

Esercizi

3.4 Progettare un selettore di ingresso-uscita, per tre messaggi di ingresso e tre messaggi di uscita.

3.3 Unità aritmetiche-logiche

I sistemi di elaborazione operano usualmente su *parole* (cioè stringhe binarie) di lunghezza prefissata n . L'informazione relativa a un problema è rappresentata da un insieme di parole prese come dati elementari, ed è assai meno comune intervenire all'interno di una parola operando indipendentemente su parti diverse di essa.

I motivi per individuare nella parola (anziché nel bit singolo) il costituente di base dell'informazione da elaborare risulteranno evidenti quando studieremo la struttura dei sistemi. Per ora notiamo che anche le reti combinatorie si avvalgono di questo schema, poiché, una volta costruite, legano al numero prefissato di variabili di ingresso la lunghezza delle stringhe binarie su cui operare.

Le parole possono rappresentare numeri, elementi lessicali di un linguaggio o altro. Esse però sono spesso elaborate *come se fossero numeri*, trasferendo poi a livello logico il significato di operazioni aritmetiche su informazioni diverse. Questa interpretazione è dominio delle tecniche di programmazione in senso lato, ed esula dai nostri scopi. Ci occupiamo invece in questo paragrafo di studiare la costituzione interna delle reti che eseguono operazioni aritmetiche, estendendo poi il loro funzionamento ad alcune operazioni logiche.

3.3.1 Un richiamo sulla rappresentazione dei numeri

Un numero intero relativo A è individuato dal *segno* S e dal *modulo* M . In notazione binaria A è rappresentato con un stringa di n bit: $A = a_{n-1}a_{n-2} \dots a_0$, ove il bit più significativo a_{n-1} rappresenta S ($a_{n-1} = 0$ se positivo, $a_{n-1} = 1$ se negativo), e i restanti bit $a_{n-2} \dots a_0$ rappresentano M secondo varie notazioni.

Le tre notazioni più comuni sono quelle in *segno e modulo*, *complemento a 1* e *complemento a 2*. La rappresentazione dei numeri positivi ($a_{n-1} = 0$) è comune ai tre casi: il modulo M vi è rappresentato nell'usuale notazione posizionale (codice *binario naturale*), cioè

$$\sum_{l=0}^{n-2} a_l 2^l = M.$$

Per i numeri negativi ($a_{n-1} = 1$) si ha invece:

1) in segno e modulo: M è rappresentato in codice binario naturale, come per i numeri positivi;

2) in complemento a 1: M è rappresentato dal valore $N = 2^{n-1} - 1 - M$ (espresso in codice binario naturale);

3) in complemento a 2: M è rappresentato dal valore $N = 2^{n-1} - M$ (espresso in codice binario naturale).

I punti 2) e 3) possono essere invocati anche per la trasformazione opposta, cioè per riconvertire un numero negativo (espresso in complemento a 1 o a 2) in numero positivo, poiché si ha:

$$2^{n-1} - 1 - N = 2^{n-1} - 1 - (2^{n-1} - 1 - M) = M,$$

e

$$2^{n-1} - N = 2^{n-1} - (2^{n-1} - M) = M.$$

Ricordando che in codice binario naturale risulta:

$$2^{n-1} = 100 \dots 0 \quad (n \text{ bit})$$

e

$$2^{n-1} - 1 = 11 \dots 1 \quad (n-1 \text{ bit})$$

abbiamo la seguente regola pratica per il cambiamento di segno di un numero qualsiasi, positivo o negativo.

Regola Costruzione dell'opposto di un numero binario $a_{n-1} \dots a_0$:

- 1) in segno e modulo: si sostituisce il bit a_{n-1} con \bar{a}_{n-1} ;
- 2) in complemento a 1: si sostituisce ogni bit a_i con \bar{a}_i ($0 \leq i \leq n-1$);
- 3) in complemento a 2: si sostituisce ogni bit a_i con \bar{a}_i ($0 \leq i \leq n-1$) e si somma 1 al numero binario rappresentato, in codice binario naturale, dalla stringa risultante.

Per esempio, per $n=4$, gli interi relativi nelle tre notazioni binarie sono rappresentati nella tabella 3.1, su cui può anche riscontrarsi la regola di costruzione degli opposti.

Per quanto riguarda il punto 3) della regola (complemento a 2), si noti che la configurazione $00 \dots 0$ ($+0$) si trasforma in $11 \dots 1$ e quindi,

Tabella 3.1
Notazioni binarie per interi relativi ($n=4$)

	Positivi o nulli Tutte le notazioni	Negativi o nulli			
			Segno e modulo	Complemento a 1	Complemento a 2
+0	0000	-0	1000	1111	
+1	0001	-1	1001	1110	1111
+2	0010	-2	1010	1101	1110
+3	0011	-3	1011	1100	1101
+4	0100	-4	1100	1011	1100
+5	0101	-5	1101	1010	1011
+6	0110	-6	1110	1001	1010
+7	0111	-7	1111	1000	1001

addizionando 1, in 00...0 con riporto di 1 verso la posizione $(n+1)$ -esima: trascurando tale riporto perché fuori parola, il cambiamento di segno trasforma la stringa 00...0 in sé stessa. Vi sono in sostanza due modi di rappresentare lo zero nelle notazioni in segno e modulo e in complemento a 1, e un solo modo in complemento a 2.

Gli interi rappresentati in una delle notazioni viste sono evidentemente compresi in modulo tra 0 e $2^{n-1}-1$, limitazione che deriva dalla lunghezza n della parola. I numeri non interi vengono rappresentati attraverso le cifre più significative intere e decimali (nel massimo numero consentito dal campo ad esse destinato) e la posizione della virgola; non approfondiamo qui il problema, perché nulla aggiungerebbero di sostanziale allo studio dell'unità aritmetica per gli interi.

3.3.2 L'addizionatore

L'addizione rappresenta un esempio tipico di problema che può essere affrontato in modo combinatorio o sequenziale, e con essa tutte le altre operazioni aritmetiche.

Supponiamo inizialmente che gli addendi A e B siano rappresentati (senza segno) in codice binario naturale: $A = a_{n-1}a_{n-2} \dots a_0$, $B = b_{n-1}b_{n-2} \dots b_0$; e che la loro somma sia $S = s_{n-1}s_{n-2} \dots s_0$ (nel caso la somma dovesse richiedere $n+1$ bit, genereremo un segnale di "supero di capacità"). E' anzitutto ovvio che il problema può risolversi con una rete combinatoria a $2n$ ingressi (cui collegare "in parallelo" A e B), e n uscite su cui si rileva S (più un'uscita che indica l'eventuale supero di capacità). Posponiamo la discussione su tale rete per studiare che cosa avviene se l'addizione è eseguita, come nell'usuale procedi-

mento a mano, cifra a cifra (cioè bit a bit) partendo dalle cifre meno significative.

L'addizione di due cifre può generare un riporto da sommare alle cifre successive: la situazione standard è dunque quella di sommare due bit a , b in posizioni corrispondenti negli addendi, più il bit r di riporto dalla posizione precedente, e generare in corrispondenza il bit s di somma, e il bit r^* di riporto per le cifre successive. Le varie situazioni che possono presentarsi sono elencate nella tabella di figura 3.9a.

Sulla scorta delle indicazioni fornite dalla figura 3.9a possiamo eseguire l'addizione tra i numeri A e B , notando che i bit meno significativi a_0 , b_0 non devono essere sommati a riporti precedenti, cioè per essi si pone $r = 0$. Ad esempio, per $A = 010111$, $B = 000101$ ($n = 6$) abbiamo:

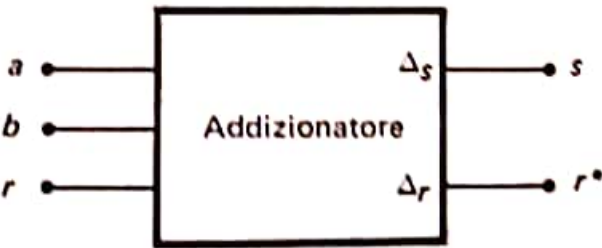
riporto r	0	0	1	1	1	(0)
addendo A	0	1	0	1	1	1
addendo B	0	0	0	1	0	1
	<hr/>					
somma S	0	1	1	1	0	0

Evidentemente s e r^* sono funzioni combinatorie di a , b , r , e possono essere generate da una rete standard detta *blocco addizionatore* (fig. 3.9b), sulla cui struttura indagheremo tra breve. Comunque costruito, il blocco genera s e r^* con ritardi Δ_s e Δ_r rispetto all'applicazione dei segnali di ingresso.

