Lez. 31/10/2014
(ven., 2h)

Moltiplicazione Intera

Con il metodo della moltiplicazione "a mano", si sfruttano le caratteristiche della notazione posizionale. In particolare, si costruiscono n numeri di $2n$ cifre, che vengono poi sommati tra loro per ottenere il risultato (n è il numero di cifre utilizzate per gli operandi).(*)

(*) per semplicità consideriamo operandi con lo stesso numero di cifre. Se il moltiplicando ha m cifre e il moltiplicatore n, si dovranno sommare n numeri da $2m$ cifre.

Esempio:

$$A = (98)_{10} = (1100010)_2$$
$$B = (27)_{10} = (0011011)_2$$

$n=2$ nel caso decimale

$n=7$ nel caso binario

98 x
27 =
---
(0)686
196– (0)

2646 = $C = A \times B$ =

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

Note.

1) Qualsiasi sia la base del sistema di rappresentazione, con $2n$ cifre per il risultato non c'è overflow. Infatti

$$C_{max} = A_{max} \times B_{max} = (\beta^n - 1) \times (\beta^n - 1) < \beta^{2n} - 1$$

2) Nel caso decimale, bisogna conoscere il prodotto tra tutte le possibili coppie di cifre — le cosiddette "tabelline". Nel caso binario invece, ciascuno degli n addendi è 0 se la cifra del moltiplicatore (B) ad esso relativa è 0, mentre è una versione traslata del moltiplicando se tale cifra è 1.

In sostanza, il risultato della moltiplicazione binaria si può ottenere considerando soltanto le posizioni in cui il moltiplicatore ha cifra 1. Se queste posizioni sono $h \leq n$, allora dovremo sommare solo h numeri a $2n$ cifre. Tali numeri sono versioni traslate verso sinistra del moltiplicando. L'entità della traslazione corrisponde alla posizione dove il moltiplicatore ha cifra 1.
Nel caso dell'esempio, il moltiplicatore ha un 1 in posizione 0, 1, 3 e 4. Per ottenere il prodotto si dovranno quindi sommare 4 numeri, ed esattamente

- il moltiplicando traslato a sin. di 0 posizioni = A
- " 1 posizione = 2 x A
- " 3 posizioni = 8 x A
- " 4 " = 16 x A

Infatti $A + 2xA + 8xA + 16xA = (1+2+8+16)xA =$

$= 27 \times A = C$

In generale, possiamo scrivere

$$C = A \times B = A \times (2^{n-1} \cdot b_{n-1} + 2^{n-2} \cdot b_{n-2} + \dots + 2^3 \cdot b_3 + 2^2 \cdot b_2 + 2^1 \cdot b_1 + 2^0 \cdot b_0) =$$

$$= (2^{n-1} \times A) b_{n-1} + (2^{n-2} \times A) b_{n-2} + \dots + (2^3 \times A) b_3 + (2^2 \times A) b_2 + (2^1 \times A) b_1 + (2^0 \times A) b_0 =$$

$$= A_{n-1} b_{n-1} + A_{n-2} b_{n-2} + \dots + A_3 b_3 + A_2 b_2 + A_1 b_1 + A_0 b_0 =$$

$$= \sum_{i=0}^{n-1} A_i b_i \quad , \text{ con}$$

$A_i = 2^i \times A$ = A raddoppiato i volte

b_i = i-sima cifra di B = (*)
= resto della divisione di B per 2^i, ottenibile dimezzando B i volte, ovvero traslando B a destra i volte e guardando il valore di LSB ottenuto.

(*) $B_i \doteq B \% 2^i$

Sempre con i numeri dell'esempio, traslamo B ripetutamente:

b_4 b_3 b_1 b_0

$B_0 = 0\,0\,1\,1\,0\,1\,1 = 27 = B\%1$

$B_1 = 0\,0\,0\,1\,1\,0\,1 = 13 = B\%2$

$B_2 = 0\,0\,0\,0\,1\,1\,0 = 6 = B\%4$

$B_3 = 0\,0\,0\,0\,0\,1\,1 = 3 = B\%8$

$B_4 = 0\,0\,0\,0\,0\,0\,1 = 1 = B\%16$

$B_5 = 0\,0\,0\,0\,0\,0\,0 = 0 = B\%32$

$B_{n-1} = B_6 = 0\,0\,0\,0\,0\,0\,0 = 0 = B\%64 = B\%2^{n-1}$

Notiamo ancora una volta come l'LSB della versione "sliftata" a destra di i posizioni fornisca il valore di b_i.

Sulla proprietà del sistema binario gli Egizi avevano basato il loro algoritmo di moltiplicazione, che si serviva appunto di raddoppi (del moltiplicando) e dimezzamenti (del moltiplicatore):

colonna dei dimezzamenti (interi)	colonna dei raddoppi
27 = B	A = 98 +
13	196 +
~~6 (pari)~~	~~392~~
3	784 +
1	1568 =
	2646

Si considerano solo le righe dove i numeri dimezzati sono DISPARI, e si sommano i numeri raddoppiati corrispondenti.

Notiamo che questo algoritmo esegue solo 4 dimezzamenti, fermandosi quando $B_i = 1$.
Il numero max di dimezzamenti richiesti è $n - 1 = 6$.

