Parte xxv

Circuiti logici

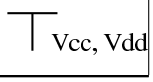
Operatori logici e porte logiche

La logica formale studia le proposizioni dichiarative, dove per proposizione si intende l'insieme di soggetto e verbo. È una ***proposizione dichiarativa*** quella proposizione nei confronti della quale è possibile stabilire se è vera o se è falsa. *Vero* o *Falso* sono gli unici valori che può assumere una proposizione dichiarativa. La proposizione che non si può suddividere in altre proposizioni, si dice essere elementare.

Il valore di una proposizione dichiarativa (*Vero* o *Falso*) può essere espresso in vari modi, a seconda del contesto. Generalmente, si attribuisce alla cifra numerica uno il significato di *Vero*, mentre a zero

«a2» 2013.11.11 --- Copyright © Daniele Giacomini -- appunti2@gmail.com http://informaticalibera.net

si attribuisce il valore *Falso*, così come nell'applicazione elettronica si utilizza un livello di tensione vicino a quello di alimentazione per rappresentare *Vero* e un valore di tensione vicino a zero per rappresentare *Falso*.

Falso	0	0 V	
Vero	1	tensione di alimentazione	

La variabile che può assumere solo il valore risultante da una proposizione dichiarativa, è una ***variabile logica***. Un'***espressione logica*** o una ***funzione logica*** è quella che produce un risultato *Vero* o *Falso*. L'espressione o funzione logica può essere costituita da proposizioni dichiarative, da valori costanti (espressi secondo la forma prevista per rappresentare *Vero* o *Falso*) e da variabili logiche. Per connettere o comunque per intervenire nei valori delle varie componenti dell'espressione, si utilizzano degli operatori che rappresentano delle funzioni elementari.

Si distinguono generalmente gli operatori logici in «unari» e in «connettivi logici», per distinguere se intervengono in un solo valore logico, oppure su due o più valori logici. Gli operatori logici si possono vedere come delle scatoline, aventi uno o più ingressi, ma con una sola uscita: in tal caso, si chiamano ***porte logiche***.

Gli operatori logici hanno forme differenti di rappresentazione, in base al contesto e in base alle limitazioni tipografiche a cui si deve sottostare. Pertanto, ogni volta che si legge un documento che tratta questo genere di argomenti, è necessario inizialmente comprendere qual è la simbologia adottata, tenendo conto che anche all'interno dello stesso testo si possono alternare simbologie differenti.

Tabella u97.2. Alcune notazioni alternative degli operatori logici comuni, associate alla simbologia delle porte logiche corrispondenti.

a	a	a	a	a — a
a	NOT a	a’	$\overline{a}$	a — $\overline{a}$
a b	a AND b	a · b	a ∧ b	a, b — a ∧ b
a b	a OR b	a + b	a ∨ b	a, b — a ∨ b
a b	a XOR b		a ⊕ b	a, b — a ⊕ b

Per definire il significato degli operatori logici si utilizzano delle tabelle di verità, mettendo a confronto le variabili in ingresso con l’esito finale.

Operatori unari

Gli operatori logici unari ottengono in ingresso un solo valore logico; sono disponibili l’invertitore logico (NOT) e il non-invertitore logico.

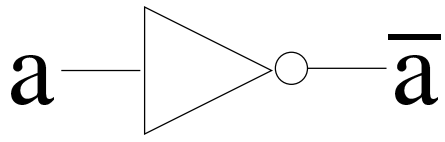
L’invertitore logico è l’operatore unario che inverte il valore logico ricevuto in ingresso: se in ingresso riceve il valore *Vero* (1), in uscita genera il valore *Falso* (0); se in ingresso riceve il valore *Falso* (0), in uscita genera il valore *Vero* (1).

A titolo di esempio, se la variabile logica «A» contiene il risultato della proposizione dichiarativa «Antonio mangia», l’espressione lo-

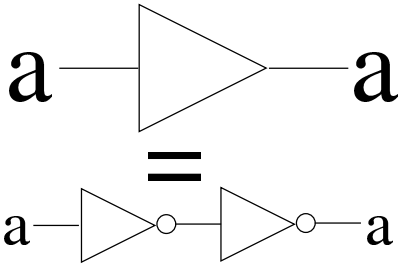
gica «NOT A» è equivalente alla proposizione dichiarativa «Antonio non mangia».

Il non-invertitore logico è l'operatore unario che presenta in uscita lo stesso valore ricevuto in ingresso. Il nome che viene dato a questo tipo di operatore allude alla presenza ipotetica di due negazioni consecutive che si eliminano a vicenda. Per esempio, se la variabile logica «A» contiene il risultato della proposizione dichiarativa «Antonio mangia», l'espressione logica «NOT A» è equivalente alla proposizione dichiarativa «Antonio non mangia», ma nello stesso modo, «NOT (NOT A)» è equivalente alla proposizione originale: «Antonio mangia».

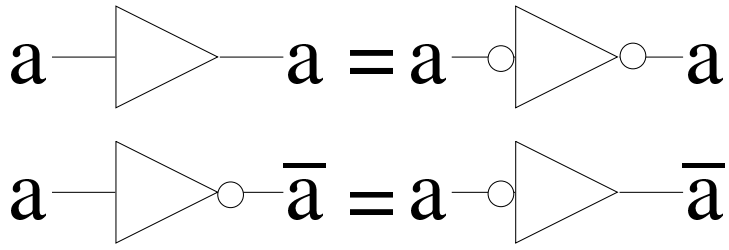
a	$\overline{a}$
0	1
1	0



a	a
0	0
1	1



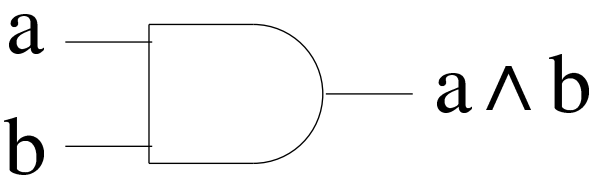
Dagli esempi mostrati si dovrebbe intendere il fatto che un piccolo cerchio rappresenta un'inversione logica, e si può collocare all'ingresso o all'uscita di una funzione logica rappresentata graficamente. Pertanto, valgono le equivalenze seguenti:



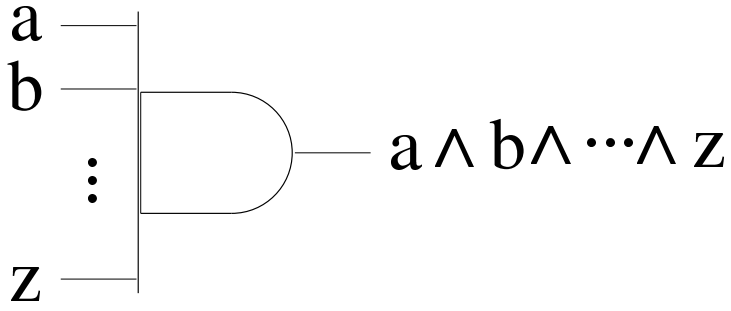
Connettivo AND

I connettivi logici sono gli operatori che utilizzano due ingressi. Il connettivo AND restituisce il valore *Vero* solo se entrambi i valori in ingresso sono pari a *Vero*. Per esempio, se la variabile logica «A» contiene il risultato della proposizione dichiarativa «Antonio mangia» e la variabile «B» contiene il risultato di «Piero legge», l'espressione «A AND B» equivale alla proposizione «Antonio mangia e Piero legge».

a	b	a ∧ b
0	0	0
0	1	0
1	0	0
1	1	1



La porta logica AND, quando ha più di due ingressi, esprime l'equivalente di un'espressione in cui tutte le variabili in ingresso sono collegate dall'operatore AND:

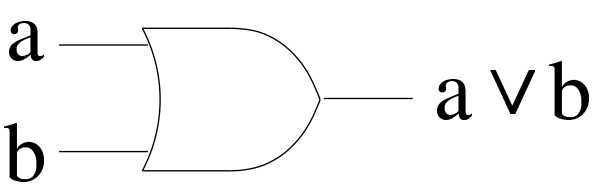


Connettivo OR

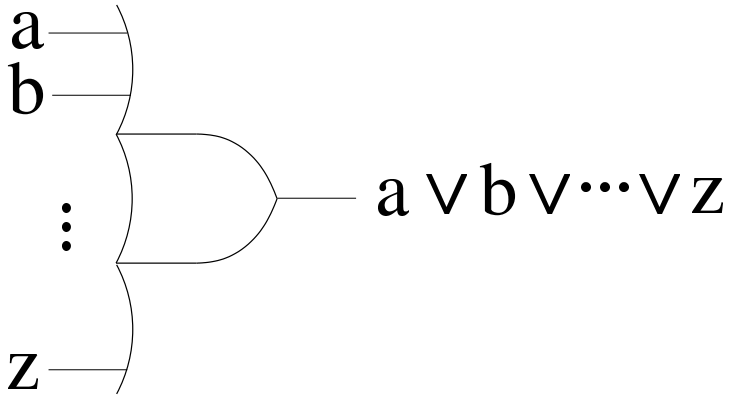
Il connettivo OR restituisce il valore *Vero* se almeno uno dei due ingressi dispone di un valore pari a *Vero*. Per esempio, se la variabile logica «A» contiene il risultato della proposizione dichiarativa «Antonio mangia» e la variabile «B» contiene il risultato di «Piero leg-

ge», l'espressione «A OR B» equivale alla proposizione «Antonio mangia e/o Piero legge».

a	b	a ∨ b
0	0	0
0	1	1
1	0	1
1	1	1



La porta logica AND, quando ha più di due ingressi, esprime l'equivalente di un'espressione in cui tutte le variabili in ingresso sono collegate dall'operatore OR:

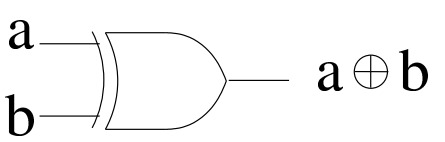


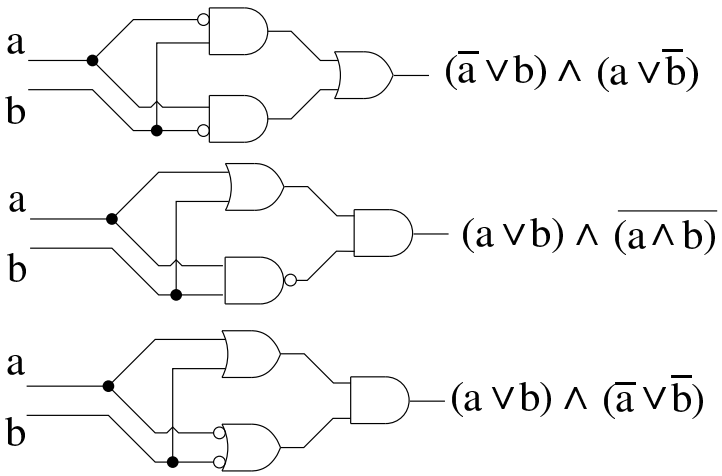
Connettivo XOR

«

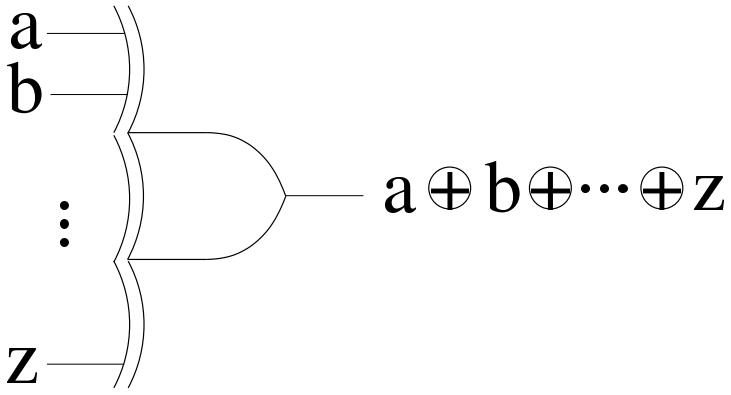
Il connettivo XOR restituisce il valore *Vero* se solo uno dei due ingressi dispone di un valore pari a *Vero*. Per esempio, se la variabile logica «A» contiene il risultato della proposizione dichiarativa «Antonio mangia» e la variabile «B» contiene il risultato di «Piero legge», l'espressione «A XOR B» equivale alla proposizione «Antonio mangia oppure Piero legge». Va comunque osservato che il connettivo XOR si considera derivato dagli altri e può essere tradotto in forme diverse, ma equivalenti.

a	b	a⊕b
0	0	0
0	1	1
1	0	1
1	1	0





Anche la porta logica XOR può avere più ingressi, come avviene per AND e OR:



Ordine di precedenza

Le espressioni logiche vanno risolte tenendo conto che gli operatori hanno un ordine di precedenza: NOT, AND, OR. Per esempio, ***a***+***b***·***c'*** va inteso come ***a***+(***b***·(***c'***)). Naturalmente, l'ordine di precedenza può essere modificato utilizzando opportunamente le parentesi.

Valori irrilevanti nelle tabelle di verità

A volte si possono semplificare le tabelle di verità di una funzione logica quanto alcune variabili in ingresso, in certe circostanze, possono assumere qualsiasi valore senza interferire nel risultato; in quei

casi, al posto del valore si mette convenzionalmente una lettera «x». A titolo di esempio, si consideri la funzione logica che si ottiene dall'espressione $\boldsymbol{A}\cdot(\boldsymbol{B}+\boldsymbol{C})$. Nella figura successiva si vede la tabella di verità completa e a fianco una versione semplificata della stessa.

Figura u97.11. Semplificazione delle tabelle di verità in presenza di condizioni in cui i valori di certe variabili diventano irrilevanti.

A	B	C	A(B+C)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

=

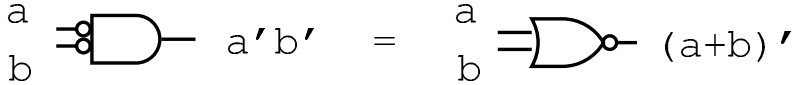
A	B	C	A(B+C)
0	**x**	**x**	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Equivalenze

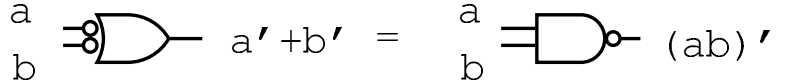
«

Attraverso gli operatori logici si possono costruire delle espressioni complesse, nelle quali esiste la facoltà di applicare delle trasformazioni, di solito con lo scopo di cercare di semplificarle, mantenendo però lo stesso risultato. Per questo sono molto importanti i teoremi di De Morgan, secondo i quali:

$a' \cdot b' = (a+b)'$



$a' + b' = (a \cdot b)'$

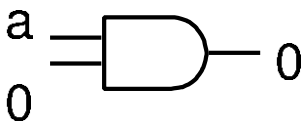


Mettendo assieme le tabelle della verità dei vari operatori logici con i teoremi di De Morgan, si ottengono le equivalenze della tabella successiva su cui si basa l'algebra «logica», ovvero l'algebra di Boole.

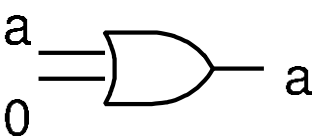
Tabella u97.13. Equivalenze dell'algebra di Boole. Si intende che ab rappresenti semplicemente $a \cdot b$ in modo compatto.

(1)	$a+0 = a$	(2)	$a \cdot 0 = 0$
(3)	$a+1 = 1$	(4)	$a \cdot 1 = a$
(5)	$a+a = a$	(6)	$a \cdot a = a$
(7)	$a+a' = 1$	(8)	$a \cdot a' = 0$
(9)	$a+b = b+a$	(10)	$ab = ba$
(11)	$a+(b+c) = (a+b)+c$	(12)	$a(bc) = (ab)c$
(13)	$a(b+c) = ab+ac$	(14)	$a+bc = (a+b)(a+c)$
(15)	$(a+b)' = a'b'$	(16)	$(ab)' = a'+b'$
(17)	$(a')' = a$		

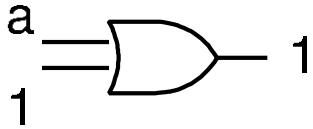
(1) a·0=0



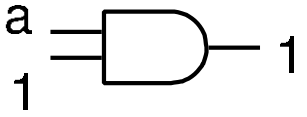
(2) a+0=a



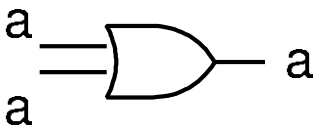
(3) a+1=1



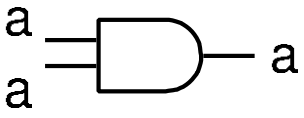
(4) a·1=1



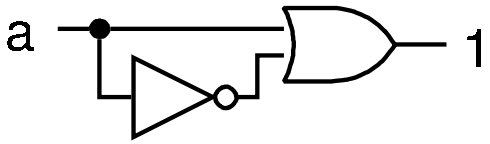
(5) a+a=a

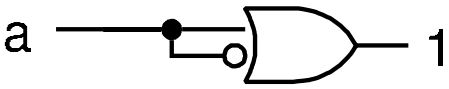


(6) a·a=a

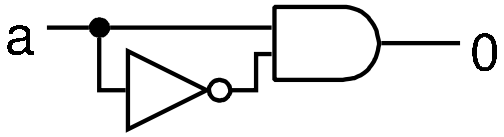


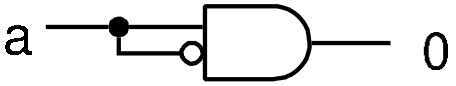
(7) a+a'=1



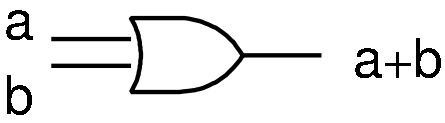


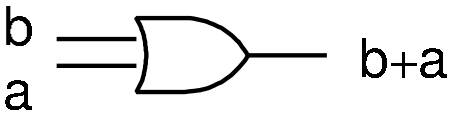
(8) a·a'=0



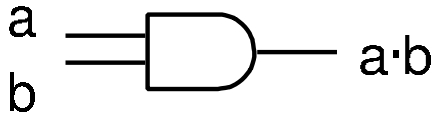


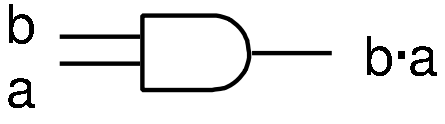
(9) a+b=b+a



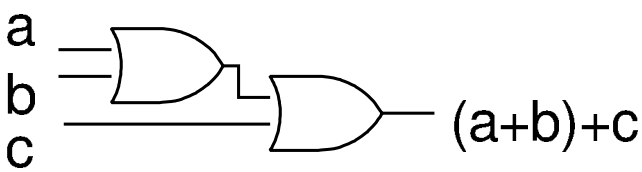


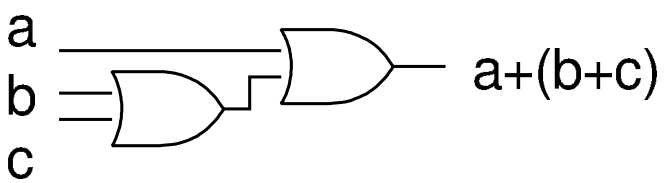
(10) a·b=b·a

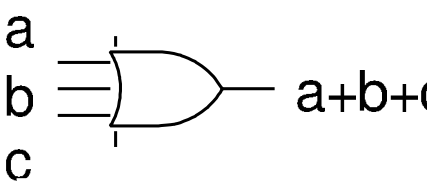




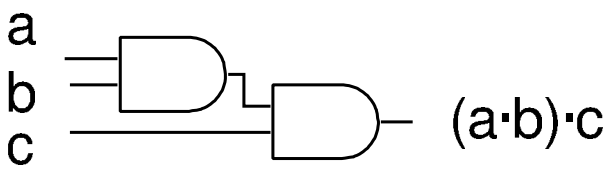
(11) a+(b+c)=(a+b)+c

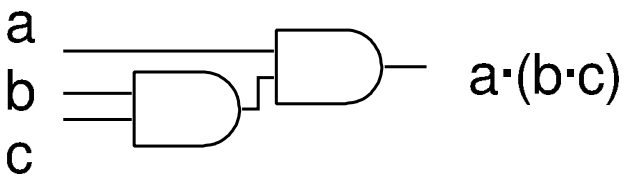


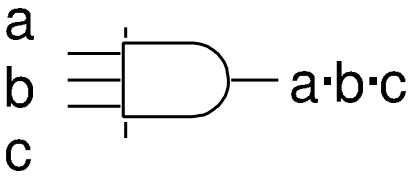




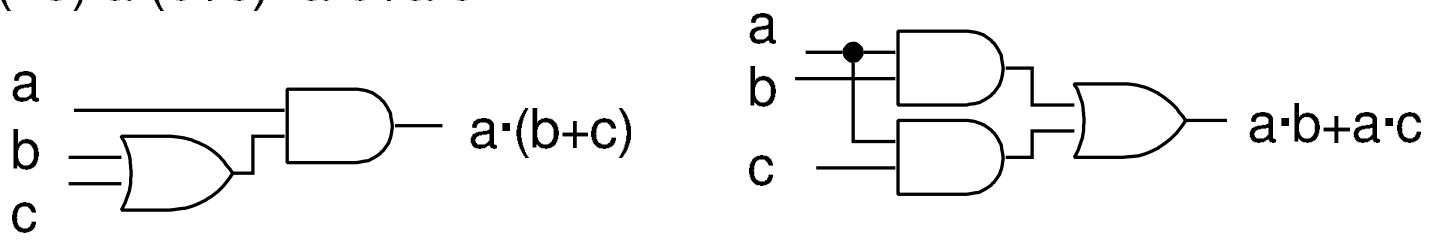
(12) a·(b·c)=(a·b)·c



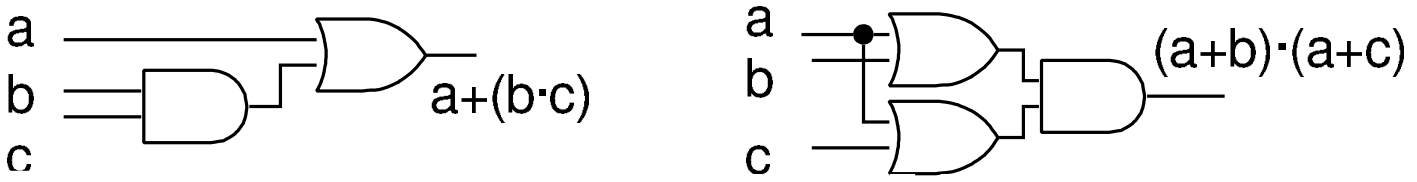




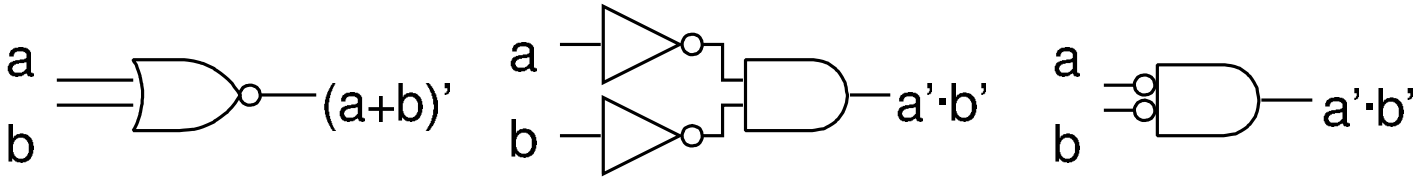
(13) a·(b+c)=a·b+a·c



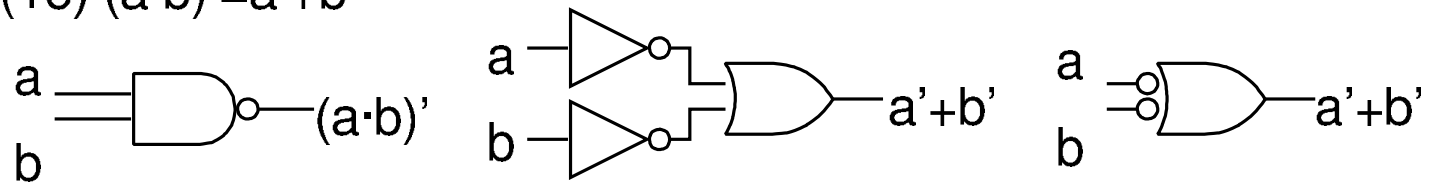
(14) a+(b·c)=(a+b)·(a+c)



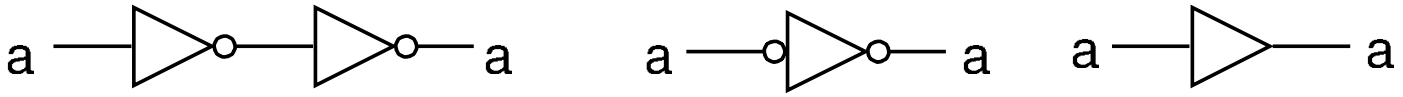
(15) (a+b)'=a'·b'



(16) (a·b)'=a'+b'



(17) (a')'=a



Quando nelle espressioni si utilizzano solo gli operatori logici tradizionali (AND, OR, NOT), i teoremi di De Morgan consentono di trasformare facilmente una funzione logica nel suo complemento. In pratica, data la funzione ***f***, si ottiene la negazione logica ***f***' seguendo la procedura seguente:

- si scambiano gli operatori AND originali con gli operatori OR, mettendo delle parentesi per non modificare l'ordine di valutazione;
- si scambiano gli operatori OR originali con gli operatori AND, facendo in modo di evitare che questo cambiamento cambi l'ordine di esecuzione della valutazione;
- si complementano tutte le variabili (operatore NOT).

Si veda l'esempio seguente:

$$f = a+b'c+de'+g'h'$$
$$f' = a'(b+c')(d'+e)(g+h)$$

Somma dei prodotti

«

Una funzione logica è un'espressione composta da variabili, costanti logiche e operatori logici, la quale produce un risultato logico (*Vero* o *Falso*). Tuttavia, una funzione logica può essere descritta semplicemente attraverso la tabella di verità che abbina i valori delle variabili al risultato atteso per ogni combinazione delle stesse.

È possibile sintetizzare il comportamento di una funzione logica attraverso ciò che si definisce come ***somma dei prodotti***: si parte dalla tabella di verità che si vuole ottenere e si elencano i ***prodotti fondamentali*** logici che descrivono ogni condizione degli ingressi.

Tabella u97.14. Prodotti fondamentali delle funzioni aventi due, tre e quattro variabili.

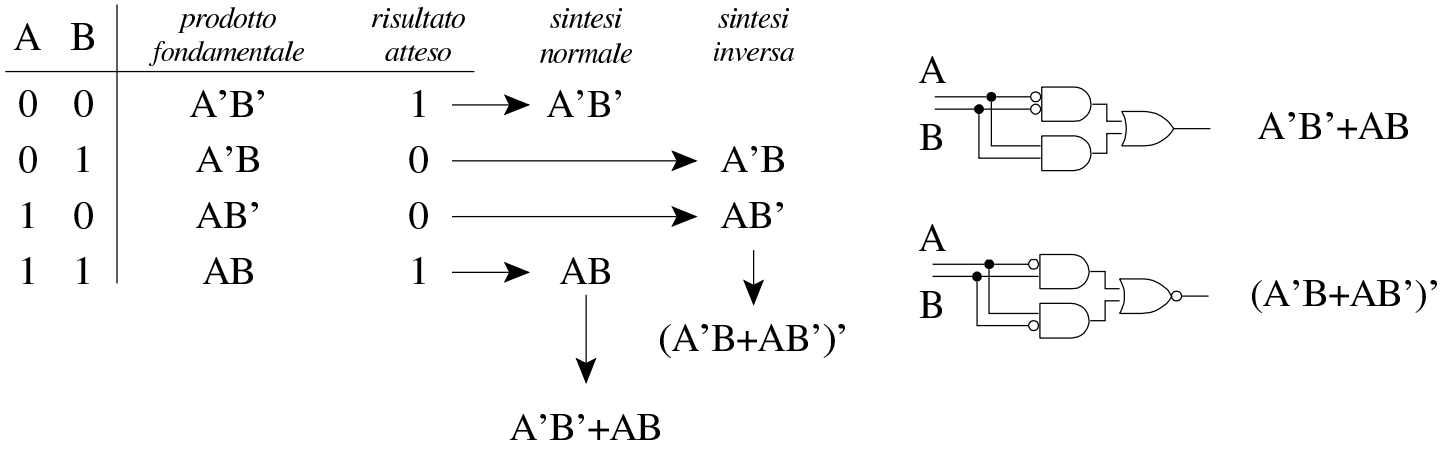
A	B	*prodotto fondamentale*
0	0	A’B’
0	1	A’B
1	0	AB’
1	1	AB

A	B	C	*prodotto fondamentale*
0	0	0	A’B’C’
0	0	1	A’B’C
0	1	0	A’BC’
0	1	1	A’BC
1	0	0	AB’C’
1	0	1	AB’C
1	1	0	ABC’
1	1	1	ABC

A	B	C	D	*prodotto fondamentale*
0	0	0	0	A’B’C’D’
0	0	0	1	A’B’C’D
0	0	1	0	A’B’CD’
0	0	1	1	A’B’CD
0	1	0	0	A’BC’D’
0	1	0	1	A’BC’D
0	1	1	0	A’BCD’
0	1	1	1	A’BCD
1	0	0	0	AB’C’D’
1	0	0	1	AB’C’D
1	0	1	0	AB’CD’
1	0	1	1	AB’CD
1	1	0	0	ABC’D’
1	1	0	1	ABC’D
1	1	1	0	ABCD’
1	1	1	1	ABCD

Per esempio, la funzione nota come NXOR (funzione di due variabili che si avvera solo quando le due variabili hanno lo stesso valore) si sintetizza secondo il procedimento che si può comprendere intuitivamente dalla figura successiva.

Figura u97.15. Sintesi a partire dalla tabella di verità della funzione NXOR, attraverso gli operatori logici fondamentali.



L'esempio della figura permette di ottenere due alternative, secondo quello che è noto come «somma dei prodotti», ma ciò non toglie che ci possano essere altre possibilità di sintesi.

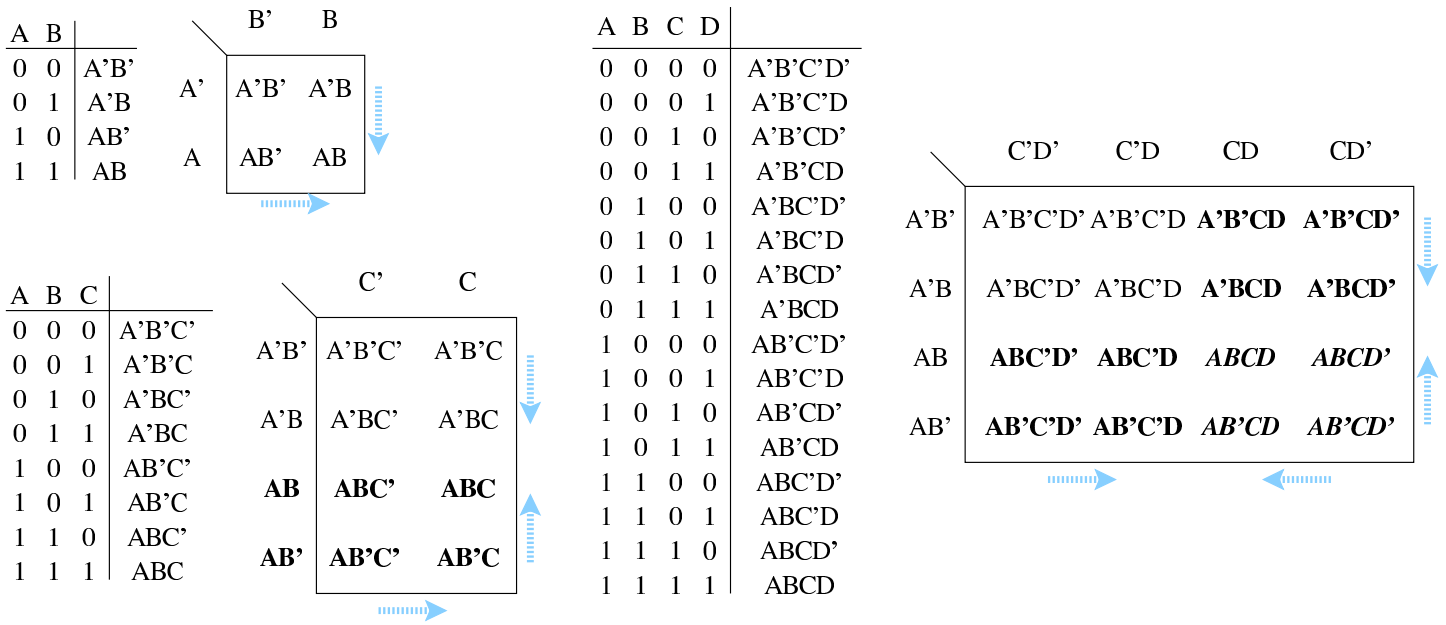
Mappe di Karnaugh

«

Una funzione logica può essere rappresentato in una mappa di Karnaugh, a partire dalla sua tabella di verità; attraverso tale mappa si può poi cercare un modo per semplificare l'espressione che sintetizza la funzione e di conseguenza il circuito che la realizza.

La mappa di Karnaugh è una rappresentazione bidimensionale dei prodotti fondamentali di una tabella di verità da analizzare: in pratica si dividono a metà i prodotti fondamentali e si rappresentano in un rettangolo. Nelle immagini successive si mostrano tre mappe, vuote, relative al caso di due, tre e quattro variabili: bisogna fare molta attenzione alla sequenza dei prodotti fondamentali, perché questa non corrisponde a quella che si usa normalmente nelle tabelle della verità.

Figura u97.16. Mappe di Karnaugh contenenti la definizione dei prodotti fondamentali a cui ogni cella fa riferimento.



A	B	
0	0	A’B’
0	1	A’B
1	0	AB’
1	1	AB

	B’	B
A’	A’B’	A’B
A	AB’	AB

A	B	C	D	
0	0	0	0	A’B’C’D’
0	0	0	1	A’B’C’D
0	0	1	0	A’B’CD’
0	0	1	1	A’B’CD
0	1	0	0	A’BC’D’
0	1	0	1	A’BC’D
0	1	1	0	A’BCD’
0	1	1	1	A’BCD
1	0	0	0	AB’C’D’
1	0	0	1	AB’C’D
1	0	1	0	AB’CD’
1	0	1	1	AB’CD
1	1	0	0	ABC’D’
1	1	0	1	ABC’D
1	1	1	0	ABCD’
1	1	1	1	ABCD

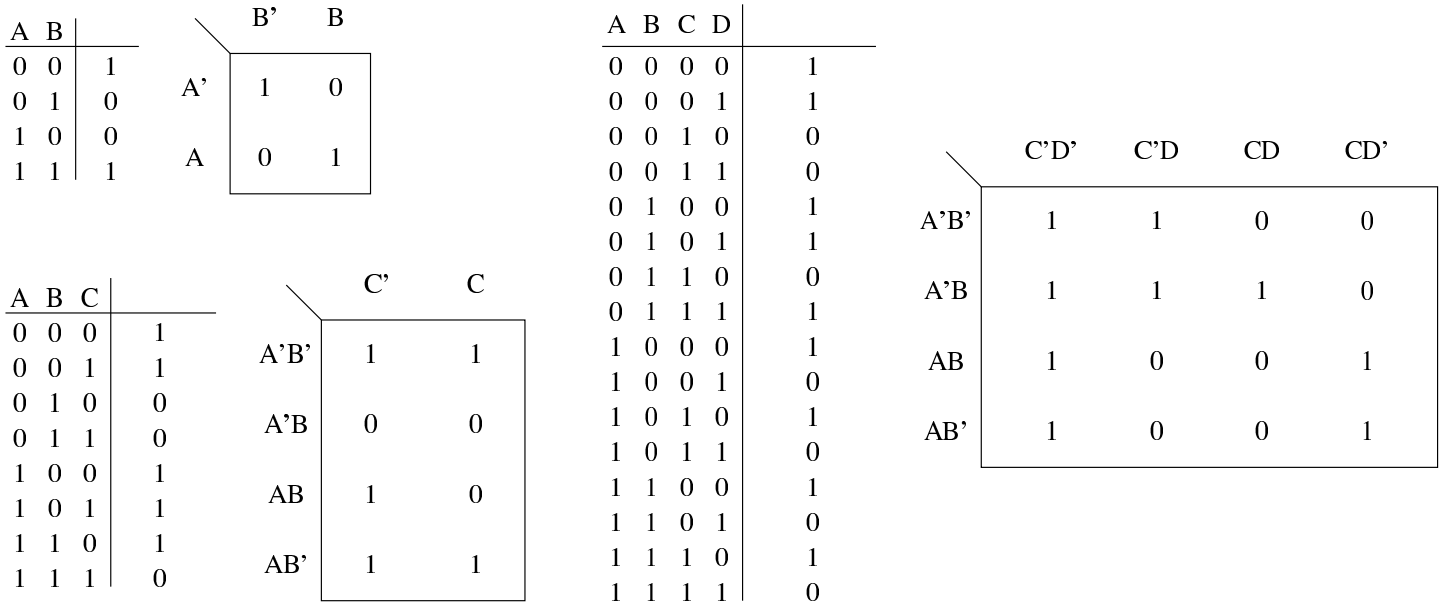
A	B	C	
0	0	0	A’B’C’
0	0	1	A’B’C
0	1	0	A’BC’
0	1	1	A’BC
1	0	0	AB’C’
1	0	1	AB’C
1	1	0	ABC’
1	1	1	ABC

	C’	C
A’B’	A’B’C’	A’B’C
A’B	A’BC’	A’BC
AB	**ABC’**	**ABC**
AB’	**AB’C’**	**AB’C**

	C’D’	C’D	CD	CD’
A’B’	A’B’C’D’	A’B’C’D	**A’B’CD**	**A’B’CD’**
A’B	A’BC’D’	A’BC’D	**A’BCD**	**A’BCD’**
AB	**ABC’D’**	**ABC’D**	***ABCD***	***ABCD’***
AB’	**AB’C’D’**	**AB’C’D**	***AB’CD***	***AB’CD’***

Le mappe vanno compilate mettendo nelle celle la cifra 1 in corrispondenza dei prodotti fondamentali validi. Le immagini successive propongono degli esempi, confrontando la tabella di verità con la mappa compilata corrispondente.

Figura u97.17. Mappe di Karnaugh compilate in base alle tabelle di verità che appaiono al loro fianco.



A	B	
0	0	1
0	1	0
1	0	0
1	1	1

	B’	B
A’	1	0
A	0	1

A	B	C	D	
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

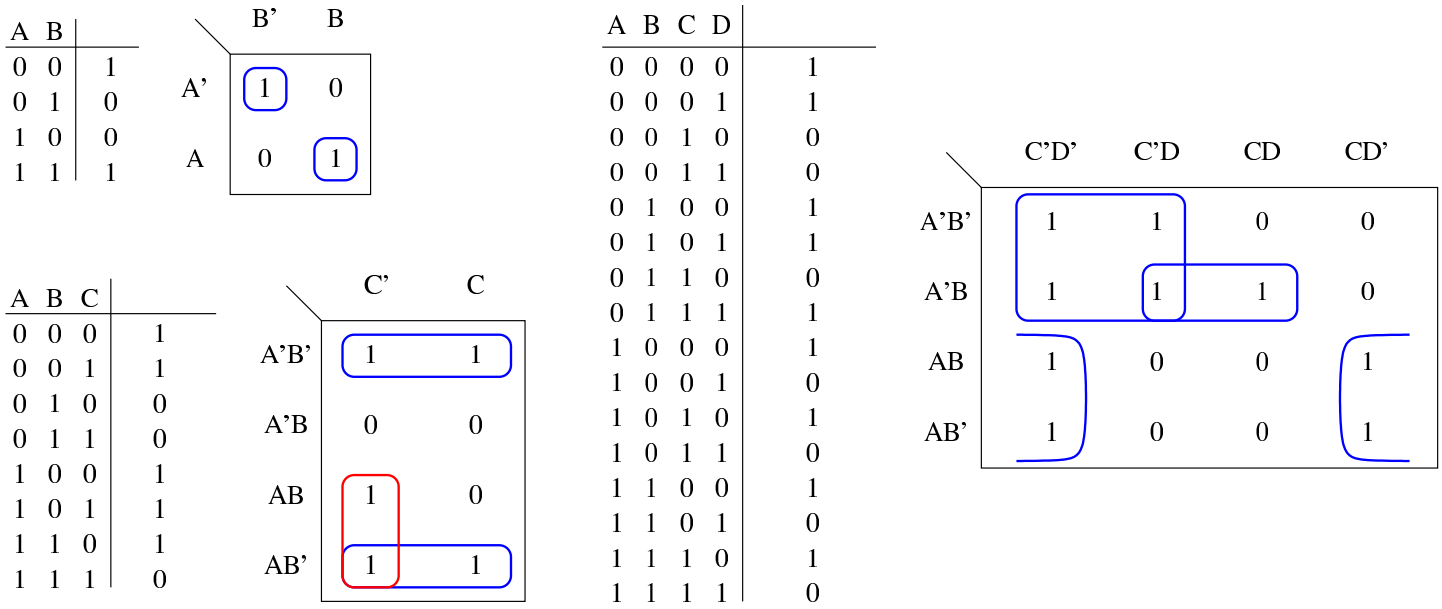
A	B	C	
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

	C’	C
A’B’	1	1
A’B	0	0
AB	1	0
AB’	1	1

	C’D’	C’D	CD	CD’
A’B’	1	1	0	0
A’B	1	1	1	0
AB	1	0	0	1
AB’	1	0	0	1

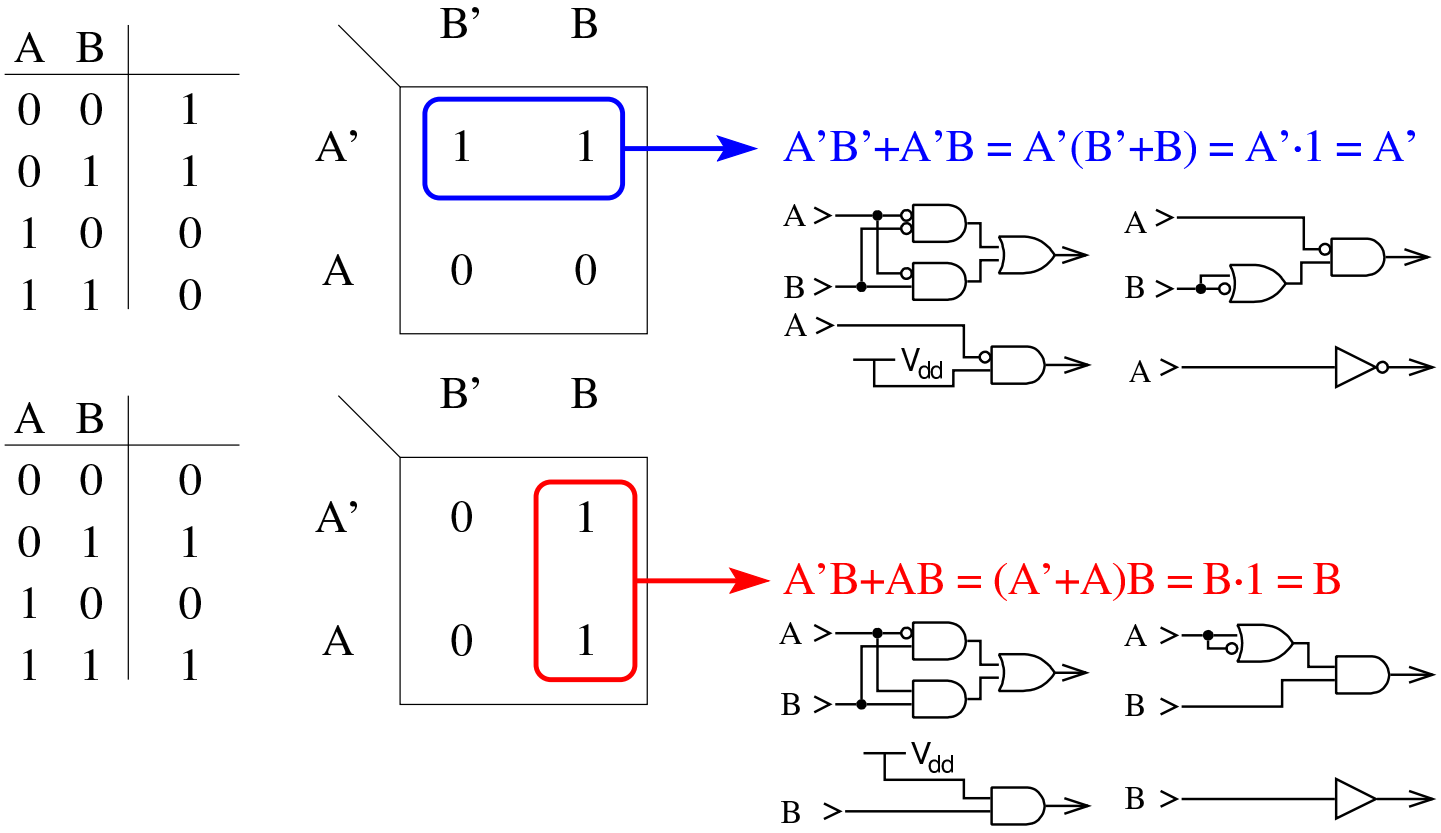
Una volta compilata la mappa, si vanno a raggruppare le celle che contengono la cifra 1 e che si trovano adiacenti in senso orizzontale o verticale.

Figura u97.18. Mappe di Karnaugh con le celle raggruppate verticalmente e orizzontalmente.



Quando due celle adiacenti, in verticale o in orizzontale, contengono il valore 1, una variabile che riguarda le due celle diventa inutile. La figura successiva dimostra intuitivamente il procedimento.

Figura u97.19. Semplificazione in presenza di due celle adiacenti con valore 1.



La semplificazione deriva dal fatto che ***x*** OR ***x'*** è sempre vero; pertanto, quando un raggruppamento rettangolare dimostra che una variabile appare sia nella sua forma normale, sia negata, significa che si può semplicemente ignorare.

Figura u97.20. Esempi di semplificazioni con mappe a due e tre variabili.

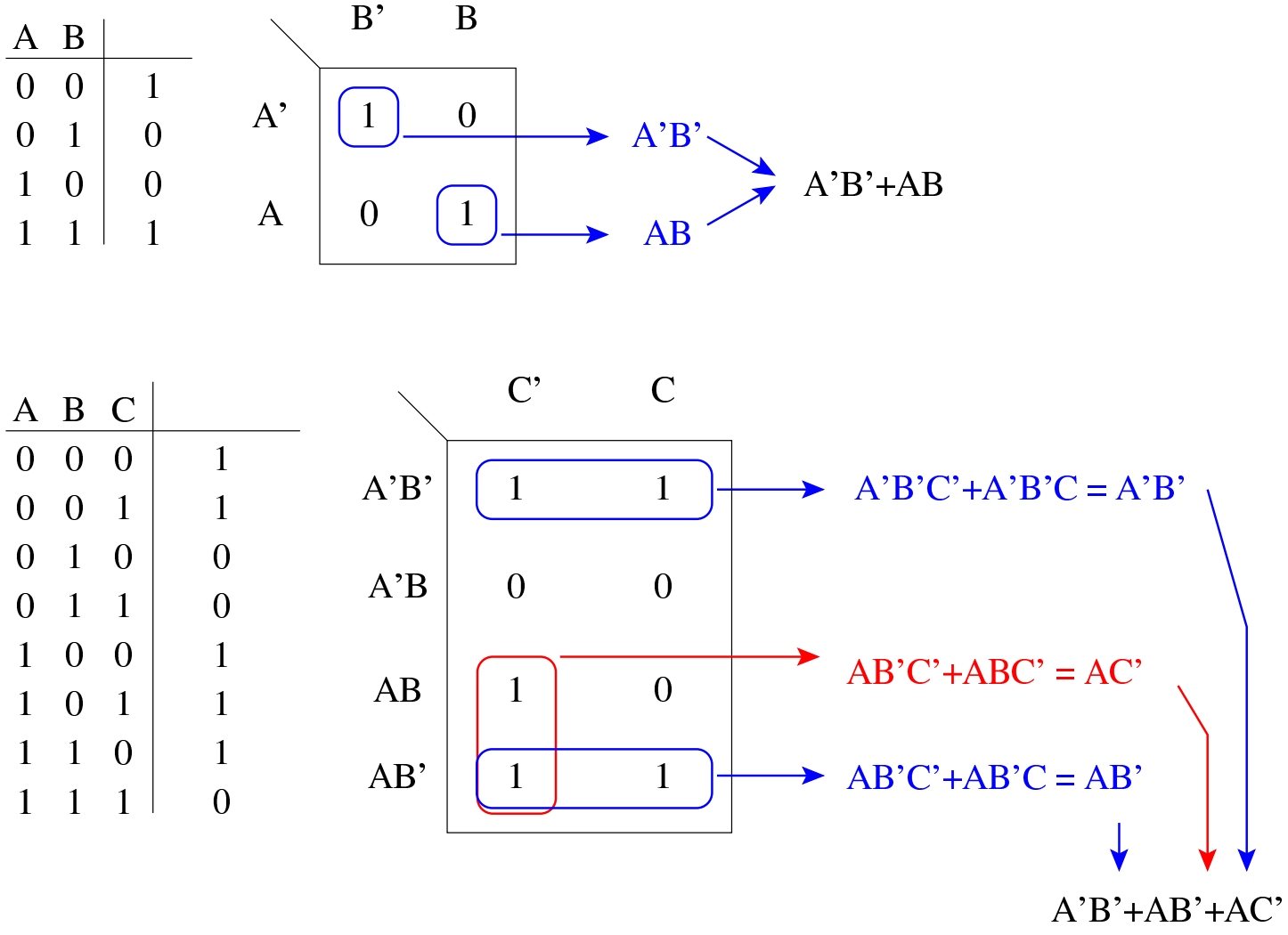
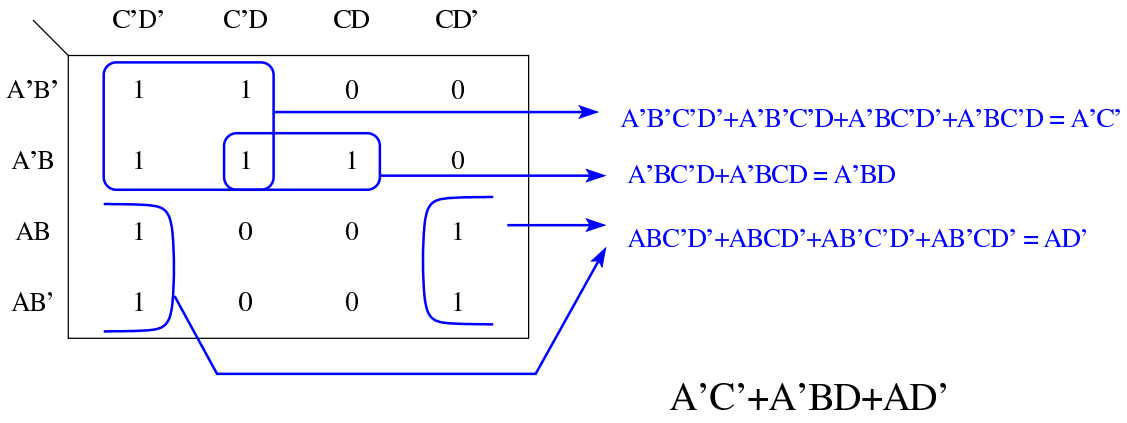


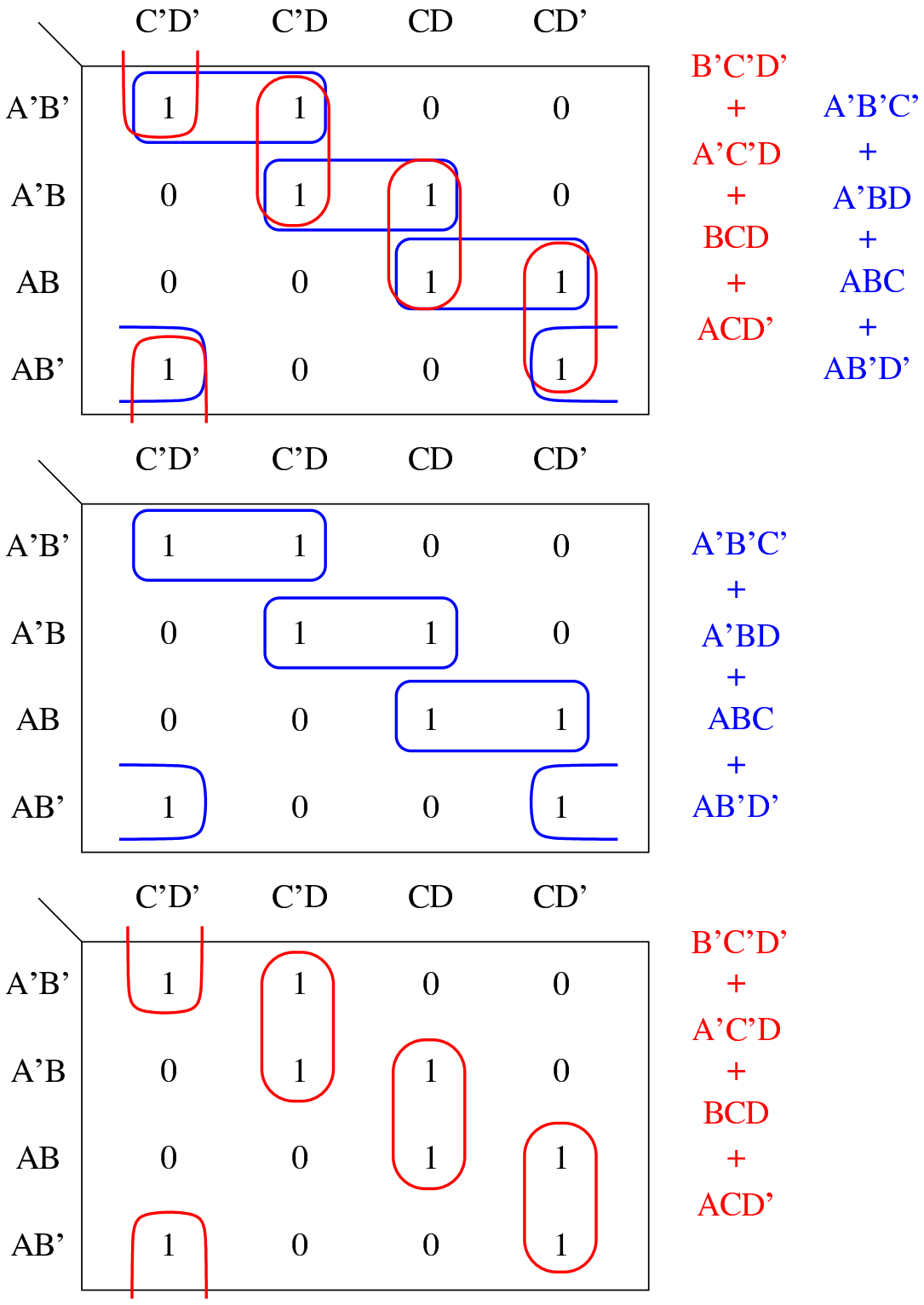
Figura u97.21. Esempio di semplificazione con mappa a quattro variabili.

A	B	C	D	
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0



Nella delimitazione delle zone che consentono di ottenere una semplificazione logica, si possono ignorare le zone che si sovrappongono completamente con altre, come si vede nella figura successiva, dove si possono produrre due risultati alternativi equivalenti.

Figura u97.22. Esempi di semplificazioni in presenza di aree sovrapposte e ridondanti.



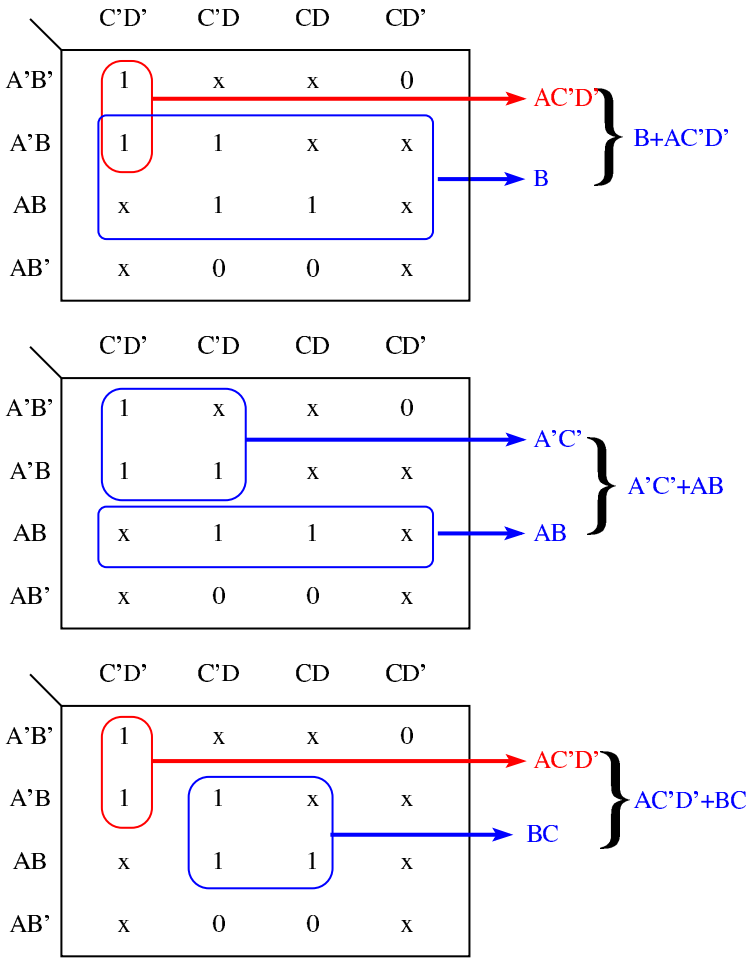
Le mappe di Karnaugh permettono di semplificare facilmente una funzione logica, purché non si superino le quattro variabili: di-

versamente sarebbe necessaria una rappresentazione a tre o più dimensioni.

In situazioni particolari, può accadere che sia indifferente il valore prodotto da alcune combinazioni delle variabili di ingresso. In questi casi, per sintetizzare la rete combinatoria si può scegliere liberamente il valore di uscita che risulta più conveniente nell'ottica della semplificazione.

Figura u97.23. Esempio di semplificazione in presenza di esiti indifferenti; si mostrano tre ipotesi di soluzione.

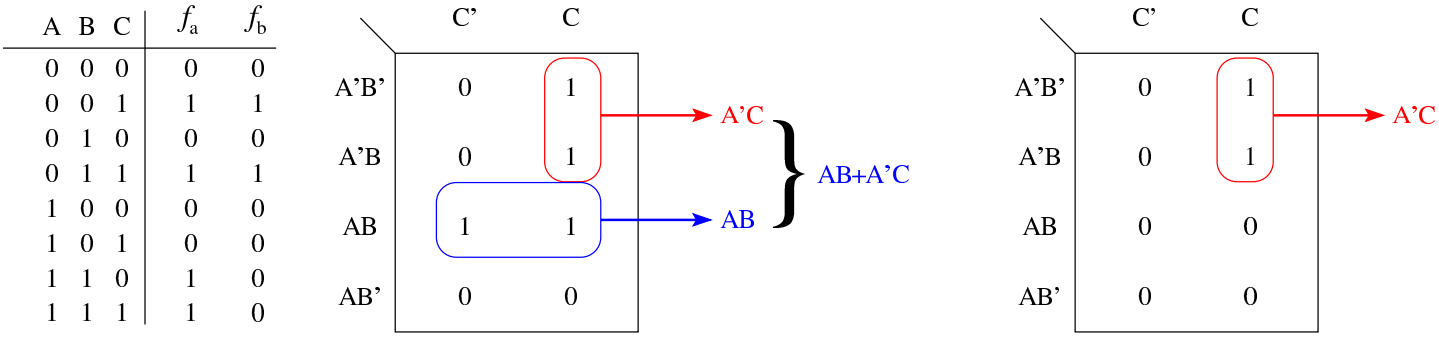
A	B	C	D	
0	0	0	0	1
0	0	0	1	x
0	0	1	0	0
0	0	1	1	x
0	1	0	0	1
0	1	0	1	1
0	1	1	0	x
0	1	1	1	x
1	0	0	0	x
1	0	0	1	0
1	0	1	0	x
1	0	1	1	0
1	1	0	0	x
1	1	0	1	1
1	1	1	0	x
1	1	1	1	1



Le mappe di Karnaugh si possono usare anche in presenza di più funzioni che condividono le stesse variabili di ingresso. In tal caso si valutano tante mappe quante sono le funzioni per cercare una sintesi

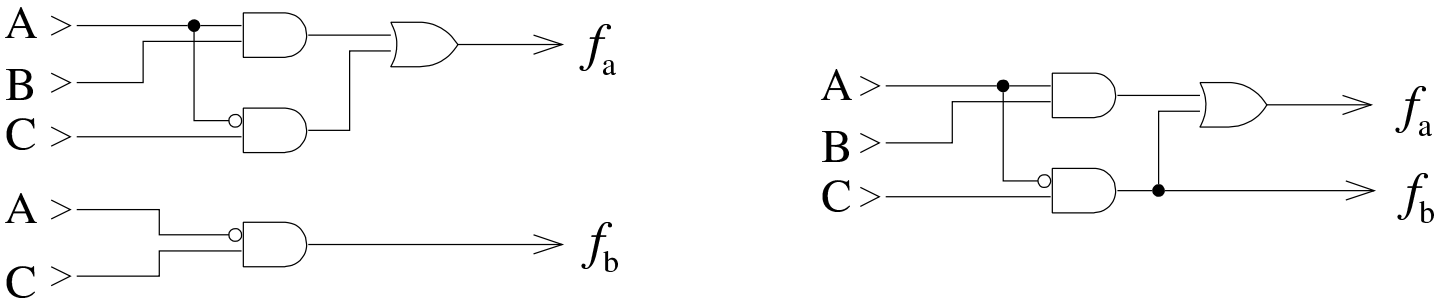
per ognuna; successivamente, se si realizzano queste funzioni attraverso un circuito logico, si cercano elementi comuni per condividerli senza duplicarli.

Figura u97.24. Due funzioni delle stesse variabili, sintetizzate attraverso due mappe.



A	B	C	f_a	f_b
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	0

Figura u97.25. Soluzione in forma di circuito logico, prima separata, poi unita risparmiando una porta logica.



Mappe di Karnaugh con la funzione XOR

«

Le mappe di Karnaugh danno soluzioni formate attraverso gli operatori logici fondamentali (AND, OR, NOT); tuttavia, spesso è comodo avvalersi dell'operatore XOR, al pari di quelli fondamentali. Nelle mappe di Karnaugh l'operatore XOR si manifesta in modo particolare, come si vede nella figura successiva.

Figura u97.26. Mappe di Karnaugh relative a funzioni XOR a due, tre e quattro variabili.

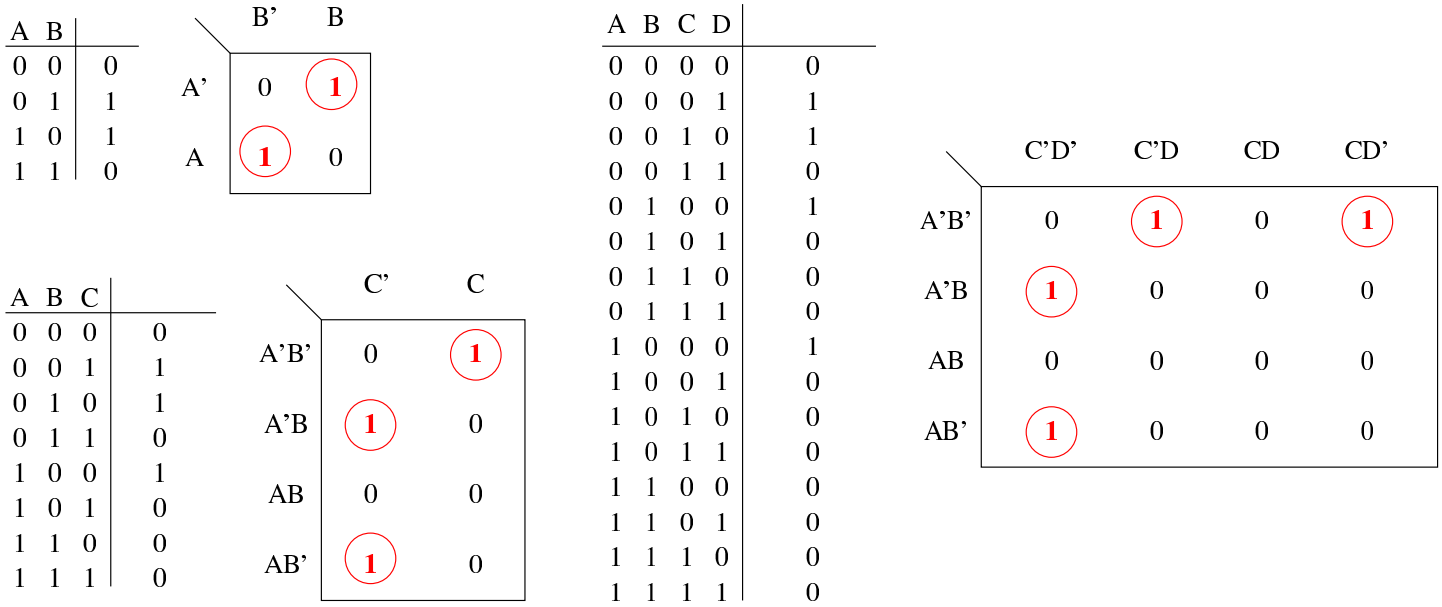
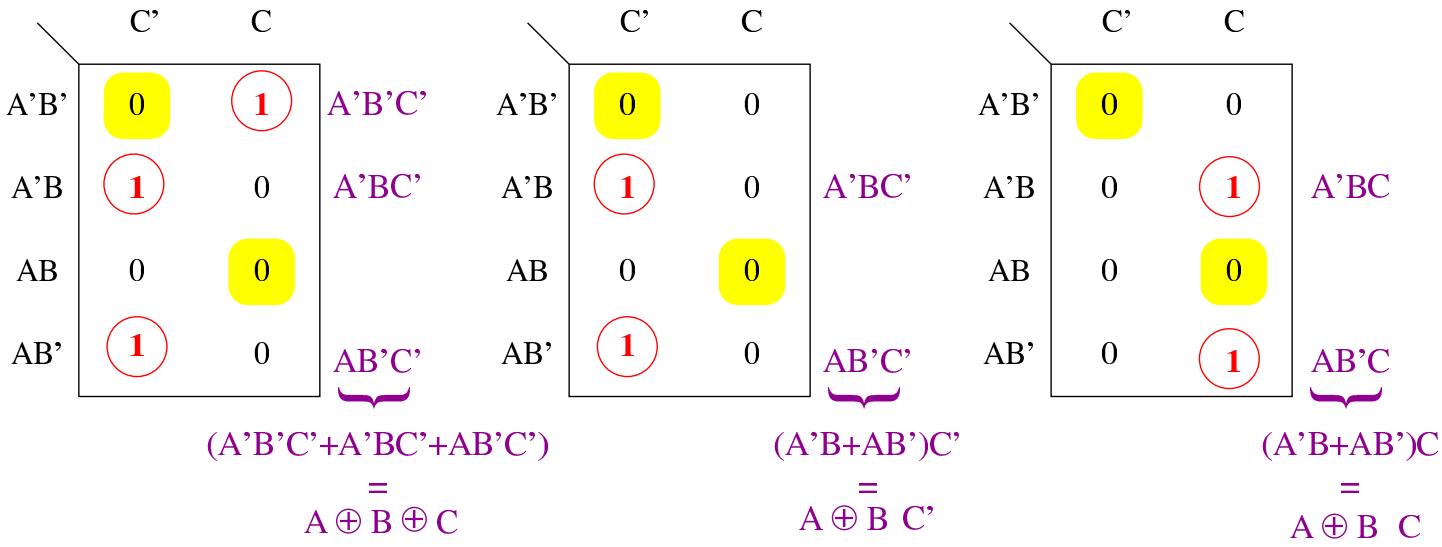


Figura u97.27. Esempi di individuazione di funzioni XOR a due variabili nelle mappe di Karnaugh a ingressi variabili. Le zone colorate di giallo sono impossibili, ammesso che le variabili usate per le funzioni XOR non siano negate.



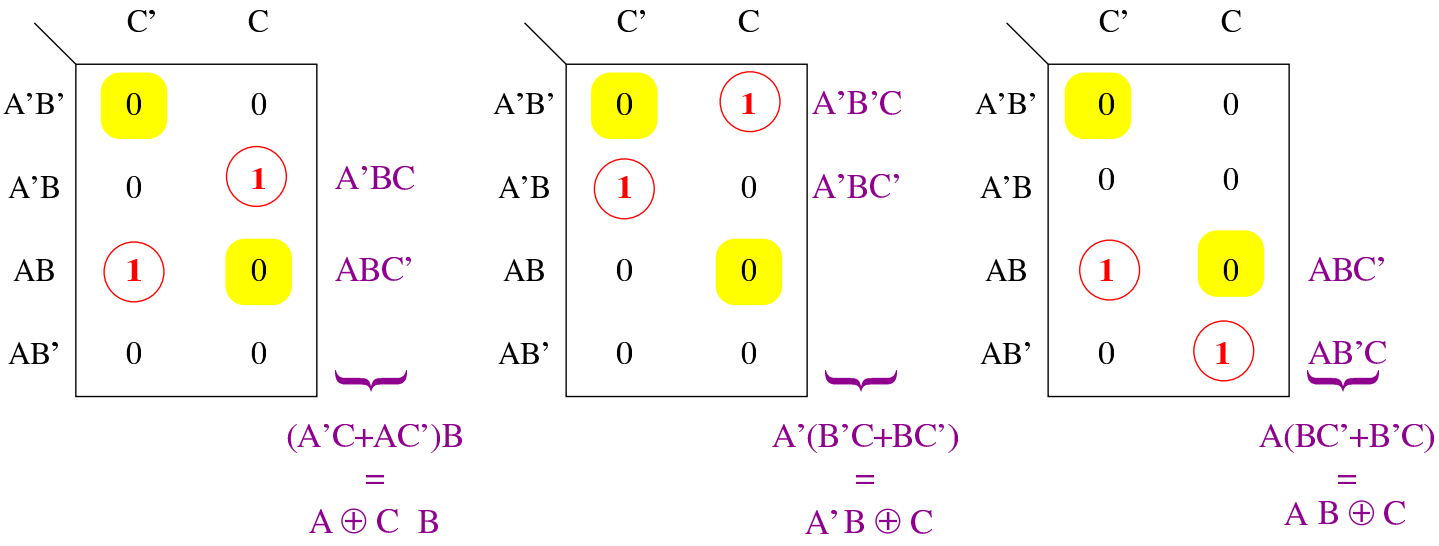
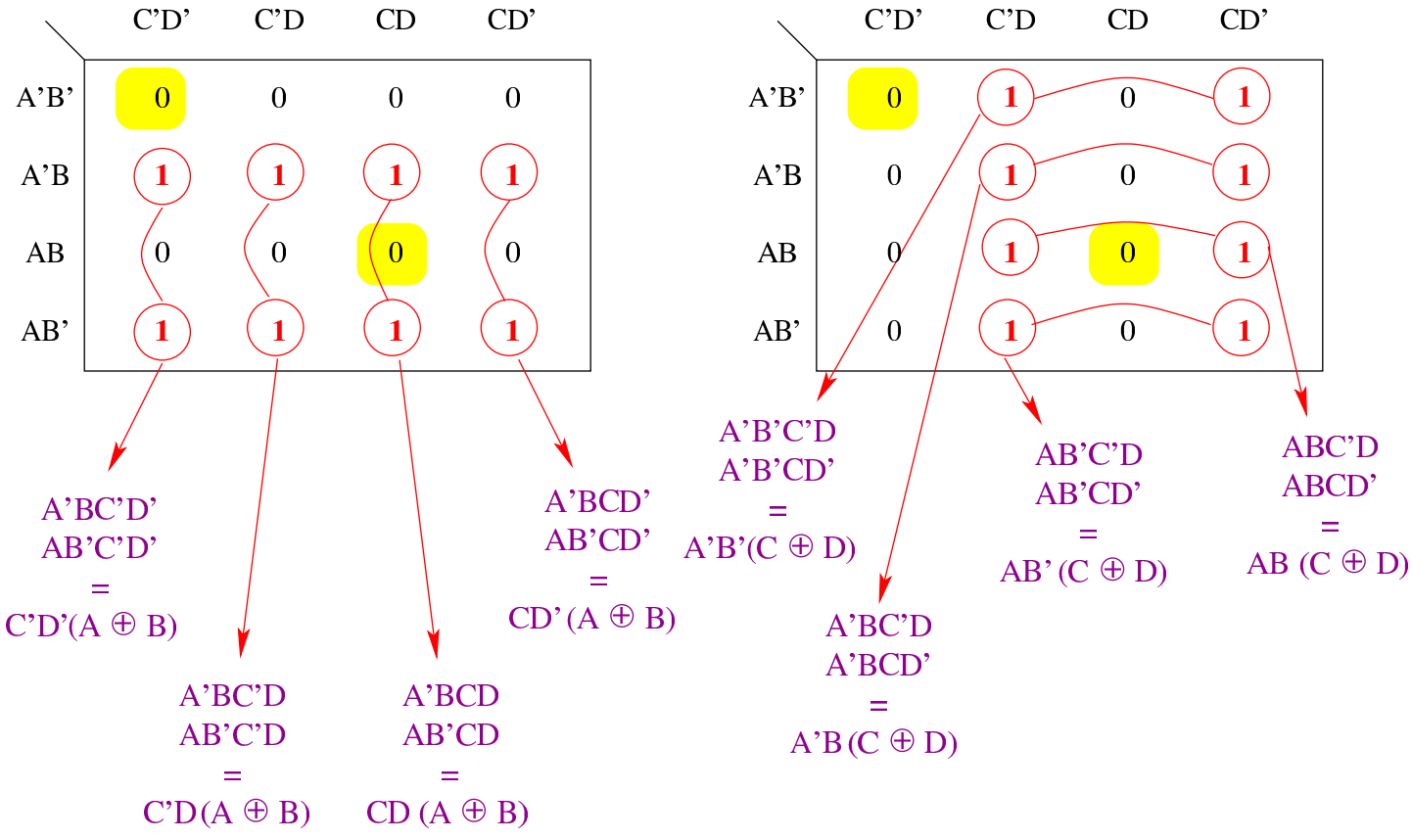


Figura u97.28. Esempi di individuazione di funzioni XOR a due variabili nelle mappe di Karnaugh a quattro ingressi: le zone in giallo sono impossibili, ammesso che le variabili usate per le funzioni XOR non siano negate.