Un solo blocco addizionatore non è sufficiente a realizzare una rete

a	b	r	s	r^*
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a)



(b)

Figura 3.9

L'addizione di tre bit a , b , r . (a) Valori della somma s e del riporto r^* ; (b) simbolo del blocco addizionatore.

combinatoria per l'esecuzione dell'addizione tra numeri di n bit, poiché sarebbe necessario a ogni passo memorizzare il riporto generato, che dovrebbe essere addizionato al passo successivo (occorrerebbe quindi una rete sequenziale). E' possibile però costruire una rete combinatoria relativamente semplice e altamente modulare usando più blocchi addizionatori: tale rete, detta usualmente *addizionatore parallelo* o semplicemente *addizionatore*, è riportata nella figura 3.10. Il funzionamento è ovvio; per addendi in codice binario naturale senza segno, il riporto r_n dello stadio più significativo si interpreta come supero di capacità (se $r_n = 1$ la somma richiede $n + 1$ bit significativi anziché n); sull'ingresso r_0 del primo stadio si applica la costante 0.

Calcolare il tempo T di funzionamento dell'addizionatore parallelo non è cosa ovvia. Dallo schema di figura 3.10 appare che il massimo numero di blocchi è attraversato dai segnali a_0, b_0 , che tramite i segnali r_1, \dots, r_{n-1} possono concorrere alla formazione di s_{n-1} (trascuriamo r_n che non fa strettamente parte del risultato). Dovrebbe quindi risultare: $T = (n-1) \Delta_r + \Delta_s$. Abbiamo detto "dovrebbe", perché potremo confermare che i segnali a_0, b_0 attraversano effettivamente tutti gli stadi, solo dopo aver constatato che esiste almeno una situazione di ingresso, per cui il valore di s_{n-1} è funzione di a_0, b_0 . Tali situazioni sono di fatto numerose (vedi esercizio 3.6); per esempio nell'addizione:

Stadio	5	4	3	2	1	0
r	1	1	1	1	1	(0)
A	0	1	1	1	1	1 +
B	0	0	0	0	0	1 =
S	1	0	0	0	0	0

ogni riporto r_i può essere generato correttamente solo dopo che la rete ha valutato il riporto r_{i-1} .

Il fenomeno visto è solitamente denominato *propagazione del riporto*; tale propagazione si arresta in media dopo pochi stadi: il lettore potrà controllare che nell'addizione:

Stadio	5	4	3	2	1	0
r	1	1		1	1	
A	0	1	1	0	1	1
B	0	1	1	0	0	1
S	1	1	0	1	0	0

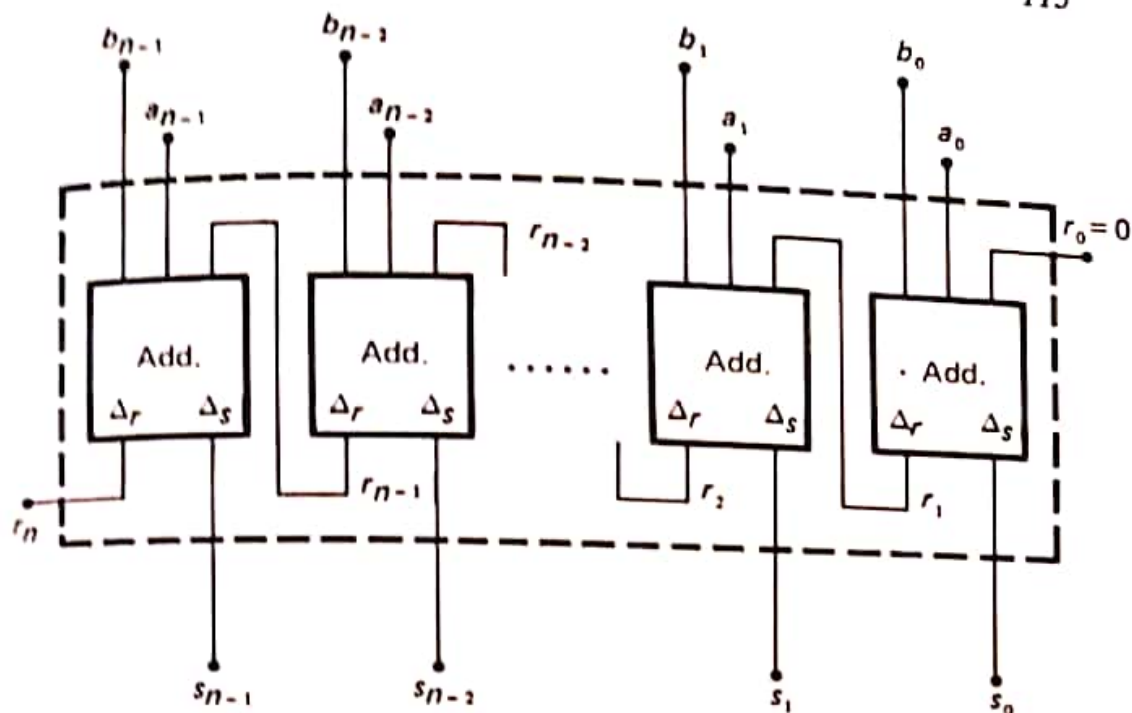


Figura 3.10
Addizionatore parallelo.

il riporto r_1 si propaga attraverso due stadi, concorrendo alla formazione di s_1 e s_2 ; il riporto r_4 si propaga attraverso *un solo stadio*, concorrendo alla formazione di s_4 ma non alla formazione di r_5 (quindi di s_5) che è generato immediatamente da a_4, b_4 .

Da quanto detto deriva che il tempo di funzionamento dell'addizionatore parallelo è funzione dei dati di ingresso. Poiché tali dati possono presentarsi nella configurazione più sfavorevole, è in genere necessario attendere un tempo $T = (n-1) \Delta_r + \Delta_s$ prima di interpretare il risultato. Per valutare l'entità del fenomeno, dobbiamo studiare la struttura interna di ogni blocco addizionatore, e calcolare Δ_r e Δ_s .

Dalla tabella di figura 3.9a ricaviamo immediatamente le mappe di Karnaugh, e le realizzazioni SP minime, per le funzioni s e r^* (fig. 3.11a).

Poiché, come si è visto, il tempo T è legato sostanzialmente a Δ_r , è importante realizzare la r^* a due livelli, mentre poco influisce sul ritardo complessivo la realizzazione di s . Per tale motivo, mentre la r^* è sempre realizzata nella forma indicata nella figura 3.11a (o equivalenti NAND o NOR a due livelli), moltissime forme diverse sono state proposte per la s , con lo scopo di ridurre la complessità della rete. Nella figura 3.11b indichiamo come un'interessante forma a 5 livelli di s possa essere costruita in modo "artigianale" per modificazioni successive della mappa di \bar{r}^* , proponendo infine per il blocco addizionatore

