8 *UNITA' ARITMETICHE*

8.1 Rappresentazione dei numeri in virgola fissa

In un sistema di numerazione a base *r*, un numero *N*, positivo, è rappresentato dal vettore:

$$N_r = (x_{n-1}, x_{n-2}, \ldots x_0, x_{-1}, \ldots x_{-m}) \tag{8.1.1}$$

dove x_i può assumere uno degli *r* valori:

$$x_i = 0, 1, \ldots (r-1) \tag{8.1.2}$$

Le prime *n* cifre, da x_0 a x_{n-1}, rappresentano la parte intera di N_r, mentre le rimanenti *m* quella frazionaria.

Il *valore* del numero, espresso in base 10, è dato da:

$$N_{10} = \sum_{i=-m}^{n-1} x_i r^i \tag{8.1.3}$$

Nelle rappresentazioni di numeri segnati, il carattere più a sinistra è, generalmente, riservata al segno.

Le rappresentazioni più comuni dei numeri con segno sono:

- *segno e valore assoluto*;
- *complemento alla base r diminuita*;
- *complemento alla base r*.

Nella prima rappresentazione, *segno e valore assoluto*, i numeri positivi, in base *r*, si scrivono come il vettore (8.1.1), mentre quelli negativi come:

$$-N_r = -(x_{n-1}, x_{n-2}, \ldots x_0, x_{-1}, \ldots x_{-m}) \tag{8.1.4}$$

Nella convenzione col segno "–", si opera sui valori assoluti dei numeri e si applica il segno al risultato secondo regole predefinite.

Nelle rappresentazioni seguenti, si opera aritmeticamente sul segno come se fosse una cifra appartenente al numero e ciò è più conveniente se si vogliono realizzare unità aritmetiche che operino, indifferentemente, su numeri positivi o negativi applicando le medesime regole.

Nella rappresentazione *complemento alla base r diminuita*, il numero positivo è ancora dato dalla (8.1.1), dove, però, la cifra più significativa, riservata al segno, vale sempre zero:

$$N_r = (0, x_{n-2}, \ldots x_0, x_{-1}, \ldots x_{-m}) \tag{8.1.5}$$

mentre il corrispondente numero opposto è espresso come:

$${}^{(r-1)}N_r = [(r-1), \bar{x}_{n-2}, \ldots \bar{x}_0, \bar{x}_{-1}, \ldots \bar{x}_{-m}] \tag{8.1.6}$$

dove

$$\bar{x}_i + x_i = (r-1) \tag{8.1.7}$$

e si definisce $\bar{x}_i$ come il *complemento della* x_i *alla base r diminuita.*

In questa notazione risulta:

$$N_r + {}^{(r-1)}N_r = [(r-1)_{n-1}, \ldots (r-1)_{-m}] \tag{8.1.8}$$

$$N_r + {}^{(r-1)}N_r = \sum_{i=-m}^{n-1} (r-1) r^i = r^n - r^{-m} \tag{8.1.9}$$

che, nel caso di un numero intero, m=0, diventa:

$$N_r + {}^{(r-1)}N_r = r^n - 1 \tag{8.1.10}$$

Il *valore*, espresso in base 10, segno e valore assoluto, di un numero segnato di (n+m) cifre, nel caso di complemento alla base diminuita, è dato da:

$$N_{10} = -\frac{S}{r-1} r^{n-1} + \sum_{i=-m}^{n-2} k_i r^i + \frac{S}{r-1} r^{-m} \tag{8.1.11}$$

dove S è la cifra che rappresenta il segno, che vale 0 o $(r-1)$, e si sono rappresentate genericamente con k_i le altre cifre, siano esse x_i o $\overline{x}_i$.

Nella rappresentazione *complemento alla base r* , il numero positivo è ancora rappresentato dalla (8.1.5), mentre il suo opposto si scrive come:

$$^{(r)}N_r = [(r-1), \overline{x}_{n-2}, \ldots \overline{x}_0, \overline{x}_{-1}, \ldots \overline{x}_{-m}] + r^{-m} \qquad (8.1.12)$$

ovvero, in base alla (8.1.6):

$$^{(r)}N_r = {}^{(r-1)}N_r + r^{-m} \qquad (8.1.13)$$

pertanto, in questa notazione, la (8.1.9) diventa:

$$N_r + {}^{(r)}N_r = r^n \qquad (8.1.14)$$

Il *valore*, espresso in base 10, segno e valore assoluto, di un numero segnato di (*n*+*m*) cifre, nel caso di complemento alla base, è dato da:

$$N_{10} = -\frac{S}{r-1} r^{n-1} + \sum_{i=-m}^{n-2} k_i r^i \qquad (8.1.15)$$

dove S è la cifra che rappresenta il segno e le k_i hanno il significato dato loro nella (8.1.11).

Val la pena di notare che la notazione alla base diminuita prevede due rappresentazioni dello 0, rappresentazione positiva e rappresentazione negativa, infatti applicando le (8.1.5) e (8.1.6), ricordando la (8.1.7), otteniamo:

$$0^+ = (0, 0, \ldots\ldots 0)$$
$$0^- = [(r-1), (r-1), \ldots\ldots (r-1)]$$

Conseguenza di questa ambiguità è il fatto che le somme tra numeri di segno opposto, che diano risultato positivo, oppure somme di numeri negativi sono mancanti di una unità, mentre sono corrette quelle fra numeri di segno opposto che diano risultato negativo oppure quelle fra numeri positivi. Supponiamo ad esempio di voler eseguire le somme :

$$34+(-20)=14$$

(–34)+20=–14
(–34)+(–34)=–68

Per n=3, nella notazione alla base 10 diminuita, diventano:

034+979=[1]013
965+020=985
965+965=[1]930

Applicando la (8.1.11) al numero 985 e al numero 930 si ottengono i risultati, con segno e valore assoluto:

–100+85+1=–14
–100+30+1=–69

Per ottenere il risultato corretto sarebbe sufficiente sommare, con peso uno, il riporto dalla cifra più significativa.

Nella rappresentazione alla base non diminuita questa ambiguità nella rappresentazione dello zero è evitata come si vede riscrivendo la (8.1.12) per lo zero che diventa, ricordando le (8.1.8) e (8.1.9):

$$0^- = [(r-1),(r-1),.....(r-1)]+1 = [r^n]+0^+ = 0$$

essendo r^n fuori dal campo delle n cifre considerate.

Le operazioni precedenti, scritte nella notazione alla base 10 non diminuita, diventano:

034+980=[1]014
966+020=986
966+966=[1]932

Applicando la (8.1.15) al numero 986 e al numero 932 si ottengono i risultati, corretti, espressi con segno e valore assoluto:

–100+86=–14
–100+32=–68

Nella rappresentazione alla base non diminuita, l'eliminazione dell'ambiguità nello zero è stata ottenuta a spese dei numeri positivi, infatti, in questa notazione, date n cifre, si rappresentano i numeri positivi da 0 a $(r^{n-1}-1)$, mentre nei numeri negativi è definito anche $-r^{n-1}$.

In entrambe le notazioni si hanno risultati scorretti se si sommano numeri dello stesso segno che superino la capacità di rappresentazione definita dalle *n* cifre. Ad esempio, restando nella rappresentazione base 10, non diminuita, con *n*=3, danno risultati scorretti:

$$(-90)+(-90)=920+920=[1]840$$

e

$$90+90=180$$

ricordando che la cifra più significativa può essere solamente 0 o 9.

Nel caso della base 2, che interessa le applicazioni digitali, la (8.1.11), convenzione del complemento a 1, diventa:

$$N_{10} = -S2^{n-1} + \sum_{i=-m}^{n-2} k_i 2^i + S2^{-m} \qquad (8.1.16)$$

mentre la (8.1.15), convenzione del complemento a 2, base non diminuita, diventa:

$$N_{10} = -S2^{n-1} + \sum_{i=-m}^{n-2} k_i 2^i \qquad (8.1.17)$$

Ricordando la (8.1.7), che nel caso *r*=2 diventa:

$$\overline{x}_i + x_i = 1 \qquad (8.1.18)$$

si vede che, in base 2, $\overline{x}_i$, è il complemento booleano di x_i, come definito dal P4a di §1.1, quando si associno alle cifre 1 e 0 i corrispondenti valori logici.

Come già visto nella convenzione complemento a 2 il numero 0 ha rappresentazione unica, mentre in quella complemento a 1 ha le due possibili rappresentazioni, 0 positivo:...00000 e 0 negativo:...11111.

Nelle unità aritmetiche digitali la notazione complemento a 2 è generalmente preferita proprio per la mancanza di ambiguità nella rappresentazione dello zero.

Per una scrittura e lettura più compatta dei numeri binari si usa frequentemente la notazione esadecimale (base 16) che raggruppa i bit quattro a quattro. Questa notazione richiede la definizione, con un singolo carattere, dei valori, da 0 a 15, che può

assumere la quartina. Si usano pertanto i numeri da 0 a 9, per i corrispondenti valori, e le lettere da *A* a *F* per i valori da 10 a 15.