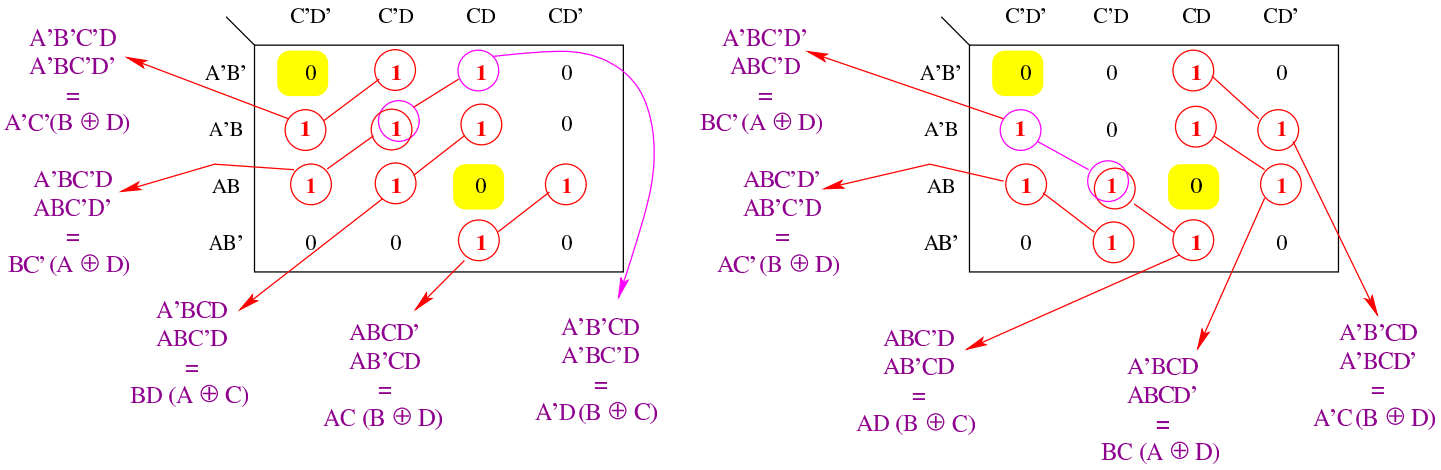
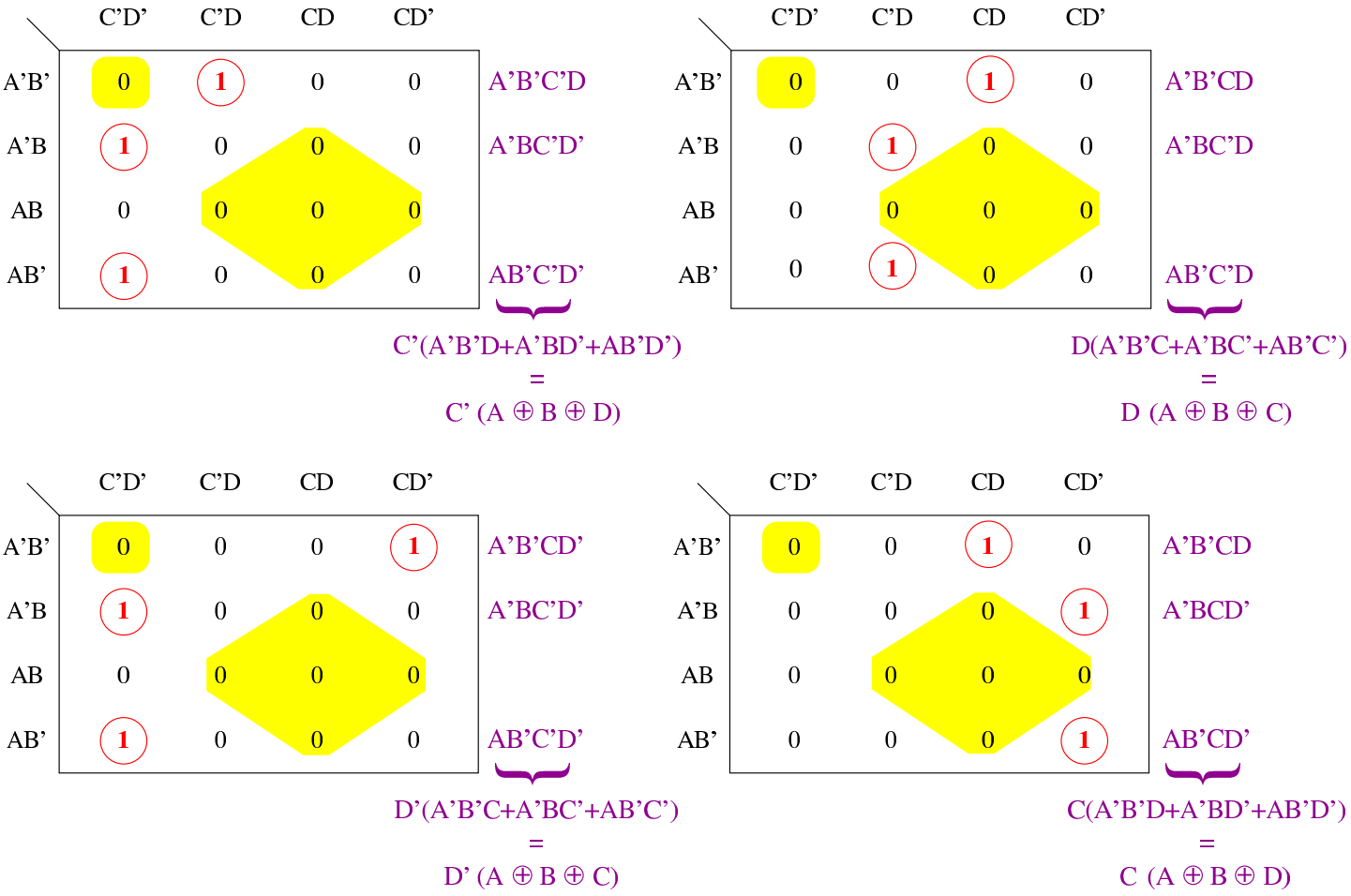
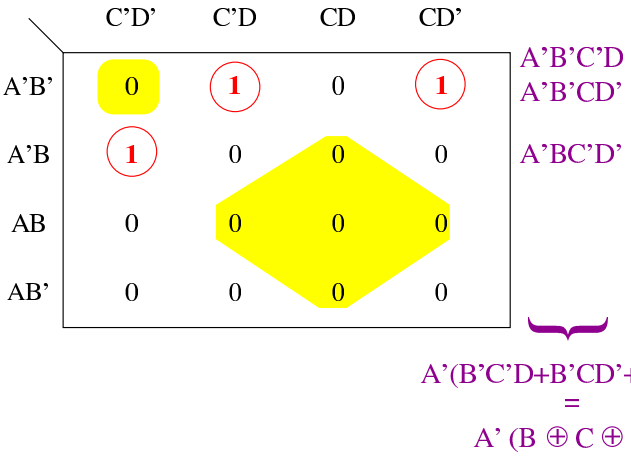
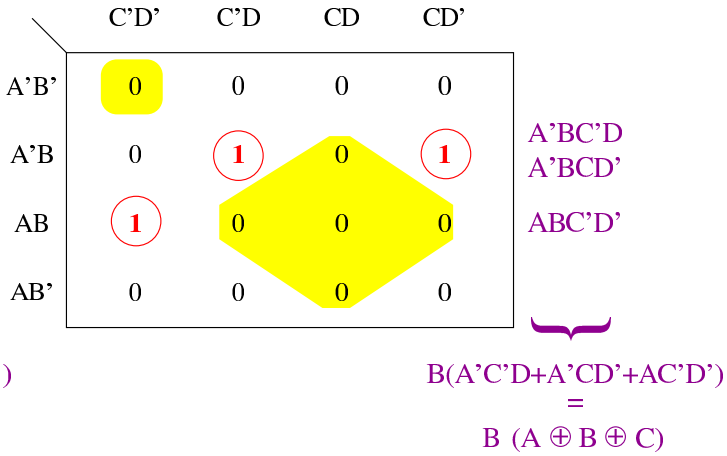
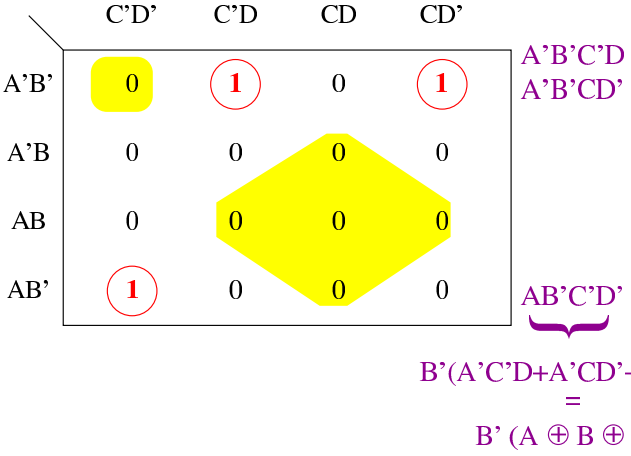


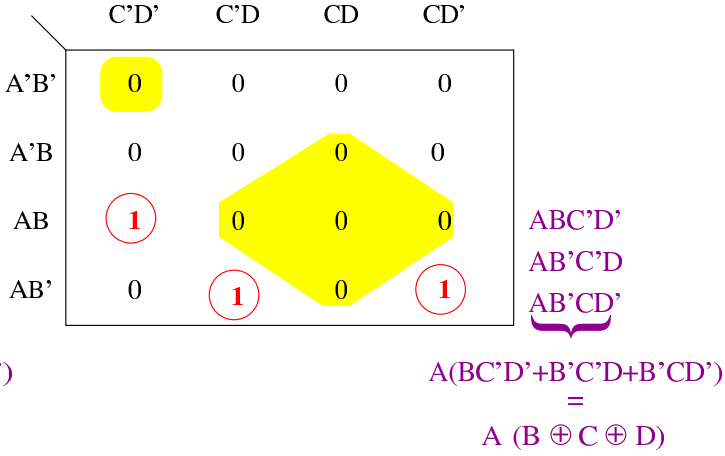
Figura u97.29. Esempi di individuazione di funzioni XOR a tre variabili nelle mappe di Karnaugh a quattro ingressi: le zone in giallo sono impossibili, ammesso che le variabili usate per le funzioni XOR non siano negate.




C'D'
C'D
CD
CD'
A'B'
A'B
AB
AB'
A'B'C'D
A'B'CD'
A'BC'D'
A'(B'C'D+B'CD'+BC'D')
=
A' (B ⊕ C ⊕ D)


C'D'
C'D
CD
CD'
A'B'
A'B
AB
AB'
A'BC'D
A'BCD'
ABC'D'
B(A'C'D+A'CD'+AC'D')
=
B (A ⊕ B ⊕ C)

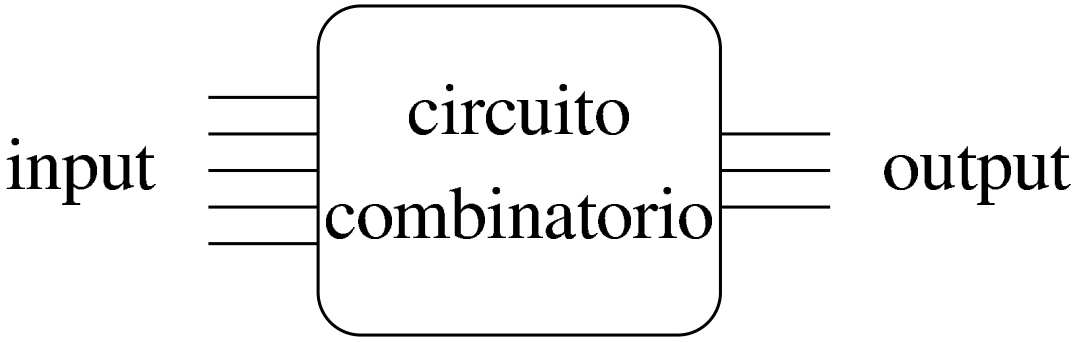

C'D'
C'D
CD
CD'
A'B'
A'B
AB
AB'
A'B'C'D
A'B'CD'
AB'C'D'
B'(A'C'D+A'CD'+AC'D')
=
B' (A ⊕ B ⊕ C)


C'D'
C'D
CD
CD'
A'B'
A'B
AB
AB'
ABC'D'
AB'C'D
AB'CD'
A(BC'D'+B'C'D+B'CD')
=
A (B ⊕ C ⊕ D)

«a2» 2013.11.11 --- Copyright © Daniele Giacomini -- appunti2@gmail.com http://informaticalibera.net

Un ***circuito combinatorio***, ovvero una ***rete combinatoria***, è un sistema di porte logiche, connesse opportunamente tra loro, organizzato con un insieme di ingressi e un insieme di uscite, nel quale i valori logici delle uscite sono determinati direttamente e univocamente dai

valori logici presenti negli ingressi. Il circuito combinatorio si può rappresentare, complessivamente, come una scatola composta da ingressi e da uscite, con una tabella di verità che stabilisce i valori delle uscite in base ai valori degli ingressi.

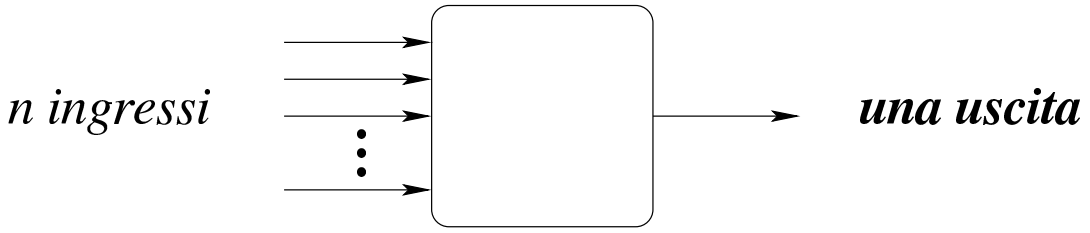


Quello raffigurato sopra è un esempio di circuito combinatorio con cinque ingressi e tre uscite, ma la proporzione tra quantità di ingressi e quantità di uscite dipende solo dalla funzione che deve realizzare tale circuito, ovvero dallo scopo che con questo ci si prefigge di raggiungere.

Si osservi che un circuito combinatorio, per essere tale, non deve essere influenzato dalla variabile tempo e nemmeno dalla variabile casuale dovuta all'accensione del circuito; pertanto, su tali circuiti non si considera il problema del tempo di propagazione, necessario a far sì che le uscite raggiungano i valori previsti in base ai valori pervenuti in ingresso.

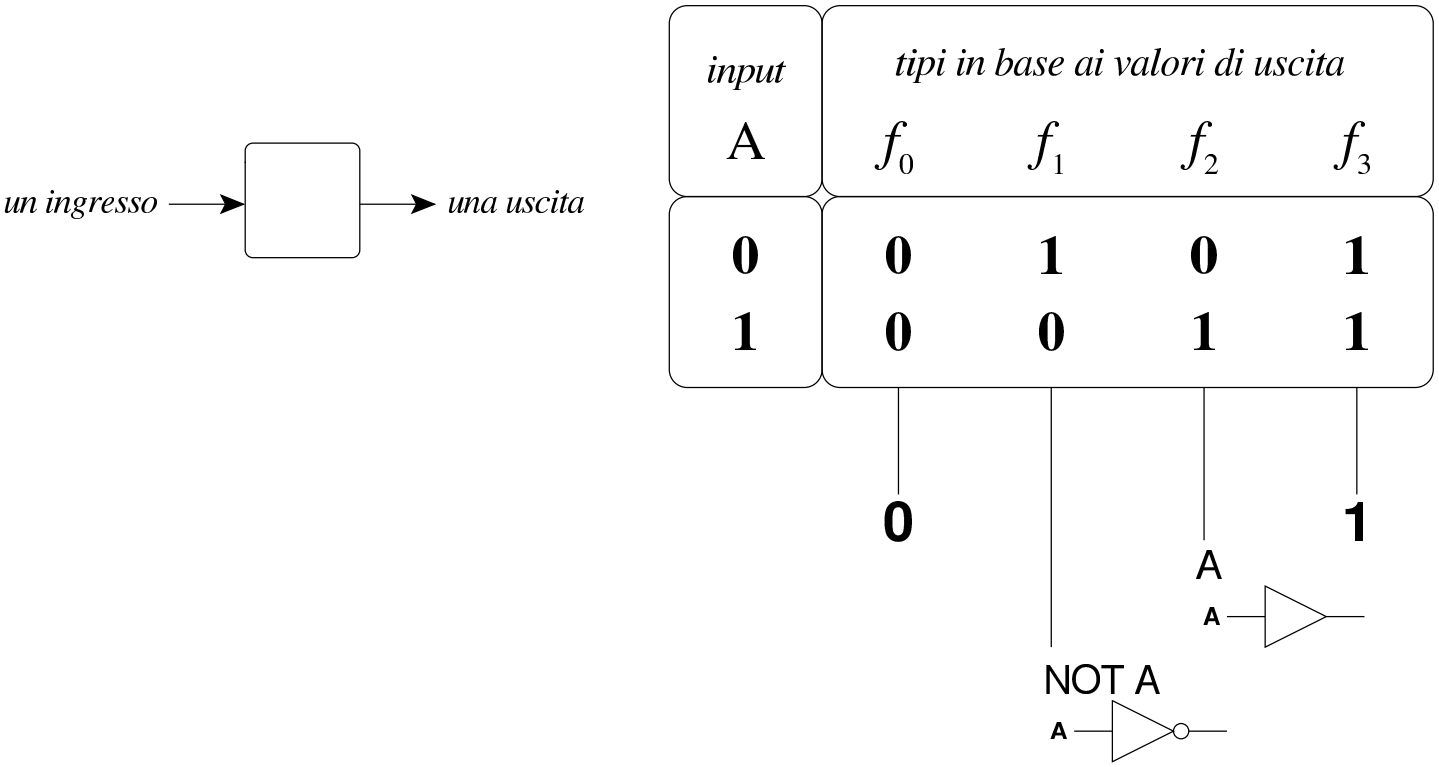
I circuiti combinatori più semplici sono quelli che dispongono di una sola uscita e realizzano quindi delle porte logiche, più o meno complesse.

Figura u98.2. Circuiti combinatori con una sola uscita.



Per comprendere la questione, si può osservare che un circuito composto da un solo ingresso e da una sola uscita può essere di quattro tipi differenti, come evidenziato dallo schema successivo.

Schema u98.3. Tabella di verità per quattro funzioni di una variabile.



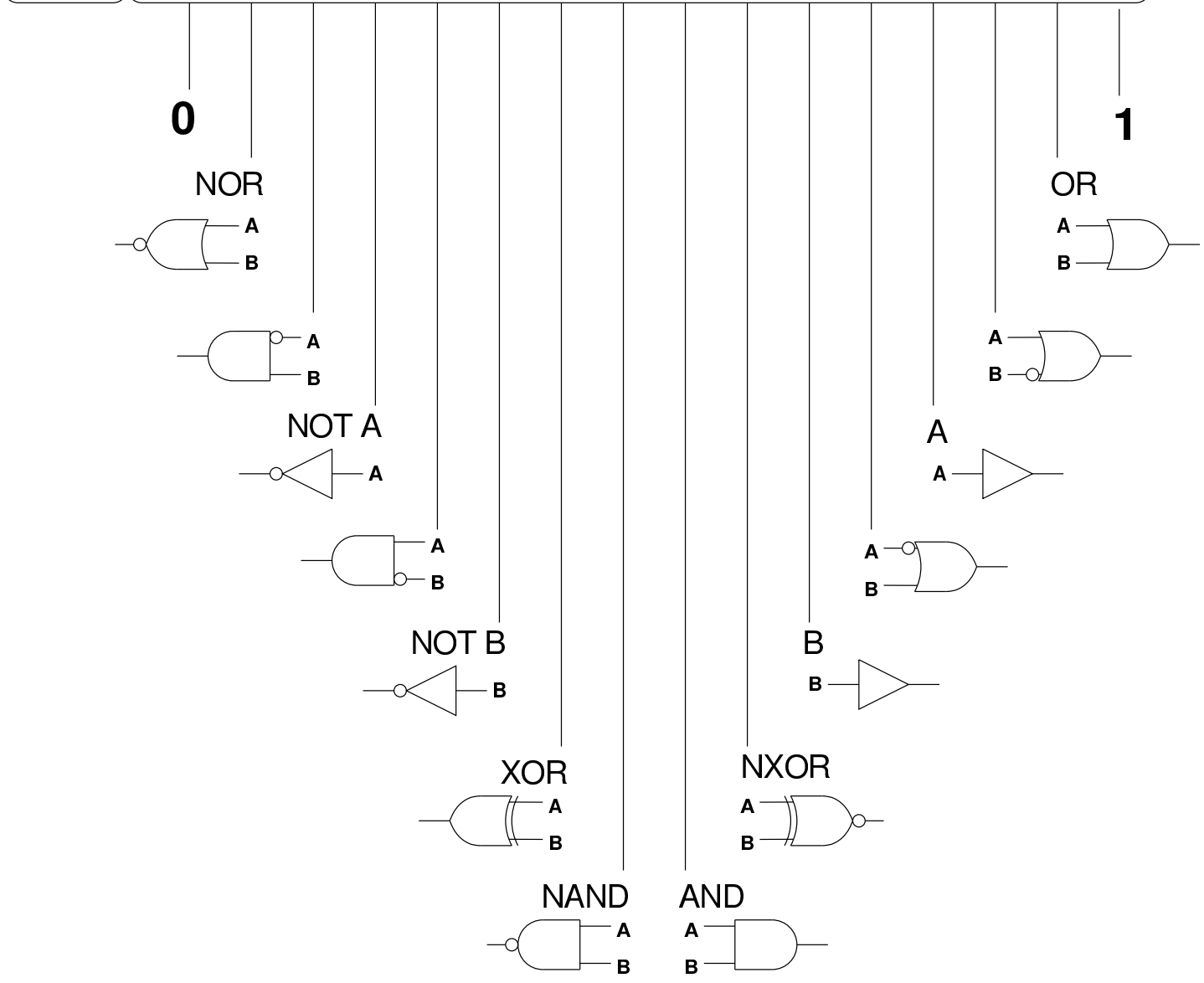
Come si vede dalle annotazioni contenute nello schema, il circuito corrispondente alla funzione f_1, coincide con il circuito invertente (ovvero NOT), mentre quello corrispondente alla funzione f_2 coincide con il circuito non-invertente.

In un circuito combinatorio con **due ingressi** e un'uscita, ci sono

16 funzioni possibili, come si vede nello schema successivo, dove si evidenziano in particolare le corrispondenze con le porte logiche comuni, indicate assieme al loro nome standard.

Schema u98.4. Tabella di verità per sedici funzioni di due variabili.

input		*tipi in base ai valori di uscita*															
A	B	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1



Decodificatore

«

Il decodificatore (*decoder*) è un circuito combinatorio che attiva una sola uscita, selezionandola in base alla combinazione di valori presenti negli ingressi, tenendo conto per ogni combinazione delle variabili di ingresso deve esserci un'uscita differente da attivare. Pertanto, per n ingressi ci sono 2^n uscite. Inoltre, solitamente, tale circuito combinatore è provvisto anche di un ingresso di controllo ulteriore, disattivando il quale si fa in modo che nessuna uscita risulti attiva, indipendentemente dagli altri valori in ingresso.

Figura u98.5. Schema a blocchi di un decodificatore a quattro uscite. I due ingressi di selezione sono contrassegnati dalle sigle a_0 e a_1; l'ingresso di abilitazione è indicato dalla sigla ***en*** (*enable*); le uscite sono indicate con le sigle da y_0 a y_3. Lo schema di destra è reso in una forma più compatta, dove gli ingressi di selezione risultano raccolti assieme.

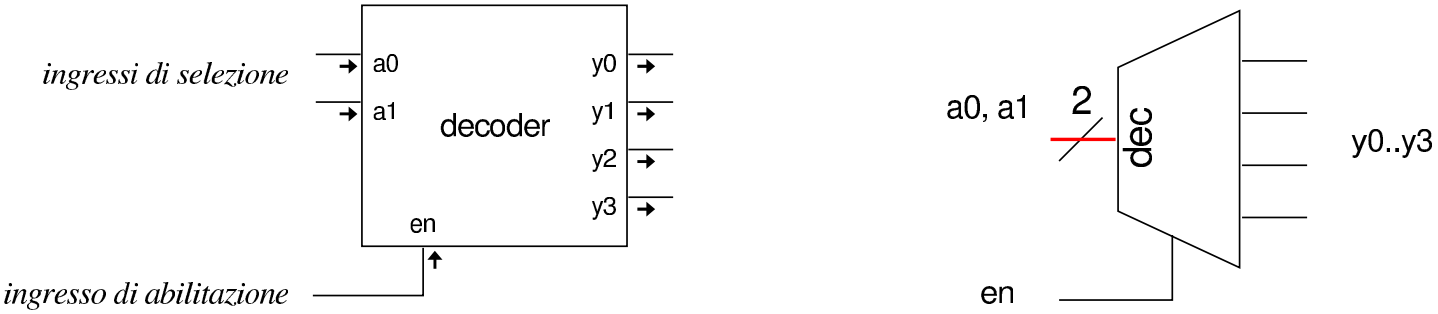
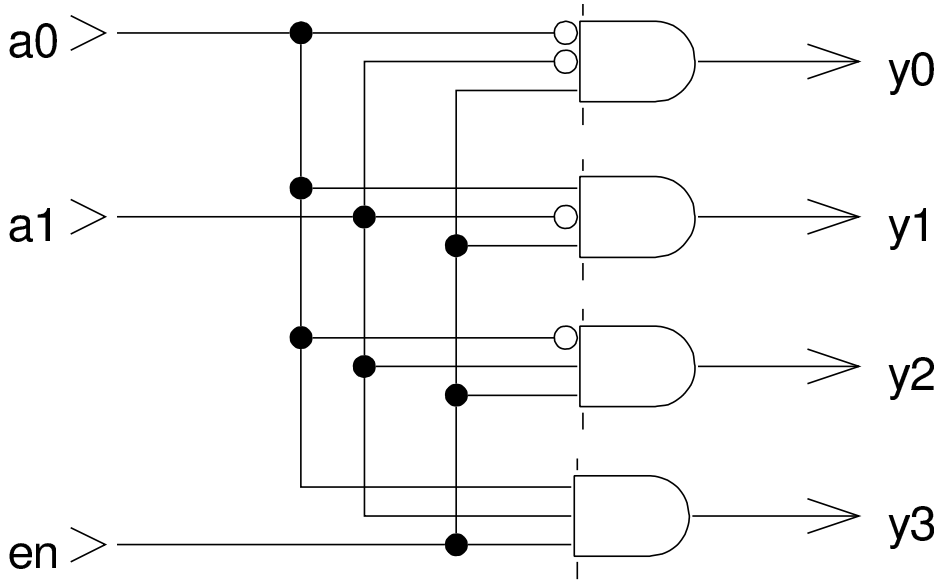


Tabella u98.6. Tabella di verità per un decodificatore a quattro uscite, da y_0 a y_3; due ingressi di selezione, a_0 e a_1; un ingresso di abilitazione en.

en	a_1	a_0	y_3	y_2	y_1	y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Figura u98.7. Esempio di realizzazione di un decodificatore a quattro uscite: i due ingressi di selezione sono contrassegnati dalle sigle a_0 e a_1; l'ingresso di abilitazione è indicato dalla sigla en; le uscite sono indicate con le sigle da y_0 a y_3.

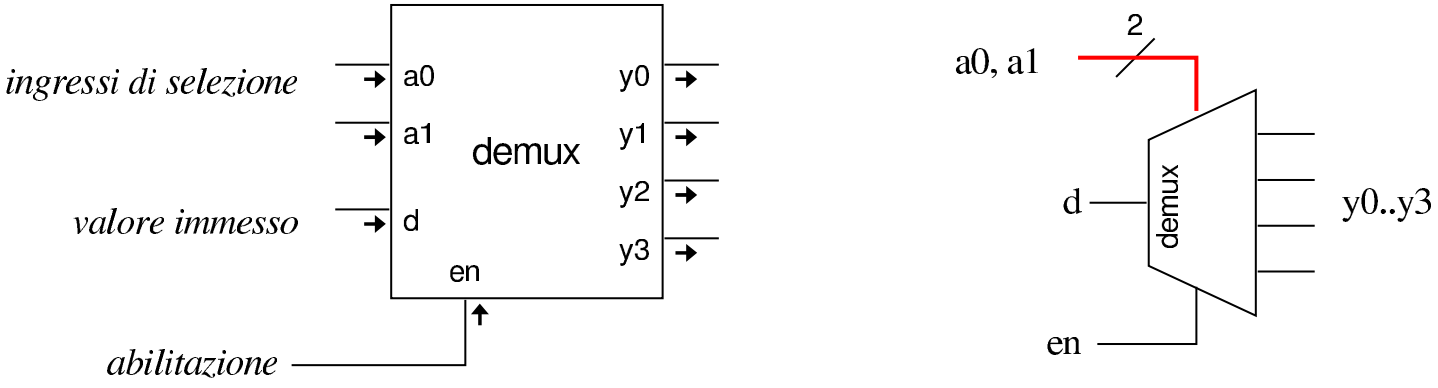


Demultiplatore

«

Il ***demultiplatore*** (*demultiplexer*), spesso abbreviato con la sigla ***demux***, svolge un lavoro simile al decodificatore, ma in qualità di commutatore del valore contenuto in un ingresso aggiuntivo.

Figura u98.8. Schema a blocchi di un demultiplatore a quattro uscite. I due ingressi di selezione sono contrassegnati dalle sigle a_0 e a_1; l'ingresso dati corrisponde alla variabile d; l'ingresso di abilitazione è indicato dalla sigla ***en***; le uscite sono indicate con le sigle da y_0 a y_3. Nel disegno di destra si vede una versione più compatta.

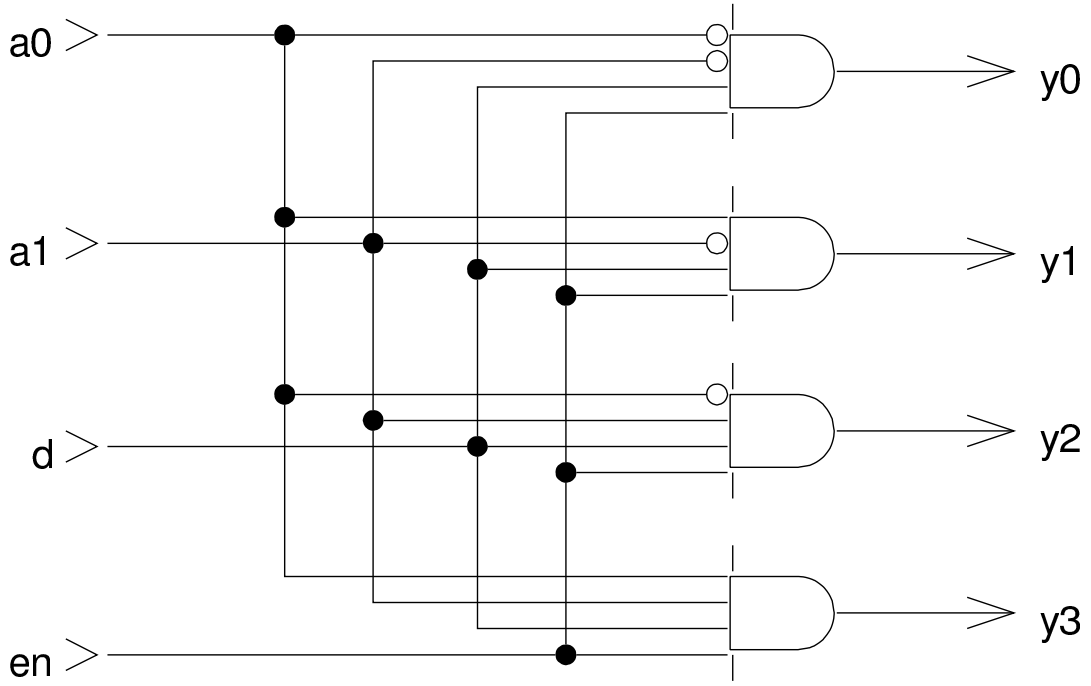


In questo caso, a differenza del decodificatore, l'uscita y_n selezionata riporta lo stesso valore dell'ingresso d.

Tabella u98.9. Tabella di verità per un demultiplatore a quattro uscite, da y_0 a y_3; due ingressi di selezione, a_0 e a_1; un ingresso dati d; un ingresso di abilitazione en.

en	a_1	a_0	d	y_3	y_2	y_1	y_0
0	x	x	d	0	0	0	0
1	0	0	d	0	0	0	d
1	0	1	d	0	0	d	0
1	1	0	d	0	d	0	0
1	1	1	d	d	0	0	0

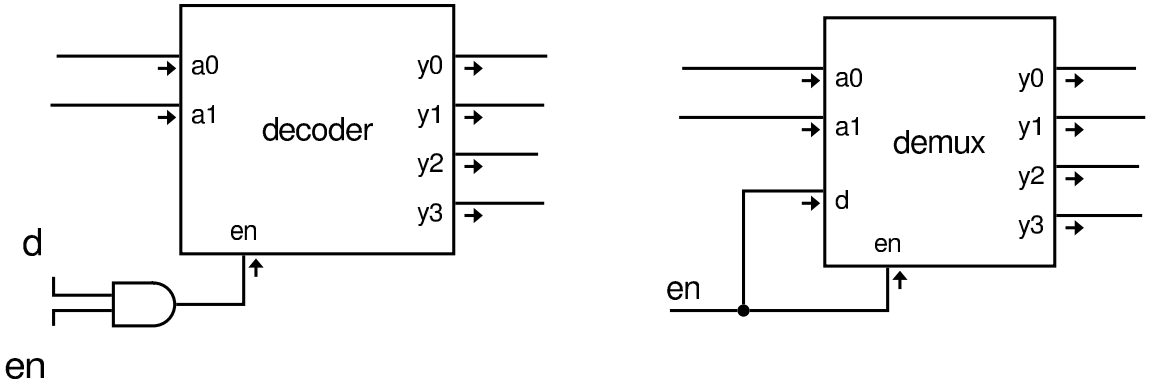
Figura u98.10. Esempio di realizzazione di un demultiplatore a quattro uscite. I due ingressi di selezione sono contrassegnati dalle sigle a_0 e a_1; il dato da commutare nell'uscita selezionata viene letto dall'ingresso d; l'ingresso di abilitazione è indicato dalla sigla en; le uscite sono indicate con le sigle da y_0 a y_3.



Il demultiplatore può essere ottenuto facilmente da un deco-

dificatore munito di ingresso di abilitazione; in modo analogo, un demultiplatore può essere ridotto a funzionare come un decodificatore.

Figura u98.11. A sinistra un decodificatore adattato per funzionare come demultiplatore; a destra un demultiplatore adattato per funzionare come decodificatore.



Multiplatore

«

Il ***multiplatore*** (*multiplexer*), spesso abbreviato con ***mux***, è un circuito combinatorio che seleziona il valore di una sola porta di entrata e lo riproduce nell'uscita. La selezione della porta di entrata dipende dalla combinazione di valori contenuta in un insieme di porte di selezione. Per n porte di selezione si può scegliere tra 2^n porte di entrata.

Figura u98.12. Multiplatore a quattro entrate. I due ingressi di selezione sono contrassegnati dalle sigle a_0 e a_1; le entrate hanno le sigle da d_0 a d_3; l'ingresso di abilitazione è indicato dalla sigla ***en*** (*enable*); l'uscita è indicate con la variabile ***y***. Lo schema di destra rappresenta una forma compatta, nella quale le variabili di selezione sono raccolte assieme in una linea multipla.

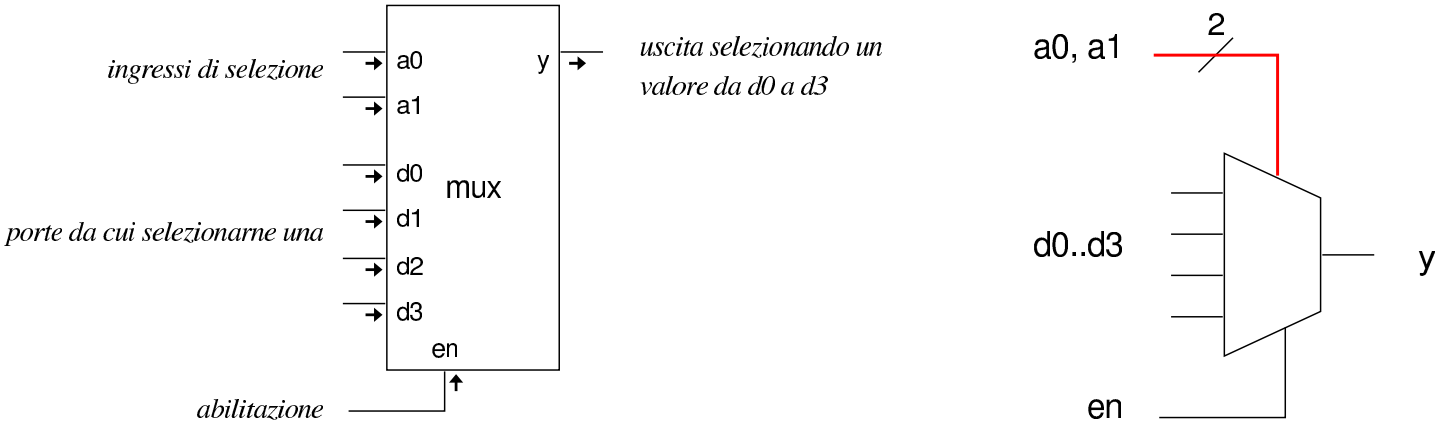
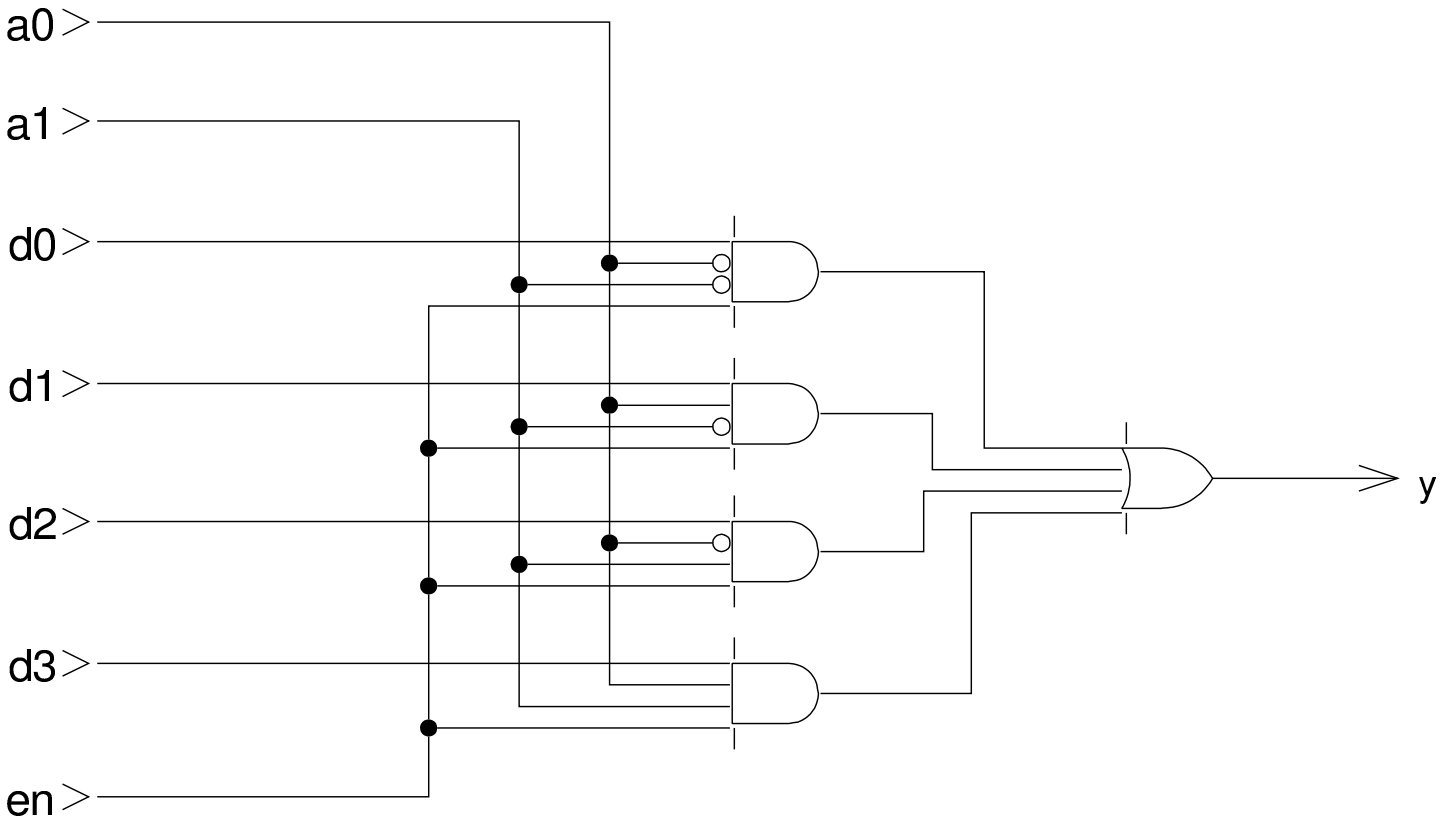


Tabella u98.13. Tabella di verità per un multiplatore a quattro ingressi di dati, da d_0 a d_3; due ingressi di selezione, da a_0 e a_1; un ingresso di abilitazione ***en***; un'uscita ***y***.

en	a_1	a_0	d_3	d_2	d_1	d_0	***y***
0	***x***	***x***	d_3	d_2	d_1	d_0	0
1	0	0	d_3	d_2	d_1	d_0	d_0
1	0	1	d_3	d_2	d_1	d_0	d_1
1	1	0	d_3	d_2	d_1	d_0	d_2
1	1	1	d_3	d_2	d_1	d_0	d_3

Figura u98.14. Esempio di realizzazione di un multiplatore a quattro entrate.



Demultiplatori e multiplatori in parallelo

«

Quando i demultiplatori o i multiplatori vengono usati per commutare dati composti da più bit e quindi trasportati da più linee, servono tanti demultiplatori o multiplatori quanti sono tali bit, mettendo però assieme tutti gli ingressi di selezione e di abilitazione. Tuttavia, nel disegno di un circuito tali linee multiple si annotato in forma compatta. Le figure successive mostrano un demultiplatore e un multiplatore che trattano dati a quattro bit.

Figura u98.15. Demultiplatore con entrata e uscite composte da vettori di quattro valori.

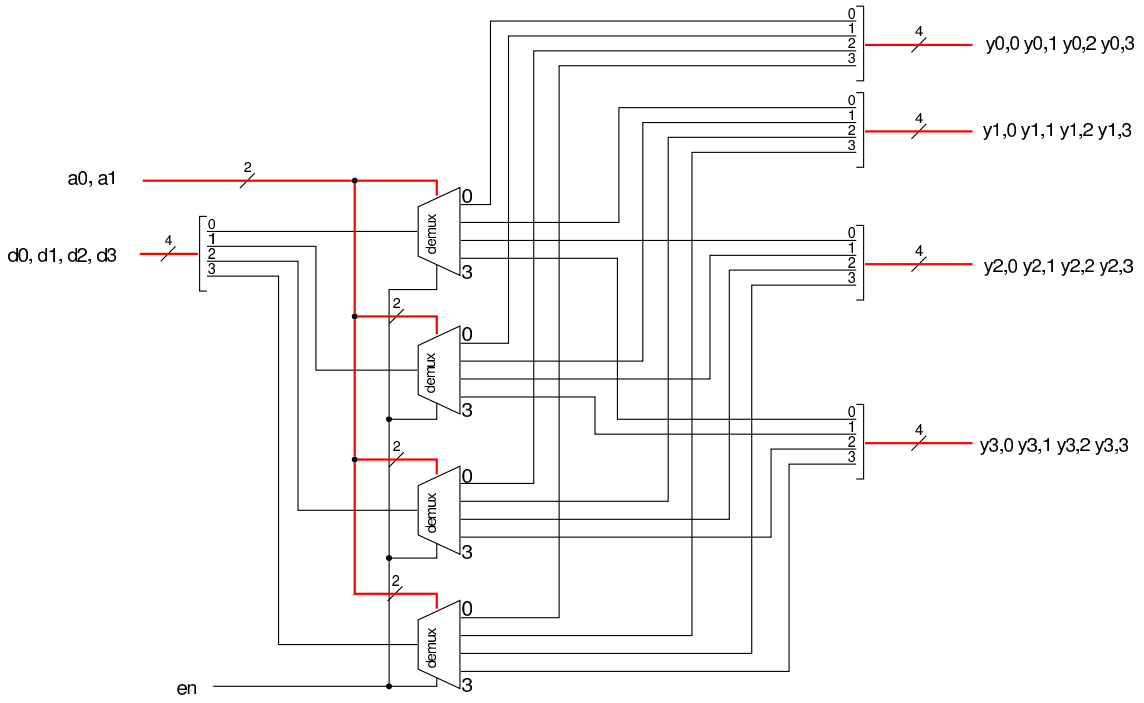
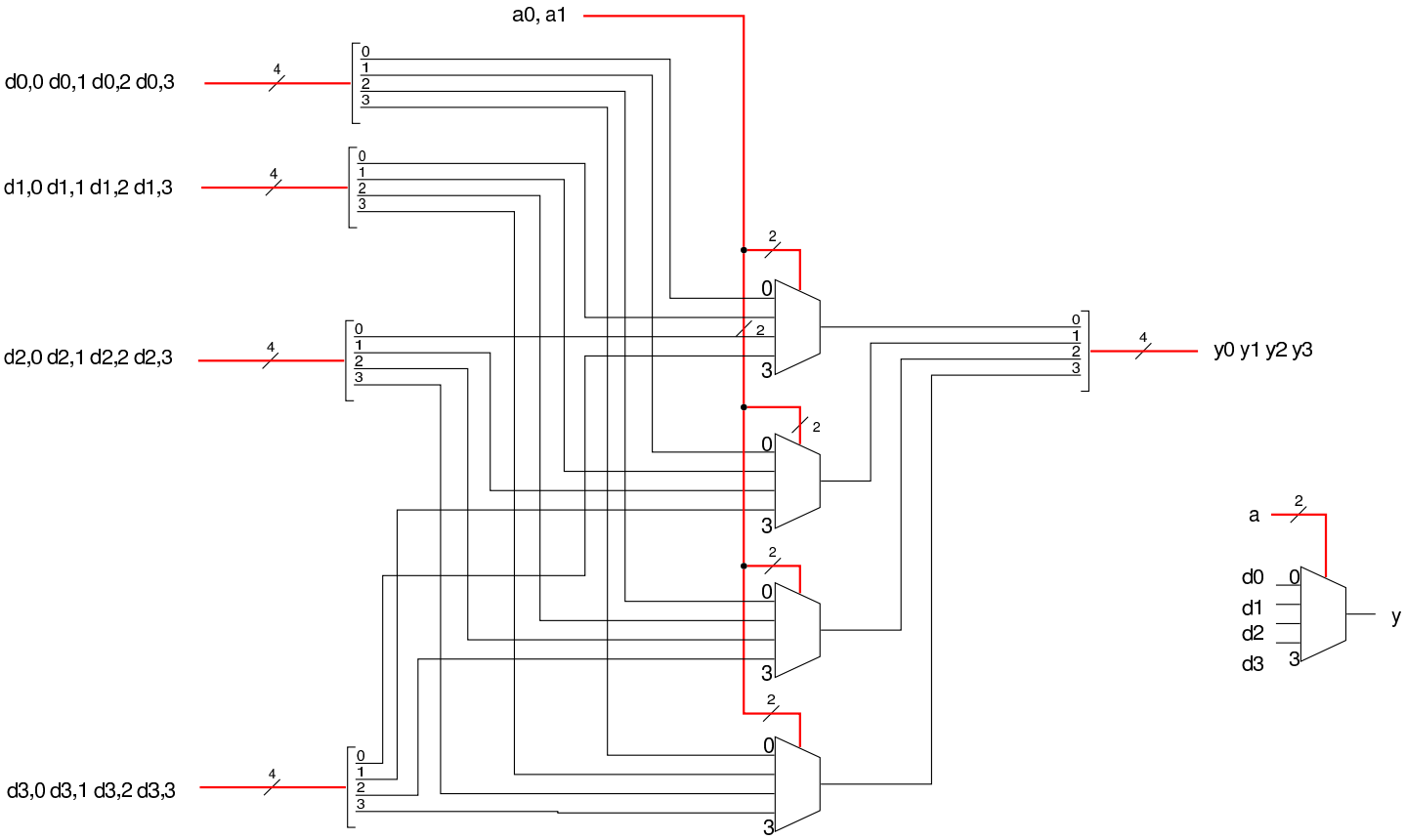


Figura u98.16. Multiplatore con entrate e uscita composte da vettori di quattro valori.



Codificatore binario

Un ***codificatore binario*** (*binary encoder* è un circuito combinatorio nei cui ingressi ci può essere una sola linea attiva e ha lo scopo di tradurre l'ingresso selezionato in un numero binario nelle uscite. In pratica, si tratta generalmente di 2^n ingressi e di n uscite. Vengono mostrati esempi di codificatori con n pari a 1, 2 e 3.

Tabella u98.17. Tabella di verità per un codificatore binario avente due ingressi e una uscita.

w_1	w_0	y_0
0	1	0
1	0	1

Figura u98.18. Il codificatore binario a due ingressi è un circuito inutile, in quanto basta collegare il secondo ingresso all'uscita per ottenere il risultato.

Tabella u98.19. Tabella di verità per un codificatore binario avente quattro ingressi e due uscite.

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Figura u98.20. Codificatore binario a quattro ingressi e due uscite. Il primo ingresso non viene collegato, perché in quella circostanza l'uscita è comunque pari a zero.

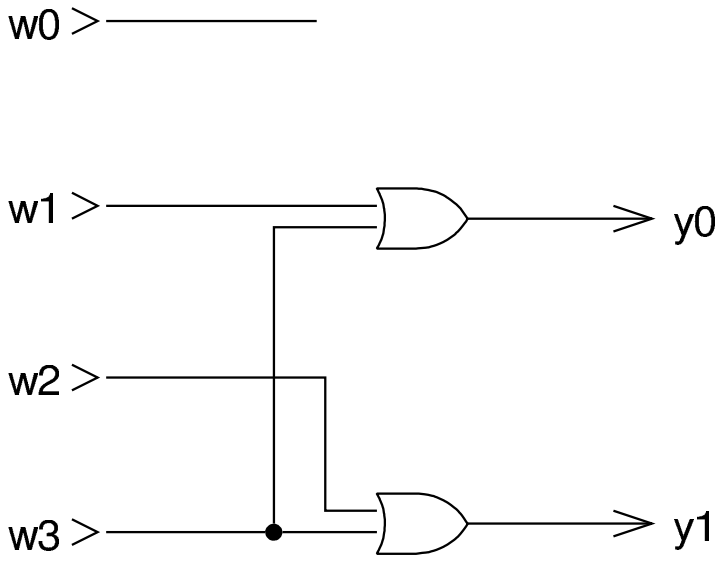
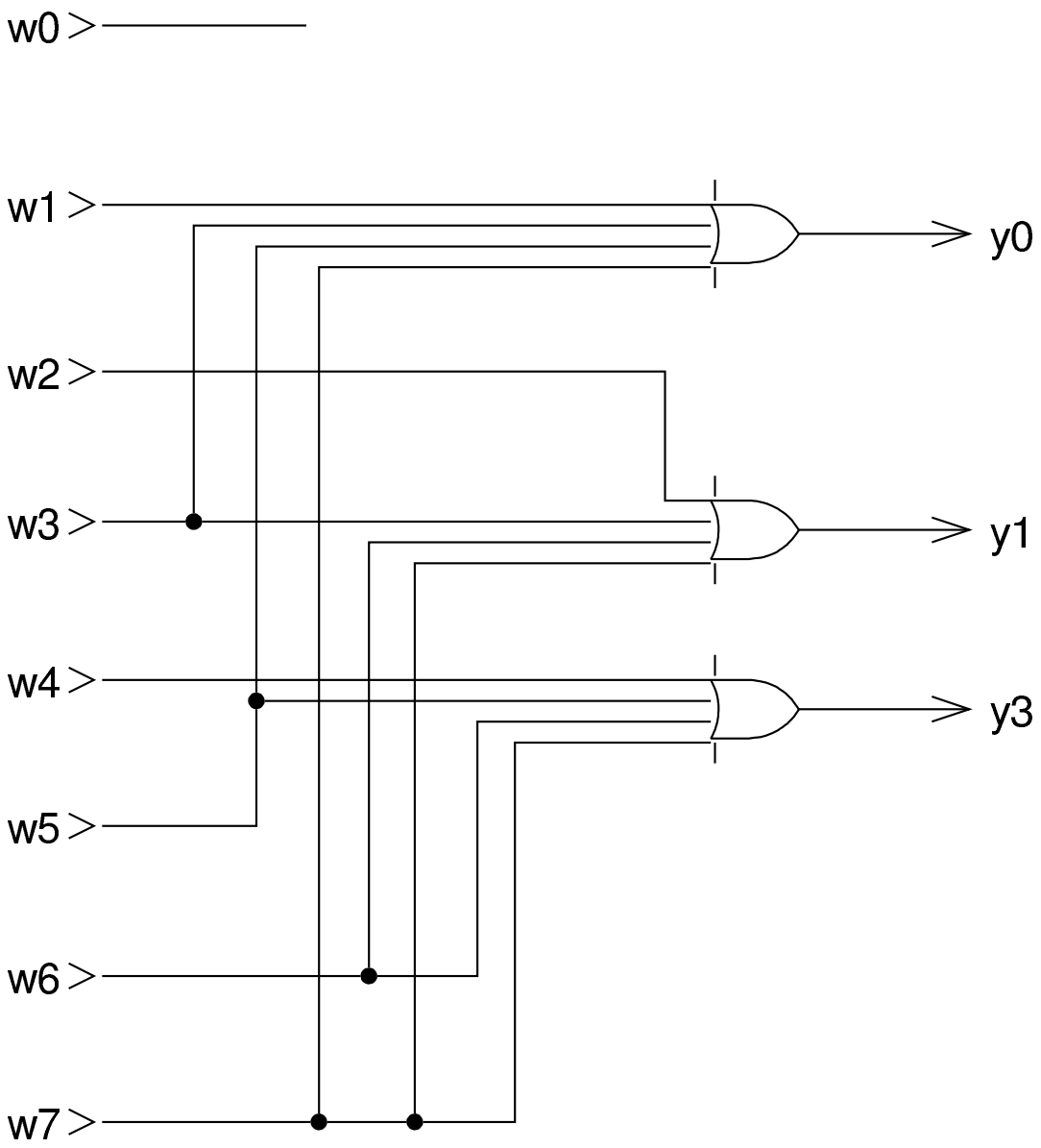


Tabella u98.21. Tabella di verità per un codificatore binario avente otto ingressi e tre uscite.

w_7	w_6	w_5	w_4	w_3	w_2	w_1	w_0	y_2	y_1	y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Figura u98.22. Codificatore binario a otto ingressi e tre uscite.



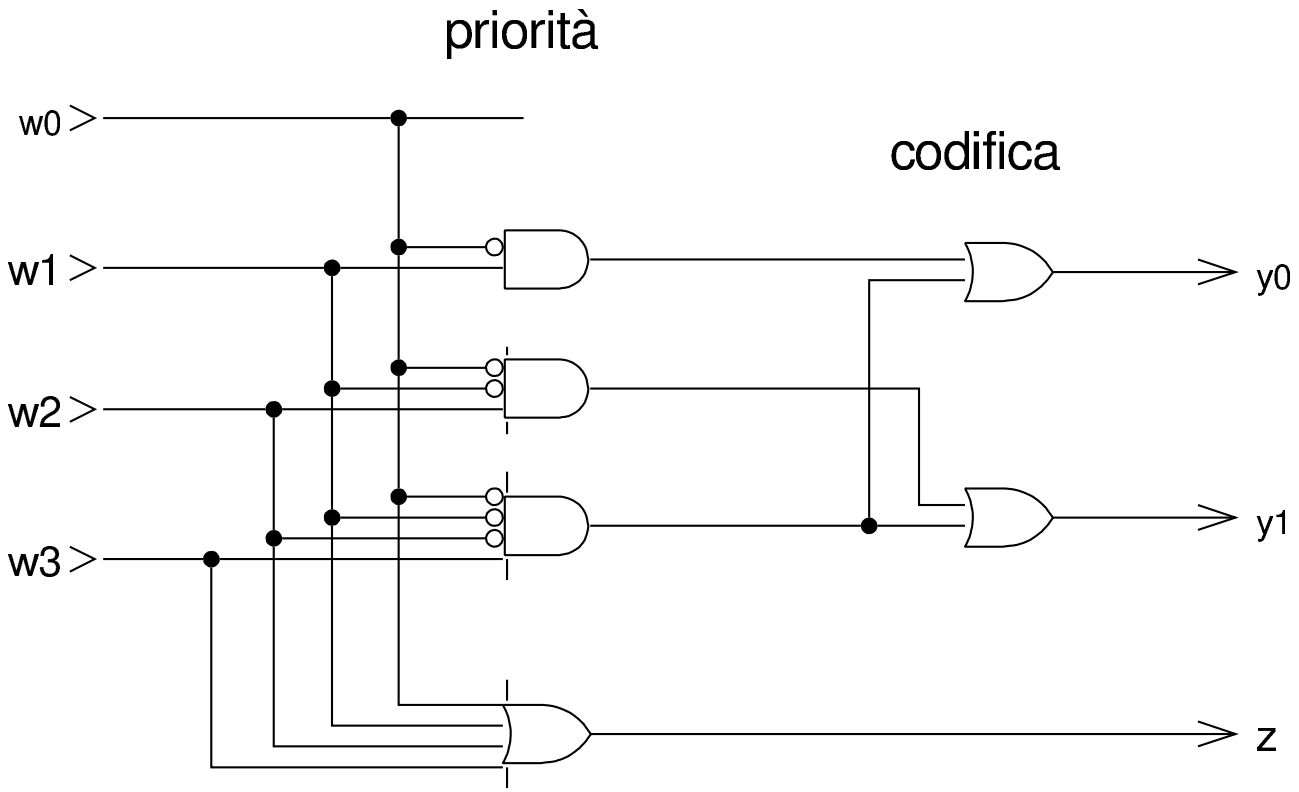
Codificatore di priorità

Il ***codificatore di priorità*** (*priority encoder*) è un codificatore binario che ammette l'attivazione di qualunque ingresso, emettendo il numero corrispondente all'ingresso attivo avente priorità rispetto agli altri. In questo caso è ammissibile anche il fatto che nessun ingresso sia attivo, pertanto, può essere presente un'uscita aggiuntiva che si attiva quando in ingresso c'è almeno un valore attivo.

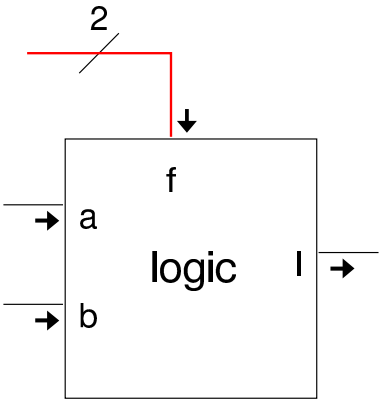
Tabella u98.23. Tabella di verità per un codificatore di priorità avente quattro ingressi. L'ingresso attivo che ha indice più basso ha priorità rispetto agli altri, pertanto si usa la lettera ***x*** per indicare un valore indifferente della variabile corrispondente; l'uscita *z* si attiva se esiste almeno un ingresso attivo e quando l'uscita *z* è a zero, non ha importanza il valore delle altre uscite.

w_3	w_2	w_1	w_0	z	y_1	y_0
0	0	0	0	0	x	x
x	x	x	1	1	0	0
x	x	1	0	1	0	1
x	1	0	0	1	1	0
1	0	0	0	1	1	1

Figura u98.24. Codificatore di priorità a quattro ingressi.

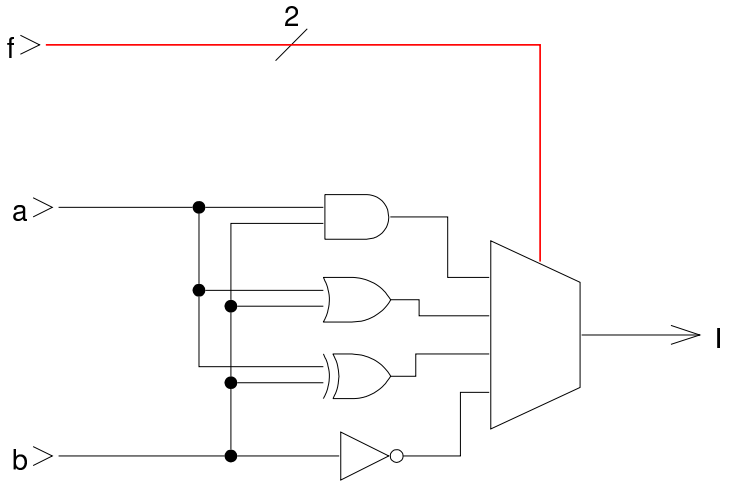


Con l'ausilio di un multiplatore è possibile ottenere facilmente un'unità logica, in grado di eseguire le operazioni logiche comuni, scegliendo quella desiderata attraverso un valore di selezione. Per esempio, si potrebbe realizzare un circuito che svolge il compito schematizzato in questo disegno:



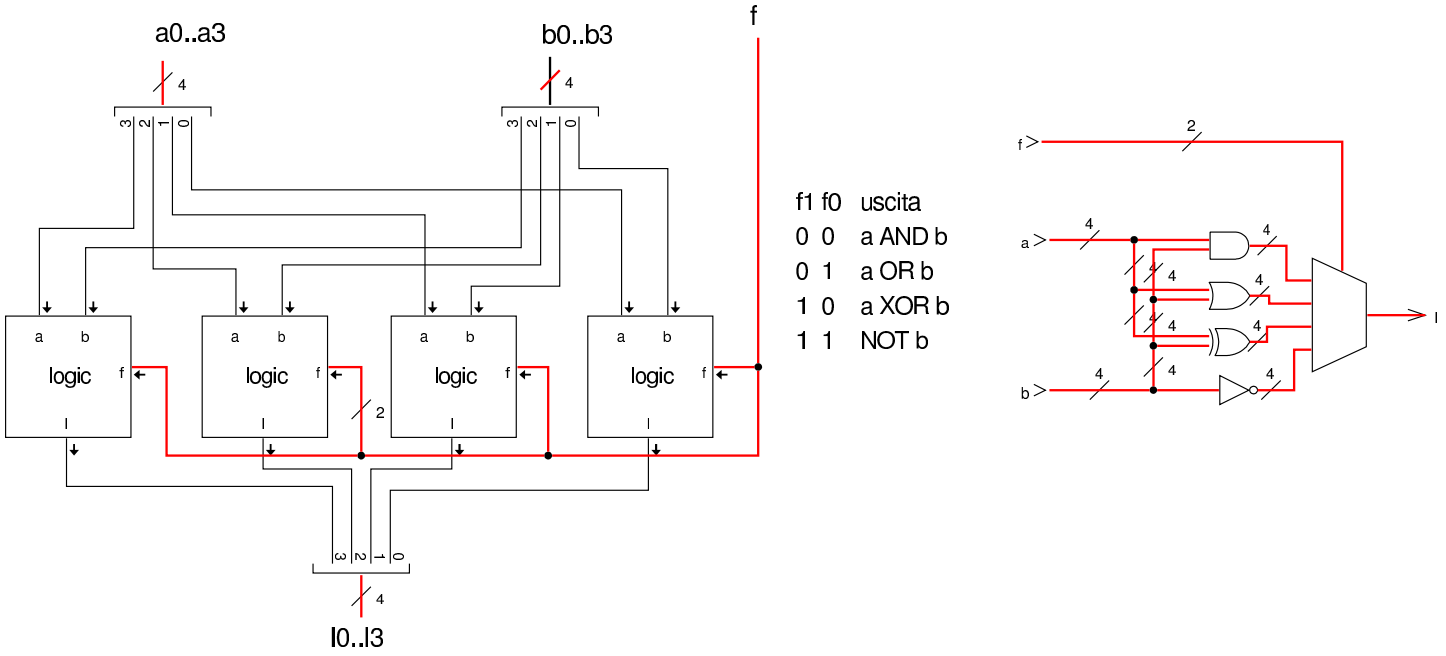
f1	f0	uscita
0	0	a AND b
0	1	a OR b
1	0	a XOR b
1	1	NOT b

Figura u98.26. Esempio di realizzazione di un'unità logica con quattro opzioni.



Le unità logiche di questo tipo sono utili se si applicano a coppie di ingressi che dispongono, ciascuno, di più bit. Per ottenere questo risultato basta abbinare delle unità semplici, a una sola uscita.

Figura u98.27. Le unità logiche possono essere messe in parallelo per operare su interi binari a più cifre. Lo schema di destra è identico dal punto di vista circuitale, ma utilizza un modo compatto per rappresentare la duplicazione delle porte logiche in parallelo.



Scorrimento

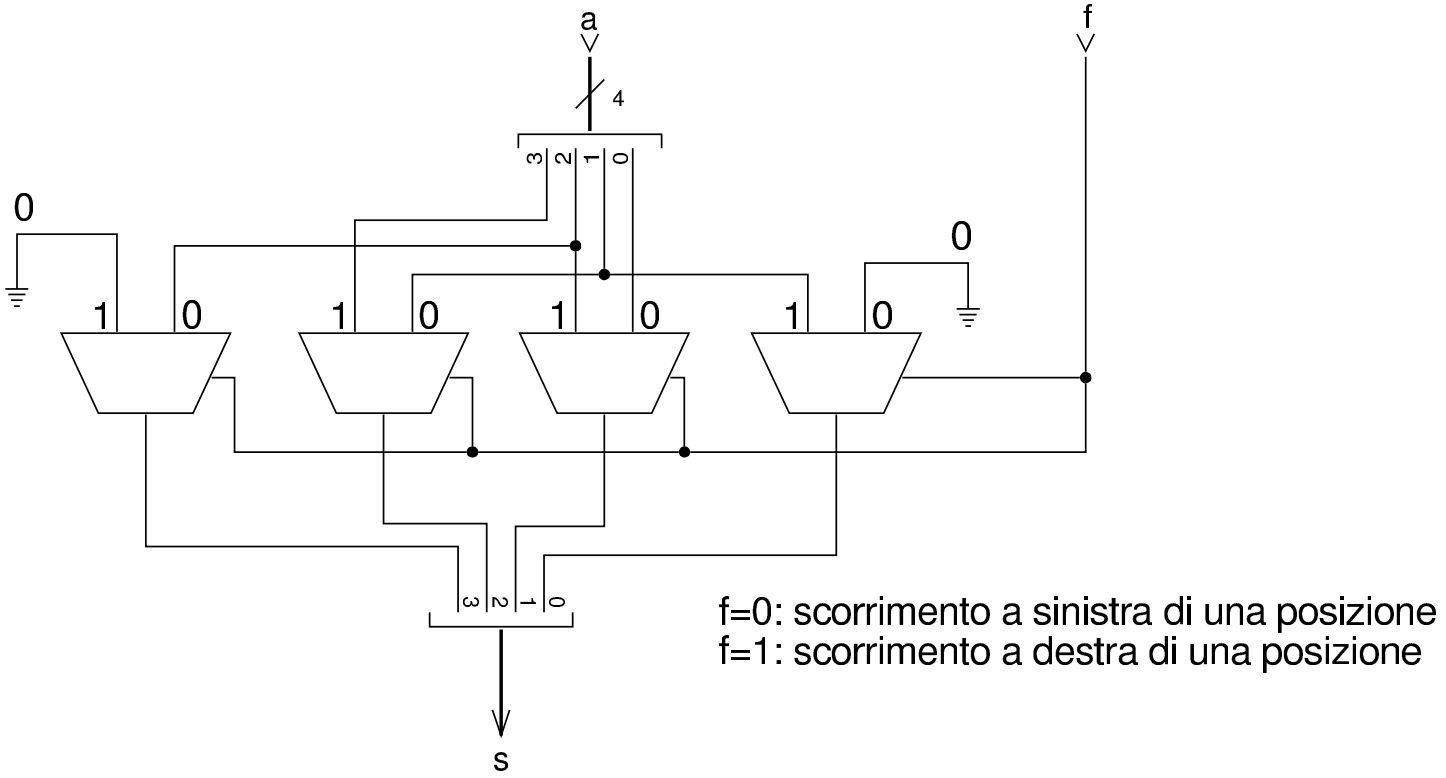
Un tipo di operazione molto importante che si può realizzare con dei circuiti combinatori, è lo scorrimento dei bit. In pratica si ha un gruppo di n ingressi ordinati, da 0 a $n-1$, un gruppo di n uscite, ordinate nello stesso modo; eventualmente ci può essere un riporto in ingresso e uno in uscita. Spesso si considera lo spostamento di una sola unità, perché si possono ottenere spostamenti di più unità ripetendo la stessa operazione ciclicamente.

Il tipo più semplice di scorrimento è quello logico, nel quale lo spostamento a destra fa inserire uno zero nella posizione più significativa, mentre quello a sinistra lo fa entrare dalla parte destra. Nelle

figure successive, per ottenere il valore zero si vede semplicemente un collegamento a massa, la quale si intende essere implicitamente al potenziale corrispondente allo zero.

1	1	1	0	*valore originale*	1	1	1	0	*valore originale*
1	1	0	0	*scorrimento logico a sinistra*	0	1	1	1	*scorrimento logico a destra*

Figura u98.29. Scorrimento logico, a sinistra o a destra, di quattro cifre binarie, con l'uso di multiplatori a due ingressi. Il verso dello scorrimento viene stabilito dal valore fornito nell'ingresso ***f***. Lo scorrimento fa perdere una cifra e fa inserire uno zero dal lato opposto.



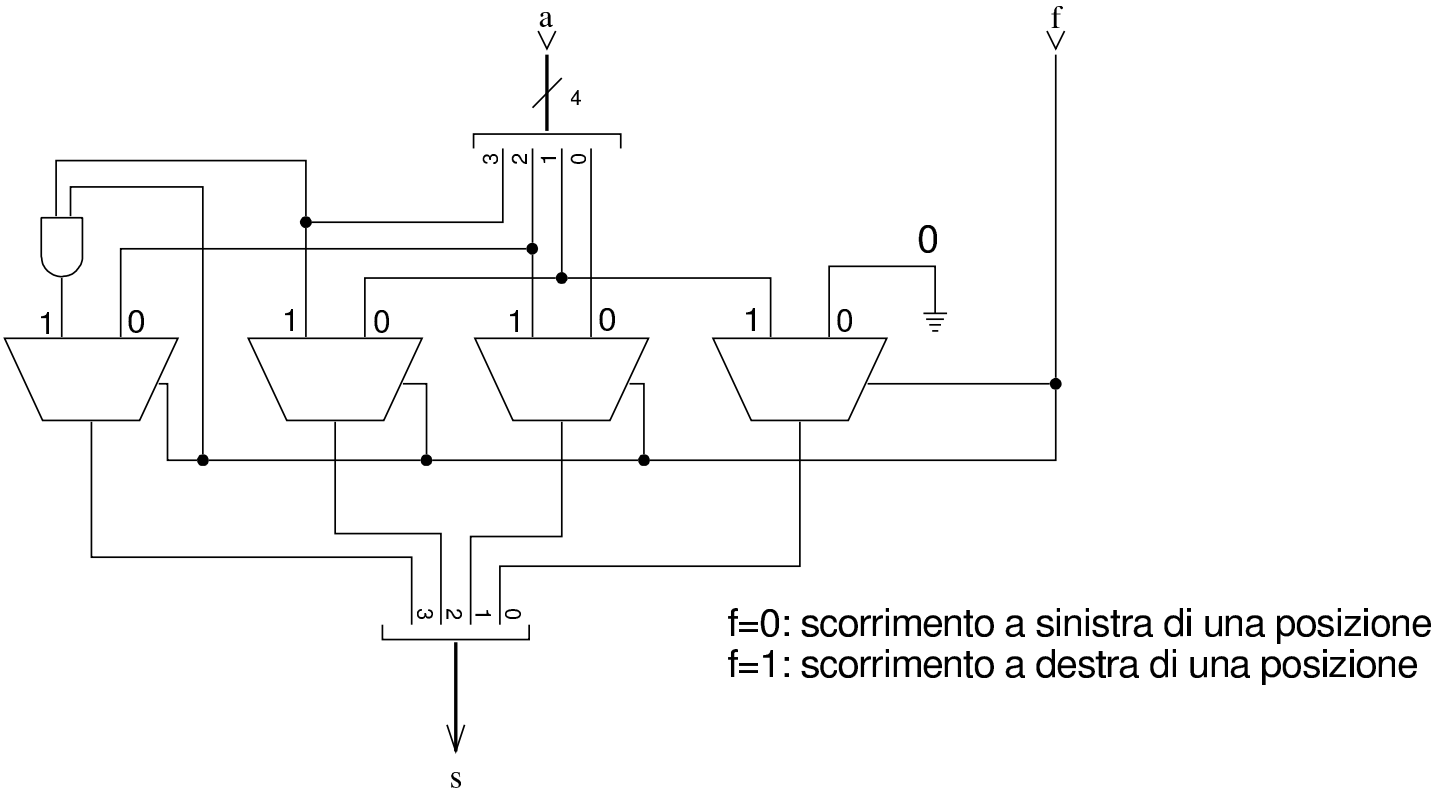
Lo scorrimento aritmetico avviene come lo scorrimento logico, con la differenza che lo scorrimento a destra deve mantenere inalterato il segno. In pratica, nello scorrimento a destra di tipo aritmetico, la cifra che viene immessa nella posizione più significativa, è la copia di quella che era originariamente in quella posizione. Lo scorrimento

aritmetico si chiama così perché corrisponde alla moltiplicazione o alla divisione per due (la base di numerazione); va però osservato che lo scorrimento a sinistra può provocare un'inversione indesiderata di segno.

1	1	1	0	*valore originale*	1	1	1	0	*valore originale*
1	1	0	0	*scorrimento aritmetico a sinistra*	1	1	1	1	*scorrimento aritmetico a destra*
0	1	1	0	*valore originale*	0	1	1	0	*valore originale*
1	1	0	0	*scorrimento aritmetico a sinistra (*)*	0	0	1	1	*scorrimento aritmetico a destra*
1	0	1	0	*valore originale*	0	1	1	0	*valore originale*
0	1	0	0	*scorrimento aritmetico a sinistra (*)*	0	0	1	1	*scorrimento aritmetico a destra*

(*) nello scorrimento a sinistra si può produrre un'inversione di segno (overflow)

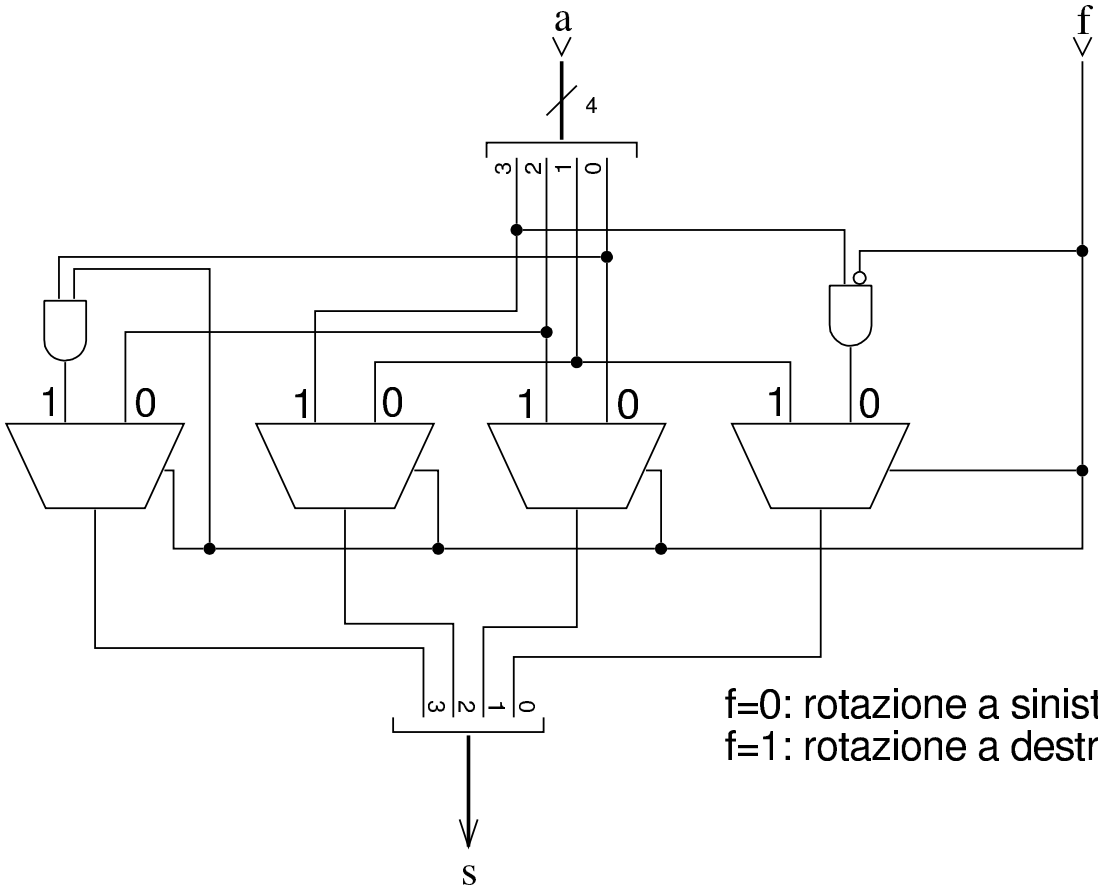
Figura u98.31. Scorrimento aritmetico, a sinistra o a destra, di quattro cifre binarie, con l'uso di multiplatori a due ingressi. Il verso dello scorrimento viene stabilito dal valore fornito nell'ingresso ***f***.



Lo scorrimento circolare comporta il recupero della cifra che fuoriesce da una parte, dal lato opposto, sia nello scorrimento a sinistra, sia in quello a destra.

1	1	1	0	*valore originale*		1	1	1	0	*valore originale*
1	1	0	1	*rotazione a sinistra*		0	1	1	1	*rotazione a destra*
0	1	1	0	*valore originale*		0	1	1	0	*valore originale*
1	1	0	0	*rotazione a sinistra*		0	0	1	1	*rotazione a destra*
1	0	1	0	*valore originale*		1	0	1	0	*valore originale*
0	1	0	1	*rotazione a sinistra*		0	1	0	1	*rotazione a destra*

Figura u98.33. Scorrimento circolare di quattro cifre binarie, con l'uso di multiplatori a due ingressi. Il verso dello scorrimento viene stabilito dal valore fornito nell'ingresso ***f***.



Lo scorrimento circolare con riporto, utilizza il riporto preesistente per la cifra da immettere, da un lato o dall'altro, mentre mette nel riporto in uscita la cifra che viene espulsa.

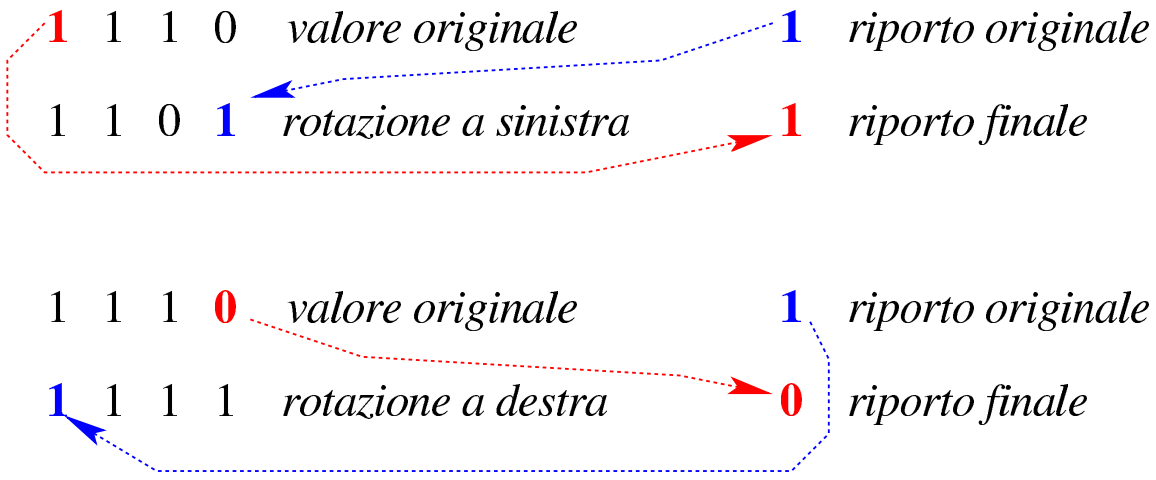
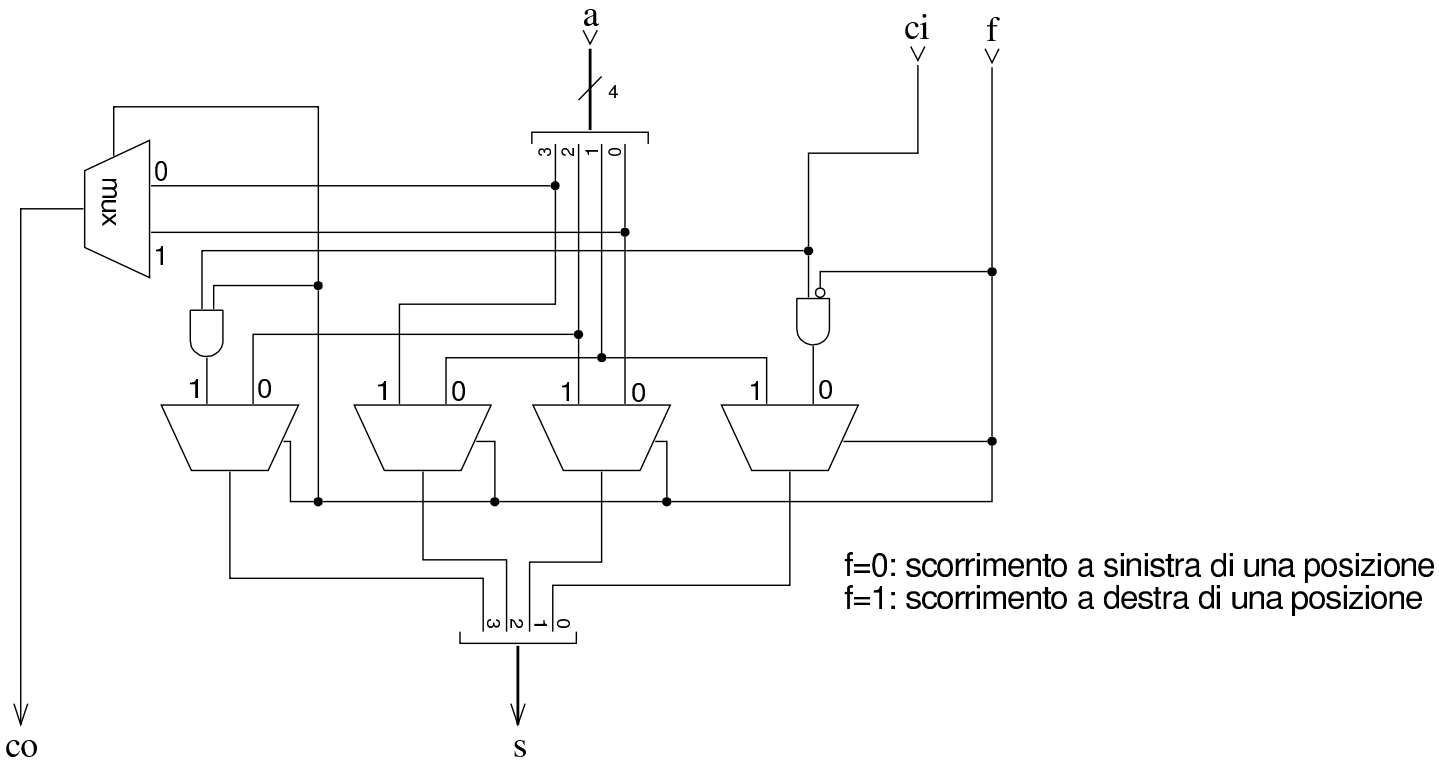


Figura u98.35. Scorrimento circolare con riporto, di quattro cifre binarie, con l'uso di multiplatori a due ingressi. Il verso dello scorrimento viene stabilito dal valore fornito nell'ingresso ***f***.