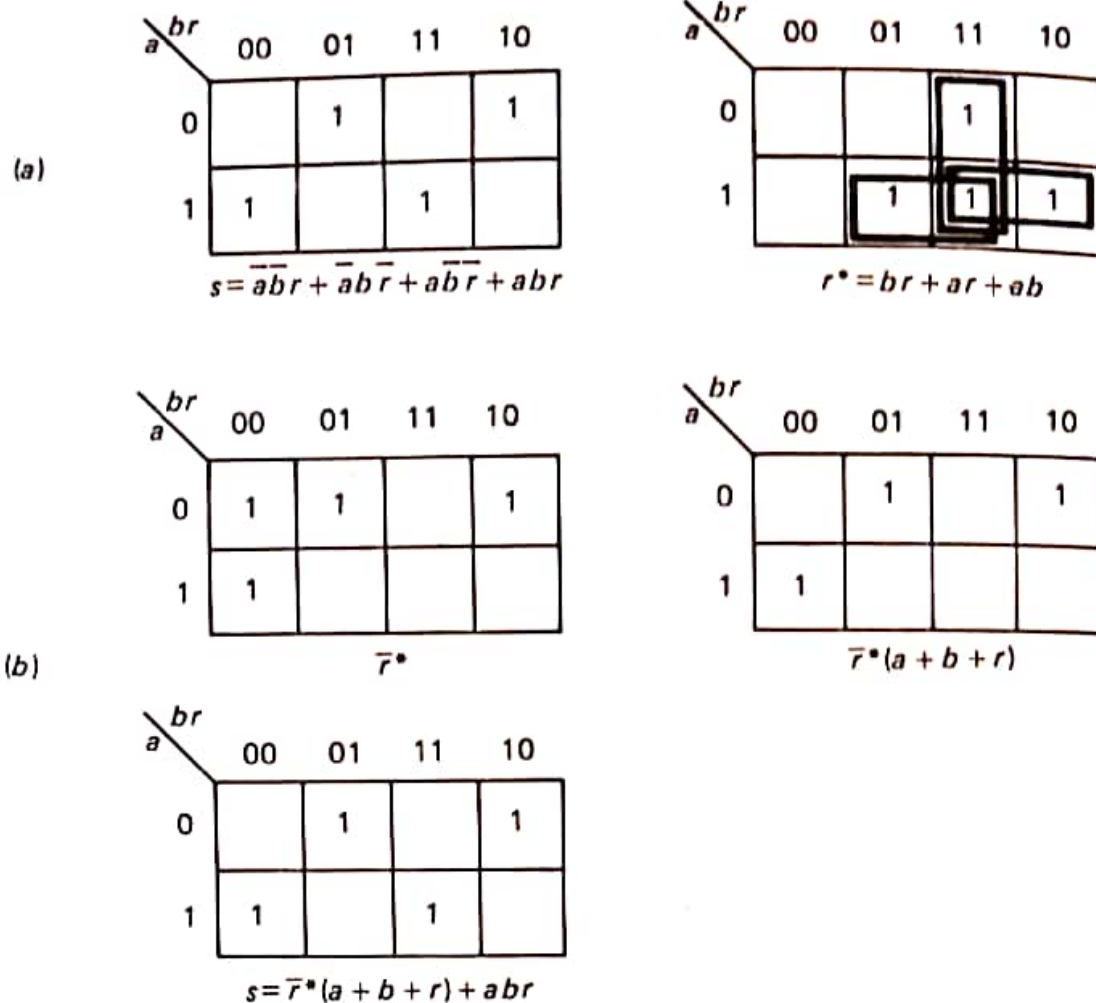


Figura 3.11

Costruzione delle espressioni algebriche per le funzioni s e r^* del blocco addizionatore. (a) Forme SP minime; (b) realizzazione più economica della s , in forma a 5 livelli (\bar{r}^* richiede tre livelli).

la rete di figura 3.12, basata sulle forme così ottenute:

$$r^* = br + ar + ab,$$

$$s = \bar{r}^*(a + b + r) + abr.$$

Notiamo che la rete non richiede le variabili di ingresso in forma negata. Rimandiamo il lettore alla bibliografia per lo studio di tante altre reti possibili.

Possiamo ora prendere nuovamente in esame il fenomeno della propagazione del riporto. Con lo schema della figura 3.12, l'addizionatore parallelo risulta una rete a $(n-1)2 + 5 = 2n + 3$ livelli, poiché i segnali a_0, b_0 , per raggiungere l'uscita s_{n-1} , attraversano due livelli (funzione

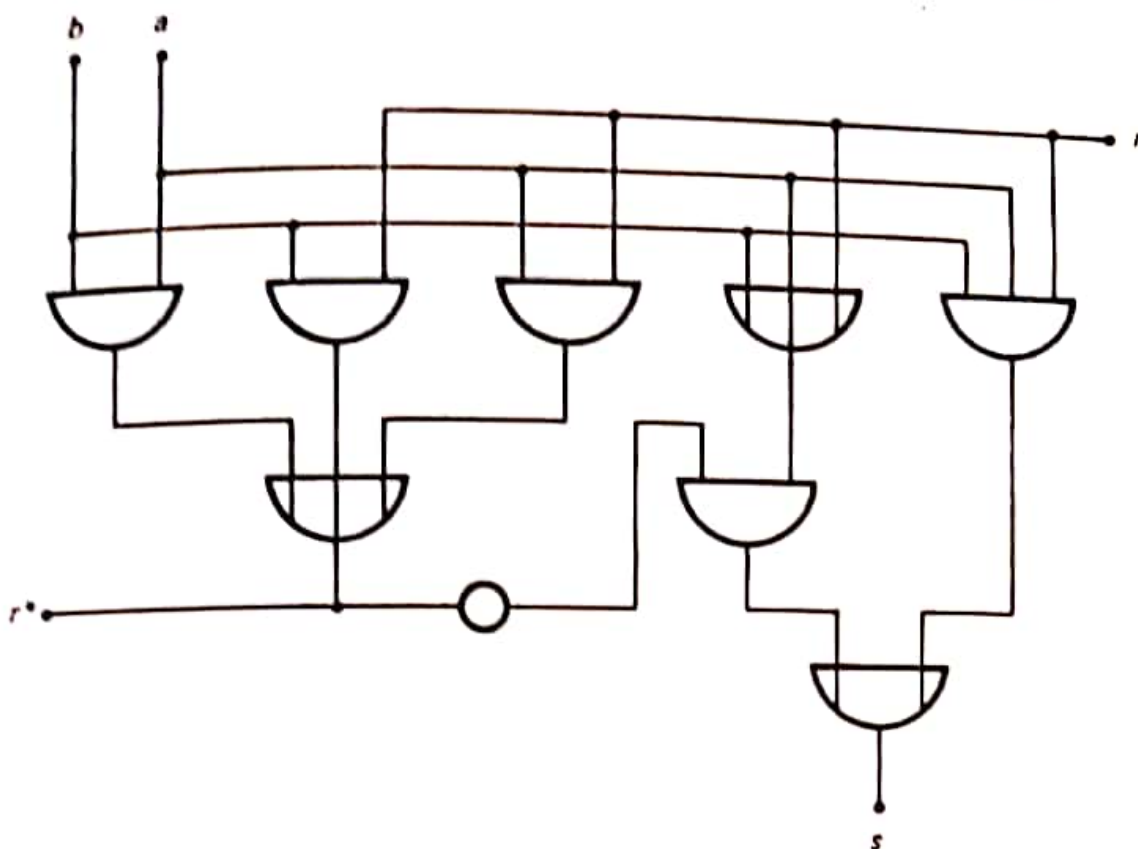


Figura 3.12

Una rete per il blocco addizionale.

r^*) nei primi $n-1$ blocchi, e cinque livelli (funzione s) nell' n -esimo blocco.

Il gran numero di livelli è ovviamente compensato dalla semplicità della rete: notiamo infatti che nulla vieterebbe in linea di principio di costruire un addizionale "veramente" parallelo, come rete combinatoria a $2n$ ingressi e n uscite (più l'uscita di "supero") realizzata a due livelli. Tale rete produrrebbe il risultato con un ritardo dovuto unicamente ai due livelli, e la propagazione del riporto sarebbe totalmente eliminata; d'altra parte la sua realizzazione sarebbe così costosa da non poter essere presa in considerazione se non per valori piccolissimi di n .

Per ridurre il ritardo di propagazione del riporto sono stati studiati numerosi schemi: ne discuteremo brevemente due, come esempi di progetto di reti combinatorie. Il primo schema è un ovvio compromesso tra l'addizionale di figura 3.10 e la rete a due livelli cui si è accennato prima: si suddivide l'addizionale in p gruppi di q stadi ciascuno ($p \cdot q = n$); si realizza ciascun gruppo come rete combinatoria a 2 livelli, con $1 + 2q$ ingressi e $q + 1$ uscite (l'ultima uscita è il riporto del gruppo).

il primo ingresso è il riporto dal gruppo precedente); si collega ogni gruppo al successivo attraverso il riporto. Il tempo di funzionamento della rete è ora $T' = (p-1) \Delta_r' + \Delta_s'$, ove Δ_r' e Δ_s' sono i ritardi dei gruppi per generare il riporto, e la somma. Rispetto al caso precedente abbiamo $T' \cong T/q$. La scelta dei valori di p e q è determinata da un compromesso tra il costo della rete, che cresce in modo complesso con q , e il tempo di funzionamento, che cresce linearmente con p .