Osservazione. La moltiplicazione a mano (o in colonna) in base 10 e la versione in base 10 della moltiplicazione egizia di pagina precedente (questi ultimo metodo è tuttora impiegato in alcune zone rurali dell'Africa per eseguire moltiplicazioni con l'abaco) usano entrambi il sistema posizionale: il primo quello in base 10, il secondo quello in base 2. In entrambi i casi, il numero di somme da effettuare non supera n, ovvero il numero di cifre per esprimere i fattori (e in particolare il moltiplicatore) nella base prefissata (2 nel caso base 10, 7 nel caso base 2). Quindi la complessità degli algoritmi che usano il sistema posizionale non dipende dalla grandezza degli operandi, ma dal numero di cifre usate per rappresentarli (*). Naturalmente, numeri più grandi hanno bisogno di più cifre, ma prefissato il numero di cifre da usare, il tempo di calcolo non dipende dal valore degli operandi (o meglio, il particolare valore del moltiplicatore può essere sfruttato per abbreviare il procedimento, che comunque richiede un tempo massimo dipendente dal numero di cifre del moltiplicatore, non dal massimo valore del moltiplicatore).

(*) ovvia non dipende da B_1 ma da log$_B$(B_1) con B base del sistema di num.

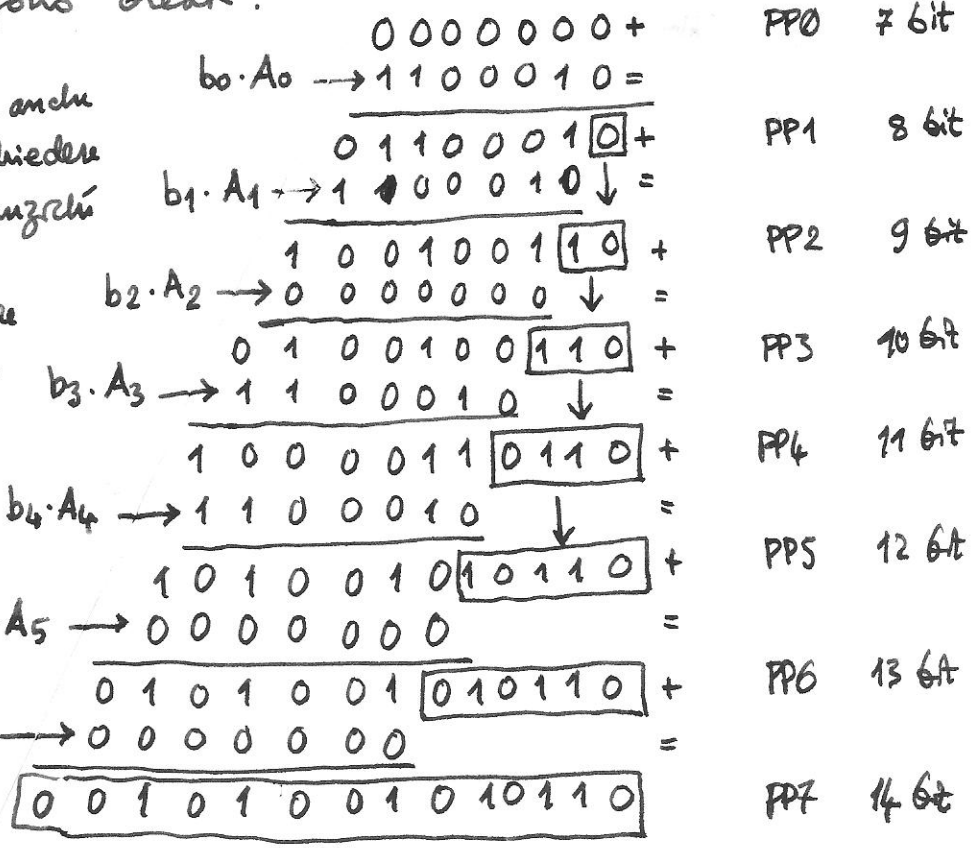
Invece, nel caso di moltiplicazione come somma ripetuta:

$$C = A \times B = \underbrace{A + A + A \ldots + A + A}_{B \text{ volte}}$$

(cfr. la "Pascalina" del 1642)

il numero di somme dipende dal valore di B, e in particolare il calcolo di AXB richiede un tempo diverso che per BXA.

Un difetto degli algoritmi visti (molt. in colonna, molt. egizia) è di costruire n addendi e sommarli tutti insieme. Ciò può creare problemi di riporto nelle somme, qualora n sia grande (ogni la somma delle cifre omologhe in una colonna può generare riporti su varie colonne successive), e comunque richiede la memorizzazione degli addendi stessi, ciò che rende poco pratica l'implementazione con una macchina sequenziale o con un programma. Una versione più pratica dell'algoritmo della molt. in colonna fa uso di un registro che accumula i vari addendi, ogni volta che questi vengono creati.



Questo algoritmo ha anche il vantaggio di richiedere somme di n cifre, anziché a 2n cifre.

Il registro accumulatore è inizializzato a 0, di n cifre e contiene il PP0 (prodotto parziale 0).

Ogni nuovo PP calcolato ha una cifra in più del precedente.

L'ultimo PP ha dunque 2n cifre, e coincide col risultato.

Delle $n+i$ cifre del PP_i, le i cifre meno significative sono definitive, cioè non cambiano più fino al risultato finale. Esse devono essere via via memorizzate. Al termine dell'ultimo stadio di somma, la somma a n cifre produce le $n+1$ cifre finali del prodotto.

somma ↓ carry ↳

Resta da individuare il modo per produrre gli n bit più significativi del numero $b_i \cdot A_i$, da sommare al registro accumulatore al passo i dell'algoritmo. Abbiamo:

$$b_i A_i = b_i (2^i \times A) = (b_i \times A) \times 2^i = \cancel{B_i}$$

$$= b_i \cdot (a_{n-1}\, a_{n-2} \cdots a_2\, a_1\, a_0)_2 \times 2^i =$$

$$= b_i \cdot (\underbrace{a_{n-1}\, a_{n-2} \cdots a_2\, a_1\, a_0}_{n \text{ bit}}\ \underbrace{0\ 0 \ldots 0}_{i \text{ bit}})_2$$