Addizionatore

L'addizione in binario, eseguita con la stessa procedura consueta per il sistema di numerazione decimale, non genera mai un riporto superiore a uno. Lo si può verificare facilmente attraverso la tabella successiva.

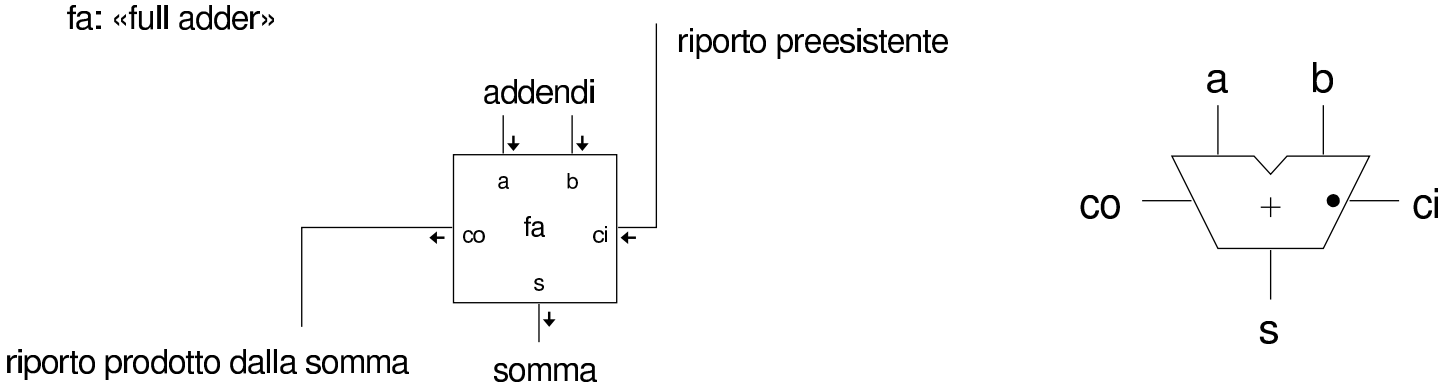
Tabella u98.36. Addizione binaria.

primo addendo	secondo addendo	riporto preesistente	riporto generato	risultato
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1

primo addendo	secondo addendo	riporto preesistente	riporto generato	risultato
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Pertanto, il circuito combinatorio che può svolgere questo lavoro deve avere tre ingressi (primo addendo, secondo addendo, riporto preesistente) e due uscite (risultato e riporto generato). Di solito si usa la lettera ***c*** (*carry*) per indicare un riporto; in questo caso si distingue tra riporto in ingresso (***ci***) e riporto in uscita (***co***).

Figura u98.37. Schema a blocchi di un addizionatore. La sigla «fa» sta per *full adder*, ovvero «addizionatore completo».



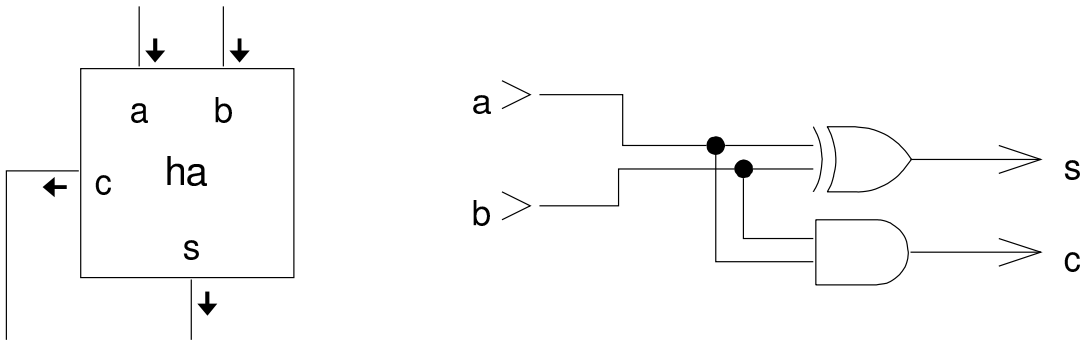
Si considera tradizionalmente che l'addizionatore completo (quello che si vede nello schema appena mostrato) sia costituito da due moduli più piccoli, noti come ***semiaddizionatori***, ovvero *half adder*. Il semiaddizionatore è un circuito combinatorio con due ingressi, ***a*** e ***b***, corrispondenti alle variabili da sommare, e due uscite, ***s*** e ***c***, corrispondenti al risultato della somma e al suo riporto.

Tabella u98.38. Tabella di verità per il semiaddizionatore.

a	*b*	*c*	*s*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

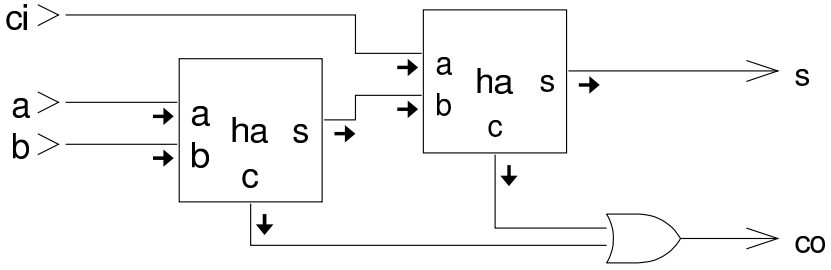
Si può intuire facilmente che, nel semiaddizionatore, l'uscita ***s*** si ottenga con una porta XOR e che il riporto si ottenga con una porta AND.

Figura u98.39. Schema a blocchi ed esempio di realizzazione di un semiaddizionatore. La sigla «ha» sta per *half adder*.



Per arrivare a ottenere l'addizionatore completo, occorre sommare anche il riporto precedente.

Figura u98.40. Schema a blocchi dell'addizionatore completo, ottenuto attraverso due semiaddizionatori e realizzazione conseguente.



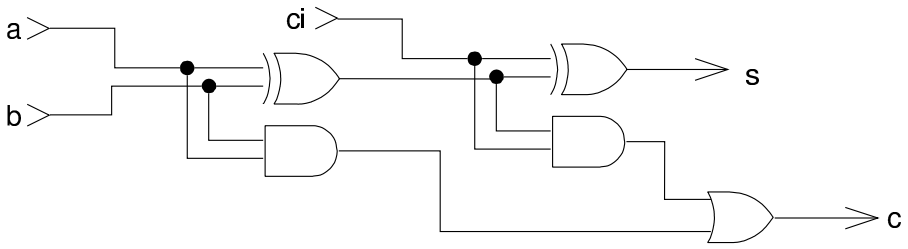
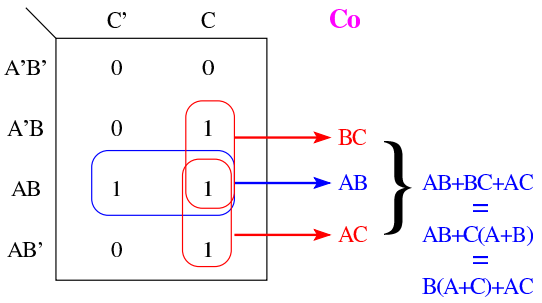
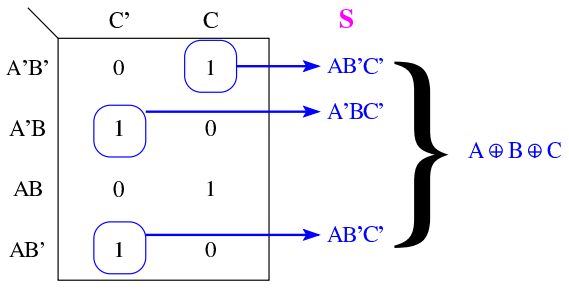
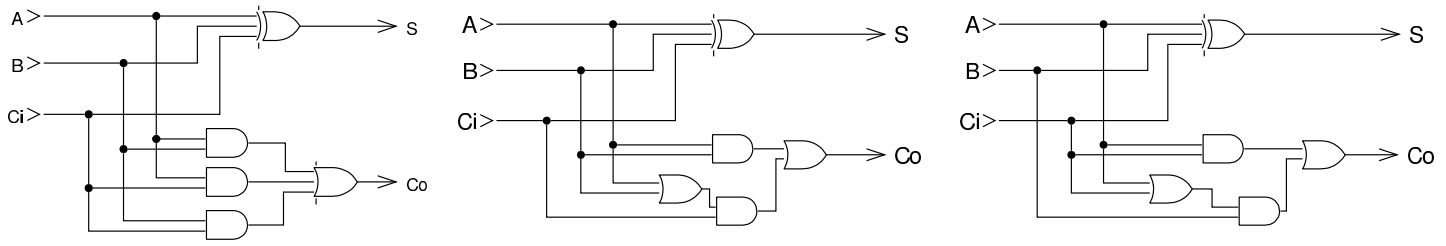


Figura u98.41. Soluzioni alternative per l'addizionatore completo, con l'aiuto delle mappe di Karnaugh.

A	B	C	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



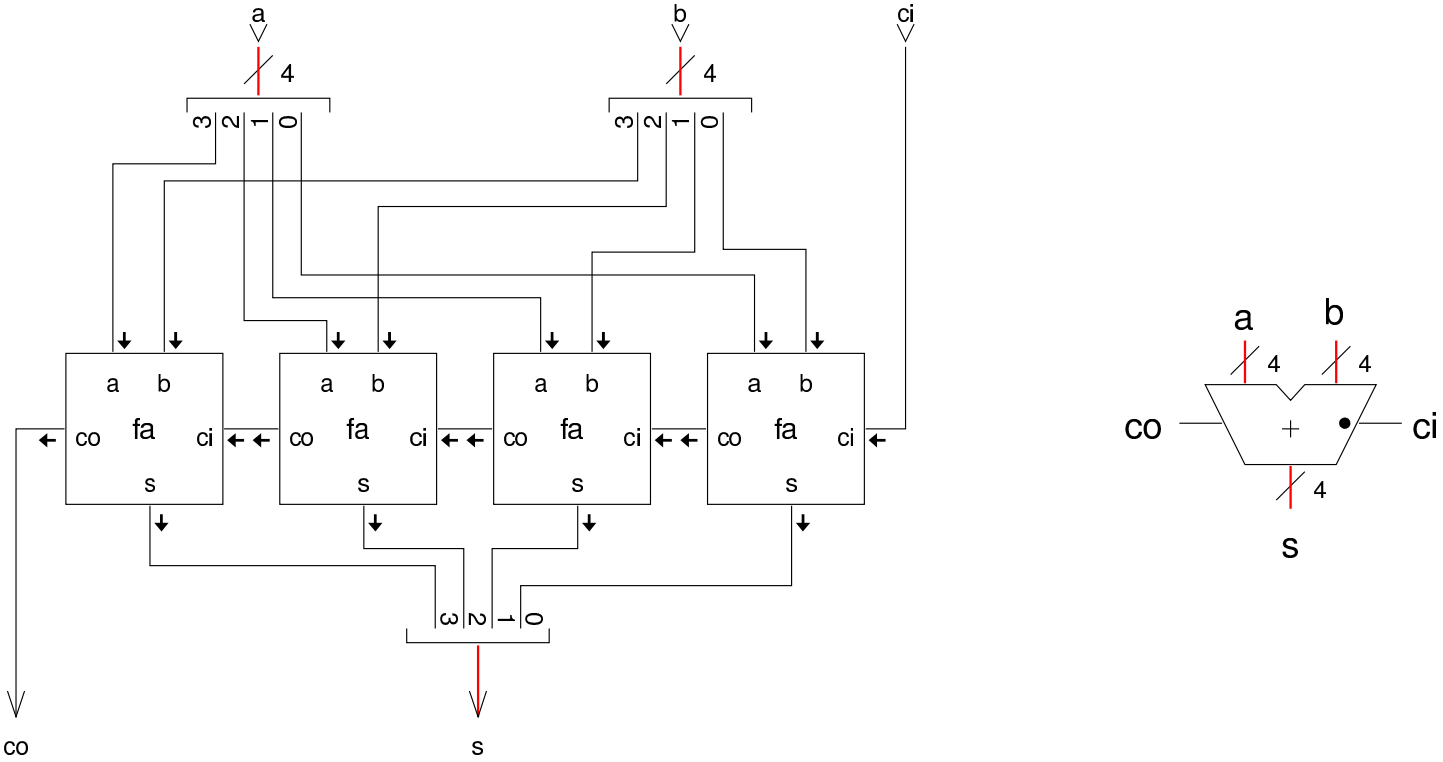




Perché l'addizionatore sia utile, è necessario che intervenga su un numero binario composto da diverse cifre, pertanto occorre assemblare in parallelo più addizionatori, passando opportunamente il riporto, dalla cifra meno significativa a quella più significativa, una

cifra alla volta.

Figura u98.42. Addizionatore a quattro cifre, costruito secondo la modalità della «propagazione del resto». Nella parte destra si vede una rappresentazione compatta di un addizionatore, senza specificare in che modo sia effettuato il calcolo.



Il circuito dell'addizionatore descritto, opera correttamente per la somma di numeri positivi o negativi, quando i numeri negativi si rappresentano attraverso la tecnica del complemento a due.

Il complemento a due che serve a trasformare il sottraendo, si ottiene con il complemento a uno del suo valore, sommandogli una unità attraverso il riporto in ingresso.

Sottrazione

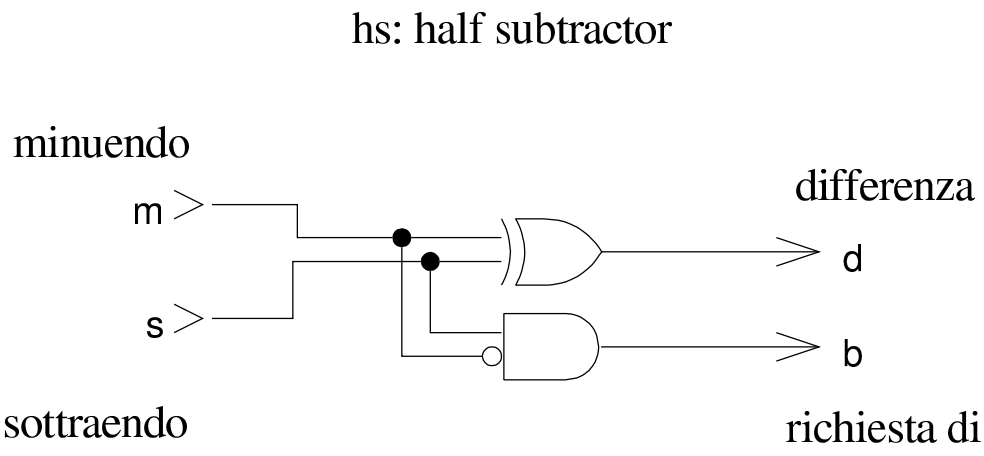
«

Per la sottrazione, si parte, anche in questo caso, dal semi-sottrattore, ovvero *half subtractor*. Trattandosi di una sottrazione, è necessario distinguere tra gli operandi il minuendo e il sottraendo; in pratica, negli esempi vengono usate queste variabili: $d=m-s$.

Tabella u98.43. Tabella di verità per il semisottrattore.

m	s	b	d
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Figura u98.44. Semisottrattore.



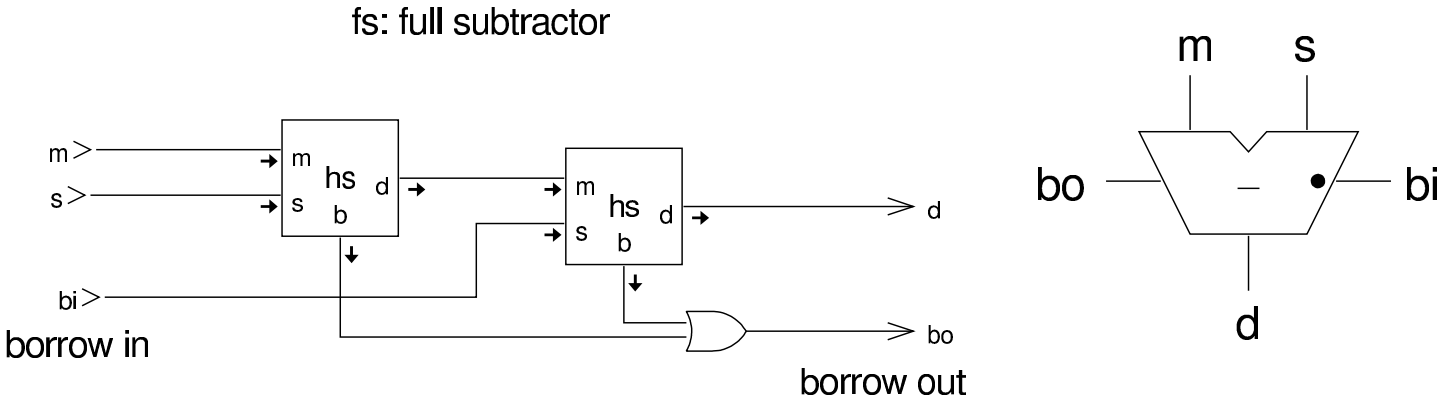
Al posto del riporto c'è un'uscita denominata *borrow* che rappresenta la richiesta del prestito di una cifra. Questa uscita diventa attiva quando il minuendo è pari a zero, mentre il sottraendo è pari a uno; ovvero quando il sottraendo è maggiore del minuendo. Il sottrattore completo ha un ingresso in più, costituito dalla richiesta di una cifra proveniente dallo stadio precedente; per gestire questo nuovo

ingresso si possono usare due semisottrattori, dove il secondo sottrae al risultato del primo la richiesta di prestito.

Tabella u98.45. Tabella di verità per il semisottrattore.

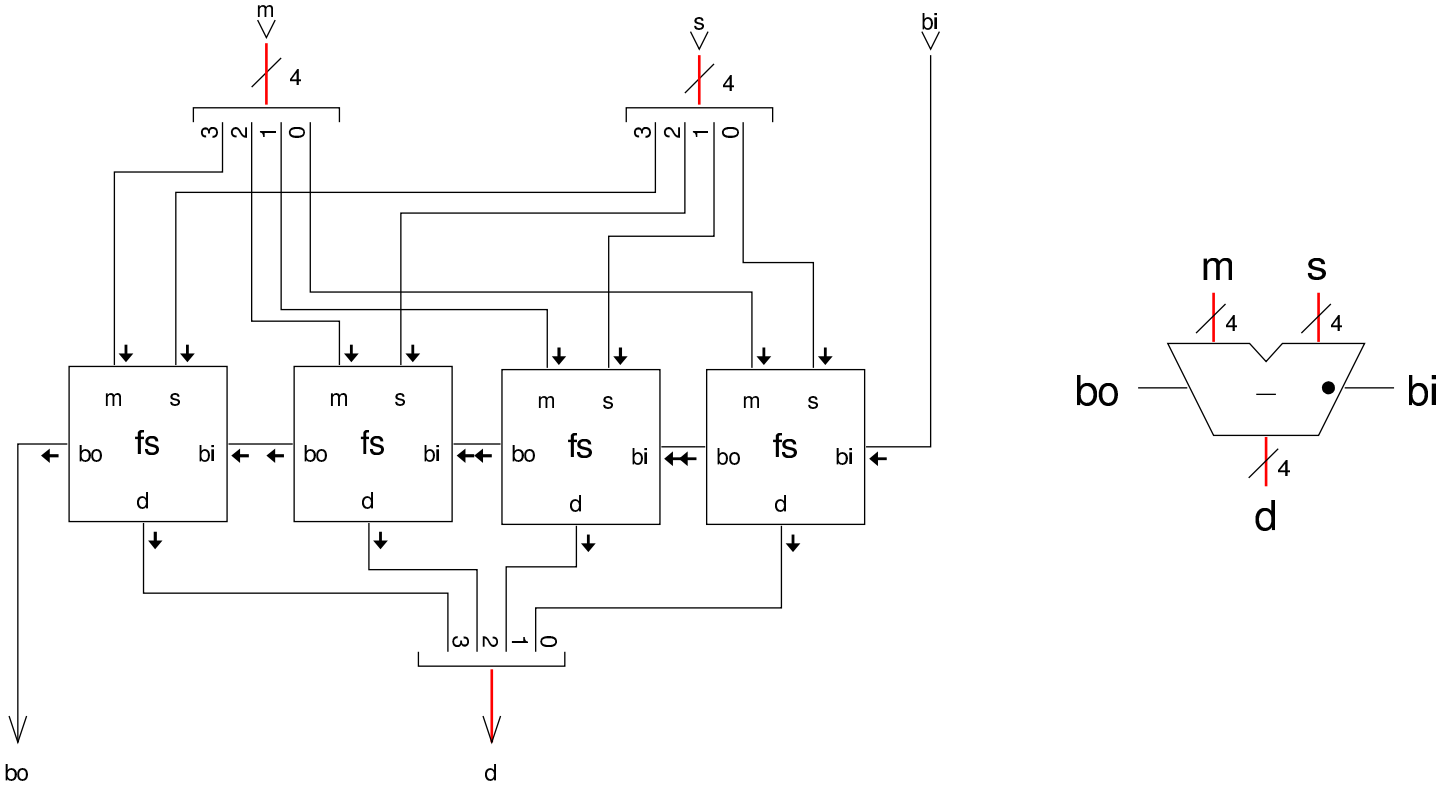
m	s	b_i	b_o	d
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Figura u98.46. Sottrattore completo.



Come nel caso della somma, si possono creare delle catene di sottrattori completi; tuttavia, nel confronto con il circuito della somma, modificato per effettuare la sottrazione, la linea *borrow* (quella della richiesta del prestito), funziona in modo inverso, in quanto se attiva, rappresenta una cifra da togliere.

Figura u98.47. Sottrazione su più bit simultaneamente.

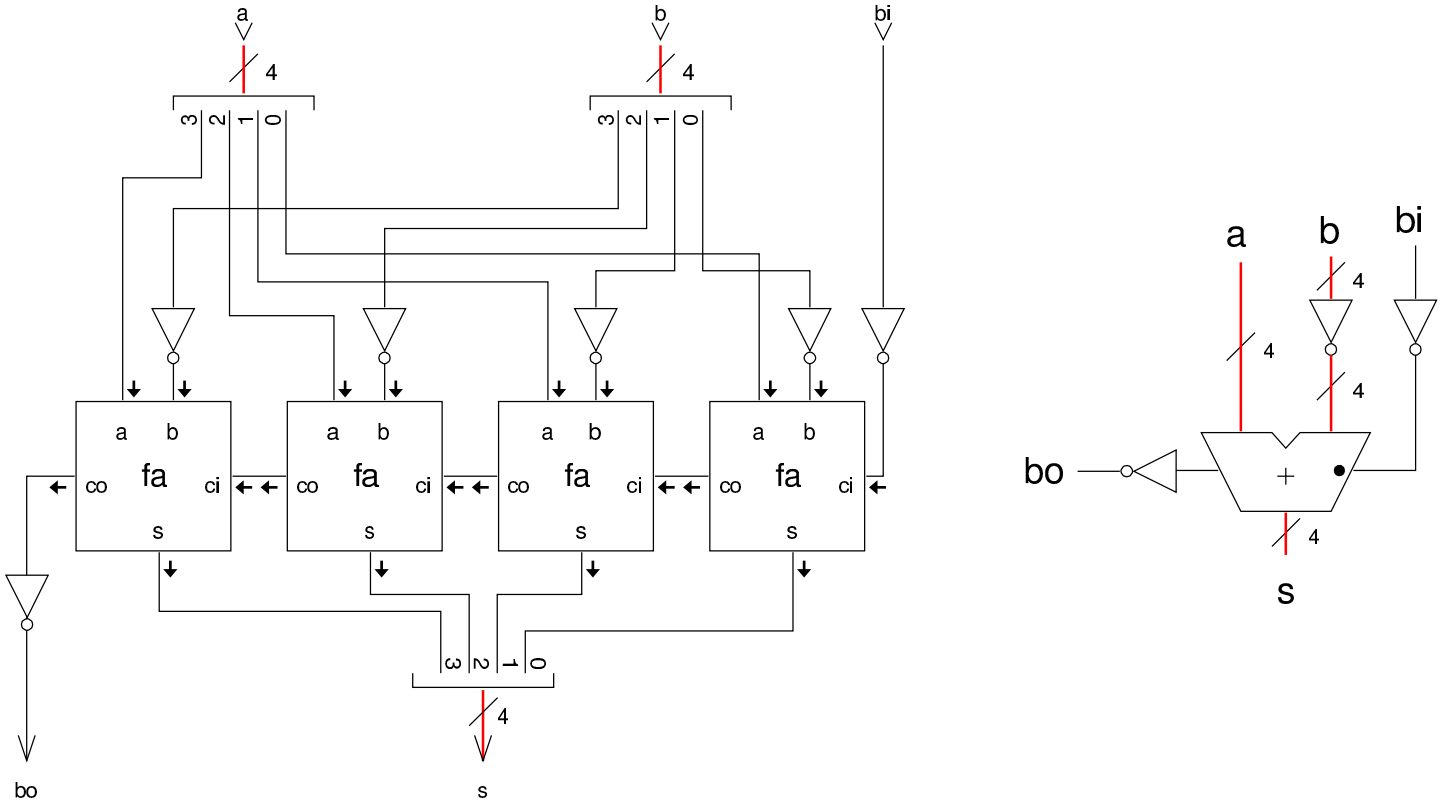


Somma e sottrazione assieme

«

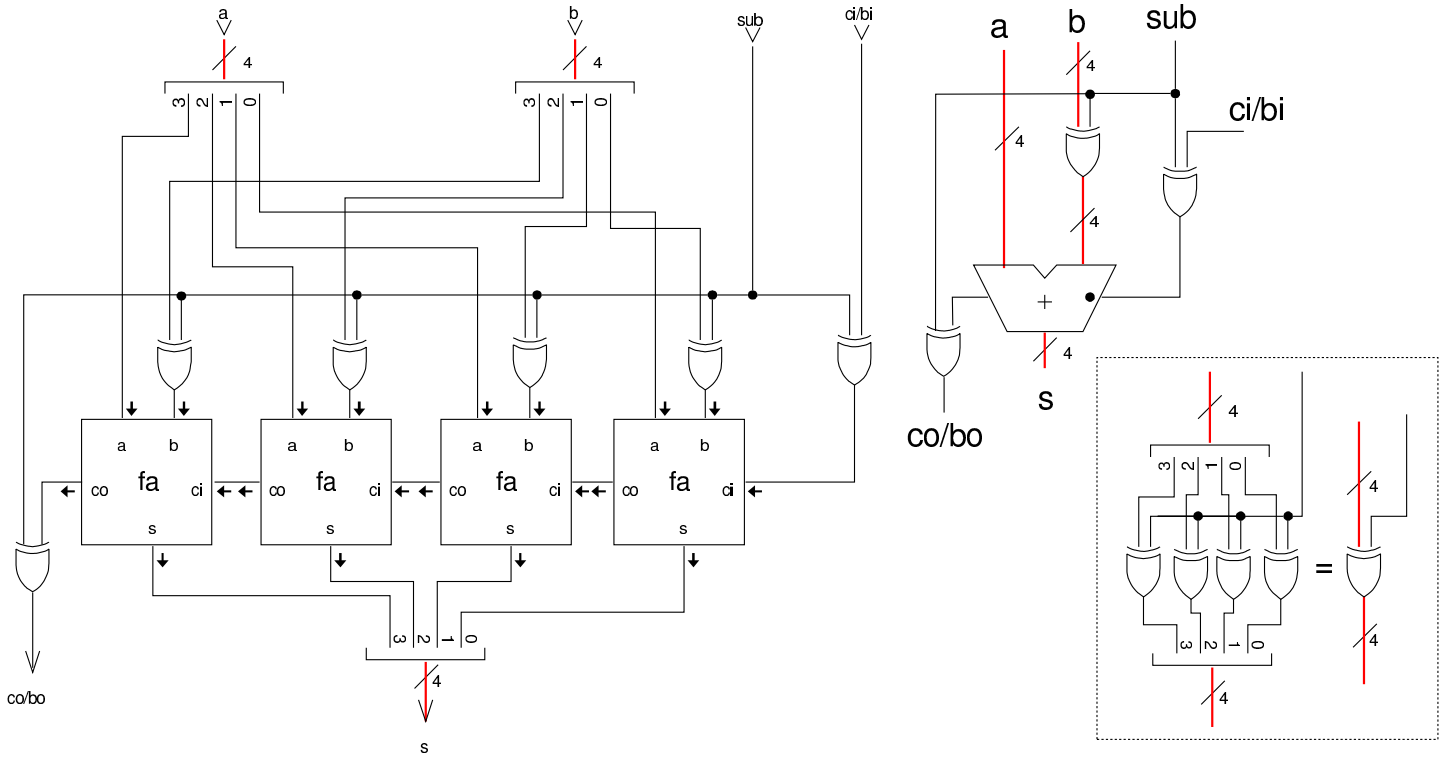
Il circuito dell'addizionatore opera correttamente per la somma di numeri positivi o negativi, quando i numeri negativi si rappresentano attraverso la tecnica del complemento a due; pertanto, per trasformare l'addizionatore in un circuito che invece sottrae uno dei due operandi, è sufficiente calcolare per quello il complemento a due e per ottenerlo si devono invertire tutti gli ingressi, compreso il riporto. Va ricordato che nella sottrazione, il riporto assume il significato della richiesta di una cifra.

Figura u98.48. Sottrattore a quattro cifre: in questo caso si tratta precisamente di $a-b$. Nella parte destra si vede una rappresentazione compatta della stessa cosa, dove il simbolo della porta NOT va intesa come un'abbreviazione di quattro porte NOT indipendenti.



Eventualmente, si trasforma facilmente il circuito combinatorio in un complesso unico, in grado di sommare o di sottrarre, sfruttando una porta XOR al posto della porta NOT, aggiungendo un ingresso ulteriore che permetta di stabilire se si esegue una somma o se l'operando stabilito va invece sottratto.

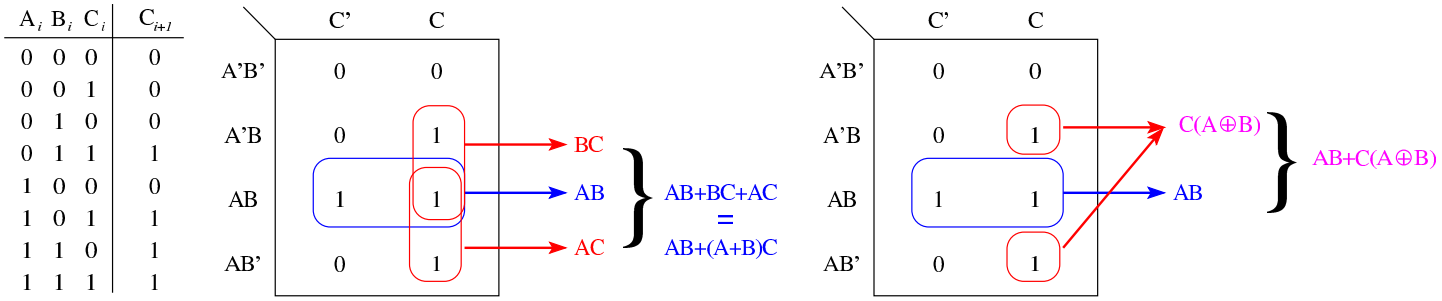
Figura u98.49. Somma o sottrazione a quattro cifre. Nella parte destra si vede una rappresentazione compatta della stessa cosa, dove il simbolo della porta XOR va inteso come un'abbreviazione di quattro porte XOR collegate assieme come si vede nel piccolo riquadro esplicativo.



Riporto anticipato

Nella somma di una quantità significativa di bit, la propagazione del riporto richiede un tempo relativamente elevato, dato che la somma di una certa cifra è corretta solo se è già avvenuta la somma di quelle che la precedono. Per poter accelerare l'esecuzione della somma, è necessario che per ogni cifra si possa sapere, nel tempo più breve possibile, qual è il riporto generato fino allo stadio precedente. Nelle espressioni successive, le variabili A_i, B_i e C_i, rappresentano i due addendi e il riporto in ingresso dello stadio i; pertanto, il riporto generato da questo stadio è rappresentato dalla variabile C_{i+1}.

Figura u98.50. Sintesi alternative del calcolo del riporto.



Nella figura si vede che il riporto si può sintetizzare in vari modi e in uno di questi appare anche l'uso dell'operatore XOR. Di quelle mostrate nella figura si scelgono due espressioni equivalenti:

$$C_{i+1} = A_iB_i+(A_i+B_i)C_i = A_iB_i+(A_i \oplus B_i)C_i$$

Da queste espressioni, si dichiarano due variabili nuove, $\boldsymbol{G_i}$ e $\boldsymbol{P_i}$, le quali stanno rispettivamente per «generazione» e «propagazione», per cui si definisce che il riporto $\boldsymbol{C_{i+1}}$ è prodotto come funzione delle variabili $\boldsymbol{G_i}$, $\boldsymbol{P_i}$ e $\boldsymbol{C_i}$.

$$C_{i+1} = G_i+P_iC_i$$

È evidente che $\boldsymbol{G_i}$ equivale a $\boldsymbol{A_iB_i}$, mentre $\boldsymbol{P_i}$ può essere considerata pari a $\boldsymbol{A_i+B_i}$ oppure $\boldsymbol{A_i \oplus B_i}$, indifferentemente. Se il primo riporto generato ($\boldsymbol{C_1}$) si può ottenere come $\boldsymbol{C_1=G_0+P_0C_0}$, il secondo si ottiene come $\boldsymbol{C_2=G_1+P_1(G_0+C_0P_0)}$, e di conseguenza si può proseguire per determinare i riporti successivi. Vengono mostrate le soluzioni per i primi quattro riporti:

$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1$$
$$C_2 = G_1 + P_1 (G_0 + P_0 C_0)$$
$$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0$$
$$C_3 = G_2 + P_2 C_2$$
$$C_3 = G_2 + P_2 (G_1 + P_1 G_0 + P_1 P_0 C_0)$$
$$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$
$$C_4 = G_3 + P_3 C_3$$
$$C_4 = G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0)$$
$$C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$$

Per semplificare le espressioni, si definiscono $\boldsymbol{P_n}$ e $\boldsymbol{G_n}$ nel modo seguente:

$$P_n = P_{n-1} P_{n-2} P_{n-1} \cdots P_1 P_0$$

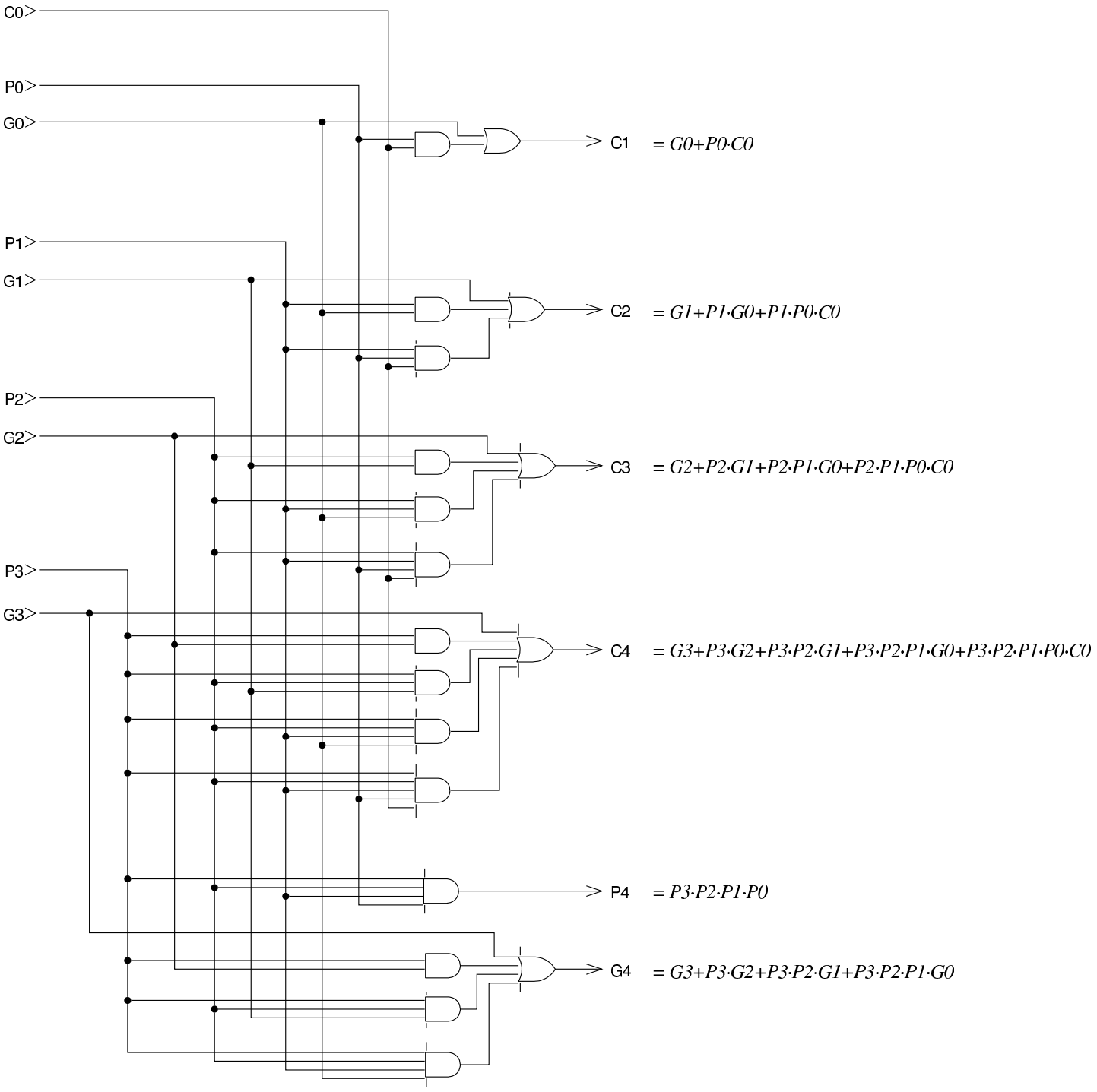
$$G_n = G_{n-1} + P_{n-1} G_{n-2} + P_{n-2} P_{n-1} G_{n-3} + \cdots + P_{n-2} P_{n-1} \cdots P_1 G_0$$

Avendo definito ciò, il riporto $\boldsymbol{C_n}$ si può definire come:

$$C_n = G_n + P_n C_0$$

La figura successiva, mostra un circuito combinatorio che determina i riporti di quattro cifre binarie, partendo dal riporto iniziale e dai valori di $\boldsymbol{B_{3..0}}$ e $\boldsymbol{G_{3..0}}$, come descritto dalle equazioni che definiscono questa relazione. Il disegno contiene anche la logica necessaria a determinare il valore di $\boldsymbol{B_4}$ e $\boldsymbol{G_4}$ che possono servire per collegare assieme più moduli di questo tipo.

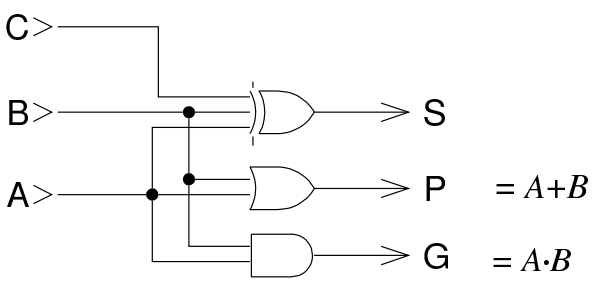
Figura u98.56. Schema per la determinazione di quattro riporti, a partire dal riporto iniziale (C_0) e dai valori di P_i e G_i, come descritto nelle equazioni logiche.

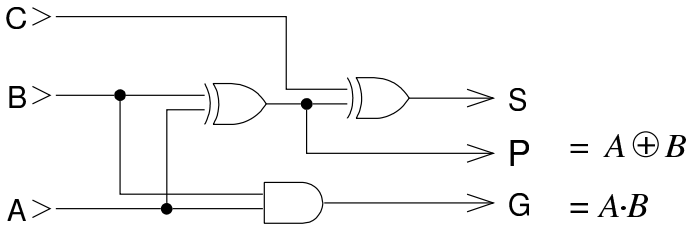


Avendo la necessità di disporre delle uscite G e P, si può sintetizzare un modulo per l'addizione privo della funzione che determina il

riporto in uscita, dal momento che questo compito viene affidato al modulo che si vede nella figura precedente. Ci sono due soluzioni alternative, nelle quali il valore di ***P*** viene determinato nei due modi diversi già descritti con le tabelle di Karnaugh.

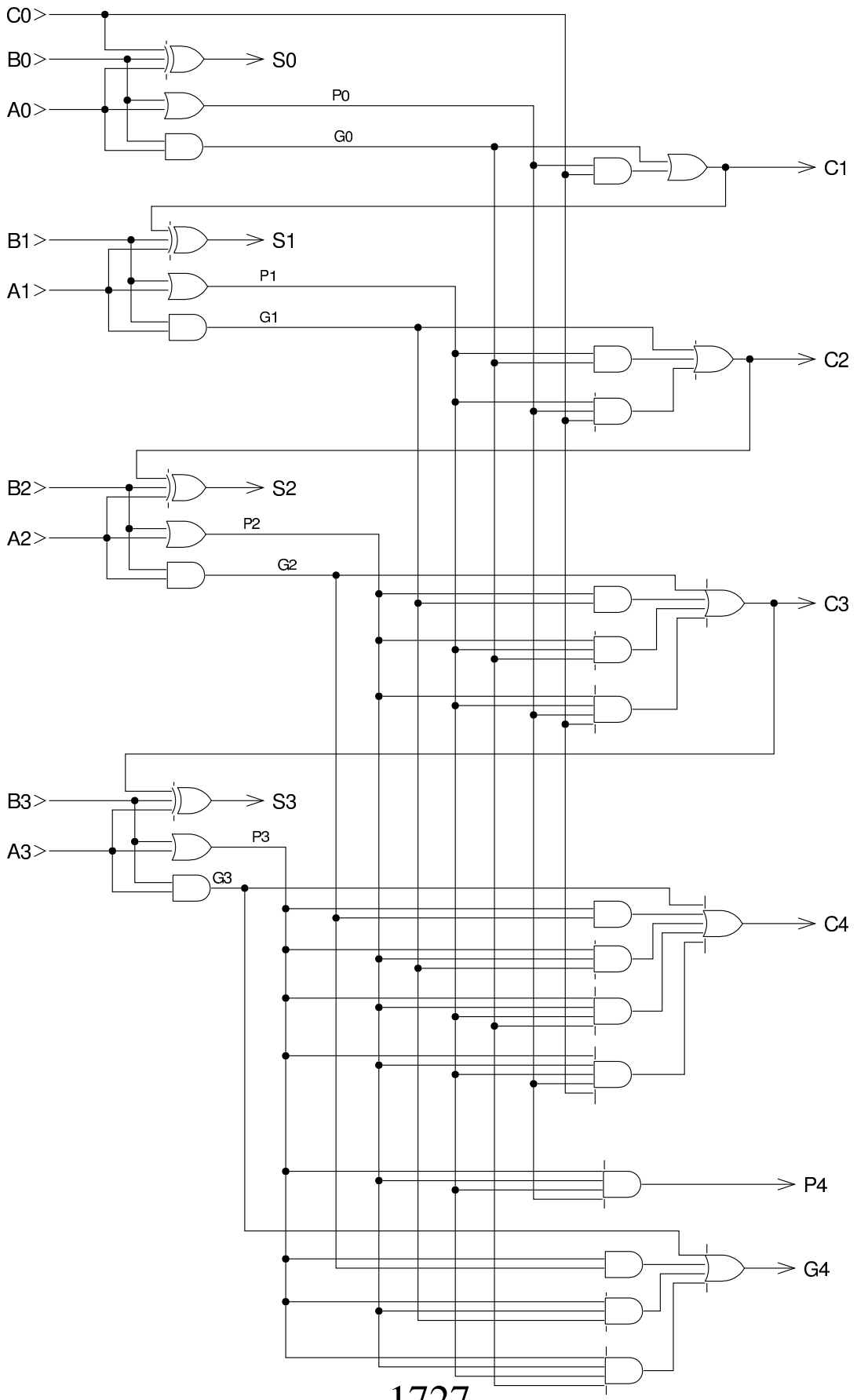
Figura u98.57. Moduli per l'addizione con le uscite ***P*** e ***G***, ma privi dell'uscita con il riporto per la cifra successiva.





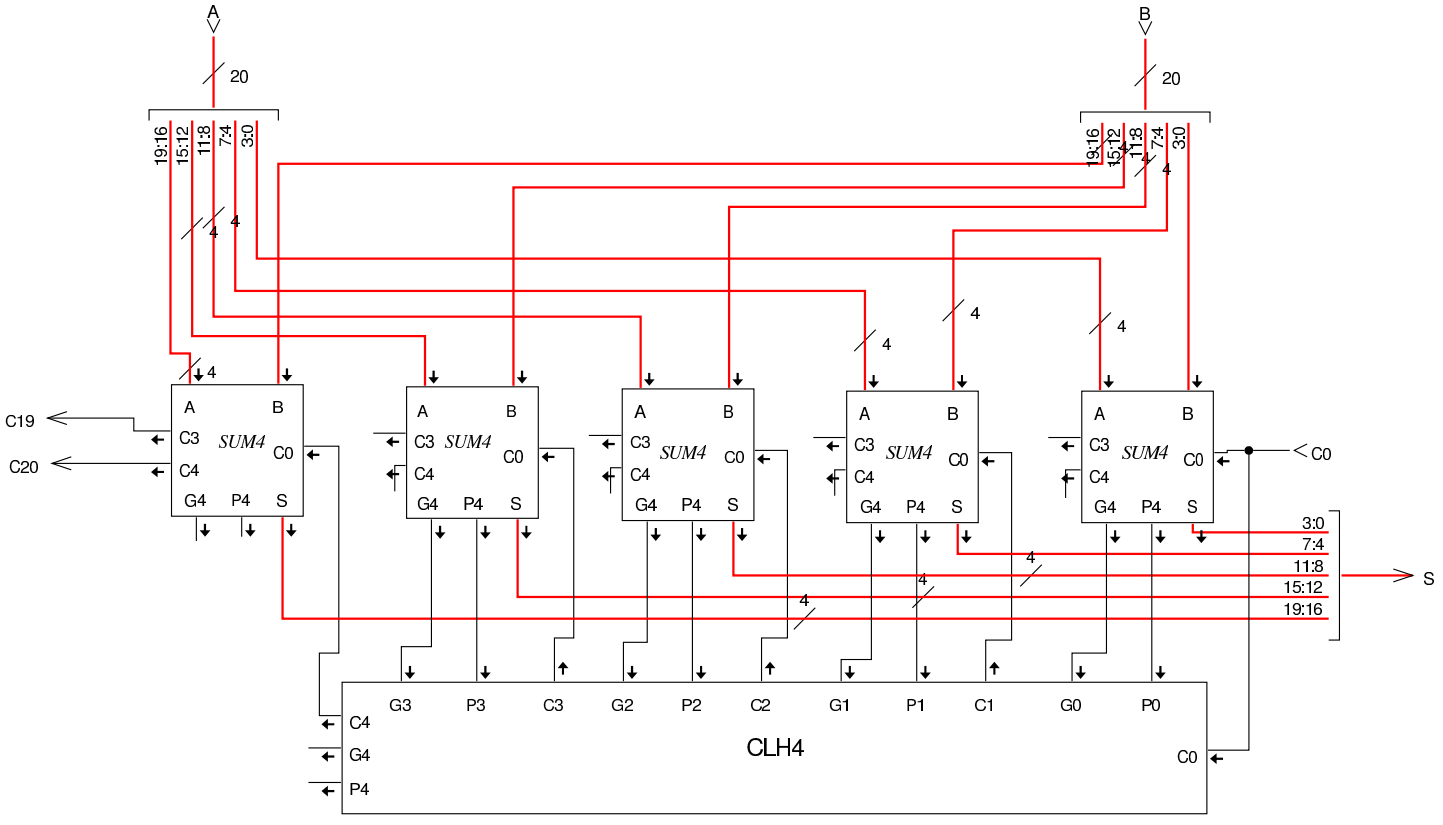
La figura successiva mostra un addizionatore a quattro cifre, dove si utilizzano i moduli di addizione e di determinazione del riporto già apparsi.

Figura u98.58. Addizionatore a quattro cifre.



È possibile collegare assieme i moduli mostrati, quando non si può disporre in partenza della quantità di cifre che servono. Questo collegamento comporta un aumento del ritardo di propagazione, ma si tratta comunque di un grande miglioramento rispetto al calcolo del riporto in cascata, come mostrato nelle sezioni precedenti. Nella figura successiva, il modulo SUM4 corrisponde allo schema di figura u98.58, dal quale si prelevano solo l'ultimo e il penultimo riporto (perché in sezioni successive viene mostrato che questi due consentono di determinare la presenza di uno straripamento); invece, il modulo CLH4 corrisponde allo schema di figura u98.56, il quale viene usato per mettere assieme i moduli di addizione.

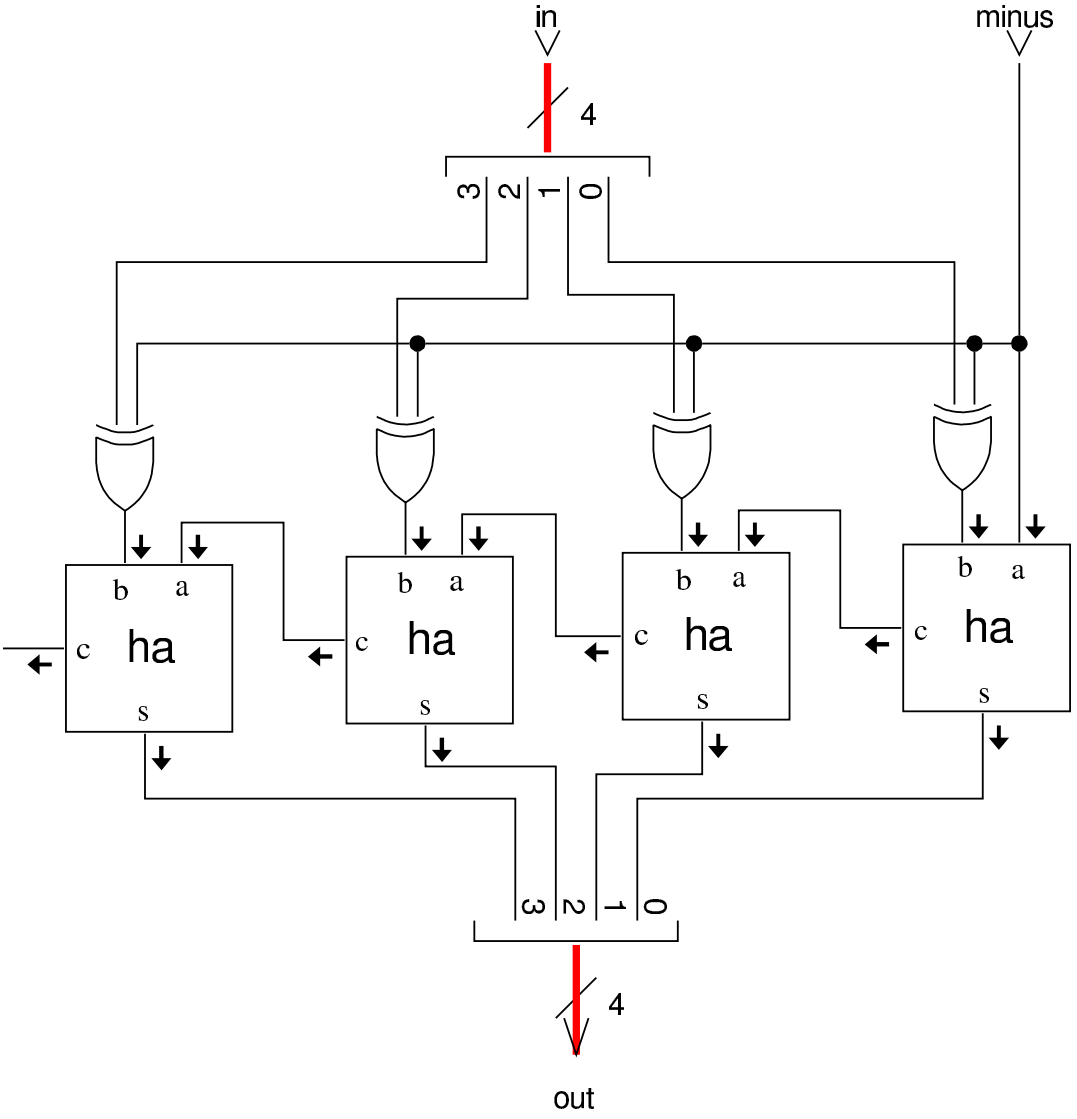
Figura u98.59. Addizionatore a 20 cifre, ottenuto partendo da moduli da quattro cifre concatenati assieme.



Complemento a due

Se si ha la necessità di invertire il segno di un numero intero, rappresentato in forma binaria, c'è bisogno di costruire un circuito che esegua il complemento a uno, invertendo le cifre binarie, sommando poi una unità per trovare il complemento a due.

Figura u98.60. Calcolo del complemento a due, corrispondente all'inversione del segno di un numero intero a quattro cifre.



L'esempio proposto nella figura mostra che il dato in ingresso, viene invertito (negato) se risulta attivo il segnale ***minus***, ma lo stesso segnale ***minus*** viene sommato, producendo alla fine il complemento a

due. Se invece il segnale ***minus*** non fosse attivo, il dato in ingresso non verrebbe alterato e non ci sarebbe l'incremento di un'unità, per cui il risultato sarebbe lo stesso, senza variazione.

Moltiplicazione

La moltiplicazione binaria è un procedimento che richiede la somma e lo scorrimento. Per poter tenere conto del segno, il calcolo andrebbe fatto come se si operasse su una quantità doppia di cifre; per esempio, se moltiplicatore e moltiplicando sono di sole quattro cifre binarie, il risultato deve poter essere di otto cifre e il calcolo andrebbe fatto come se anche gli operandi fossero di questo rango. Si osservino gli esempi che appaiono nella figura successiva.

Figura u98.61. Moltiplicazione di numeri a quattro cifre binarie, senza segno e con segno.

moltiplicazione di interi senza segno	*moltiplicando negativo, moltiplicatore positivo*	*moltiplicando positivo, moltiplicatore negativo*	*moltiplicando negativo, moltiplicatore negativo*
0 0 0 0 **1 0 1 1** ×	1 1 1 1 **1 0 1 1** ×	0 0 0 0 **0 1 1 1** ×	1 1 1 1 **1 0 1 1** ×
0 0 0 0 **1 1 0 1** =	0 0 0 0 **0 1 0 1** =	1 1 1 1 **1 1 0 1** =	1 1 1 1 **1 1 0 1** =
0 0 0 0 **1 0 1 1** +	1 1 1 1 **1 0 1 1** +	0 0 0 0 **0 1 1 1** +	1 1 1 1 **1 0 1 1** +
0 0 0 **0 0 0 0** 0 +	0 0 0 **0 0 0 0** 0 +	0 0 0 **0 0 0 0** 0 +	0 0 0 **0 0 0 0** 0 +
0 0 **1 0 1 1** 0 0 +	1 1 **1 0 1 1** 0 0 +	0 0 **0 1 1 1** 0 0 +	1 1 **1 0 1 1** 0 0 +
0 **1 0 1 1** 0 0 0 +	0 **0 0 0 0** 0 0 0 +	0 **0 1 1 1** 0 0 0 +	1 **1 0 1 1** 0 0 0 +
0 0 0 0 0 0 0 0 +	0 0 0 0 0 0 0 0 +	0 1 1 1 0 0 0 0 +	1 0 1 1 0 0 0 0 +
0 0 0 0 0 0 0 0 +	0 0 0 0 0 0 0 0 +	1 1 1 0 0 0 0 0 +	0 1 1 0 0 0 0 0 +
0 0 0 0 0 0 0 0 +	0 0 0 0 0 0 0 0 +	1 1 0 0 0 0 0 0 +	1 1 0 0 0 0 0 0 +
0 0 0 0 0 0 0 0 =	0 0 0 0 0 0 0 0 =	1 0 0 0 0 0 0 0 =	1 0 0 0 0 0 0 0 =
1 0 0 0 1 1 1 1	1 1 1 0 0 1 1 1	1 1 1 0 1 0 1 1	0 0 0 0 1 1 1 1

Nella figura, l'esempio di sinistra mostra la moltiplicazione tra 11 e 13; trattandosi di numeri privi di segno, vanno aggiungi degli zeri

nelle posizioni più significative e di conseguenza va svolta la moltiplicazione. Nel secondo esempio, i numeri vanno intesi con segno e si tratta della moltiplicazione tra −5 e +5; in questo caso, il numero che viene inteso come negativo, deve essere completato con cifre a uno, per mantenere il segno negativo, e di conseguenza la moltiplicazione ne tiene conto. Nel terzo esempio è il moltiplicatore a essere negativo: 7·−3. In tal caso è il moltiplicatore che viene completato con le cifre a uno nella parte più significativa, condizionando di conseguenza il calcolo della moltiplicazione. L'ultimo esempio è uguale al primo, con la differenza che i due valori sono intesi con segno, pertanto sono completati con cifre a uno. In conclusione, la moltiplicazione deve tenere conto del fatto che i numeri siano con segno o senza; nel caso lo siano, devono essere estesi nella porzione più significativa con la cifra necessaria a mantenere il segno che hanno.

Per risolvere il problema in forma di circuito combinatorio, occorre incrociare i valori di moltiplicando e moltiplicatore, verificando in ogni posizione utile la coincidenza di valori a uno. I due disegni successivi vanno sovrapposti idealmente, in quanto mostrano il concetto in due fasi: le uscite delle porte AND devono essere sommate verticalmente per generare il prodotto.

Figura u98.62. Intreccio tra moltiplicando e moltiplicatore, abbinato a delle porte AND.

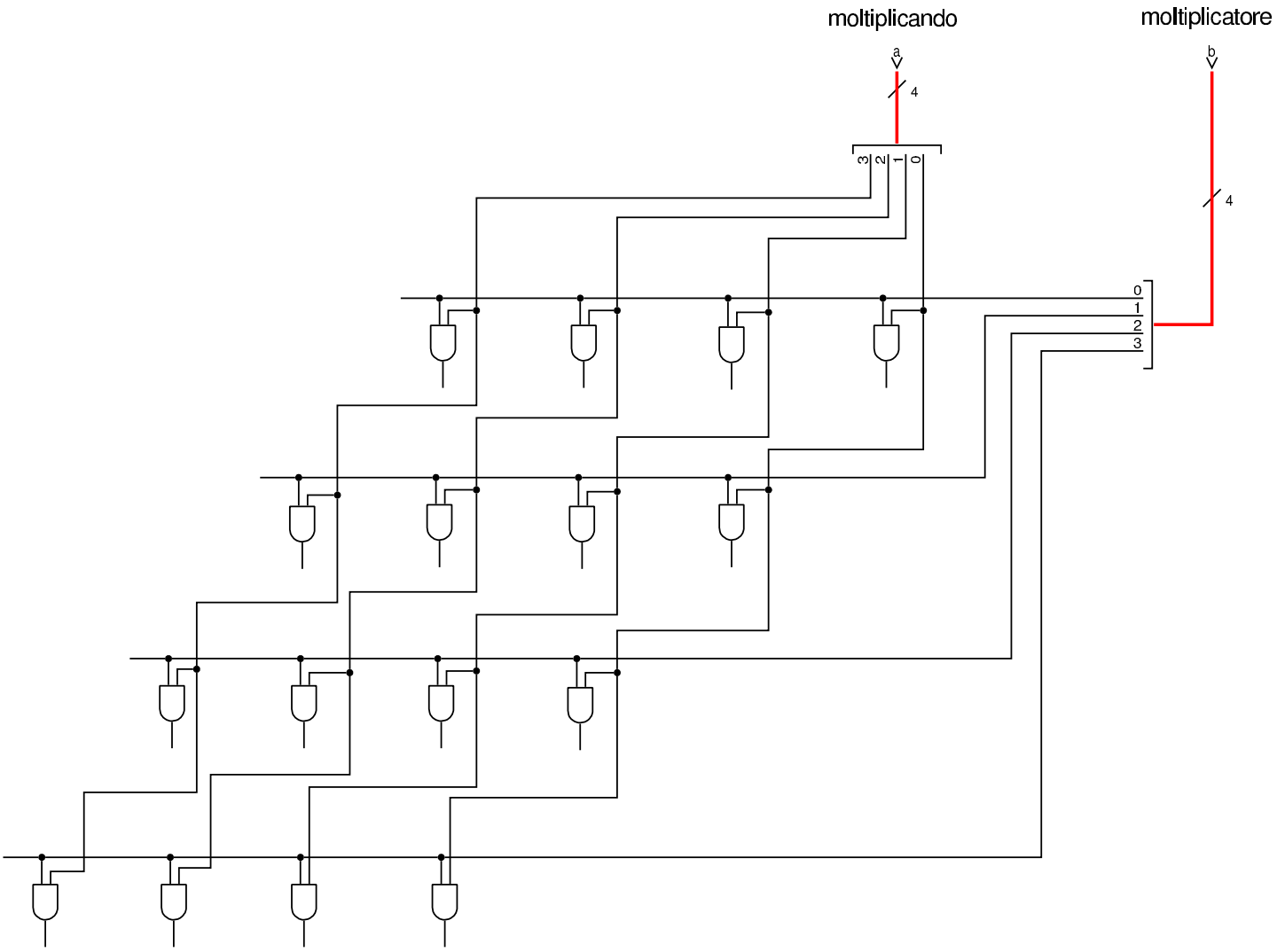
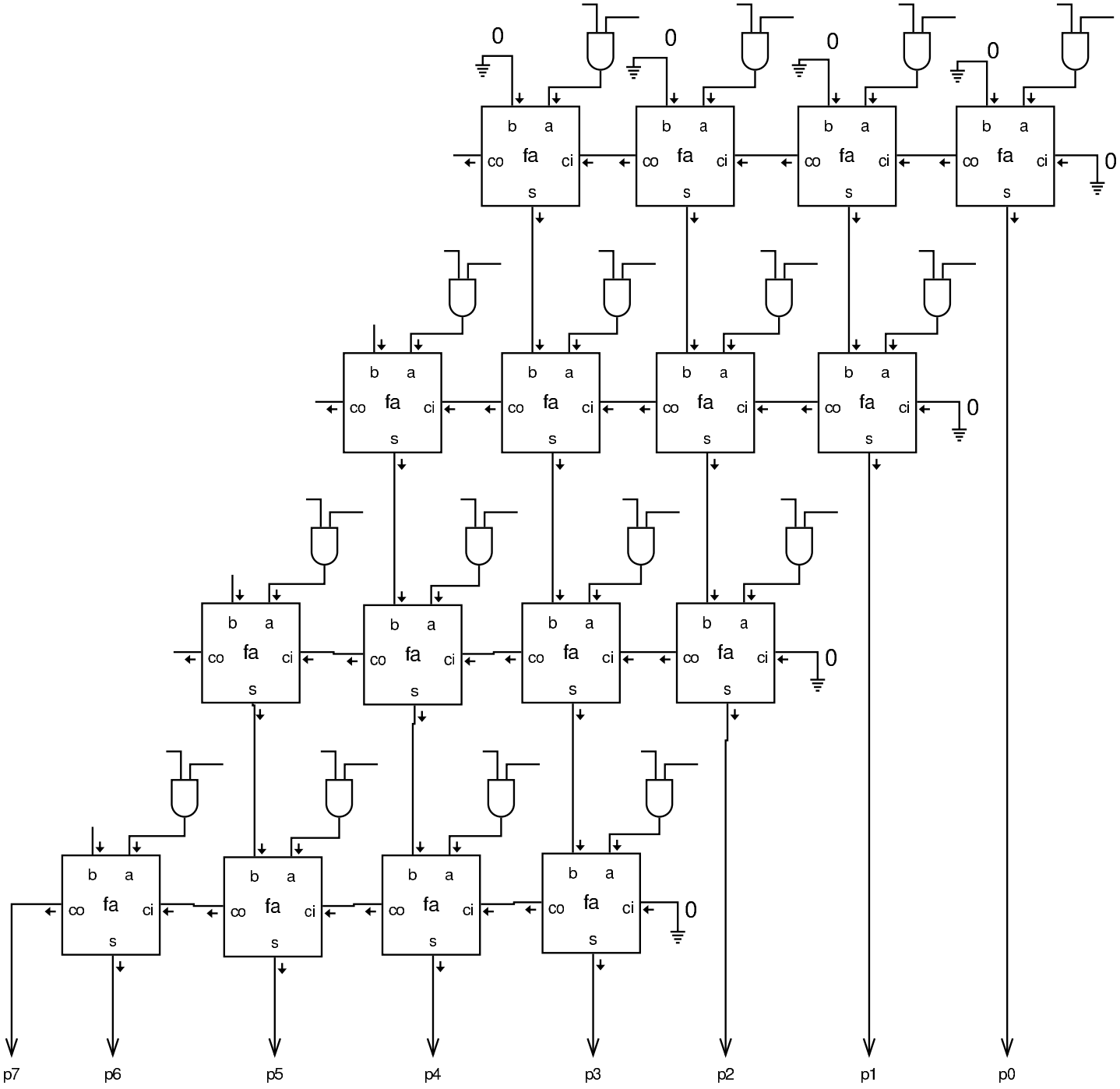


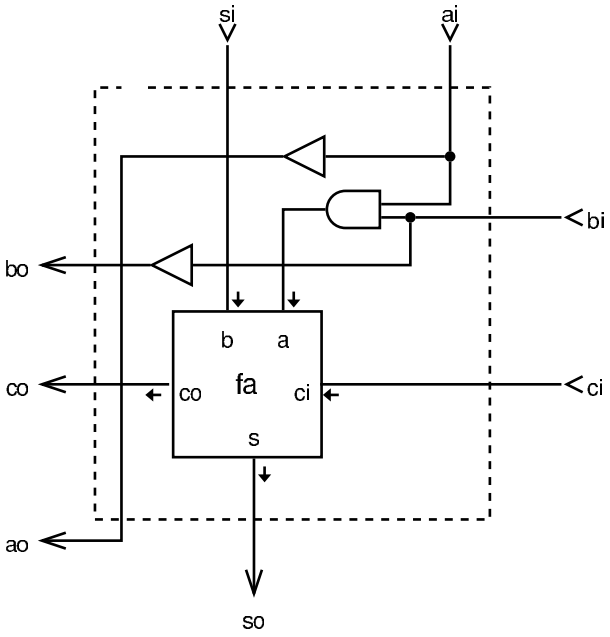
Figura u98.63. Quanto generato dalle porte AND, viene sommato con degli addizionatori completi.



Per mettere assieme la moltiplicazione, svolta dall'operatore AND, e la somma, occorre costruire delle celle apposite, utilizzano un addizionatore completo. Come si vede dalla figura, uno degli addendi riceve il risultato del prodotto ottenuto con l'operatore AND, mentre

l'altro addendo riporta il valore proveniente dalla riga superiore.

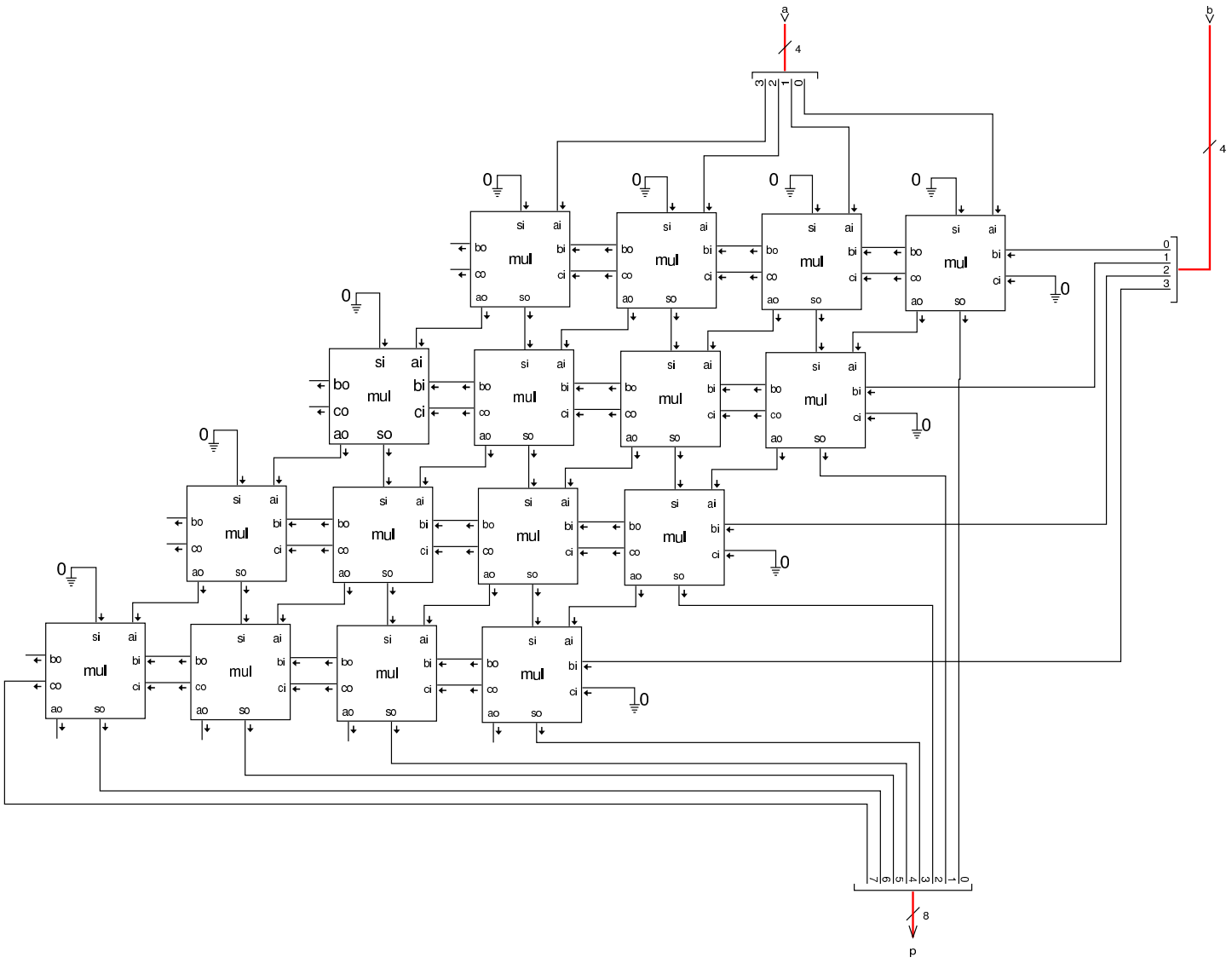
Figura u98.64. Cella usata per la costruzione della matrice che somma e fa scorrere il moltiplicando.



si = sum input
ai = a input = moltiplicando
bi = b input = moltiplicatore
ci = carry in
so = sum output = ((ai AND bi) + si + ci)
ao = ai
co = carry out = riporto della somma risultante da so
bo = bi

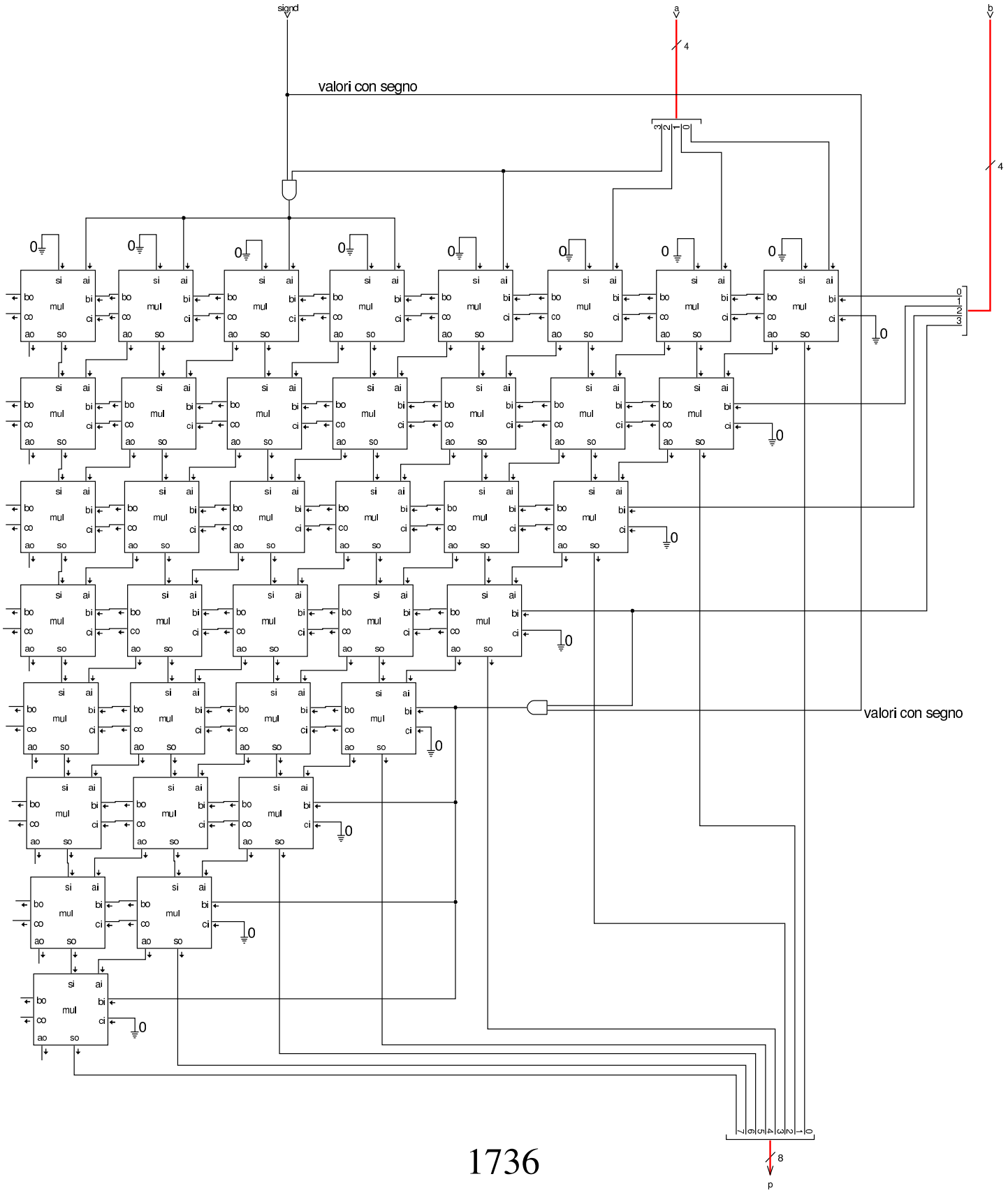
Le celle della figura vanno connesse per costruire le somme che costituiscono la moltiplicazione. La figura successiva mostra una soluzione limitata al caso di numeri privi di segno.

Figura u98.65. Moltiplicazione di interi **senza segno**, a quattro cifre, producendo un risultato a otto cifre: $p=a \cdot b$.



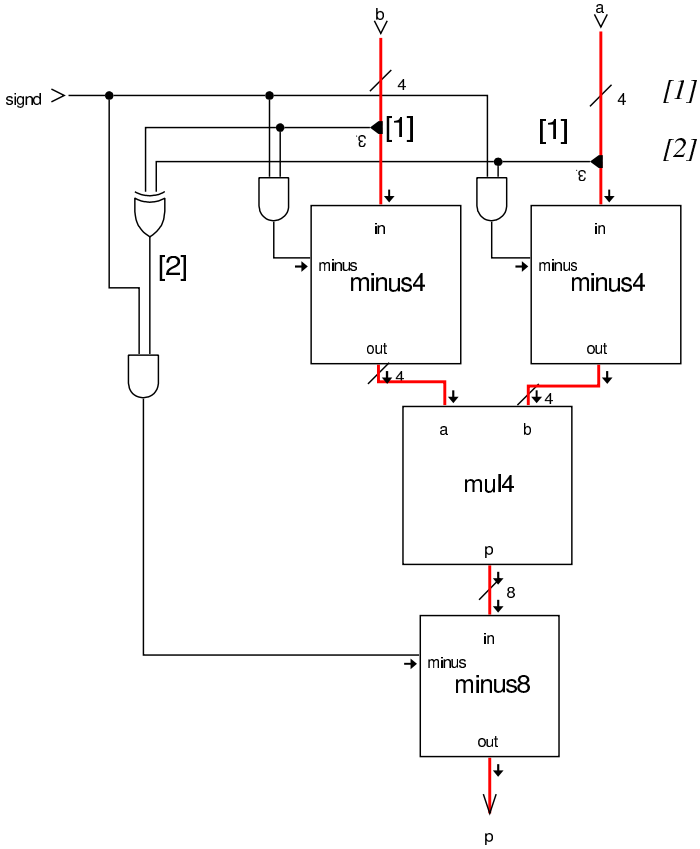
Per risolvere il problema dei valori con segno, occorre un ingresso aggiuntivo, per comunicare il fatto che si stia usando o meno numeri con segno, quindi occorrerebbe estendere la maglia come se si moltiplicassero valori con un numero doppio di cifre, estese secondo il segno che viene loro attribuito.