Il secondo schema prevede nuovamente la suddivisione della rete in p gruppi di q stadi, ma, anziché accelerare la propagazione del riporto all'interno di ogni gruppo, si accelera la propagazione del riporto tra gruppi. Si costruisce ogni gruppo mediante q blocchi addizionatori collegati in modo standard (come in fig. 3.10); si lascia propagare il riporto naturalmente all'interno di ogni gruppo (operazione che richiede un tempo $q \cdot \Delta_r$); si studiano le condizioni secondo cui il riporto all'uscita del gruppo $(i-1)$ -esimo ha effetto sul gruppo $(i+1)$ -esimo, e si costruisce una rete logica che permetta a tale riporto di essere applicato immediatamente a quest'ultimo gruppo, senza dover attraversare il gruppo i -esimo. Lo schema risultante è quello di figura 3.13: il riporto r_i è applicato direttamente al gruppo i -esimo, e ai successivi $(i+1)$, $(i+2)$ ecc. solo se i segnali c_i, c_{i+1} ecc. hanno valore 1. Studiamo ora come si stabilisce il valore di questi segnali.

Prendiamo in considerazione il gruppo i -esimo: gli ingressi che lo interessano siano $a_{q-1}^i, \dots, a_0^i, b_{q-1}^i, \dots, b_0^i$, nonché il riporto prece-

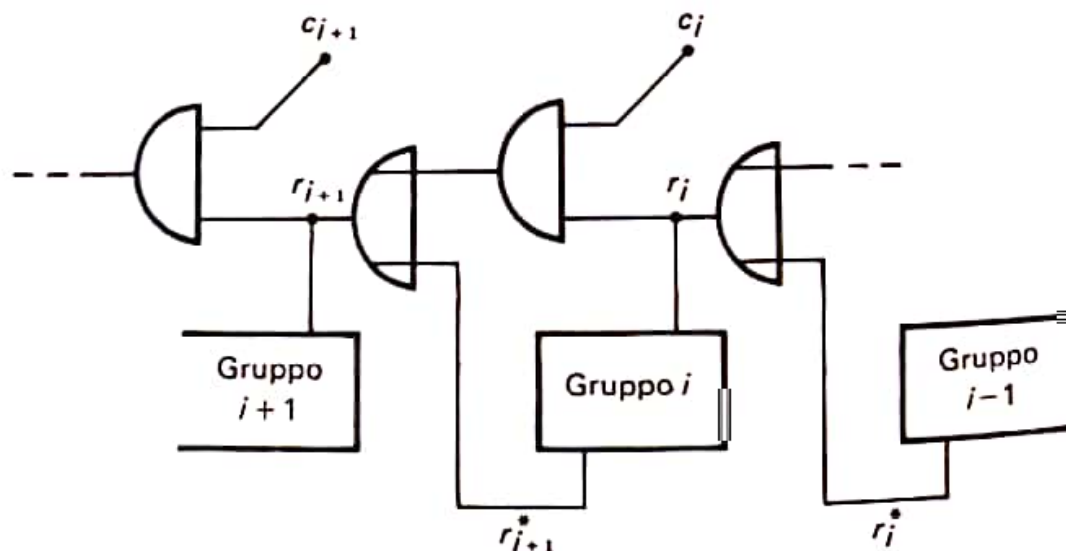


Figura 3.13

Suddivisione di un addizionatore in gruppi di stadi, per accelerare la propagazione del riporto tra gruppi.

dente r_i . Se i due ingressi di uno stadio a_j^i, b_j^i hanno entrambi valore 0, la propagazione del riporto r_i all'interno del gruppo viene ivi arrestata, altrimenti prosegue. Il segnale c_i , quindi, si esprime come:

$$c_i = (a_0^i + b_0^i) \cdot (a_1^i + b_1^i) \cdot \dots \cdot (a_{q-1}^i + b_{q-1}^i).$$

Si ha così la rete di figura 3.14, che mostra in dettaglio i collegamenti all'interno di ogni gruppo. Si noti che il riporto r_{i+1}^* è generato all'interno del gruppo senza attendere l'apporto del segnale precedente r_i ; r_{i+1}^* costituisce anch'esso riporto per il gruppo successivo, e prende parte alla formazione del riporto complessivo r_{i+1} .

Un attento studio degli schemi delle figure 3.13 e 3.14 mostra come il tempo massimo T'' di funzionamento dell'addizionatore sia ora dato dal tempo $q \cdot \Delta_r$ in cui il riporto r_1^* si genera entro il primo gruppo; più il tempo $(p-2) \cdot \Delta_e$ in cui esso raggiunge l'ultimo gruppo sul circuito esterno dei riporti (Δ_e è il ritardo relativo a ciascuno dei $p-2$ gruppi intermedi, dovuto a un blocco OR seguito da un AND, fig. 3.13); più il tempo $(q-1) \cdot \Delta_r + \Delta_s$ in cui tale riporto si propaga entro l'ultimo gruppo. Si ha in conclusione:

$$T'' = (2q-1) \Delta_r + (p-2) \Delta_e + \Delta_s.$$

Anche in questo schema la scelta dei valori di p e q deriverà da un compromesso tra costo della rete, che cresce ora con p a causa del circuito su cui corrono i riporti (se ogni gruppo è realizzato su un singolo circuito integrato, si devono connettere p di questi circuiti), e tempo di funzionamento, che è ora una complessa funzione di p e q (vedi esercizio 3.10).

Abbiamo fino a questo punto considerato che l'addizionatore operi su addendi A, B senza segno. Lo stesso funzionamento genera la somma di numeri positivi in una delle notazioni precedentemente studiate, notando che l'eventuale supero di capacità è ora indicato da $r_{n-1} = 1$ (riporto emergente dalla somma dei moduli): se non vi è supero, lo stadio n -esimo addiziona il bit $r_{n-1} = 0$ con i bit di segno $a_{n-1} = 0, b_{n-1} = 0$, generando il segno $S_{n-1} = 0$ del risultato. L'impiego, importantissimo, dell'addizionatore per numeri relativi sarà oggetto del prossimo paragrafo.