Ad esempio risulta:

$$100111001010 = 9CA = 9 \bullet 16^2 + C \bullet 16 + A = 2506$$

Ricordiamo che, nella notazione binaria segnata, i numeri positivi, indipendentemente dal numero di cifre definite, sono preceduti, alla sinistra della prima cifra significativa, da una sequenza infinita di zeri, mentre, quelli negativi, da una infinita sequenza di uno alla sinistra della cifra 1 del segno.

8.2 La Somma

a)-Somme di numeri binari segnati

Il sommatore, descritto nel §3.1, opera correttamente sui numeri segnati nella convenzione complemento a 2.

Nel sottrattore, descritto nello stesso paragrafo, qualora si operi con minuendo minore del sottraendo si ottiene un risultato negativo consistente con la notazione complemento a 2.

Qualora si adottasse la notazione complemento a 1, il sommatore fornirebbe risultati che andrebbero corretti, sommando *I* al risultato, quando si avesse un riporto C_n, come visto nel paragrafo precedente.

Inoltre, nel caso di numeri di ugual segno, si ha un errore, evidenziato dal segnale *E*, se il risultato risulta di segno opposto a quello degli addendi. Le due condizioni sono espresse, rispettivamente, dalle relazioni:

$$I = C_n \tag{8.2.1}$$

$$E = (AB\overline{S} + \overline{A}\overline{B}S)_{n-1} \tag{8.2.2}$$

$$E = (\overline{A \oplus B})_{n-1}(A \oplus S)_{n-1}$$

Il circuito si realizza come in fig. 8.2.1.

La condizione di errore espressa dalla (8.2.2) vale, comunque, anche per la somma in complemento a 2 che, tuttavia, non richiede la correzione espressa dalla (8.2.1).

In entrambe i casi la condizione di errore viene eliminata se, operando su numeri a *n* bit, si prevede un risultato a (*n*+1) bit, aggiungendo alla sinistra del sommatore un'altra cella che duplichi quella relativa alla cifra (*n*-1).

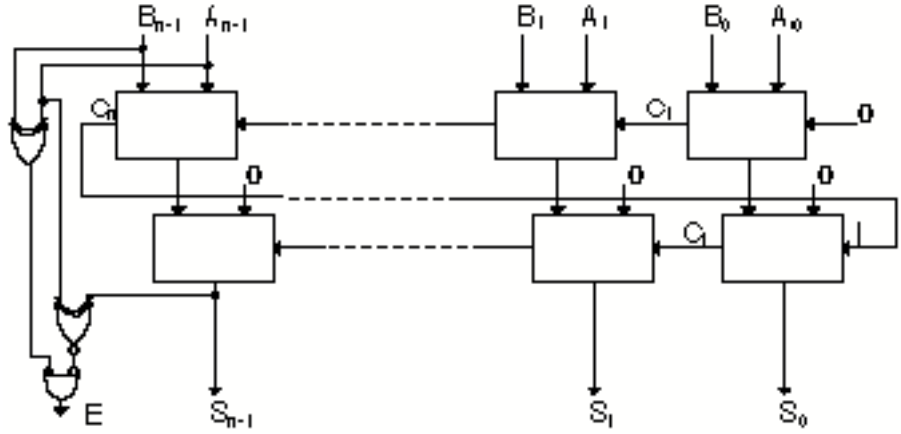


Figura 8.2.1

b)-Somme di numeri decimali codificati

Nel capitolo 6 abbiamo visto la rappresentazione dei numeri decimali codificati in binario (BCD). Nel caso si vogliano fare somme di numeri così codificati, il sommatore binario non opera correttamente perché, ad esempio, sommando 4 (0100) e 6 (0110) il risultato che esso fornisce, correttamente, è 10 ma rappresentato come (1010), mentre nella notazione BCD vorremmo ottenere 0 (0000) con riporto di uno al sommatore della cifra BCD successiva.
In generale si può dire che il risultato non è corretto ogni volta che si ottenga uno dei sei codici non ammessi: (1010), (1011), (1100), (1101), (1110), (1111). Il risultato è scorretto anche ogni volta che si ottenga un riporto, dal bit più significativo, anche se il risultato della somma è un codice ammesso. Ad esempio sommando 8 (1000) e 9 (1001) si ottiene 1 (0001) con riporto di uno, invece che 7 (0111) con riporto di uno al sommatore della cifra BCD successiva.

Si vede immediatamente che la correzione del risultato si ottiene sommando 6 al risultato parziale, quando esso non è corretto secondo la convenzione BCD, e che la condizione di correzione, [+6], è data o dalla presenza di un riporto o dal riconoscimento di un codice non ammesso:

$$[+6] = S_8 ' S_4 ' + S_8 ' S_2 ' + C_8 \qquad (8.2.3)$$

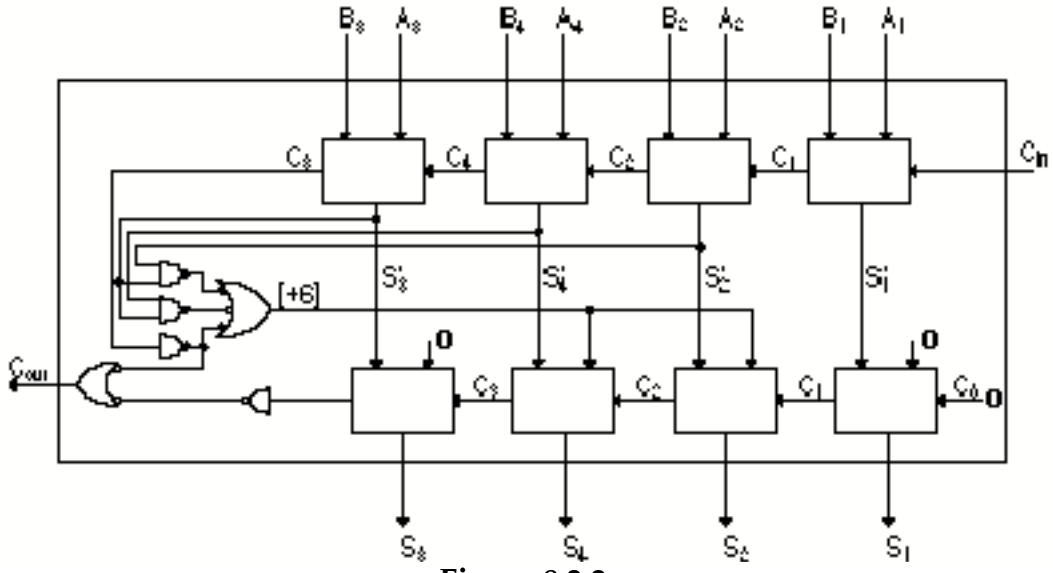


Figura 8.2.2

Il riporto, al sommatore della cifra immediatamente più significativa, è l'OR dei riporti del primo e del secondo sommatore.

Pertanto un sommatore BCD, realizzato con sommatori binari, si ottiene come in fig. 8.2.2.

Per operare su numeri decimali codificati si può usare anche la codifica *eccesso di 3*, che permette una certa semplificazione del circuito sommatore.

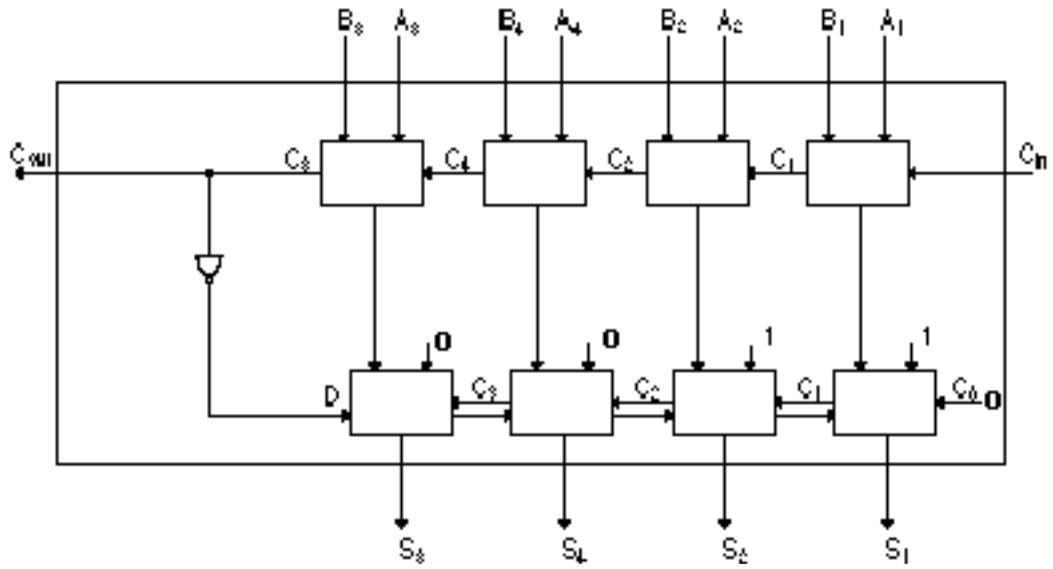


Figura 8.2.3

Nel codice eccesso di 3 le cifre decimali sono rappresentate dal codice BCD al quale si aggiunge 3. Pertanto, sommando due numeri così codificati, il risultato è il valore BCD aumentato di 6, come si vede dalla semplice relazione che segue:

$$[A_{BCD} + 3] + [B_{BCD} + 3] = [A + B]_{BCD} + 6$$

Per essere coerenti con la codifica è quindi necessario sottrarre 3 alla somma ottenuta. Tutto ciò è vero quando la somma rientri nei codici ammessi.

