

Il progetto dei circuiti logici ii parte

M. Rebaudengo, M. Sonza Reorda, L. Sterpone

Politecnico di Torino
Dip. di Automatica e Informatica



Progetto a livello di registri

Caratteristiche:

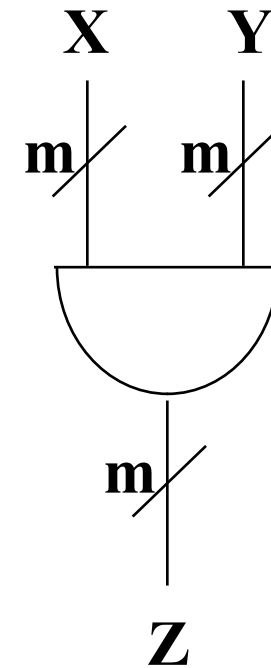
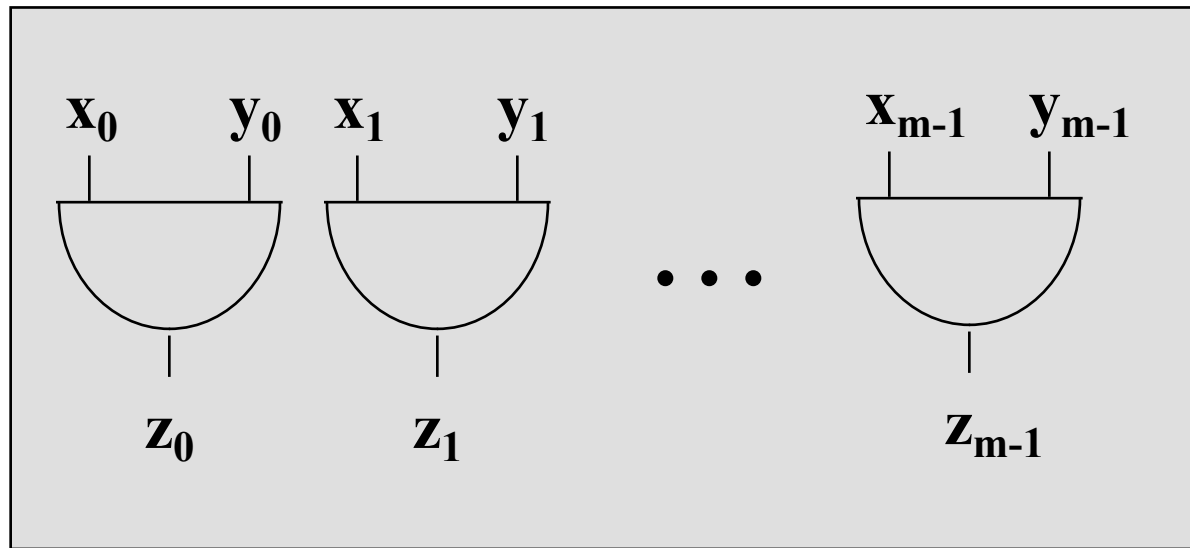
- l'unità di dato manipolata è la parola
- i componenti usati nel progetto sono blocchi (combinatori o sequenziali) per la memorizzazione o la trasformazione di parole
- in alcuni casi può essere utile adottare un'algebra booleana operante su vettori di bit; le funzioni utilizzate sono quindi