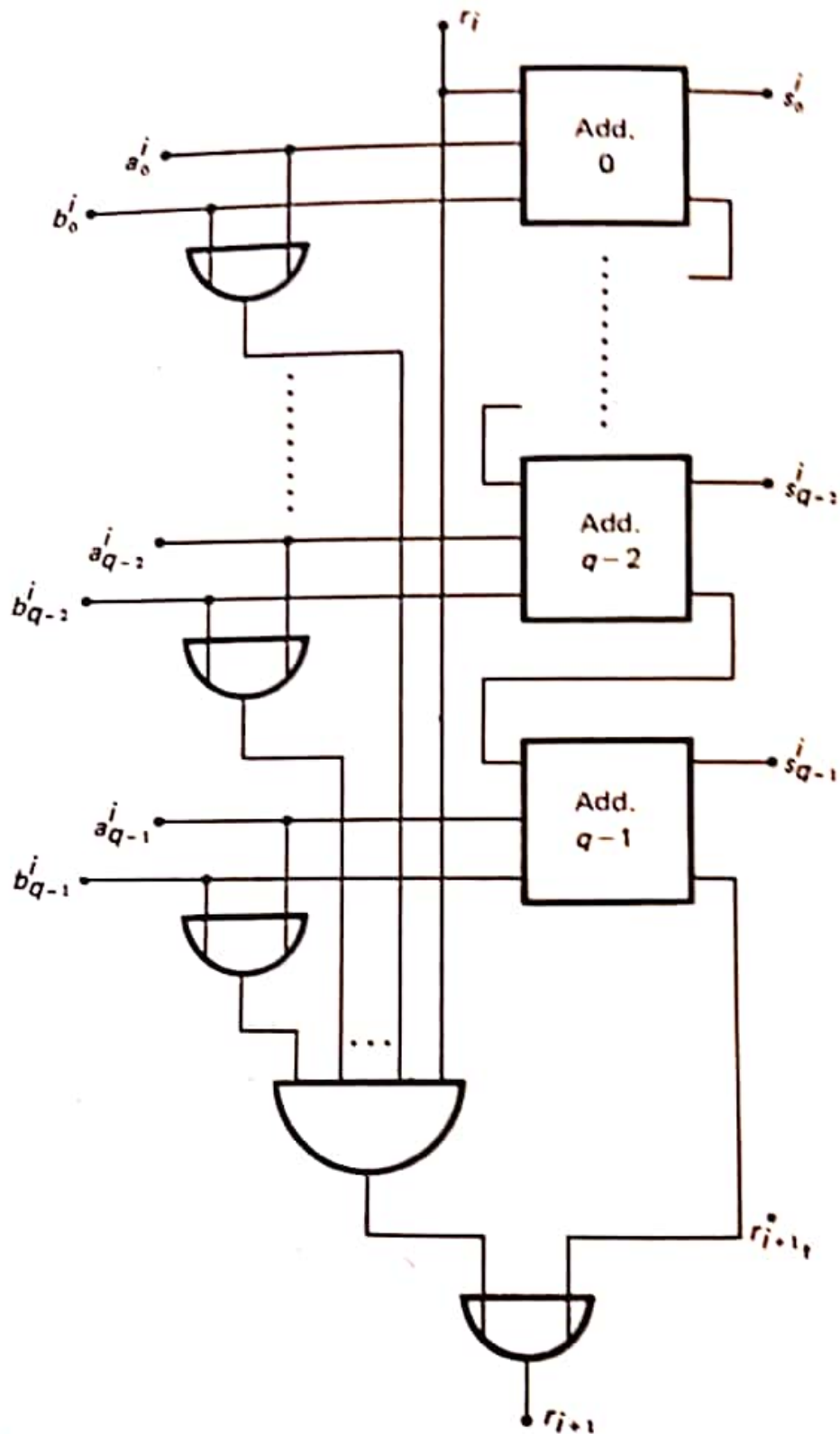
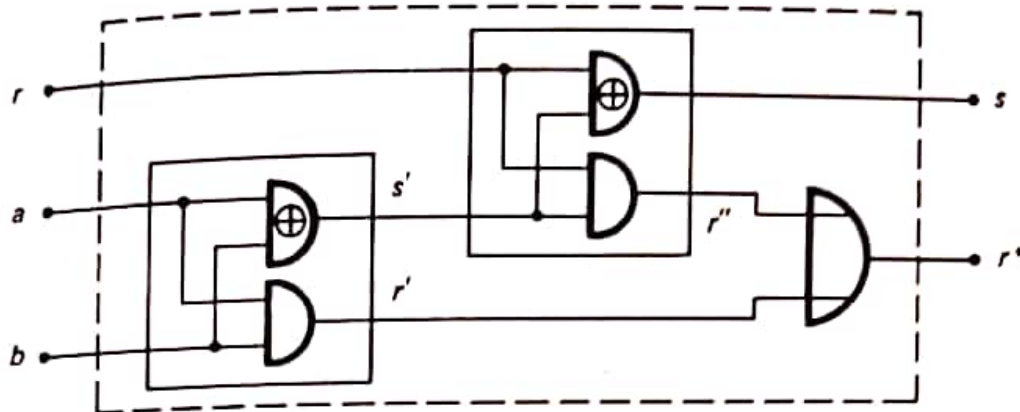


Figura 3.14
 Dettaglio di un gruppo per la rete di figura 3.13.

Esercizi

3.5 Un blocco addizionatore può essere realizzato secondo lo schema:



ove \oplus è l'operatore OR esclusivo ($x \oplus y = 1$ se e solo se $x \neq y$). Discutere lo schema e giustificarlo.

3.6 Calcolare il numero di configurazioni distinte per due addendi di n bit che richiedano il tempo massimo di funzionamento per l'addizionatore parallelo di figura 3.10.

3.7 Costruire un "sottrattore parallelo" per interi senza segno in codice binario naturale. (Problema impegnativo. Studiare preliminarmente il meccanismo del "prestito" nella sottrazione, analogo a quello del riporto nell'addizione.)

3.8 Costruire un addizionatore a due livelli per addendi di due bit.

3.9 Un addizionatore rapido a 15 bit è diviso in 5 gruppi di 3 bit ciascuno. Per i due schemi discussi nel testo, si studi la propagazione del riporto relativo alle addizioni:

```

0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 +
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 1 0 1 1 0 1 1 1 0 0 1 0 1 +
0 0 1 1 0 0 1 0 0 1 0 1 0 1 1

```

3.10 Per l'addizionatore rapido delle figure 3.13, 3.14 si studino i valori di p e q , in funzione di n , che rendono minimo il tempo T'' di funzionamento, immaginando che ogni stadio sia realizzato con la rete di figura 3.12, e i blocchi logici AND, OR, NOT abbiano tutti ritardo unitario. (Cenni alla soluzione. Abbiamo: $\Delta_r = 2$, $\Delta_e = 2$, $\Delta_s = 5$, $pq = n$, e quindi: $T'' = (2q - 1)2 + (p - 2)2 + 5 = 4q + 2p - 1 = 4q + 2n/q - 1$.)

Il minimo di T'' si ottiene ponendo: $dT''/dq = 4 - 2n/q^2 = 0$, da cui: $q = \sqrt{n/2}$, $p = \sqrt{2n}$. Provare questi risultati per diversi valori di n , approssimandoli agli interi più opportuni.)

3.3.3 Addizione e sottrazione tra numeri relativi

Studiato l'addizionatore come rete di base per ottenere la somma di interi positivi, vediamo ora come la stessa rete possa essere impiegata per ottenere la somma di interi *relativi*. Il problema è importantissimo, poiché avendosi $A + (-B) = A - B$, l'addizionatore potrà essere utilizzato anche per eseguire la sottrazione, se il sottraendo vi si presenta con il segno cambiato. E poiché attraverso l'iterazione di addizioni e sottrazioni si ottengono la moltiplicazione e la divisione, la stessa rete potrà essere impiegata per eseguire tutte le operazioni aritmetiche. Questo di fatto accade nei calcolatori, a meno che non siano richieste reti specializzate per eseguire operazioni aritmetiche a grandissima velocità.

Per operare efficacemente tra numeri relativi sono essenziali le notazioni in complemento: nel seguito di questo testo ci limiteremo a studiare le operazioni in complemento a 2, che sono le più frequentemente adottate, e sono comunque facilmente estensibili ad altre notazioni. Vediamo anzitutto alcuni esempi di addizioni tra numeri relativi di quattro bit, eseguite con l'addizionatore:

$$\begin{array}{rcl}
 0101 & (+5) & 0101 & (+5) & 1001 & (-7) \\
 +0000 & (+0) & +1110 & (-2) & +0010 & (+2) \\
 \hline
 =0101 & (+5) & =0011 & (+3) & =1011 & (-5)
 \end{array}$$

$$\begin{array}{rcl}
 0100 & (+4) & 1101 & (-3) \\
 +1100 & (-4) & +1101 & (-3) \\
 \hline
 =0000 & (+0) & =1010 & (-6)
 \end{array}$$

Tutte queste operazioni danno *risultato corretto*, poiché le uscite dell'addizionatore rappresentano la *somma algebrica* tra gli addendi espressa in *complemento a 2* (cioè nella stessa notazione dei dati).

Formalmente ciò si giustifica in modo molto semplice. Detti A , B e R gli addendi e il risultato; S_A , M_A , S_B , M_B e S_R , M_R i loro segni e moduli; e posto che *non vi sia supero di capacità* ($M_R \leq 2^{n-1} - 1$), si ha nei vari casi (controllare sugli esempi):

1) $A \geq 0$, $B \geq 0$: $M_R = M_A + M_B$; $S_R = S_A + S_B = 0 + 0 = 0$ (positivo).

- 2) $A > 0, B < 0, M_A \geq M_B$: $M_R = M_A + 2^{n-1} - M_B = M_A - M_B$ con riporto $r_{n-1} = 1$ verso la posizione del segno (si ricordi che $2^{n-1} = 10 \dots 0$); $S_R = S_A + S_B + r_{n-1} = 0 + 1 + 1 = 0$ con riporto $r_n = 1$ (perduto).
- 3) $A \geq 0, B < 0, M_A < M_B$: $M_R = M_A + 2^{n-1} - M_B = 2^{n-1} - (M_B - M_A)$; $S_R = S_A + S_B = 0 + 1 = 1$ (negativo).
- 4) $A < 0, B < 0$: $M_R = 2^{n-1} - M_A + 2^{n-1} - M_B = 2^{n-1} - (M_A + M_B)$ con riporto $r_{n-1} = 1$; $S_R = S_A + S_B + r_{n-1} = 1 + 1 + 1 = 1$ con riporto $r_n = 1$ (perduto).