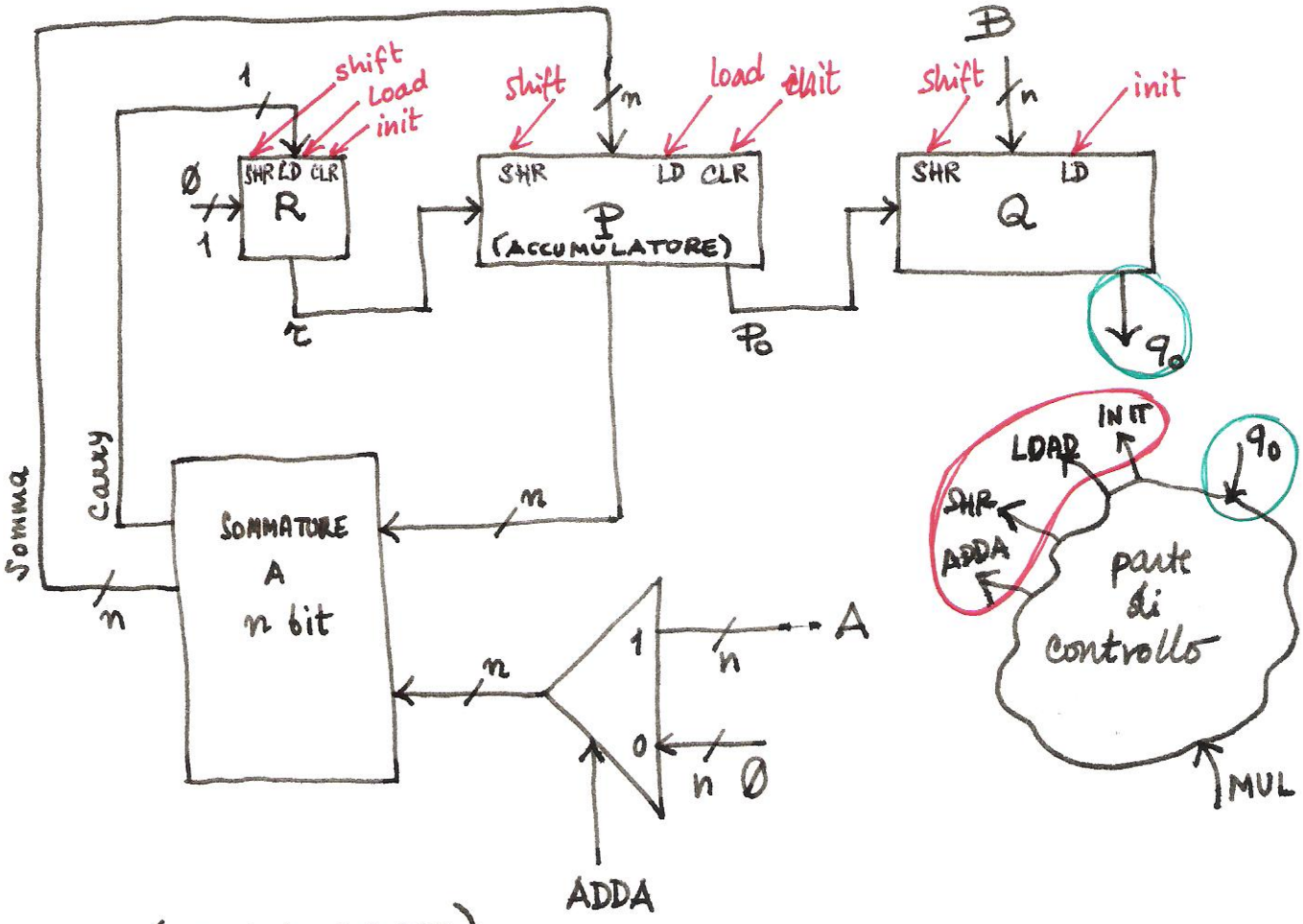
Segue dunque che gli n bit da sommare all'accumulatore al passo i sono

$$\begin{cases} a_{n-1}\, a_{n-2} \cdots a_2\, a_1\, a_0 & \text{se } b_i = 1 \\ 0\ 0 \ldots 0\ 0\ 0 & \text{se } b_i = 0 \end{cases}$$

ossia bisogna sommare al contenuto dell'accumulatore il contenuto di A (moltiplicando) oppure 0; la decisione va presa in base al valore di b_i, che è ottenibile — come già visto — esaminando il contenuto del bit b_0 di un registro a scorrimento inizializzato con B che a ogni passo "sfitta" i suoi bit a destra di una posizione:

b_i = LSB (registro B dopo i scorrimenti a destra).

Lo schema realizzativo della parte operativa della macchina può essere dunque il seguente:



(-1. WAIT FOR MUL)

0. $R \leftarrow 0$; $P \leftarrow 0$; $Q \leftarrow B$; (INIT)