Figura u98.66. Soluzione completa, ma eccessivamente dispendiosa, per la moltiplicazione di due valori espressi a soli quattro bit.



Si può risolvere il problema della moltiplicazione quando si hanno valori con segno, adattare un circuito moltiplicatore di valori privi di segno, provvedendo a calcolare il complemento a due (ovvero a cambiare di segno) quando i valori risultano essere negativi. Nello schema della figura successiva, la funzione ***mul4*** corrisponde a un circuito moltiplicatore che opera solo su valori privi di segno; ***minus4*** e ***minus8*** sono circuiti che si occupano di invertire il segno se l'ingresso ***minus*** risulta attivo.

Figura u98.67. Adattamento di un moltiplicatore di valori senza segno, in modo da recepire anche valori con segno.



[1] viene letto il bit più significativo del gruppo, per determinare il segno

[2] i segni di moltiplicando e moltiplicatore sono diversi

Nella figura si vede che dagli ingressi ***a*** e ***b***, viene prelevato il bit più significativo, per determinare se i valori rispettivi sono negativi; se lo sono, i loro valori vengono moltiplicati per -1, prima di passare alla moltiplicazione; al termine, il risultato viene moltiplicato per -1

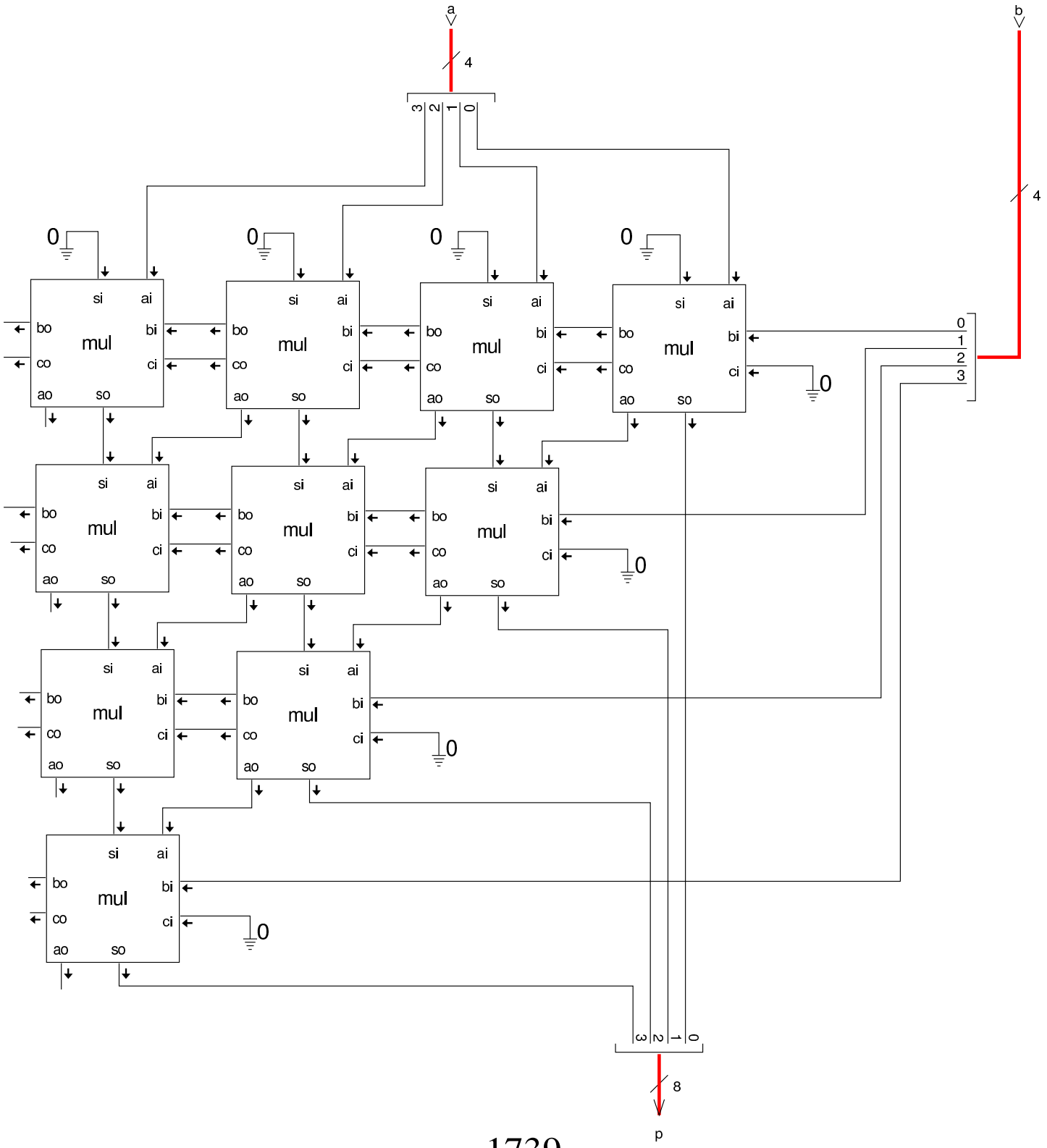
solo se i segni di ***a*** e ***b*** sono diversi. Naturalmente, tutto questo è subordinato al fatto che venga richiesto un calcolo che tenga conto dei segni, attraverso l'ingresso ***signd***. Il circuito denominato ***minus4*** corrisponde a quanto mostrato nella sezione u0.13; naturalmente, ***minus8*** è solo un'estensione dello stesso a 8 bit.

Se invece si riduce il risultato della moltiplicazione a una quantità di cifre uguale a quelle di moltiplicando e di moltiplicatore, ammesso che il risultato non venga troncato, ciò che si ottiene è valido, indipendentemente dai segni, senza bisogno di dover comunicare la gestione dei segni al circuito. Si mostra nuovamente un esempio di calcolo della moltiplicazione, simile a quello iniziale, per dimostrare questa affermazione.

Figura u98.68. Moltiplicazione di numeri a quattro cifre binarie, troncando il risultato alla stessa ampiezza di moltiplicando e di moltiplicatore.

moltiplicazione di interi senza segno	*moltiplicando negativo, moltiplicatore positivo*	*moltiplicando positivo, moltiplicatore negativo*	*moltiplicando negativo, moltiplicatore negativo*
1011×	**1101**×	**0101**×	**1011**×
1101=	**0010**=	**1110**=	**1101**=
1011+	**0000**+	**0000**+	**1011**+
0000+	**1010**+	**1010**+	**0000**+
1100+	**0000**+	**0100**+	**1100**+
1000=	**0000**=	**1000**=	**1000**=
1111	1010	0110	1111
risultato non valido perché troncato troppo presto	*risultato valido*	*risultato valido*	*risultato valido*

Figura u98.69. Circuito combinatorio semplificato per il prodotto tra due numeri, senza dover tenere conto della gestione del segno, ma con il risultato troncato al rango di moltiplicando e moltiplicatore.

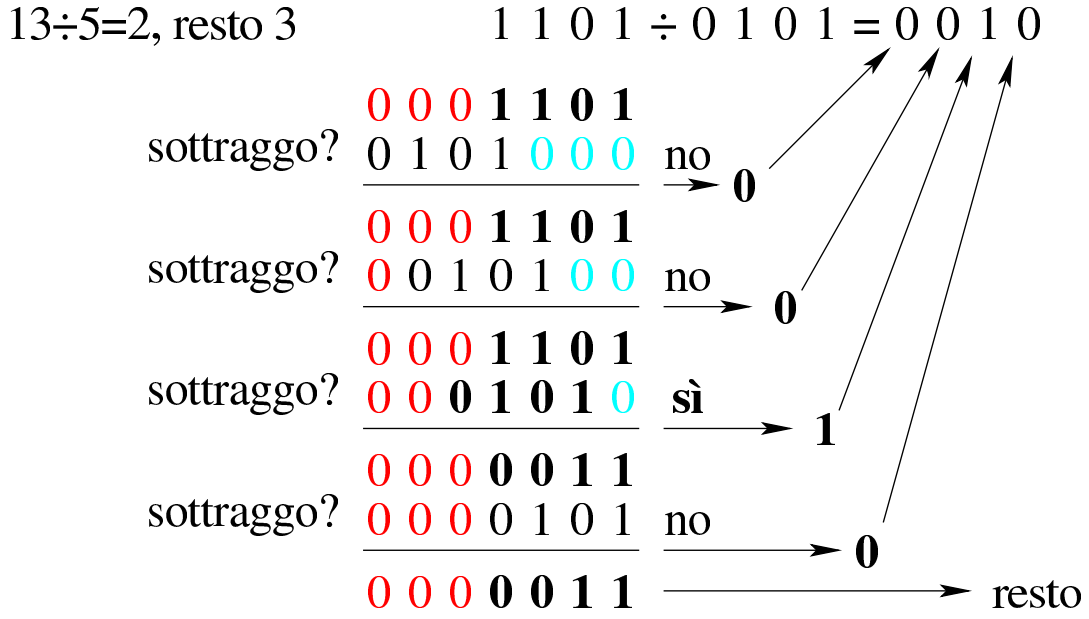


Divisione

«

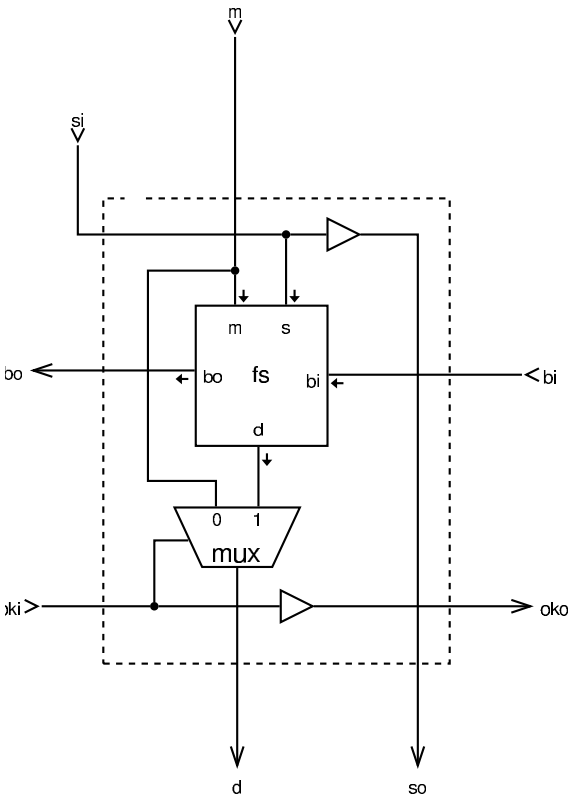
La divisione richiede la sottrazione e lo scorrimento, come si può vedere dall'esempio successivo, in cui il divisore viene confrontato con il dividendo, partendo dalle cifre più significative, sottraendo il divisore al dividendo solo quando ciò è possibile. In tal modo, il risultato intero della divisione è dato dal successo o meno di tali sottrazioni, mentre il resto è ciò che rimane dopo tutte le sottrazioni.

Figura u98.70. Procedimento usato per la divisione intera: dividendo e divisore si intendono privi di segno.



Per risolvere il problema attraverso un circuito combinatorio, si possono usare dei moduli per la sottrazione completa, da integrare con un controllo sul risultato, il quale deve essere prodotto solo se la sottrazione è ammissibile, altrimenti va riproposto il valore originale anche in uscita. Come nel caso della moltiplicazione, si possono costruire delle celle apposite.

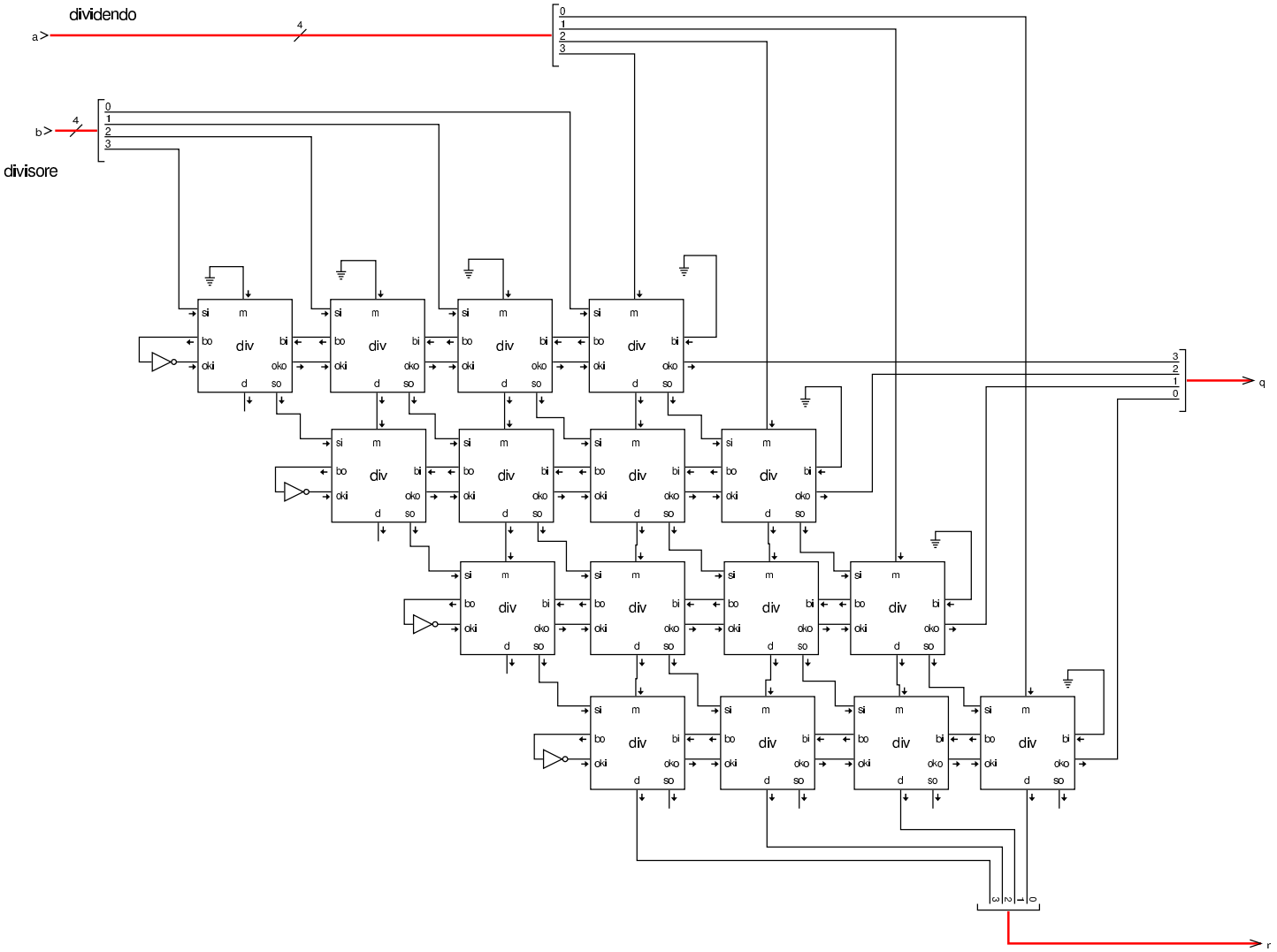
Figura u98.71. Cella usata per la sottrazione condizionata.



fs = full subtractor, sottrattore completo
m = minuendo
si = sottraendo in ingresso
so = sottraendo, copiato in uscita
bi = borrow in, richiesta di prestito di una cifra dal modulo precedente
bo = borrow out, richiesta di prestito di una cifra al modulo successivo
oki = ok in, la sottrazione è valida, ingresso
oko = ok out, la sottrazione è valida, copia in uscita

Con queste celle, dopo ogni sottrazione, si ottiene l'informazione se c'è la richiesta di un prestito o meno, dall'uscita ***bo*** (*borrow out*: se c'è tale richiesta di prestito, significa che la sottrazione non può avere luogo, quindi, tale segnale viene invertito e usato per avvisare tutte le celle di una fila che devono riproporre il valore originale anche in uscita, rinunciando alla sottrazione.

Figura u98.72. Esempio di soluzione per la divisione di valori privi di segno.

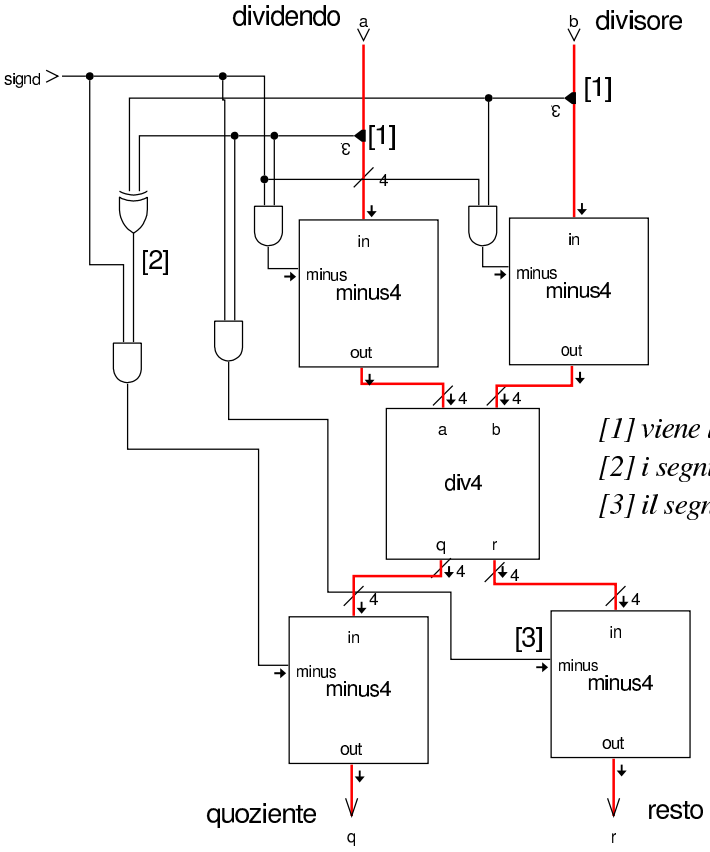


La divisione è complicata al riguardo della gestione dei valori con segno. Si pongono i casi seguenti, con cui stabilire il segno che devono avere i risultati:

Dividendo	Divisore	Quoziente	Resto
+	+	+	+
+	−	−	+
−	+	−	−
−	−	+	−

Per risolvere il problema della divisione di valori con segno, conviene aggiungere a ciò che divide valori senza segno, un controllo che inverta opportunamente i segni di dividendo, divisore, quoziente e resto, quando necessario.

Figura u98.74. Completamento del divisore con la gestione dei valori con segno: ***div4*** è il divisore di valori privi di segno; ***minus4*** si occupa di invertire il segno del valore in ingresso, se il segnale ***signd*** è attivo.

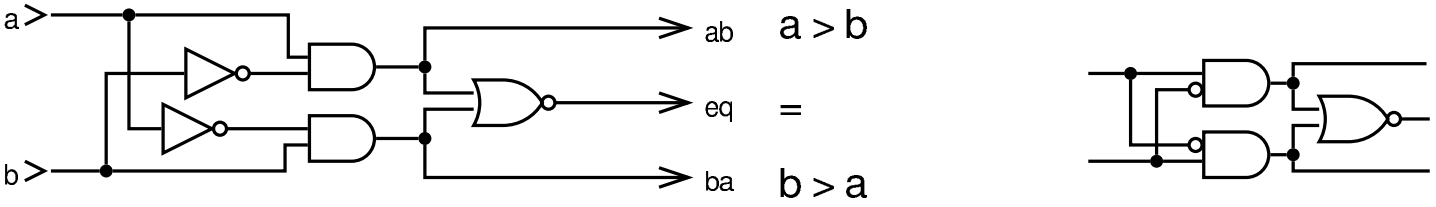


[1] viene letto il bit più significativo del gruppo, per determinarne il segno
[2] i segni di dividendo e divisore sono diversi
[3] il segno del resto dipende soltanto dal segno del dividendo

Comparazione

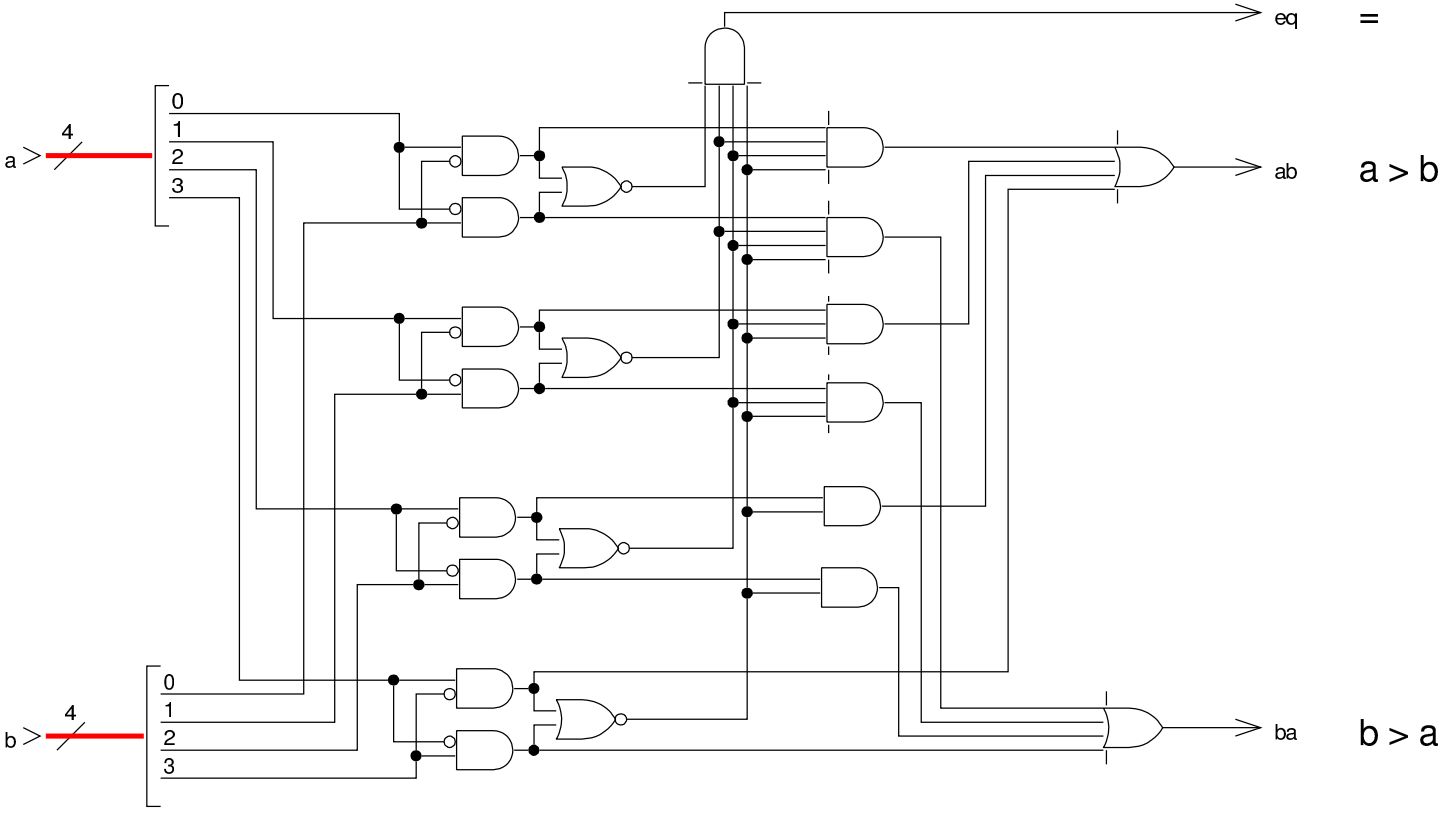
La comparazione tra due valori espressi in forma binaria, richiede il confronto, bit per bit, partendo dalla cifra più significativa. Da ogni confronto bit per bit, occorre sapere se sono uguali o se uno dei due sia maggiore. La figura successiva mostra la realizzazione di questo confronto tra due soli valori logici.

Figura u98.75. Confronto tra due soli ingressi; nel disegno di destra si usa una forma compatta per rappresentare gli ingressi negati delle due porte AND.



Nel confronto tra più bit, si fanno le stesse verifiche a coppia, facendo prevalere la prima differenza a partire dal bit più significativo.

Figura u98.76. Confronto tra numeri senza segno, a quattro bit.

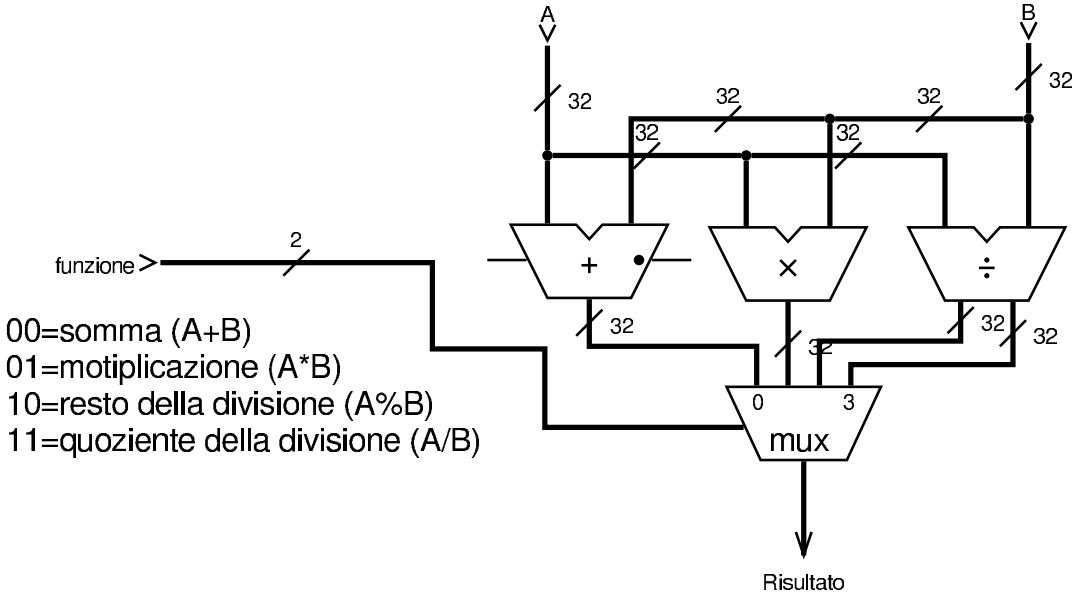


Costruzione di un'unità aritmetico-logica

L'unità aritmetico-logica, nota come ALU (*arithmetic-logic unit*), è un circuito combinatorio che racchiude le funzionalità di calcolo principali di un CPU. Complessivamente si tratta dell'unione di più circuiti combinatori, ognuno dei quali compie un solo tipo o pochi tipi di calcoli; tale complesso si ottiene attraverso l'uso opportuno di multiplatori per selezionare il risultato a cui si è interessati.

«a2» 2013.11.11 --- Copyright © Daniele Giacomini -- appunti2@gmail.com http://informaticalibera.net

Figura u99.1. Schema esemplificativo di come si possono connettere assieme più componenti per formare una sola ALU: ogni componente riceve una copia dei valori in ingresso, ma l'uscita viene selezionata da un multiplatore, attraverso un codice che rappresenta la funzione da richiedere alla ALU.



Negli esempi, dove possibile, si suddivide il lavoro in moduli da 8 cifre binarie; inoltre, le varie componenti vengono realizzate in modo che possano operare con ranghi differenti di bit: 8, 16, 24 e 32.

Indicatori

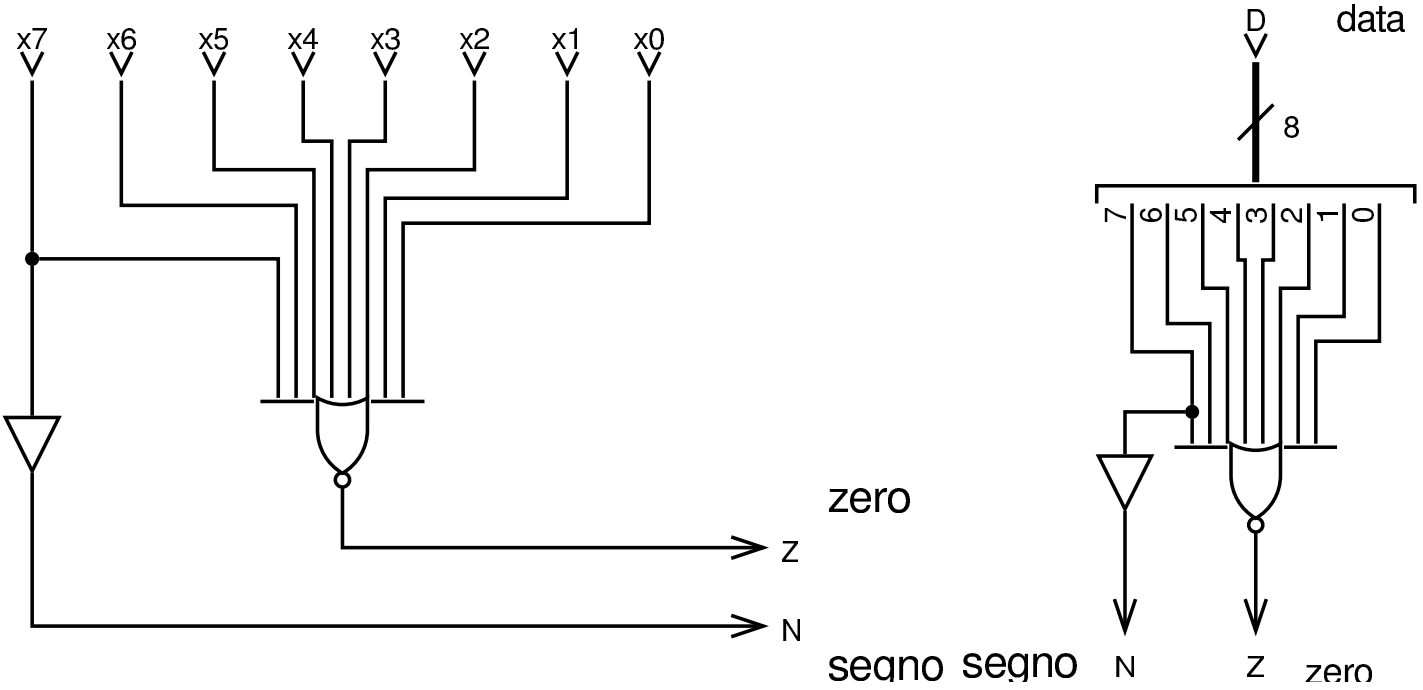
«

Quando opportuno, devono essere fornite informazioni aggiuntive sui risultati dei calcoli, cosa che si fa normalmente attraverso degli indicatori, costituiti in pratica da delle uscite aggiuntive. Gli indicatori più comuni sono: riporto, zero, segno e traboccamento.

Denominazione	Sigla comune	Descrizione
zero	Z	Indica che il risultato dell'operazione è pari a zero.
segno negativo	N	Indica che, nel risultato, la cifra più significativa è pari a uno, per cui, se la rappresentazione numerica tiene conto del segno, si tratta di un numero negativo.
riporto *carry*	C	Indica che il risultato complessivo dovrebbe avere una cifra in più rispetto a quanto ottenuto, il quale è così troncato della sua cifra più significativa.
traboccamento *overflow*	O	Si manifesta quando il risultato non è valido perché risulta troncato nella parte più significativa, o perché risulta di segno diverso rispetto a quello che dovrebbe avere. Nella somma lo si determina confrontando gli ultimi due riporti e trovando che sono differenti.

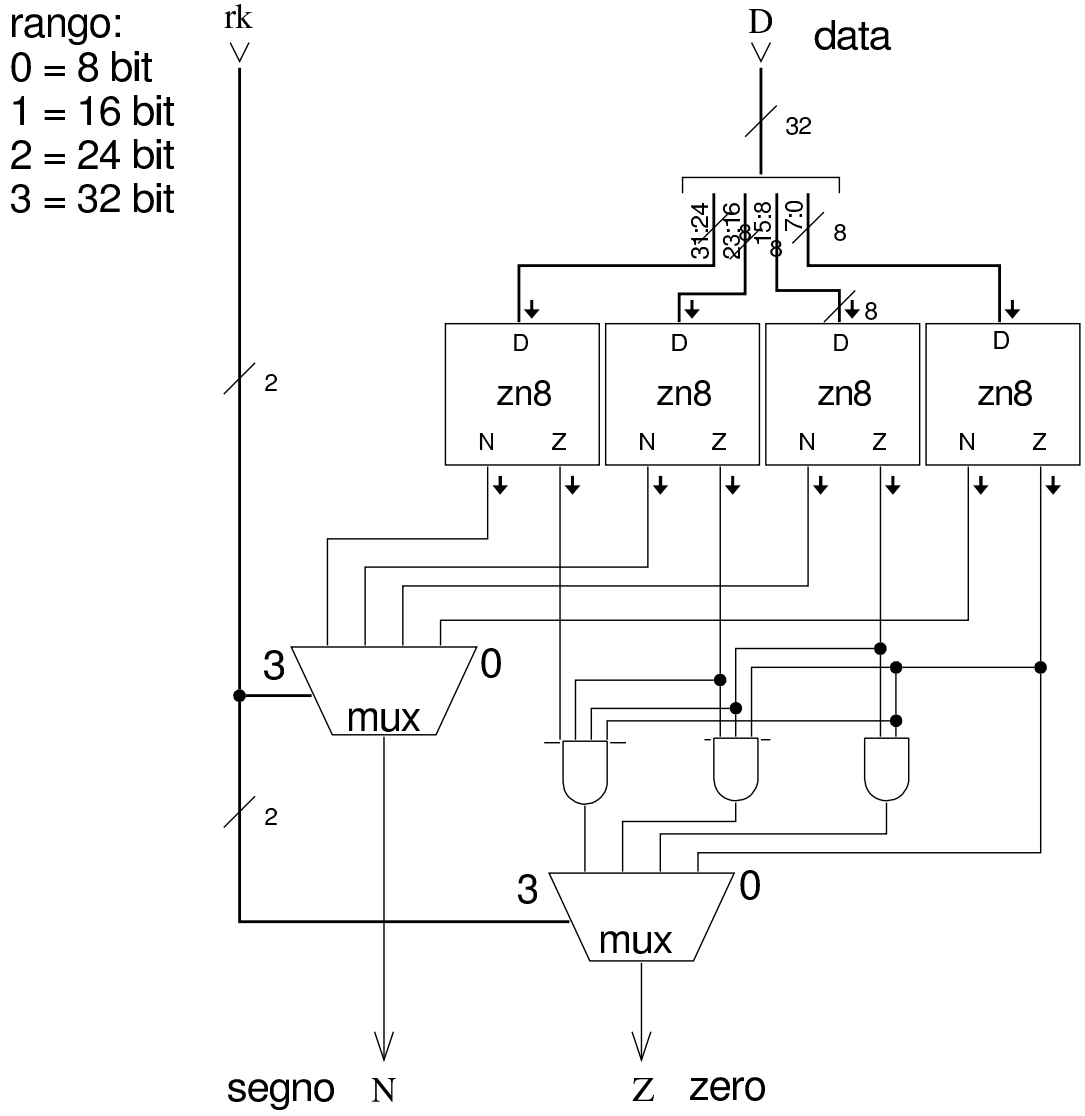
La verifica del risultato pari a zero può essere fatta poco prima dell'uscita dell'ALU, con l'ausilio di una porta NOR, la quale produce un risultato solo quando nessun ingresso è attivo; allo stesso modo, si determina facilmente il segno del risultato (ammesso che il contesto consideri la presenza di un segno), controllando il valore del bit più significativo.

Figura u99.3. Modulo **`zn8`**: controllo del risultato pari a zero e della presenza eventuale di un segno. A destra appare lo stesso circuito in modo compatto, dove gli otto ingressi sono raccolti opportunamente.



Volendo gestire, a seconda dei casi, interi con rango diverso, si può utilizzare quanto già visto nella figura precedente, in forma di modulo, costruendo un sistema un po' più complesso.

Figura u99.4. Modulo **`zn32`**: valutazione degli indicatore «zero» e «segno», per interi da 8 a 32 bit. L'ingresso ***rk*** (*rank*) consente di stabilire il rango: si va da zero che rappresenta solo 8 bit, fino a 3 che richiede un rango di 32 bit. Il modulo **`zn8`** è descritto nella figura u99.3.

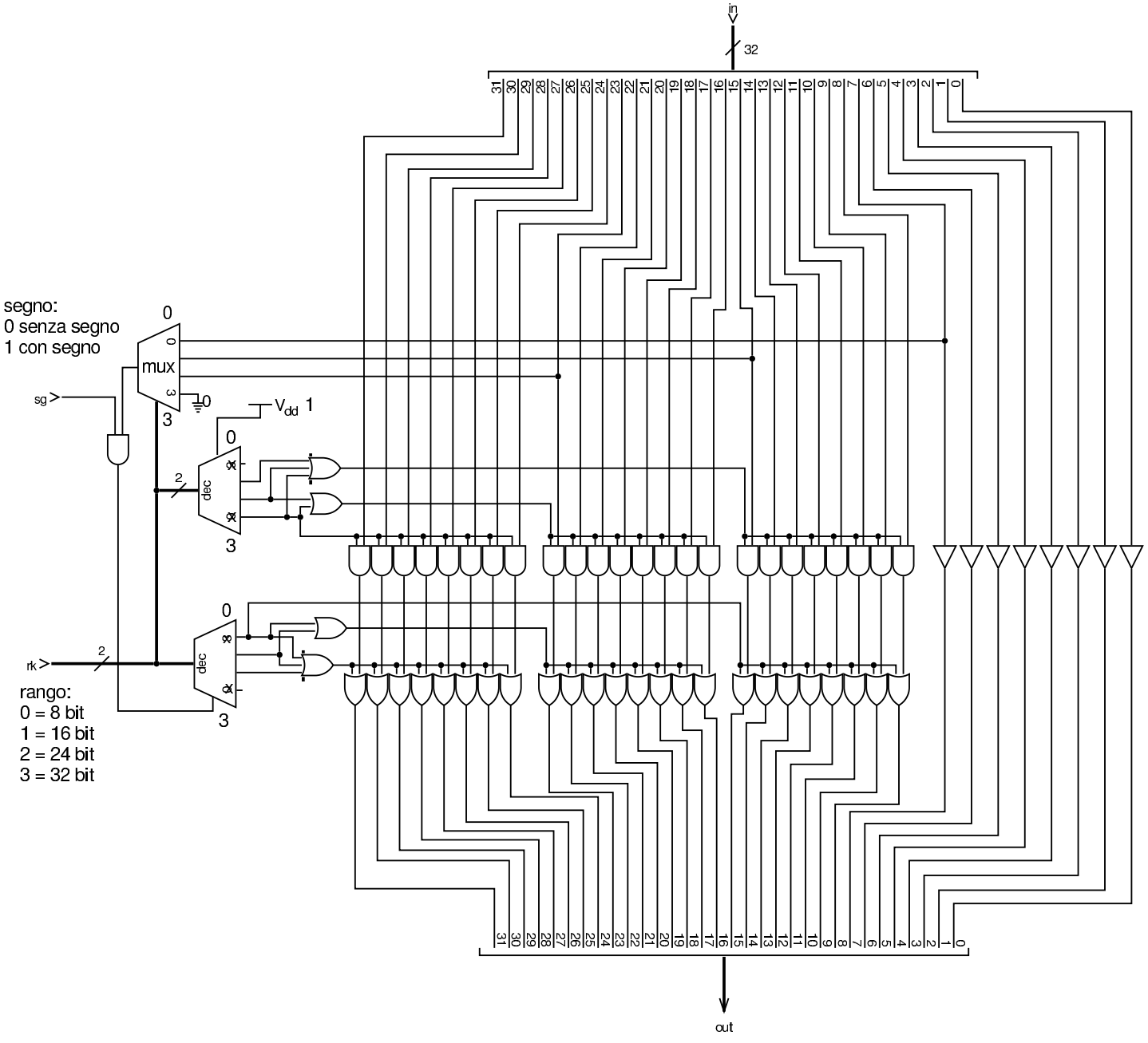


Troncamento di un valore in base al rango

I moduli che vengono presentati nelle sezioni successive, consentono di operare su valori a 32 bit, con la possibilità di limitare l'intervento a ranghi inferiori (ma sempre multipli di otto). I risultati

che si ottengono sono validi solo nell'ambito del rango selezionato, ma ci possono essere dei contenuti ulteriori, da ignorare, nei bit più significativi oltre al rango voluto. Il modulo successivo, consente di filtrare opportunamente ciò che è al di fuori del rango desiderato, con la possibilità di estendere il segno, in modo da non creare confusione, o comunque per ridurre la possibilità di errori.

Figura u99.5. Modulo **`rk`**: filtro dei bit al di fuori del rango selezionato, tenendo conto del segno se il valore in ingresso è da intendersi con segno.



Inversione di segno

«

Per la moltiplicazione e la divisione, si rende necessario un modulo in grado di invertire il segno di un numero binario. Il modulo **`minus8`** interviene su 8 bit e serve per realizzare moduli di rango maggiore.

Figura u99.6. Modulo **`minus8`**: inverte il segno di un numero a 8 bit se l'ingresso ***M*** è attivo e lo è anche l'ingresso ***Ci***. Il modulo **`ha`** è un semiaddizionatore (*half adder*) comune; gli ingressi ***a*** e ***b*** del semiaddizionatore sono intercambiabili.

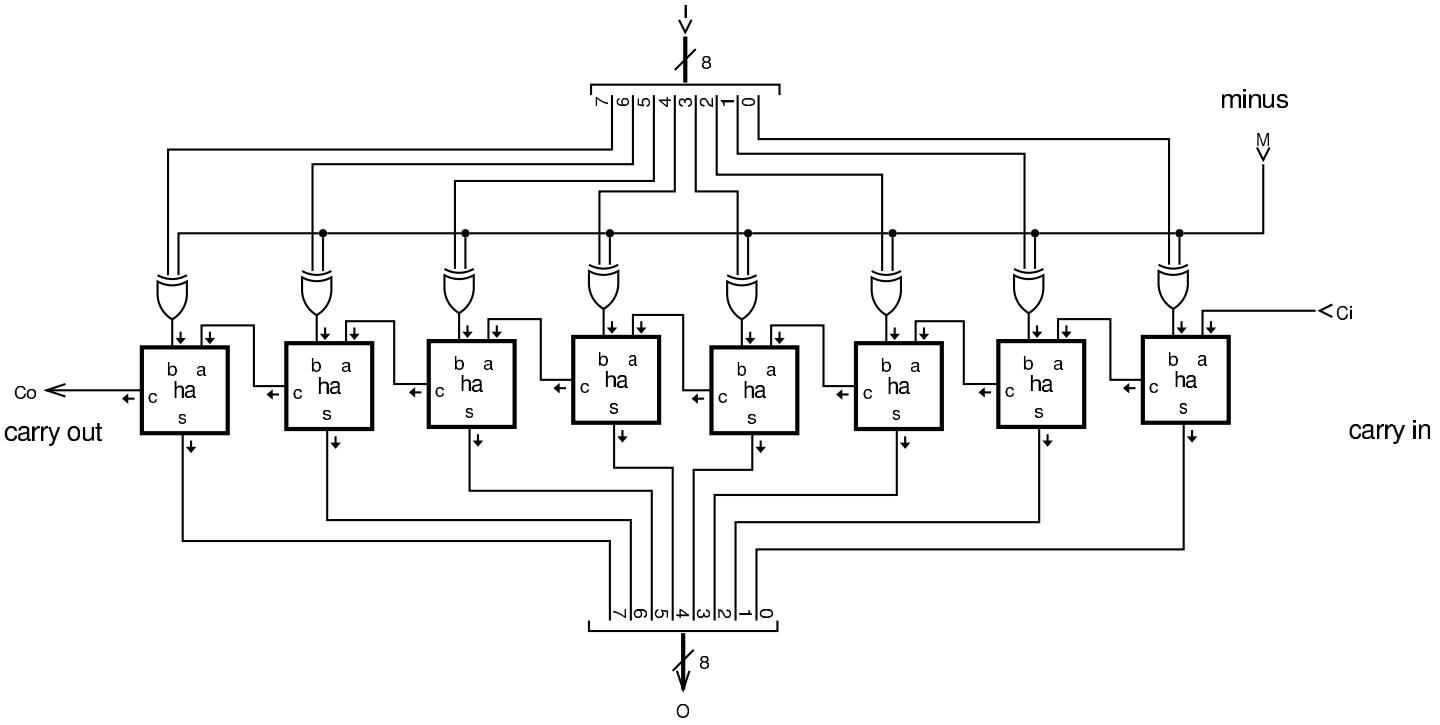
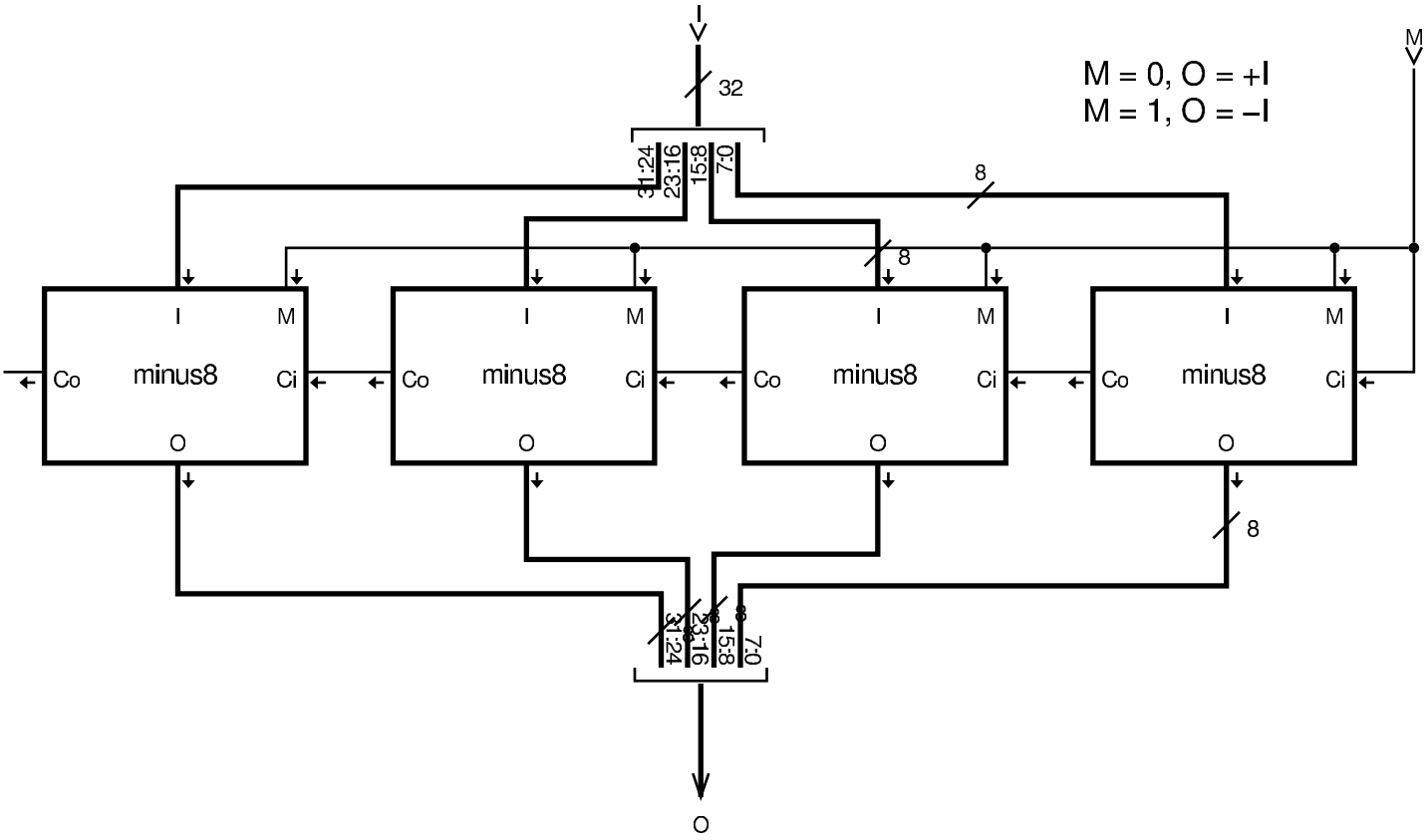


Figura u99.7. Modulo **`minus32`**: inverte il segno di un numero a 32 bit, se l'ingresso ***M*** è attivo e lo è anche l'ingresso ***Ci***. Il modulo **`minus8`** è descritto nella figura u99.6.



Somma e sottrazione

La figura successiva mostra un modulo da 8 bit per la somma e la sottrazione, gestendo opportunamente il riporto o la richiesta di prestito di una cifra e fornendo l'informazione sull'eventuale traboccamento, confrontando gli ultimi due riporti.

Figura u99.8. Modulo **`as8`**: somma e sottrazione di interi a otto cifre binarie. L'ingresso ***f*** (*function*) controlla l'applicazione di una somma o di una sottrazione; l'uscita ***O*** fornisce l'informazione sull'eventuale traboccamento. Il modulo **`fa`** è un addizionatore completo (*full adder*).

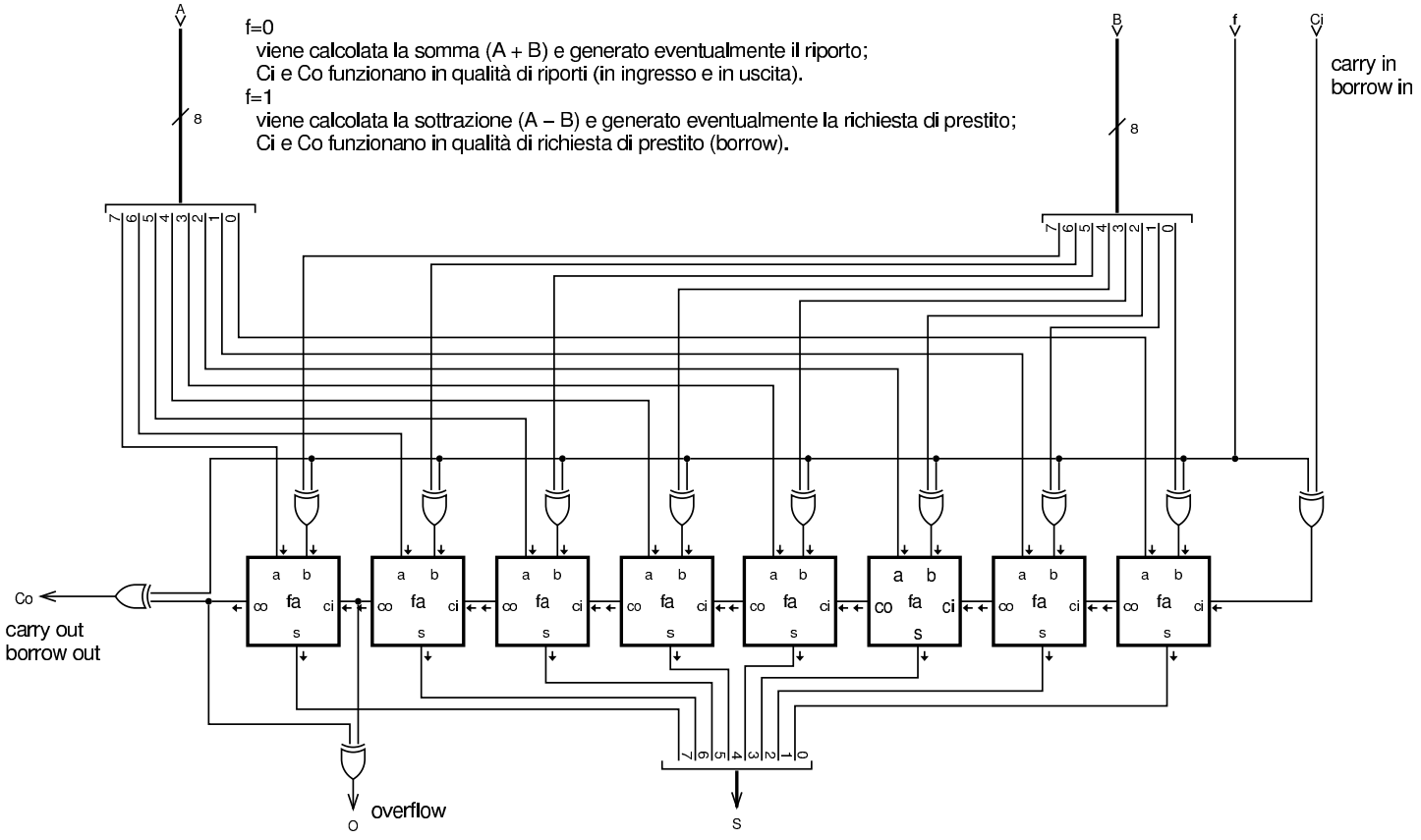
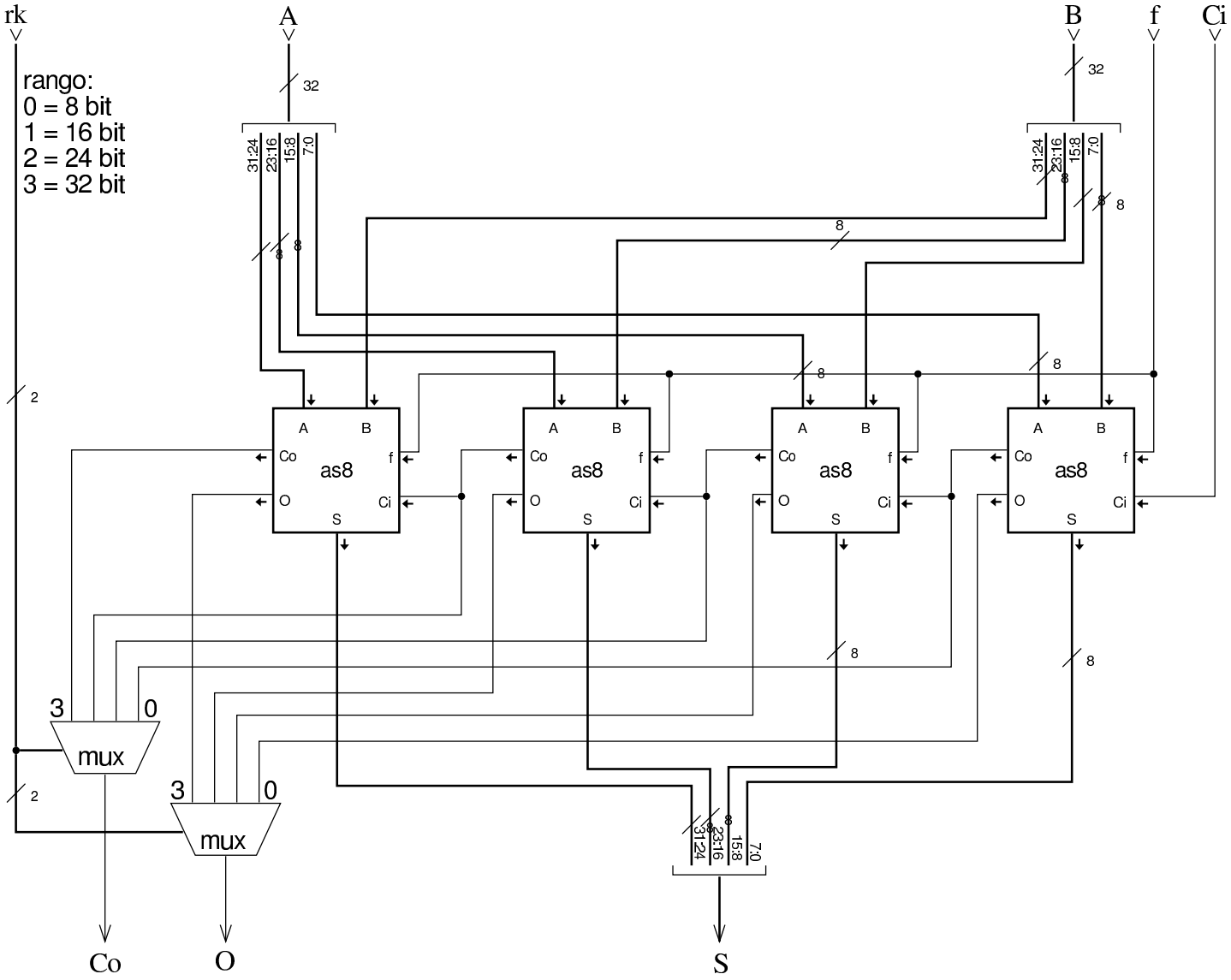


Figura u99.9. Modulo **`as32`**: somma e sottrazione con la possibilità di controllare il rango dei dati in ingresso e in uscita. Il modulo **`as8`** è descritto nella figura u99.8.



Scorrimento e rotazione

La figura successiva mostra un modulo per lo scorrimento logico e aritmetico su 8 bit. Gli ingressi e le uscite ***lin***, ***lout***, ***rin***, ***rout*** (*left*/*right in*/*out*), servono a produrre lo scorrimento e anche la rotazione su una scala multipla di 8 bit. Le uscite ***Cl*** e ***Cr*** (*carry left*/*right*) riproducono sempre il valore del bit che viene sospinto al-

l'esterno (dal lato sinistro o dal lato destro), mentre l'uscita ***O*** (*overflow*) si attiva solo in presenza di un traboccamento, dovuto a uno scorrimento aritmetico a sinistra che fa cambiare di segno al valore.

Figura u99.10. Modulo **`sh8`**, per lo scorrimento logico e aritmetico a 8 bit.

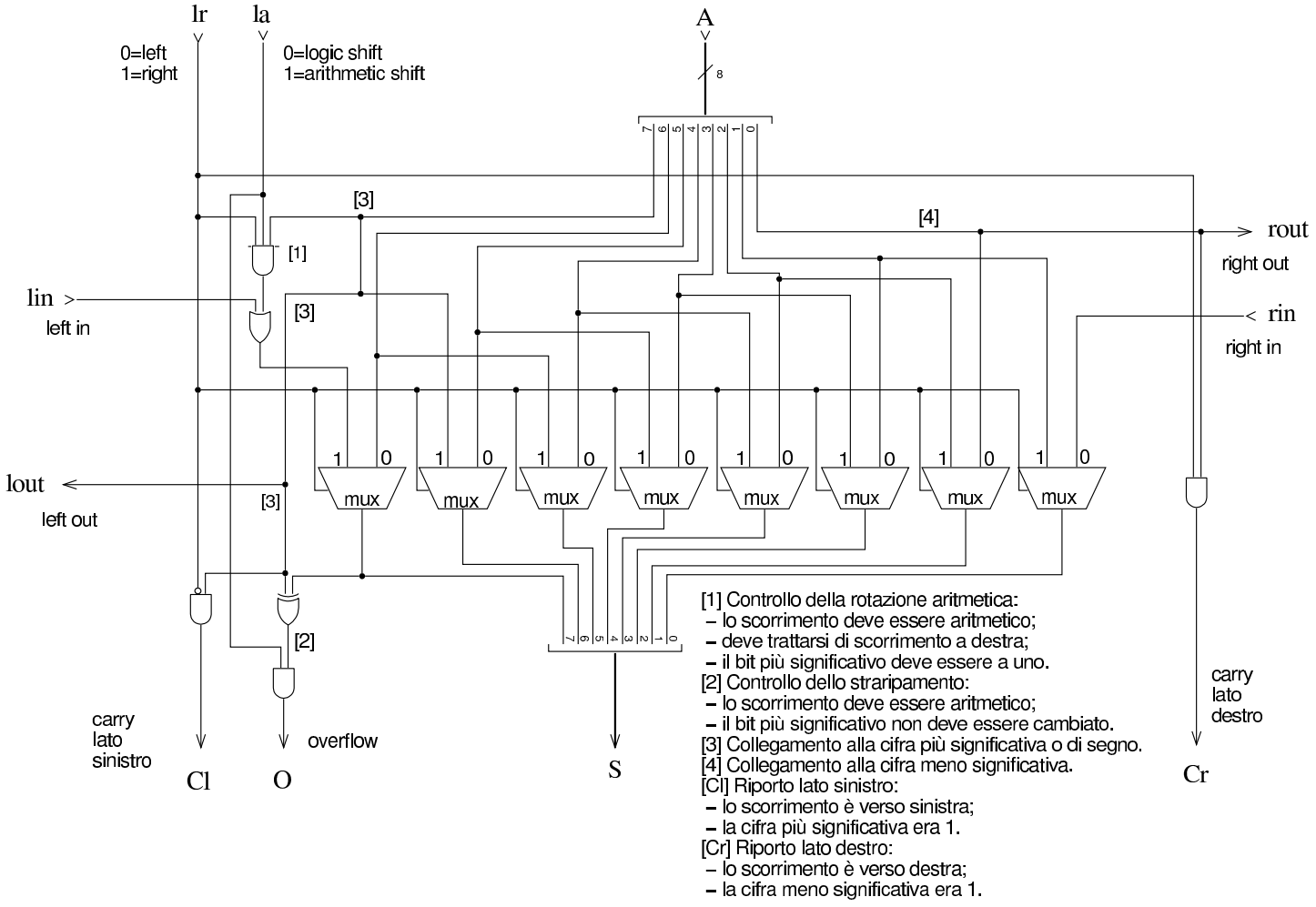


Figura u99.11. Modulo **`sh32`**: scorrimento logico e aritmetico a 32 bit, con la possibilità di intervenire su un rango inferiore. L'uscita ***C*** (*carry*) restituisce il valore del bit che viene sospinto a sinistra o a destra, in base al contesto. Il modulo **`sh8`** è descritto nella figura u99.10.

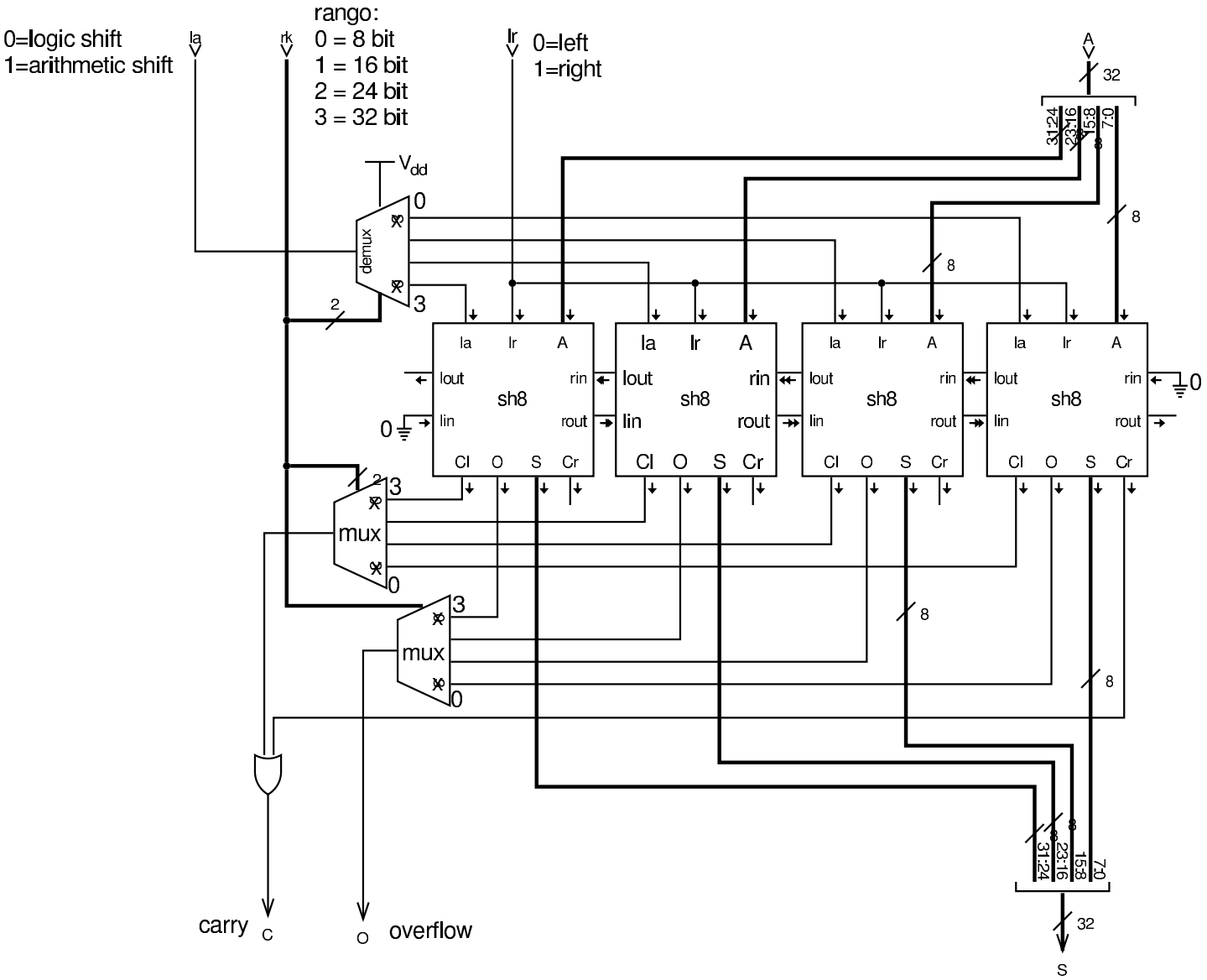


Figura u99.12. Modulo **`ro32`**: rotazione a 32 bit, con la possibilità di intervenire su un rango inferiore. Dal momento che nella rotazione non si distingue tra una modalità «logica» rispetto a una aritmetica, non è presente un'uscita che possa segnalare lo straripamento. Il modulo **`sh8`** è descritto nella figura u99.10.

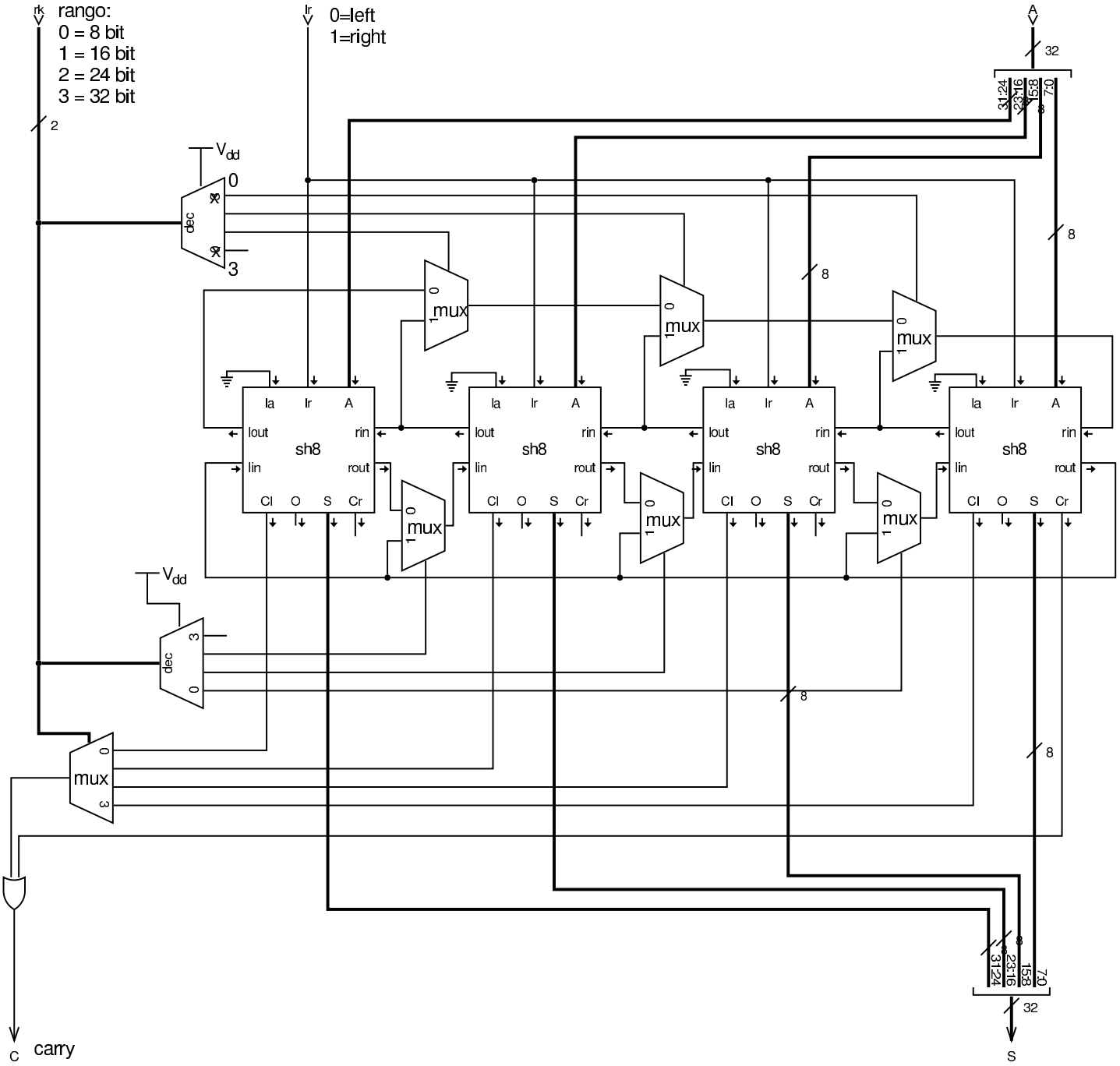
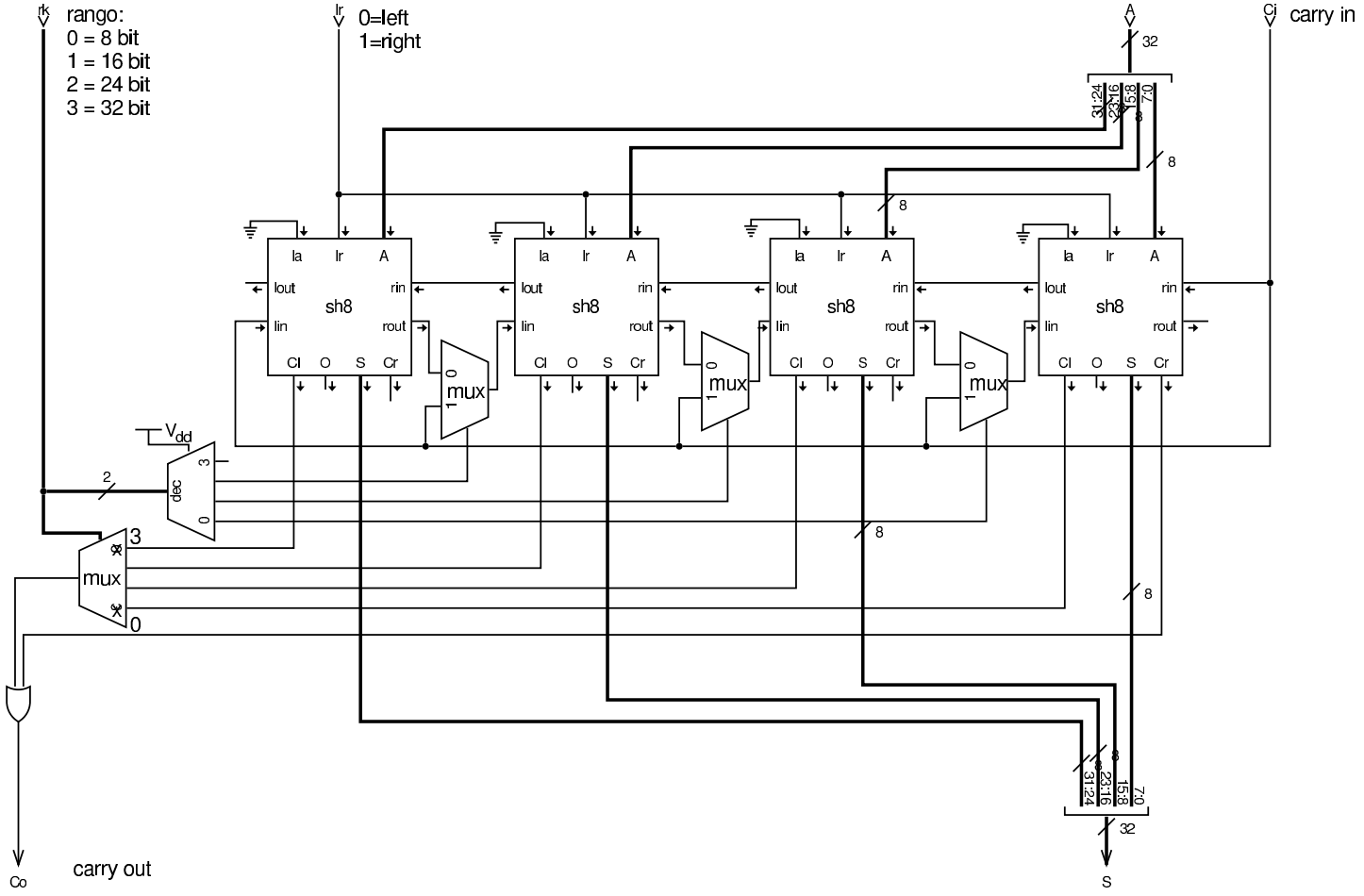


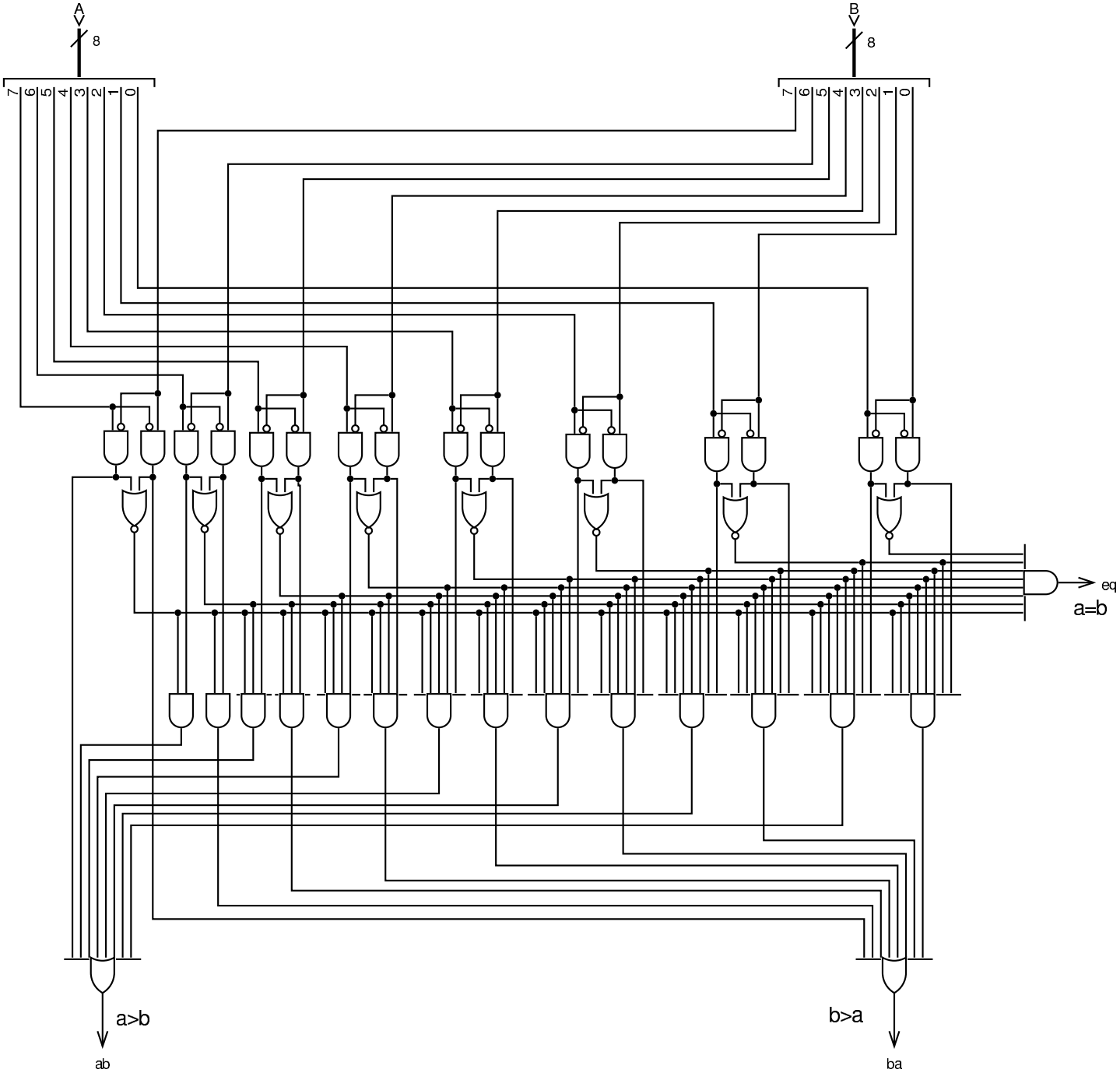
Figura u99.13. Modulo **`rc32`**: rotazione utilizzando il riporto, in quanto il valore del riporto in ingresso è ciò che viene inserito dal lato opposto alla direzione dello scorrimento. Il modulo **`sh8`** è descritto nella figura u99.10.



Comparazione di valori

La comparazione tra due valori consente di determinare se questi sono uguali o se uno dei due sia maggiore. Il modulo **`cmp8`** esegue una comparazione di valori senza segno, attivando un'uscita differente a seconda dei tre casi possibili: $a>b$, $a<b$, $a=b$.