Vediamo ora cosa accade nel caso di supero di capacità:

$$\begin{array}{rcl}
 0011 & (+3) & 1100 & (-4) & 1101 & (-3) \\
 +0110 & (+6) & +1010 & (-6) & +1011 & (-5) \\
 \hline
 =1001 & (-7)! & =0110 & (+6)! & =1000 & ?
 \end{array}$$

Quando il modulo del risultato supera $2^{n-1} - 1$, il che può avvenire se $A > 0, B > 0$ oppure $A < 0, B < 0$, l'addizionatore dà *risultato non corretto* (né d'altronde sarebbe possibile rappresentare il risultato giusto con n bit). Formalmente:

1) $A > 0, B > 0, M_A + M_B = 2^{n-1} + K$ con $K \geq 0$: $M_R = M_A + M_B = K$ (errato) con supero $r_{n-1} = 1$; $S_R = S_A + S_B + r_{n-1} = 0 + 0 + 1 = 1$ (errato).

2) $A < 0, B < 0, M_A + M_B = 2^{n-1} + K$ con $K > 0$: $M_R = 2^{n-1} - M_A + 2^{n-1} - M_B = 2^{n-1} - K$ (errato); $S_R = S_A + S_B = 1 + 1 = 0$ (errato) con riporto $r_n = 1$ (perduto).

3) $A < 0, B < 0, M_A + M_B = 2^{n-1}$: $M_R = 2^{n-1} - M_A + 2^{n-1} - M_B = 0$ con riporto $r_{n-1} = 1$; $S_R = S_A + S_B + r_{n-1} = 1 + 1 + 1 = 1$ (esatto) con riporto $r_n = 1$ (perduto).

Il caso 3) dà luogo al risultato $10 \dots 0$, che non esiste tra le configurazioni del complemento a 2 (vedi tab. 3.1), e può quindi essere aggiunto per indicare il numero -2^{n-1} (di cui non è però possibile formare l'opposto). Si veda il precedente esempio: $(-3) + (-5)$. Incluso questo caso tra i risultati corretti, vediamo che una regola (tra varie possibili) per individuare il supero di capacità è la seguente:

Regola Nell'addizione tra numeri relativi in complemento a 2 si ha supero di capacità se e solo se gli addendi hanno lo stesso segno e il risultato ha segno opposto degli addendi.

Una rete per individuare il supero di capacità ($sup=1$) è indicata nella figura 3.15a. Essa può essere inclusa come pezzo standard dell'addizionatore, in luogo del circuito di riporto r_n indicato nella figura 3.10 (utilizzabile solo per interi positivi).

La rete della figura 3.15, ove l'addizionatore è realizzato in una qualunque delle forme viste nel sottoparagrafo 3.3.2, permette di ese-

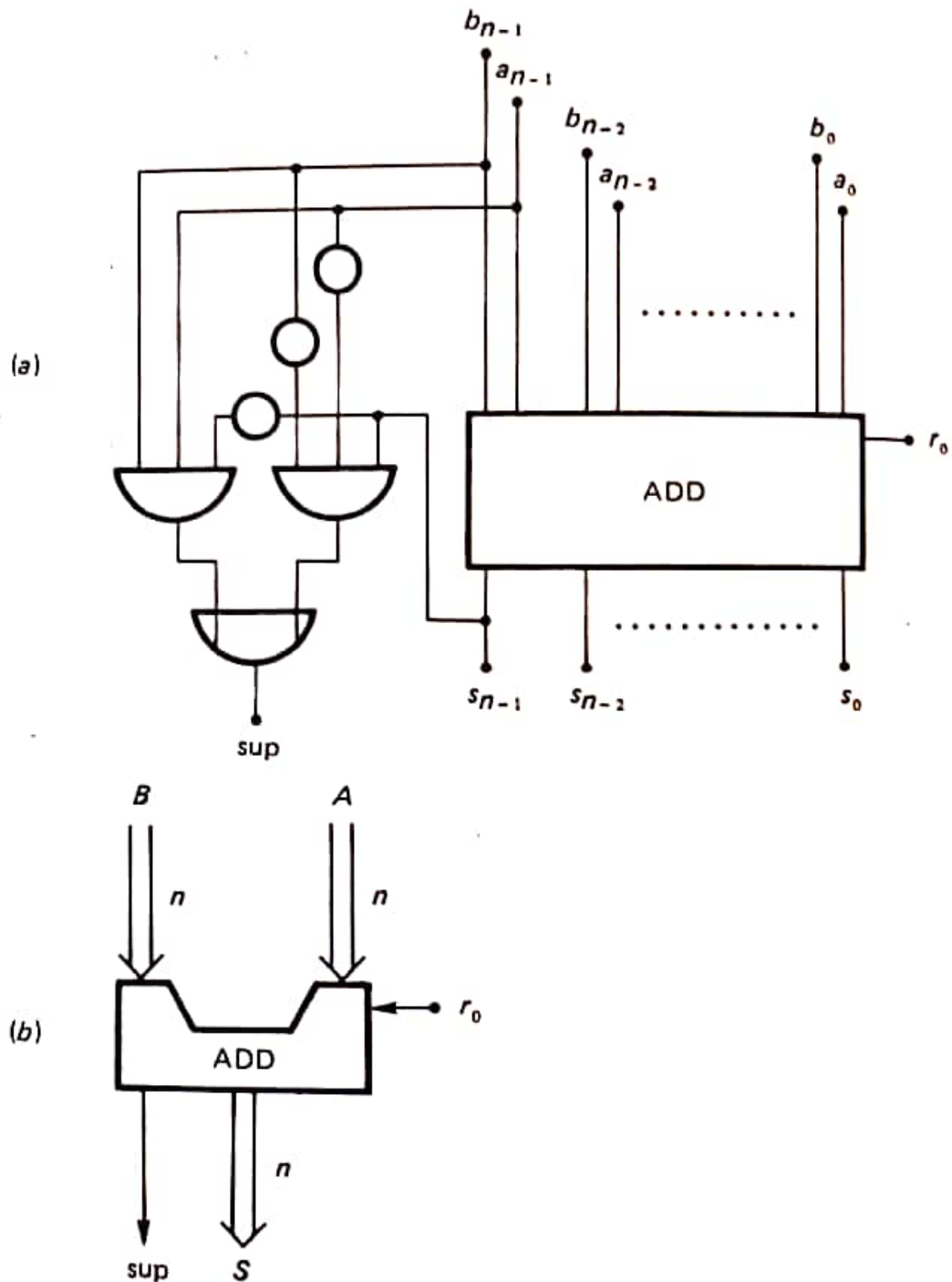


Figura 3.15

(a) Addizionatore con individuazione del supero di capacità in complemento a 2 ($sup=1$ indica supero). (b) Simbolo dell'unità.

guire l'addizione tra due interi relativi A e B (rappresentati in complemento a 2), e dà sempre risultato corretto se $sup = 0$. Ovviamente, se si cambia preventivamente il segno a uno degli addendi, si ottiene la sottrazione tra essi. Utilizzando la regola per la costruzione dell'opposto riportata nel sottoparagrafo 3.3.1 (punto 3): $-B = \bar{B} + 1$, ove \bar{B} indica la stringa di B con i bit complementati), la rete si può trasformare nella nuova struttura di figura 3.16, che esegue addizione e sottrazione tra numeri relativi: un selettore di ingresso comandato dalla variabile x_0 controlla il passaggio di B , o \bar{B} , verso l'addizionatore; il segnale $r_0 = 1$ fornisce un riporto al primo stadio dell'addizionatore, che somma così 1 al risultato.

Estenderemo ora ulteriormente questa struttura, per eseguire altre operazioni.

