


Firmware Multiplier

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

Università degli Studi di Milano

Riferimenti sul Patterson: C.6 & 3.4


Sommario

Il 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

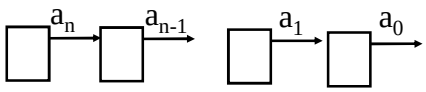
Shift (scalamento)

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

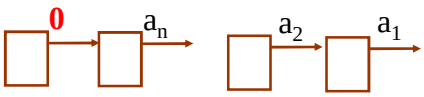
Effettuato al di fuori delle operazioni selezionate dal Mux della ALU, da un circuito denominato *Barrel shifter*.

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.

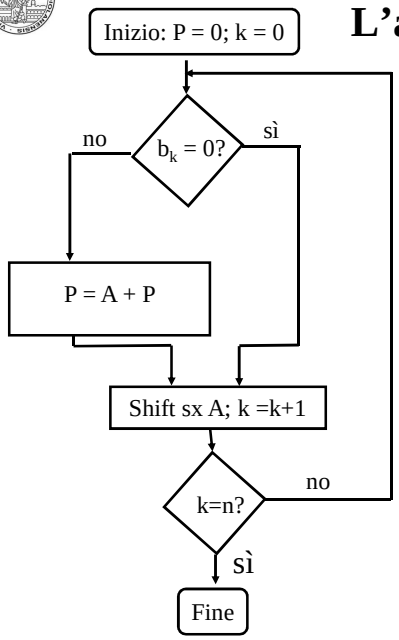
Itero per ogni bit del moltiplicatore:

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

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

Implementazione circuitale – gli attori

A - moltiplicando (shift a sx), 64 bit

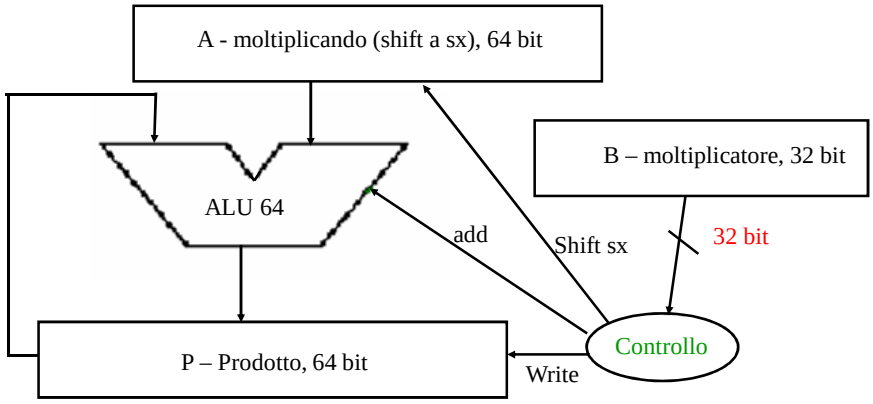
ALU 64

B – moltiplicatore, 32 bit

P – Prodotto, 64 bit

Controllo

Implementazione circuitale



Qual'è il problema?

Esercizi

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

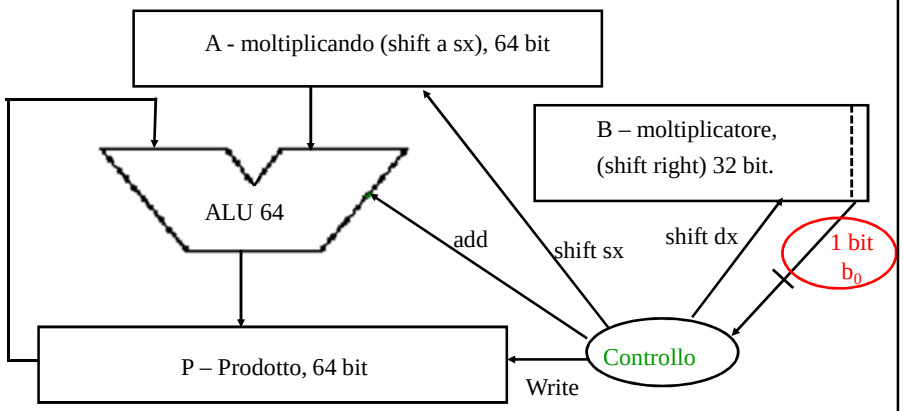
Eseguire la stessa moltiplicazione secondo l'algoritmo visto, indicando passo per passo il contenuto dei 3 componenti: A che contiene il moltiplicando, B che contiene il moltiplicatore e P che contiene somme parziali ed il risultato finale.

Sommario

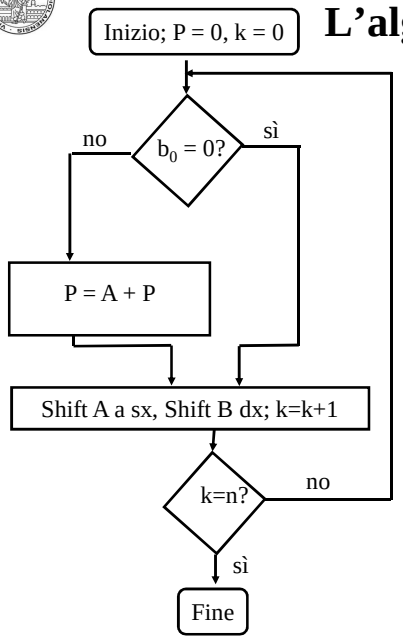
I moltiplicatori firmware

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 **0010**
2	1a: 1 ⇒ Prod = Prod + Mcando	0001	0000 **0100**	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

0010 x
0011 =

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.

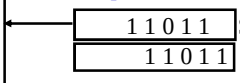
Ad ogni iterazione sommo N cifre (pari al numero di cifre del moltiplicando).

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

```
      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
```

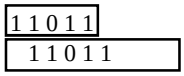

Implementazione ottimizzata - II

1ª implementazione

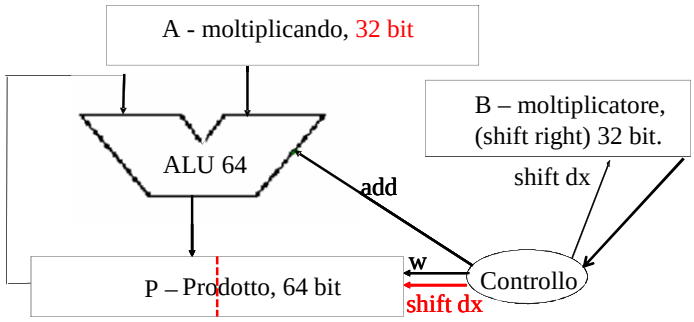


Sposto a sx il moltiplicando | Sposto a dx il prodotto
P^1 primo prodotto parziale

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?

Implementazione ottimizzata - II

1ª implementazione

2ª implementazione

1 1 0 1 1

1 1 0 1 1

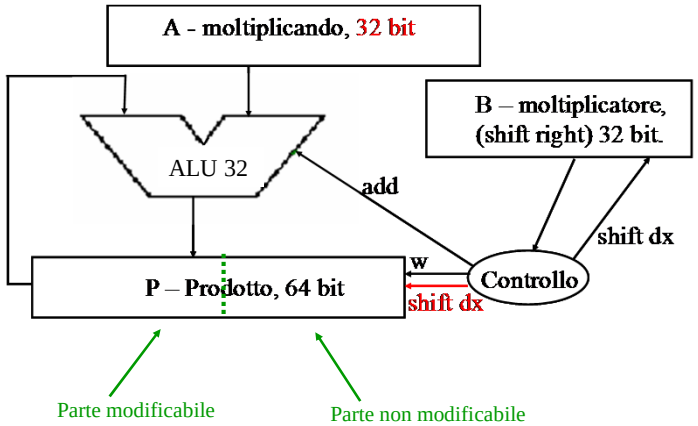
Sposto a sx il moltiplicando | Sposto a dx il prodotto

P[1] primo prodotto parziale

1 1 0 1 1

1 1 0 1 1

```
  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
```



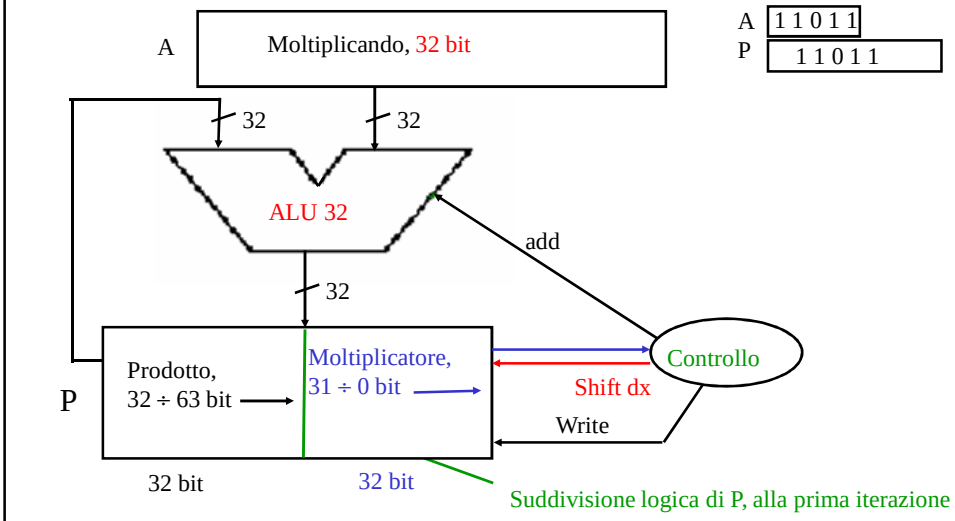
Razionale dell'implementazione - III

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.