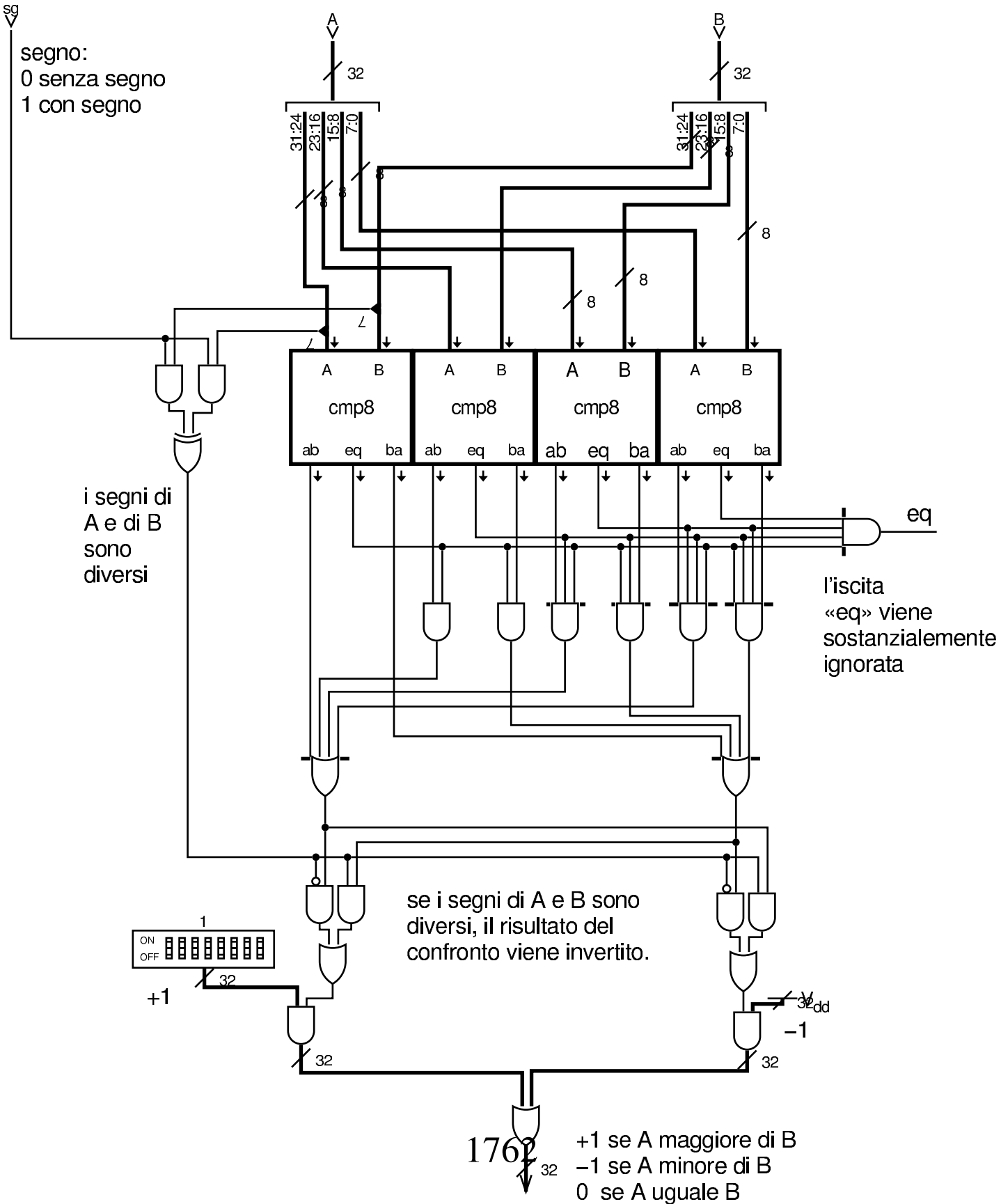
Figura u99.14. Modulo **cmp8**: comparazione di interi, senza segno, a 8 bit.



Il modulo **cmp8** può essere utilizzato per il confronto di valori espressi su una quantità multipla di bit; in tal caso, si può introdurre il controllo sul segno: se i valori da confrontare si intendono con segno, se i segni dei due valori sono discordi, l'esito del confronto va

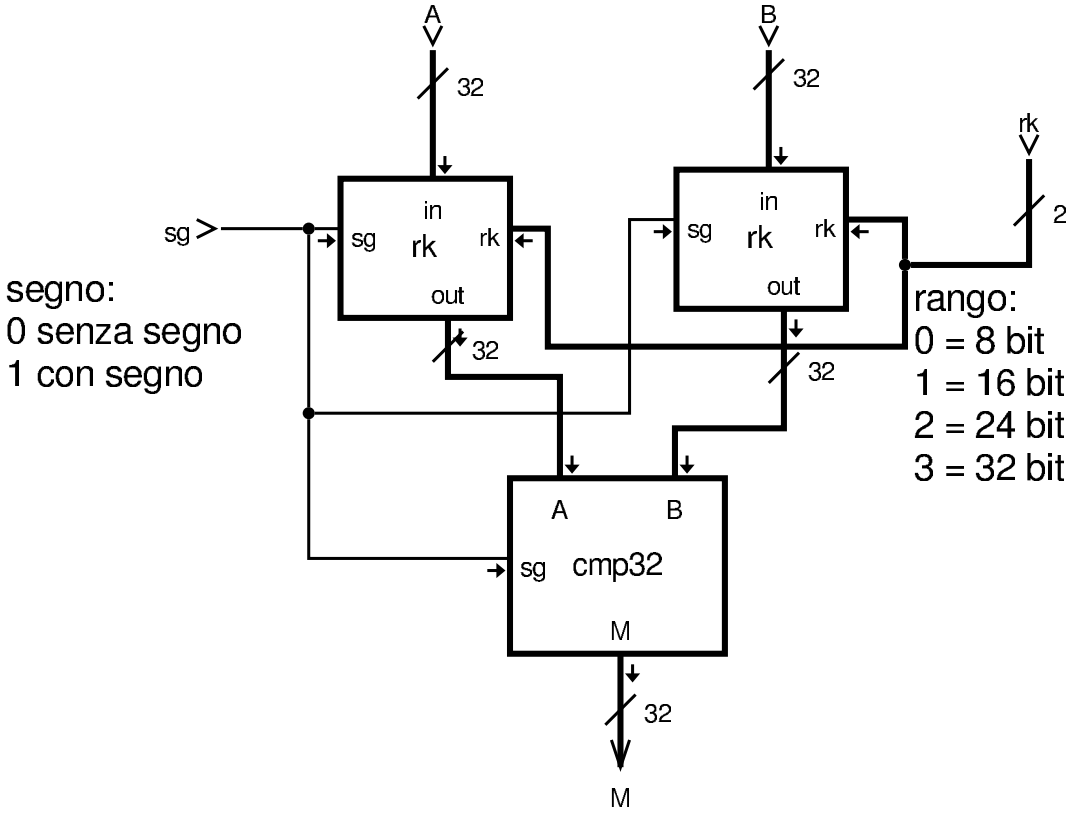
invertito. Il modulo **`cmp32`** confronta due valori a 32 bit, restituendo l'esito del confronto in forma di valore a 32 bit: zero se i valori sono uguali; +1 se $A>B$; −1 se $A<B$.

Figura u99.15. Modulo **`cmp32`**: confronto a 32 bit, con la possibilità di distinguere tra valori senza segno o con segno. Il modulo **`cmp8`** è descritto nella figura u99.14.



Il modulo **cmp32** non consente di ridurre il rango di bit preso in considerazione nel confronto; pertanto, se si desidera poter controllare il rango, occorre aggiungere il filtro del modulo **rk** (figura u99.5), come si vede nella figura successiva.

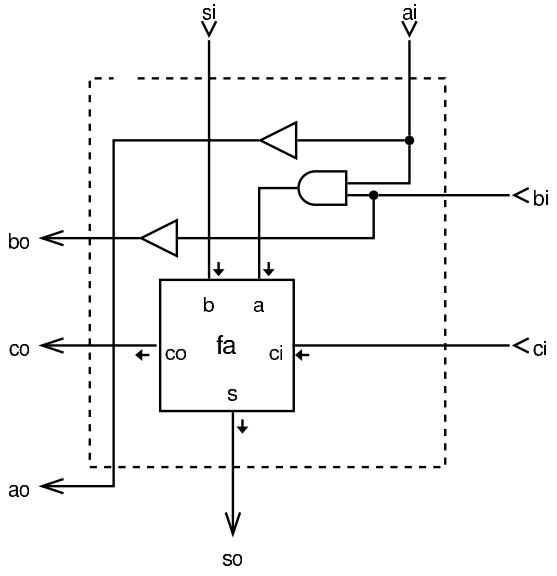
Figura u99.16. Esempio di filtro attraverso il modulo **rk** (figura u99.5).



Moltiplicazione

Come già descritto in precedenza, per la moltiplicazione ci si avvale di un modulo apposito che somma il risultato dell'operatore AND sui due valori in ingresso. Viene richiamato nella figura successiva.

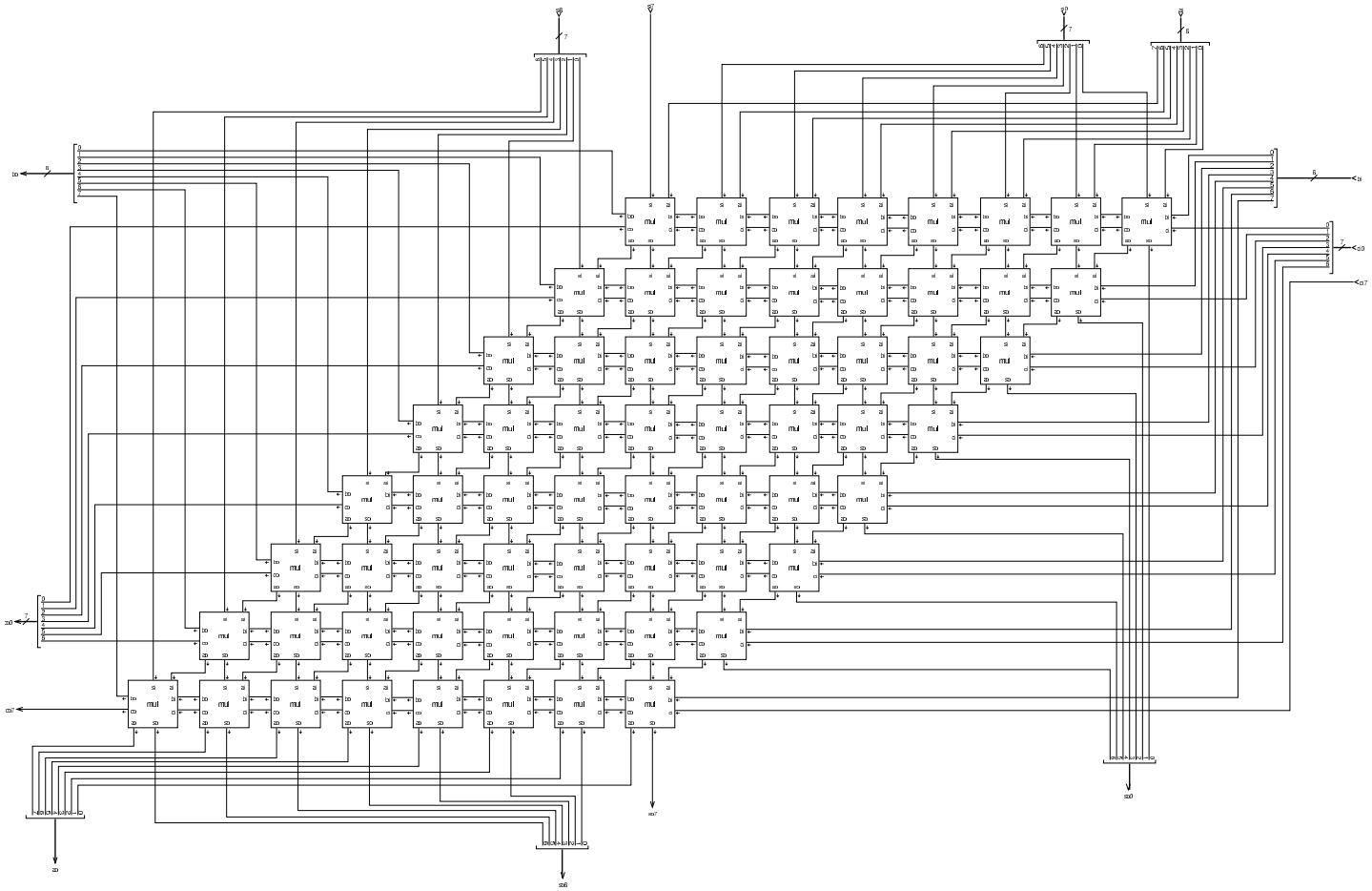
Figura u99.17. Modulo **`mul`**: moltiplicazione a un bit. Questo modulo viene poi usato per la realizzazione di quello a otto bit.



si = sum input
ai = a input = moltiplicando
bi = b input = moltiplicatore
ci = carry in
so = sum output = ((ai AND bi) + si + ci)
ao = ai
co = carry out = riporto della somma risultante da so
bo = bi

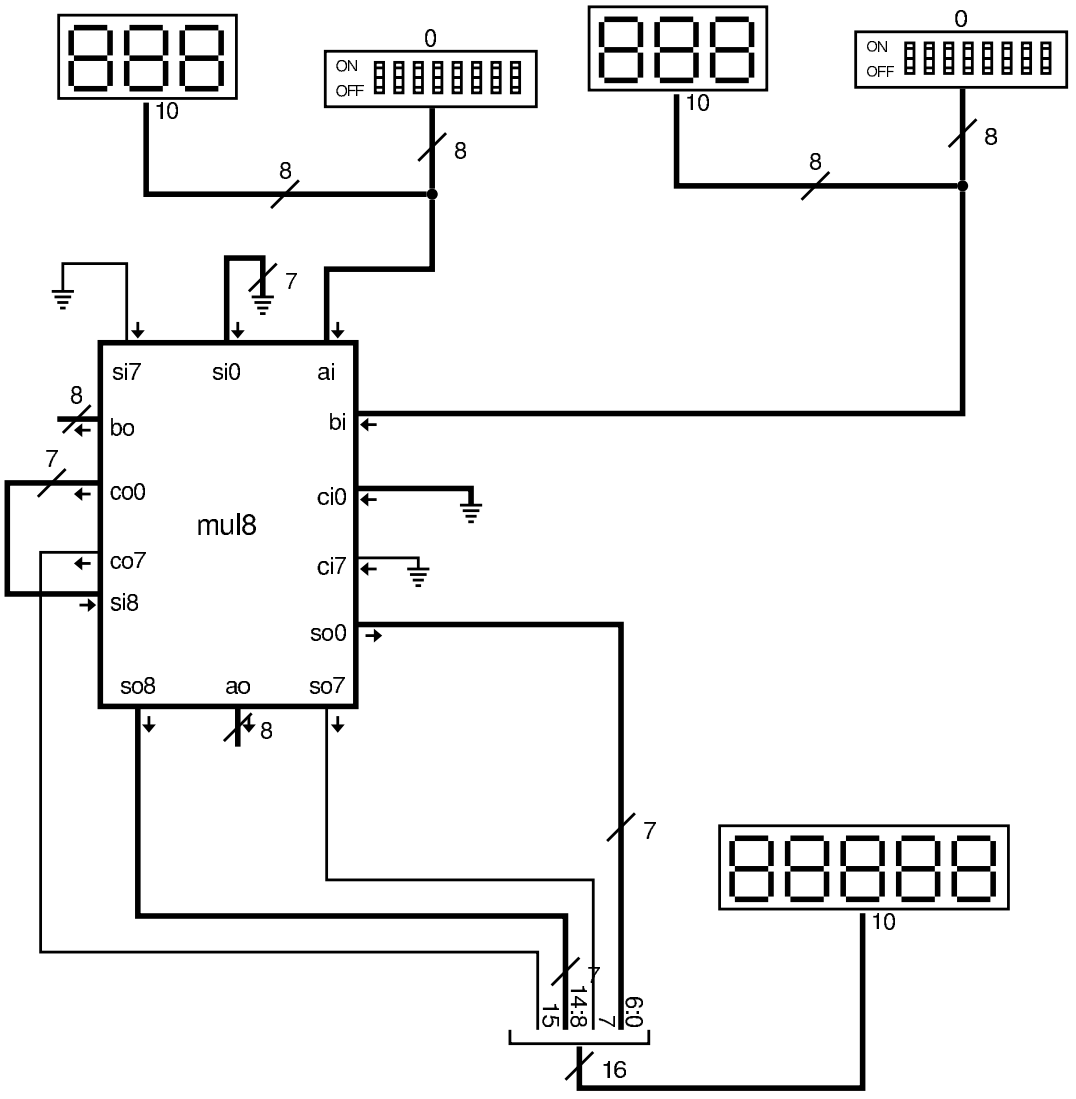
Per la moltiplicazione viene proposto il modulo **`mul8`**, a 8 bit, con tutte le connessioni necessarie a collegarlo ad altri moduli uguali, per realizzarne di più grandi in forma compatta.

Figura u99.18. Modulo **`mul8`**: moltiplicazione a 8 bit.



Dato che il modulo **`mul8`** è abbastanza complesso a causa delle connessioni di cui dispone, nella figura successiva si vede come potrebbe essere utilizzato da solo, per numeri a 8 bit.

Figura u99.19. Utilizzo del modulo **`mul8`** da solo.



Per procedere gradualmente, si vede anche come potrebbe essere sviluppato un modulo a 16 bit, per il quale servono quattro moduli **`mul8`**.

Figura u99.20. Modulo **`mul16`**: moltiplicazione a 16 bit senza segno. Il modulo **`mul8`** è descritto nella figura u99.18.

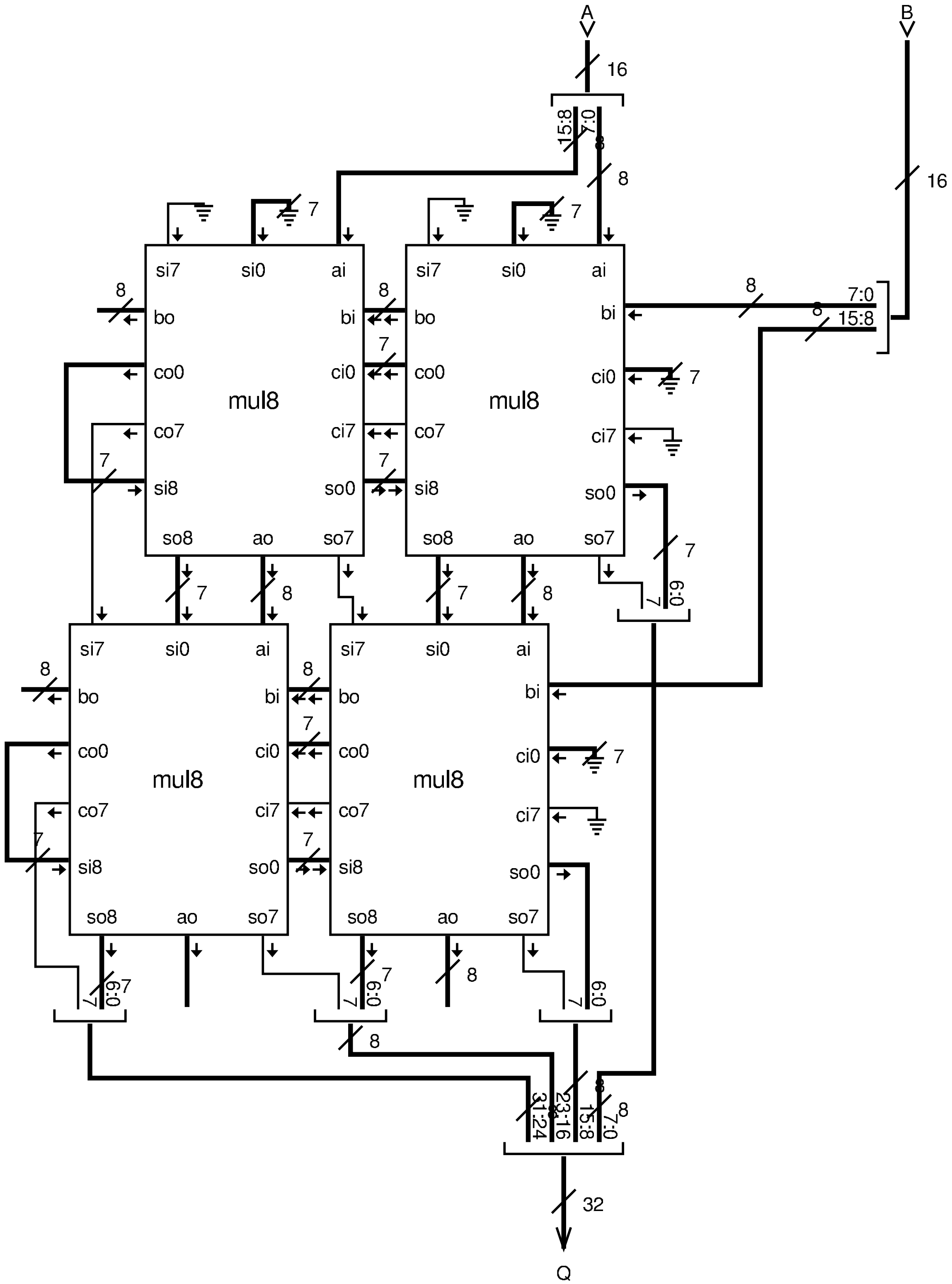
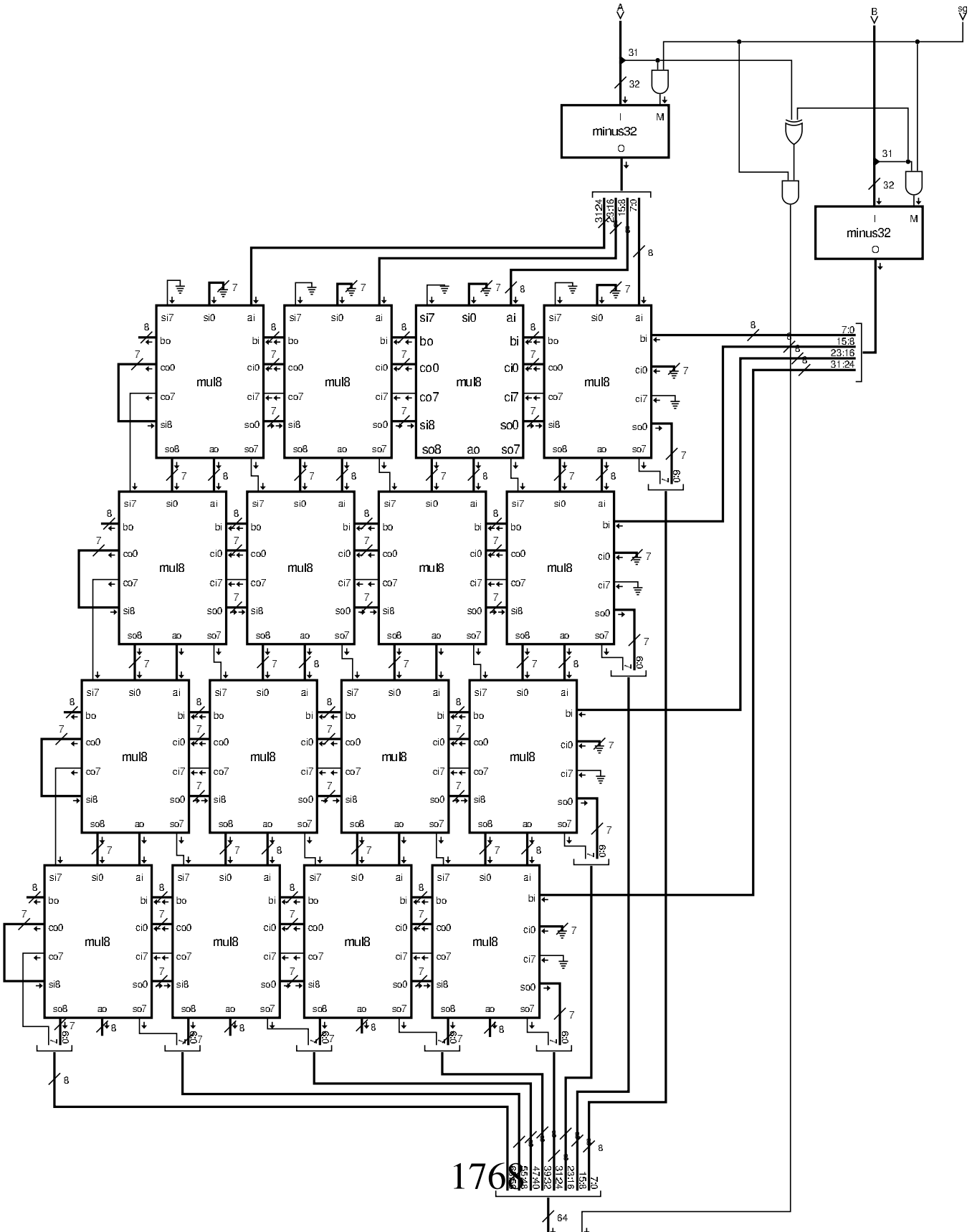


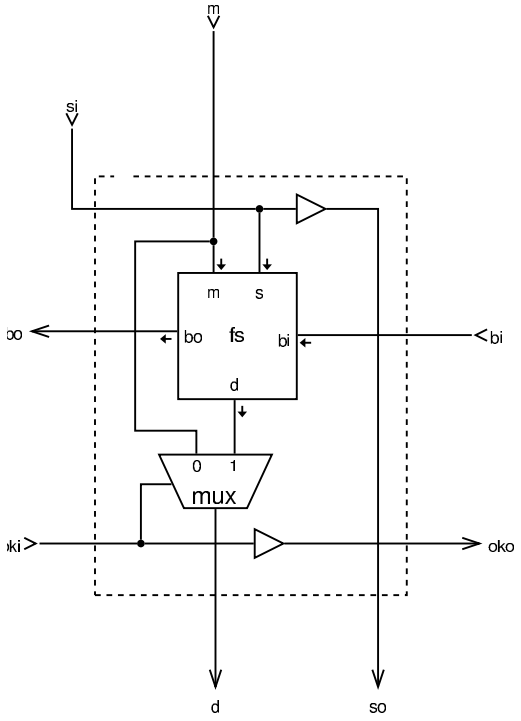
Figura u99.21. Modulo **`mul32`**: moltiplicazione a 32 bit con segno. Il modulo **`mul8`** è descritto nella figura u99.18, mentre i moduli **`minus`**... sono descritti a partire dalla figura u99.6.



Divisione

Per la divisione ci si avvale di un modulo che esegue la sottrazione del divisore dal dividendo, in fasi successive. Il modulo in questione è già apparso in precedenza nel capitolo, ma viene ripresentato per maggiore comodità.

Figura u99.22. Modulo **`div`**: componente usato per il calcolo della divisione, attraverso la sottrazione del divisore dal sottraendo. Questo modulo consente di operare su un solo bit. Ci si avvale del modulo **`fs`**, corrispondente a un sottrattore completo (*full subtractor*), descritto nella sezione u0.10.



fs = full subtractor, sottrattore completo
m = minuendo
si = sottraendo in ingresso
so = sottraendo, copiato in uscita
bi = borrow in, richiesta di prestito di una cifra dal modulo precedente
bo = borrow out, richiesta di prestito di una cifra al modulo successivo
oki = ok in, la sottrazione è valida, ingresso
oko = ok out, la sottrazione è valida, copia in uscita

Figura u99.23. Modulo **`div8`**: divisione intera a 8 bit, senza segno. Si utilizzano 64 moduli **`div`** per sottrarre dal dividendo il divisore, fino a trovare il quoziente e il resto. Il modulo è organizzato in modo da potersi connettere con altri moduli uguali e operare su interi aventi un numero maggiore di cifre. Il modulo **`div`** è descritto nella figura u99.22.

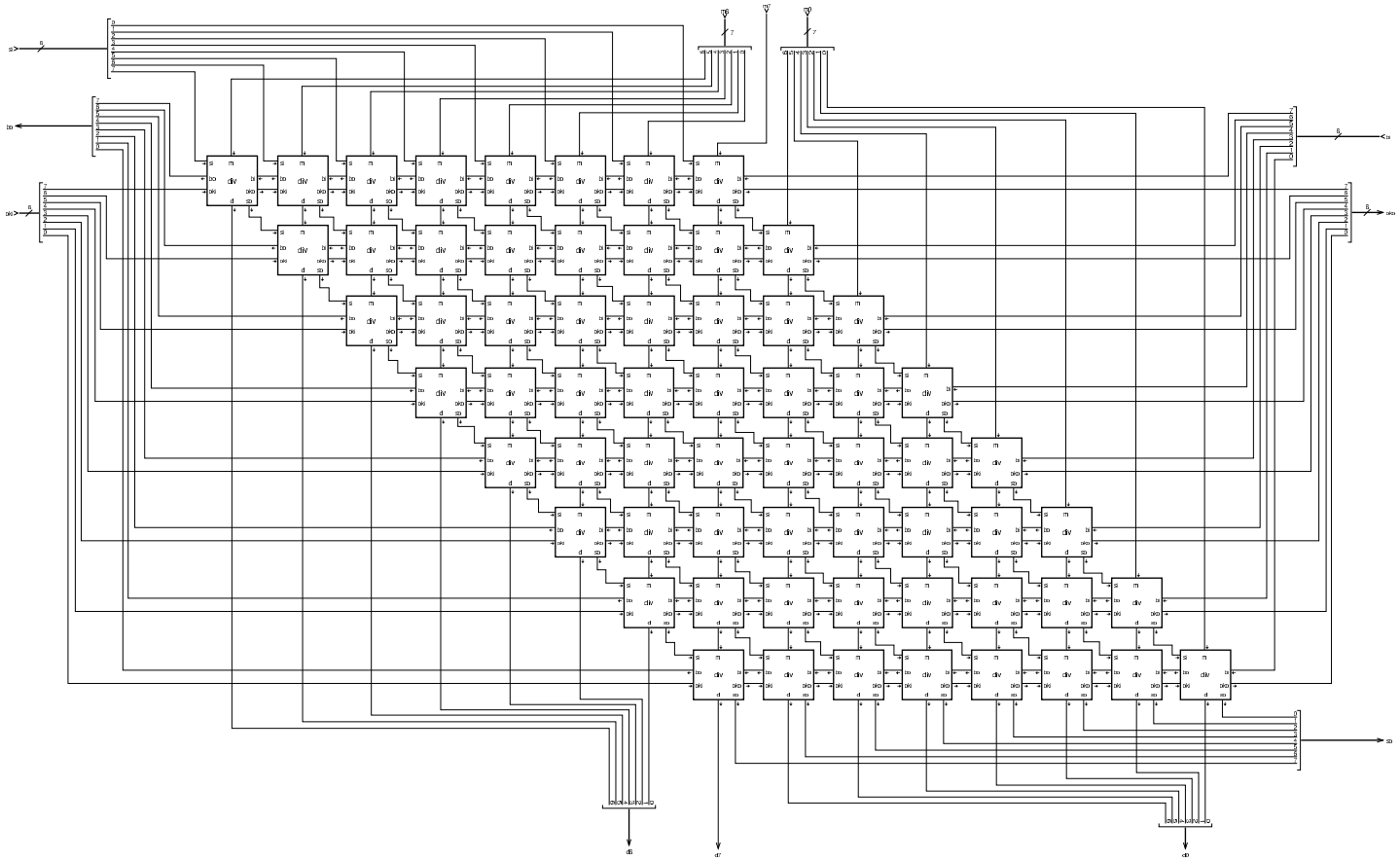


Figura u99.24. Esempio di utilizzo del modulo **`div8`**, per dimostrare in che modo vanno usati i collegamenti di cui è provvisto. Il modulo **`div8`** è descritto nella figura u99.23.

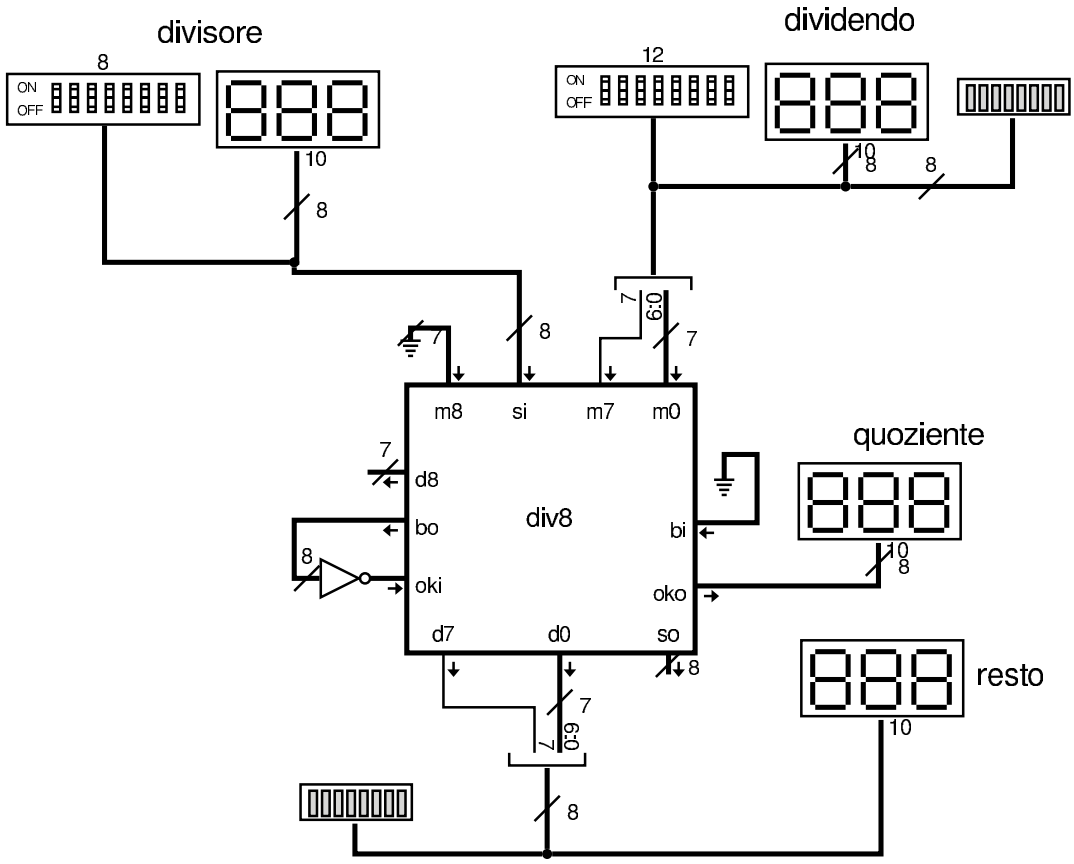


Figura u99.25. Modulo **`div16`**: divisione intera, senza segno, a 16 bit. Si utilizzano quattro moduli **`div8`** collegati opportunamente tra loro. Il modulo **`div8`** è descritto nella figura u99.23.

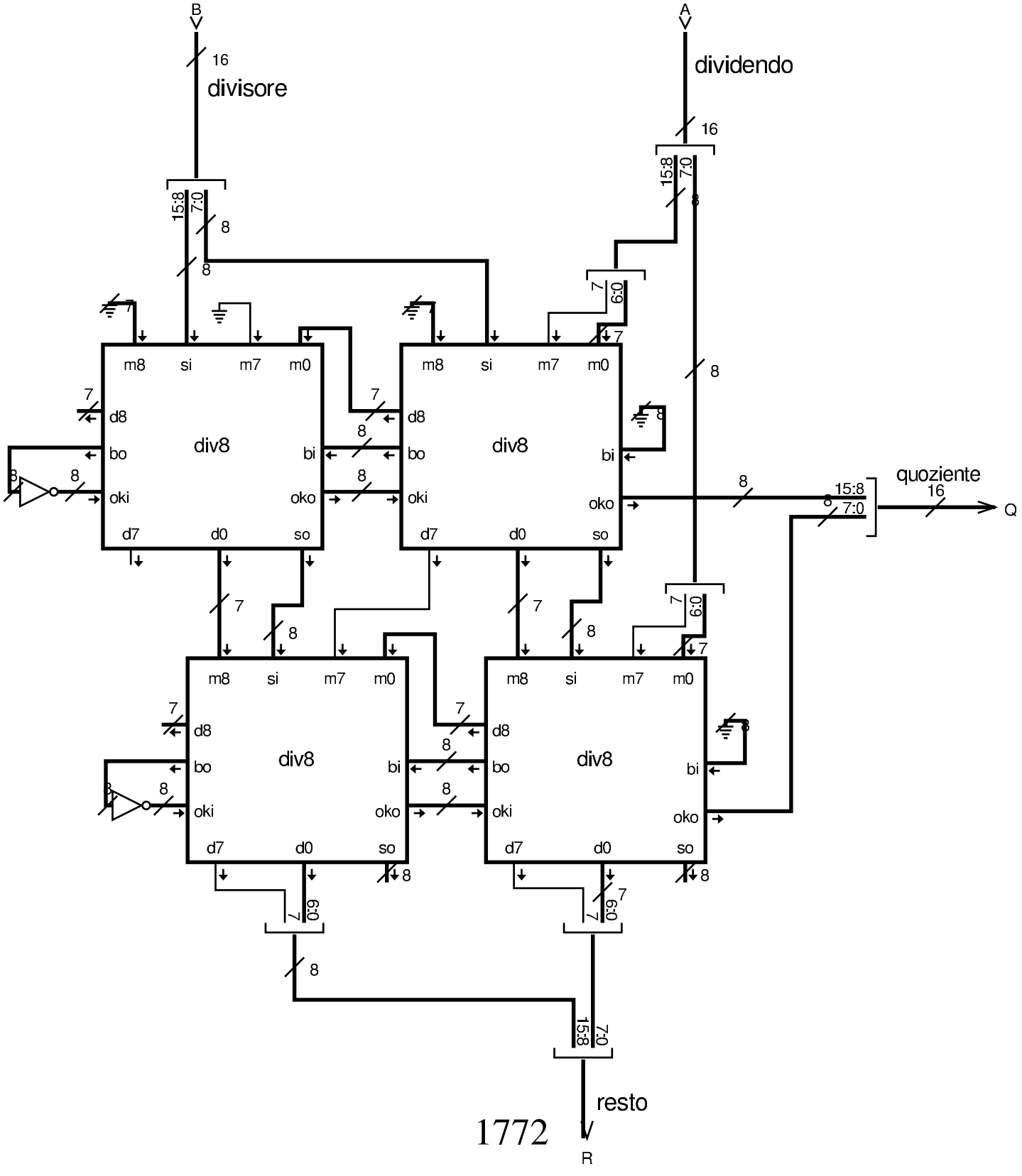
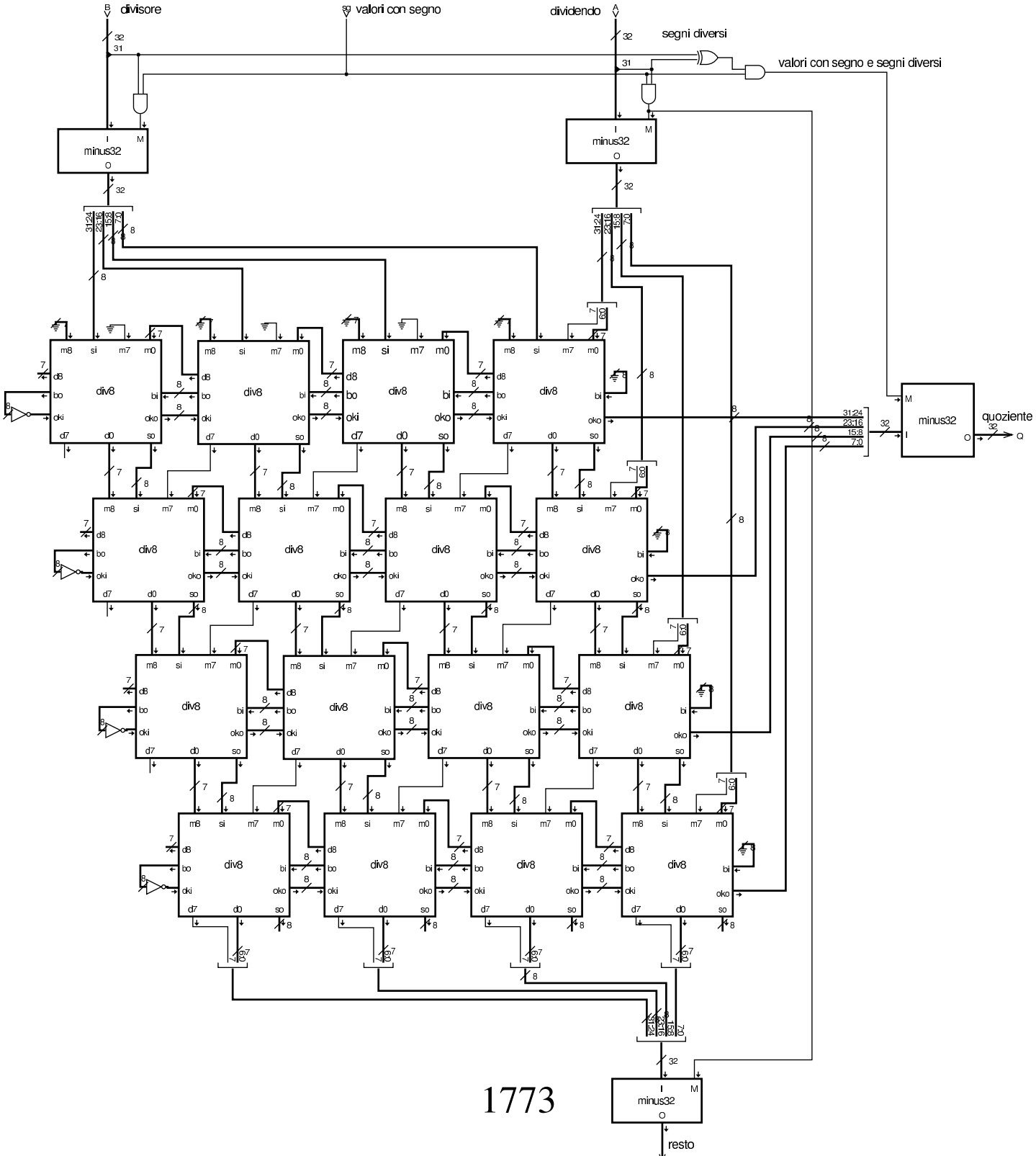


Figura u99.26. Modulo **`div32`**: divisione intera, con o senza segno, a 32 bit. Si utilizzano sedici moduli **`div8`** e quattro moduli **`minus32`** per il controllo del segno, se è richiesto un calcolo che ne tenga conto. Il modulo **`div8`** è descritto nella figura u99.23, mentre il modulo **`minus32`** è descritto nella figura u99.7.

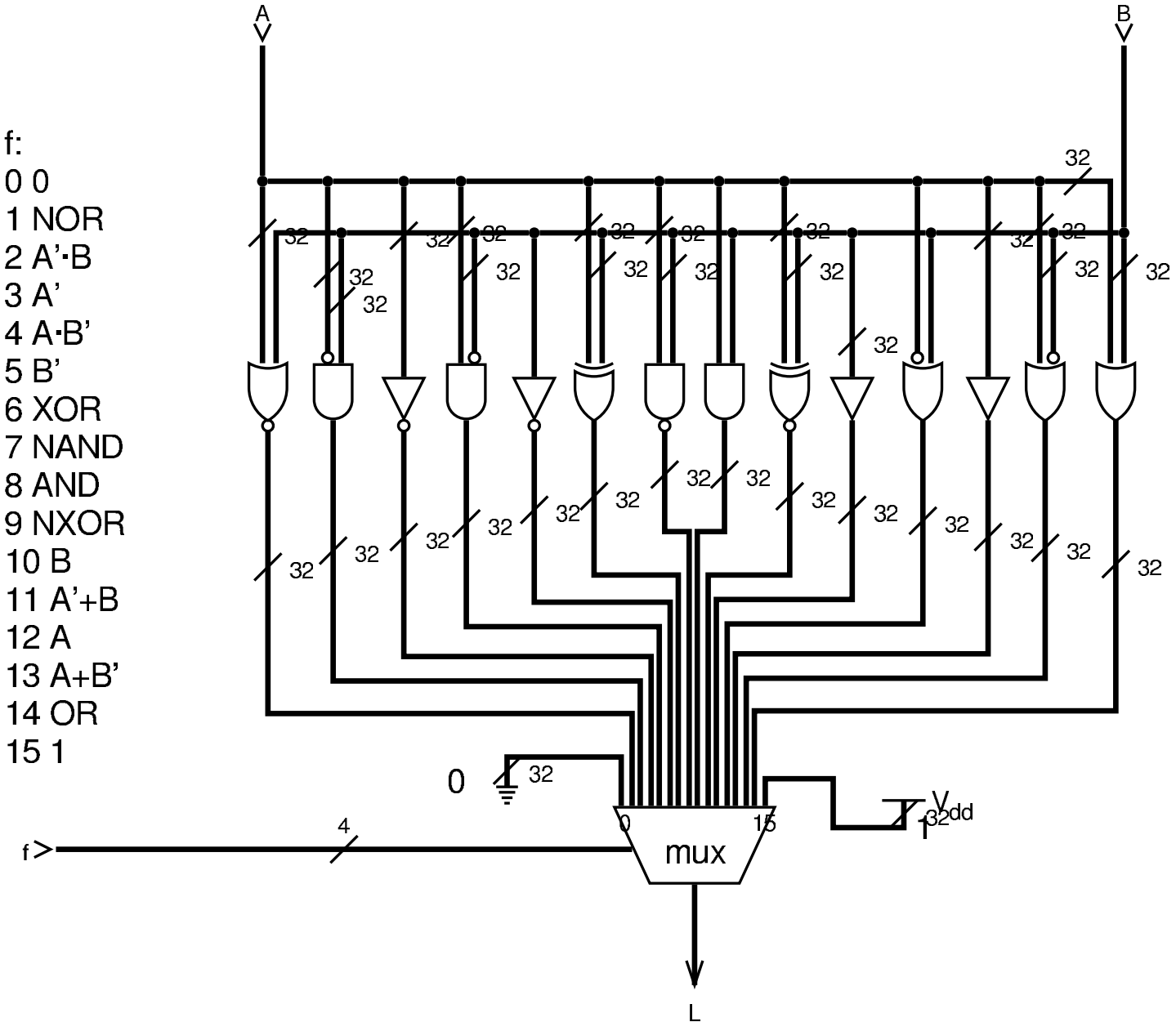


Unità logica

«

Per unità logica si intende quella che esegue operazioni booleane, bit per bit, su valori interi espressi in forma binaria. Di solito le operazioni disponibili sono poche (AND, OR, NOT, XOR), ma nell'esempio della figura successiva appaiono tutti i casi già descritti nella tabella u98.4, anche se ciò può essere superfluo.

Figura u99.27. Modulo **`logic32`**: realizza le 16 operazioni logiche che si possono ottenere da un circuito combinatorio con due ingressi e un'uscita. In questo caso, ogni componente ne rappresenta in realtà 32, in parallelo: solo l'ingresso ***f*** che serve a selezionare la funzione, è sempre lo stesso.



ALU completa

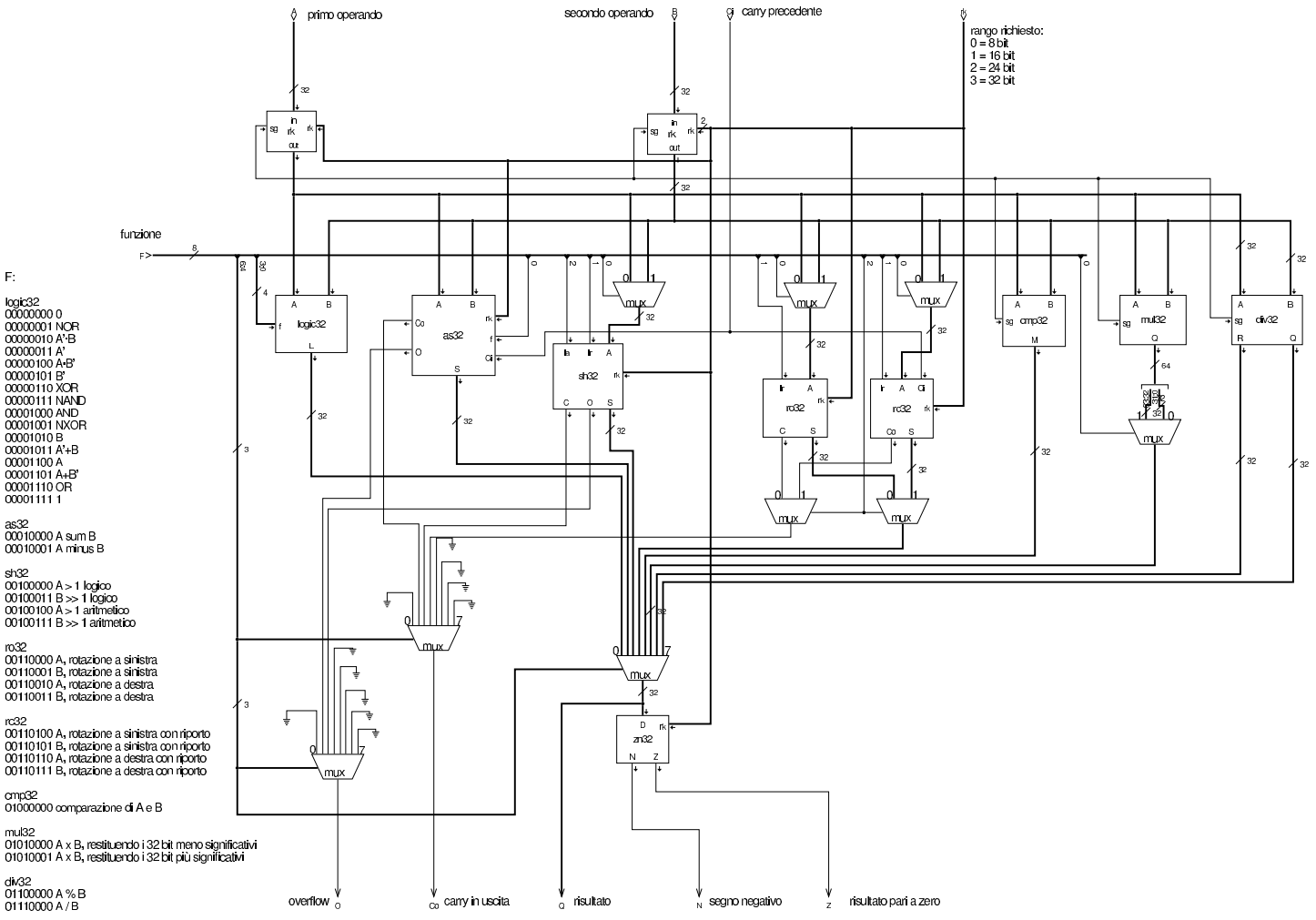
«

Nella figura successiva viene proposta una, ALU completa di tutti i moduli descritti in questo gruppo di sezioni. Valgono le osservazioni seguenti:

- i moduli della rotazione sono gestiti come se fossero uno solo, in cui la funzione di rotazione con o senza riporto avviene in base alla selezione del tipo di scorrimento; inoltre, sia per lo scorrimento, sia per la rotazione, è possibile scegliere su quale ingresso intervenire;
- per risolvere in qualche modo il problema del risultato della moltiplicazione, che occupa 64 bit, si è scelto di selezionare quale porzione del risultato si vuole ottenere, anche se ciò comporta in pratica un raddoppio del tempo necessario alla moltiplicazione, perché ogni volta il calcolo viene ripetuto;
- i moduli pescano dalla linea dell'ingresso $\boldsymbol{F}$ (*function*) i bit che gli servono per adeguare il proprio comportamento, tenendo conto che i primi quattro bit di $\boldsymbol{F}$ servono ai multiplatori che selezionano le uscite da prelevare, mentre i quattro bit più significativi sono quelli che sono affidati ai moduli rispettivi.

Si comprende che si tratta di una soluzione che non è ottimale dal punto di vista delle prestazioni, avendo soltanto uno scopo dimostrativo.

Figura u99.28. Modulo **`alu32`**: ALU completa a 32 bit. L'ingresso ***F*** determina il comportamento della ALU.



Unità aritmetico-logica e registri espandibili

«

In questa sezione viene presentato uno studio alternativo per la costruzione di una ALU da un solo bit, allineabile in parallelo per ottenere ranghi superiori, assieme a dei registri, anche questi a singolo bit, ma espandibili. Per comprendere il funzionamento e l'uso dei registri, è comunque necessario leggere la descrizione relativa ai flip-flop che appare a partire dalla sezione u101.

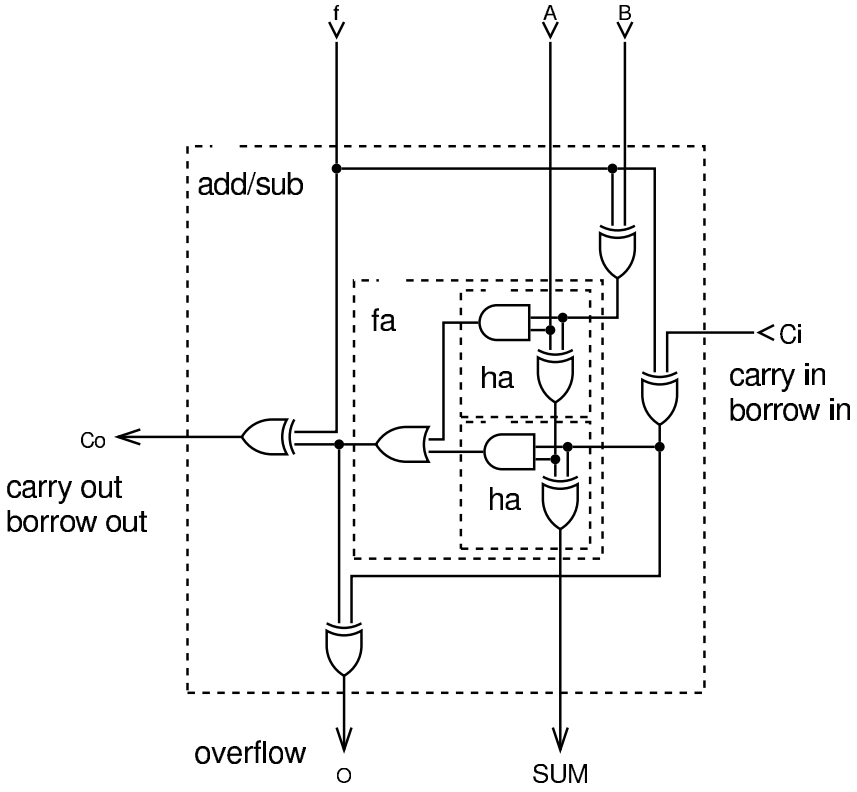
Il file sorgente di questo studio dovrebbe essere disponibile presso *allegati/circuiti-logici/modular-alu-0050.v* .

Unità aritmetico-logica a bit singolo

«a2» 2013.11.11 --- Copyright © Daniele Giacomini -- appunti2@gmail.com http://informaticalibera.net

«

Figura u100.1. Modulo **as**: per f=0, somma con riporto; per f=1 sottrazione con prestito; se il riporto (o il prestito) in ingresso è diverso da quello in uscita, si verifica lo straripamento.



f=0
viene calcolata la somma (A + B) e generato eventualmente il riporto;
Ci e Co funzionano in qualità di riporti (in ingresso e in uscita).

f=1
viene calcolata la sottrazione (A – B) e generato eventualmente la richiesta di prestito;
Ci e Co funzionano in qualità di richiesta di prestito (borrow).

Figura u100.2. Modulo **`logic`**: unità logica, comandata dall'ingresso f che seleziona il tipo di operazione logica. Nella didascalia interna dei valori assegnabili all'ingresso f, il simbolo «+» va inteso come operatore OR.

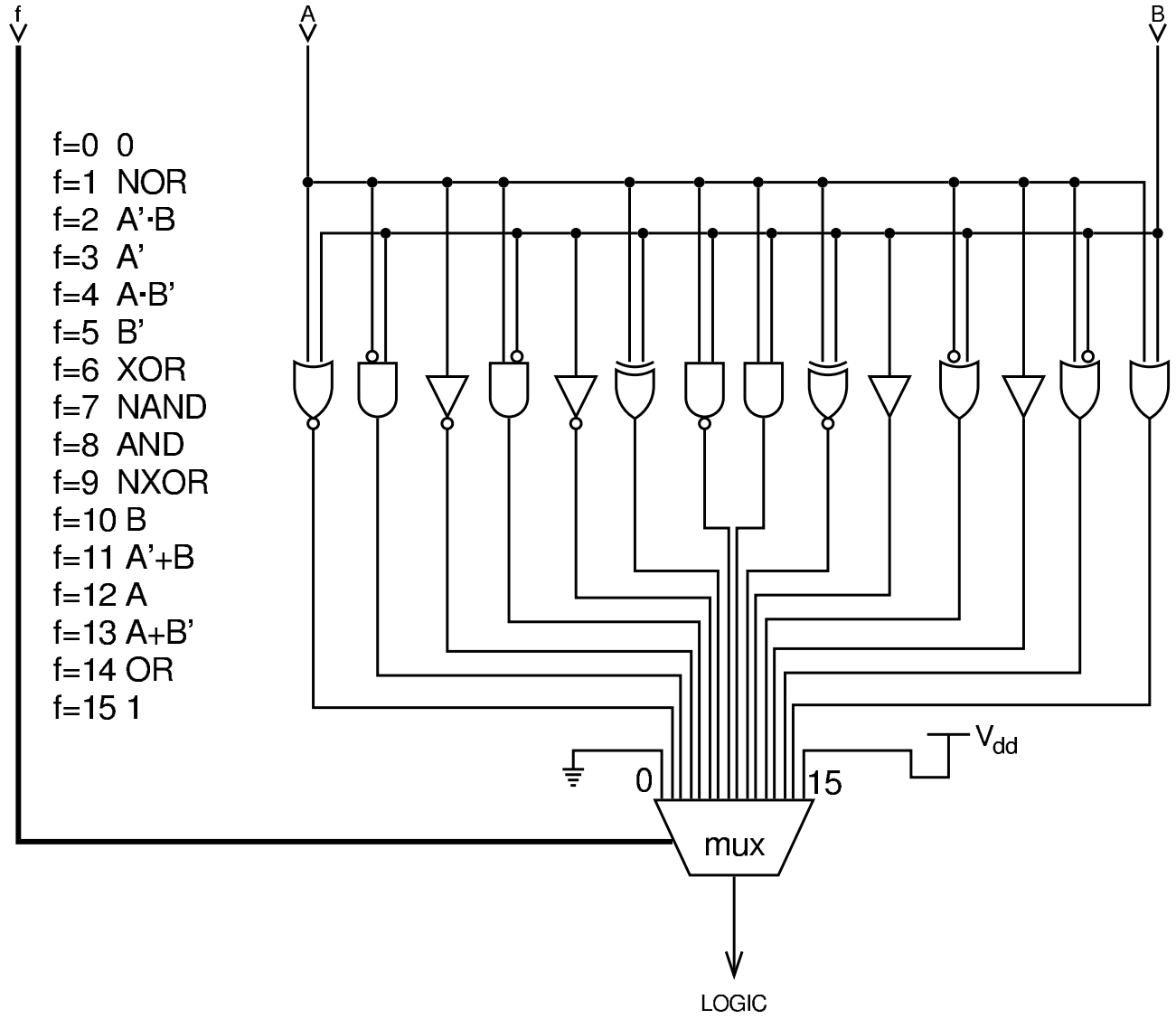


Figura u100.3. Modulo **`ltgt`**: confronto. Data la necessità di realizzare una ALU espandibile, il confronto avviene a livello di bit, usando il riporto in ingresso in caso di uguaglianza. L'esito viene trasmesso attraverso il riporto in uscita. Il modulo è composto da due parti; nella figura, nell'ordine, appaiono i moduli: **`lt`** (A<B), **`gt`** (A>B), **`ltgt`**. Il modulo complessivo **`ltgt`** esegue il primo o il secondo confronto, in base al valore contenuto nell'ingresso ***f***. Il controllo di uguaglianza non viene fatto, perché per ottenerlo è sufficiente sfruttare l'unità logica con l'operatore NXOR.

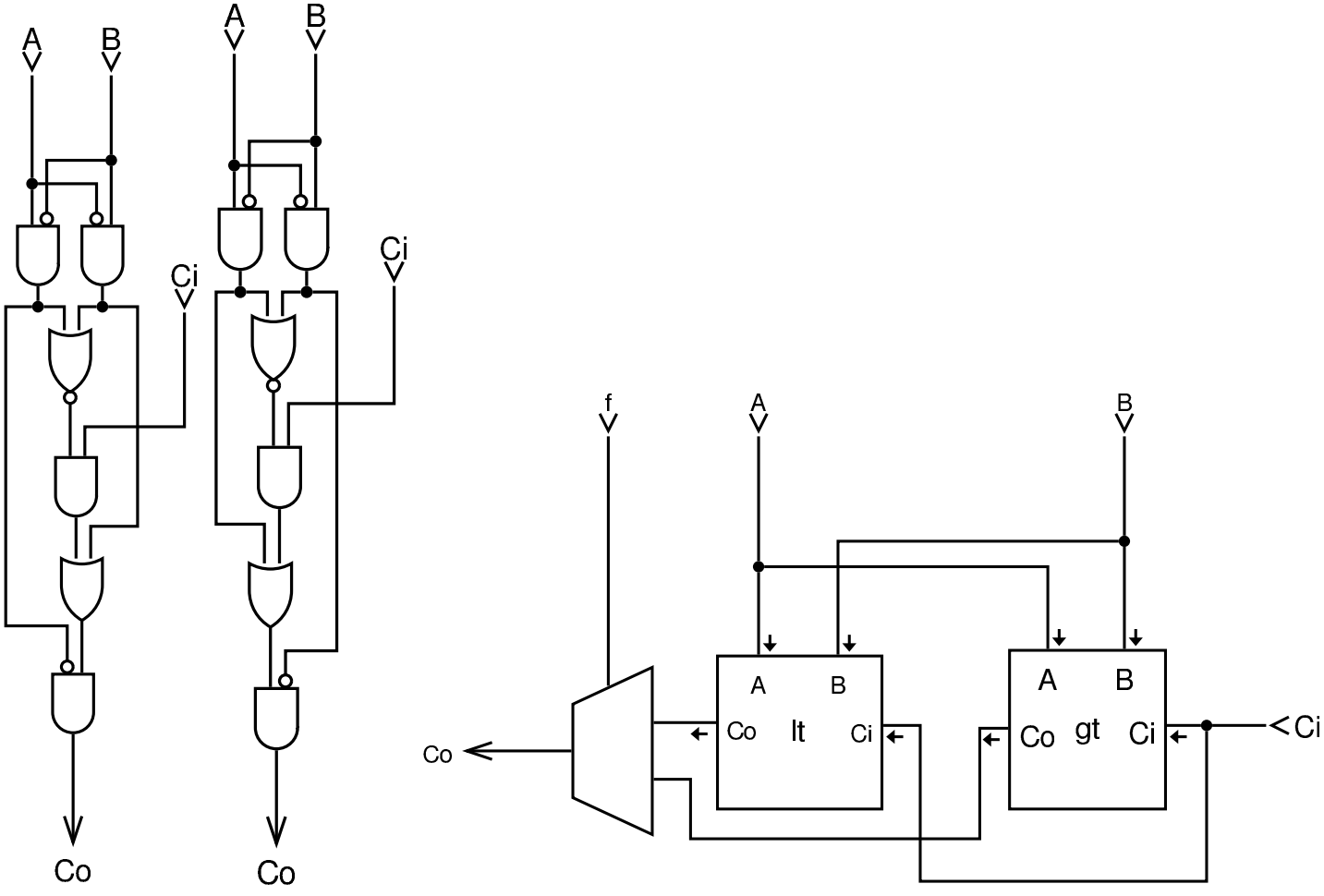


Figura u100.4. Moduli **`shl`** e **`shr`**: scorrimento a sinistra e a destra. Lo scorrimento avviene necessariamente utilizzando il riporto; per la precisione, per lo scorrimento a sinistra si usa il riporto «normale», mentre per lo scorrimento a destra si usa un riporto apposito, da sinistra a destra, indicato con le sigle ***Di***, ***Do***. Il riporto a destra non tiene conto del segno, perché in questa fase si considera un solo bit. L'ingresso ***f*** serve, in entrambi i casi, a selezionare l'operando sul quale intervenire: ***A*** o ***B***.

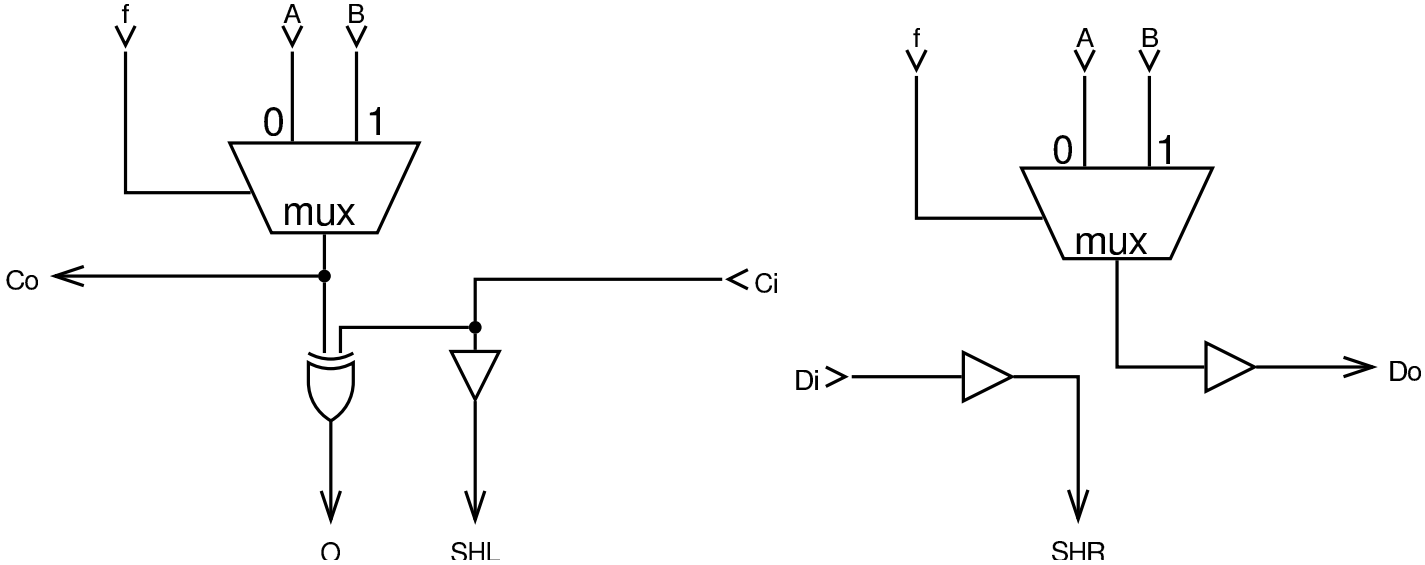


Figura u100.5. Modulo **z**: Zero. Determina se il valore è pari a zero e se lo sono anche i moduli precedenti (attraverso il riporto). Se fino a questa cifra i valori sono a zero, attiva il riporto in uscita. L'ingresso ***f*** consente di selezionare quale operando verificare.

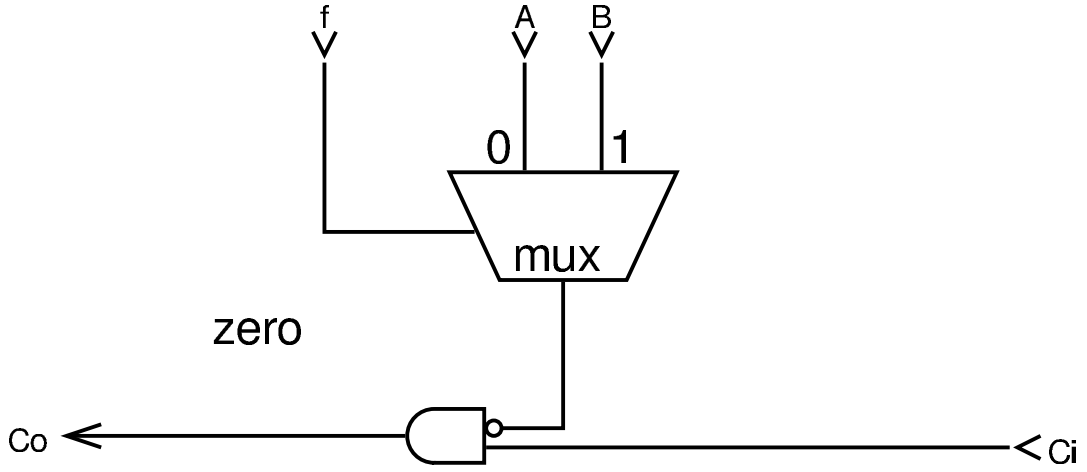


Figura u100.6. Modulo **`alu`**: ALU completa dei vari componenti aritmetici, logici e di scorrimento. Quando l'ingresso f ha il bit più significativo pari a zero, contiene la funzione dell'unità logica, altrimenti contiene le funzioni per gli altri moduli. Gli altri moduli, quando contengono a loro volta l'ingresso f, questo lo traggono da f_3 di quello principale, ovvero dal quarto bit.

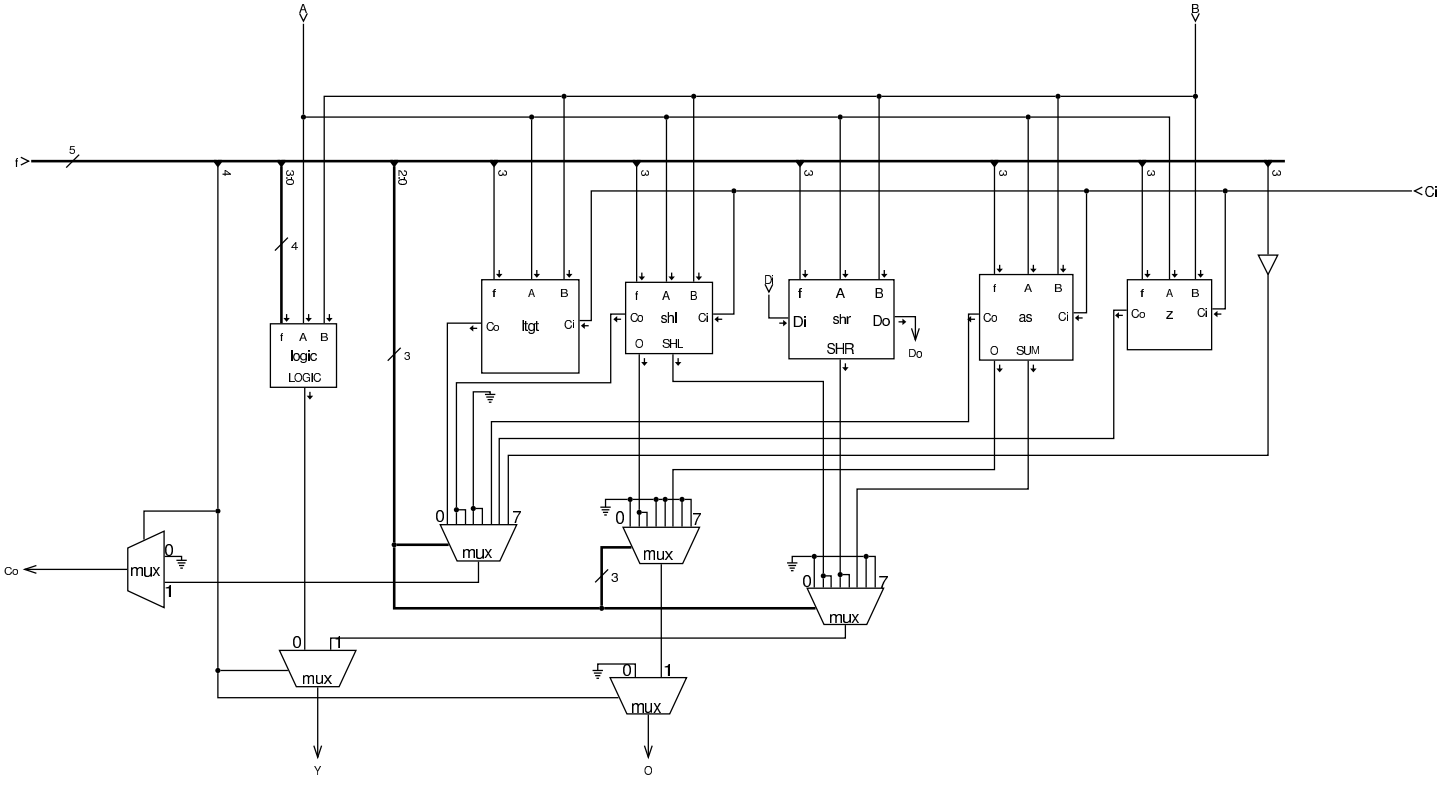
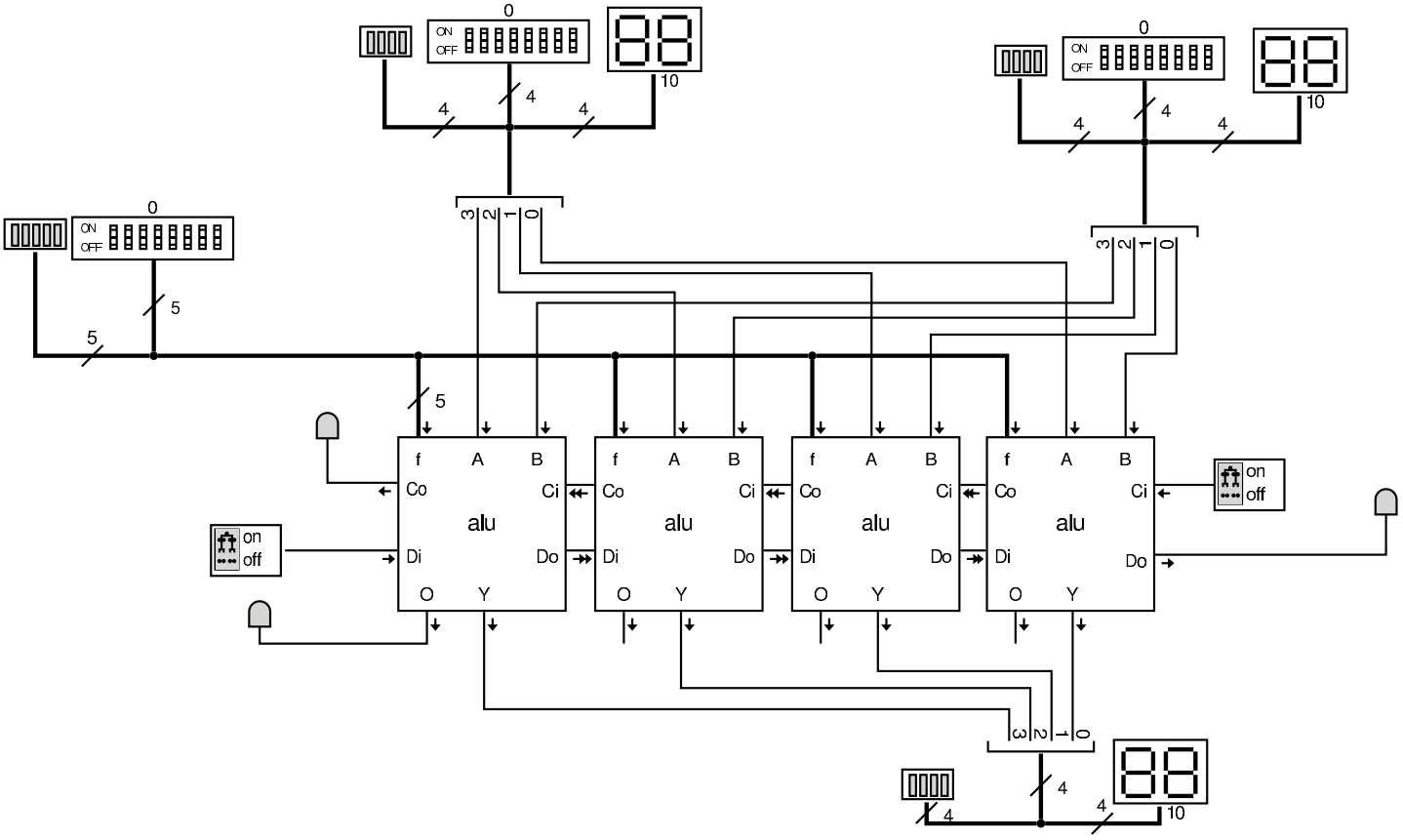


Figura u100.7. Esempio di utilizzo in parallelo del modulo **`alu`**, per ottenere un'elaborazione a quattro bit. Vanno osservate in particolare le connessioni delle linee di riporto ***C*** e ***D***.



Registri

Attorno alla ALU vengono collocati dei registri, ovvero delle unità di memoria in grado di memorizzare, ognuna, un solo valore. Per i registri si utilizzano flip-flop D, semplici, come nella prima delle figure. I flip-flop sono descritti a partire dalla sezione u101; qui, per comprendere il meccanismo, basti intendere che ricevono l'informazione da memorizzare attraverso l'ingresso ***D***, ma solo quando l'ingresso ***E*** (*enable*) è attivo, quindi mantengono l'informazione disponibile nell'uscita ***Q***.

Figura u100.8. Modulo **d**: Flip-flop D semplice, usato per la realizzazione dei registri.

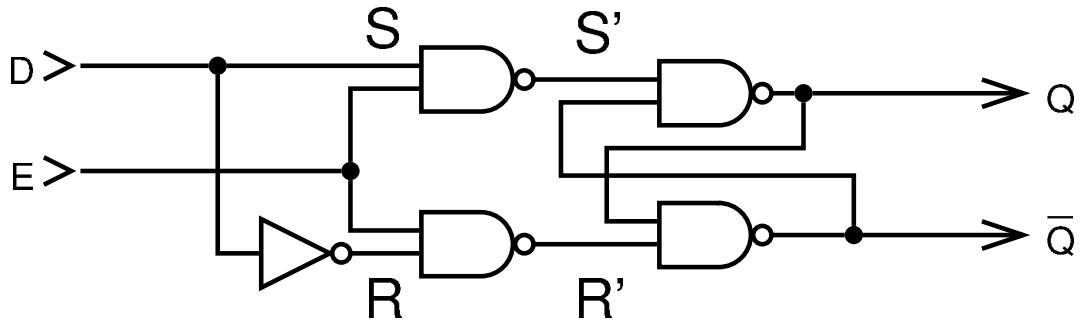


Figura u100.9. Modulo **`ralu`**: ALU e registri, a un solo bit. Gli ingressi ***SRC*** e ***DST***, determinano, rispettivamente, l'origine e la destinazione del movimento di un'informazione, attraverso il bus. In pratica, ***SRC*** seleziona cosa mettere nel bus, e ***DST*** seleziona quale registro deve salvare questa informazione. L'ingresso ***ALU*** contiene la funzione da far svolgere alla ALU, nel caso sia selezionata l'origine corrispondente al terzo registro, che in pratica è l'uscita della ALU stessa. L'ingresso ***E*** va attivato quando tutti gli altri ingressi sono stati configurati correttamente, per il tempo necessario al registro di destinazione per recepire l'informazione contenuta nel bus. Va osservato che la ALU esegue le sue operazioni sulla base del contenuto dei primi due registri, e lo fa continuamente; inoltre, il terzo registro non esiste, in quanto è il riferimento usato per individuare l'uscita della ALU, per cui non è possibile scrivere direttamente in questo terzo registro, senza intervenire con una funzione della ALU. Gli ingressi ***SRC*** e ***DST*** hanno un bit in più, necessario per controllare altri otto registri esterni, necessari per l'accesso alla memoria.