3.3.4 Una unità aritmetica-logica

Anche se le operazioni di addizione e sottrazione sono basilari, altre operazioni, seppur semplicissime, sono altrettanto importanti nel fun-

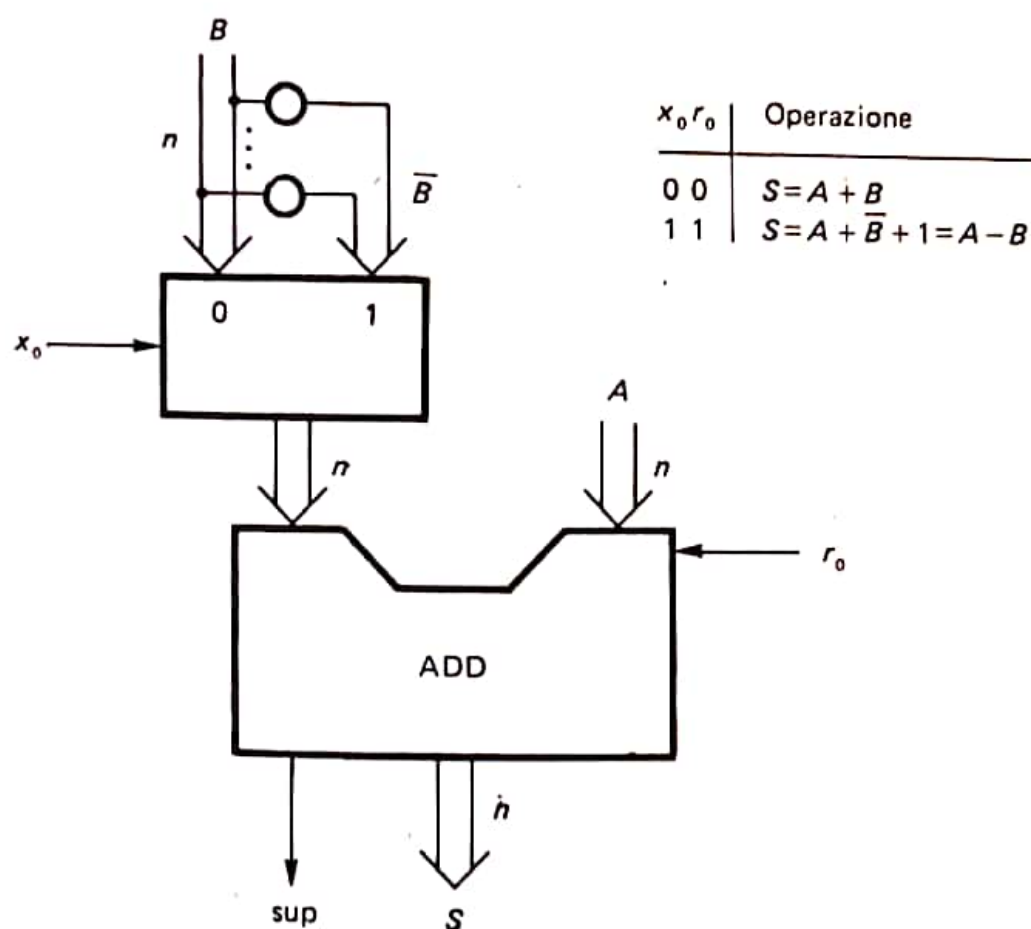
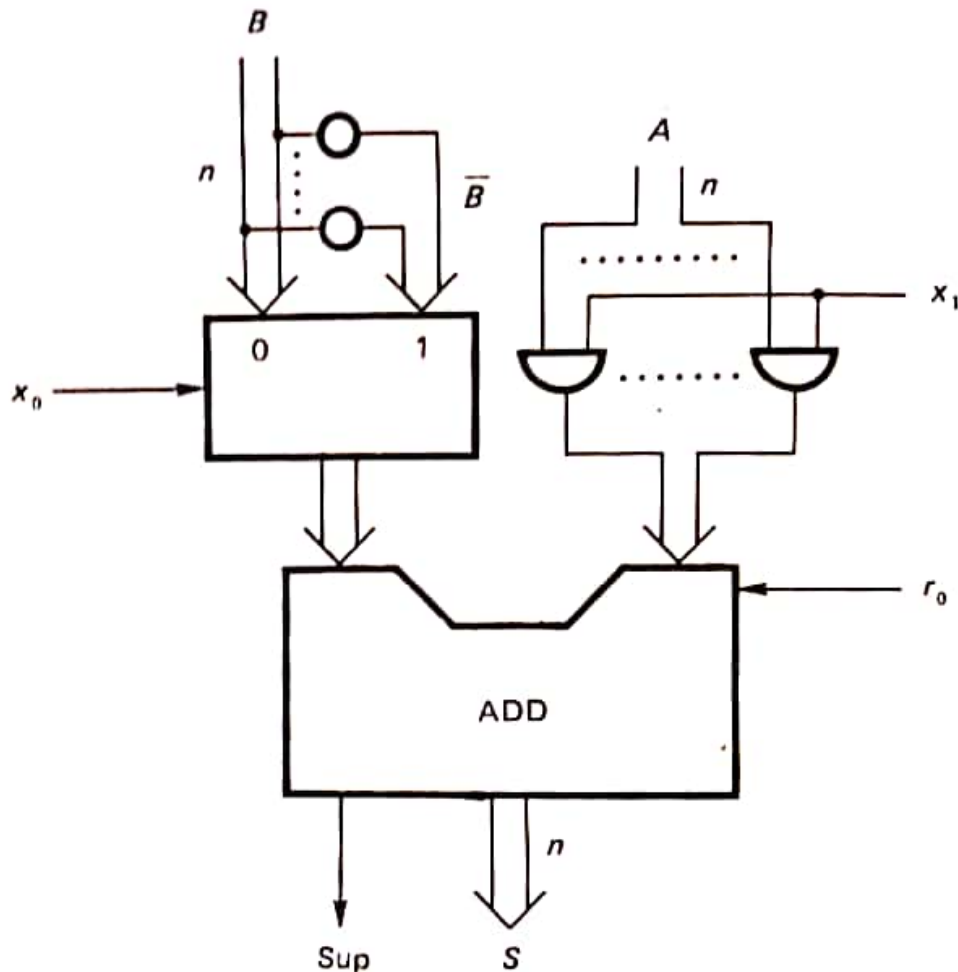


Figura 3.16

Rete che esegue l'addizione e la sottrazione tra numeri relativi.

zionamento dei sistemi. Come infatti vedremo, questi includono una *unità aritmetica-logica* (o *ALU, Arithmetic-Logic Unit*) che esegue un insieme di operazioni di base: ne studiamo qui un esempio, proponendo come esercizio lo sviluppo di altre possibili unità.

Anzitutto richiediamo all'unità di poter operare su un solo dato, oltre che su due. Poiché il cuore dell'unità è un addizionatore, dovremo far giungere al posto di uno dei dati una configurazione costante, tipi-



$x_1 x_0 r_0$	Operazione
0 0 0	$S = 0 + B = B$
0 0 1	$S = 0 + \overline{B} + 1 = B + 1$
0 1 0	$S = 0 + \overline{B} = \overline{B}$
0 1 1	$S = 0 + \overline{B} + 1 = -B$
1 0 0	$S = A + B$
1 0 1	$S = A + B + 1$
1 1 0	$S = A + \overline{B} = A - B - 1$
1 1 1	$S = A + \overline{B} + 1 = A - B$

Figura 3.17
ALU derivata dalla rete di figura 3.16.

camente la stringa 00 ... 0 (intero + 0). Possiamo per esempio bloccare l'arrivo di A con uno strato di blocchi AND controllati da una nuova variabile x_1 , secondo lo schema di figura 3.17.

Questa rete è già in grado di compiere otto operazioni differenti, se si controllano indipendentemente le variabili x_1, x_0, r_0 ; di queste operazioni solo due (indicate in parentesi nella figura) possono essere considerate poco interessanti. E' invece interessante, come vedremo in seguito, la semplice operazione $S=B$, in cui la rete funziona da selettore per B .

La semplice ALU della figura 3.17 esegue operazioni *aritmetiche*, tranne la $S=\bar{B}$ che si dice *logica*, perché trasforma i singoli bit di B indipendentemente l'uno dall'altro. Similmente un'operazione logica eseguita su due parole, $S=A \text{ op } B$, genera indipendentemente i bit $s_i = a_i \text{ op } b_i$. Esempi tipici di operazioni logiche tra parole sono l'AND e l'OR, per esempio:

$$A = 1\ 1\ 0\ 1\ 0, \quad B = 1\ 0\ 0\ 1\ 1;$$

$$A \text{ AND } B = 1\ 0\ 0\ 1\ 0, \quad A \text{ OR } B = 1\ 1\ 0\ 1\ 1.$$

Se la ALU è costruita attorno a un addizionatore, le operazioni logiche tra le parole di ingresso possono essere eseguite solo annullando l'effetto dei riporti, per rendere le operazioni sui bit indipendenti tra loro. Ciò si ottiene forzando un valore costante (0 o 1) su tutti i riporti (compreso r_0): si deve cioè intervenire sulla struttura interna dell'addizionatore.