$$x: (B^m)^n \rightarrow B^m$$

Componenti

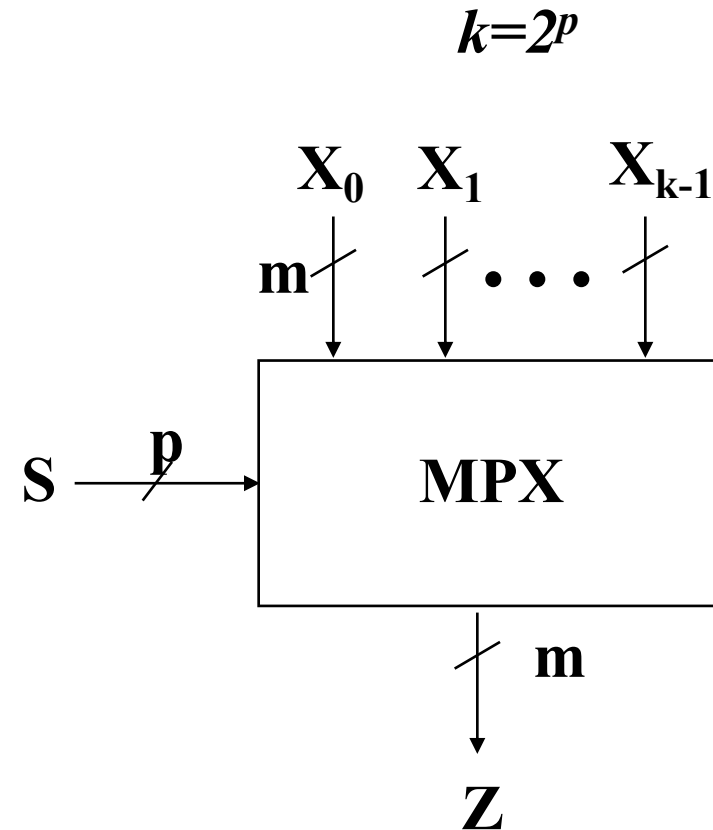
- **Componenti combinatori**
 - **Porte logiche operanti su parole**
 - **Multiplexer**
 - **Decodificatori e codificatori**
 - **Moduli aritmetici (ALU, sommatori, ecc.)**
- **Componenti sequenziali**
 - **Registri**
 - **Contatori**
 - **FPGA**
 - **Bus**
 - **Memorie.**

Porte logiche operanti su parole



Multiplexer

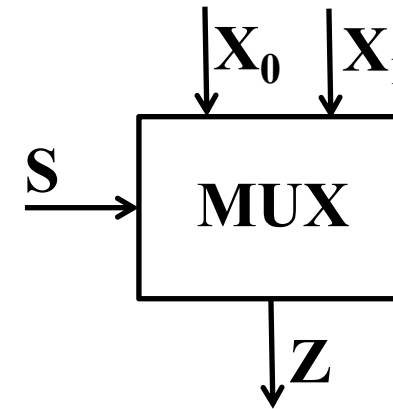
Servono per connettere una fra k fonti di dato X_i a una destinazione, a seconda del valore di p segnali di selezione S , dove



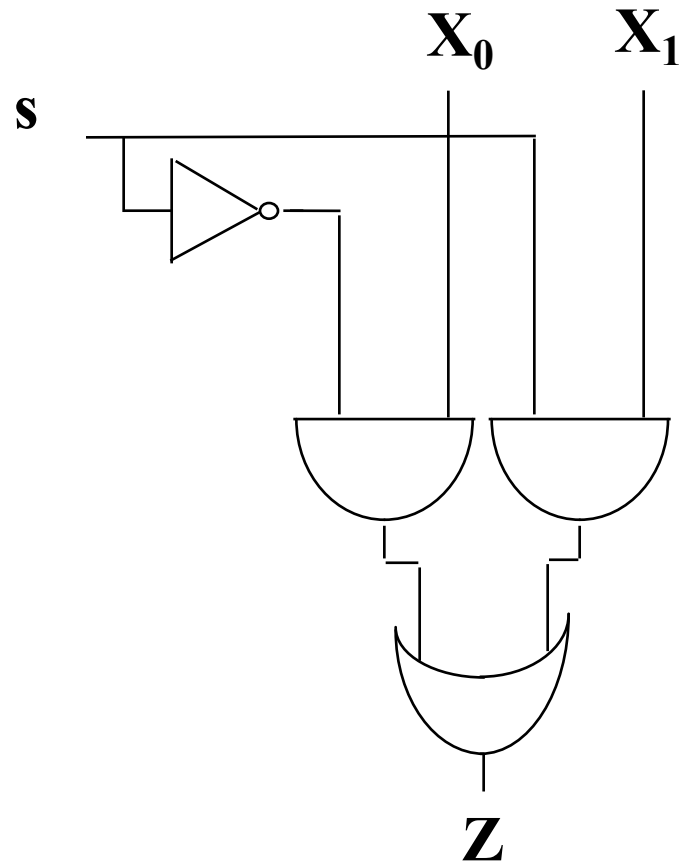
Multiplexer: funzione

La tavola della verità di un multiplexer 2×1 con ingressi di dato X_0 e X_1 , ingresso di controllo S ed uscita Z è la seguente:

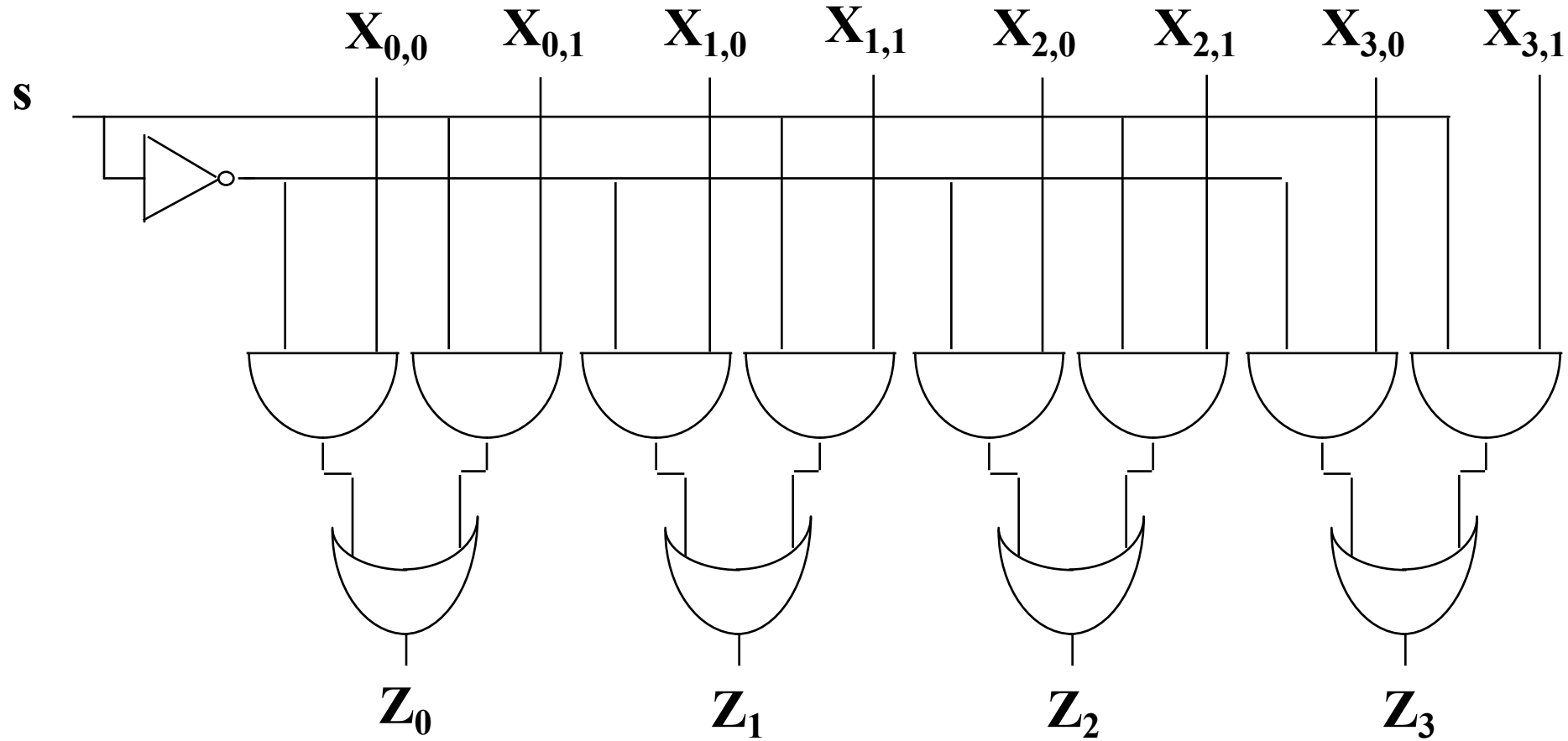
S	Z
0	X_0
1	X_1



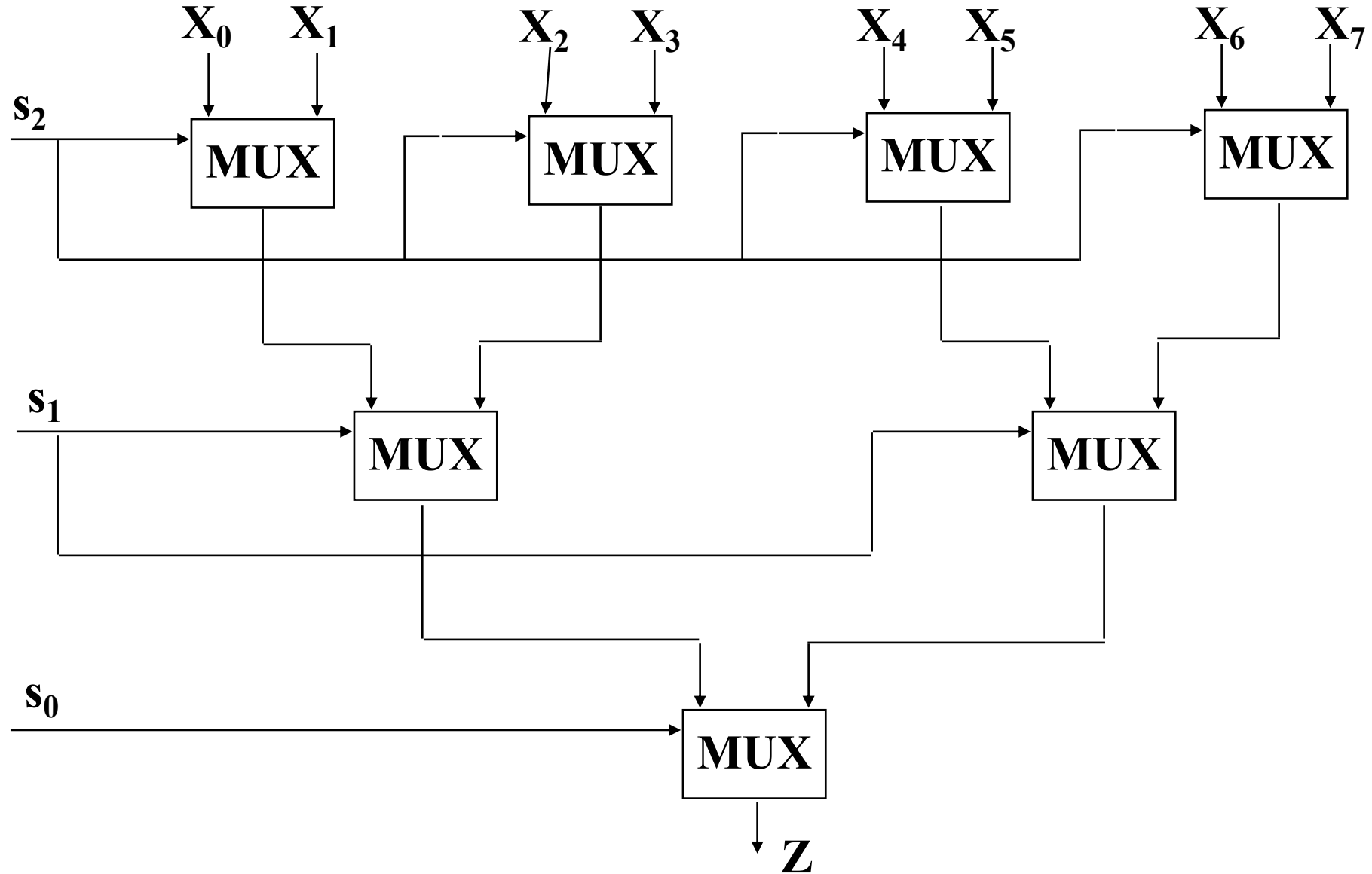
Multiplexer: 2×1 , 1 bit



Multiplexer: $2 \times 1, 4$ bit



Multiplexer: 8×1 , 1 bit