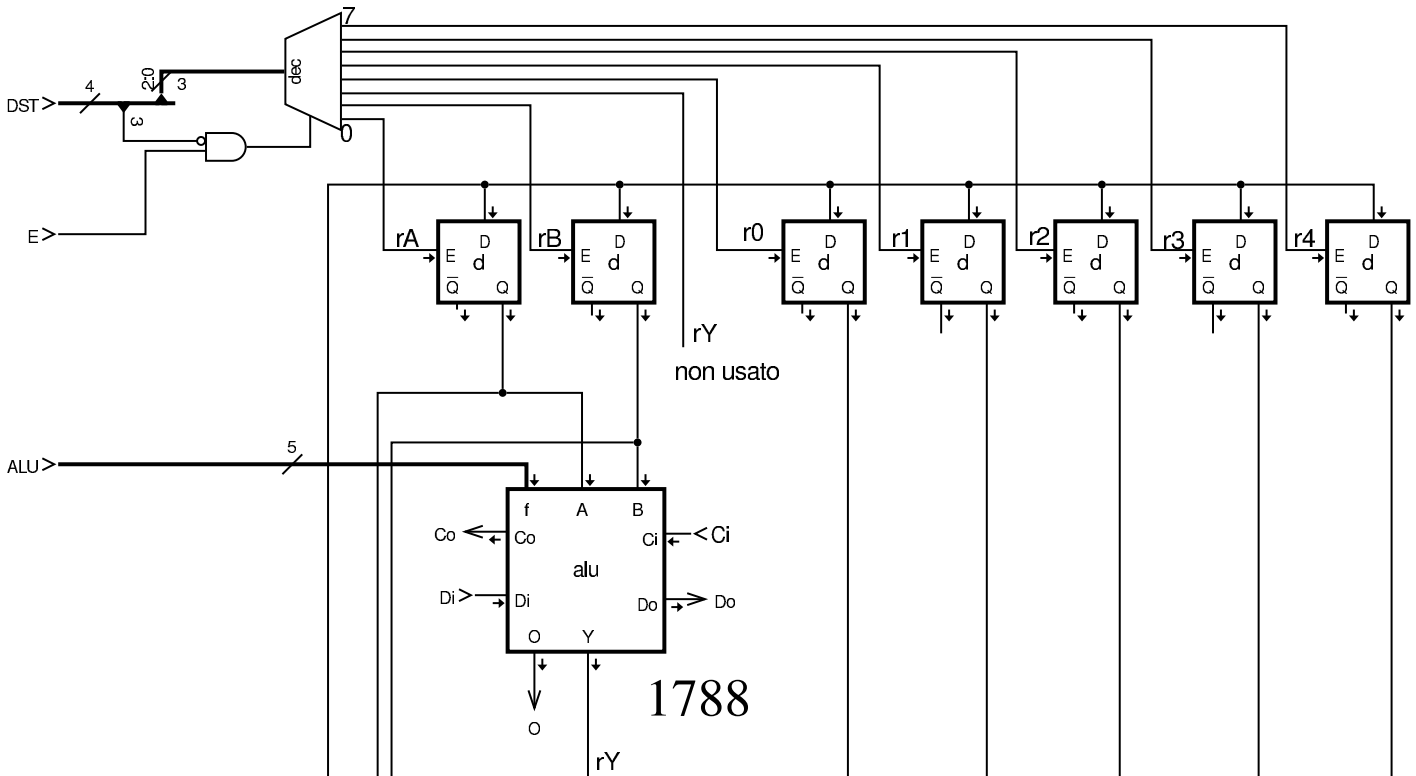


Figura u100.10. Modulo **`ralu4`**: collocazione in serie di quattro moduli a un solo bit. Lo stesso procedimento si seguirebbe per ranghi superiori.

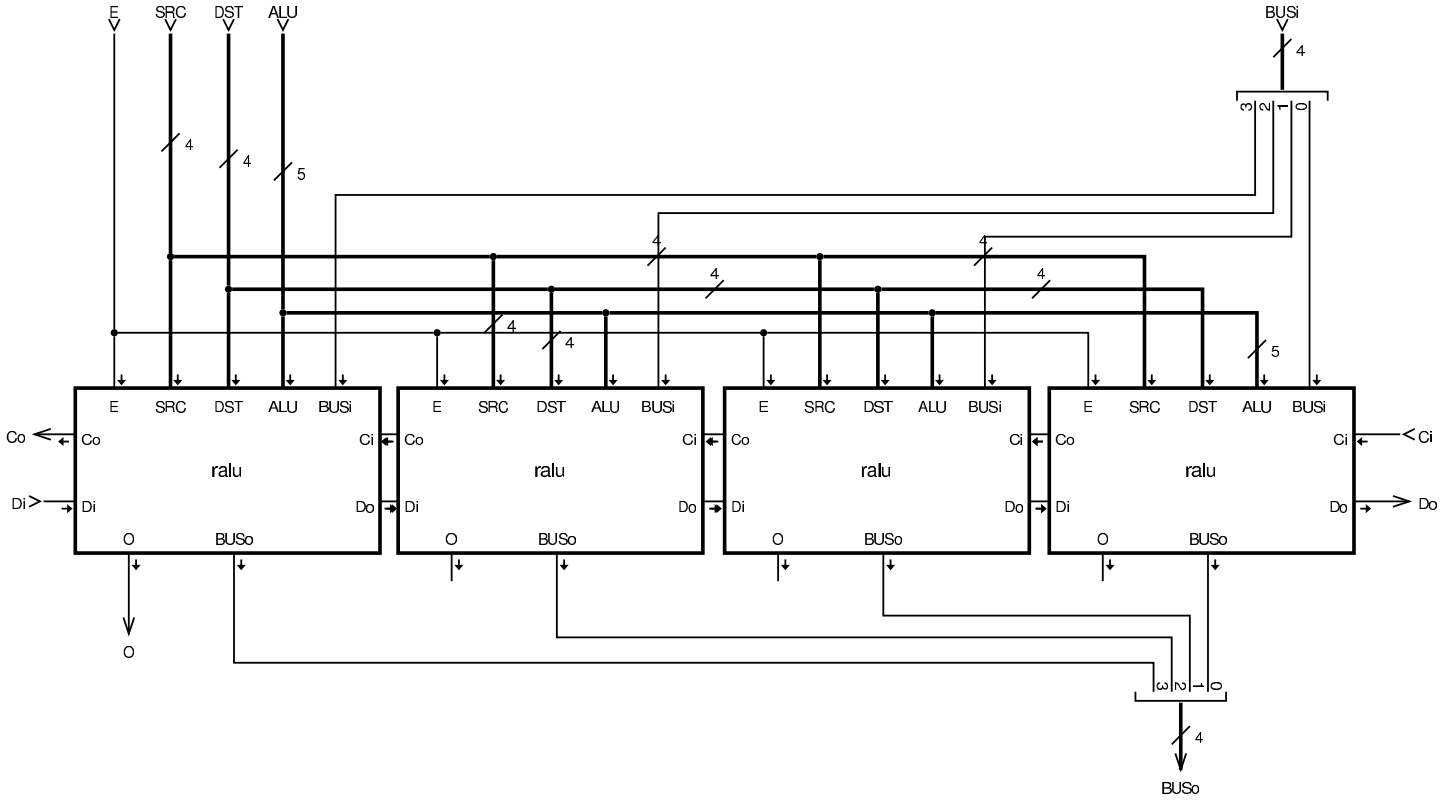
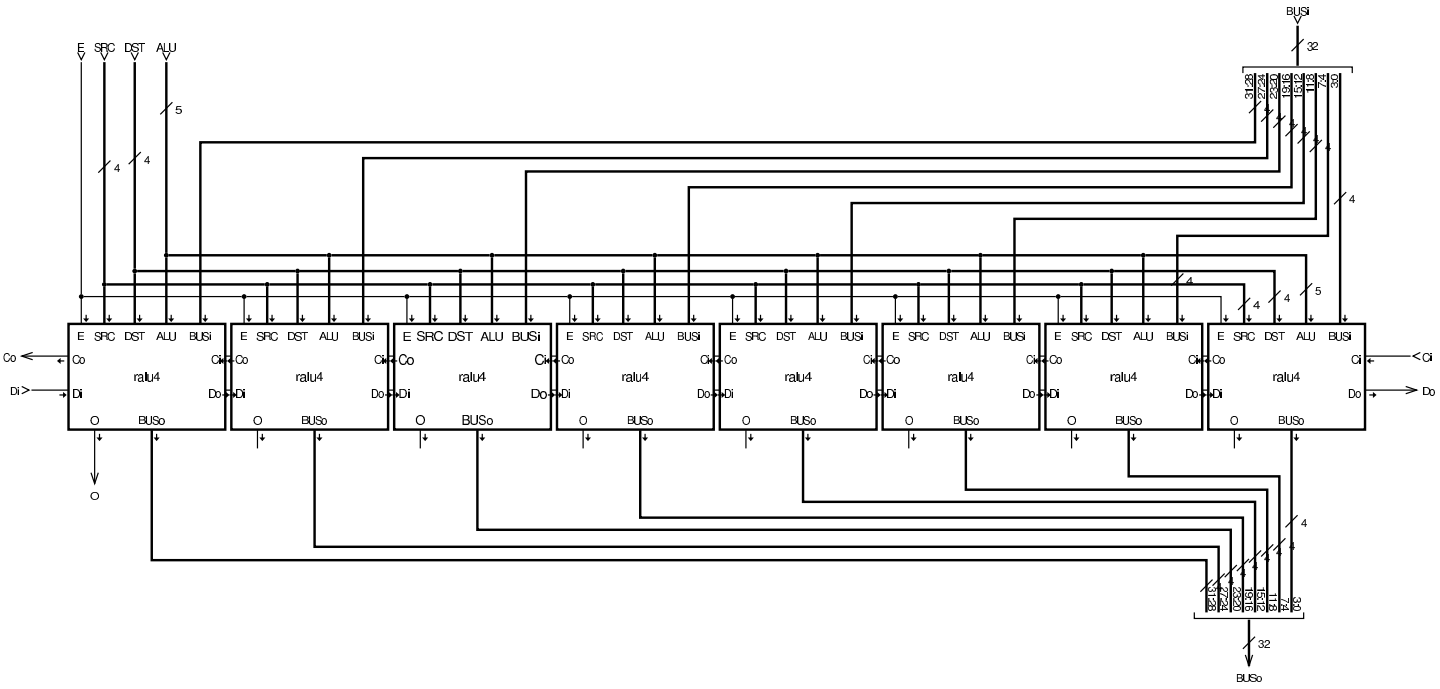


Figura u100.11. Modulo **`ralu32`**: aggregazione di più moduli più piccoli, per un rango più ampio.



Flip-flop

I circuiti combinatori hanno la caratteristica di fornire un certo valore di uscita, unicamente sulla base dei valori presenti in ingresso, anche se la formazione del valore di uscita richiede comunque un piccolo lasso di tempo, rispetto alla disponibilità dei dati di partenza. Pertanto, si dice che i circuiti combinatori non hanno memoria, in quanto non tengono conto di ciò che è avvenuto in precedenza. Al contrario, un ***circuito sequenziale***, tiene conto della dinamica con cui provengono i dati in ingresso, mantenendo uno ***stato***, dal quale dipendono gli effetti dei cambiamenti successivi dei dati in ingresso.

I circuiti sequenziali si basano su componenti noti come ***memorie***, le quali si realizzano attraverso i ***flip-flop***. All'interno della categoria delle memorie, la documentazione scientifica e tecnica tende a

«a2» 2013.11.11 --- Copyright © Daniele Giacomini -- appunti2@gmail.com http://informaticalibera.net

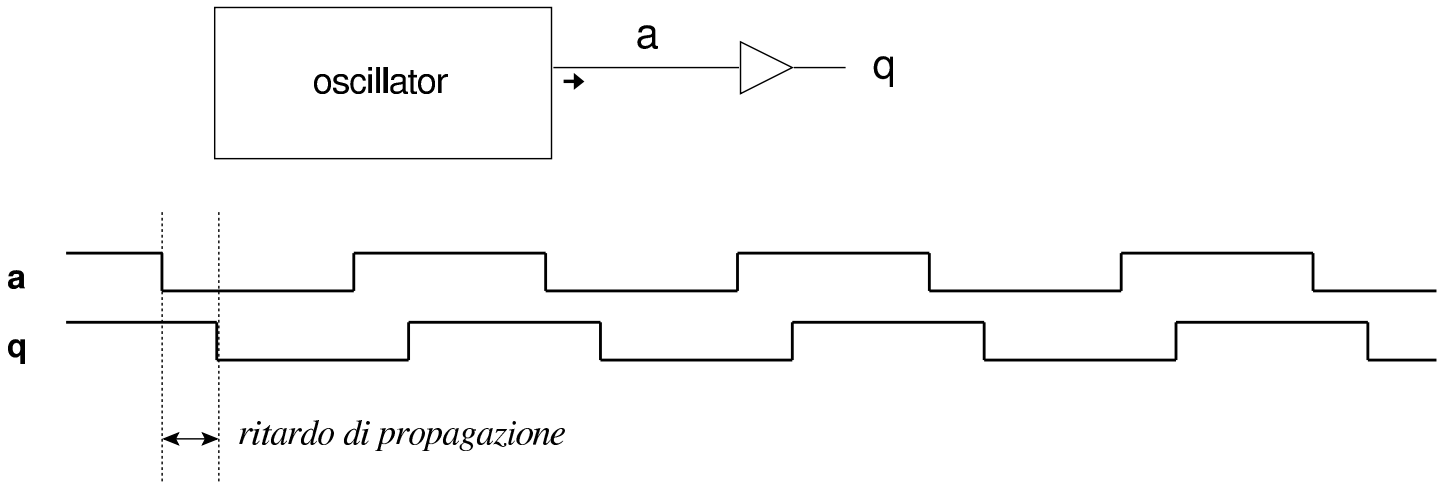
usare il termine ***latch*** che si riferisce alla proprietà di questi circuiti di «scattare» da uno stato a un altro, riservando la definizione di flip-flop solo a un insieme ristretto di tali circuiti, in quanto aventi caratteristiche più sofisticate. Tuttavia, qui si preferisce usare la qualifica di flip-flop per tutti questi circuiti, distinguendo di volta in volta il livello di sofisticazione di ogni variante.

Ritardo di propagazione

«

In un circuito combinatorio comune, perché l'uscita si adegui ai cambiamenti avvenuti in ingresso, si richiede un piccolo intervallo di tempo, noto come ***ritardo di propagazione***. Questo ritardo dipende dalle caratteristiche fisiche e meccaniche del dispositivo con cui si realizza effettivamente il circuito combinatorio.

Figura u101.1. Esempio di ritardo nell'attraversamento di un *buffer*, ovvero di una porta logica non invertente: come si intende dallo schema, l'ingresso è pilotato da un oscillatore che produce un segnale alterno e nell'uscita si ottiene lo stesso segnale, ma leggermente ritardato.



Il ritardo di propagazione si può sfruttare per ottenere un impulso molto breve, come si vede nella figura successiva.

Figura u101.2. Esempio di impulso generato sfruttando il ritardo di propagazione. In questo schema si intende che gli ingressi della porta AND siano bilanciati correttamente, nel senso che il cambiamento di stato da un ingresso o dall'altro, si rifletta sull'esito restituito nello stesso tempo.

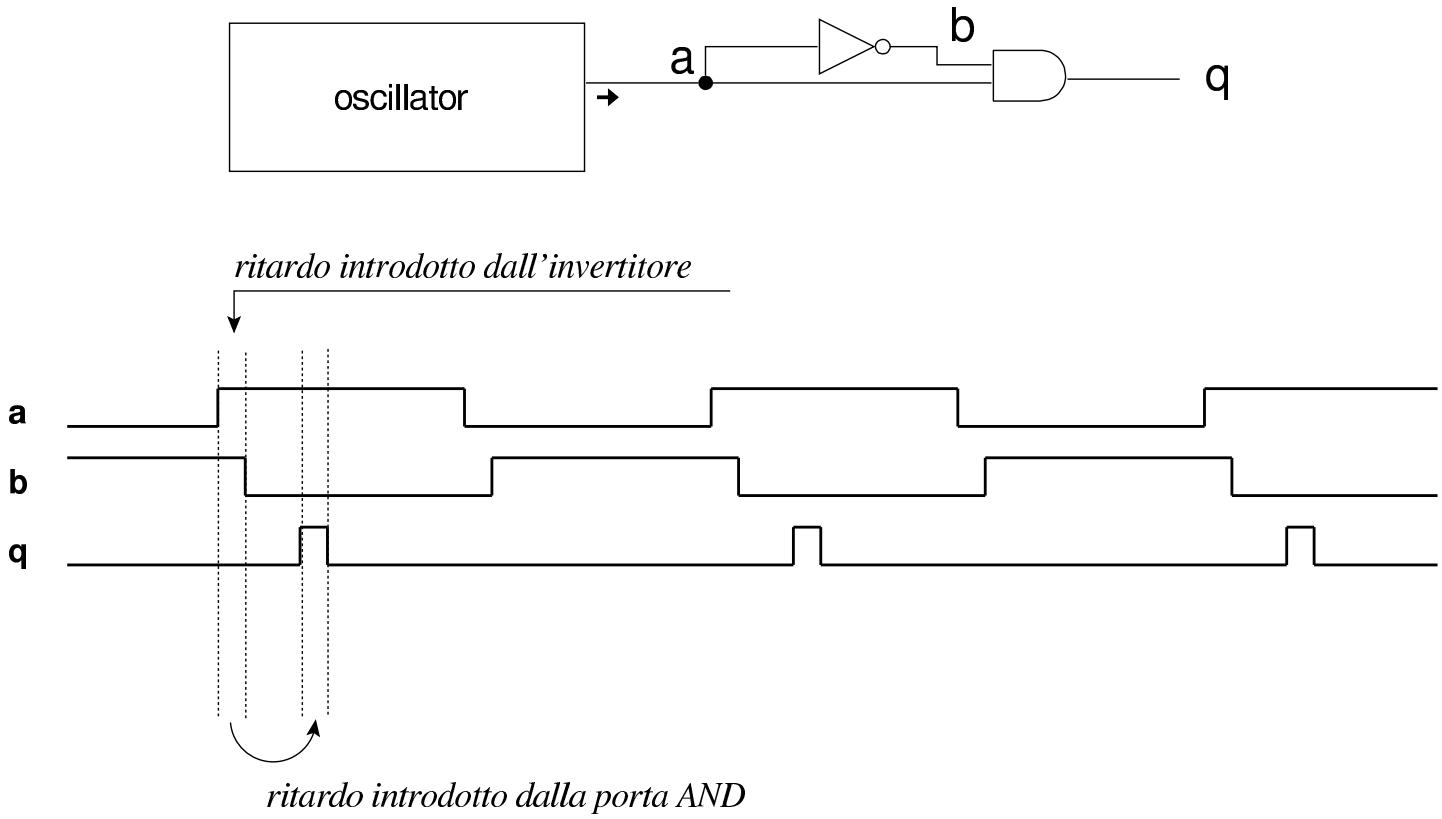


Tabelle di verità

La tabella di verità è un modo con cui si rappresenta il comportamento di un circuito combinatorio, per cui si mostrano i valori in uscita come funzione dei valori in entrata, senza contare il ritardo di propagazione e la dinamica con cui si formano i dati in ingresso. Trattando invece di circuiti sequenziali, le tabelle di verità, ammesso che si utilizzino, vanno interpretate di volta in volta in base al contesto particolare. Nelle sezioni successive si introducono i flip-flop e nelle tabelle di verità si vuole considerare negli ingressi anche la

variazione di stato; pertanto si usa la simbologia seguente:

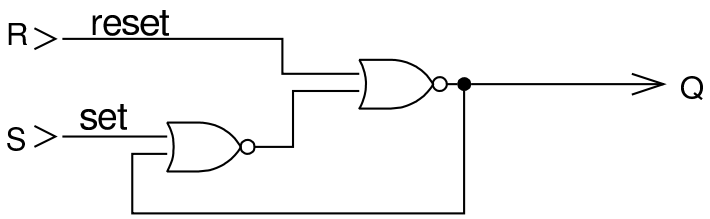
0	zero stazionario
_/‾	variazione da zero a uno
1	uno stazionario
‾_	variazione da uno a zero

Flip-flop SR elementare

«

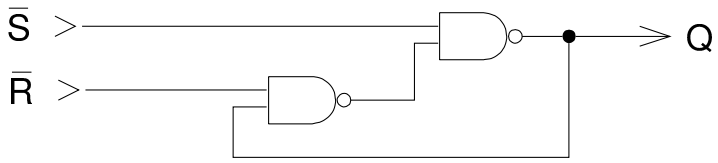
L'esempio più semplice di circuito sequenziale che mantiene la memoria del proprio stato, è quello che si può vedere nelle due figure successive, realizzato rispettivamente con porte NOR e NAND. Nel caso del circuito con porte NOR, si parte da uno stato iniziale in cui l'uscita ***Q*** ha un valore indeterminabile (potrebbe essere zero oppure uno, in base a fattori imponderabili), quindi, attivando l'ingresso ***S*** (*set*), anche solo con un breve impulso, l'uscita ***Q*** si attiva e rimane attiva fino a quando non riceve un impulso dall'ingresso ***R*** (*reset*); nel caso del circuito con porte NAND, gli ingressi ***S'*** e ***R'*** funzionano in modo negato.

Figura u101.5. Circuito elementare di tipo *set-reset* con porte NOR.



S	R	Q
0	0	*invariato*
0	_/‾	0
0	1	0
0	‾_	0
_/‾	0	1
_/‾	_/‾	0
_/‾	1	0
_/‾	‾_	1
1	0	1
1	_/‾	0
1	1	0
1	‾_	1
‾_	0	1
‾_	_/‾	0
‾_	1	0
‾_	‾_	*non ammissibile!*

Figura u101.6. Circuito elementare di tipo *set-reset* con porte NAND.

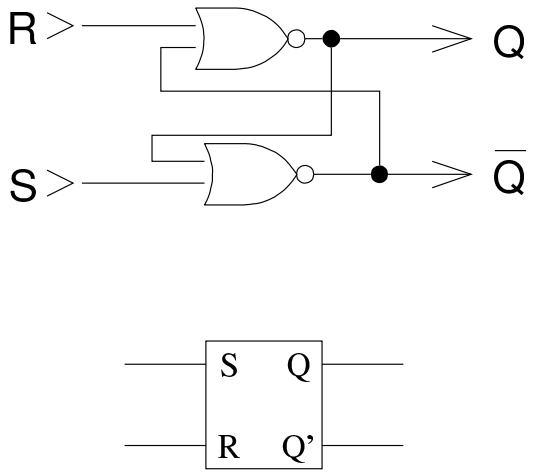


S'	R'	Q
0	0	1
0	_/‾	1
0	1	1
0	‾_	1
_/‾	0	0
_/‾	_/‾	*non ammissibile!*
_/‾	1	1
_/‾	‾_	0
1	0	0
1	_/‾	0
1	1	*invariato*
1	‾_	0
‾_	0	1
‾_	_/‾	1
‾_	1	1
‾_	‾_	1

Nelle tabelle di verità che appaiono a fianco dei disegni, si può osservare che esiste una situazione inammissibile: nel primo caso quando entrambi gli ingressi passano da uno a zero; nel secondo, quando passano da zero a uno. Se si tentasse di farlo, si metterebbe il circuito in risonanza, come viene spiegato meglio in seguito.

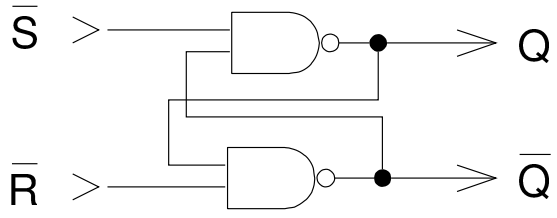
Le ultime due figure apparse rappresentano in realtà i flip-flop SR elementari, i quali però si disegnano solitamente aggiungendo un'uscita ulteriore, ***Q'***, che assume un valore inverso rispetto a ***Q***.

Figura u101.7. Flip-flop SR elementare, realizzato con porte NOR.



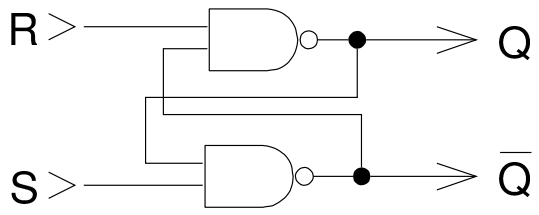
S	R	Q	Q'
0	0	*invariato*	
0	_/¯	0	1
0	1	0	1
0	¯_	0	1
_/¯	0	1	0
_/¯	_/¯	0	1
_/¯	1	0	1
_/¯	¯_	1	0
1	0	1	0
1	_/¯	0	1
1	1	0	0
1	¯_	1	0
¯_	0	1	0
¯_	_/¯	0	1
¯_	1	0	1
¯_	¯_	*non ammissibile!*	

Figura u101.8. Flip-flop SR elementare, realizzato con porte NAND.



S'	R'	Q	Q'
0	0	1	1
0	_/¯	1	0
0	1	1	0
0	¯_	1	0
_/¯	0	0	1
_/¯	_/¯	*non ammissibile!*	
_/¯	1	1	0
_/¯	¯_	0	1
1	0	0	1
1	_/¯	0	1
1	1	*invariato*	
1	¯_	0	1
¯_	0	1	0
¯_	_/¯	1	0
¯_	1	1	0
¯_	¯_	1	0

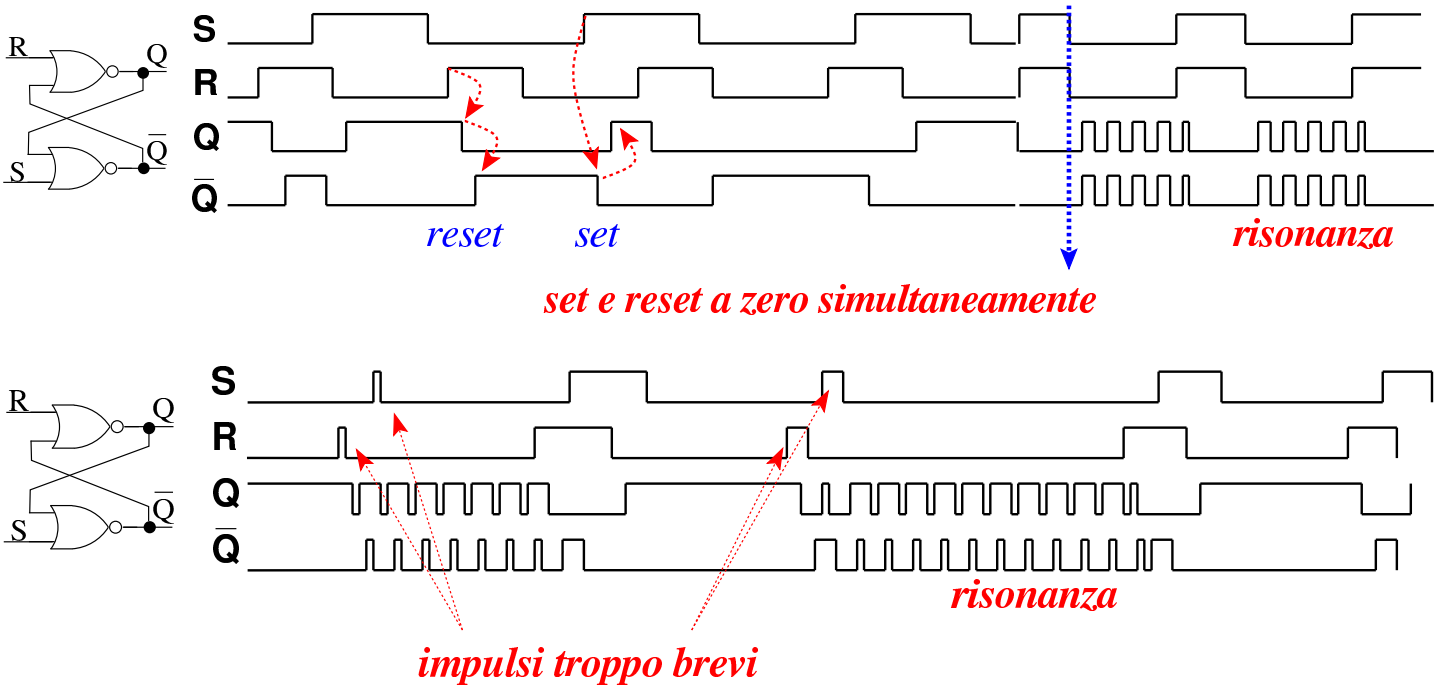
Figura u101.9. Flip-flop SR elementare, realizzato con porte NAND cambiando il significato degli ingressi.



S	R	Q	Q'
0	0	1	1
0	_/‾	0	1
0	1	0	1
0	‾_	0	1
_/‾	0	1	0
_/‾	_/‾	*non ammissibile!*	
_/‾	1	0	1
_/‾	‾_	1	0
1	0	1	0
1	_/‾	1	0
1	1	*invariato*	
1	‾_	1	0
‾_	0	1	0
‾_	_/‾	0	1
‾_	1	0	1
‾_	‾_	1	0

Il flip-flop SR elementare realizzato con porte NAND è equivalente a quello realizzato con porte NOR, se al primo si invertono gli ingressi; tuttavia, il flip-flop SR con porte NAND si usa spesso cambiando nome agli ingressi, come avviene nell'ultima figura mostrata, ma in tal caso, il comportamento non risulta uguale a quello fatto con porte NOR, anche se vi si avvicina. Per questo problema, quando si disegna un flip-flop SR elementare come scatola, bisogna chiarire a quale tabella di verità si sta facendo riferimento; tuttavia, l'uso di flip-flop SR elementari è molto limitato, pertanto nel disegno dei circuiti è meglio evitare di rappresentarli come scatole.

Figura u101.10. Tracciati di un flip-flop SR elementare basato su porte NOR, evidenziando la situazione critica che si crea quando entrambi gli ingressi si trovano attivati assieme e poi si azzerano simultaneamente e quella che si crea quando l'attivazione degli ingressi ha una durata troppo breve.

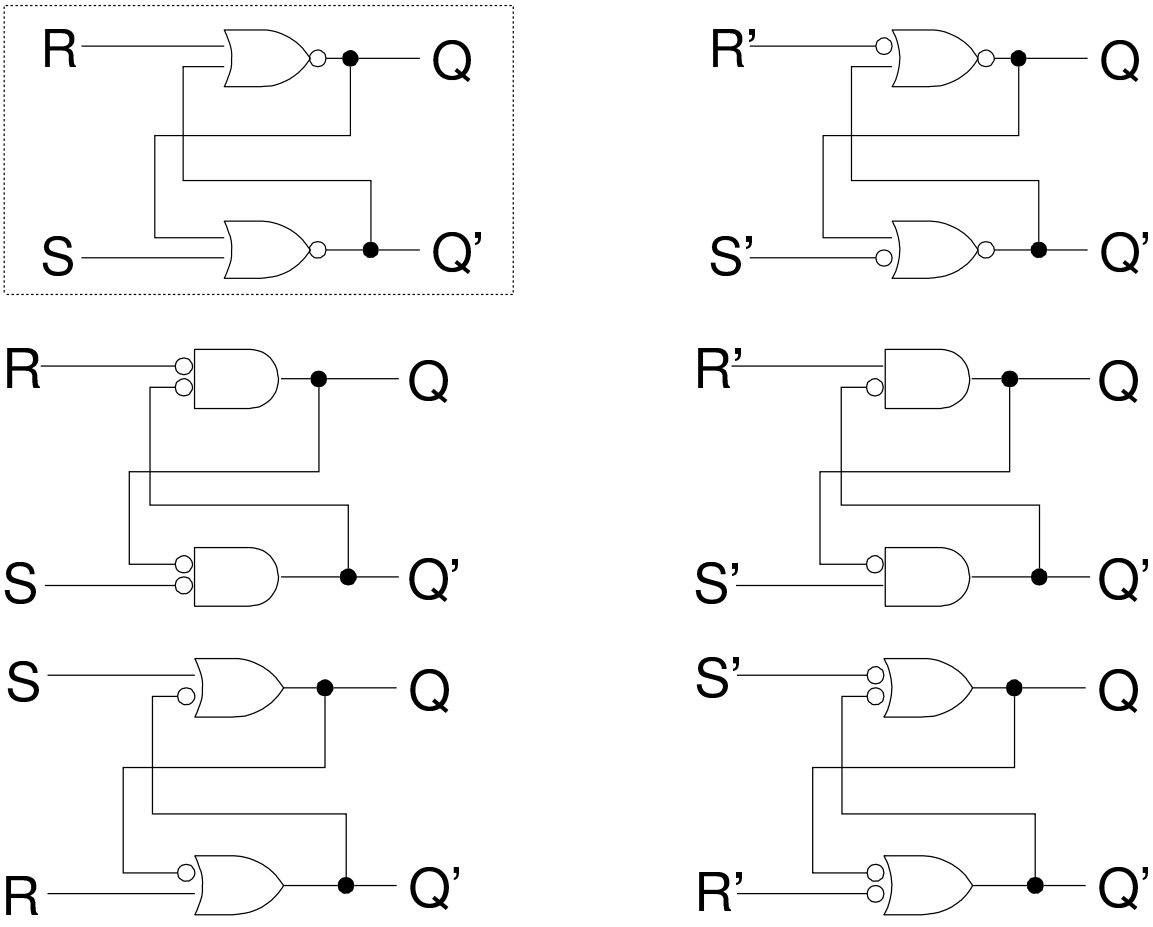


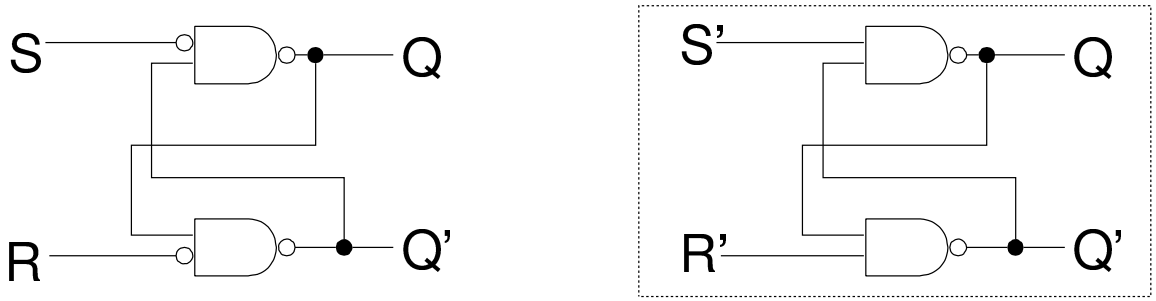
La figura mostra il tracciato dei valori delle entrate e quello delle uscite di un flip-flop SR elementare e con questo si può vedere l'effetto del ritardo di propagazione, ma soprattutto ciò che accade quando gli ingressi da attivi passano simultaneamente a zero: a causa del ritardo di propagazione, le due uscite rimangono per un po' a zero, poi, però, trovando gli ingressi a zero, si attivano simultaneamente e innescano un circolo vizioso. Pertanto, nel flip-flop SR elementare, è necessario evitare la condizione inammissibile già mostrata nelle tabelle di verità. Ma anche un impulso troppo breve in uno degli ingressi può procurare un effetto di risonanza: in generale, la durata minima di un impulso negli ingressi deve essere tale

da consentire al flip-flop di cambiare stato, quando tale impulso lo prevederebbe.

Nonostante la sua limitazione, a proposito della presenza di condizioni di ingresso inammissibili, il flip-flop SR elementare costituisce la base per tutti gli altri tipi di flip-flop. Per tale motivo è importante conoscere in quali altre forme può essere realizzato, come dimostrato nella figura successiva.

Figura u101.11. Flip-flop SR elementare in varie forme alternative, nelle quali occorre fare attenzione all'ordine degli ingressi. Nella colonna di sinistra si mostrano nella versione a ingressi positivi, mentre in quella destra appaiono nella versione a ingressi invertiti. Nei due gruppi sono evidenziati i due tipi più comuni: NOR e NAND.



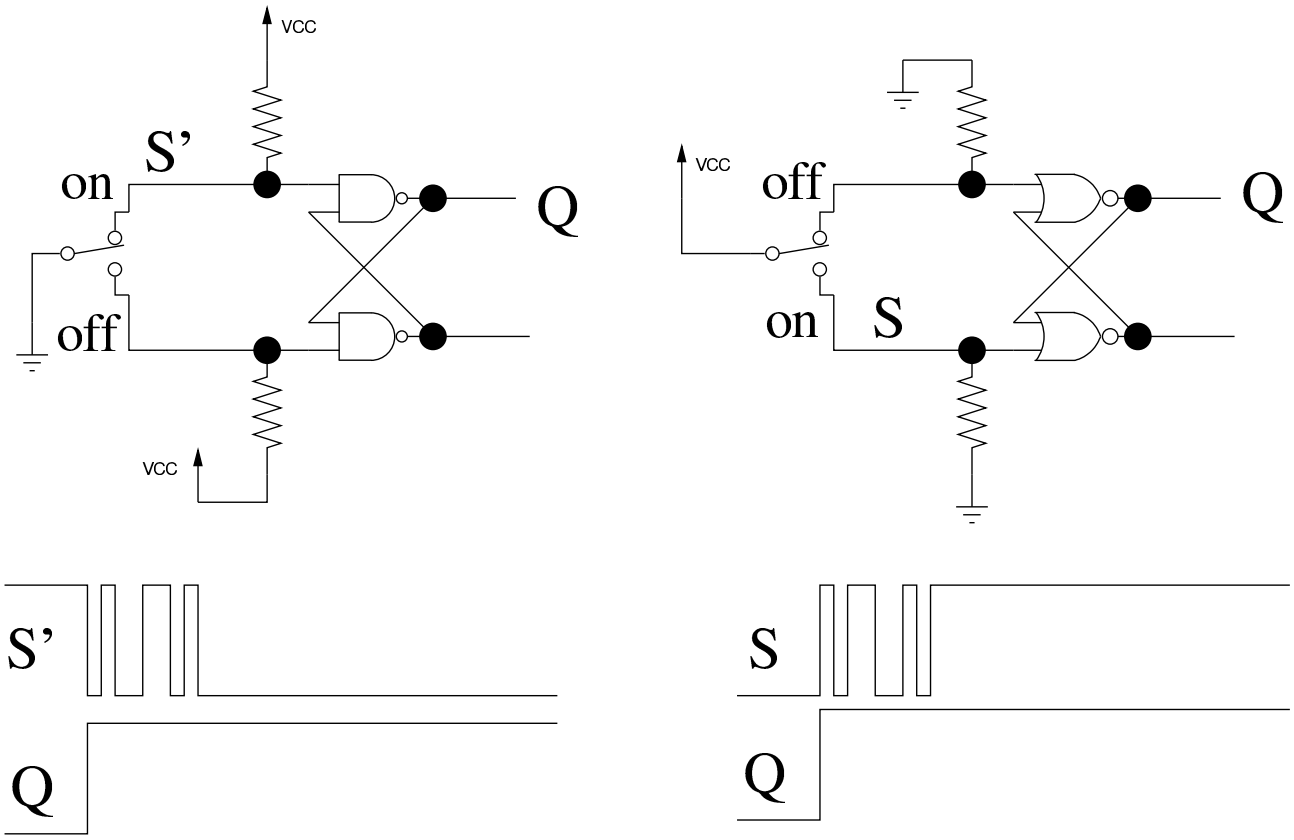


Interruttori senza rimbalzi

«

Nella realizzazione pratica di circuiti logici (elettronici) si ha spesso la necessità di utilizzare degli interruttori o pulsanti. Ma nella vita reale, tali componenti hanno il problema dei rimbalzi, nel senso che l'apertura o la chiusura di un interruttore comporta la creazione di impulsi indesiderabili. Per impedire che tali impulsi arrivino a un circuito, si utilizzano i flip-flop SR elementari, per esempio nella modalità che si può vedere nella figura successiva.

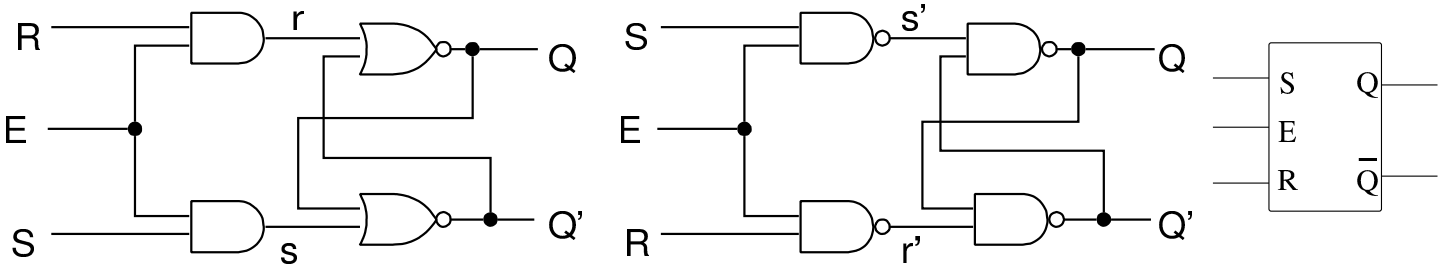
Figura u101.12. Utilizzo di un flip-flop SR elementare per filtrare i rimbalzi di un interruttore.



Flip-flop SR con gli ingressi controllati

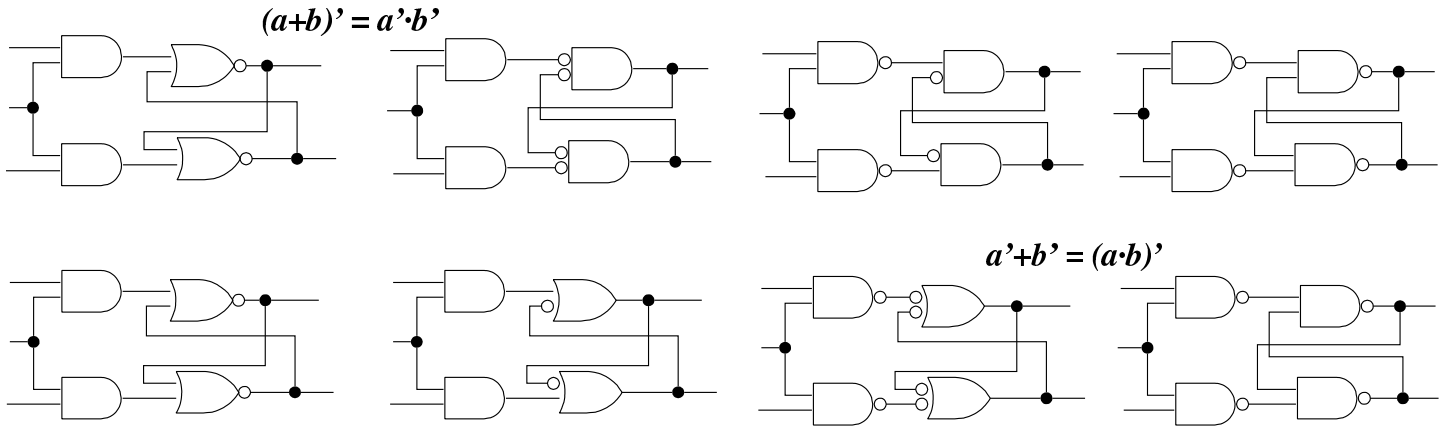
Il flip-flop SR elementare può essere esteso aggiungendo un controllo di abilitazione degli ingressi, con due porte AND o NAND, a seconda del tipo: in pratica, così facendo, quando l'ingresso di abilitazione è attivo, il flip-flop SR funziona normalmente, mentre diversamente è come se entrambi gli ingressi si trovassero a zero.

Figura u101.13. Flip-flop SR controllato: quando l'ingresso ***E*** (*enable*) è attivo, funziona come un flip-flop SR elementare; se invece l'ingresso ***E*** non è attivo, è come se gli ingressi ***S*** e ***R*** fossero disattivati a loro volta. Il circuito appare nelle due versioni più comuni, assieme alla simbologia usata normalmente per rappresentare questo tipo di flip-flop.



I due schemi che appaiono nella figura sono equivalenti e lo si dimostra facilmente, con l'aiuto dei teoremi di De Morgan, come si vede nei disegni seguenti.

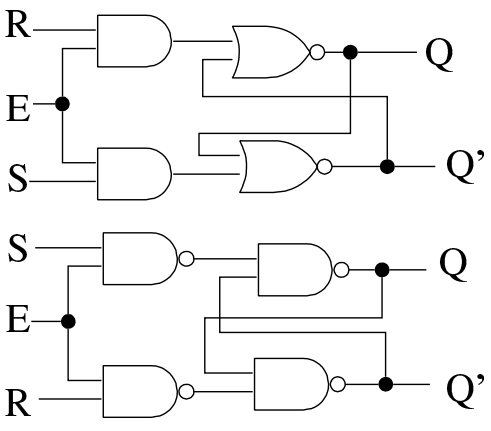
Figura u101.14. Dimostrazione dell'equivalenza dei due modi di rappresentare i flip-flop SR controllati.



Nel flip-flop SR controllato rimangono i problemi di innesco della risonanza già descritti: in pratica, è indispensabile che l'ingresso di abilitazione (***E***) sia attivo solo quando gli ingressi ***S*** e ***R*** si trovano in una condizione valida, ma rispetto al flip-flop SR elementare,

non è nemmeno ammesso che gli ingressi siano attivi simultaneamente, perché all'abbassarsi della linea di abilitazione si creerebbe inevitabilmente l'innesco.

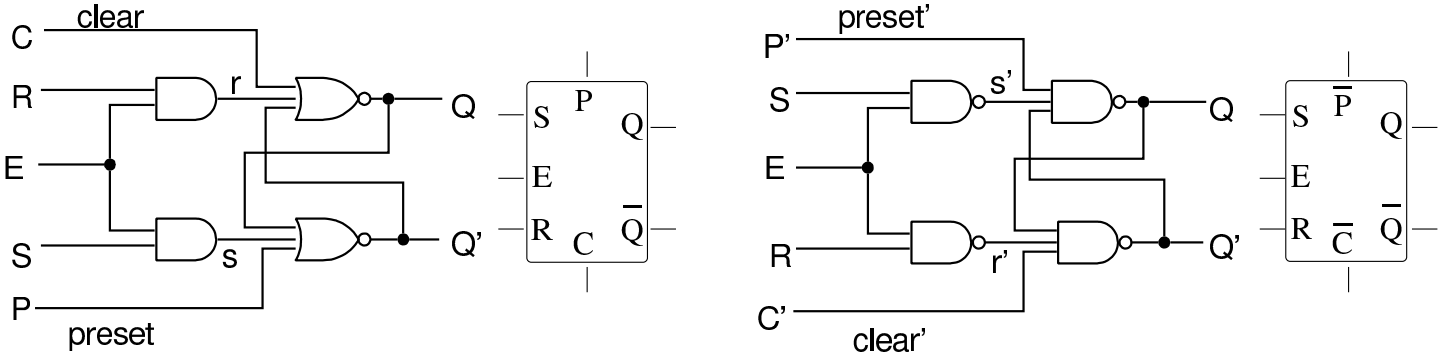
Figura u101.15. Tabella di verità per il flip-flop SR controllato, limitatamente ai casi più significativi.



E	S	R	Q	Q'
0	**x**	**x**	*invariato*	
_/‾‾_	**0**	**0**	*invariato*	
_/‾‾_	**0**	**1**	**0**	**1**
_/‾‾_	**1**	**0**	**1**	**0**
_/‾‾_	**1**	**1**	*non ammissibile!*	
⋮	⋮	⋮		
1	‾_	‾_	*non ammissibile!*	
‾_	**1**	**1**	*non ammissibile!*	

Quando si mette in funzione un flip-flop, lo stato delle uscite è indeterminabile. Per poter inizializzare il flip-flop SR controllato, è necessario estendere gli ingressi delle porte del flip-flop SR elementare, come mostrato nella figura successiva. Va osservato che a seconda di come si realizza il flip-flop SR, può darsi che per inizializzare il flip-flop possa richiedersi un valore a uno o a zero.

Figura u101.16. Flip-flop SR controllato, con ingressi di inizializzazione: quando l'ingresso ***P*** (*preset*) è attivo, si forza l'attivazione dell'uscita ***Q***; quando l'ingresso ***C*** (*clear*) è attivo, si forza l'attivazione dell'uscita ***Q'***. Il circuito viene mostrato nelle due varianti realizzative comuni, assieme alla rappresentazione simbolica complessiva. Va osservato che nella seconda modalità, gli ingressi ***P*** e ***C*** sono invertiti (negati).



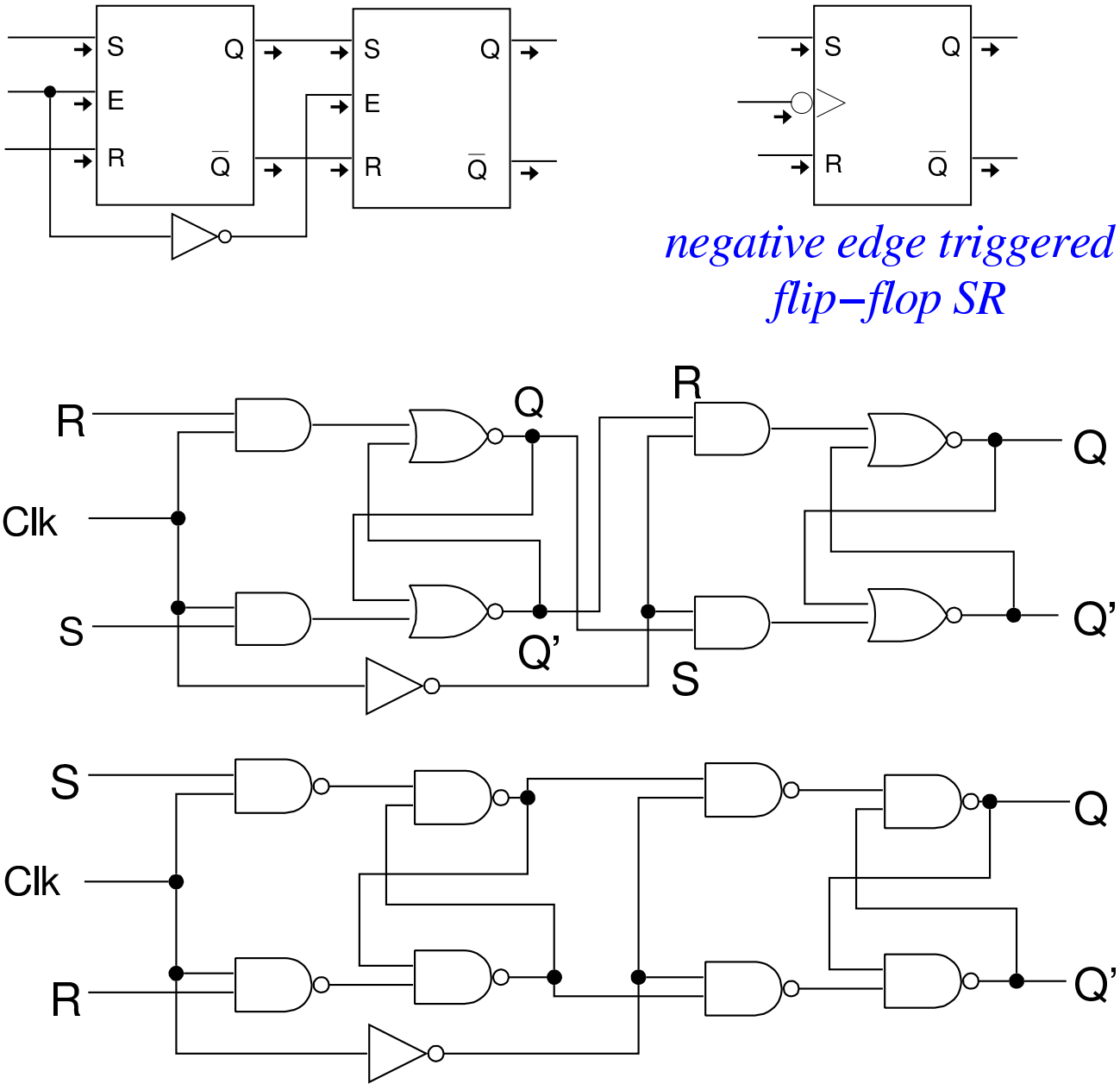
Flip-flop SR sincrono «edge triggered»

«

Nel flip-flop controllato attraverso l'ingresso di abilitazione (o di *clock*), quando tale ingresso di abilitazione è attivo, i valori degli altri ingressi possono cambiare e il loro cambiamento può trasmettersi regolarmente nel flip-flop modificando eventualmente lo stato delle uscite. Per fare in modo che il controllo di abilitazione fotografi la situazione degli ingressi, occorre che sia pilotato da un impulso abbastanza breve, ma non troppo, tale per cui in quel lasso di tempo i valori degli ingressi non possano cambiare. Per ovviare a questo problema, si possono mettere due flip-flop SR controllati in cascata (*master-slave*), dove il secondo riceve il segnale di abilitazione invertito rispetto al primo; tuttavia, in tal caso l'aggiornamento del flip-flop complessivo si ottiene nel momento in cui il segnale di abilitazione si disattiva, ovvero in corrispondenza del margine negativo

(*negative edge*).

Figura u101.17. Flip-flop SR sincrono a margine negativo: quando il segnale di abilitazione passa da attivo a zero, il flip-flop si aggiorna. Il circuito viene mostrato a blocchi e nelle due varianti realizzative comuni, assieme alla rappresentazione simbolica complessiva, dove va osservato che l'ingresso di abilitazione viene annotato con un triangolo, per sottolineare il fatto che il segnale viene recepito in corrispondenza della sua variazione.



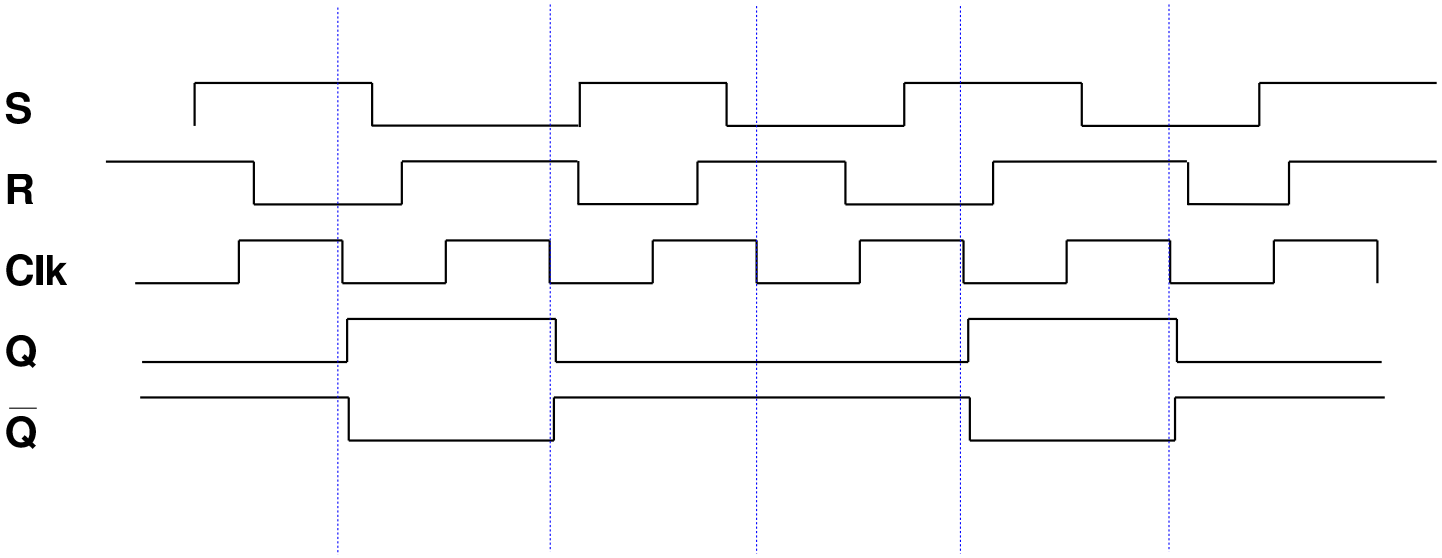


Figura u101.18. Completamento del flip-flop SR sincrono a margine negativo, con gli ingressi di inizializzazione.

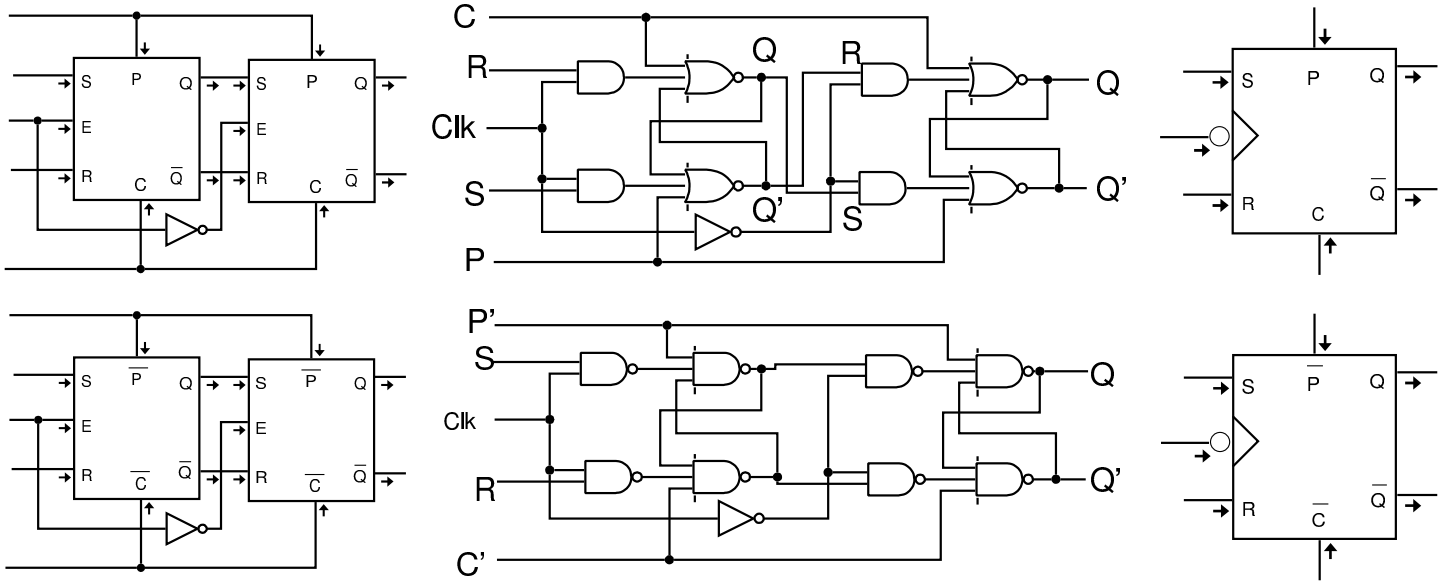
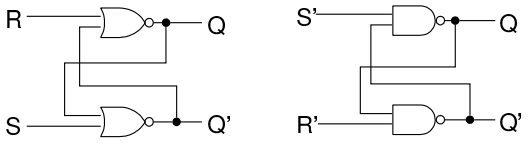
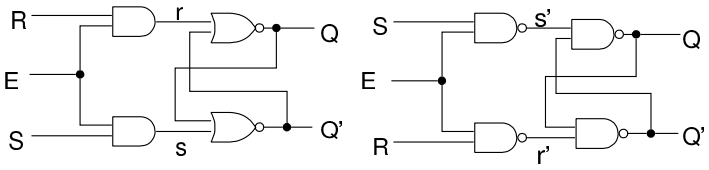


Figura u101.19. Riassunto delle varie tipologie di flip-flop SR.

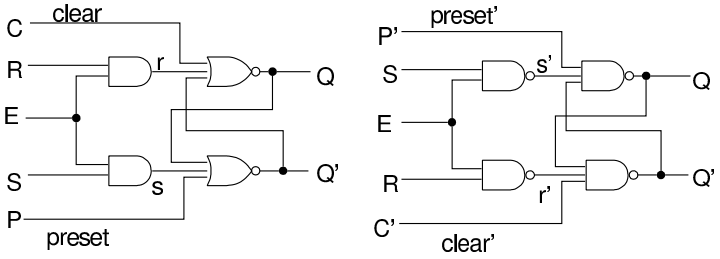
flip–flop SR elementare



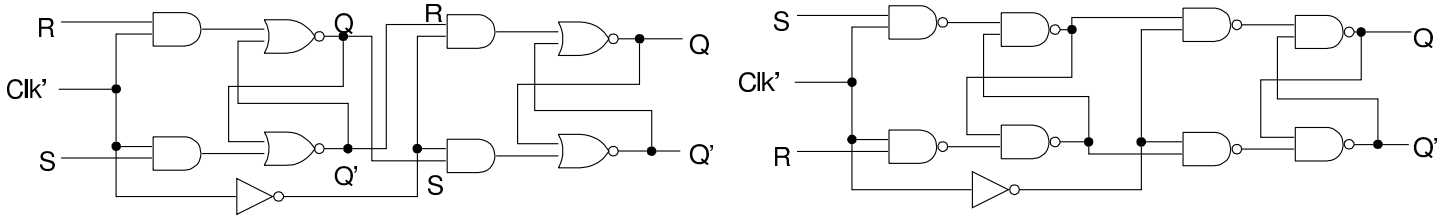
flip–flop SR controllato



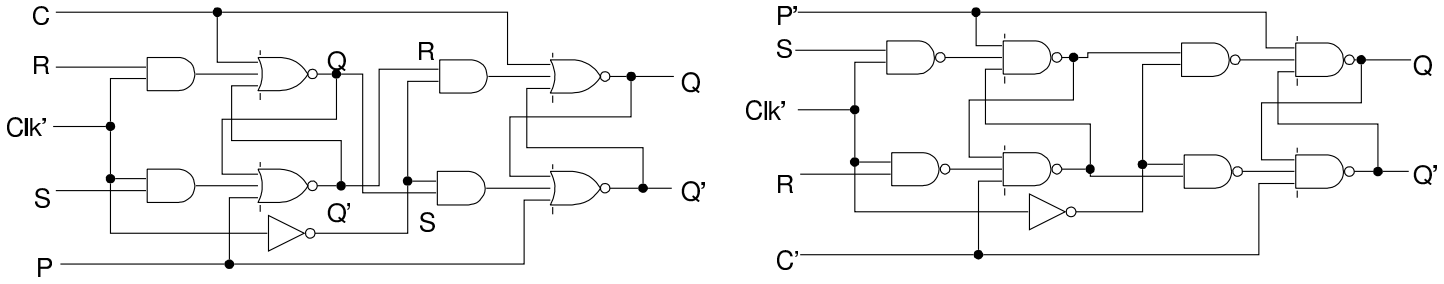
flip–flop SR controllato inizializzabile



flip–flop SR in cascata (negative edge triggered)



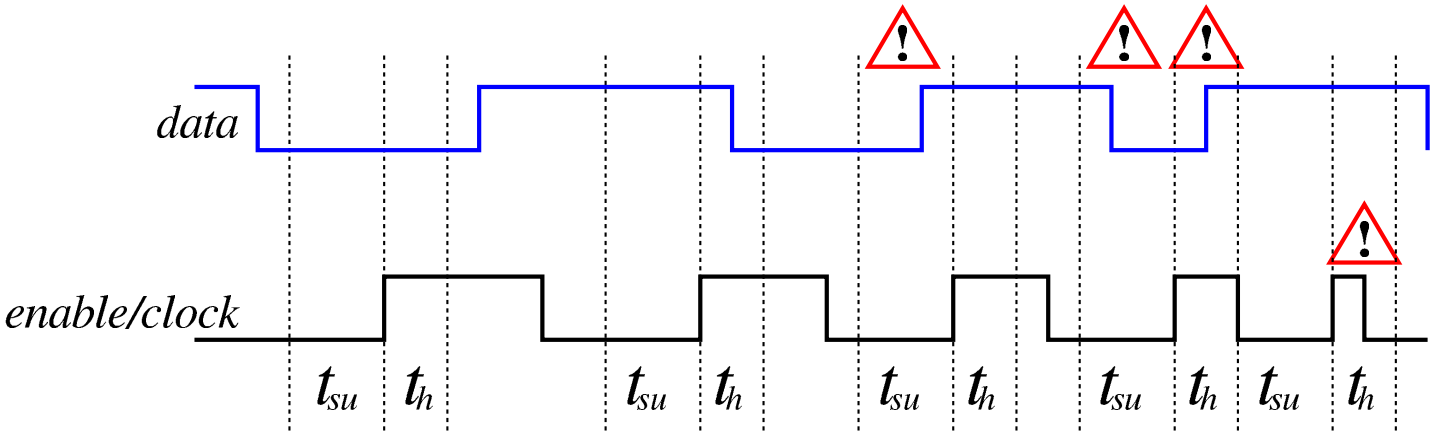
flip–flop SR in cascata (negative edge triggered) inizializzabile



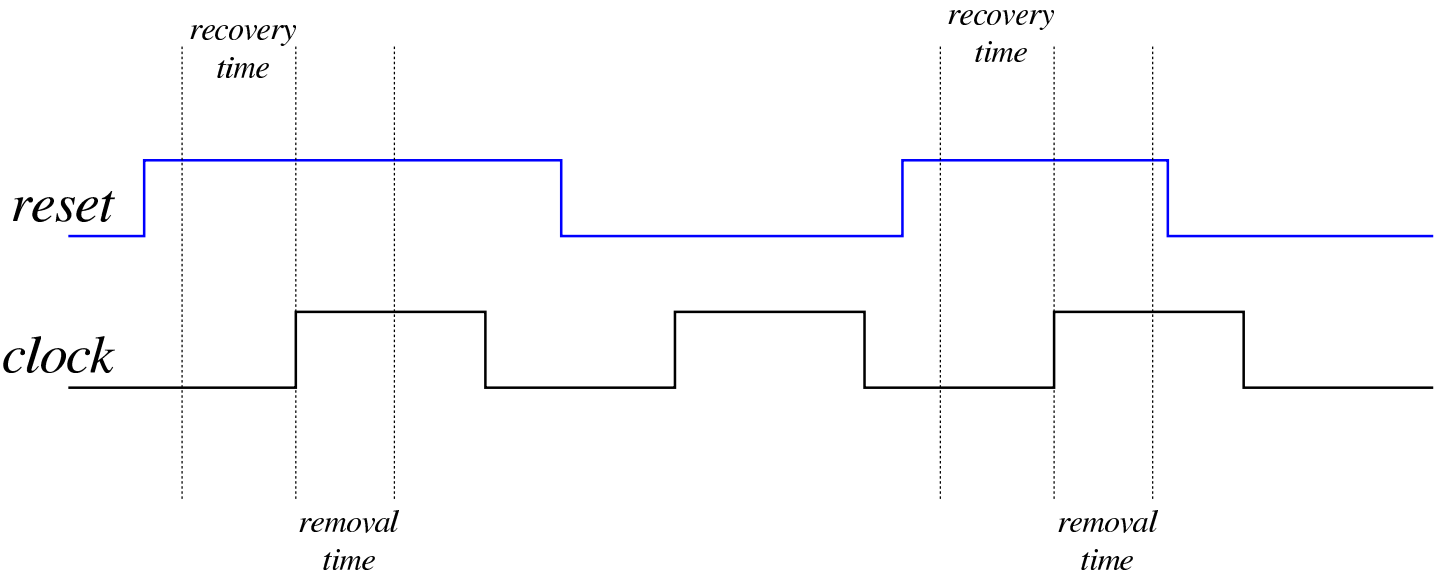
Tempo: *setup/hold* e *recovery/removal*

Si distinguono due intervalli di tempo significativi per i componenti sincroni, ovvero quelli che hanno un ingresso dati controllato da un ingresso di abilitazione o di *clock*. Si tratta del tempo di attivazione, *setup time*, noto con la sigla $\boldsymbol{t}_{su}$, e del tempo di mantenimento, *hold time*, noto con la sigla $\boldsymbol{t}_{h}$.

Figura u101.20. Esempio di situazioni corrette e non corrette, relativamente ai vincoli del tempo di attivazione (t_{su}) e del tempo di mantenimento (t_h).



In contesti diversi, quando si vuole sottolineare il fatto che il dato in ingresso è un'informazione asincrona, si usa un'altra terminologia: *recovery time* e *removal time*.

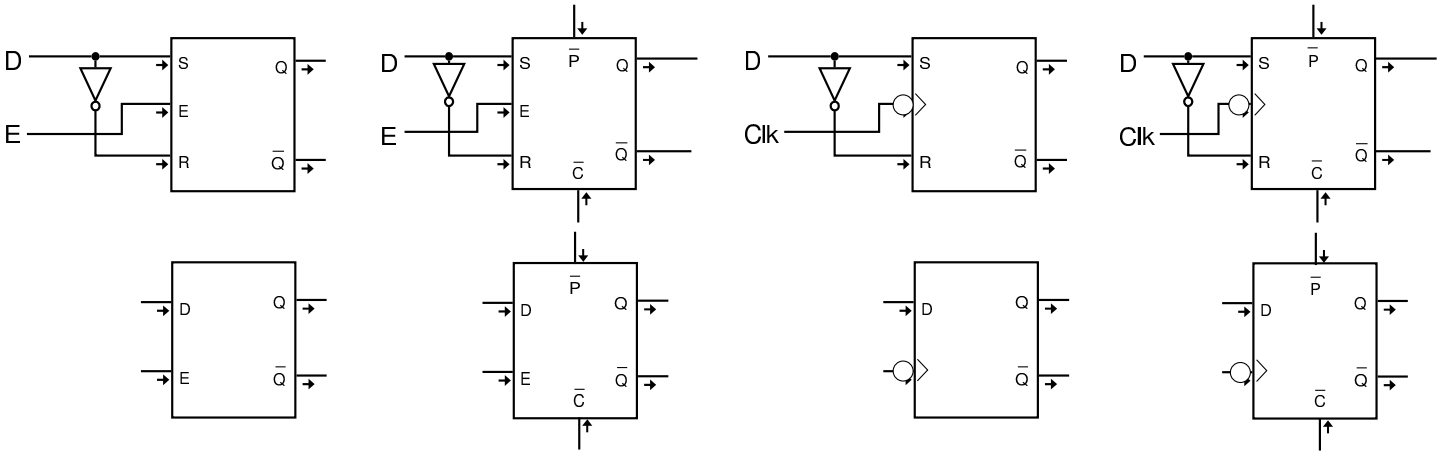


Flip-flop D

«

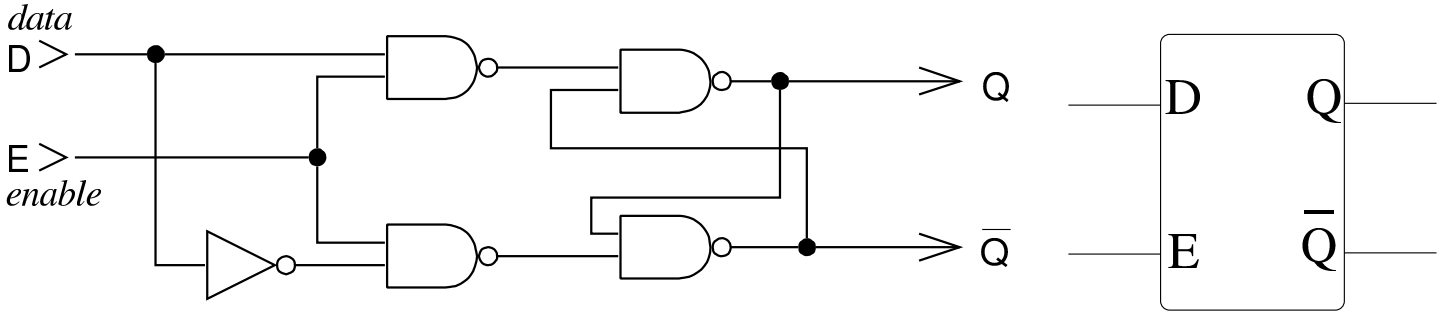
Il flip-flop D (*data*) si ottiene da un flip-flop SR, collegando assieme i due ingressi, invertendo però l'ingresso ***R***. In pratica, il flip-flop D è utile solo quando c'è almeno il controllo di abilitazione degli ingressi.

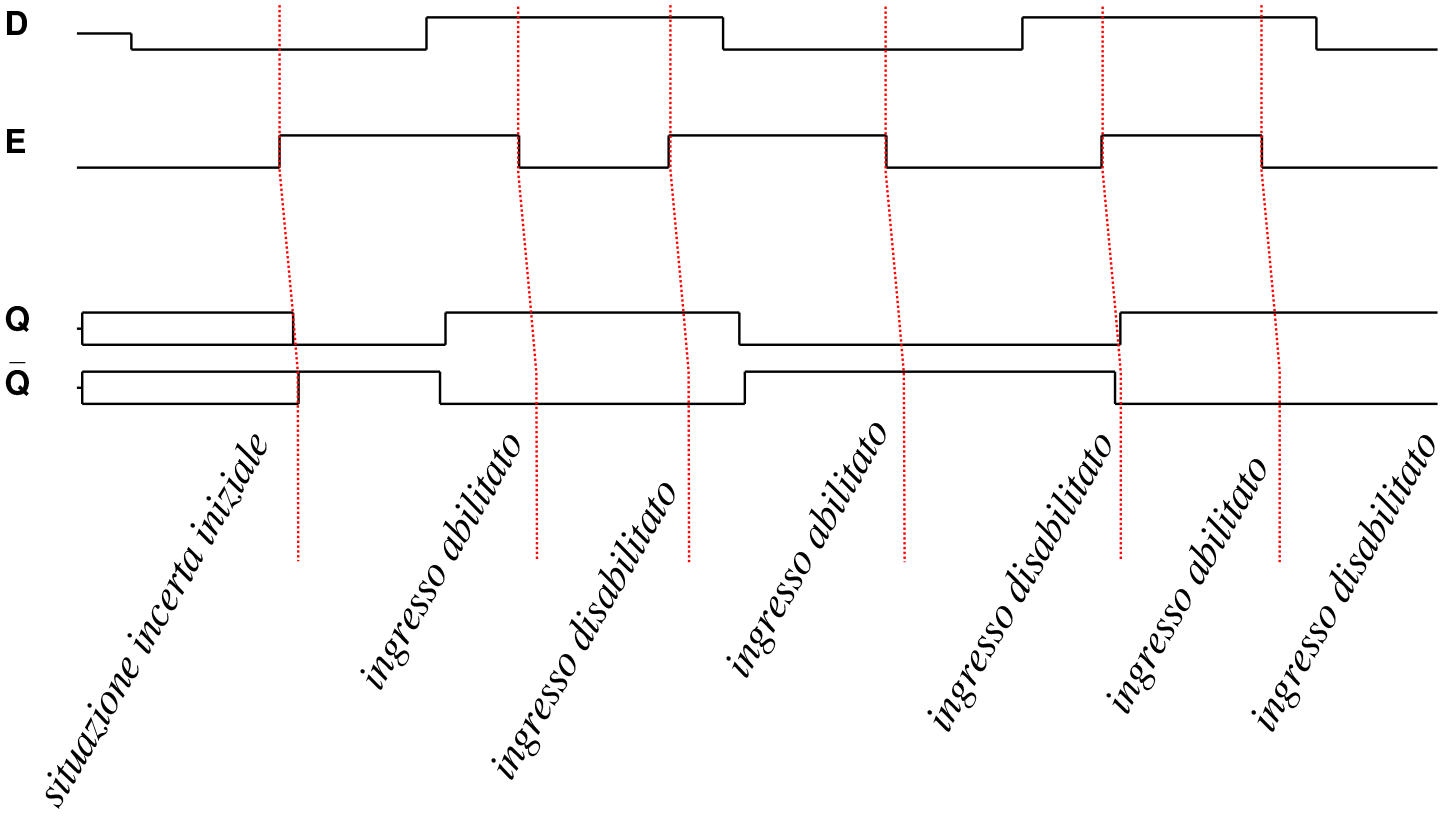
Figura u101.22. Flip-flop D ottenuto dal flip-flop SR, nelle varie tipologie ammissibili, a confronto con i simboli corrispondenti. Gli ingressi di inizializzazione e di clock sono stati mostrati solo nella versione negata, essendo la più comune.



Sul flip-flop D valgono le stesse considerazioni fatte sul flip-flop SR, per quanto riguarda il tempo di attivazione e il tempo di mantenimento dell'impulso di abilitazione. Rispetto al flip-flop SR, essendoci un solo ingresso dati, non ci sono combinazioni inammissibili.

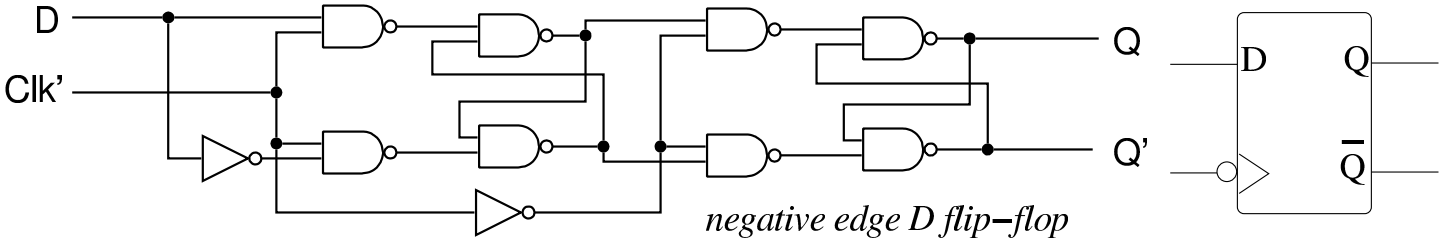
Figura u101.23. Flip-flop D con ingresso di abilitazione semplice, realizzato utilizzando porte NAND. Quando l'ingresso ***E*** è attivo, il circuito recepisce il dato dall'ingresso ***D*** e lo riproduce attraverso l'uscita ***Q*** (invertendolo nell'uscita ***Q'***).





Il flip-flop D, attivato dalla variazione del segnale di clock, può essere realizzato in cascata, come già visto per il flip-flop RS, Tuttavia esiste un circuito alternativo più efficiente: in tal caso, il margine di attivazione del flip-flop D, diventa quello positivo.

Figura u101.24. Flip-flop D in cascata e nella sua realizzazione classica. Il tracciato riguarda la seconda versione che risulta funzionare a margine positivo.