Poniamo che nella ALU della figura 3.17 ogni stadio di addizione sia realizzato con lo schema della figura 3.12: gli stadi possono essere connessi tra loro in semplice cascata (fig. 3.10), o con qualsiasi schema di propagazione rapida, purché le reti che realizzano i riporti siano tutte accessibili. Per esempio due nuovi segnali di controllo l_1, l_0 possono essere inviati al circuito del riporto della figura 3.12, nei punti indicati nel nuovo schema della figura 3.18.

Le funzioni realizzate dai blocchi addizionatori ($0 \leq i \leq n-1$) sono ora:

$$r_{i+1} = l_1 + l_0 b_i a_i + b_i r_i + a_i r_i;$$

$$s_i = \bar{r}_{i+1} (a_i + b_i + r_i) + a_i b_i r_i.$$

I valori $l_1 = 0, l_0 = 1$ non alterano il vecchio valore di r_{i+1} e in conseguenza di s_i : in questo modo l'addizionatore, e quindi la ALU, funzionano come discusso in precedenza.

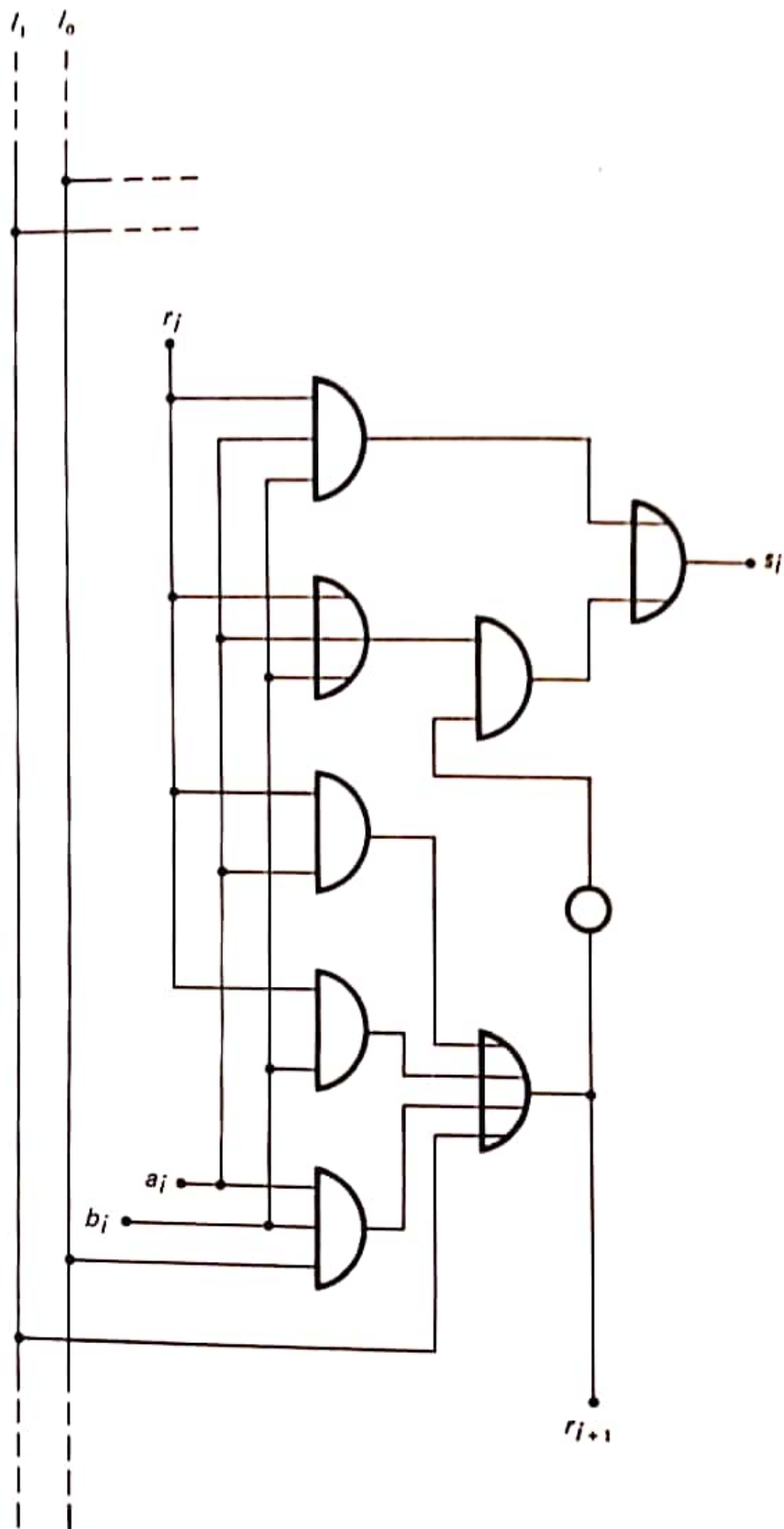


Figura 3.18
Segnali di controllo per operazioni logiche.

Ponendo invece $l_1 = 1$, si ha:

$$r_{i+1} = 1, \quad 0 \leq i \leq n-1,$$

cioè si distrugge l'informazione trasmessa tra gli stadi dai riporti interni all'addizionatore; ponendo inoltre $r_0 = 1$, si ha:

$$s_i = \overline{1}(a_i + b_i + 1) + a_i b_i 1 = a_i b_i, \quad 0 \leq i \leq n-1,$$

e si realizza così l'AND tra i singoli bit delle parole A e B . (In questo caso il valore di l_0 è influente).

Ponendo $l_1 = 0$, $l_0 = 0$ e $r_0 = 0$ si ha:

$$r_{i+1} = 0, \quad 0 \leq i \leq n-1,$$

quindi

$$s_i = \overline{0}(a_i + b_i + 0) + a_i b_i 0 = a_i + b_i, \quad 0 \leq i \leq n-1,$$

e si realizza così l'OR tra i bit di A e B .

Arricchendo la ALU di figura 3.17 con i collegamenti di l_1 , l_0 , si ottiene la nuova ALU di figura 3.19 per cui indichiamo le operazioni più importanti.

Poiché le otto operazioni indicate possono essere enumerate con tre segnali c_2 , c_1 , c_0 , è spesso opportuno far circolare per il sistema solo questi, includendo nella ALU una rete combinatoria che costruisca i segnali l_1 , l_0 , x_1 , x_0 , r_0 come funzioni di c_2 , c_1 , c_0 .

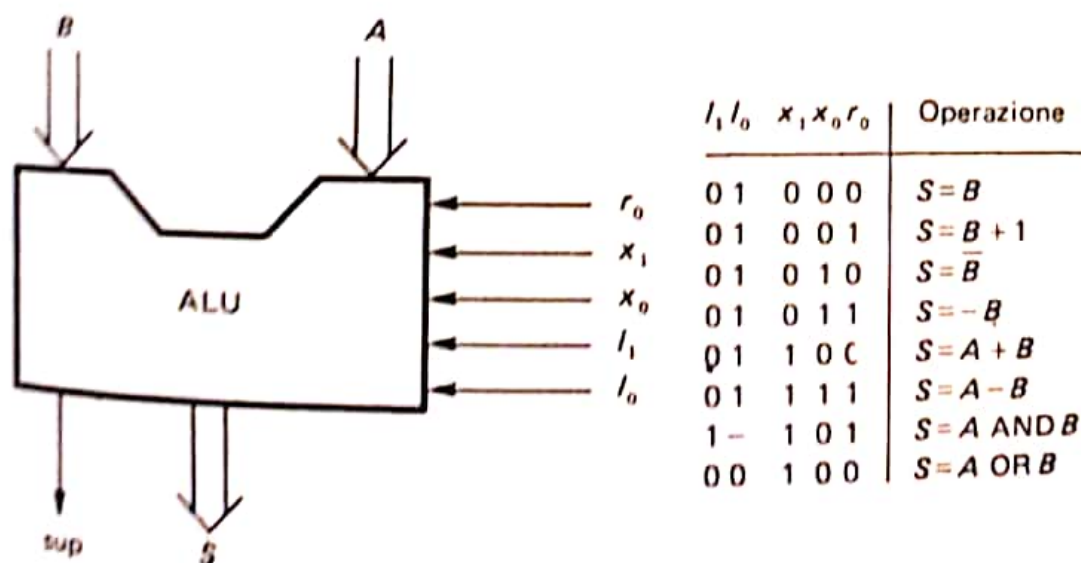


Figura 3.19

ALU derivante dalla fusione degli schemi delle figure 3.17 e 3.18.