La condizione di correzione, in questa codifica, coincide con la generazione di un riporto. Nel caso si debba applicare la correzione, [+6], basta sommare 3, essendo il risultato della prima somma aumentato già di 3 rispetto al codice. Concludendo, se il primo sommatore non ha riporto, ovvero non richiede correzione, si deve sottrarre 3 al risultato parziale, se c'è riporto, ovvero è richiesta correzione, si deve aggiungere 3 al risultato parziale per una rappresentazione consistente col codice eccesso di 3.

Ricordando come si è costruito un sommatore-sottrattore nel §3.1, il circuito sommatore ad eccesso di 3, si realizza come in fig. 8.2.3 tenendo presente che il circuito a valle esegue la differenza, se *D*=**1**, e la somma se *D*=**0**.

8.3 Il Prodotto di numeri binari non segnati

Il prodotto di due numeri binari, positivi non segnati, di *n* bit si può calcolare con le stesse regole utilizzate normalmente nel calcolo decimale, ovvero eseguendo tutti i prodotti parziali e, successivamente, sommandoli. Notiamo inoltre che il prodotto di due numeri di *n* bit è sempre contenuto in un numero di 2*n* bit.

Ad esempio, se vogliamo eseguire il prodotto 3 per 5, di due numeri a 3 bit, si ottiene, nel modo convenzionale:

$$\begin{array}{r} 0\ 1\ 1 \\ \underline{1\ 0\ 1} \\ 0\ 1\ 1 \\ 0\ 0\ 0\ \ \\ \underline{0\ 1\ 1}\ \ \ \\ 0\ 1\ 1\ 1\ 1 \end{array}$$

e il risultato, 15 espresso in forma binaria, è corretto.

Il numero più grande esprimibile con 3 bit è 7. Il prodotto 7 per 7 eseguito nello stesso modo, dà come risultato 49, espresso in base due, che è contenuto in 6 bit come precedentemente detto e come risulta dall'operazione seguente:

$$\begin{array}{r} 1\ 1\ 1 \end{array}$$

$$
\begin{array}{r}
\underline{1\ 1\ 1} \\
1\ 1\ 1 \\
1\ 1\ 1 \\
\underline{1\ 1\ 1} \\
1\ 1\ 0\ 0\ 0\ 1
\end{array}
$$

Se notiamo che la moltiplicazione per 1 è, come ovvio, il numero stesso da moltiplicare, e che la moltiplicazione per 2, analogamente alla moltiplicazione per 10 nella numerazione decimale, si ottiene con lo spostamento a sinistra di una posizione, è evidente che la meccanizzazione delle operazioni precedenti si realizza facilmente disponendo di un sommatore e di un registro a scorrimento opportunamente connessi.

Notiamo, inoltre, che i bit del moltiplicatore definiscono, in successione, se si debba aggiungere alla somma dei prodotti parziali già calcolata, il valore 0 o il moltiplicando.

a)-Moltiplicatore sequenziale

A titolo di esempio disegnamo un moltiplicatore, detto appunto "add and shift", per due numeri di n bit. Supponiamo che inizialmente il moltiplicando, M, sia caricato in un registro di tipo D, e il moltiplicatore, m, sia caricato negli n bit più a destra di un registro a $(2n+1)$ bit del tipo di fig. 6.5.5 (registro D/S), i cui restanti $(n+1)$ bit a sinistra contengano tutti 0. Lo schema generale è dato in fig. 8.3.1. Il multiplexer seleziona gli ingressi provenienti dal registro D, se il segnale *Sel* vale **1**, mentre seleziona tutti zero se il segnale *Sel* vale **0**.

Il moltiplicatore del nostro esempio, esegue il calcolo del prodotto in otto passi, ognuno consistente in un'addizione, con caricamento del risultato nella parte a sinistra del registro D/S, seguito da uno spostamento a destra di tutti i bit contenuti nell'intero registro D/S. Il bit del riporto dalla cifra più significativa partecipa al processo. Il bit meno significativo del moltiplicatore aziona, ad ogni passo, il selettore (*Sel*) del multiplexer.

Il controllo dell'unità, nel caso di numeri a 8 bit, sarà effettuato da una macchina a stati del tipo di fig. 8.3.2a che fornisca una sequenza di segnali, inizializzata da un comando "Esegui", con la temporizzazione descritta in fig. 8.3.2b.

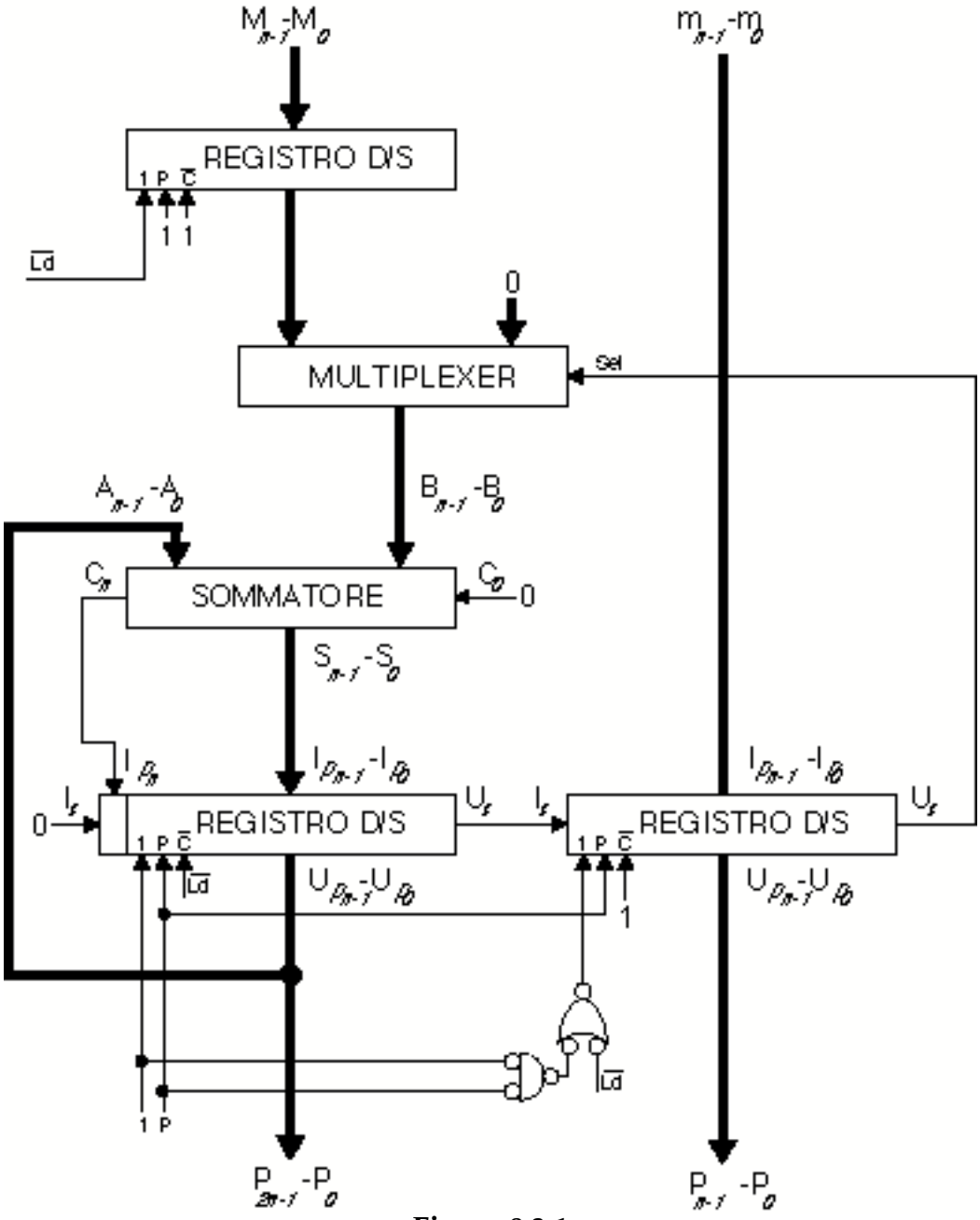
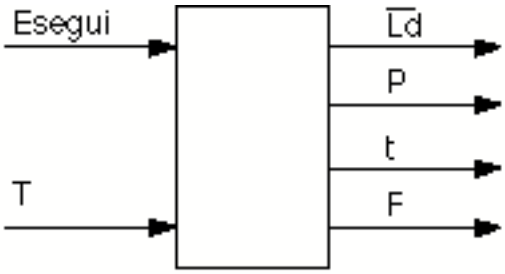
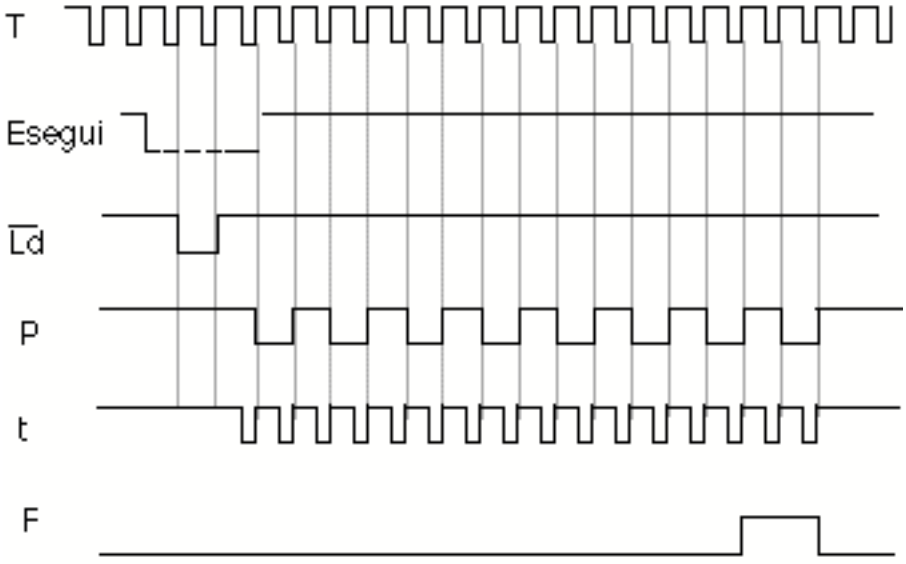