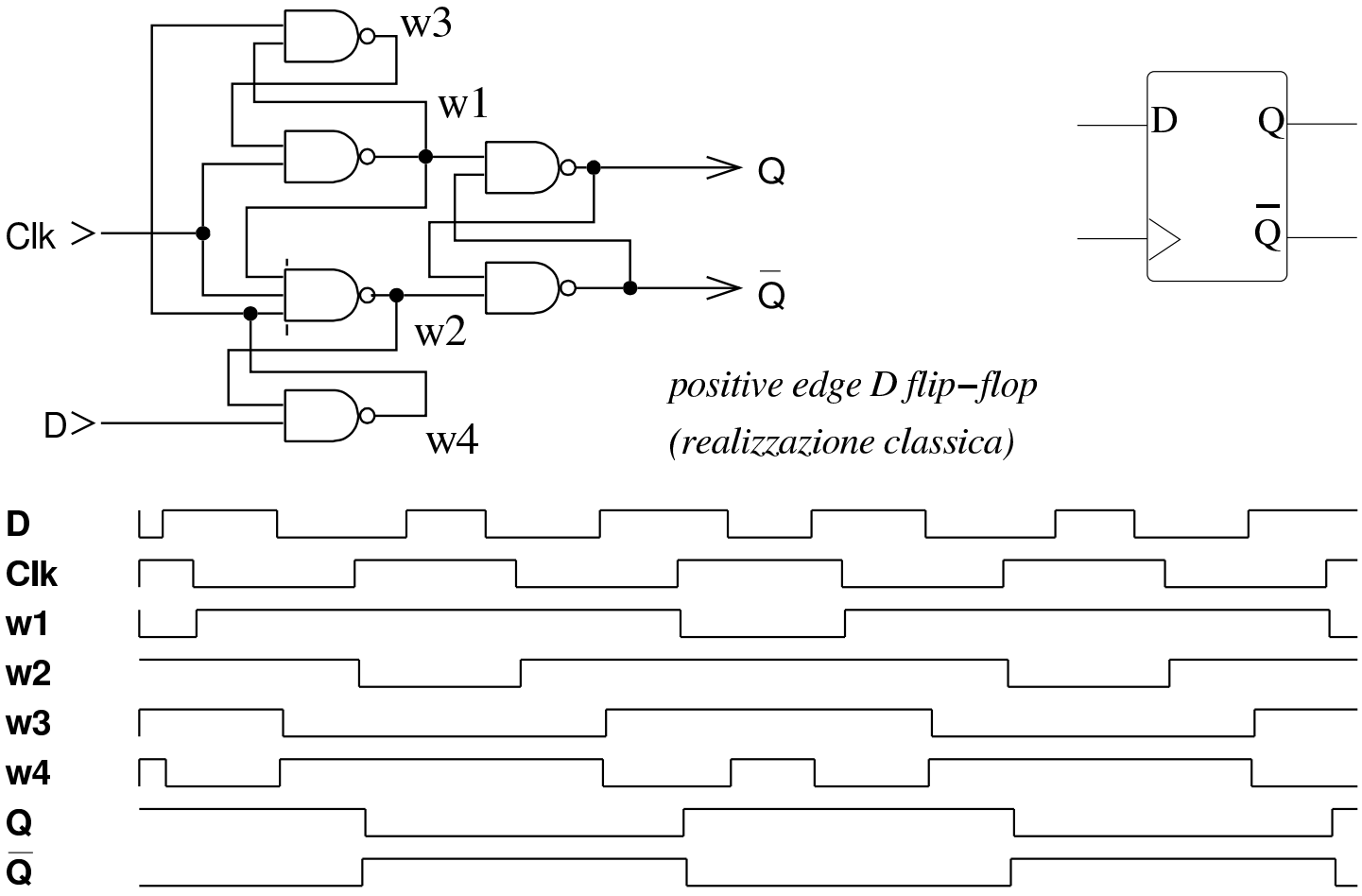
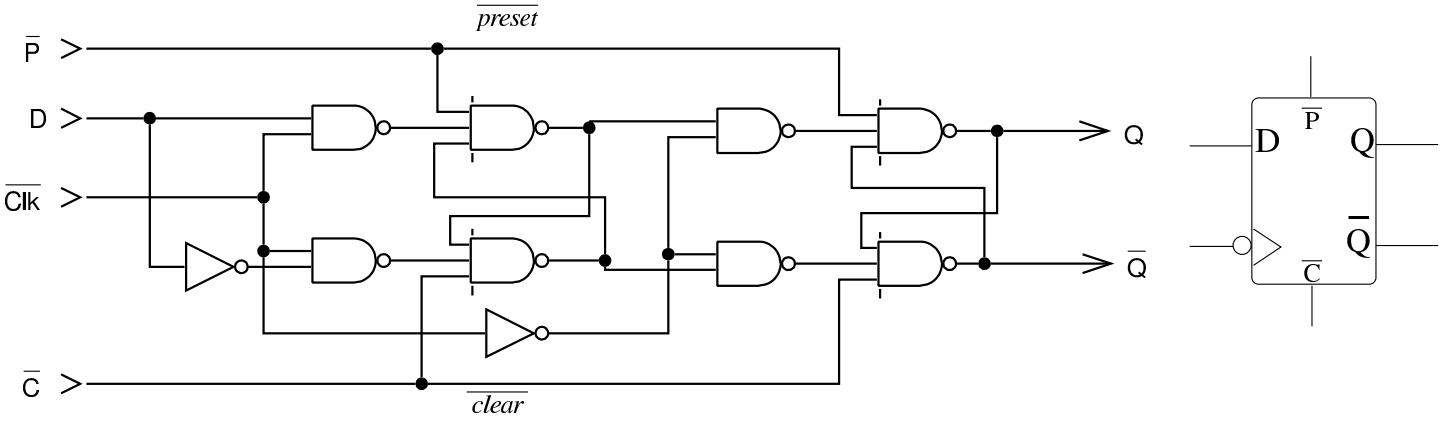
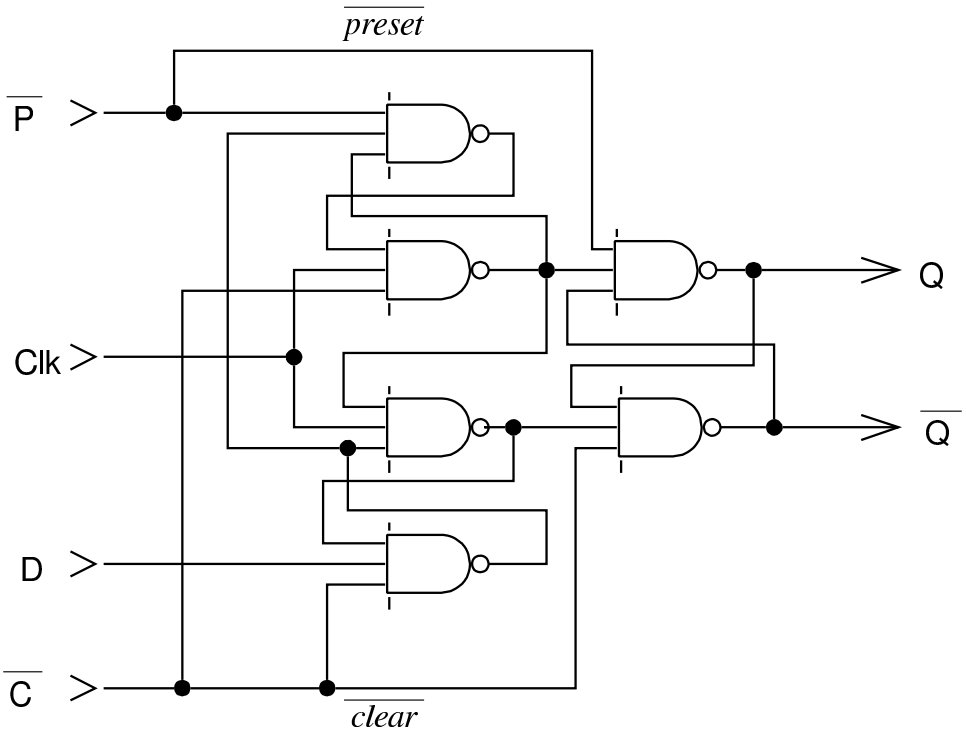
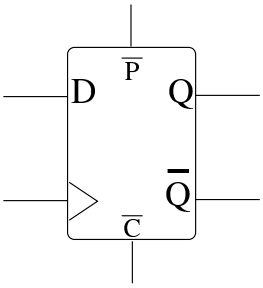


Figura u101.25. Flip-flop D in cascata e nella sua realizzazione classica, con gli ingressi di inizializzazione.





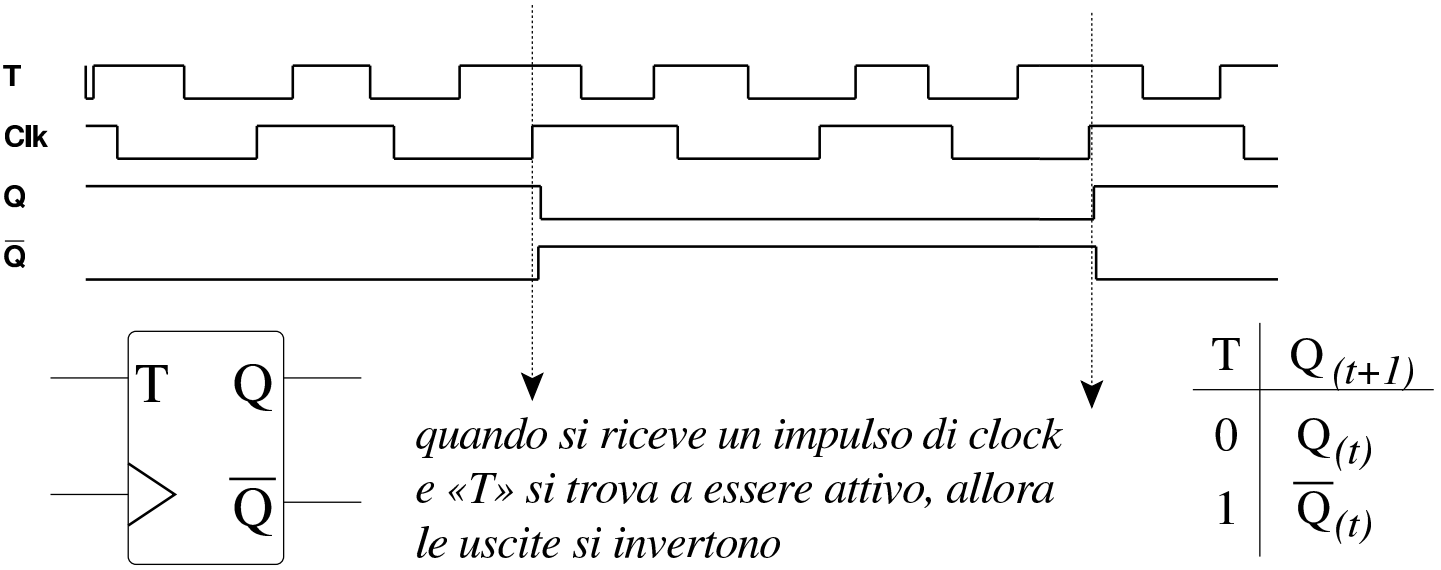


Flip-flop T

«

Estendendo leggermente un flip-flop D a margine positivo, è possibile ottenere un flip-flop T (*toggle*), il quale ha lo scopo di invertire il valore delle uscite quando l'ingresso ***T*** si attiva per la presenza di un impulso di *clock*. Nella tabella della verità si usa la notazione $\boldsymbol{Q}_{(t)}$ per indicare il valore dell'uscita ***Q*** nel momento ***t*** e la notazione $\boldsymbol{Q}_{(t+1)}$ per indicare il valore dell'uscita ***Q*** nel momento successivo ***t+1***, corrispondente all'impulso di *clock* successivo.

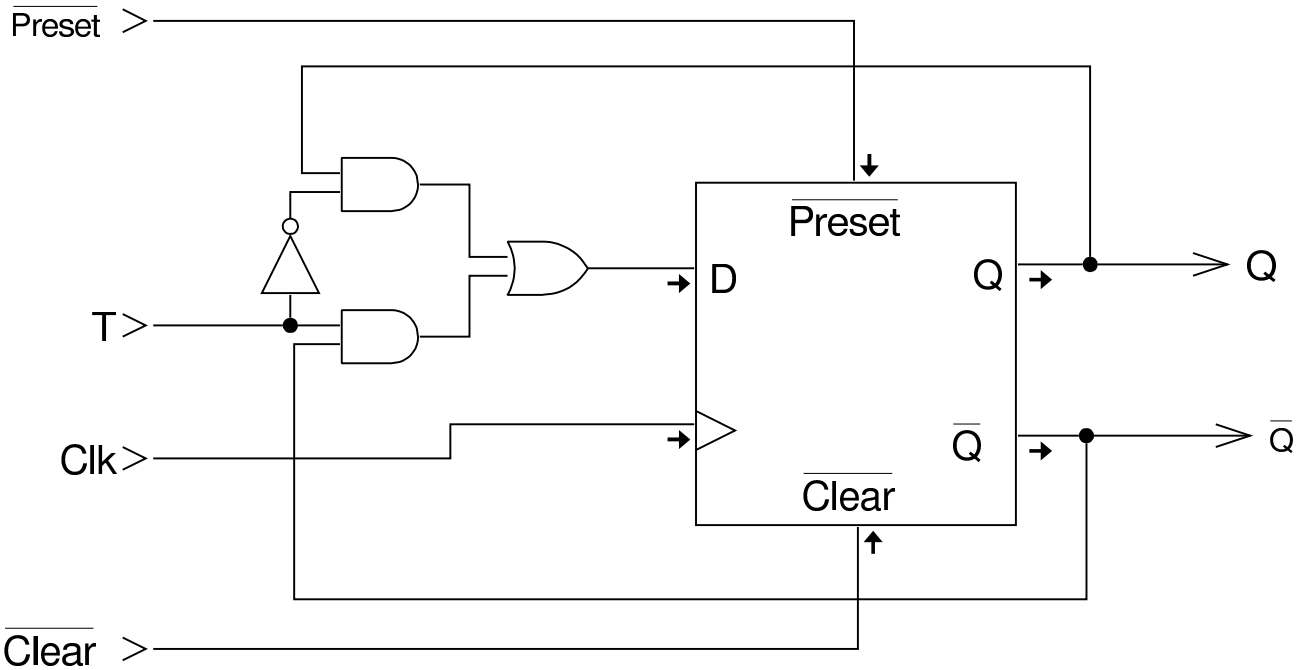
Figura u101.26. Tracciato, simbolo del flip-flop T e tabella della verità.



T	$Q_{(t+1)}$
0	$Q_{(t)}$
1	$\overline{Q}_{(t)}$

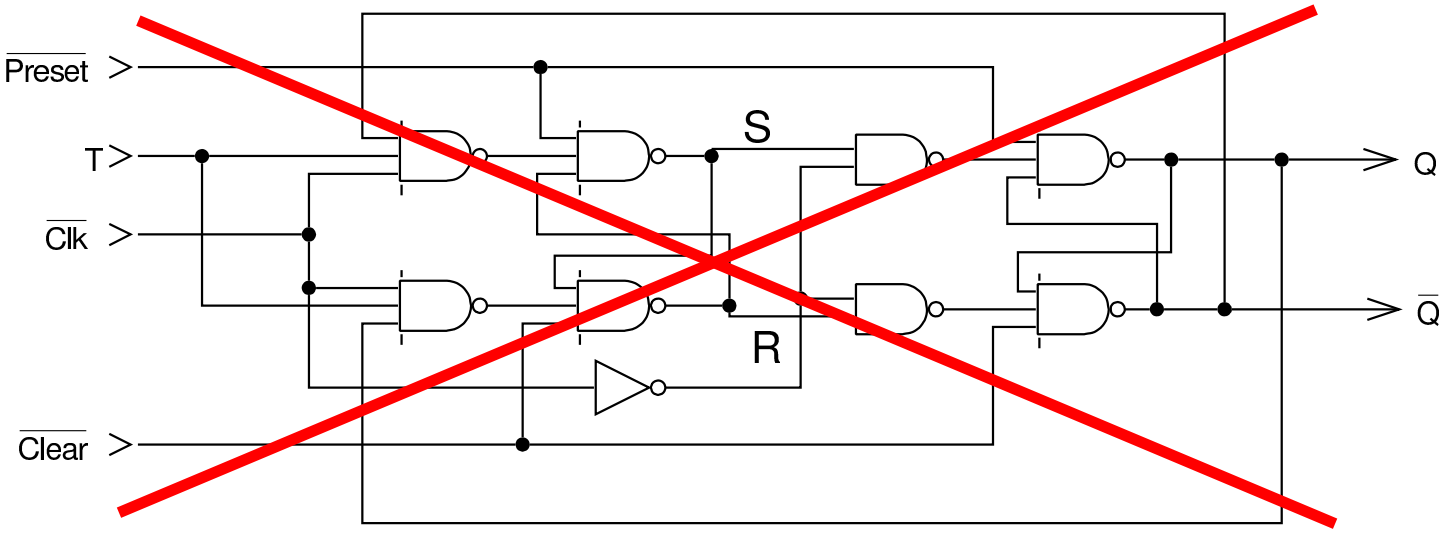
Per realizzare il flip-flop T è necessario tenere in considerazione il fatto che inizialmente non si conosce lo stato delle uscite, quindi è importante poter azzerare il flip-flop all'avvio. Dal momento che si tratta di estendere il flip-flop D, nella figura successiva si parte da quello che dispone degli ingressi di azzeramento e di impostazione.

Figura u101.27. Realizzazione del flip-flop T, partendo da un flip-flop D, munito di ingressi di azzeramento e di impostazione. Questa realizzazione è da preferire rispetto a quella della figura successiva.



La figura successiva mostra una realizzazione diversa del flip-flop T che in sostanza deriva da un flip-flop JK descritto nel libro *Digital Computer Electronics* di Malvino e Brown, nel capitolo dedicato ai flip-flop. Tuttavia, attraverso l'uso di un simulatore, si determina che se l'ingresso ***T*** riceve un impulso che termina prima che ci sia la variazione negativa del clock, si ottiene ugualmente lo scambio dei valori nelle uscite, mentre ciò non dovrebbe avvenire secondo lo schema di funzionamento previsto per un flip-flop T.

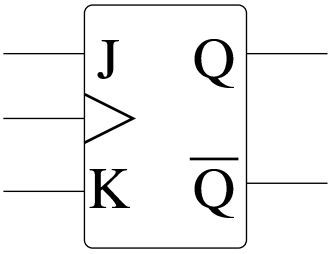
Figura u101.28. Realizzazione del flip-flop T, modificando un flip-flop SR in cascata, fatto a sua volta con porte NAND. In tal caso, lo scambio dei valori di uscita avviene in corrispondenza della variazione negativa dell'impulso di clock (*negative edge triggered*), ma bisogna fare attenzione all'ingresso ***T***, la cui attivazione viene recepita anche se si azzera prima della variazione negativa del clock.



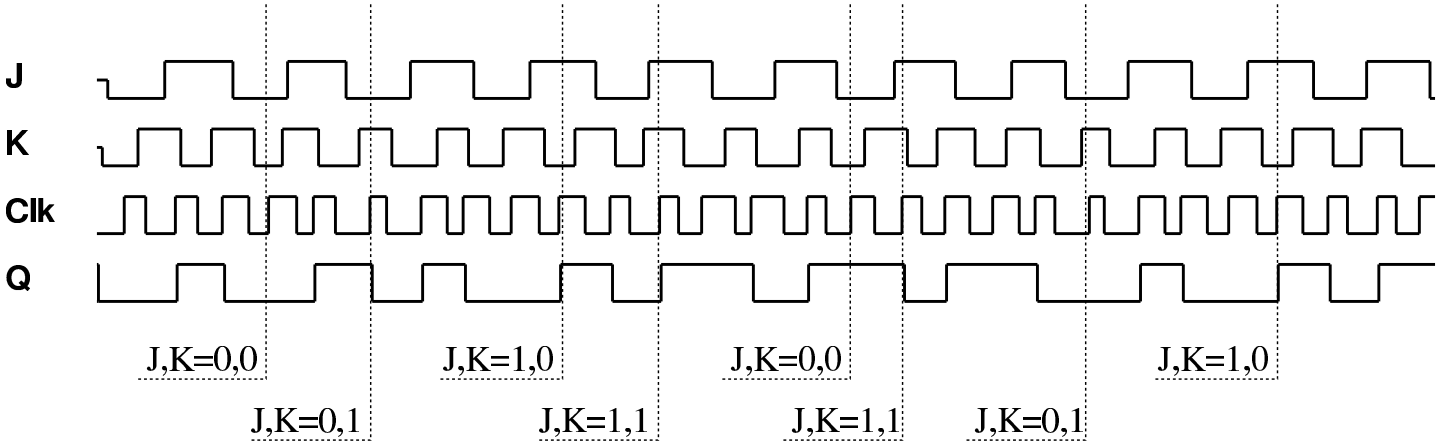
Flip-flop JK

Un'altra variante del flip-flop D, simile al flip-flop T, ma che richiama il funzionamento del flip-flop SR, è il flip-flop JK. Come per il flip-flop T, nella tabella della verità si usa la notazione $Q_{(t)}$ per indicare il valore dell'uscita Q nel momento t e la notazione $Q_{(t+1)}$ per indicare il valore dell'uscita Q nel momento successivo $t+1$, corrispondente all'impulso di *clock* successivo.

Figura u101.29. Simbolo del flip-flop JK, tabella della verità e tracciato. Si osservi che gli ingressi ***J*** e ***K*** si comportano in modo analogo a gli ingressi ***S*** e ***R*** di un flip-flop SR, con la differenza che l'attivazione di entrambi provoca solo lo scambio dei valori delle uscite, senza altre complicazioni.

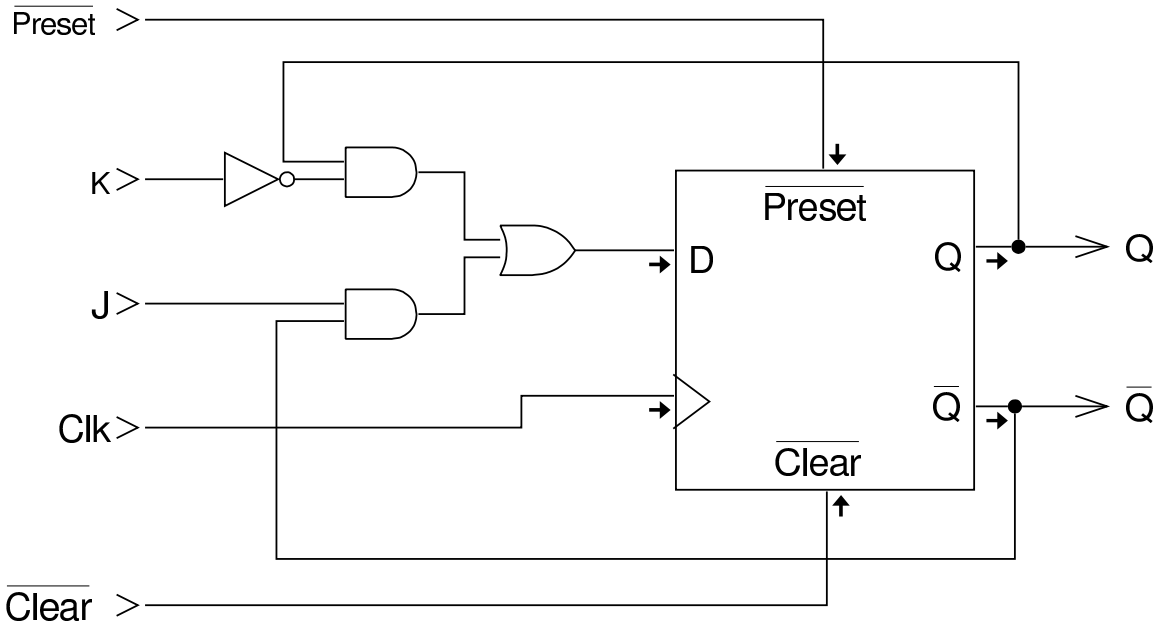


J	K	$Q_{(t+1)}$	
0	0	$Q_{(t)}$	*invariato*
0	1	0	*reset*
1	0	1	*set*
1	1	$\overline{Q}_{(t)}$	*scambio dei valori*



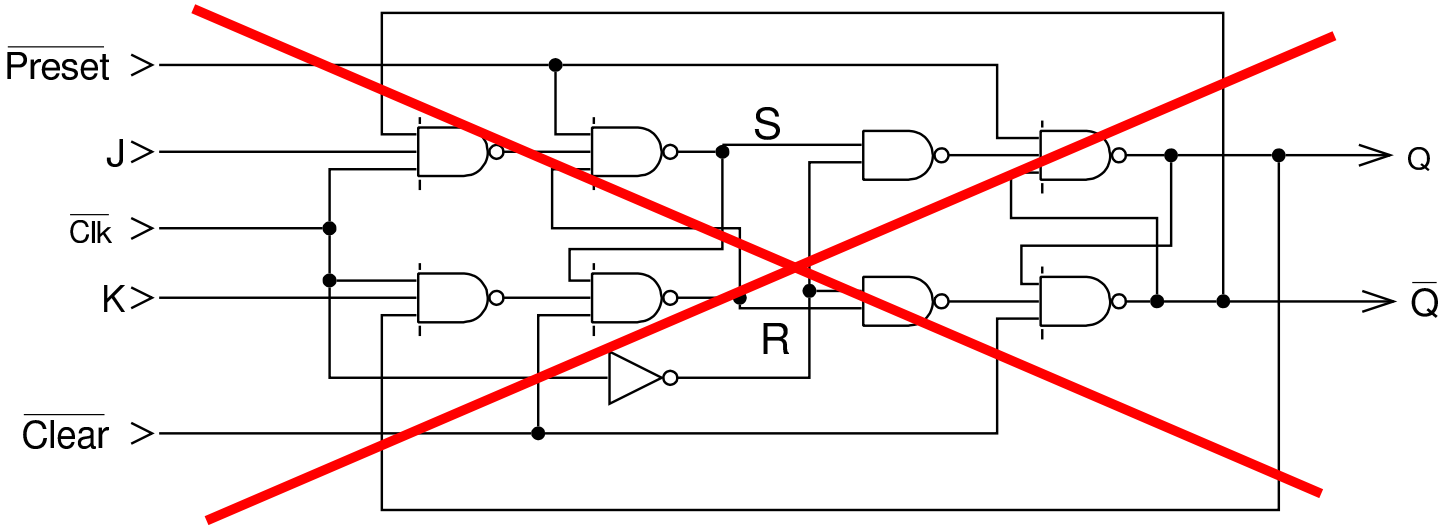
Per realizzare il flip-flop JK è necessario tenere in considerazione il fatto che inizialmente non si conosce lo stato delle uscite, quindi è importante poter azzerare il flip-flop all'avvio. Dal momento che si tratta di estendere il flip-flop D, nella figura successiva si parte da quello che dispone degli ingressi di azzeramento e di impostazione.

Figura u101.30. Realizzazione del flip-flop JK, partendo da un flip-flop D, munito di ingressi di azzeramento e di impostazione. Questa realizzazione è da preferire rispetto a quella della figura successiva.



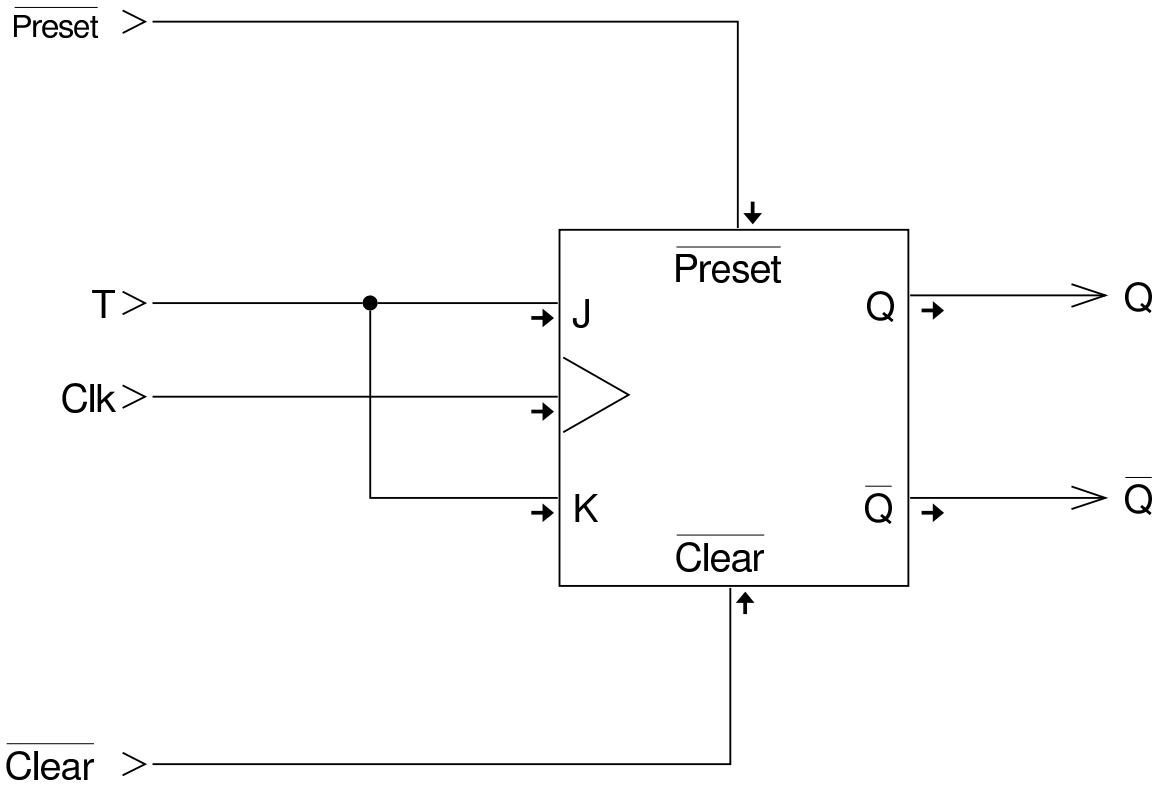
La figura successiva mostra una realizzazione diversa del flip-flop JK, tratta dal libro *Digital Computer Electronics* di Malvino e Brown, nel capitolo dedicato ai flip-flop. Tuttavia, attraverso l'uso di un simulatore, si determina che se gli ingressi ***J*** e ***K*** vengono attivati e disattivati simultaneamente, prima che ci sia la variazione negativa del clock, al momento della variazione negativa del clock si ottiene ugualmente lo scambio dei valori nelle uscite, mentre ciò non dovrebbe avvenire secondo lo schema di funzionamento previsto per un flip-flop JK.

Figura u101.31. Realizzazione del flip-flop JK, partendo da un flip-flop SR in cascata: in tal caso, l'impulso di clock ha effetto in corrispondenza del margine negativo (*negative edge triggered*). Bisogna però fare attenzione al fatto che l'attivazione e successiva disattivazione simultanea dei valori degli ingressi, prima della variazione negativa del clock, provoca lo scambio dei valori delle uscite, come se gli ingressi fossero ancora attivi in quel momento.



È evidente che disponendo di un flip-flop JK è possibile ottenere un flip-flop T, semplicemente unendo gli ingressi ***J*** e ***K*** che diventano così l'ingresso ***T***.

Figura u101.32. Adattamento di un flip-flop JK per ottenere un flip-flop T.



Registri

I ***registri*** sono delle batterie di flip-flop (di norma si tratta di flip-flop D) con le quali è possibile memorizzare valori binari o costruire dei contatori di vario tipo. Di norma i registri si realizzano con flip-flop sincroni che recepiscono il valore in ingresso al momento della variazione dell'impulso di clock (*edge triggered*).

Negli esempi delle sezioni successive si utilizzano flip-flop D, secondo la realizzazione classica, oppure flip-flop derivati da questo tipo. Pertanto, si tratta di flip-flop a margine positivo (pilotati dalla variazione positiva dell'impulso di clock).

«a2» 2013.11.11 --- Copyright © Daniele Giacomini -- appunti2@gmail.com http://informaticalibera.net

Figura u102.1. Flip-flop D secondo la realizzazione classica.

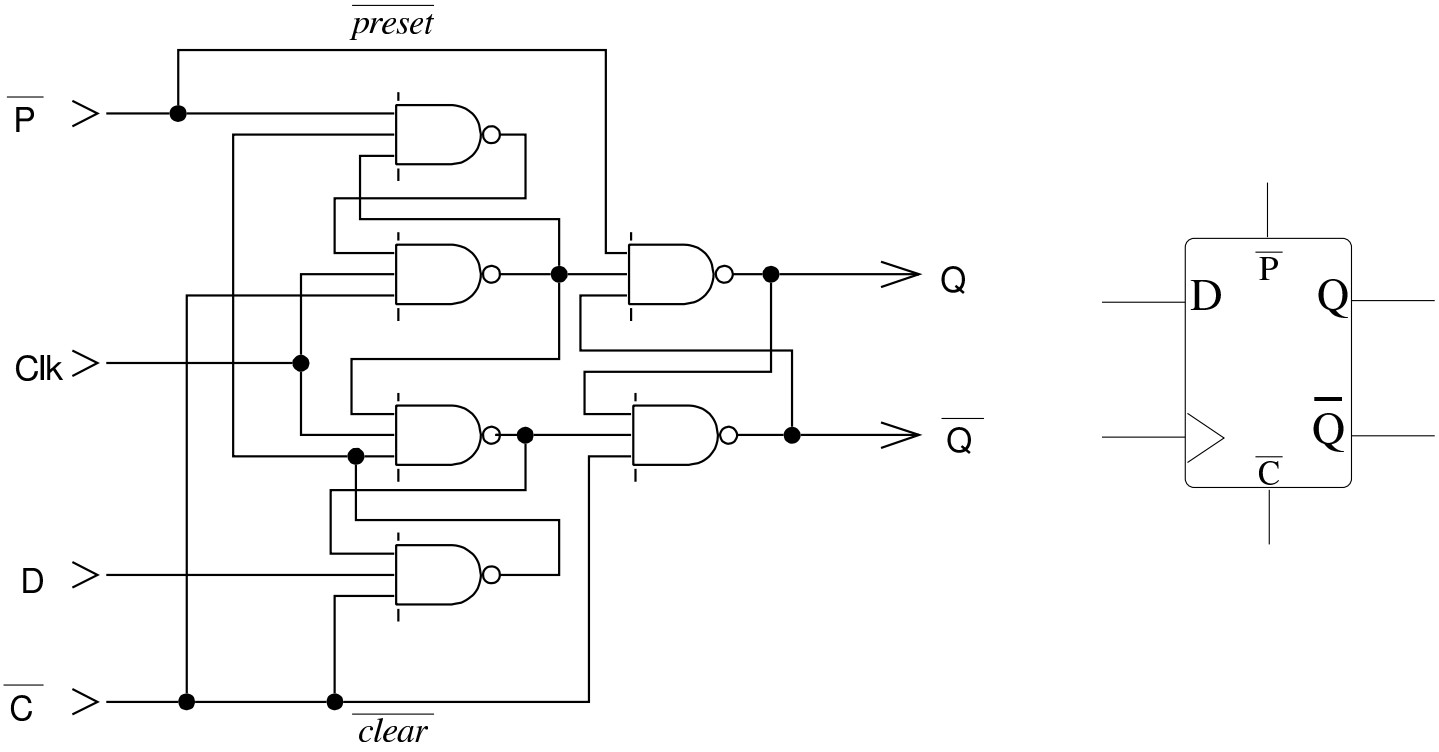
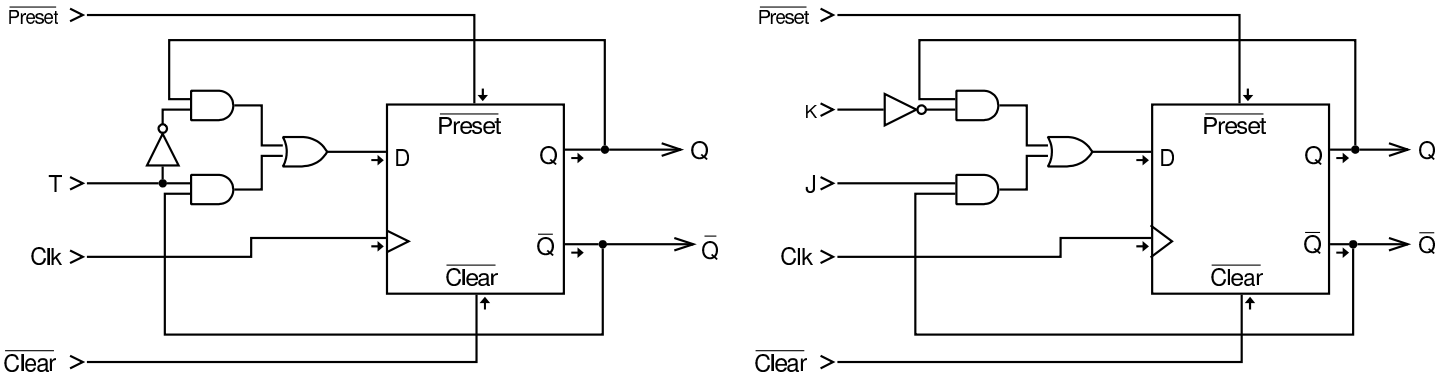


Figura u102.2. flip-flop T e JK, partendo dal flip-flop D.

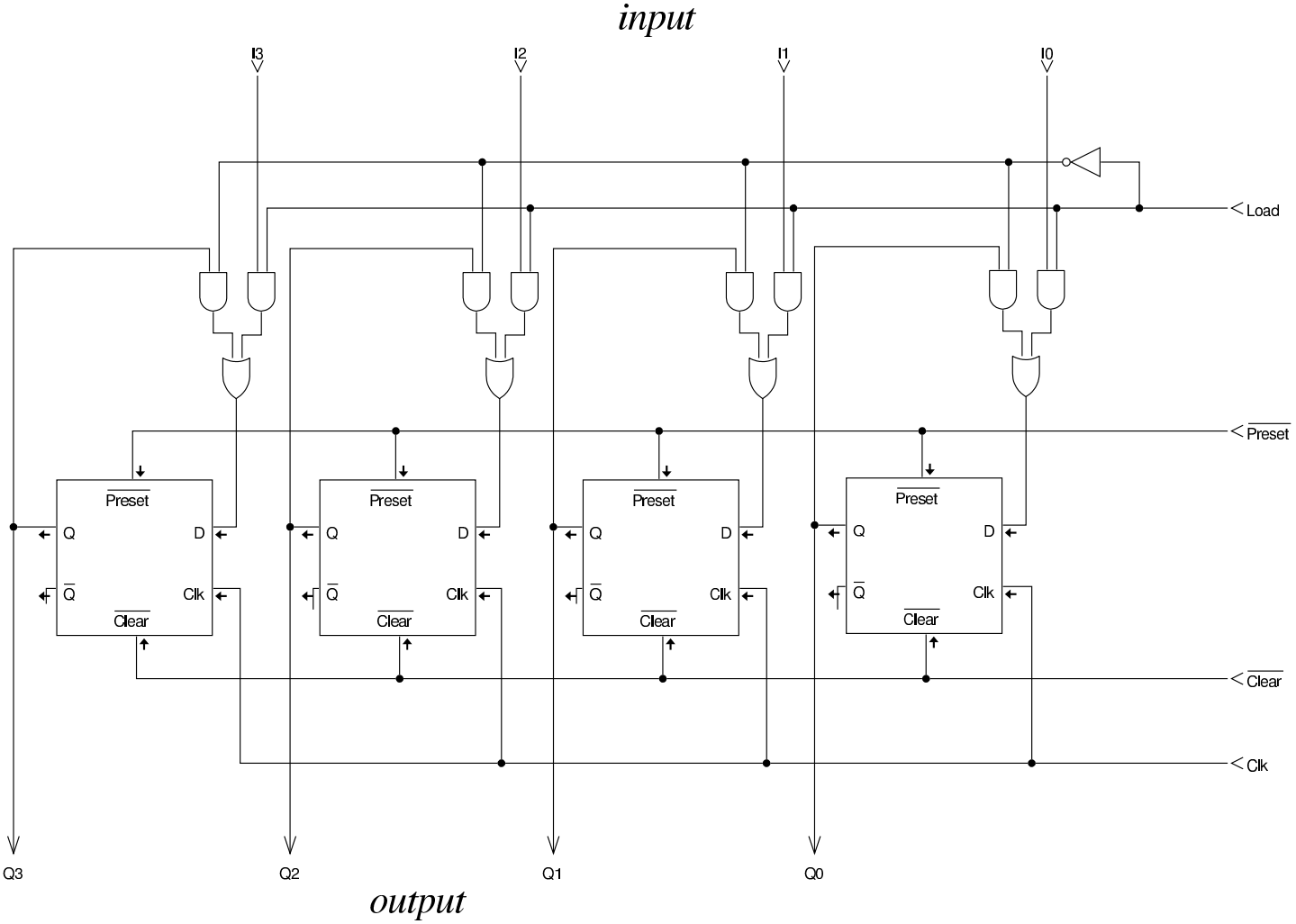


Nelle figure che appaiono nelle sezioni successive, i flip-flop possono avere l'ingresso di clock indicato con la sigla ***Clk*** (senza il triangolo che fa riferimento al margine), tuttavia si tratta sempre di flip-flop pilotati dalla variazione positiva dell'ingresso di clock.

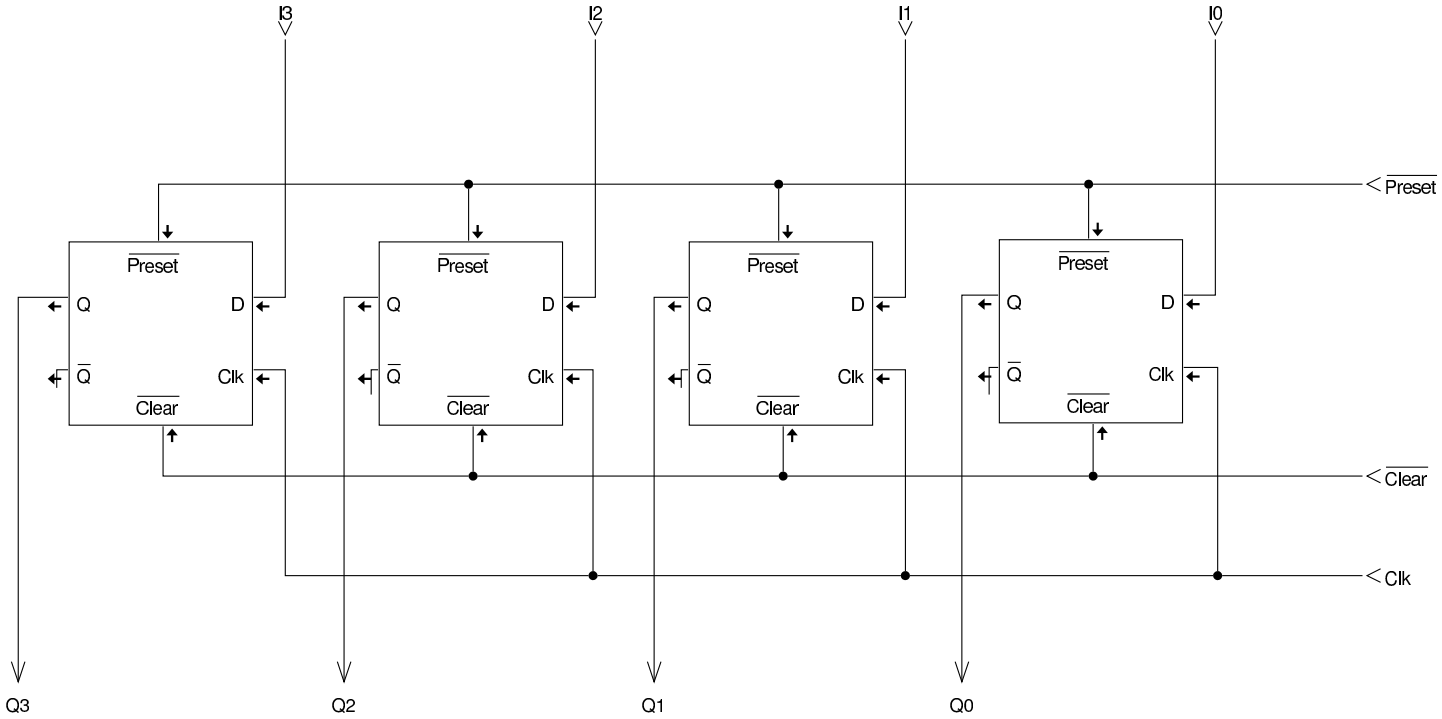
Registri semplici

Il registro più semplice è quello che è in grado di raccogliere un valore composto da più bit e di conservarlo fino a quando non vengono abilitati nuovamente gli ingressi. Nella figura successiva, il valore contenuto nel registro può essere letto dalle uscite da $\boldsymbol{Q_0}$ a $\boldsymbol{Q_3}$ e può essere immesso attraverso gli ingressi da $\boldsymbol{I_0}$ a $\boldsymbol{I_3}$. Il valore proveniente dagli ingressi da $\boldsymbol{I_0}$ a $\boldsymbol{I_3}$, viene raccolto solo quando è attivo l'ingresso ***Load***, altrimenti i flip-flop, a ogni impulso di clock, ricopiano nel proprio ingresso lo stesso valore che mostrano in uscita.

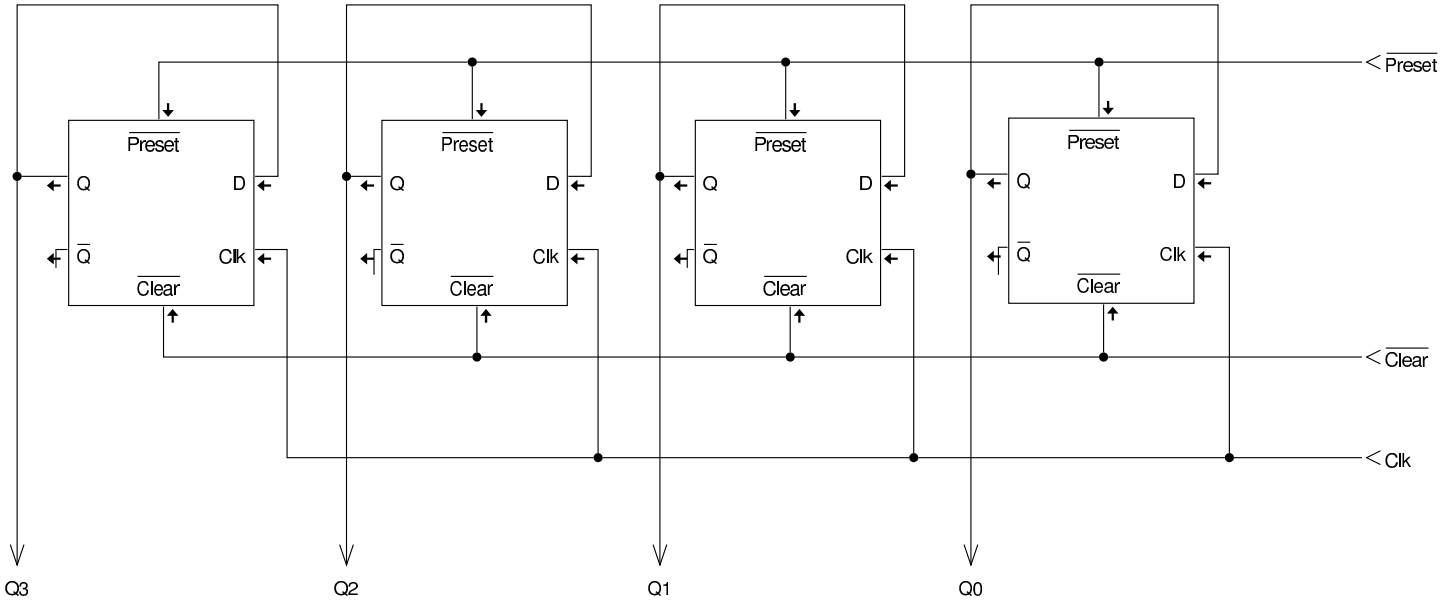
Figura u102.3. Registro a 4 bit: quando l'ingresso ***Load*** è attivo e si presenta una variazione positiva del clock, il registro accumula il valore proveniente dagli ingressi (da I_0 a I_3), altrimenti mantiene il valore accumulato precedentemente.



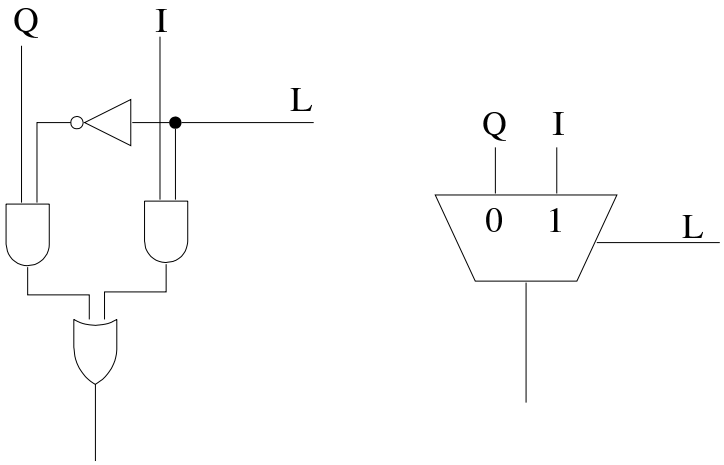
Se si analizza lo schema della figura appena apparsa, si vede che, quando il registro deve accettare il valore in ingresso, è come se il circuito si riducesse all'immagine successiva:



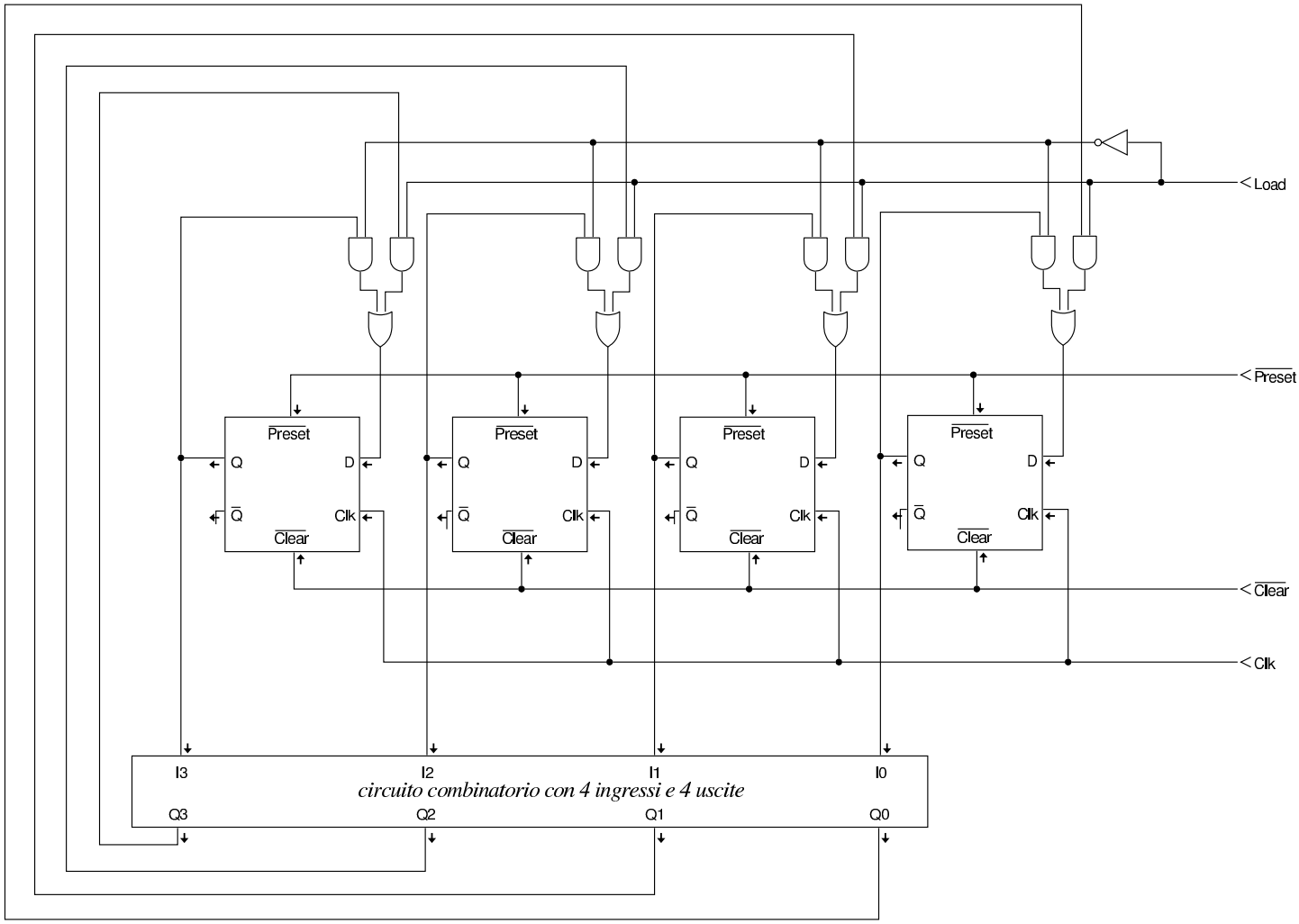
Quando invece si tratta di mantenere il valore memorizzato, è come se il circuito si riducesse allo schema della figura successiva:



Deve essere chiaro che gli ingressi ***D*** di questo tipo di registro sono controllati da un multiplatore, il quale, a ogni impulso di clock, sceglie se recepire un valore nuovo o se rinnovare quello già memorizzato in precedenza.



Una caratteristica importante di un registro comune è quella di potersi aggiornare a partire dal dato che esso stesso contiene, nell'istante in cui viene recepita la variazione dell'ingresso di clock che consente tale aggiornamento. Si osservi l'esempio della figura successiva:



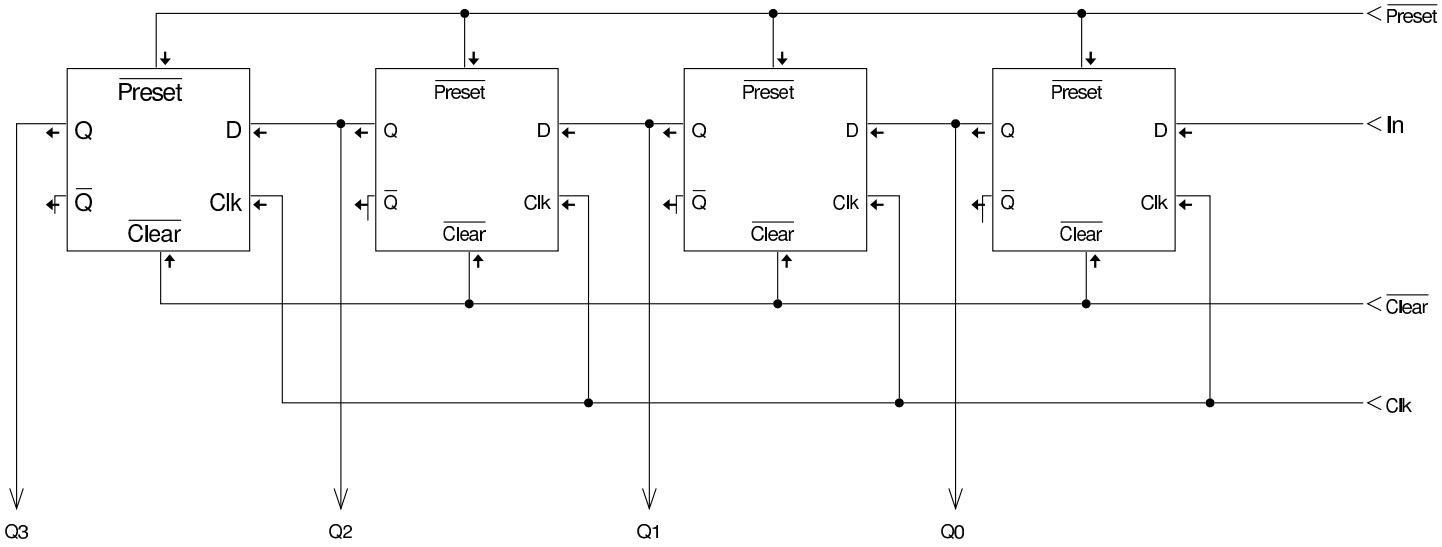
All'uscita del registro, si trova un circuito combinatorio non meglio

precisato, il quale trasforma quanto proviene dal registro, fornendo poi il valore trasformato in ingresso al registro stesso: quando l'ingresso ***Load*** è attivo e si manifesta la variazione positiva del segnale di clock, il registro recepisce il nuovo valore trasformato. In pratica, si ottiene quello che in programmazione si rappresenta solitamente come ***x=f(x)***, ovvero l'aggiornamento di una variabile con il risultato di un'espressione che la contiene.

Registri a scorrimento

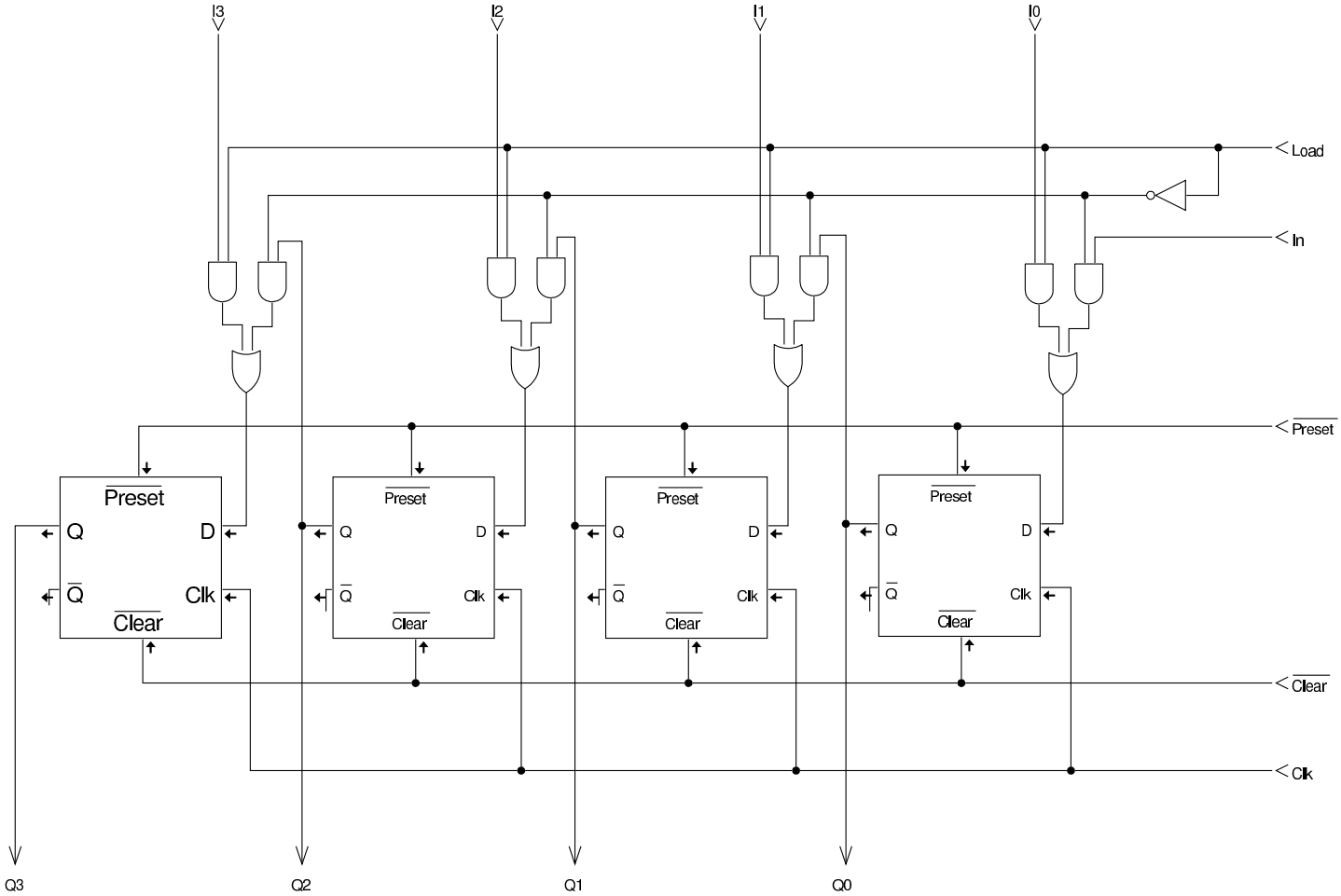
La figura successiva mostra un registro che recepisce un solo valore logico, nell'ingresso ***In***, quando si presenta la variazione positiva del segnale di clock. In quel momento, il primo flip-flop a destra aggiorna il proprio valore con il dato ottenuto dall'ingresso ***In***, mentre il secondo flip-flop si aggiorna ottenendo il valore che il primo flip-flop aveva precedentemente, e così di seguito fino all'ultimo.

Figura u102.8. Registro a scorrimento a 4 bit: quando si verifica una variazione positiva del clock, il valore in ingresso viene accumulato dal primo flip-flop a destra, mentre il secondo recepisce il valore precedente del primo, continuando fino all'ultimo. Video: *http://www.youtube.com/watch?v=I_ncbpQnCZ4* .



Un registro a scorrimento, come quello della figura precedente, può essere esteso in modo da consentire il caricamento di un valore. In tal caso si aggiunge un ingresso (***Load***), con il quale si seleziona la funzione svolta dal registro nel momento della variazione (positiva) del segnale di clock: caricamento o scorrimento.

Figura u102.9. Registro a scorrimento a 4 bit con caricamento: quando l'ingresso ***Load*** è attivo, il registro recepisce il valore dagli ingressi da I_0 a I_3, mentre diversamente fa scorrere il valore contenuto nei flip-flop, acquisendo quanto contenuto nell'ingresso ***In*** nel primo flip-flop. L'acquisizione o lo scorrimento avvengono in corrispondenza di un margine positivo del clock.



Contatori asincroni con flip-flop T

I contatori sono registri che incrementano o decrementano il proprio valore binario a ogni impulso di clock. Sono asincroni quei contatori i cui flip-flop non sono controllati tutti dallo stesso segnale di clock. Il contatore più semplice è costituito da flip-flop T, in quanto questo tipo di flip-flop, quando l'ingresso ***T*** è attivo, inverte il valore delle

uscite ogni volta che riceve la variazione del segnale di clock che serve a farlo scattare. Le figure mostrano l'uso dei flip-flop T in cascata, nel senso che l'ingresso di clock del successivo è pilotato dall'uscita del precedente: per questo si tratta di contatori asincroni.

Figura u102.10. Contatore asincrono crescente a 4 bit: il flip-flop successivo è pilotato dall'uscita ***Q'*** del flip-flop precedente. Video: *http://www.youtube.com/watch?v=-mfL6B0w9UI* .

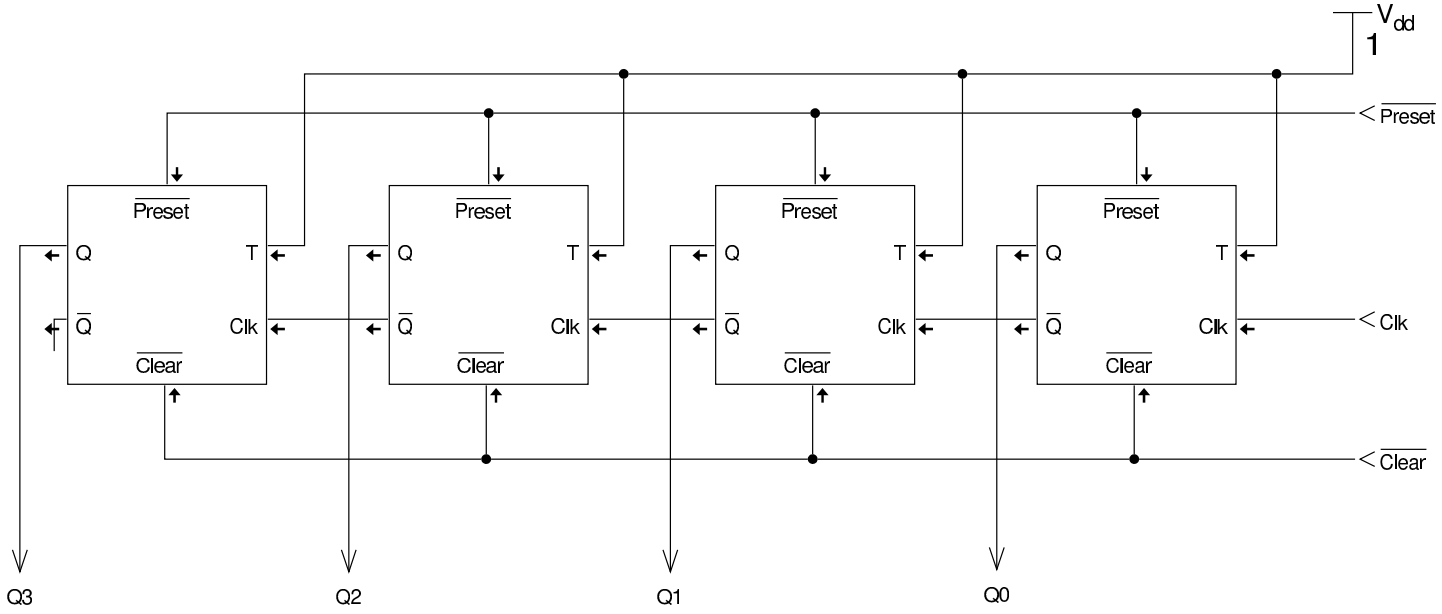
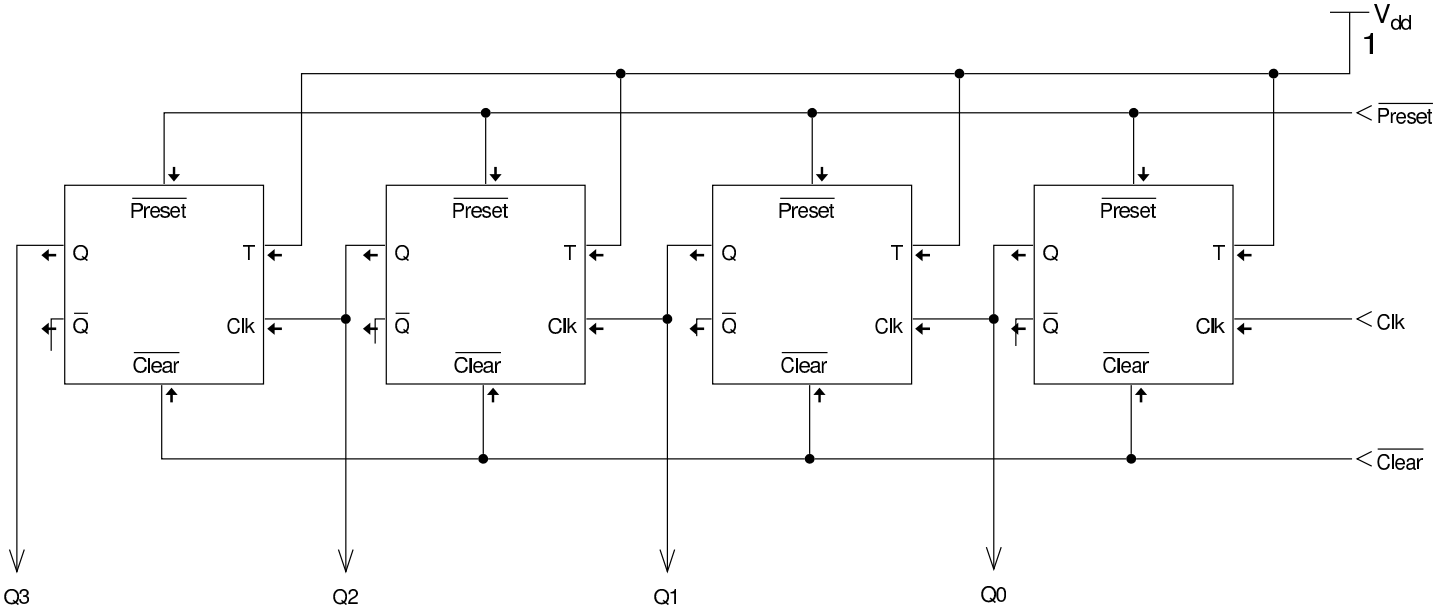


Figura u102.11. Contatore asincrono decrescente a 4 bit: il flip-flop successivo è pilotato dall'uscita ***Q*** del flip-flop precedente. Video: *http://www.youtube.com/watch?v=fvhina5QEnI* .

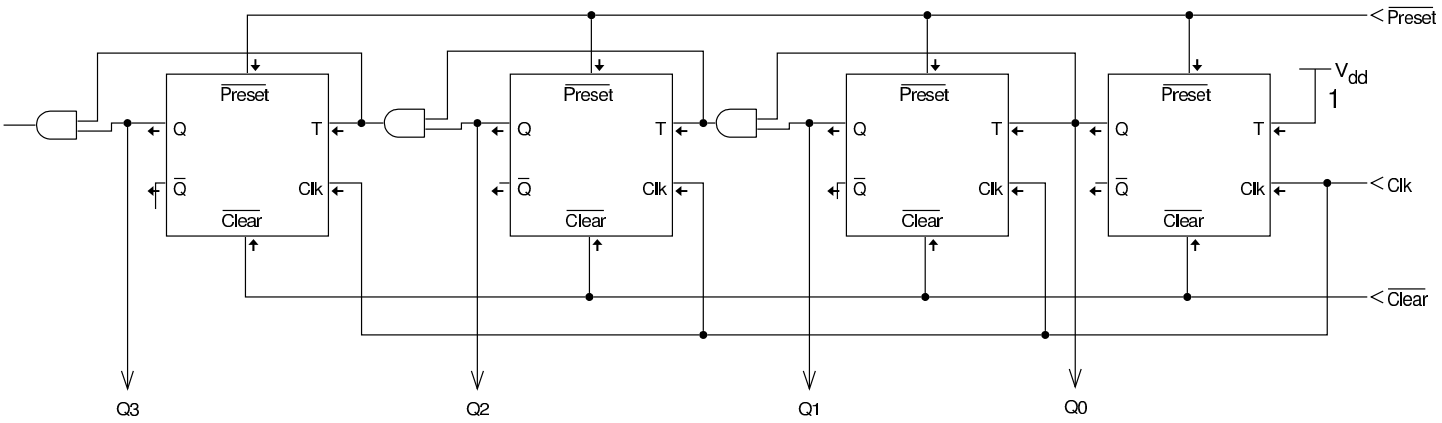


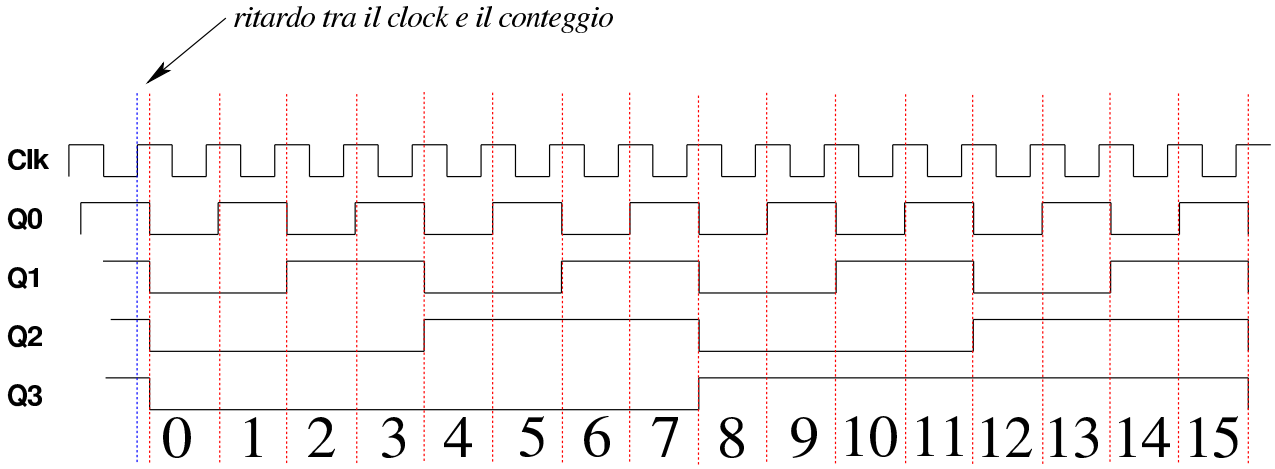
Il difetto dei contatori asincroni sta nel fatto che le oscillazioni delle uscite successive sono leggermente in ritardo rispetto a quelle delle uscite antecedenti. Questo problema si accentua al crescere del rango del registro, perché i ritardi di accumulano a ogni passaggio, da un flip-flop al successivo.

Contatori sincroni con flip-flop T

I contatori sincroni possono essere costruiti con flip-flop T, ma in modo differente da quanto descritto nella sezione precedente; tuttavia, quando si modificano questi contatori per consentire l'acquisizione di un valore binario di partenza, è necessaria una costruzione basata su flip-flop D.

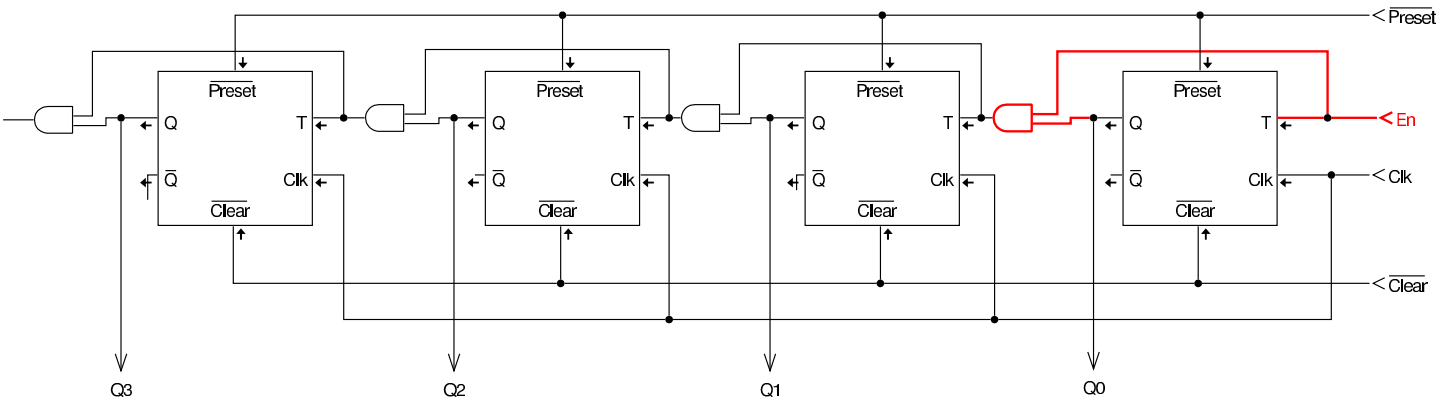
Figura u102.12. Contatore sincrono crescente a 4 bit: le uscite sono sincrone tra loro, ma inevitabilmente sono leggermente in ritardo rispetto all'impulso di clock. Video: *http://www.youtube.com/watch?v=Nh5CzRopNdY* .





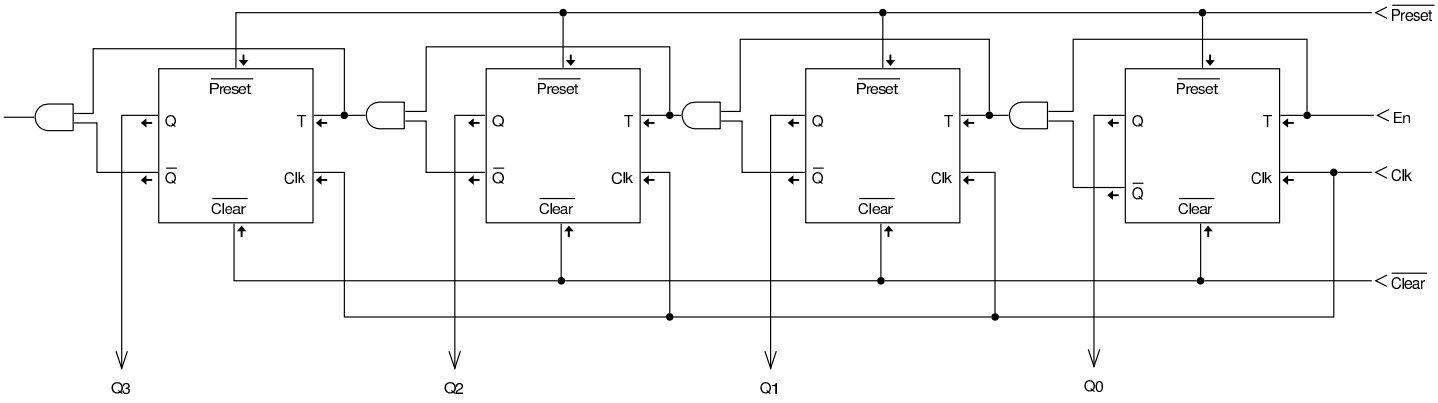
Il contatore sincrono richiede che l'ingresso ***T*** del primo flip-flop sia sempre attivo. Tuttavia, se si controlla tale ingresso è possibile sospendere il processo di conteggio, senza azzerare il valore raggiunto. Pertanto, si può aggiungere un ingresso di abilitazione, da collegare all'ingresso ***T*** del primo flip-flop, proprio per controllare il procedere del conteggio.

Figura u102.13. Contatore sincrono crescente a 4 bit con controllo di abilitazione (ingresso ***En***).



La realizzazione di un contatore sincrono decrescente procede in maniera analoga, invertendo le uscite nella sequenza di flip-flop T.

Figura u102.14. Contatore sincrono decrescente a 4 bit con controllo di abilitazione.



Contatori sincroni con flip-flop D

In generale, è più conveniente realizzare dei contatori sincroni attraverso dei flip-flop D, dato che possono poi essere modificati facilmente per consentire il caricamento parallelo di un valore. Nelle figure di questa sezione si mostrano, per ora, solo contatori equivalenti a quelli già apparsi nella sezione precedente.

Figura u102.15. Contatore sincrono crescente a 4 bit con controllo di abilitazione, basato su flip-flop D.