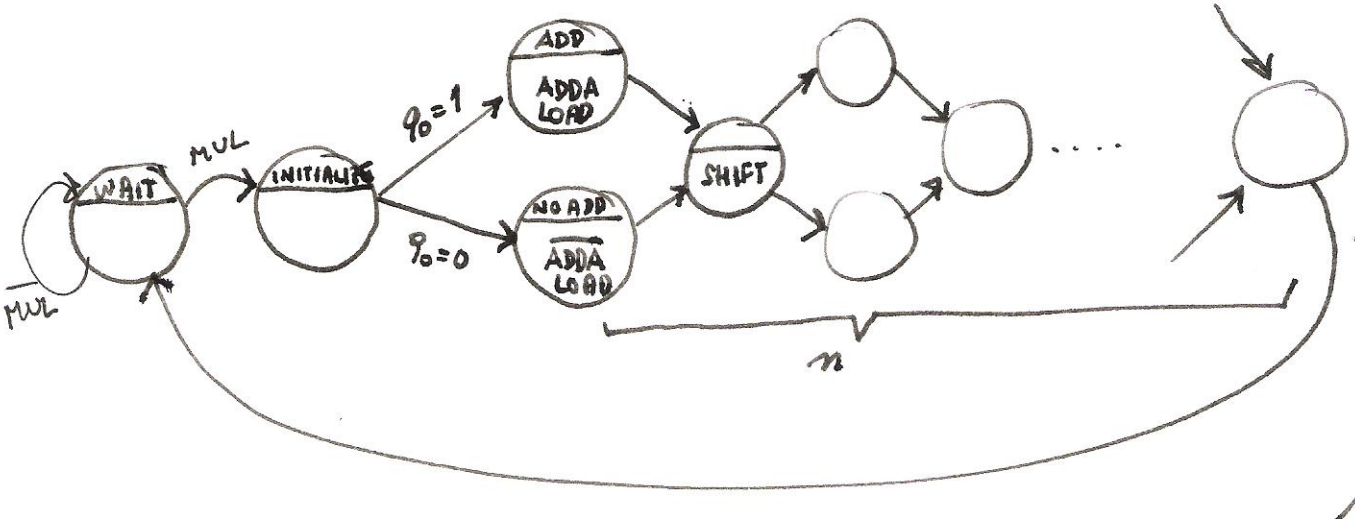
FOR $i = 1 \dots n$: {

1. IF $q_0 = 0$ THEN ADDA = 0, ELSE ADDA = 1; LOAD

2. SHIFT }

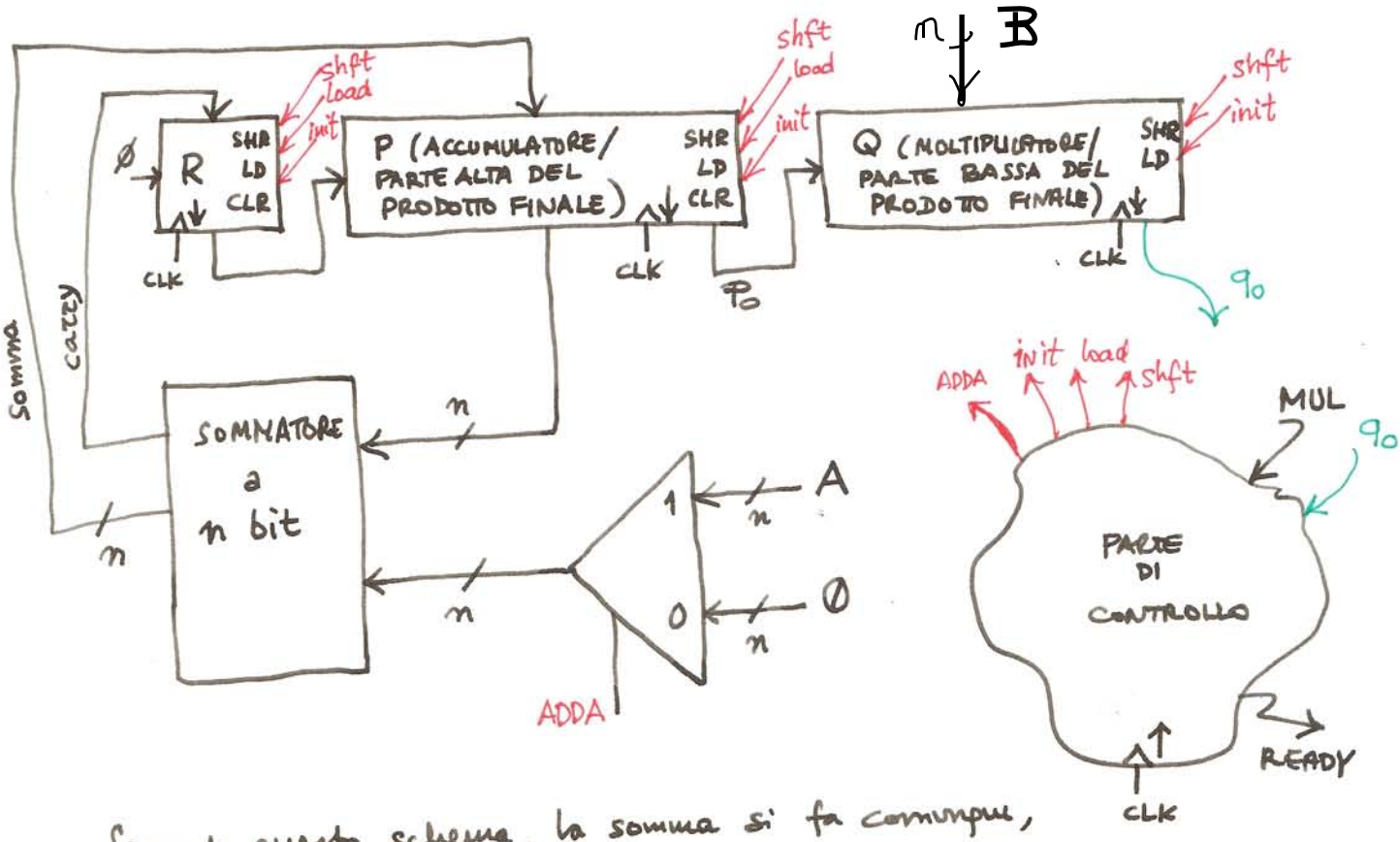
problema del race-around...

si risolve facendo commutare l'accumulatore sui fronti di discesa e la parte di controllo su quelli di salita