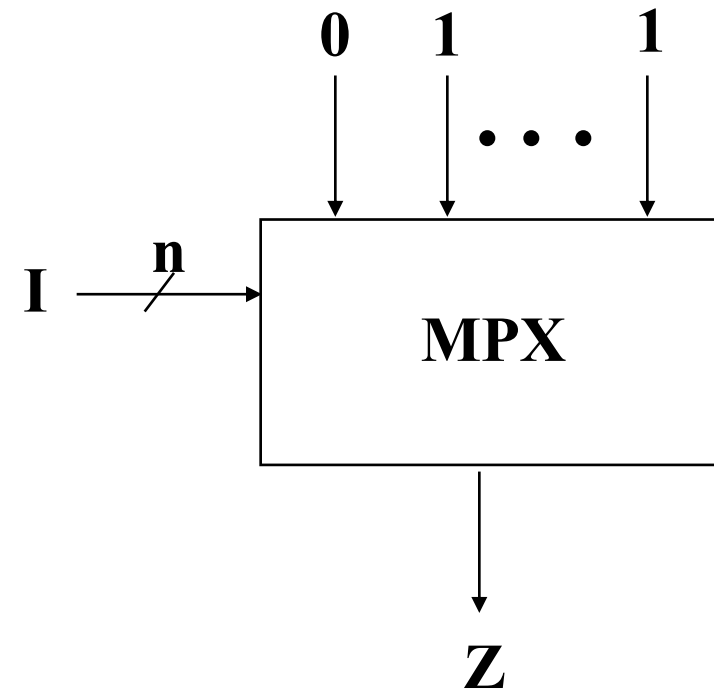


Multiplexer: uso nella sintesi

Permettono l'implementazione di una qualsiasi funzione combinatoria.

$$Z=f(i_0,i_1,...,i_{n-1})$$

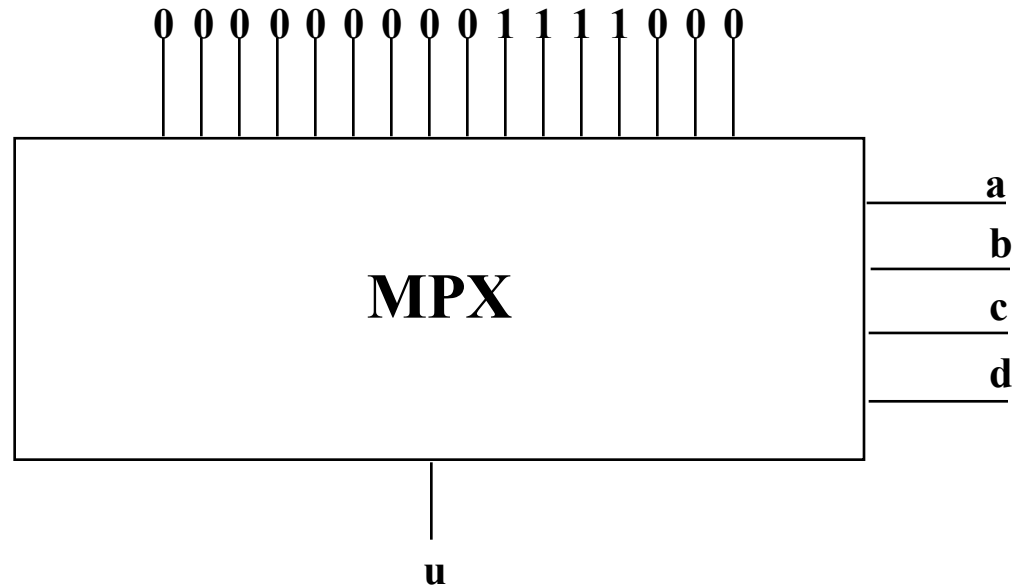
Ad ogni ingresso di dato
corrisponde un minterm.



Sintesi tramite multiplexer: esempio

Si vuole progettare un circuito con 4 ingressi a, b, c, d la cui uscita u valga 1 quando $8 < (abcd) < 13$.