M n-1 -M 0
m n-1 -m 0
REGISTRO D/S
1 P C
1 1
Ld
0
MULTIPLEXER
Sel
A n-1 -A 0
B n-1 -B 0
Cn
SOMMATORE
C0
0
S n-1 -S 0
I ρn
I ρn-1 -I ρ0
Is
0
REGISTRO D/S
Us
U ρn-1 -U ρ0
1
P 2n-1 -P 0
P n-1 -P 0

Figura 8.3.1



a)



b)

Figura 8.3.2

Il dispositivo di controllo fornisce anche un segnale di fine conversione, *F*, che coincide con l'ultimo ciclo dell'operazione.

L'unità descritta precedentemente è di tipo *sequenziale* e, alla fine dell'operazione, il dato relativo al moltiplicatore *m* viene perso, visto che il risultato del prodotto, *P*, è contenuto nei 2*n* bit di destra del registro D/S. Si può comunque sempre aggiungere un altro registro D e caricarlo, allo stesso modo del registro riservato a *M*, con *m*, se ciò fosse necessario.

b)-Moltiplicatore combinatorio

Il prodotto fra due numeri non segnati si può ottenere anche attraverso la creazione dei singoli prodotti parziali e la loro somma. Discutiamo il caso di numeri a 4 bit, la estensione a un numero maggiore di bit risulterà immediata. Siano da moltiplicare i numeri *A* e *B*:

$$(A_3, A_2, A_1, A_0)$$
$$(B_3, B_2, B_1, B_0)$$

Il prodotto, P, si ottiene come segue:

$$\begin{array}{rr} A_3\ A_2\ A_1\ A_0 & \text{x} \\ B_3\ B_2\ B_1\ B_0 & = \\ \hline X_{03}X_{02}X_{01}X_{00} & + \\ X_{13}X_{12}X_{11}X_{10} & + \\ X_{23}X_{22}X_{21}X_{20} & + \\ \underline{X_{33}X_{32}X_{31}X_{30}} & = \\ P_7\ P_6\ P_5\ P_4\ P_3\ P_2\ P_1\ P_0 & \end{array}$$

E' evidente che risulta:

$$B_m \bullet A_n = X_{mn}$$

pertanto le singole X_{mm} si ottengono molto semplicemente con circuiti AND. Quindi utilizzando AND e sommatori a due bit, il circuito, completamente *combinatorio*, si disegna come in fig. 8.3.3.

Per realizzare, in generale, un moltiplicatore di due numeri da n bit, sono necessari n^2 AND e [n(n-1)] sommatori a due bit. Il tempo di risposta è imposto dal riporto che può dover circolare, in orizzontale e in verticale, attraverso 2n sommatori. Il ritardo totale del circuito è proporzionale a 2n.

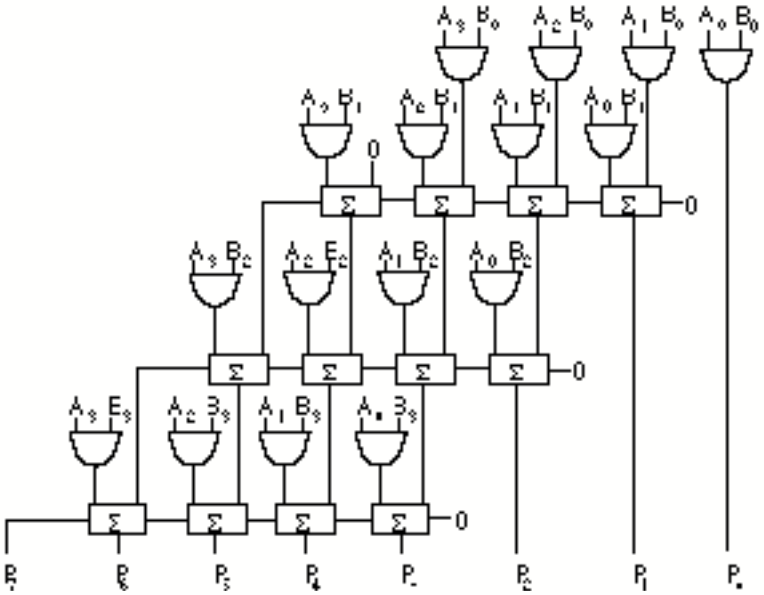


Figura 8.3.3

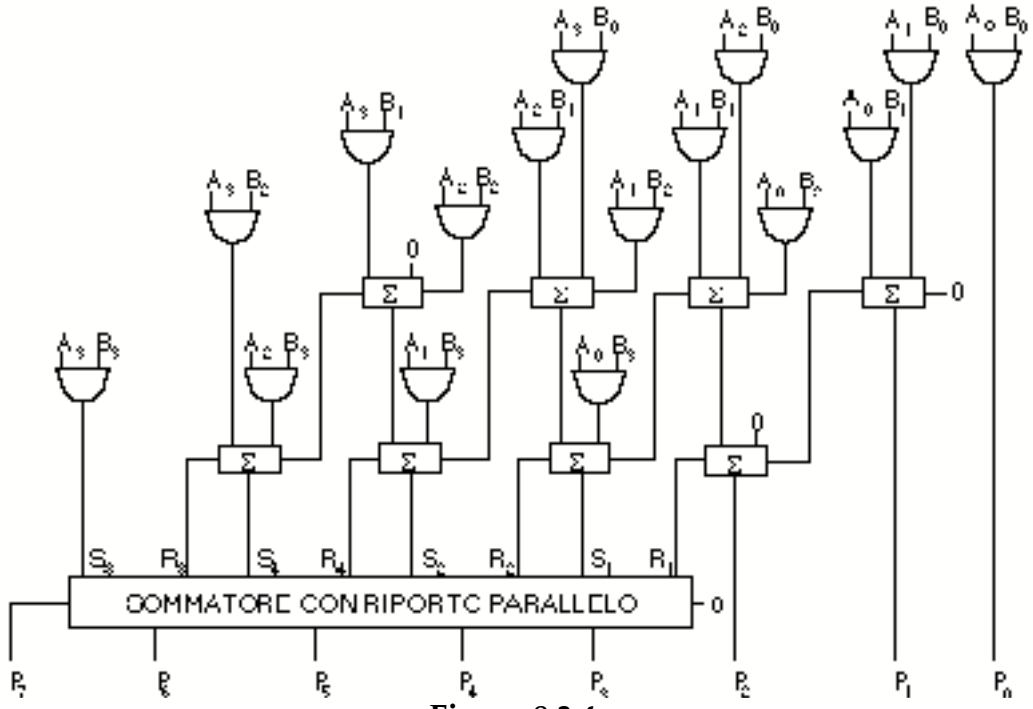


Figura 8.3.4

Riorganizzando le connessioni dei sommatori come in fig. 8.3.4, dove si sommano verticalmente tutti i prodotti parziali con uguale somma (m+n) degli indici, si ottiene una risposta più rapida.

In questa struttura si usano [n(n-2)] sommatori a un bit e un sommatore con riporto parallelo a n bit. Per ottenere il risultato il riporto deve propagarsi, solamente in verticale, attraverso gli (n-2) sommatori e attraverso il sommatore con riporto parallelo.

Nell'ipotesi di poter usare un solo sommatore parallelo che abbia lo stesso ritardo del sommatore ad un bit, il che tuttavia è possibile solo per n abbastanza piccolo, il ritardo del circuito è proporzionale a (n-1).

Vi sono altre varianti per l'esecuzione della moltiplicazione tramite somma di prodotti parziali del tipo X_{mn}, citiamo, ad esempio, senza entrare nel dettaglio, l'algoritmo di Booth e l'albero di Wallace (Wallace-tree).

c)-Moltiplicatore combinatorio con ROM

Rimanendo nel campo dei circuiti di tipo combinatorio che sommano prodotti parziali, è evidente che si potrebbe ottenere un notevole vantaggio in semplicità circuitale e in velocità, se si riuscissero a realizzare prodotti parziali di più di un bit.

Consideriamo ad esempio il caso che si vogliano moltiplicare fra loro numeri di 4 bit. Nel §6.7 abbiamo visto che le memorie

possono essere usate per realizzare funzioni booleane combinatorie: disponendo di una memoria con 256 bytes, si possono considerare le 8 linee d'indirizzo come costituite dal moltiplicando, *A*, i 4 bit meno significativi ad esempio, e dal moltiplicatore, *B*, gli altri 4 bit. In questo caso basta scrivere nella memoria, nella locazione individuata dall'indirizzo formato da (*A*,*B*), il contenuto ($A \bullet B$). La memoria diventa, in questo caso, una vera tavola pitagorica che fornisce tutti i prodotti di due numeri di quattro bit con un tempo di calcolo pari al suo tempo di accesso.

Un moltiplicatore di queste dimensioni tuttavia, è di interesse limitato, sebbene sia possibile realizzare in questo modo moltiplicatori anche a 8 per 8 bit con memorie di 2^{16} parole di 16 bit.

Vogliamo piuttosto vedere come sia possibile, in generale, estendere la capacità del moltiplicatore posto che si disponga di tavole di moltiplicazione per numeri da *n* bit.

Supponiamo che si disponga di memorie-moltiplicatrici per numeri di 4 bit e si voglia realizzare un moltiplicatore da 8 per 8 bit e siano (*M*,*N*) e (*R*,*S*), rispettivamente moltiplicando e moltiplicatore, dove evidentemente *M*, *N*, *R* e *S* sono numeri di 4 bit ciascuno. In definitiva dobbiamo eseguire la seguente moltiplicazione:

$$[(M \bullet 2^4) + N] \bullet [(R \bullet 2^4) + S] = P$$

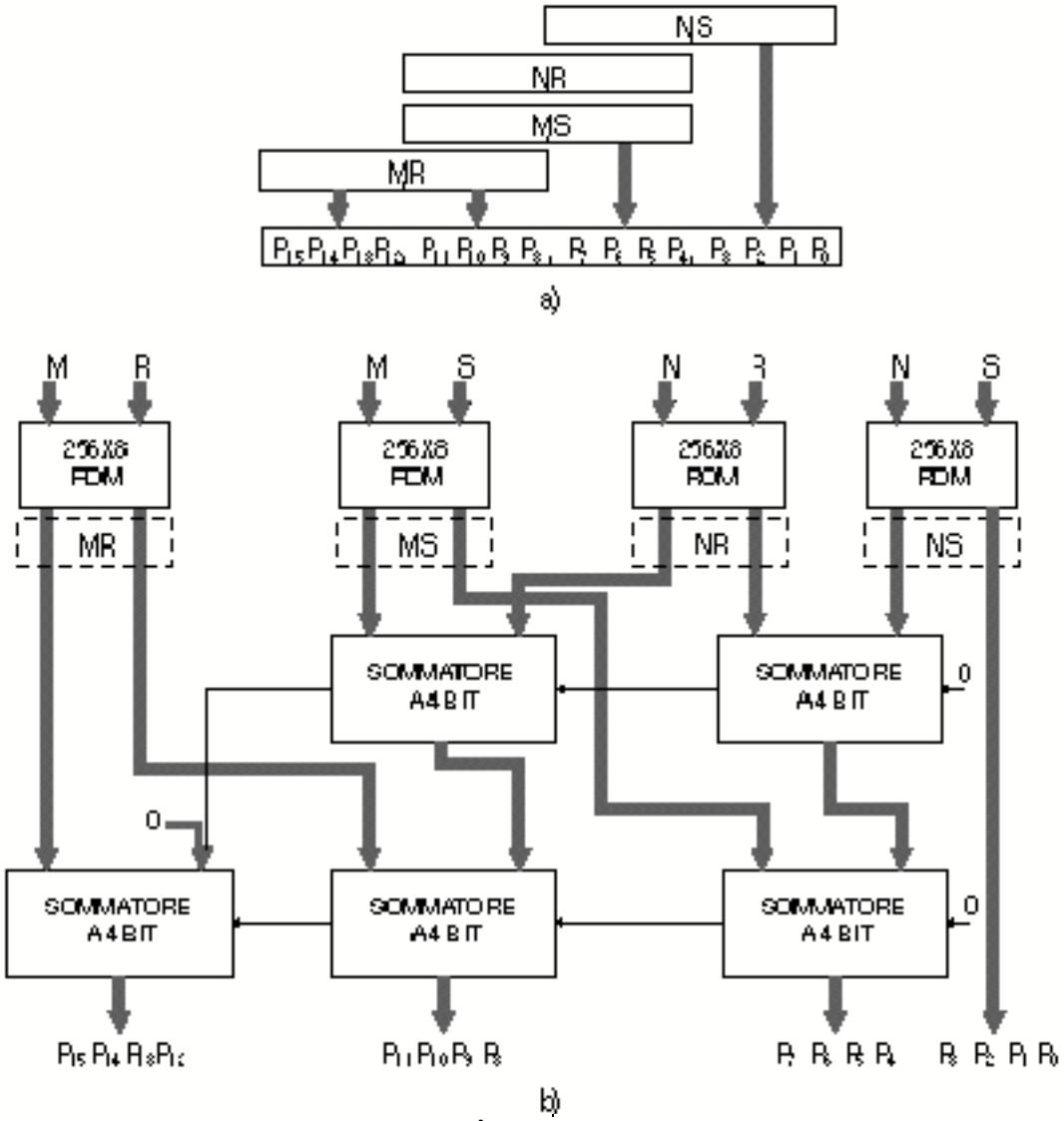


Figura 8.3.5

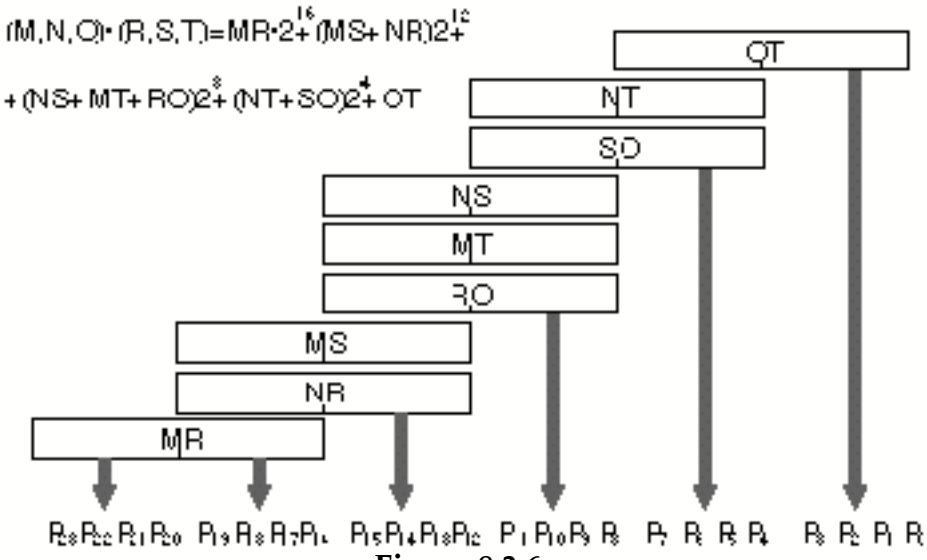


Figura 8.3.6

da cui risulta che il prodotto *P* si può ottenere dalla:

$$P = MR \bullet 2^8 + (NR + MS)2^4 + NS$$

Si vede immediatamente che i quattro prodotti parziali *MR*, *NR*, *MS* e *NS* si ottengono da quattro memorie; è poi sufficiente, per ottenere il prodotto finale, sommate i prodotti parziali secondo un adeguato allineamento degli stessi, visto che la moltiplicazione di un numero per una potenza di 2 equivale ad uno spostamento a sinistra, del numero, di un numero di posizioni uguale al valore dell'esponente.

Pertanto, disponendo di quattro memorie e di sommatori a 4 bit del tipo già precedentemente utilizzato, il moltiplicatore si realizza come nella fig. 8.3.5b, mentre nella fig. 8.3.5a è schematizzato l'allineamento dei prodotti parziali da sommare.

Naturalmente questo metodo, disponendo di tabelle moltiplicatrici ROM del tipo 4 per 4 bit, si può opportunamente estendere per realizzare moltiplicatori a 12 o a 16 bit.

Nel caso della moltiplicazione 12 per 12 bit si vede dalla fig. 8.3.6, che mostra come vanno allineati i prodotti parziali, che sono necessarie 9 memorie e 13 addizionatori.

Vedremo in seguito, trattando del prodotto di numeri segnati, un altro moltiplicatore costruito con memorie moltiplicatrici 8 per 8 bit.

8.4 Il Prodotto di numeri binari segnati

Il metodo di calcolo utilizzato all'inizio del paragrafo precedente per eseguire il prodotto di 011 per 101, con risultato 001111, oppure di 111 per 111, con risultato 110001, non è consistente con la notazione segnata, ad esempio complemento a 2, che nei due casi rispettivamente, richiederebbe i risultati 110111 (–-9), e 000001 (1).

Le unità di calcolo del paragrafo precedente sono utilizzabili unicamente con numeri positivi non segnati.

Se ricordiamo quanto scritto alla fine del §8.1, che cioè i numeri negativi sono preceduti da una sequenza infinita di 1 alla sinistra del segno, è immediato riconoscere quale sia l'errore che si è commesso nell'eseguire le moltiplicazioni precedenti. Infatti i prodotti parziali negativi sono troncati a sinistra e non sono consistenti con la notazione complemento a 2.

Supponiamo di voler eseguire 101 per 011 (–3X3). Si vede che il risultato è corretto, (–9), se si propaga il segno a sinistra, per i

prodotti parziali negativi, per tante posizioni quante sono le cifre del risultato :

```
          1 1 1
          0 1 1
  1 1 1 1 1 1
  1 1 1 1 1
  0 0 0 0
  1 1 1 1 0 1
```

Anche il prodotto di un numero positivo per un numero negativo, ad esempio 011 per 111 [3X(–1)], e di due numeri negativi, ad esempio 101 per 101 [(–3)X(–3)], risultano corretti se l'ultima cifra del moltiplicatore, qualora esso sia negativo, generi come prodotto parziale, il complemento a due del moltiplicando come nelle operazioni che seguono:

```
        0 1 1                 1 0 1
        1 1 1                 1 0 1
        0 1 1           1 1 1 1 0 1
      0 1 1             0 0 0 0 0
  1 1 0 1               0 0 1 1
  1 1 1 1 0 1           0 0 1 0 0 1
```

E' immediato verificare che il riporto dal sommatore, che esegue l'addizione dei prodotti parziali, non è mai significativo nel caso di prodotto di numeri segnati.

a)-Moltiplicatore sequenziale

L'unità di fig. 8.3.1, per operare correttamente, andrà modificata nella parte che esegue, dopo l'addizione, lo spostamento a destra delle cifre contenute nel registro D/S, secondo la regola seguente: *nel registro* D/S *deve entrare un* **1** *se il moltiplicando, eventualmente complementato, è negativo,* $M^*_{n-1} = \mathbf{1}$.

E' anche importante, quindi, distinguere l'ultimo ciclo del processo perché, nel caso di moltiplicatore negativo, durante esso va generato il complemento a 2 del moltiplicando, $(M_{n-1} - M_0)$, attraverso otto XOR controllati dall'AND di *F*, che segnala l'ultimo ciclo del procedimento, e del bit più significativo del moltiplicatore.

Il circuito sequenziale che genera il prodotto di numeri segnati, con procedura di "add and shift", è quello di fig. 8.4.1.

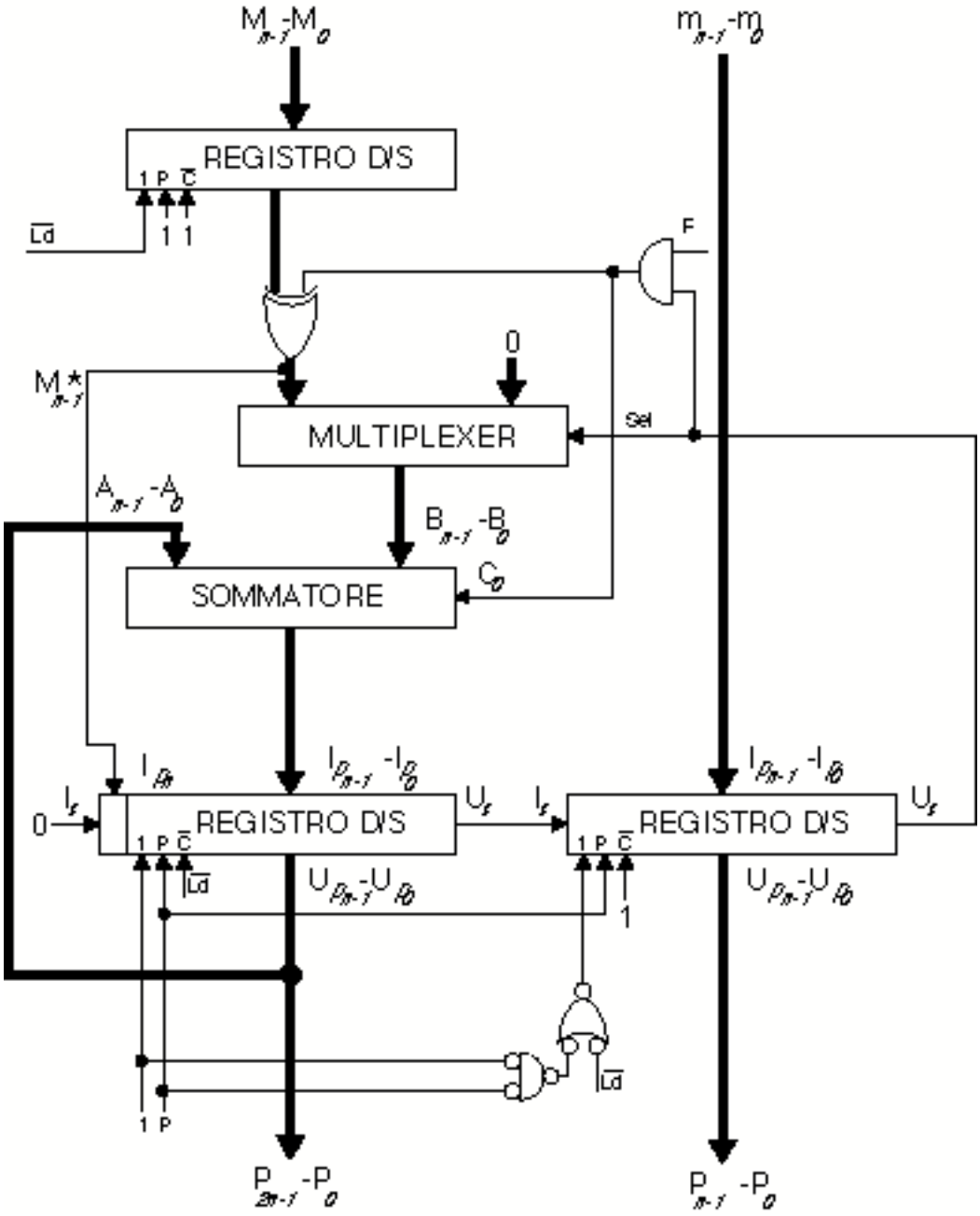


Figura 8.4.1

b)-Moltiplicatore combinatorio con ROM

Qualora si voglia generare il prodotto di due numeri segnati, ad esempio da 16 bit, partendo da ROM 8 per 8 bit, col metodo del calcolo di tutti i prodotti parziali e della loro somma dopo l'opportuno allineamento, bisogna premettere qualche considerazione sulla notazione segnata complemento a 2.

Se consideriamo un numero segnato a 16 bit (*M,N)*, ove *M* e *N* sono gruppi di otto bit, essendo *M* i bit più significativi, il valore del numero, espresso tramite la (8.1.17). è:

$$(M,N) = -S2^{15} + \sum_{i=8}^{14} m_i 2^i + \sum_{i=0}^{7} n_i 2^i \qquad (8.4.1)$$

$$(M,N) = (-S2^{7} + \sum_{i=0}^{6} m_i 2^i)2^8 + \sum_{i=0}^{7} n_i 2^i \qquad (8.4.2)$$

pertanto si vede che il valore è corretto se si interpreta come segnata la parte più significativa, *M*, e come espressa in valore assoluto, non segnato, la parte meno significativa *N*.

Se consideriamo la fig. 8.4.2, dove i gruppi sono costituiti da numeri di 8 bit, vediamo che *NS* è il prodotto fra numeri assoluti non segnati, *NR* e *MS* sono prodotti fra un numero segnato e uno assoluto mentre *MR* è il prodotto fra due segnati. Per un corretto risultato bisognerà propagare il segno, *s*, alla sinistra dei prodotti parziali che possono risultare negativi.

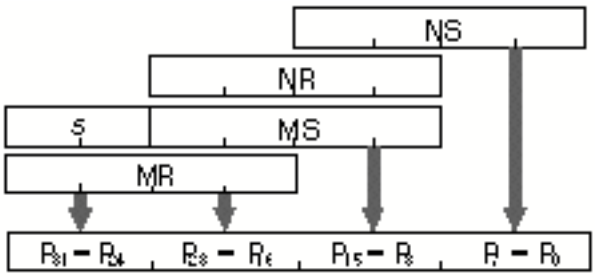


Figura 8.4.2

Se vogliamo procedere in modo analogo a quanto fatto in fig. 8.3.5b, dovremo disporre di tre tipi di memorie ROM, tutte della stessa capacità, in grado di eseguire i tre tipi di operazioni descritte e trovare un sistema per riconoscere quando deve essere propagato il segno.

Anzitutto notiamo che non c'è mai propagazione di riporto significativa dalla somma dei prodotti parziali:

$$(NR + MS)2^8 + NS \qquad (8.4.3)$$

infatti, essa vale al massimo, nel caso di *M* e *R* negativi:

$$-2^7[(2^8-1)+(2^8-1)]2^8 + (2^8-1)^2 = -16.646.655 = [\ldots F]01FE01_{16}$$

oppure nel caso di *M* e *R*positivi:

$$(2^7-1)[(2^8-1)+(2^8-1)]2^8 + (2^8-1)^2 = 16.646.145 = [\ldots 0]FE0001_{16}$$

essendo chiaro che nel caso che M ed N siano di segno opposto sicuramente la somma è contenuta in 24 bit.

Risulta pertanto importante trovare un metodo per calcolare il segno di (8.4.3) per eventualmente propagarlo a sinistra.

Se chiamiamo rispettivamente A e B i bit più significativi che intervengono nella somma di NR e MS, cerchiamo di trovare la funzione segno F, con la tabella 8.4.1 dove C è l'eventuale riporto dalla cifra immediatamente meno significativa di NR e MS.

Tabella 8.4.1

A	B	C	F
0	0	0	0
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	0
1	0	1	0
0	1	1	0
1	1	1	1

Tabella 8.4.2

modo	X	Y	$X_0 - X_7$	$Y_0 - Y_7$
non segnato	0	0	assoluto	assoluto
misto	0	1	assoluto	compl. 2
misto	1	0	compl.2	assoluto
segnato	1	1	compl.2	compl.2

Dalla tabella risulta:

$$F = B\overline{C} + AB + A\overline{C} \tag{8.4.4}$$

dalla quale deriva:

$$\overline{F} = \overline{B}C + \overline{A}\overline{B} + \overline{A}C \tag{8.4.5}$$

e, ricordando la (3.1.3), si riconosce che:

$$F = \overline{C}_{i+1} \tag{8.4.6}$$

se si fanno partecipare alla somma i bit più significativi invertiti, $\overline{A}$ e $\overline{B}$.

Ciò non comporta nessun problema per il risultato della somma, perché la (3.1.5), ricordando la (2.4.3), fornisce il risultato corretto anche se si invertono entrambi gli addendi.

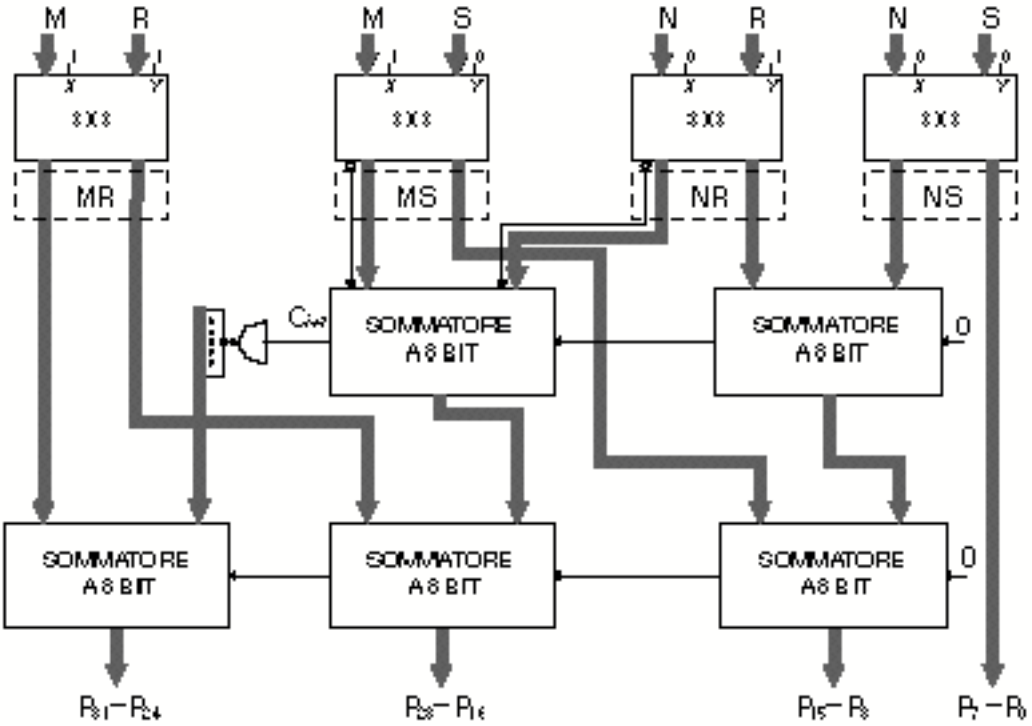


Figura 8.4.3

Il circuito che genera il prodotto si realizza come in fig. 8.4.3, disponendo di ROM che trattano numeri segnati e non segnati secondo lo stato delle linee di controllo X e Y, come definito dalla tabella 8.4.2, e di sommatori a 8 bit, eventualmente costruiti con sommatori a 4 bit.

8.5 La Divisione

Sebbene la divisione sia l'inverso della moltiplicazione, risulta un'operazione più difficile da meccanizzare.

Se consideriamo il processo di "add and shift" per la moltiplicazione, per la divisione il processo analogo sarebbe "subtract and shift", ma, mentre nel primo caso si possono generare tutti i prodotti parziali simultaneamente, per la divisione la sequenza delle operazioni è funzione del risultato della operazione precedente, sottrazione. La divisione è quindi un processo che dipende intrinsecamente dalla sequenza dei risultati dei singoli cicli che sono, tra l'altro, basati su tentativi.

Inoltre un'unità che faccia la divisione deve fornire oltre al quoziente Q, anche il resto R.

L'operazione di divisione può essere descritta in modo riccorrente come segue. Se n sono le cifre intere del dividendo $R^{(0)}$, r è la base, D è il divisore, allora possiamo scrivere:

$$R^{(i+1)} = R^{(i)} - q_{n-i} r^{n-i} D \qquad (8.5.1)$$

dove *i* è l'indice del processo ricorsivo, $R^{(i+1)}$ il resto parziale al passo *i*-esimo, q_i è il bit *i*-esimo del quoziente, a condizione che *D* sia un numero frazionario normalizzato, con la virgola a sinistra della cifra più significativa.

Sviluppando la (8.5.1), in base *r*=2, per due interi non segnati si ha:

per *i*=0 $$R^{(1)} = R^{(0)} - q_n r^n D \qquad (8.5.2)$$

dove evidentemente $R^{(0)}$ è il dividendo,

per *i*=1 $$R^{(2)} = R^{(1)} - q_{n-1} r^{n-1} D \qquad (8.5.3)$$
$$R^{(2)} = R^{(0)} - [q_n r^n + q_{n-1} r^{n-1}] D \qquad (8.5.4)$$

e per *i*=*n* $$R^{(n+1)} = R^{(n)} - q_0 D \qquad (8.5.5)$$
$$R^{(n+1)} = R^{(0)} - [q_n r^n + q_{n-1} r^{n-1} + \ldots + q_0] D \qquad (8.5.6)$$

dalla quale risulta:

$$R^{(n+1)} = R^{(0)} - QD \qquad (8.5.7)$$

dove quoziente è espresso chiaramente da:

$$Q = \sum_{i=0}^{n} q_{n-i} r^{n-i} \qquad (8.5.8)$$

Nel caso che *n* sia zero, ovvero sia *R* che *D* siano normalizzati, otteniamo:

$$R^{(i+1)} = R^{(i)} - q_{-i} r^{-i} D \qquad (8.5.9)$$

e quindi, analogamente a prima:

per *i*=0 $$R^{(1)} = R^{(0)} - q_0 D \qquad (8.5.10)$$

per *i*=1 $$R^{(2)} = R^{(1)} - q_{-1} r^{-1} D \qquad (8.5.11)$$
$$R^{(2)} = R^{(0)} - [q_0 + q_{-1} r^{-1}] D \qquad (8.5.12)$$

per *i*=*m* $$R^{(m+1)} = R^{(0)} - [q_0 + q_{-1} r^{-1} + .. + q_{-m} r^{-m}] D \qquad (8.5.13)$$
$$R^{(m+1)} = R^{(0)} - QD \qquad (8.5.14)$$

dove Q ora è un numero con 1 come parte intera e, m sono le cifre frazionarie.

Facciamo un esempio dell'uso della (8.5.1) per due numeri decimali: $R^{(0)}$=23, D=.54. In questo caso n=2.

Applicando la (8.5.2), (8.5.4) e (8.5.6), otteniamo:

per i=0 $\quad R^{(1)} = 23 - q_2 10^2 \bullet .54$

$$R^{(1)} = 23 - q_2 54$$

pertanto

$$q_2 = 0$$
$$R^{(1)} = R^{(0)} = 23.$$

Per i=1 $\quad R^{(2)} = 23 - q_1 10 \bullet .54$

dalla quale:

$$q_1 = 4$$
$$R^{(2)} = 1.4.$$

Per i=2 $\quad R^{(3)} = 1.4 - q_0 \bullet .54$

dalla quale:

$$q_0 = 2$$
$$R^{(3)} = .32.$$

In definitiva:

$$Q = 42$$
$$R = .32.$$

Facciamo ora un esempio di meccanizzazione di divisione binaria con dividendo e divisore frazionari. Ci proponiamo quindi di applicare la (8.5.9) e decidiamo di eseguire, in quattro passi, m=3 la divisone di

$$R^{(0)} = .1011$$

per $\quad D = .1001$

Per i=0

$$R^{(1)} = .1011 - q_0(.1001).$$

La cifra del quoziente parziale, q_{-i}, sarà determinata secondo la regola seguente:

$$q_{-i} = 1 \qquad \text{se} \qquad R^{(1)} \geq r^{-i} D \qquad (8.5.15)$$
$$q_{-i} = 0 \qquad \text{se} \qquad R^{(1)} < r^{-i} D. \qquad (8.5.16)$$

Pertanto

$$q_0 = 1$$

e per i=1
$$R^{(2)} = [.1011 - .1001] - q_1(.01001)$$
$$R^{(2)} = .001 - q_1(.01001)$$

dalla quale si deriva

$$q_1 = 0.$$

Per i=2

$$R^{(3)} = .001 - q_2(.001001)$$

.

dalla quale deriva che

$$q_2 = 0.$$

Infine per i=3

$$R^{(4)} = .001 - q_3(.0001001)$$

dalla quale deriva

$$q_3 = 1.$$

Concludendo abbiamo che

$$Q = 1.001$$
$$R = R^{(4)} = .0000111.$$

I due numeri che volevamo dividere, in decimale, erano .6875 e .5625, che danno come risultato della divisione $1.\overline{2}$. Noi otteniamo

un quoziente binario pari a 1.125 e un resto di .0546875. I binari frazionari approssimano, al crescere di m, il corrispondente valore decimale con maggiore accuratezza.

Si vede che nel processo ricorsivo da meccanizzare, espresso dalla (8.5.9), la difficoltà consiste nel determinare di volta in volta se q_i è zero o uno, applicando le (8.5.15) e (8.5.16), perché ciò deve essere fatto con uno o due tentativi interni al processo ricorrente. Questa scelta di q_i si realizza con una sottrazione e, eventualmente, un'addizione per ripristinare $R^{(i)}$ nel caso risulti vera la (8.5.16).

Questo tipo di divisione è detto infatti *divisione con ripristino* (*restoring division*). Esiste anche il metodo senza ripristino (*non restoring division*), che permette esecuzioni più veloci, del quale ci limitiamo a dire che la scelta del q_i viene fatta in un solo passo e si rimedia all'eventuale errore nel passo successivo.

Esistono molti altri modi per trattare la divisione, ma in questo paragrafo discutiamo solamente un metodo simile a quello utilizzato per la moltiplicazione attraverso ROM e sommatori.

Supponiamo di voler dividere un numero di 8 bit (M,N) per un numero di 4 bit (D). Il quoziente Q può essere espresso, analogamente a quanto fatto nel paragrafo precedente, dalla relazione:

$$Q = \frac{M}{D}2^4 + \frac{N}{D} \tag{8.5.17}$$

dove le frazioni rappresentano divisioni fra numeri da 4 bit, che pertanto possono essere realizzate con ROM da 256 parole. Ci chiediamo tuttavia quanti bit debba avere ciascuna parola perché il quoziente Q sia espresso con una adeguata risoluzione, infatti ciascuna delle due divisioni avrà un resto del quale bisognerà tener conto nella somma dei quozienti parziali.

Sarà necessario comunque definire un certo numero di posti alla destra della virgola per dare una rappresentazione adeguata di Q.

Le tabelle ROM che realizzano la divisione fra numeri di 4 bit, dovranno rappresentare adeguatamente i numeri da (15/1) a (1/15), valendo quest'ultimo $0,0\overline{6}$.

Tuttavia bisogna tener conto che nell'allineamento del quoziente parziale più significativo, la virgola viene spostata a destra di quattro posti, pertanto, se si vuole una risoluzione compatibile con quattro bit frazionari ($2^{-4} = 0,0625$), bisognerà realizzare una ROM per M/D con 12 bit, 8 dei quali frazionari,

mentre per la ROM che calcola N/D saranno sufficienti 8 bit, 4 dei quali frazionari.

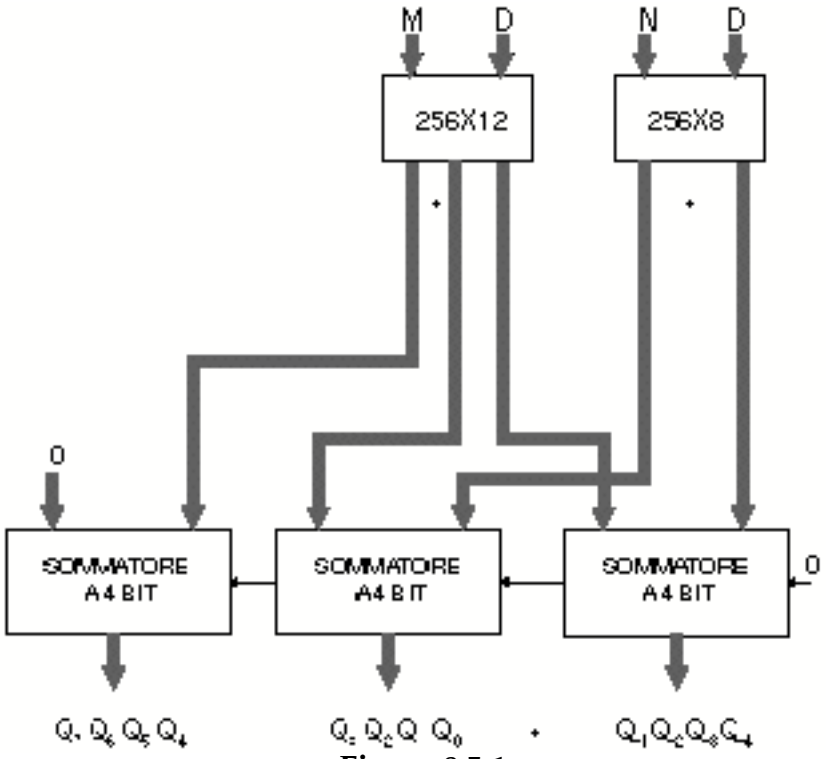


Figura 8.5.1

Il circuito che realizza la divisione di due numeri interi rispettivamente di 8 e 4 bit, con quoziente con 4 cifre frazionarie, è dato in fig. 8.5.1.

L'estensione al caso di numeri con numero maggiore di cifre è immediato: basta tener conto che il numero di cifre frazionarie nei quozienti parziali cresce progressivamente secondo il peso dei quozienti stessi.

8.6 Esercizi

a)-Disegnare una unità aritmetica di somma e sottrazione per numeri BCD.

b)-Disegnare un moltiplicatore per numeri BCD di tipo combinatorio.

c)-Disegnare un moltiplicatore BCD, combinatorio, per numeri segnati (complemento a 10).

d)-Disegnare un moltiplicatore BCD sequenziale per numeri positivi.

e)-Disegnare un moltiplicatore BCD sequenziale per numeri segnati (complemento a 10).

f)-Disegnare il divisore sequenziale descritto nel §8.5.