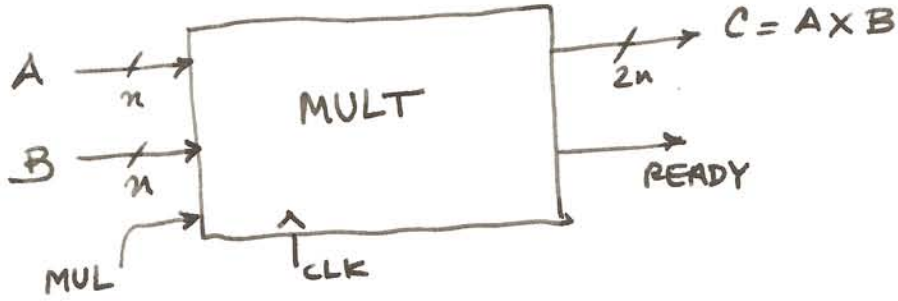


lez. 3 nov. 2016
(lun., 2h)
[vedi foto per l'ultima parte]

Riprendiamo lo schema della PO del moltiplicatore:

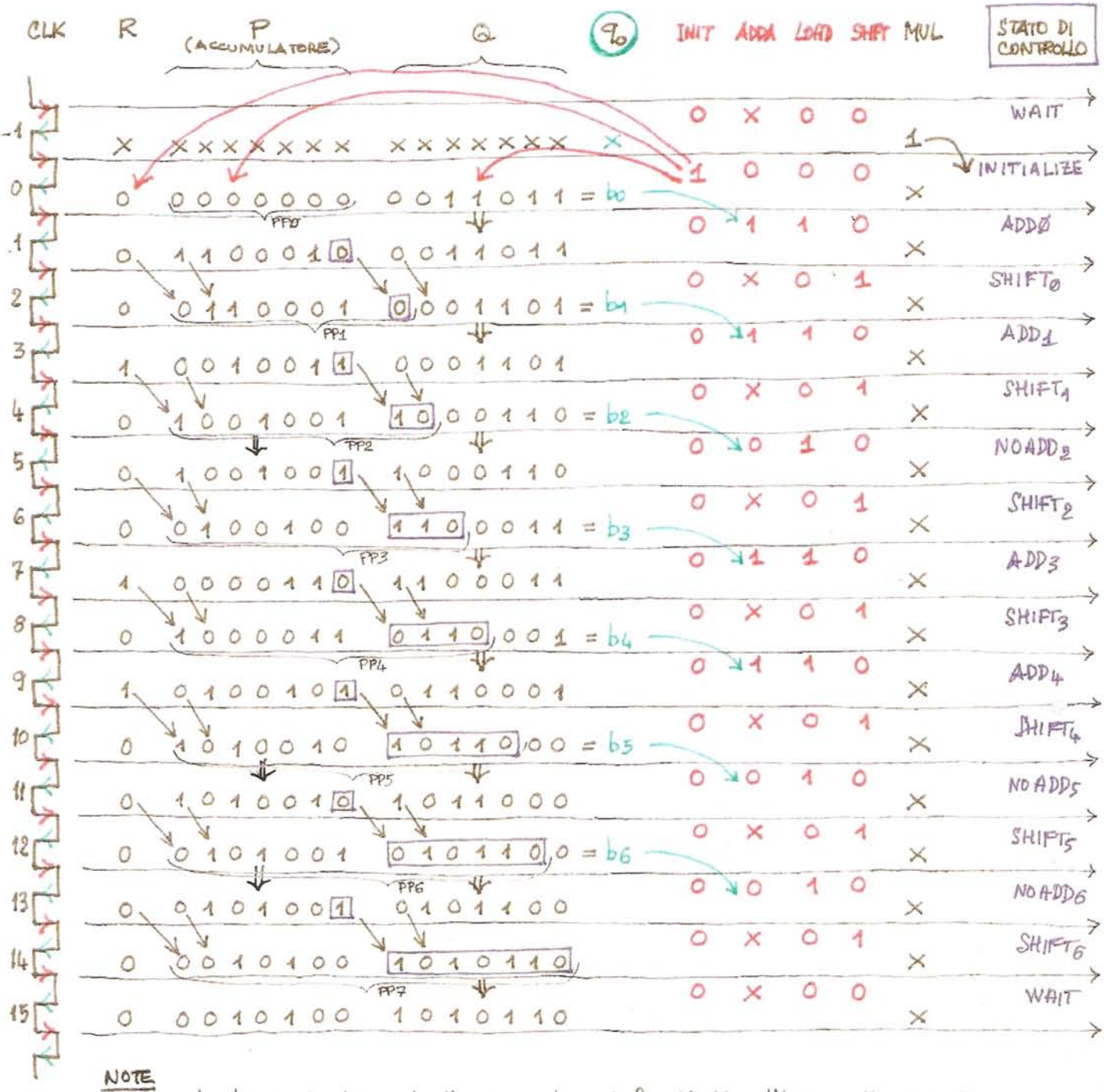


Secondo questo schema, la somma si fa comunque, o con A (se $q_0 = 1$) o con Ø (se $q_0 = 0$).
Il registro Q contiene inizialmente il moltiplicatore (B), e ad ogni shift right riceve un bit del prodotto finale, mentre espelle il bit del moltiplicatore appena usato per decidere se sommare A o Ø. Al termine delle operazioni, Q contiene la parte bassa del prodotto finale, mentre P contiene la parte alta del prodotto finale.
[vedere lo schema di funzionamento allegato]. All'esterno, la macchina si presenta così:



MOLTIPLICAZIONE INTERA — FUNZIONAMENTO: DIAGRAMMA TEMPORALE

$C = A \times B$; $A = (1100010)_2$, $B = (0011011)_2$

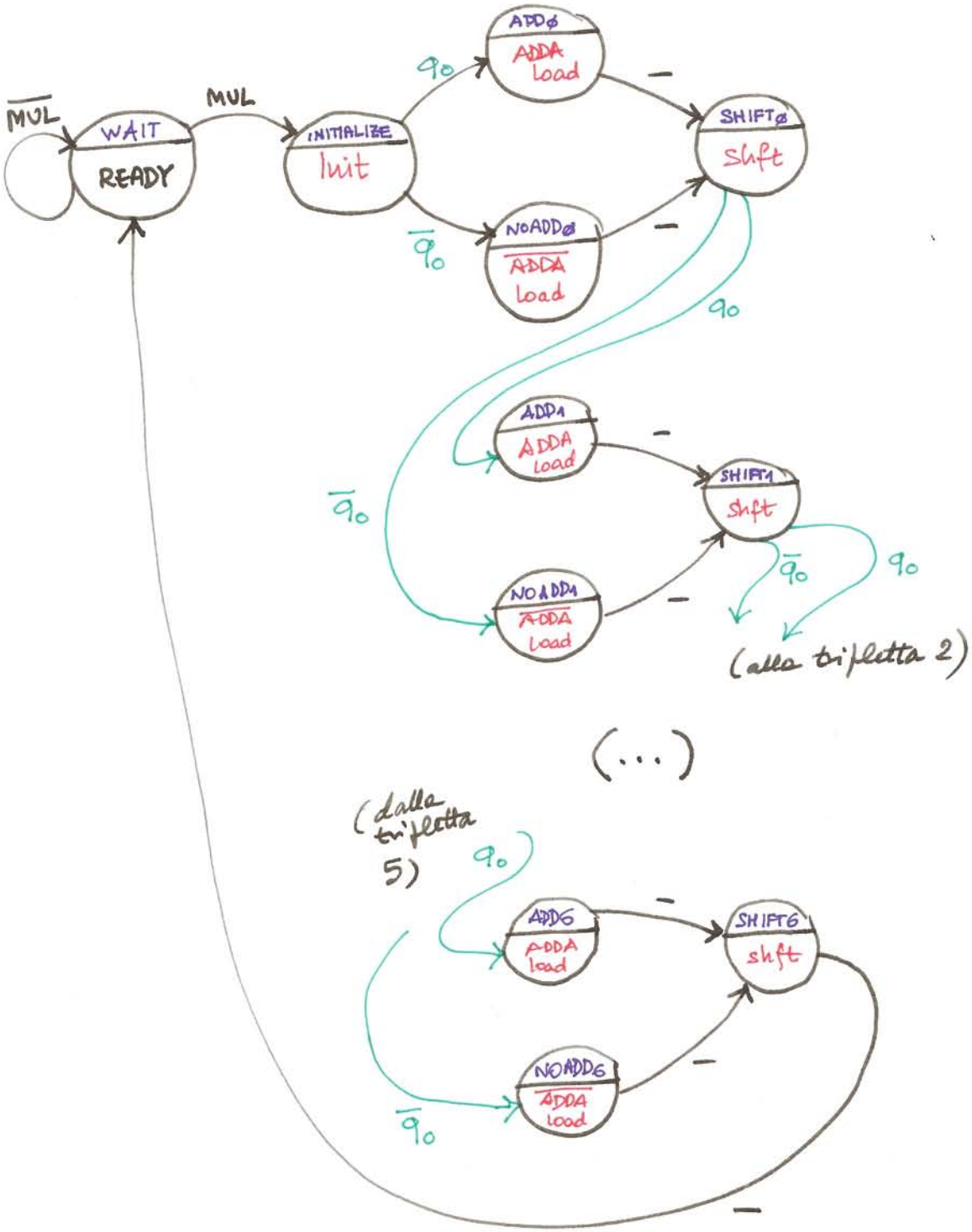


NOTE

1. La parte di controllo commuta sui fronti di salita, quella operativa sui fronti di discesa. Questo evita problemi di "race-around" (ad es. quando ADDA e LOAD sono entrambi asseriti).
2. Ad ogni somma viene prodotta una nuova cifra del prodotto finale; questa è fatta passare nel registro Q allo shift successivo. Ogni prodotto parziale è costituito dai bit di P più quelli di Q che fanno parte del prodotto finale: dunque il PP_i ha $n+i$ bit.
3. Il bit q_0 viene campionato dal controllo ogni 2 colpi di clock a partire da $t=1$. Si ha $q_0(t) = b_{(t-1)/2}$. L'algoritmo si conclude dopo $2n$ passi.

$2bis=4$. All'ultimo passo di somma, tutti gli $n+1$ bit sono finali. (somma, carry)
Si ha inoltre $PP7 = C$ (prodotto finale)

L'automa di controllo è il seguente:



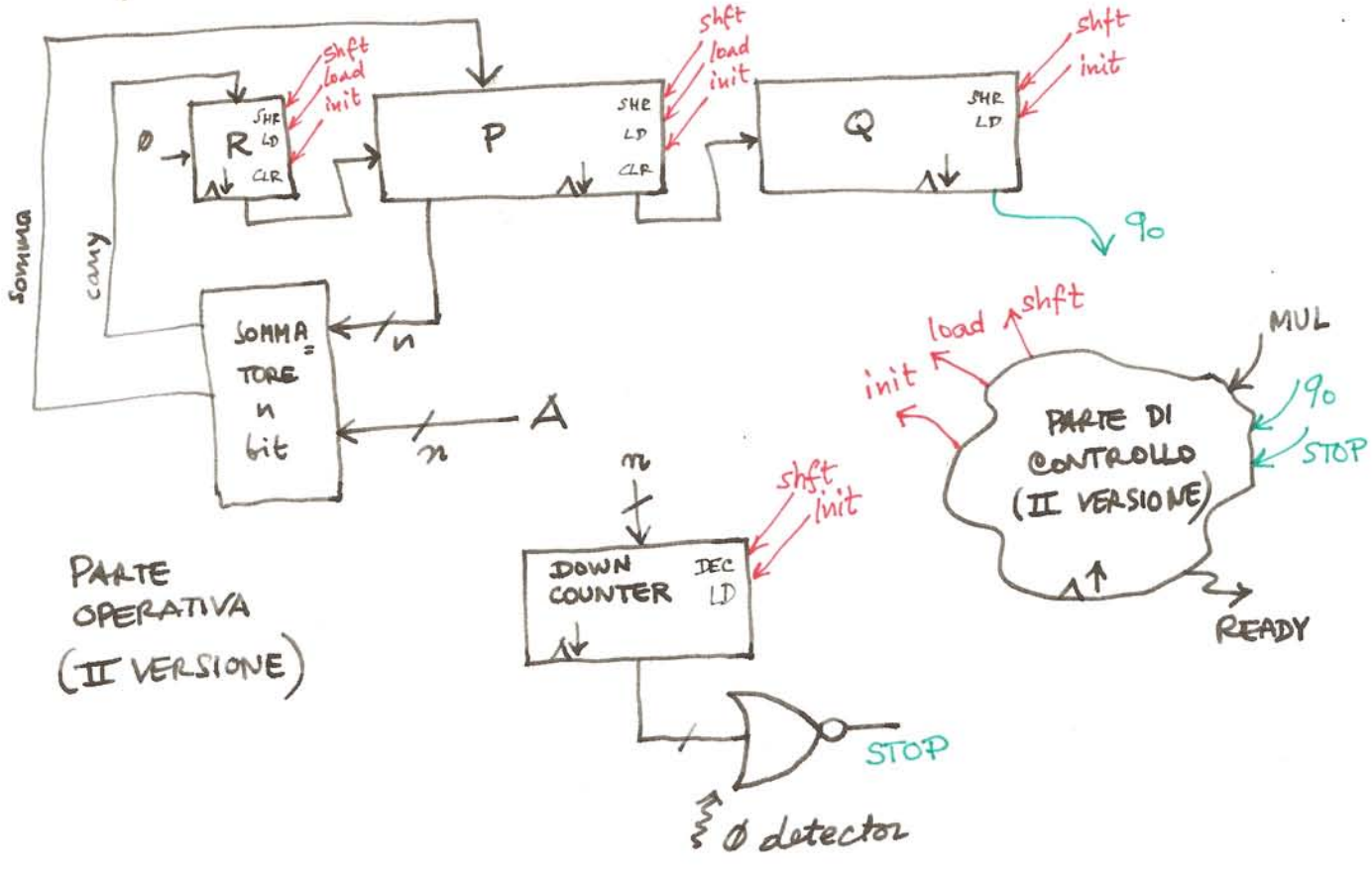
L'automa è costituito dallo stato d'attesa (WAIT), quello di inizializzazione (INITIALIZE), e da n (=7, in questo caso) triplette di stati (ADD_i, $NOADD_i$, $SHIFT_i$), $i = 0, \dots 6$. Al termine dell'ultimo stato di shift (qui $SHIFT_6$) l'automa torna allo stato di attesa, poiché il prodotto è stato correttamente calcolato.

Vediamo quindi che la parte di controllo progettata secondo lo schema di parte operativa di pag. [1] deve "conoscere" n, ossia il numero di bit usato per rappresentare gli operandi, al fine di "sapere" quando fermarsi. Secondo questo progetto, il controllo richiede dunque

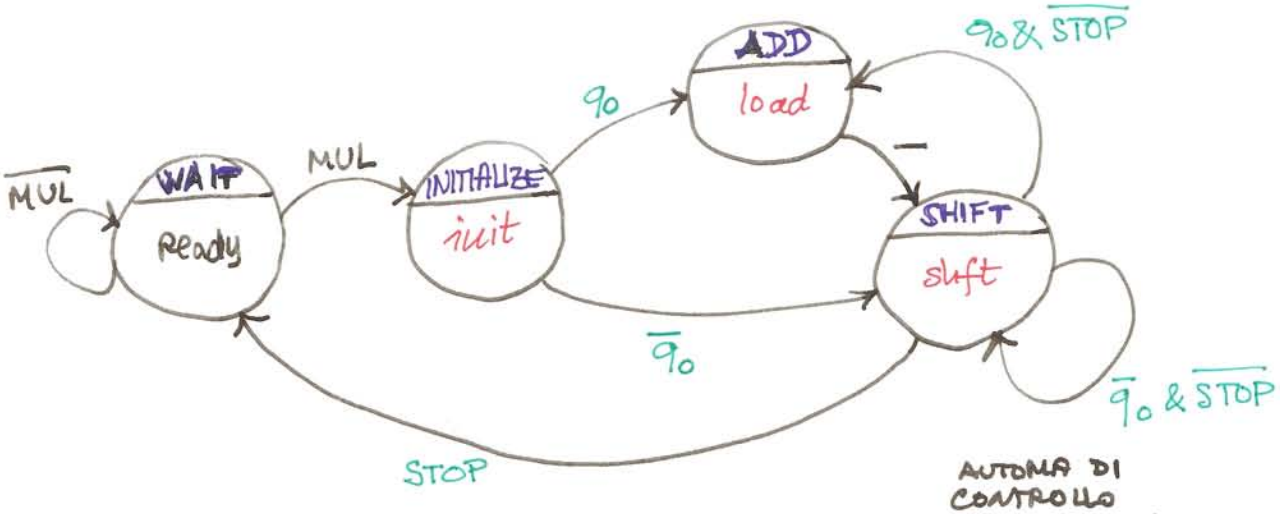
$$2 + 3 \cdot n \quad \text{stati}$$

- 2 → wait, initialize
- $3 \cdot n$ → addi, noaddi, shifti

Una versione certamente più semplice del controllo si può ottenere <u>relegando la conoscenza del numero n nella parte operativa</u>, e lasciando che sia quest'ultima ad "avvisare" (con un segnale di condizione aggiuntivo: STOP) il controllo quando n passi di calcolo siano stati (in particolare, di shift) compiuti:



Nella II versione della macchina è stato semplificato anche lo stadio di somma. Ora la somma viene fatta solo se $b_i = 1$, mentre se $b_i = 0$ si passa direttamente allo stato di shift i-simo. In particolare: il sommatore esegue comunque la somma P + A, ma questa viene effettivamente accumulata in P (attraverso il controllo load), solo se $b_i = 1$. Ciò rende superfluo il multiplexer all'ingresso del sommatore, nonché il controllo ADDA. Inoltre, viene semplificato l'automa di controllo, poiché la sezione di calcolo, prima formata dai tre stati ADD/NOADD/SHIFT, ora è formata da soli due stati (ADD / SHIFT):



AUTOMA DI CONTROLLO (II VERSIONE)

Notiamo come la nuova versione del controllo sia più flessibile della precedente. Infatti, essa può servire a controllare parti operative con un arbitrario numero di bit n per gli operandi.

Per concludere il progetto, non resta da realizzare la parte di controllo. Si può usare allo scopo la procedura di sintesi "monoblocco". Partendo dall'automa già disegnato, questa richiede di

- codificare in binario gli stati e "piazzare" il registro di stato (qui servono 2 bit, Q_1, Q_0, poiché abbiamo 4 stati)
- scrivere le funzioni $f(IN, SP) = SF$ e $g(SP) = OUT$ partendo dalla tabella, e sintetizzarle.

Usiamo la seguente codifica per gli stati:

STATO PRESENTE	CODIFICA (Q_1Q_0)	ingressi campionati nello stato
WAIT	00	MUL
INITIALIZE	01	q_0
ADD	10	—
SHIFT	11	q_0, STOP

Nell'ultima colonna della tabella sono riportati gli ingressi campionati in ciascuno stato. Questa informazione è di estrema importanza ai fini del progetto, poiché consente di semplificare la procedura realizzativa della funzione $f(IN, SP)$. Infatti, anziché procedere con la scrittura della tabella

MUL	q_0	STOP	Q_1	Q_0	Q_1'	Q_0'
($2^5 = 32$ righe)						

si può effettuare una sintesi di f basata su MULTIPLEKER usando come ingressi di selezione i bit di stato:

$$\text{STATO FUTURO} = \text{WAIT} \,\&\, f_{\text{WAIT}}(\text{MUL}) + \text{INITIALIZE} \,\&\, f_{\text{INITIALIZE}}(q_0) +$$
$$+ \text{ADD} \,\&\, f_{\text{ADD}}(-) + \text{SHIFT} \,\&\, f_{\text{SHIFT}}(q_0, \text{STOP})$$

che possono scriversi anche:

$$\begin{bmatrix} Q_1' \\ Q_0' \end{bmatrix} = \bar{Q}_1\bar{Q}_0 f_{WAIT}(MUL) + \bar{Q}_1 Q_0 f_{INITIALIZE}(q_0) + Q_1\bar{Q}_0 f_{ADD}(-) + Q_1 Q_0 f_{SHIFT}(q_0, STOP)$$

progettiamo ciascuna delle 4 funzioni separatamente:

SP = WAIT $(Q_1=0, Q_0=0)$:

MUL	$f_{WAIT}(MUL)$	
0	0	(WAIT)
1	1	(INITIALIZE)
	↓	
	0 MUL	

↳ cfr. automa pag. [4]

SP = INITIALIZE $(Q_1=0, Q_0=1)$

q_0	$f_{INITIALIZE}$	
0	1	(SHIFT)
1	$\bar{q}_0$	(ADD)
	↓	
	1 $\bar{q}_0$	

SP = ADD $(Q_1=1, Q_0=0)$

$$f_{ADD}^{(-)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad (\equiv SHIFT)$$

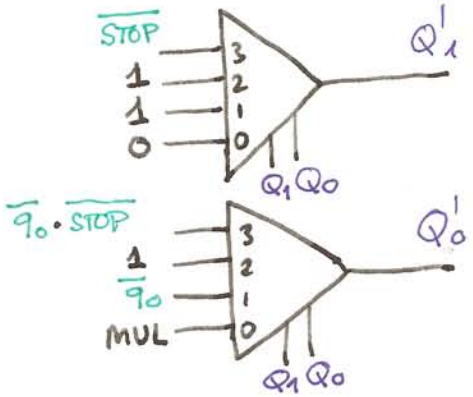
SP = SHIFT $(Q_1=1, Q_0=1)$

q_0	STOP	$f_{SHIFT}(q_0, STOP)$	
0	0	1	(SHIFT)
0	1	0	(WAIT)
1	0	0	(ADD)
1	1	0	(WAIT)
		↓ ↓	
		$\overline{STOP}$ $\bar{q}_0 \cdot \overline{STOP}$	

$$= \overline{q_0 + STOP}$$

Dunque

$$\begin{bmatrix} Q_1' \\ Q_0' \end{bmatrix} = \bar{Q}_1\bar{Q}_0 \begin{bmatrix} 0 \\ MUL \end{bmatrix} + \bar{Q}_1 Q_0 \begin{bmatrix} 1 \\ \bar{q}_0 \end{bmatrix} + Q_1\bar{Q}_0 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + Q_1 Q_0 \begin{bmatrix} \overline{STOP} \\ \bar{q}_0 \cdot \overline{STOP} \end{bmatrix}$$



Vediamo ora la funzione $g(SP)$ [cfr. slide precedente p. 4]:

Q_1 Q_0	Ready	init	load	shft
0 0	1	0	0	0
0 1	0	1	0	0
1 0	0	0	1	0
1 1	0	0	0	1

g è una funzione standard: il decoder $2 \to 4$!

Facciamo infine il disegno complessivo della parte di controllo: