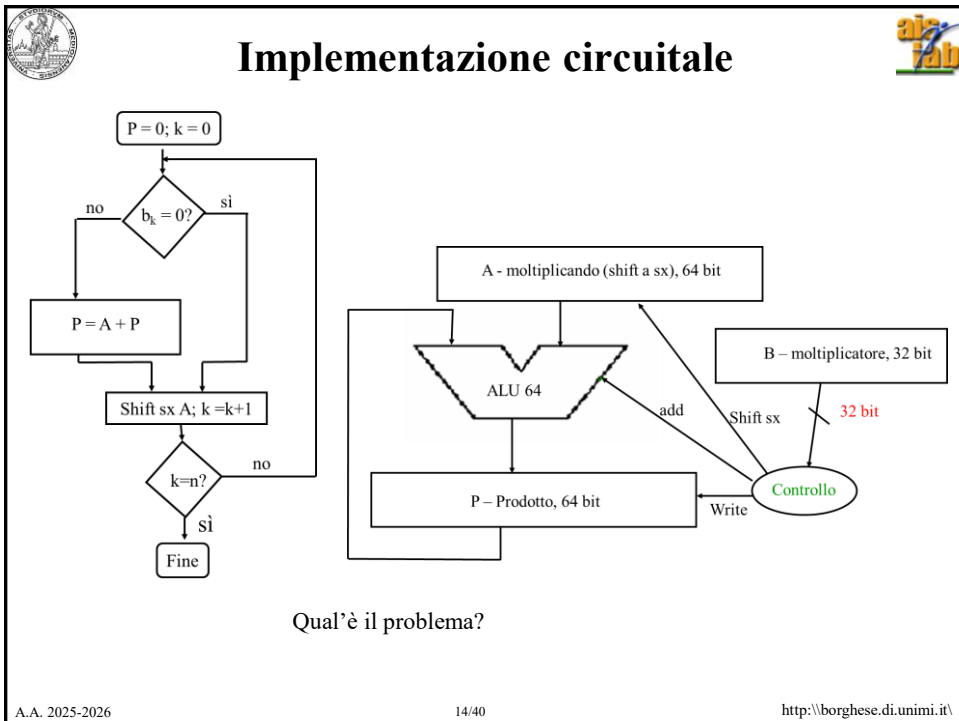
0 0 1 0 1 0 0 0 1 + S₃ = S₂ + 0
1 1 0 1 1 - - - = A * 2³

1 0 0 1 0 1 0 0 1 + S₄ = S₃ + A
0 0 0 0 0 - - - - = 0 * 2⁴

1 0 0 1 0 1 0 0 1 Final P = S₄ + 0



13



14



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 10₁₀ x

B → 1 0 1 1 = 11₁₀ =

00000000+ P

1010 = A⁰

00001010+ S₁

1010 - A¹

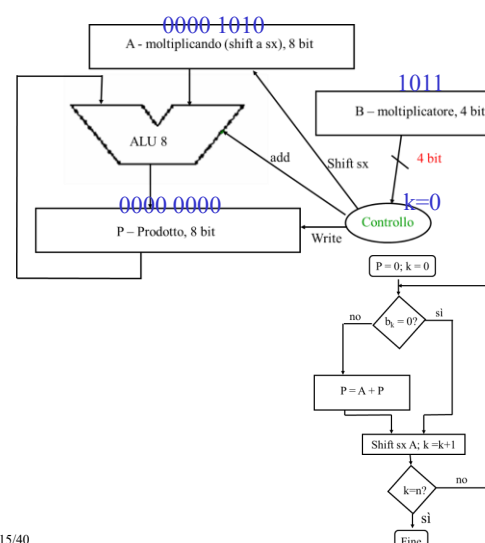
00011110+ S₂

0000 - - A²

00011110+ S₃

1010 - - - A³

P → 0 1 1 0 1 1 1 0 110₁₀



A.A. 2025-2026
15/40

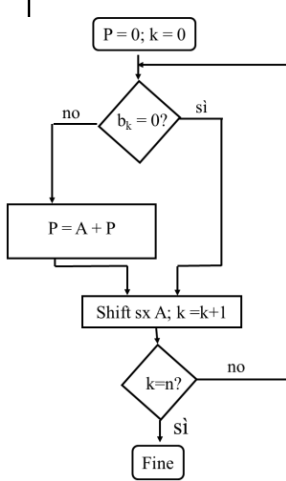
15

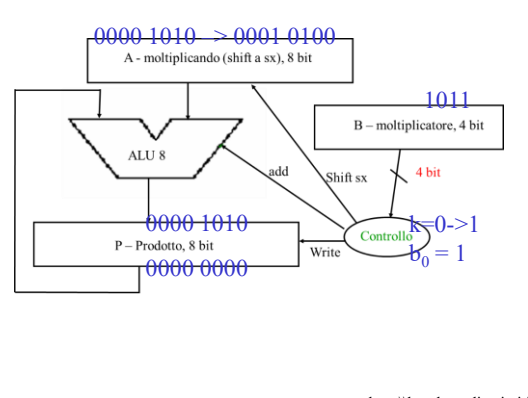


Esempio – passo 1



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





A.A. 2025-2026
16/40
<http://borghese.di.unimi.it/>

16

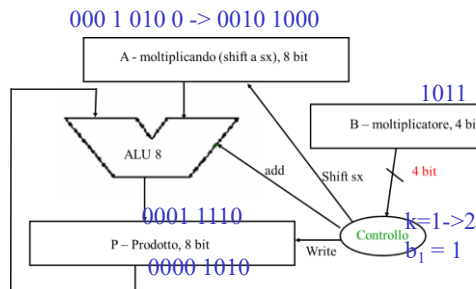
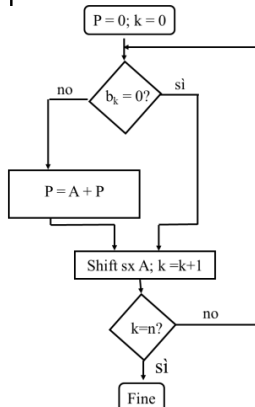


Esempio – passo 2



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

$$10 \cdot 2 + 10 = 30$$

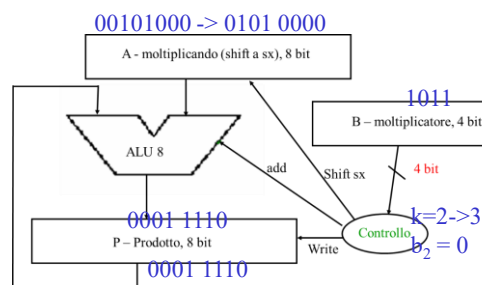
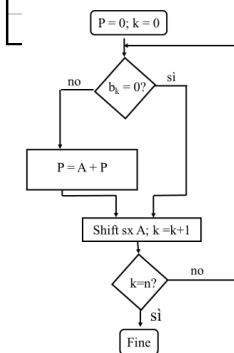


Esempio – passo 3



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

$$0 + 30 = 30$$



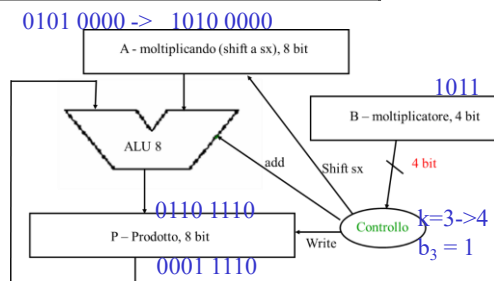
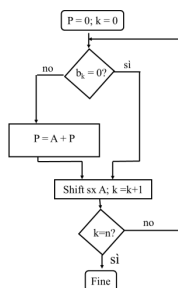


Esempio – passo 4



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

$$10 \cdot 8 + 30 = 110$$



A.A. 2025-2026

19/40

<http://borghese.di.unimi.it/>

19



Esempio – riassunto



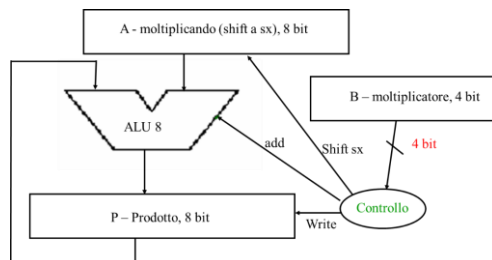
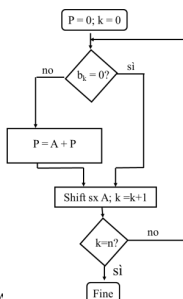
Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	b ₀ =1->P=P+A	1011	0000 1010	0000 1010
	Moltiplicando << 1	1011	0001 0100	0000 1010
2	b ₁ =1->P=P+A	1011	0001 0100	0001 1110
	Moltiplicando << 1	1011	0010 1000	0001 1110
3	b ₂ =0->Nulla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

$$=10_{10}$$

$$=30_{10}$$

$$=30_{10}$$

$$=110_{10}$$



A.A.

20/40

<http://borghese.di.unimi.it/>

20



Esercizi



Costruire il circuito HW che esegui la moltiplicazione 7×9 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



Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	$b_0=1 \rightarrow P=P+A$	1011	0000 1010	0000 1010
	Moltiplicando $\ll 1$	1011	0001 0100	0000 1010
2	$b_1=1 \rightarrow P=P+A$	1011	0001 0100	0001 1110
	Moltiplicando $\ll 1$	1011	0010 1000	0001 1110
3	$b_2=0 \rightarrow$ Nulla	1011	0010 1000	0001 1110
	Moltiplicando $\ll 1$	1011	0101 0000	0001 1110
4	$b_3=1 \rightarrow P=P+A$	1011	0101 0000	0110 1110
	Moltiplicando $\ll 1$	1011	1010 0000	0110 1110



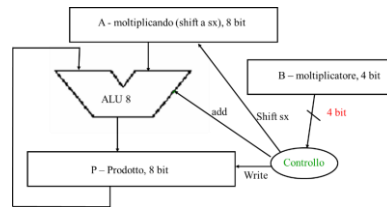
Ottimizzazione

- 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.

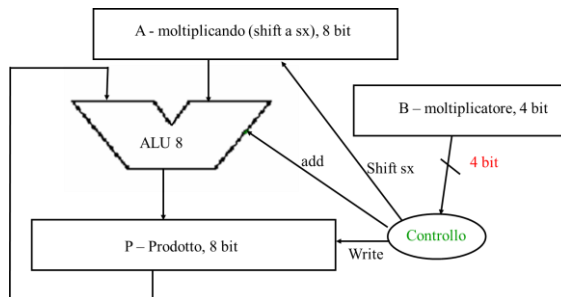
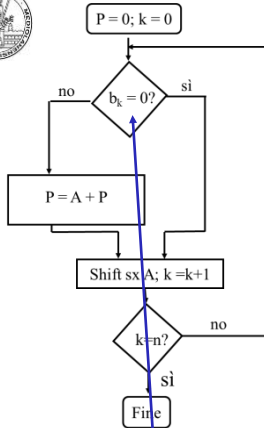
Situazione del tipo: **produttore-consumatore**.

Produco dei dati: la parola B.

Consumo i dati: 1 bit della parola B a ogni iterazione (consumo = 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.

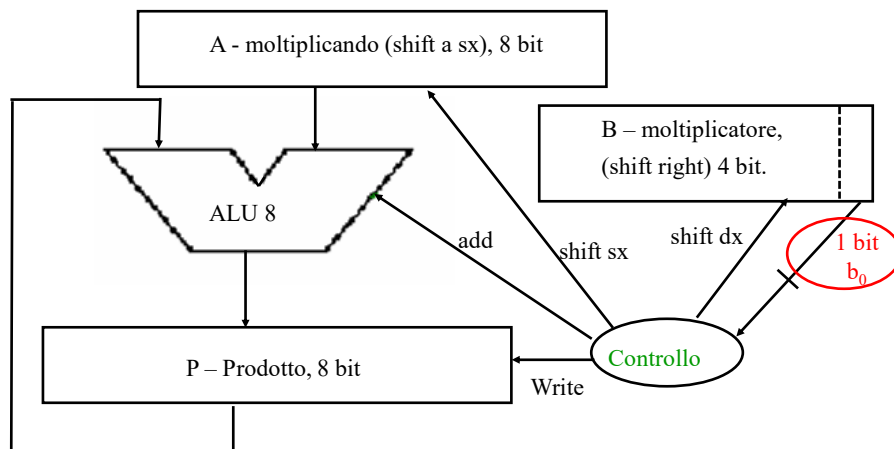
Posso ottenere lo stesso effetto in altro modo:

- 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



A.A. 2025-2026

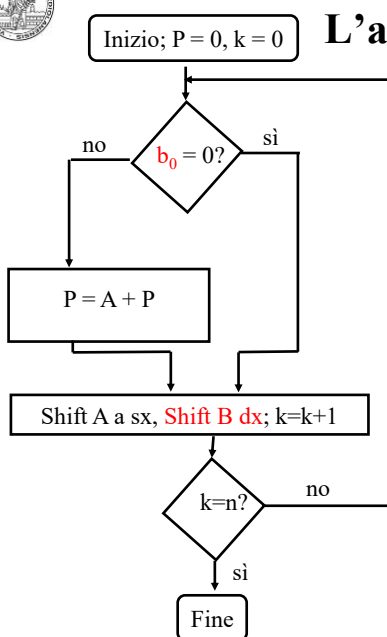
25/40

<http://borghese.di.unimi.it/>

25



L'algoritmo - I



$$\begin{array}{rcl} A & \longrightarrow & 1010x \\ B & \longrightarrow & 1011= \\ & & \hline & & 00000000+ \quad P \\ A = A * 2^0 & & 1010= \quad A^0 \\ & & \hline & & 00001010+ \quad P_1 \\ A^1 = A * 2^1 & & 1010- \quad A^1 \\ & & \hline & & 00011110+ \quad P_2 \\ A^2 = A * 2^2 & & 0000- - \quad A^2 \\ & & \hline & & 00011110+ \quad P_3 \\ A^3 = A * 2^2 & & 1010- - - \quad A^3 \\ & & \hline P & \longrightarrow & 01101110 \end{array}$$

A.A. 2025-2026

26/40

<http://borghese.di.unimi.it/>

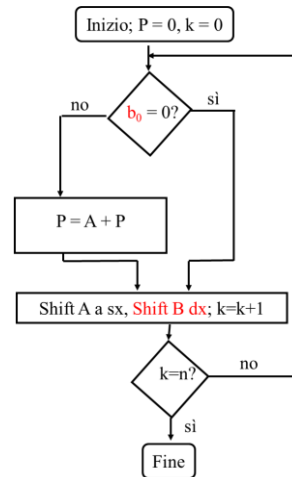
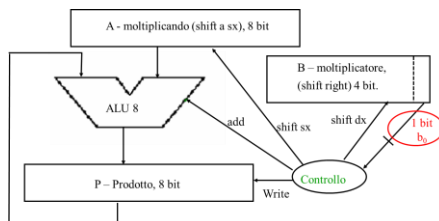
26



Esecuzione - I



Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	$b_0=1 \rightarrow P=P+A$	1011	0000 1010	0000 1010
	Moltiplicando $\ll 1$	1011	000 1010 0	0000 1010
	Moltiplicatore $\gg 1$	0 101 1	000 1010 0	0000 1010
2	$b_0=1 \rightarrow P=P+A$	0 101	000 1010 0	0001 1110
	Moltiplicando $\ll 1$	0 101	00 1010 00	0001 1110
	Moltiplicatore $\gg 1$	00 10	00 1010 00	0001 1110
3	$b_0=0 \rightarrow$ Nulla	00 10 11	00 1010 00	0001 1110
	Moltiplicando $\ll 1$	00 10	0 1010 000	0001 1110
	Moltiplicatore $\gg 1$	000 1	0 1010 000	0001 1110
4	$b_0=1 \rightarrow P=P+A$	000 1 011	0 1010 000	0110 1110
	Moltiplicando $\ll 1$	0000	1010 0000	0110 1110
	Moltiplicatore $\gg 1$	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	$b_0=1 \rightarrow P=P+A$	1011	0000 1010	0000 1010
	Moltiplicando $\ll 1$	1011	000 1010 0	0000 1010
	Moltiplicatore $\gg 1$	0 101	000 1010 0	0000 1010
2	$b_0=1 \rightarrow P=P+A$	0 101	000 1010 0	0001 1110
	Moltiplicando $\ll 1$	0 101	00 1010 00	0001 1110
	Moltiplicatore $\gg 1$	00 10	00 1010 00	0001 1110
3	$b_0=0 \rightarrow$ Nulla	00 10	00 1010 00	0001 1110
	Moltiplicando $\ll 1$	00 10	0 1010 000	0001 1110
	Moltiplicatore $\gg 1$	000 1	0 1010 000	0001 1110
4	$b_0=1 \rightarrow P=P+A$	000 1	0 1010 000	0110 1110
	Moltiplicando $\ll 1$	0000	1010 0000	0110 1110
	Moltiplicatore $\gg 1$	0000	1010 0000	0110 1110



Analisi dello shift



A scorre rispetto al prodotto P che resta fermo

A - moltiplicando (shift a sx, 8 bit)

B - moltiplicatore, (shift right) 4 bit.

ALU 8

P - Prodotto, 8 bit

Controllo

add

shift sx

shift dx

1 bit b_0

Write

```

graph TD
    Start([Inizio; P = 0, k = 0]) --> Decision1{b0 = 0?}
    Decision1 -- no --> Add[P = A + P]
    Decision1 -- si --> Add
    Add --> Shift[Shift A a sx, Shift B dx; k=k+1]
    Shift --> Decision2{k=n?}
    Decision2 -- no --> Decision1
    Decision2 -- si --> Fine([Fine])
        
```

P scorre rispetto ad A che resta fermo

Iterazione (k)	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	0000 1010	0000 0000
1	$b_0=1 \rightarrow P=P+A$	1011	0000 1010	0000 1010
	Moltiplicando << 1	1011	000 1010 0	0000 1010
	Moltiplicatore >> 1	0 101	000 1010 0	0000 1010
2	$b_0=1 \rightarrow P=P+A$	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	$b_0=0 \rightarrow$ 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	$b_0=1 \rightarrow P=P+A$	000 1	0 1010 000	0110 1110
	Moltiplicando << 1	0000	1010 0000	0110 1110
	Moltiplicatore >> 1	0000	1010 0000	0110 1110

A.A. 2025-2026 29/40 http://borghese.di.unimi.it/

29



Razionale per una seconda implementazione



Ad ogni iterazione sommo N cifre (pari al numero di cifre del moltiplicando), incolonnate opportunamente rispetto a P

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	$b_0=1 \rightarrow P=P+A$	1011	0000 1010	0000 1010
	Moltiplicando << 1	1011	000 1010 0	0000 1010
	Moltiplicatore >> 1	0 101	000 1010 0	0000 1010
2	$b_0=1 \rightarrow P=P+A$	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	$b_0=0 \rightarrow$ 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	$b_0=1 \rightarrow P=P+A$	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 = 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

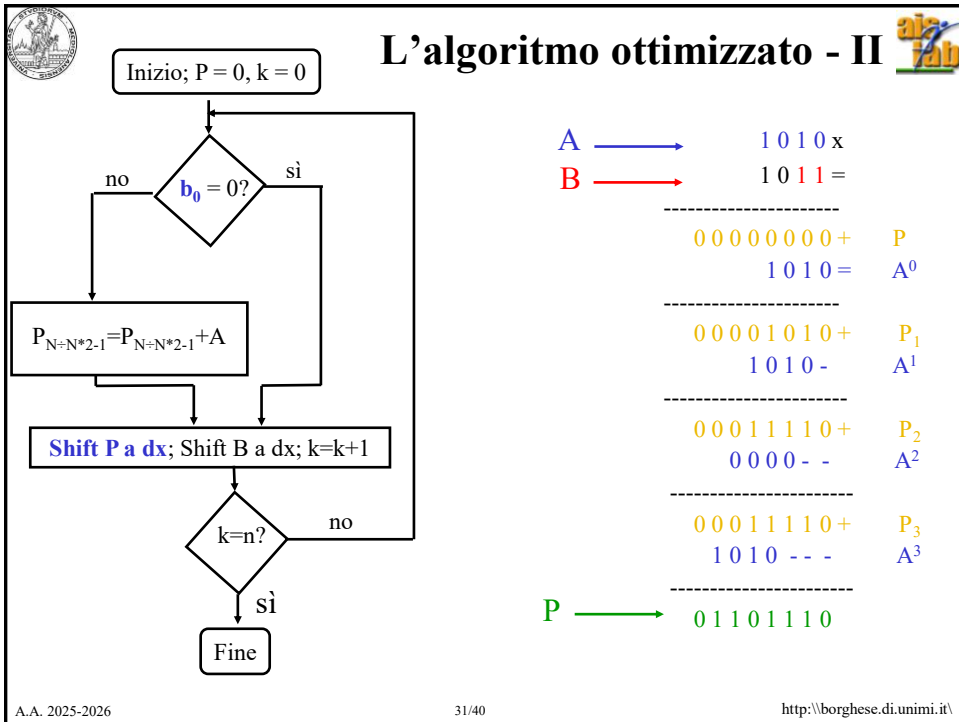
0 0 0 1 1 1 1 0 + P₃

1 0 1 0 - - - A³

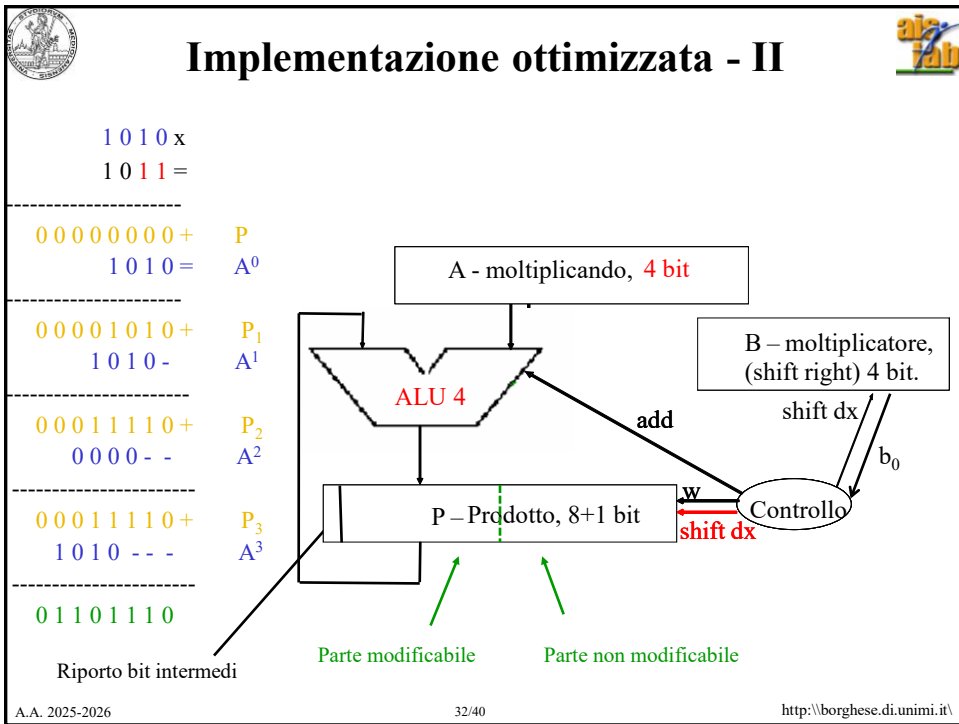
0 1 1 0 1 1 1 0

A.A. 2025-2026 30/40 http://borghese.di.unimi.it/

30



31



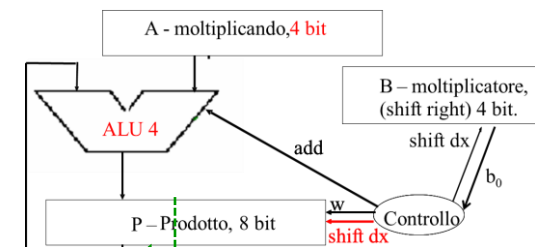
32



Esecuzione - II



Iterazione	Passo	Moltiplicatore (B)	Moltiplicando (A)	Prodotto (P)
0	Valori iniziali	1011	1010	0000 0000
1	b ₀ =1->P=P+A	1011	1010	1010 0000
	Prodotto >> 1	1011	1010	0 1010 000
	Moltiplicatore >> 1	0 101	1010	0 1010 000
2	b ₀ =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	b ₀ =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	b ₀ =1->P=P+A	000 1	1010	1 101110 0
	Prodotto >> 1	0000	1010	0110 1110
	Moltiplicatore >> 1	0000	1010	0111 1110



A.A. 2025-2026

33/40

<http://borghese.di.unimi.it/>

33



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	b ₀ =1->P=P+A	1011	1010	1010 0000
	Prodotto >> 1	1011	1010	0 1010 000
	Moltiplicatore >> 1	0 101	1010	0 1010 000
2	b ₀ =1->P=P+A	0 101	1010	1 1110 000
	Prodotto >> 1	0 101	1010	0 11110 00
	Moltiplicatore >> 1	00 10	1010	0 11110 00
3	b ₀ =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	b ₀ =1->P=P+A	000 1	1010	1 101110 0
	Prodotto >> 1	0000	1010	0110 1110
	Moltiplicatore >> 1	0000	1010	0111 1110

A.A. 2025-2026

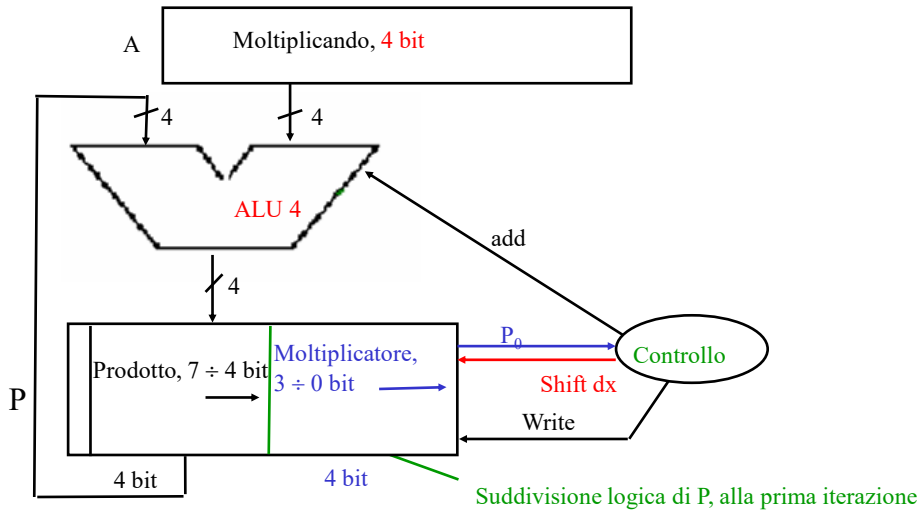
34/40

<http://borghese.di.unimi.it/>

34



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.

A.A. 2025-2026

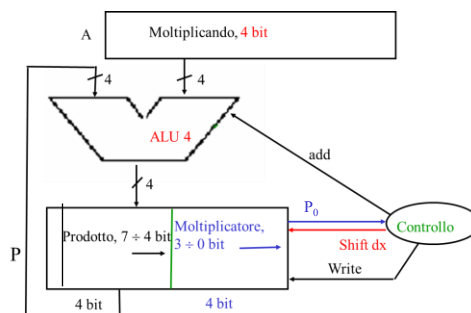
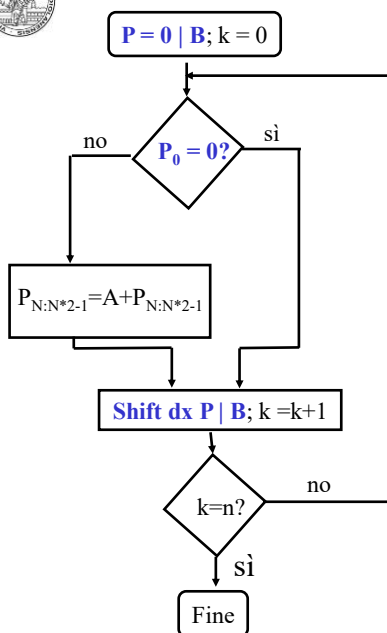
35/40

<http://borghese.di.unimi.it/>

35



Algoritmo ottimizzato

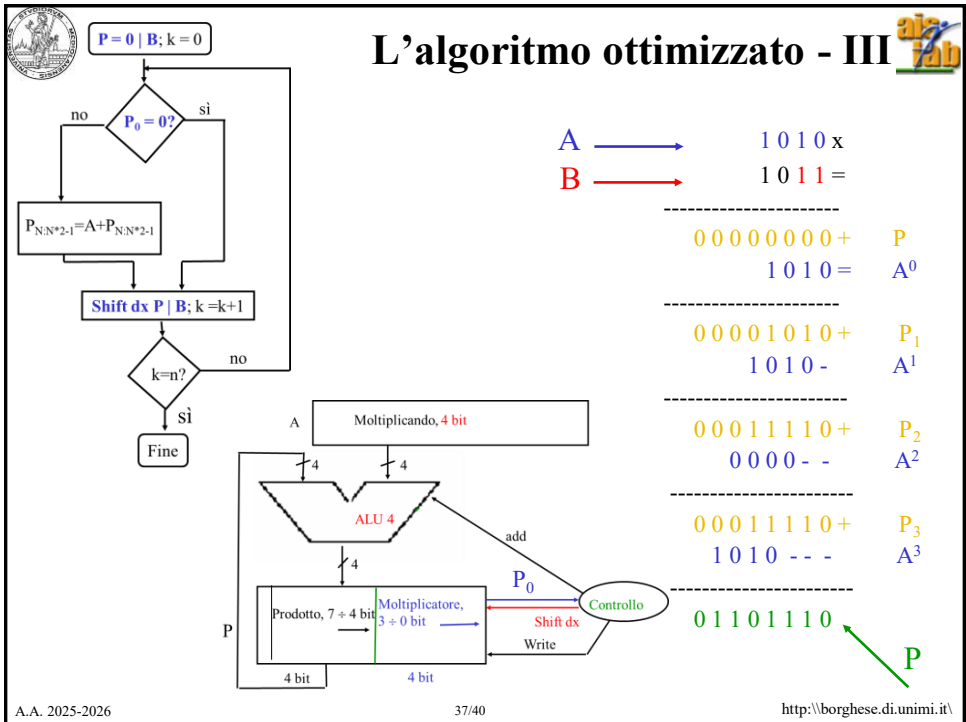


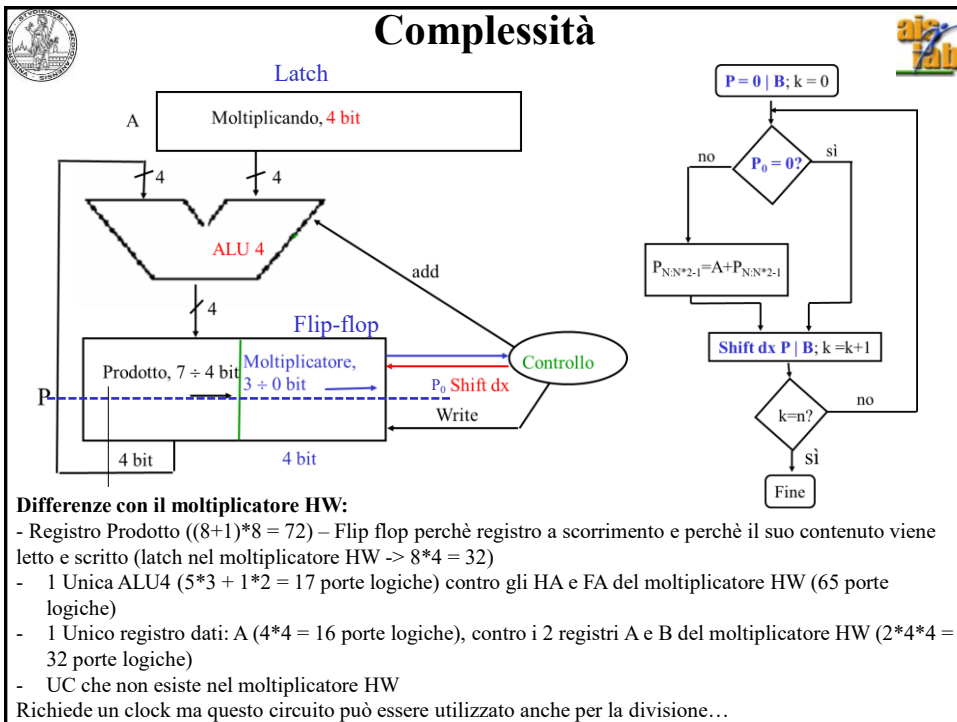
A.A. 2025-2026

36/40

<http://borghese.di.unimi.it/>

36





39



40