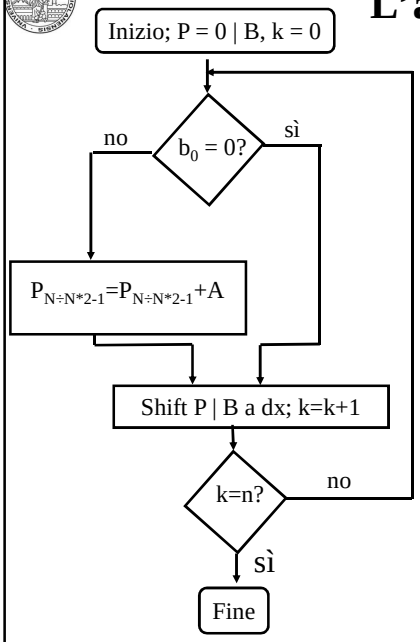
```
  1 1 0 1 1 x
      1 1 1 =
----------------------
      0 0 0 0 0 +
      1 1 0 1 1
----------------------
    1 1 1 1 1
      1 1 0 1 1 +
    1 1 0 1 1 -
----------------------
  1
    1 0 1 0 0 0 1 +
    1 1 0 1 1 - -
 ----------------------
-
  1 0  1 1 1 1 0 1
```

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



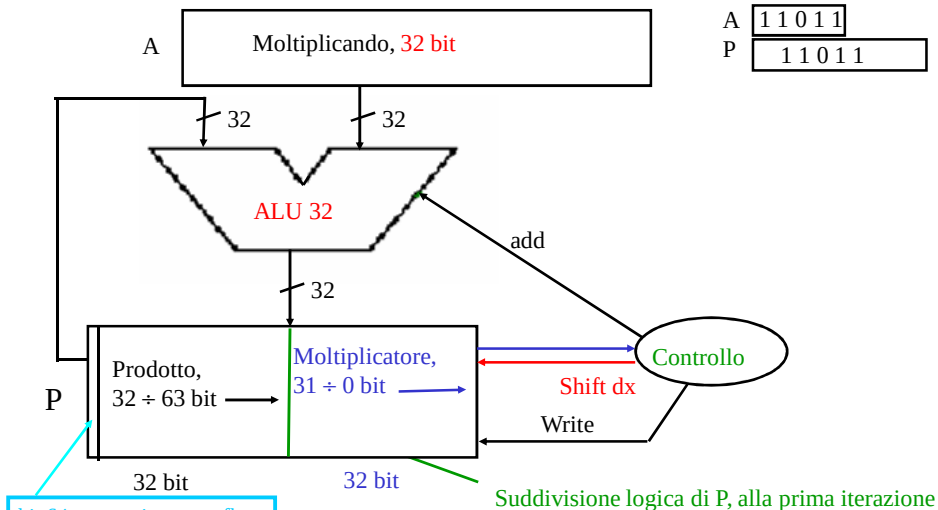

```
A ──►    1 1 0 1 1 x
B ──►      1 1 1 =
      ----------------------
            0 0 0 0 0 +
              1 1 0 1 1
      ----------------------
          1 1 1 1 1
            1 1 0 1 1 +
          1 1 0 1 1 -
      ----------------------
          1
        1 0 1 0 0 0 1 +
        1 1 0 1 1 - -
       ----------------------
       -
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 0010
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.

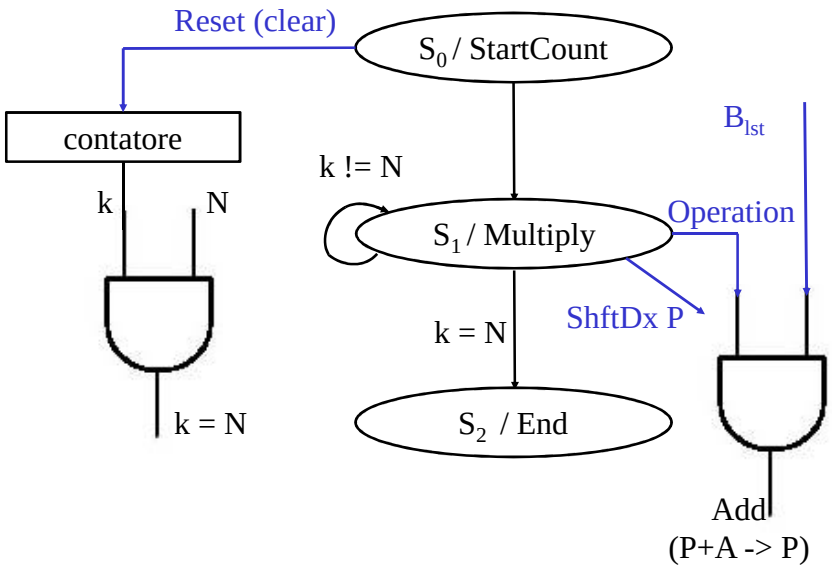
Circuito finale – moltiplicatore firmware



bit 64 – per evitare overflow nelle somme parziali
No overflow finale

Le somme parziali possono occupare un bit in più per effetto del riporto.

Unità di controllo



Sommario

I moltiplicatori firmware

Ottimizzazione dei moltiplicatori firmware