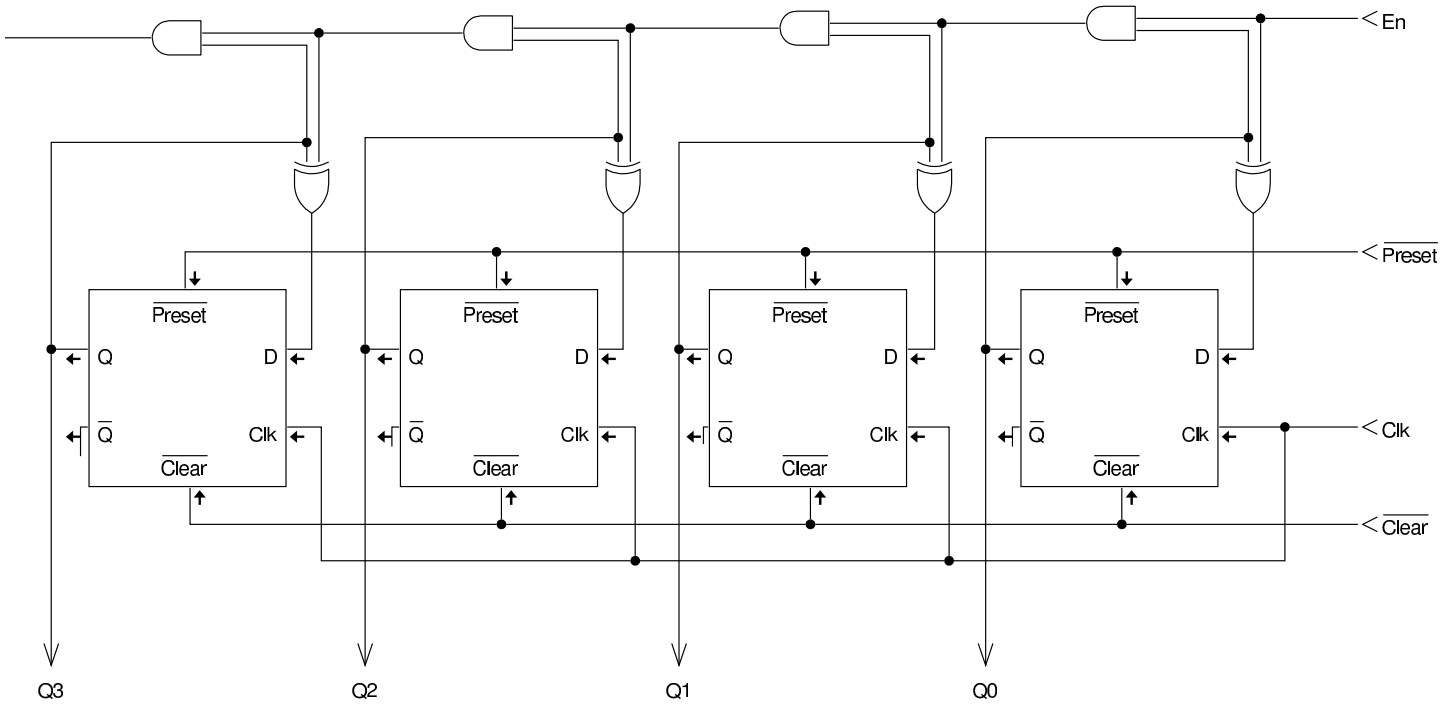
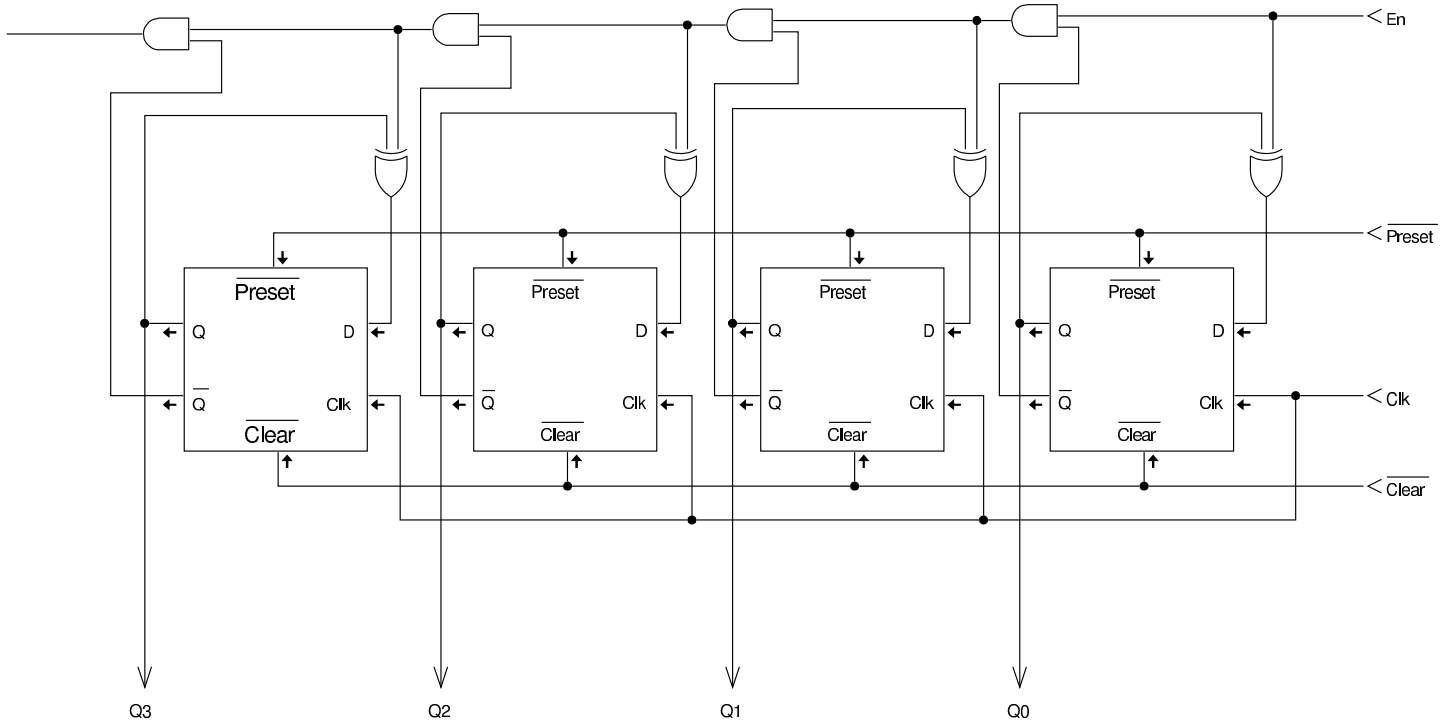


Figura u102.16. Contatore sincrono decrescente a 4 bit con controllo di abilitazione, basato su flip-flop D.



Contatori sincroni con caricamento parallelo

Il contatore è completo solo quando consente anche il caricamento parallelo. Nelle figure di questa sezione si vedono contatori basati su flip-flop D che consentono il caricamento parallelo.

Figura u102.17. Contatore sincrono crescente a 4 bit con caricamento parallelo: l'ingresso ***Load*** fa sì che il valore contenuto negli ingressi da I_0 a I_3 venga caricato nei flip-flop; poi, se l'ingresso ***En*** non è attivo, il valore viene mantenuto tale e quali, altrimenti viene incrementato a ogni impulso di clock.

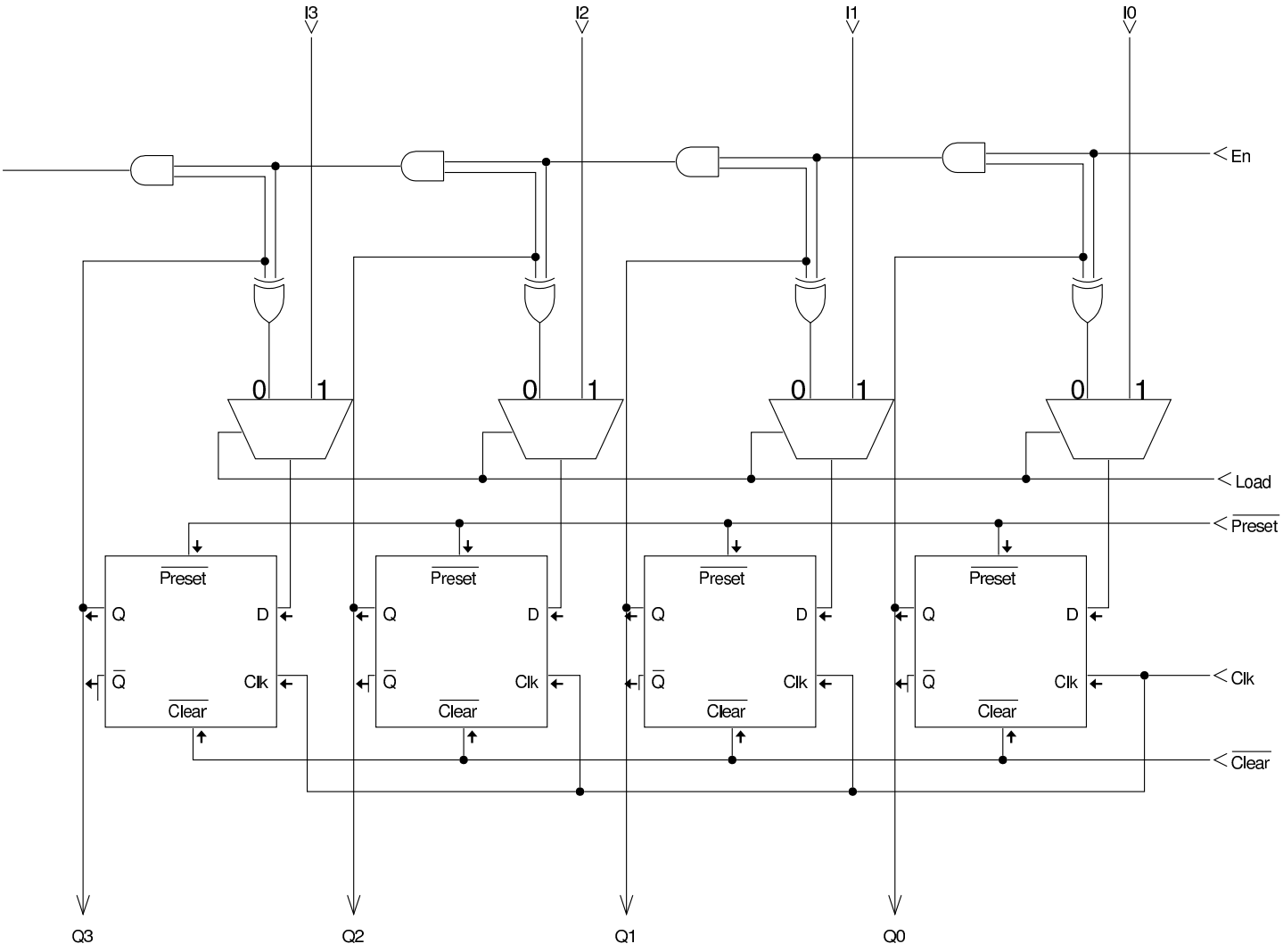


Figura u102.18. Contatore sincrono decrescente a 4 bit con caricamento parallelo: l'ingresso ***Load*** fa sì che il valore contenuto negli ingressi da I_0 a I_3 venga caricato nei flip-flop; poi, se l'ingresso ***En*** non è attivo, il valore viene mantenuto tale e quali, altrimenti viene decrementato a ogni impulso di clock.

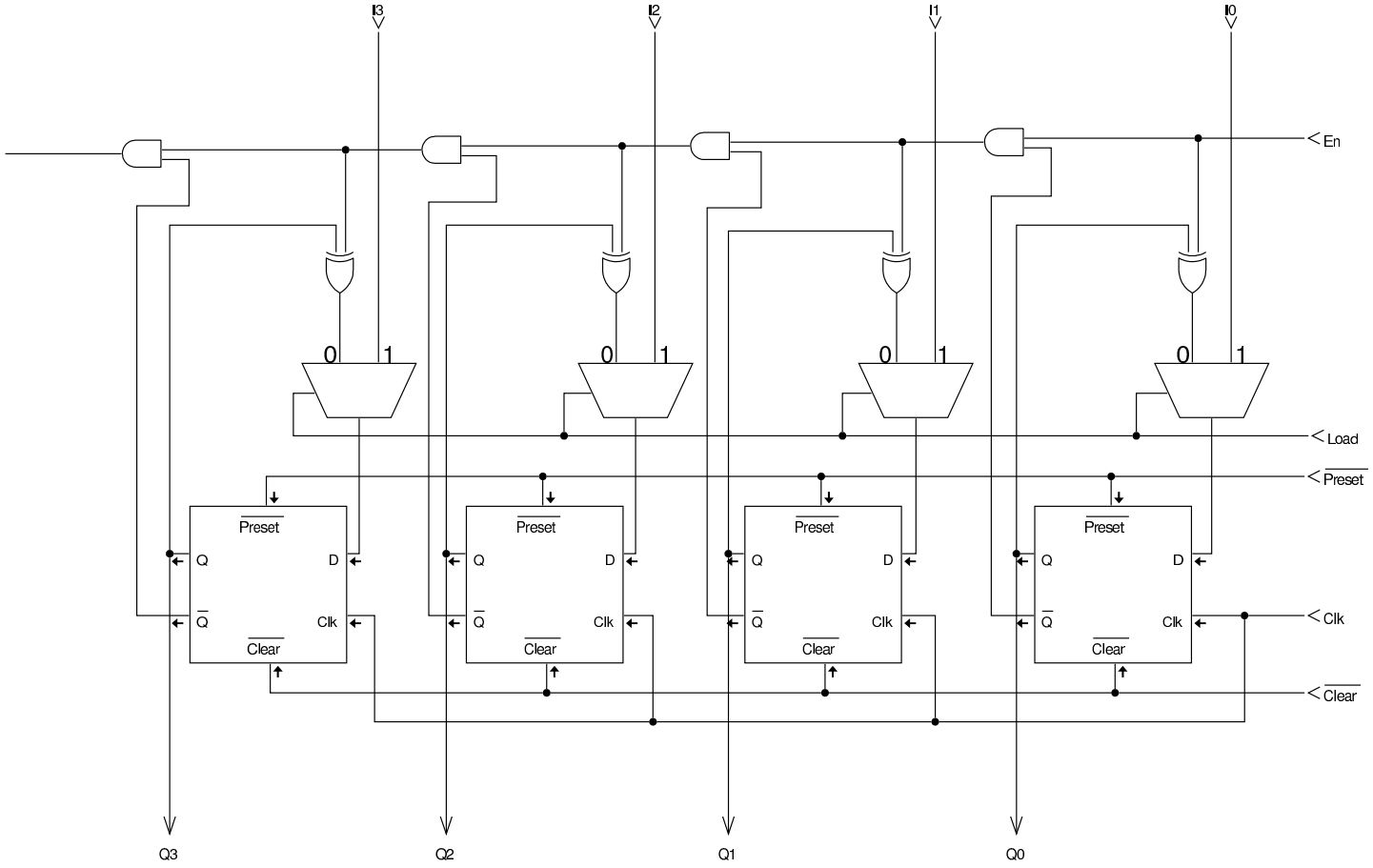
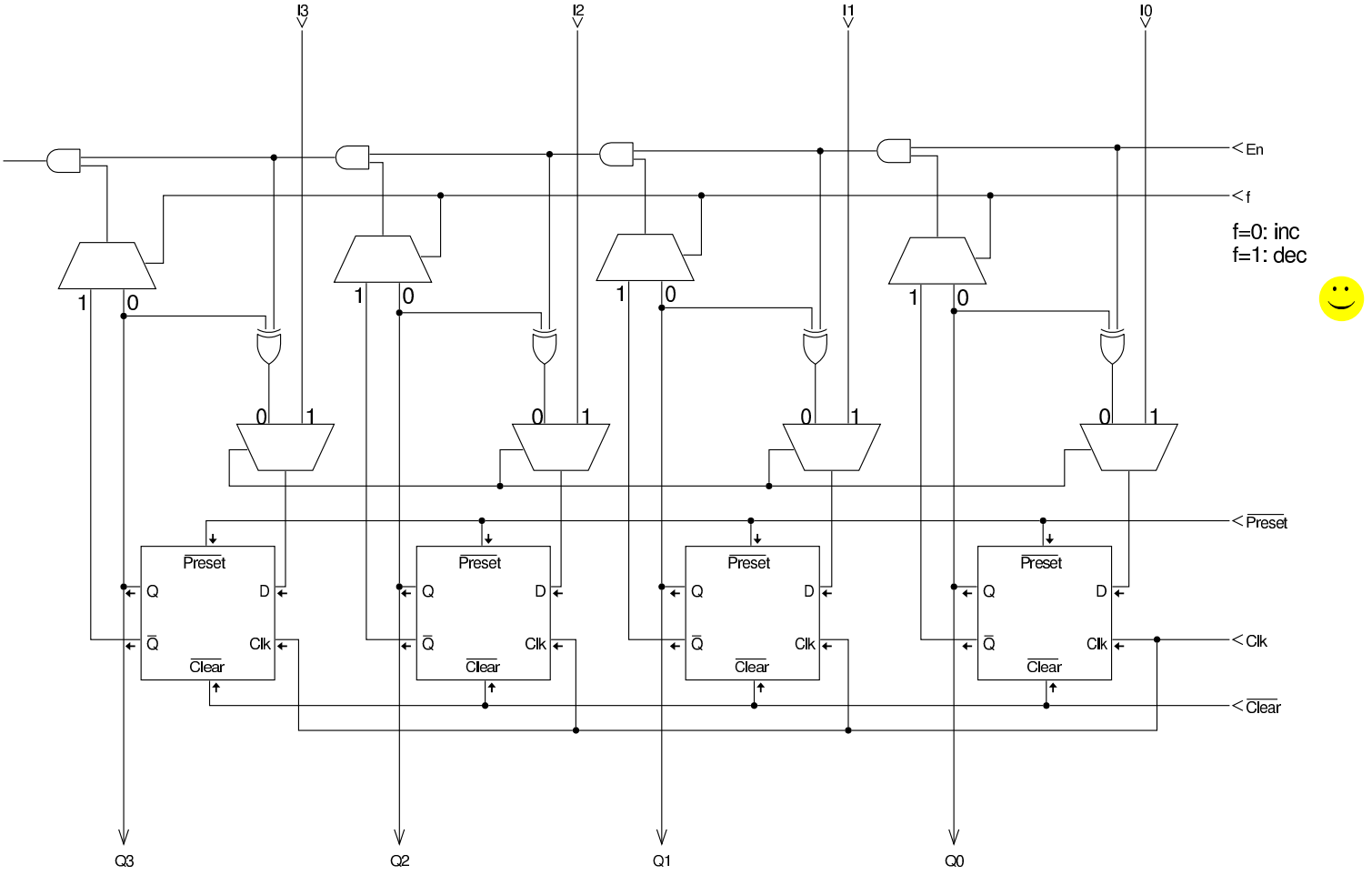


Figura u102.19. Contatore sincrono crescente o decrescente a 4 bit con caricamento parallelo: l'ingresso ***f*** consente di selezionare il funzionamento in qualità di contatore crescente o decrescente.



1838

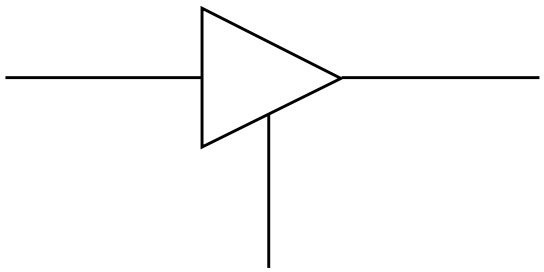
Bus e unità di controllo

Registri e circuiti combinatori vengono spesso raggruppati condividendo un certo insieme di connessioni (fili). Questo insieme di connessioni costituisce quello che è noto come ***bus***. Un bus nel quale diversi componenti hanno facoltà di scrivere e di leggere è solitamente un «bus dati», perché serve allo scambio di informazioni tra questi componenti; tuttavia, va osservato che solo un componente alla volta può scrivere, mentre non ci sono limitazioni per la lettura concorrenziale.

Bus con il buffer a tre stati

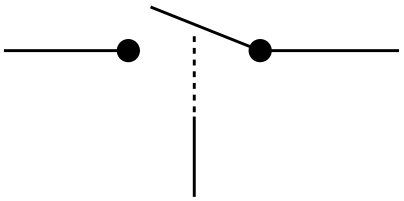
La realizzazione di un bus dati può avvenire con l'ausilio di un multiplatore o di un «buffer a tre stati» (*tri-state buffer*), ovvero un componente che funziona come porta non invertente, ma con un ingresso addizionale, con il quale è possibile abilitare il funzionamento in qualità di porta non invertente (*buffer*), oppure si può isolare completamente l'uscita. Nel disegno seguente si vede a confronto il simbolo del buffer a tre stati con quello che farebbe un interruttore tradizionale:

«a2» 2013.11.11 --- Copyright © Daniele Giacomini -- appunti2@gmail.com http://informaticalibera.net



0 = isola

1 = buffer



0 = aperto

1 = chiuso

Il buffer a tre stati, oppure il multiplatore, servono quindi per controllare la scrittura nel bus; tra le due opzioni, la scelta del buffer a tre stati è quella più economica e più semplice, perché il multiplatore comporta l'uso di molte porte logiche e, di conseguenza, introduce un ritardo di propagazione maggiore.

Nelle figure successive si vede l'adattamento di un registro semplice, con buffer a tre stati, per il collegamento a un bus; poi si vede come si possono collegare registri o altri componenti così predisposti, distinguendo tra un bus dati e un bus di controllo.

Figura u103.2. Registro semplice adattato per connettersi con un bus dati.

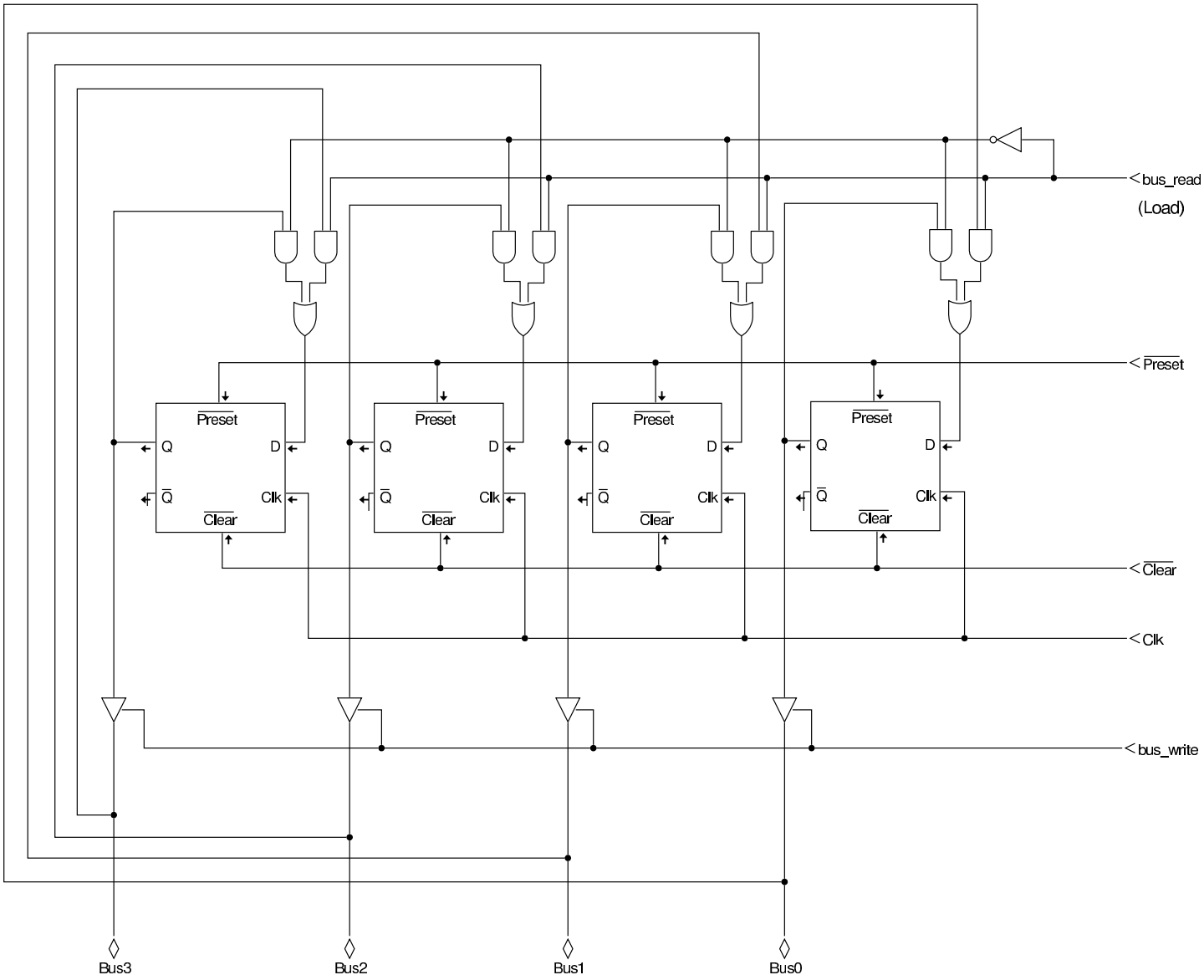


Figura u103.3. Schemi più semplici dell'adattamento di un registro. L'ingresso ***bus_read***, o ***br***, richiede al registro il caricamento del valore che si può leggere dal bus, mentre l'ingresso ***bus_write***, o ***bw***, concede al registro di immettere i propri dati nel bus.

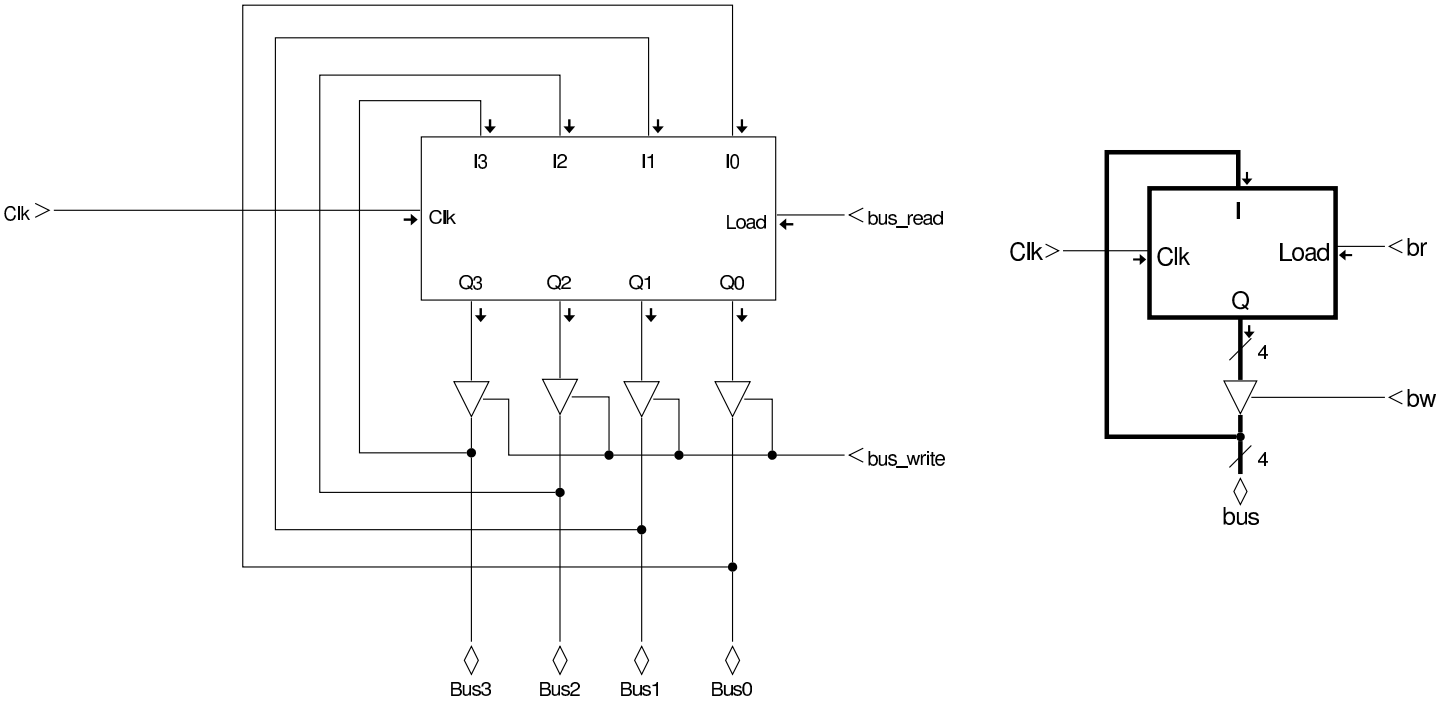
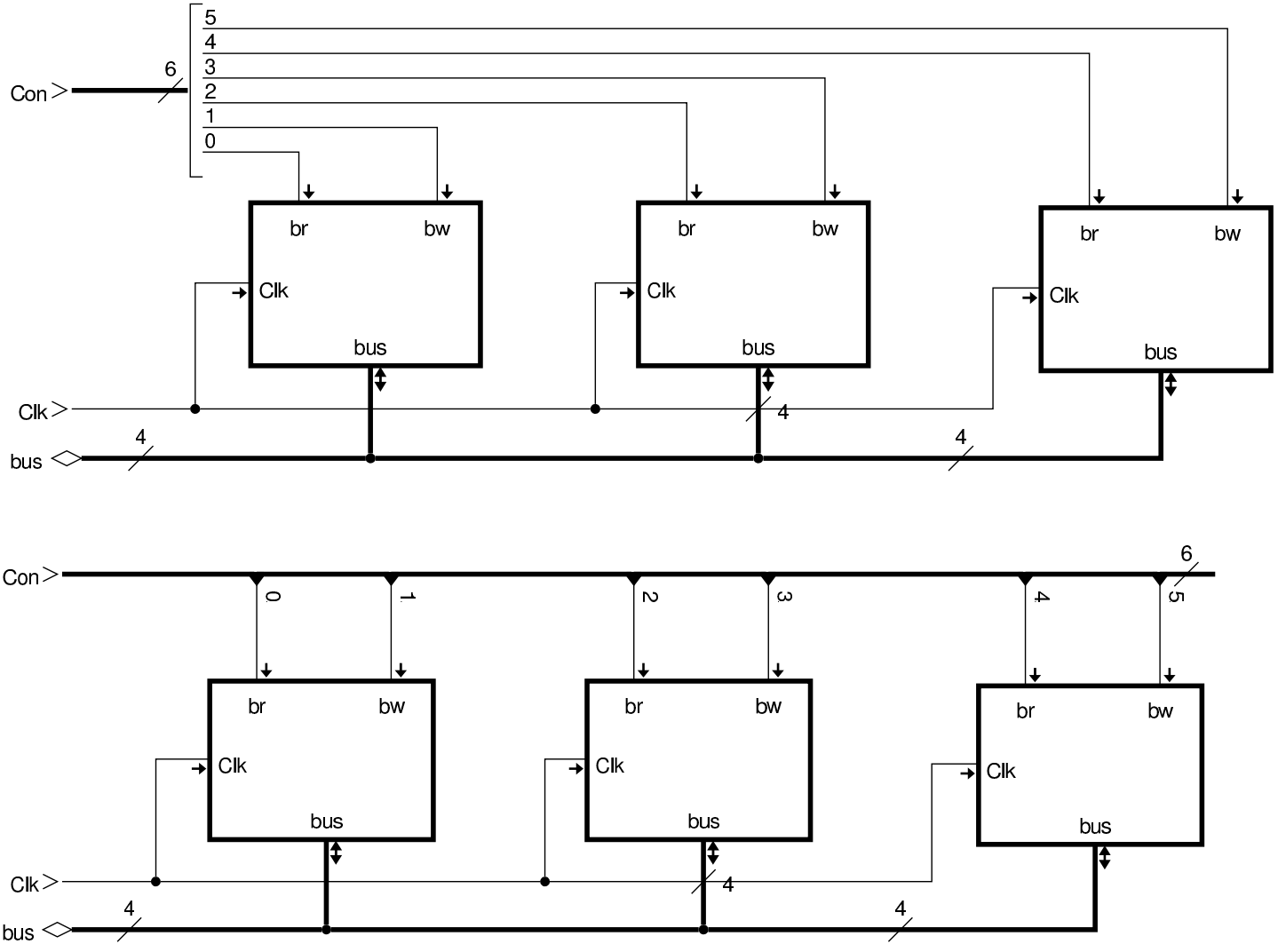


Figura u103.4. Collegamento di più componenti in un bus: gli ingressi che abilitano la lettura dal bus o la scrittura nel bus, assieme ad altri ingressi di controllo eventuali, si collegano a un bus di controllo secondario. Nel secondo disegno si vede un modo alternativo di rappresentare il collegamento al bus di controllo, prelevando un filo alla volta.

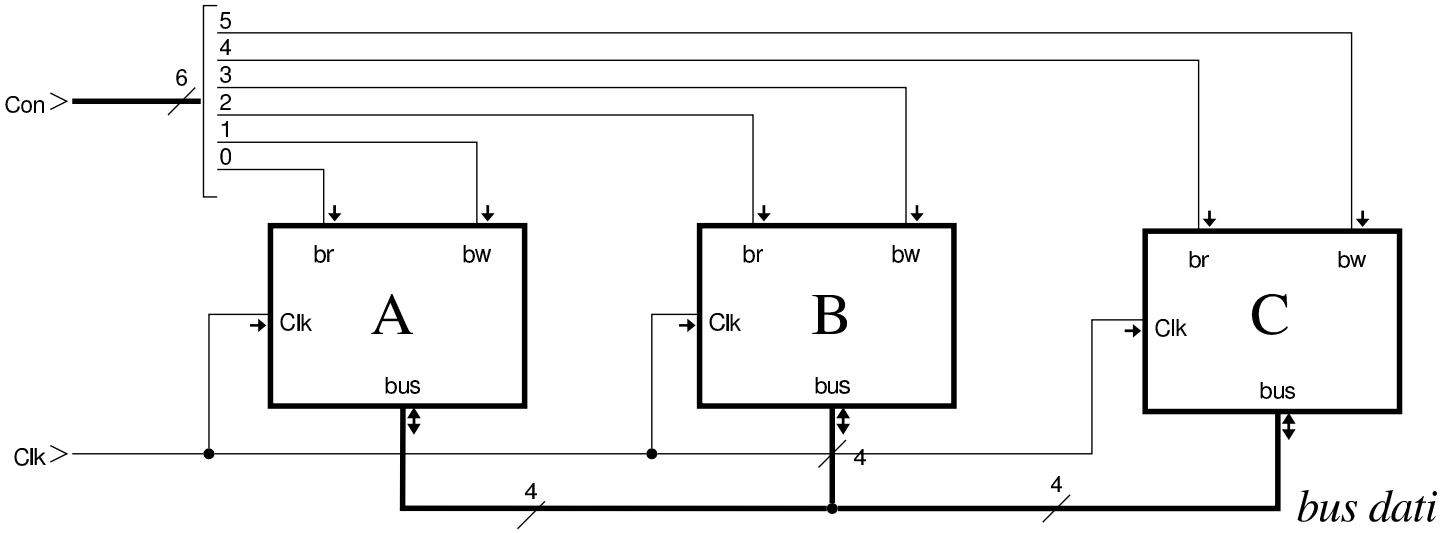


Unità di controllo del bus

Per dirigere correttamente l'utilizzo di un bus dati, è necessario un sistema di controllo, attraverso il quale scandire le fasi di ogni procedimento che si intende applicare. Per comprendere il meccanismo si

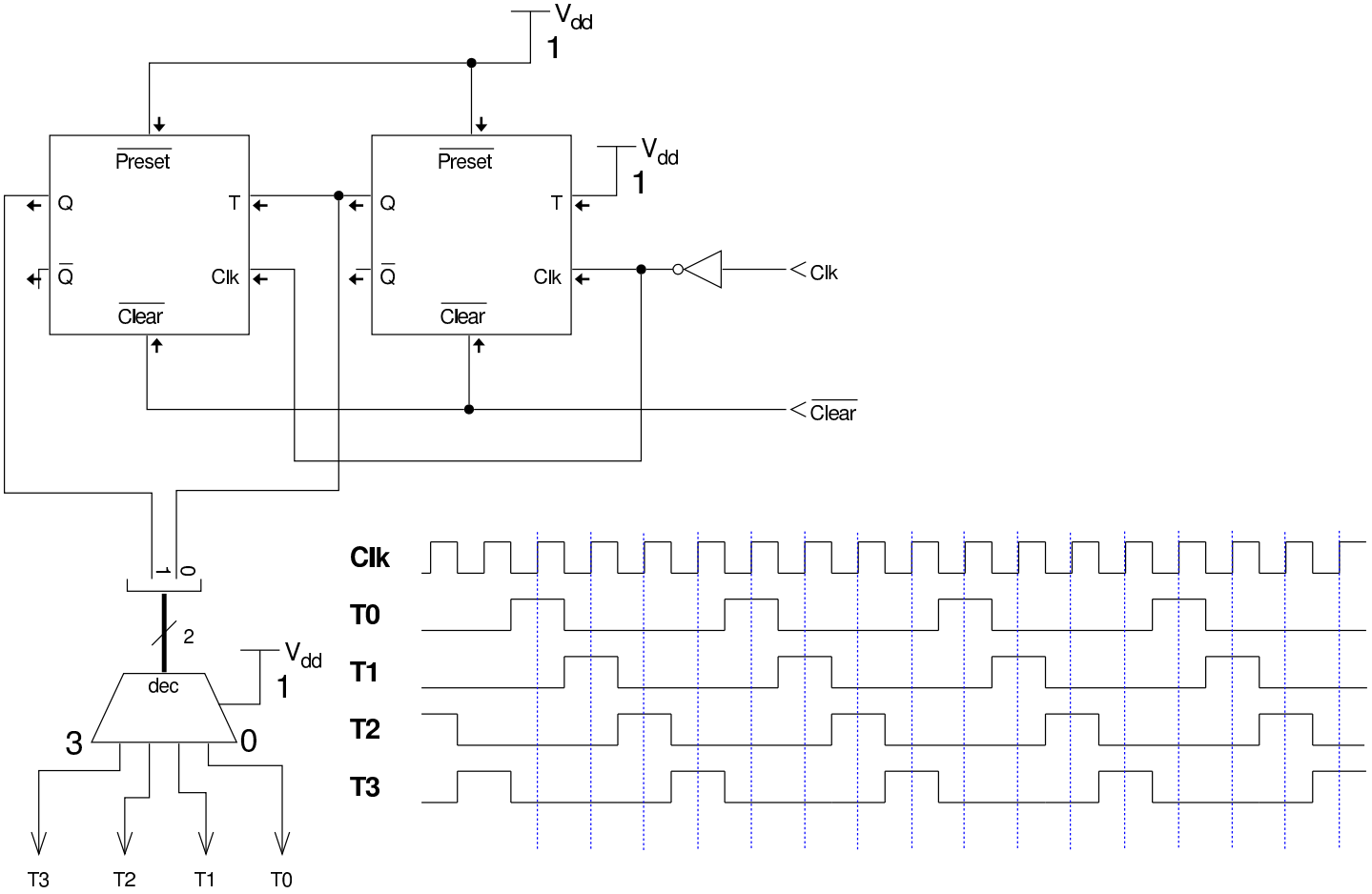
riprende lo schema già apparso in cui tre componenti sono connessi su un bus dati e sono controllati da un bus di controllo; su questi componenti si ipotizzano soltanto operazioni di trasferimento delle informazioni.

Figura u103.5. Esempio di riferimento con tre componenti che condividono un bus dati.



Per scandire le operazioni da svolgere, l'unità di controllo ha bisogno di un contatore. Nella realizzazione circuitale di un'unità di controllo si fa riferimento normalmente a un contatore a scorrimento, il quale può essere realizzato come si vede nella figura successiva. Ciò che è importante e che si evidenzia nel tracciato che appare nella figura, è che le uscite risultino attive a cavallo dell'impulso di clock usato per pilotare i componenti del bus. Per ottenere questo risultato, l'impulso di clock che viene usato dal contatore viene invertito.

Figura u103.6. Contatore a scorrimento, usato per l'unità di controllo del bus di esempio: il segnale di clock viene invertito, in modo tale che le uscite T_n siano attive a cavallo del margine positivo che viene usato dai componenti del bus dati. Video: *http://www.youtube.com/watch?v=vXco4E4SNTo* .



L'unità di controllo, realizzata attraverso circuiti logici, deve mettere assieme un decodificatore della funzione da applicare con le uscite del contatore a scorrimento. Per esempio, seguendo lo schema della figura successiva, si vede che la funzione f_1, con la quale si vuole copiare il valore del registro ***A*** nel registro ***B*** e poi dal registro ***B*** nel registro ***C***, il decodificatore attiva la seconda linea a partire dall'alto (etichettata con f_1); quindi, attraverso il collegamento di porte AND, si fa in modo di attivare le uscite ***Aw*** e ***Br*** nel momento in cui il

contatore a scorrimento ha l'uscita T_0 attiva; quindi, analogamente, si fa in modo di attivare le uscite ***Bw*** e ***Cr*** nel momento in cui il contatore a scorrimento ha l'uscita T_1 attiva. ***Aw***, ***Br***, e ***Bw*** e ***Cr***, attivate secondo la scansione descritta, vanno ad attivare gli ingressi di lettura-scrittura dei registri rispettivi, consentendo il trasferimento di dati previsto. La realizzazione dell'unità di controllo della figura successiva è estremamente semplificata e il tempo T_2 non viene nemmeno utilizzato, comportando quindi un'attesa inutile in quel momento; tuttavia, il tempo T_3 serve invece per bloccare il segnale di clock nell'unità di controllo, la quale richiede di essere azzerata per poter recepire un nuovo comando dall'ingresso ***f***.

Figura u103.7. Unità di controllo: sono previste solo quattro funzioni molto semplici, per le quali bastano solo due tempi; pertanto il tempo T_2 rimane inutilizzato e il tempo T_3 serve a bloccare l'unità di controllo fino a quando l'ingresso ***Run*** viene azzerato e poi riattivato. Video: *http://www.youtube.com/watch?v=r-DZgjy-ya0* .

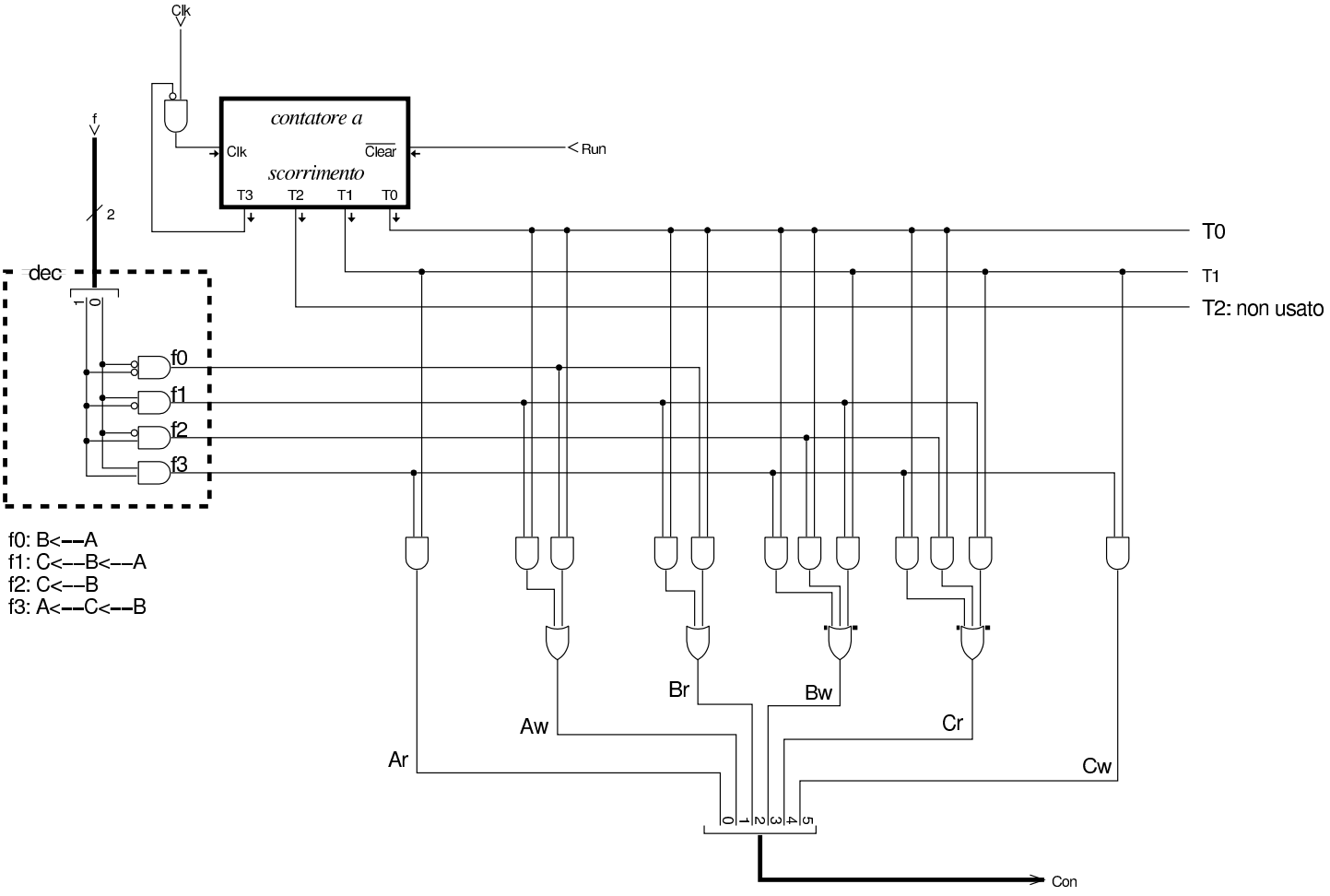
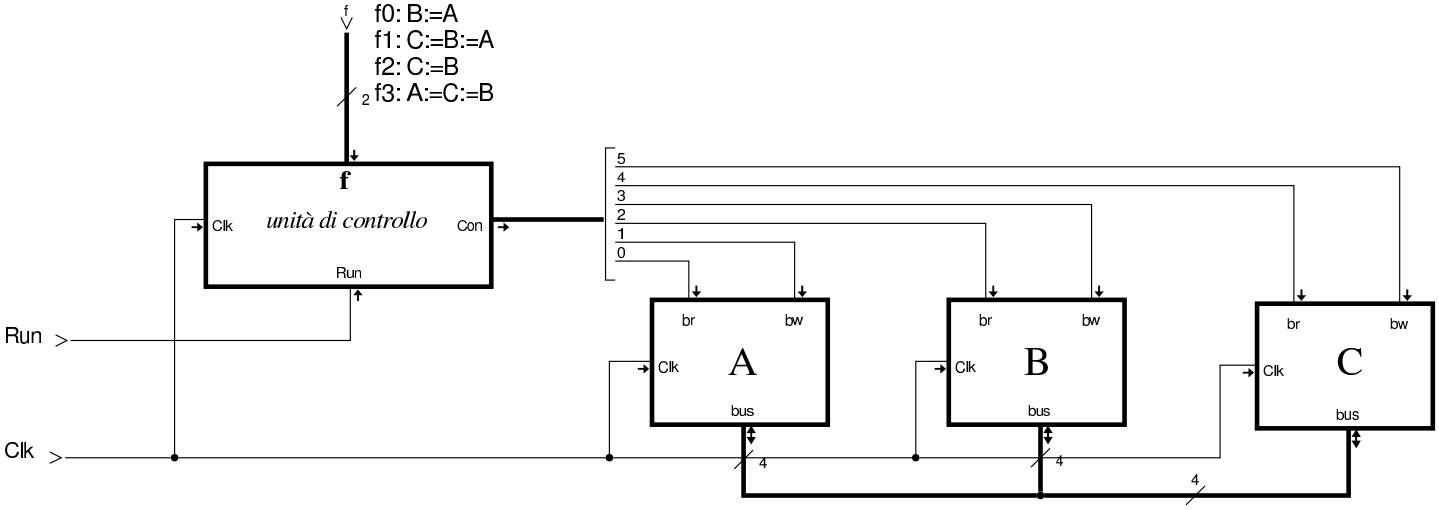


Figura u103.8. Unità di controllo connessa ai componenti che deve pilotare. Video: *http://www.youtube.com/watch?v=kpET2kEcUIo* .

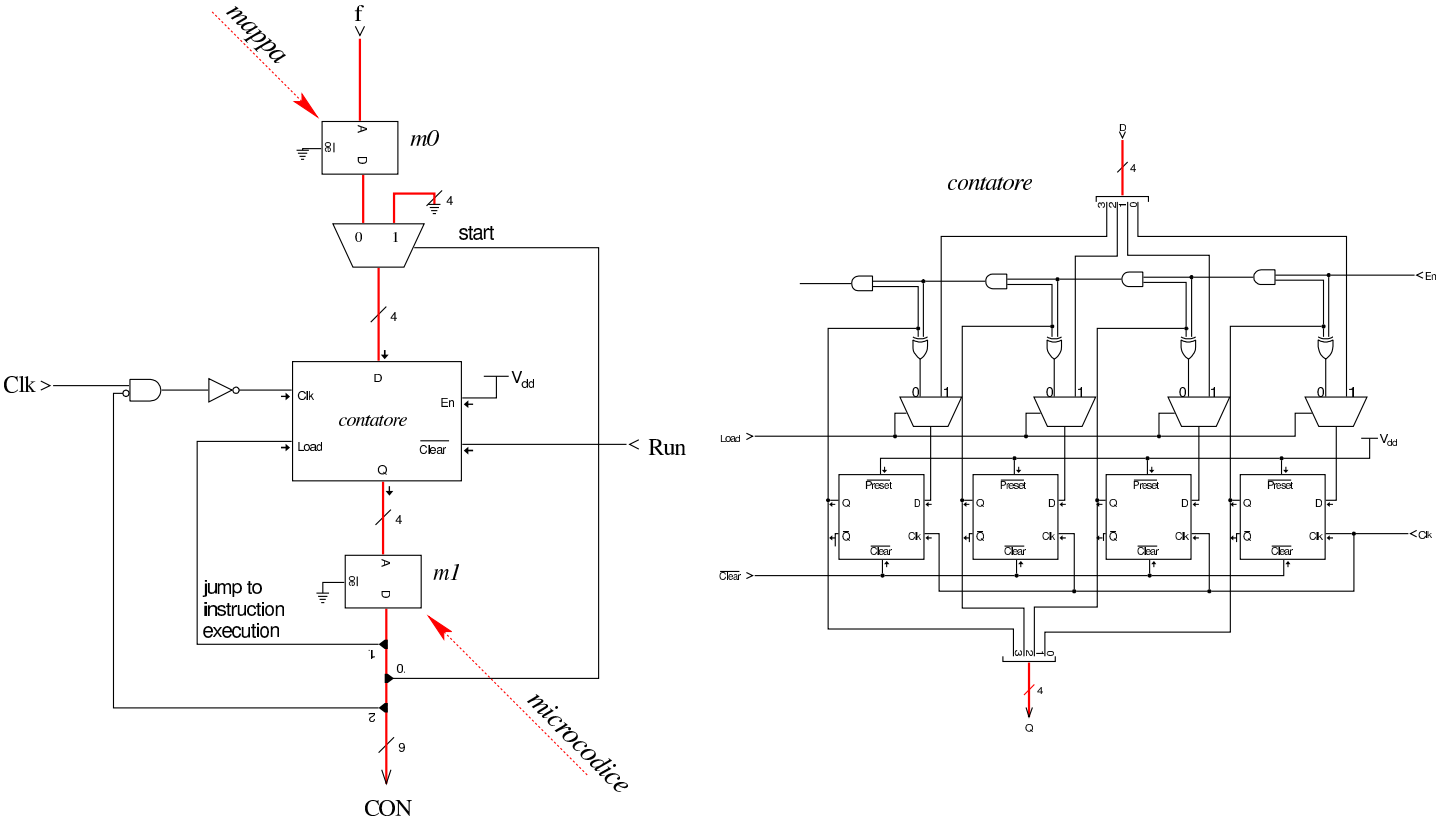


Microcodice

«

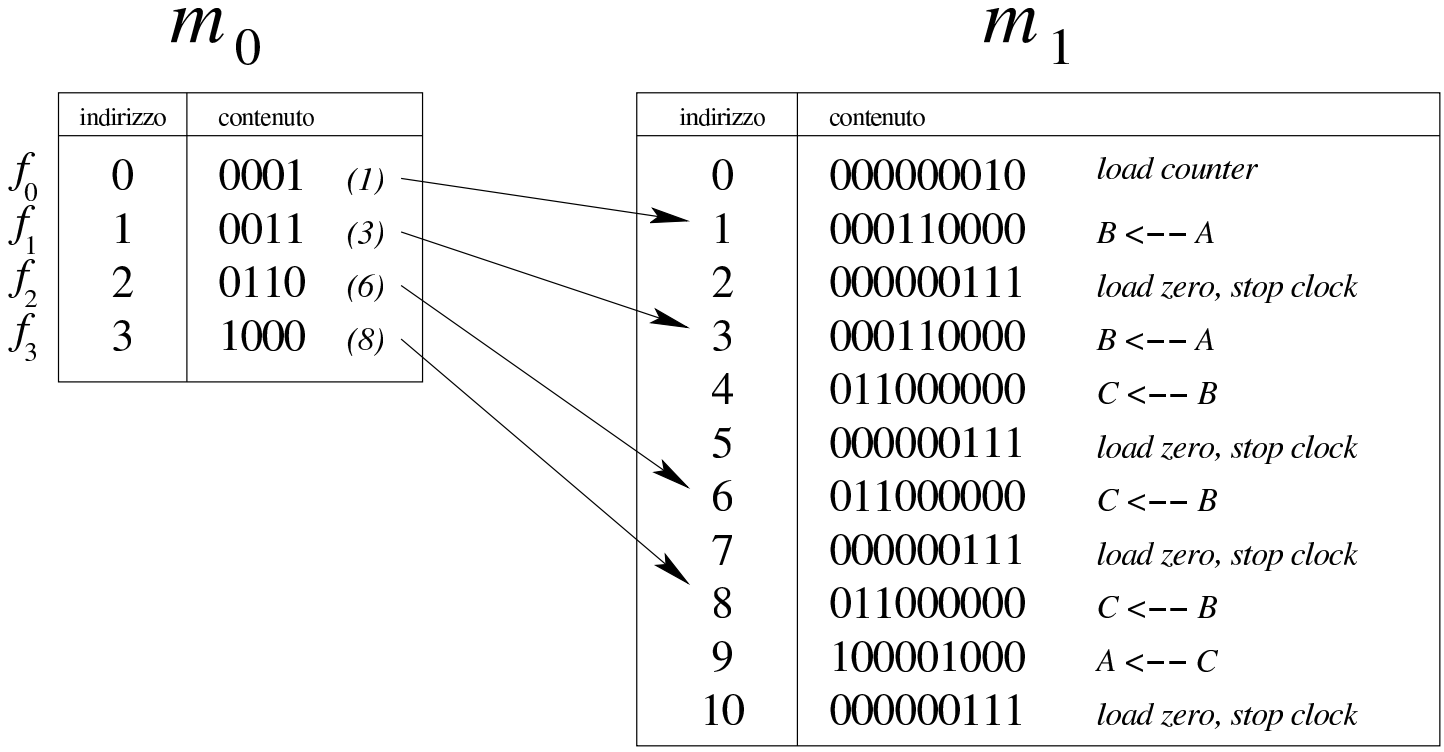
La realizzazione di un'unità di controllo di un bus dati, sotto forma di circuito logico, può risultare in un lavoro estremamente complesso. Per questa ragione, si utilizza solitamente una memoria ROM (in sola lettura), suddivisa in due parti: ***mappa*** e ***microcodice***. Le due parti della memoria ROM vanno viste come due tabelle: la prima traduce la funzione desiderata in un indirizzo che punta alla seconda, dove inizia il codice che descrive i vari passaggi da eseguire. Si osservi la figura successiva, dove a sinistra c'è lo schema dell'unità di controllo e a destra c'è lo schema del contatore utilizzato all'interno della stessa.

Figura u103.9. Unità di controllo realizzata attraverso memorie ROM: a sinistra lo schema a blocchi dell'unità, a destra la scomposizione del contatore a quattro bit (un semplice contatore incrementante, con possibilità di caricare un valore, basato su flip-flop D). Video: *http://www.youtube.com/watch?v=mBGhNP3Uujs* .



Le due memorie ROM sono denominate, rispettivamente, ***m0*** ed ***m1***. Nella figura successiva si vede il contenuto delle due memorie, rappresentato in forma tabellare, mettendo in corrispondenza l'indirizzo in ingresso della memoria (ingresso ***A***) e il contenuto che viene rappresentato nell'uscita (***D***). La memoria ***m0*** di questi esempi utilizza solo due bit per gli indirizzi, i quali corrispondono alla funzione richiesta all'unità di controllo; in uscita, la stessa memoria, produce un valore a quattro bit che rappresenta, a sua volta, un indirizzo per la memoria ***m1***.

Figura u103.10. Contenuto e collegamento tra le memorie ROM ***m0*** e ***m1*** che compongono il microcodice.



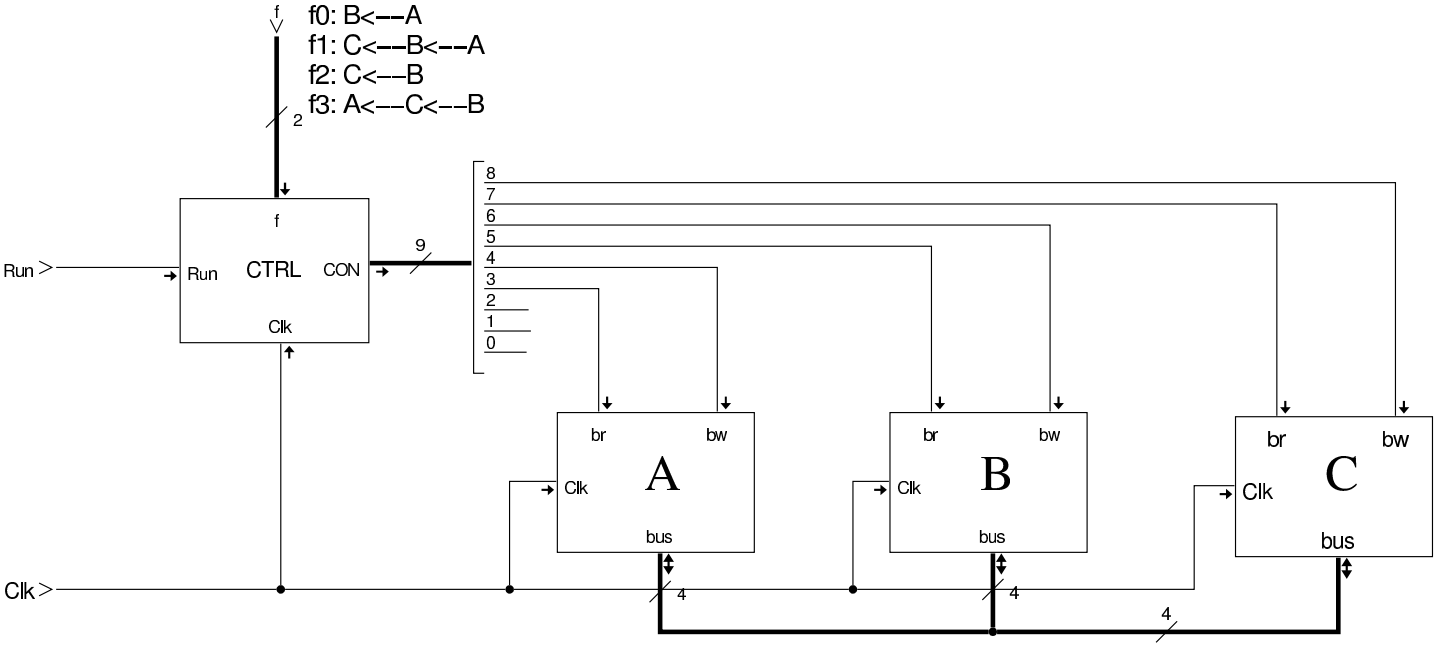
L'uscita della memoria ***m0*** viene inviata a un contatore; l'uscita del contatore serve ad accedere alla memoria ***m1*** ; l'uscita della memoria ***m1*** è ciò che serve per controllare il bus, con l'aggiunta di qualche linea per controllare la stessa unità. Per comprendere cosa succede, vengono scanditi i vari passaggi nell'elenco successivo, ipotizzando che sia richiesta la funzione f_1.

1. All'avvio l'ingresso ***Run*** è a zero, cosa che comporta l'azzeramento del contatore. Così facendo, dalla memoria ***m1*** viene selezionato l'indirizzo zero, dal quale si ottiene il valore 000000010_2. Il bit attivo, di questo valore, viene usato per richiedere al contatore il caricamento dell'indirizzo che riceve in ingresso, proveniente dalla memoria ***m0***, la quale lo produce in base alla funzione richiesta all'unità di controllo.

2. Attivando l'ingresso ***Run***, appena si presenta una **variazione negativa** del valore di ***Clk*** (l'ingresso ***Clk*** è invertito per questo) il contatore carica l'indirizzo proveniente dalla memoria ***m0*** (0011_2) e lo riproduce nell'ingresso della memoria ***m1***, dalla quale si ottiene il valore 000110000_2, corrispondenti alla richiesta di trasferire il contenuto di ***A*** in ***B***.
3. Si presenta un'altra variazione negativa del valore di ***Clk*** e il contatore viene incrementato di un'unità, selezionando da ***m1*** l'indirizzo 0100_2, con il quale la memoria ***m1*** produce il valore 011000000_2, corrispondenti alla richiesta di trasferire il contenuto di ***B*** in ***C***.
4. Si presenta un'altra variazione negativa del valore di ***Clk*** e il contatore viene incrementato di un'unità, selezionando da ***m1*** l'indirizzo 0101_2, con il quale la memoria ***m1*** produce il valore 000000111_2, corrispondenti alla richiesta di: fornire in ingresso al contatore il valore zero, caricare il contatore (con il valore zero), bloccare l'ingresso ***Clk***.

A questo punto il ciclo di esecuzione di una funzione è terminato e, per eseguirne un'altra, dopo aver fornito il valore corrispondente alla nuova funzione occorre disattivare e riattivare l'ingresso ***Run***.

Figura u103.11. Collegamento dell'unità di controllo ai componenti del bus dati: le prime tre linee del bus di controllo sono utilizzate dall'unità stessa e non servono a pilotare i moduli del bus dati.



Ciò che si scrive nelle memorie ROM che compongono l'unità di controllo può essere prodotto con strumenti che ne consentono una rappresentazione simbolica. Tkgate dispone di un compilatore che produce i file-immagine del microcodice, della mappa che consente di raggiungere le istruzioni nel microcodice attraverso la specificazione dei codici operativi, ed eventualmente anche il macrocodice, a partire da un sorgente unico. Il listato successivo mostra un sorgente compatibile con l'esempio di questa sezione.

Listato u103.12. Microcodice e codice operativo scritto per Tkgate 2.

```
map       bank[1:0]  m0;
microcode bank[8:0]  m1;
//-------------------------------------------------------
field ctrl_start[0];     // parte dall'indirizzo 0
```

```
field ctrl_load[1];      // carica l'indirizzo nel contatore.
field stop[2];           // stop clock.
field a_br[3];           // A <-- bus
field a_bw[4];           // A --> bus
field b_br[5];           // B <-- bus
field b_bw[6];           // B --> bus
field c_br[7];           // C <-- bus
field c_bw[8];           // C --> bus
//-------------------------------------------------------
op f0 {
  map f0: 0;
  +0[7:0]=0;
};
op f1 {
  map f1: 1;
  +0[7:0]=1;
};
op f2 {
  map f2: 2;
  +0[7:0]=2;
};
op f3 {
  map f3: 3;
  +0[7:0]=3;
};
//-------------------------------------------------------
begin microcode @ 0
load:
    ctrl_load;                  // CNT <-- TBL[f]

f0:
    b_br a_bw;                  // B <-- A
    ctrl_start ctrl_load stop;  // CNT <-- 0
```

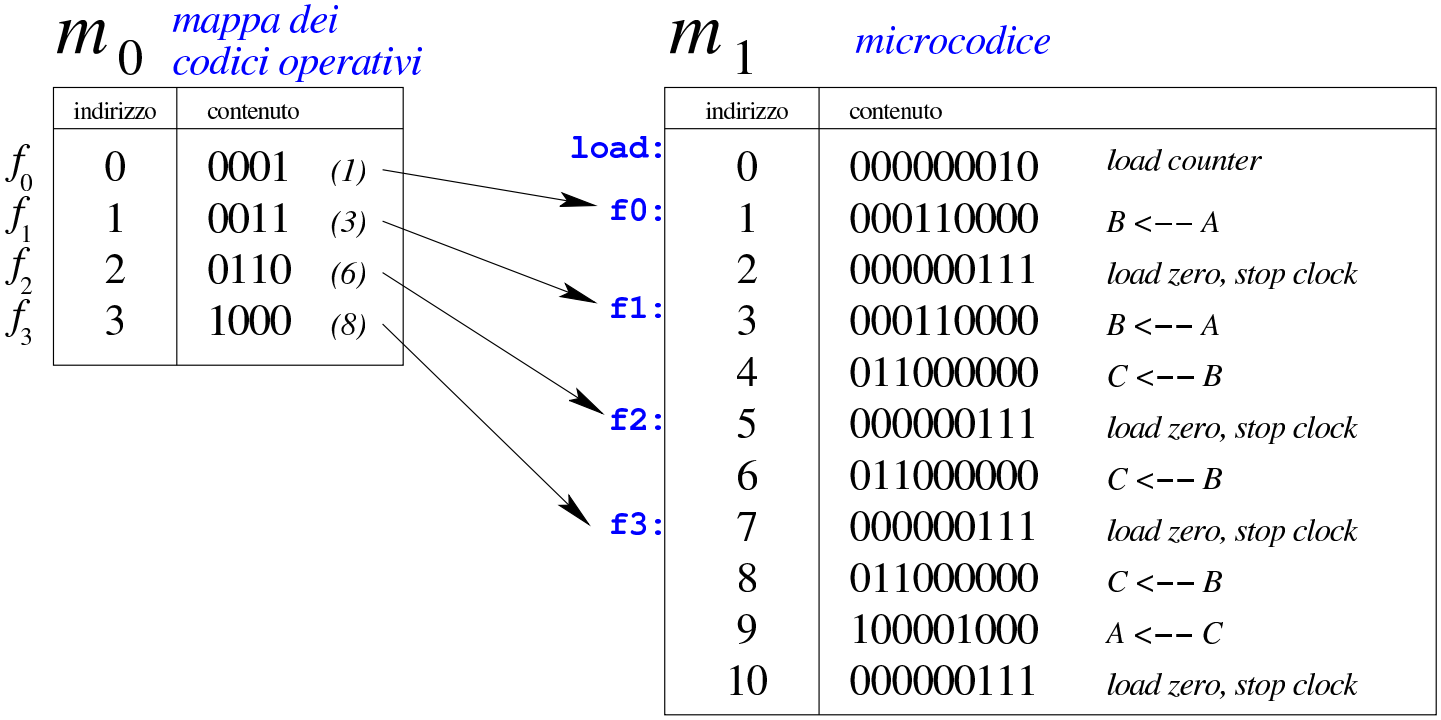
```
f1:
    b_br a_bw;                      // B <-- A
    c_br b_bw;                      // C <-- B
    ctrl_start ctrl_load stop;      // CNT <-- 0

f2:
    c_br b_bw;                      // C <-- B
    ctrl_start ctrl_load stop;      // CNT <-- 0

f3:
    c_br b_bw;                      // C <-- B
    a_br c_bw;                      // A <-- C
    ctrl_start ctrl_load stop;      // CNT <-- 0

end
```

Il contenuto della memoria che rappresenta la «mappa» è organizzato in «codici operativi». Per la precisione, il codice operativo è l'indirizzo della mappa in cui si trova, a sua volta, l'indirizzo della memoria contenente il codice operativo, da cui inizia l'esecuzione di una certa funzione. Per esempio, il codice operativo della funzione f_2 è 2, come si vede nel listato di Tkgate:

```
op f2 {
  map f2: 2;
  +0[7:0]=2;
};
```

Viene riproposta la figura che mette a confronto le due memorie, ***m0*** e ***m1***, con l'aggiunta di altri dettagli, a completamento dell'argomento.

Figura u103.14. Mappa e microcodice.



1856

Tkgate

Tkgate[1] (*http://www.tkgate.org*) è un programma per il disegno assistito (CAD) di circuiti logici, con la possibilità di utilizzare un simulatore, per il quale è possibile costruire anche dei moduli in grado di interfacciarsi con il sistema operativo ospitante.

La versione più recente di Tkgate è incompleta; tuttavia è migliore rispetto a quella ritenuta stabile. In questa sezione si annotano solo alcuni accorgimenti per poter utilizzare proficuamente la versione 2.0-b10 (beta) di Tkgate, nonostante i piccoli difetti che presenta. Va comunque osservato che la versione 2.0-b***n*** di Tkgate va compilata personalmente, dato che difficilmente si può trovare un pacchetto pronto per la propria distribuzione; per la compilazione occorre installare nel sistema i sorgenti delle librerie Tcl/Tk 8.5.

File PostScript ed EPS

I disegni realizzati attraverso Tkgate possono essere «stampati» producendo file PostScript o EPS (a seconda delle opzioni selezionate in fase di stampa). I file PostScript o EPS generati da Tkgate 2.0-b10 hanno un difetto importante che impedisce loro di essere gestiti da Ghostscript. Si osservi l'estratto seguente di codice PostScript/EPS:

«a2» 2013.11.11 --- Copyright © Daniele Giacomini -- appunti2@gmail.com http://informaticalibera.net

```
/Courier-Latin1 /Courier findfont defLatin1
/Courier-Bold-Latin1 /Courier-Bold findfont defLatin1
/Courier-Italic-Latin1 /Courier-Italic findfont defLatin1
/Courier-BoldItalic-Latin1 /Courier-BoldItalic findfont defLatin1
/Helvetica-Latin1 /Helvetica findfont defLatin1
/Helvetica-Bold-Latin1 /Helvetica-Bold findfont defLatin1
/Helvetica-Oblique-Latin1 /Helvetica-Oblique findfont defLatin1
```

Le righe evidenziate in nero provocano un errore irreversibile per Ghostscript, ma è sufficiente commentarle senza per questo impoverire il risultato tipografico del file:

```
/Courier-Latin1 /Courier findfont defLatin1
/Courier-Bold-Latin1 /Courier-Bold findfont defLatin1
%/Courier-Italic-Latin1 /Courier-Italic findfont defLatin1
%/Courier-BoldItalic-Latin1 /Courier-BoldItalic findfont defLatin1
/Helvetica-Latin1 /Helvetica findfont defLatin1
/Helvetica-Bold-Latin1 /Helvetica-Bold findfont defLatin1
/Helvetica-Oblique-Latin1 /Helvetica-Oblique findfont defLatin1
```

Per correggere questi file con l'aiuto di uno script, si può usare SED (sezione 23.5) con un comando come quello seguente:

```
$ cat file.ps↩
↪    | sed "s/^\/Courier-Italic/%\/Courier-Italic/g"↩
↪    | sed "s/^\/Courier-BoldItalic/%\/Courier-BoldItalic/g"↩
↪     > fix.ps [Invio]
```

Rappresentazione di multiplatori, demultiplatori e codificatori

«

I componenti del tipo multiplatori e simili, hanno dei terminali numerati, per sapere in che ordine sono disposti. Tkgate consente di invertire l'ordine, rispetto a quello predefinito (da sinistra a destra); tuttavia, nella rappresentazione stampata (file PostScript o EPS),

questi componenti vengono mostrati sempre come se l'ordine dei terminali fosse quello predefinito. Va osservato che questo problema esiste comunque anche nella versione stabile di Tkgate.

Per risolvere il problema della stampa, conviene annotare con dei commenti l'ordine dei terminali del multiplatore e degli altri componenti simili, poi è opportuno sopprimere il codice PostScript responsabile di questo errore.

Nel file PostScript o EPS, ogni componente ha una sua «funzione», dichiarata nel preambolo. L'estratto seguente mostra le funzioni di multiplatore e componenti analoghi, dove è stato commentato il codice responsabile della rappresentazione errata dell'ordine dei terminali:

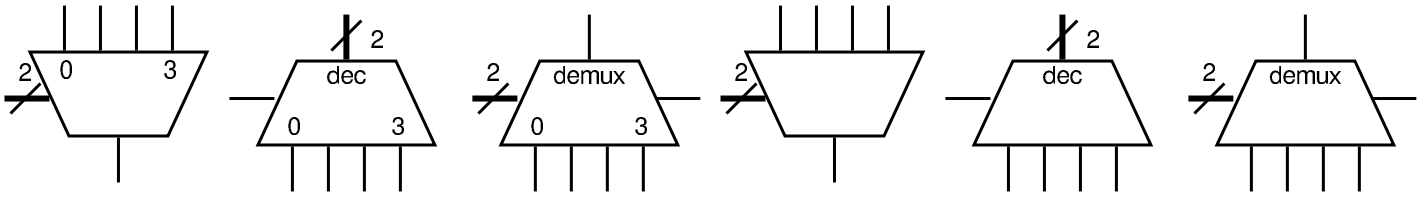
```
/mux {
    dup /mrot exch def
    startgate
    8 rfont
    -29.5 15.5 moveto
    29.5 15.5 lineto
    16.5 -12.5 lineto
    -16.5 -12.5 lineto
    closepath stroke
    dup                                 % n n
    1 add 58 exch div                   % n d1
    2 copy mul                          % n d1 dn
    mrot -90 eq mrot -180 eq or {
%       3 -1 roll 1 sub 50 string cvs exch (0) exch      % d1 (n) (0) dn
%       -29 add 7 rCT                                    % d1
%       exch -29 add 7 rCT
    } {
%       3 -1 roll 1 sub 50 string cvs exch               % d1 (n) dn
%       -29 add 7 rCT                                    % d1
%       (0) exch -29 add 7 rCT
    } ifelse
    grestore
```

```
} def
…
/demux {
    startgate
    8 rfont
    (demux) 0 5 rCT
    -16.5 12.5 moveto
    16.5 12.5 lineto
    29.5 -15.5 lineto
    -29.5 -15.5 lineto
    closepath stroke
    dup                                 % n n
    1 add 58 exch div                   % n d1
    2 copy mul                          % n d1 dn
%    3 -1 roll 1 sub 50 string cvs exch % d1 (n) dn
%    -29 add -12 rCT                          % d1
%    (0) exch -29 add -12 rCT
    grestore
} def
…
/decoder {
    startgate
    8 rfont
    (dec) 0 5 rCT
    -16.5 12.5 moveto
    16.5 12.5 lineto
    29.5 -15.5 lineto
    -29.5 -15.5 lineto
    closepath stroke
    dup                                 % n n
    1 add 58 exch div                   % n d1
    2 copy mul                          % n d1 dn
%    3 -1 roll 1 sub 50 string cvs exch % d1 (n) dn
%    -29 add -12 rCT                          % d1
%    (0) exch -29 add -12 rCT
    grestore
} def
```

Il codice PostScript/EPS non può essere corretto diversamente, per-

ché queste funzioni non ricevono alcun parametro che informi loro dell'ordine in cui vanno rappresentati i terminali.

Figura u104.4. Multiplatore, decodificatore e demultiplatore: a sinistra nella loro rappresentazione originale che, però, potrebbe essere errata, a destra togliendo i numeri d'ordine dei terminali. Se si tolgono i numeri d'ordine, vanno aggiunte delle indicazioni attraverso commenti nel disegno di Tkgate.



Clock

Tra i moduli già pronti che accompagnano Tkgate, ne esistono due che servono a generare un segnale di *clock*, ovvero un'onda quadra, strutturata in qualche modo. Si tratta precisamente della libreria di moduli denominata '**`timer`**', all'interno della quale sono disponibili i moduli '**`ONESHOT`**' e '**`OSCILLATOR`**'. Questi moduli sono scritti usando codice Verilog di Tkgate.

I due moduli prevedono un parametro, denominato ***HZ***, con il quale si presume di inserire una frequenza espressa in Hz, ma non è così: osservando il codice, si può notare che si tratta invece della durata che dovrebbe avere l'onda o l'impulso, espressa in millisecondi (ms). Tuttavia, pur contando questo fatto, la frequenza che si ottiene nel simulatore è molto differente, perché durante la simulazione non c'è alcuna connessione con l'orologio del sistema operativo ospitante, al quale si rimettono invece i moduli '**`ONESHOT`**' e '**`OSCILLATOR`**'. In pratica, questi due moduli vanno usati con il valore della «frequenza» predefinita, osservando nel simulatore se la

frequenza effettiva può andare bene per i propri fini. Va anche osservato che è inutile tentare di modificare il codice di questi moduli, in modo da produrre effettivamente la frequenza desiderata, proprio perché il simulatore non riuscirebbe a farne buon uso.

Per poter disporre di un generatore di frequenza che, in qualche modo, possa essere controllato in relazione al metro usato nella simulazione, occorre costruire qualcosa che si basi sui ritardi di propagazione. Per prima cosa serve un generatore di un impulso; precisamente serve qualcosa che parta producendo un valore azzerato, per poi passare ad attivarsi dopo un certo tempo, mantenendosi attivo per tutto il tempo successivo. Per produrre questo componente virtuale, occorre predisporre un modulo scritto direttamente secondo il linguaggio Verilog di Tkgate:

```
module one_up #(.W(1000)) (Z);
  output Z;
  reg Z;

  initial
    begin
      Z = 1'b0;
      $tkg$wait(W);
      Z = 1'b1;
    end

endmodule
```

Si tratta di un modulo privo di entrate e avente una sola uscita: all'avvio l'uscita si presenta a zero e ci rimane per il tempo specificato dal parametro ***W***, espresso in millisecondi, quindi passa a uno e ci rimane fino alla fine. Il tempo di pausa iniziale serve a consentire ai componenti di inizializzarsi; il tempo predefinito di 1000 ms, cor-

rispondente a 1 s, viene visto durante la simulazione come un tempo molto più breve, ma generalmente sufficiente: nel caso potrebbe essere aumentato.

Disponendo del modulo di inizializzazione, si può realizzare un circuito che produca un'oscillazione sfruttando il ritardo di propagazione dei componenti; poi, questa oscillazione può essere suddivisa convenientemente in base alle esigenze. Nel capitolo xxvi si mostra un oscillatore realizzato in questo modo.

[1] **Tkgate** GNU GPL

1864

Riferimenti

- *Tkgate*, http://www.tkgate.org
- Mario Italiani, Giuseppe Serazzi, *Elementi di informatica*, ETAS libri, 1973, ISBN 8845303632
- Albert Paul Malvino, Jerard A. Brown, *Digital Computer Electronics*, Glencoe/Mcgraw-Hill http://www.amazon.it/Digital-Computer-Electronics-Albert-Malvino/dp/0028005945
- Stephen Brown, Zvonko Vranesic, *Fundamentals of Digital Logic with Verilog Design*, Mcgraw-Hill http://www.amazon.com/Fundamentals-Mcgraw-Hill-Electrical-Computer-Engineering/dp/0072823151 ; *Flip-Flops, Registers, Counters, and a Simple Processor*, http://highered.mcgraw-hill.com/sites/dl/free/0072823151/56549/vra23151_ch07.pdf
- Luigi Zeni, *Elettronica dei sistemi digitali*, http://www.dii.unina2.it/Utenti/lzeni/Elettronica%20de20i%20Si20stemi%20Di20gitali/ESD-CircuitiCombinatori.pdf , http://www.dii.unina2.it/Utenti/lzeni/Elettronica%20de20i%20Si20stemi%20Di20gitali/ESD-CircuitiSequenziali.pdf
- CL GATE aspire, *Combinational & Sequential Circuits*, http://media.careerlauncher.com.s3.amazonaws.com/gate/material/2.pdf
- *CSC 4210/6210 – Computer Architecture*, http://www.cs.gsu.edu/~cscagb/csc4210/
- Olivier Carton, *Circuits et architecture des ordinateurs*, http://www.liafa.jussieu.fr/~carton/Enseignement/Architecture/archi.pdf

- Tony R. Kuphaldt, *Lessons In Electric Circuits*, *http://www.faqs.org/docs/electric/*
- *Building a Digital Computer*, *http://artematrix.org/Projects/TTL.processor/processor.design.htm*