Una possibile implementazione è



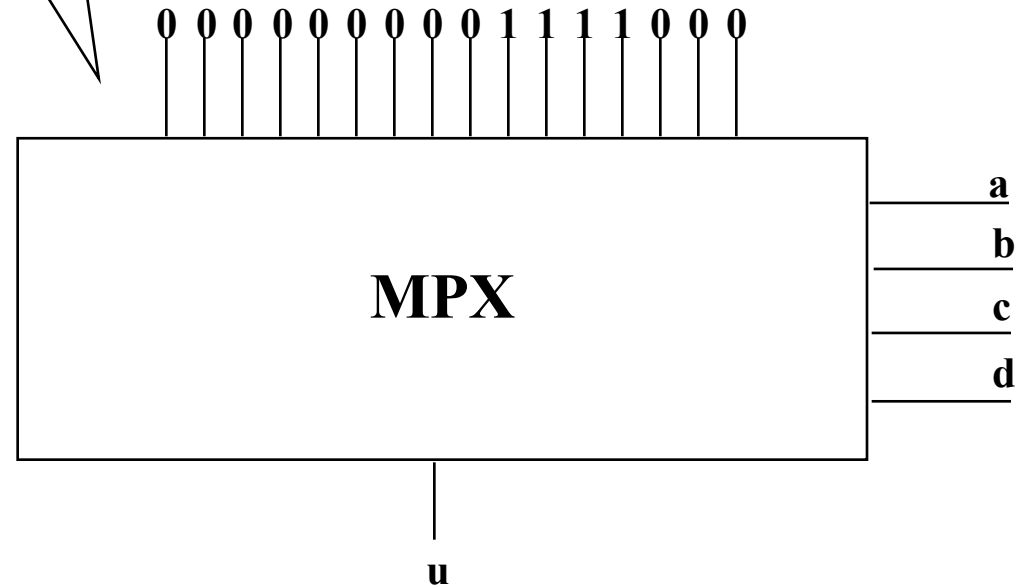
Sintesi tramite multiplexer: esempio

Questo MPX può essere
realizzato con un albero di 15
MPX da 2 a 1.

Il costo totale in termini di
porte logiche è quindi $15 \times 4 =$
60 porte logiche.

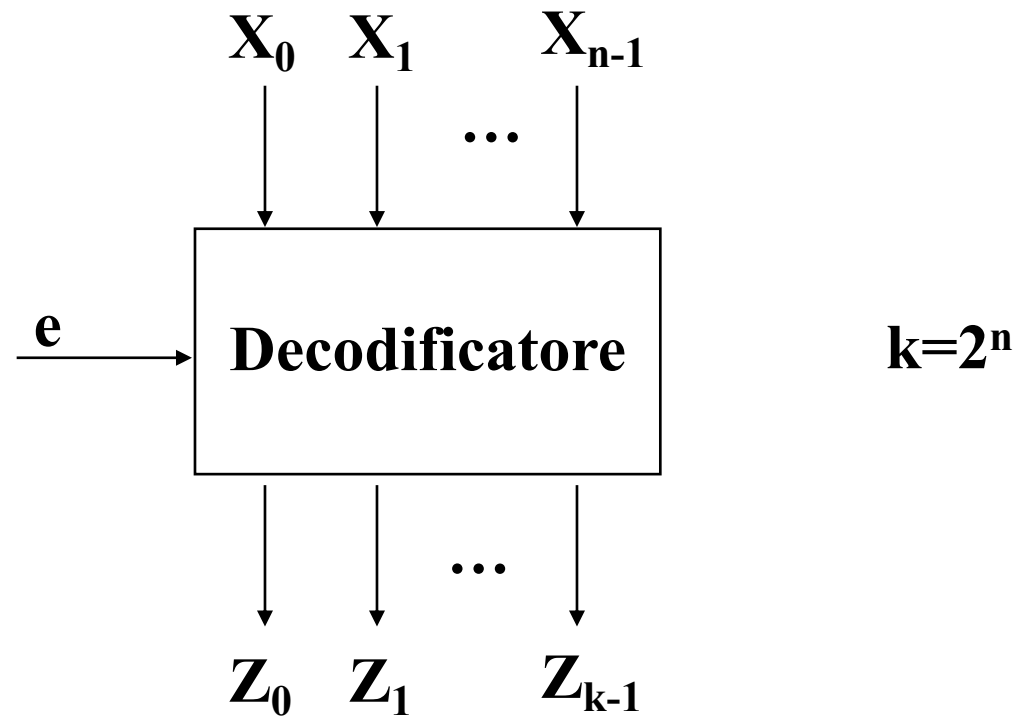
Si vuole
quando

Una possibile implementazione è



Decodificatori

Hanno n linee di ingresso e 2^n linee di uscita; di queste è attiva solo quella di indice corrispondente al valore applicato in ingresso.

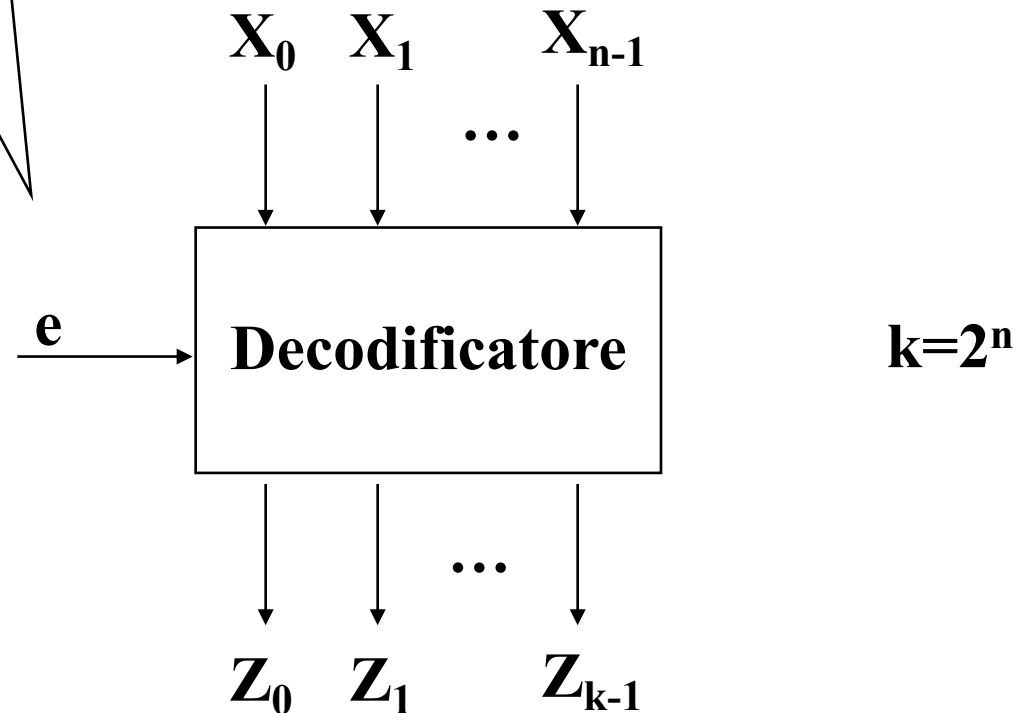


Decodificatori

Hanno
corrispo

In alcuni casi esiste un
segnale di enable; quando
non è attivo nessuna uscita
è attiva (qualunque sia il
valore degli altri ingressi).

linee di uscita; di queste è attiva solo quella di indice
to in ingresso.

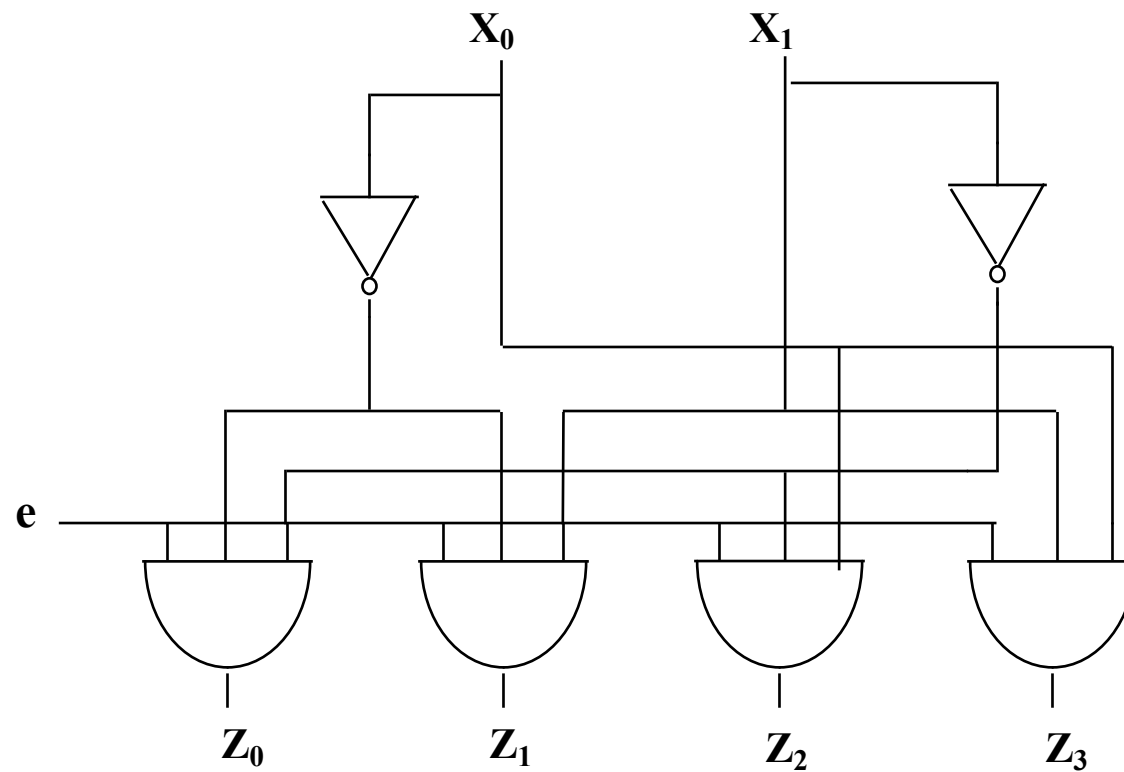


Decodificatori

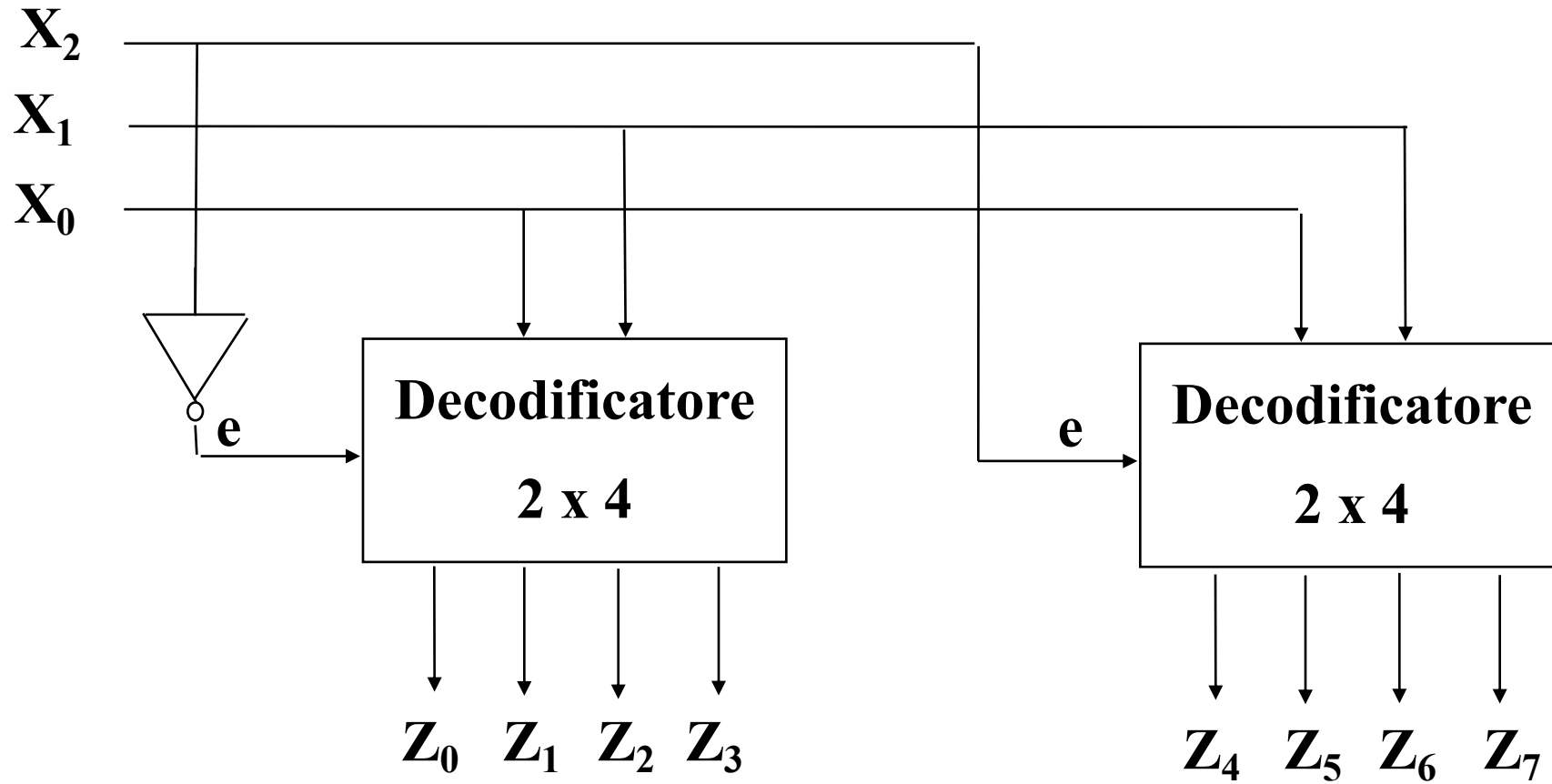
La tavola della verità di un decodificatore $2 \rightarrow 4$ è la seguente:

X	e	Z
00	1	0001
01	1	0010
10	1	0100
11	1	1000
-	0	0000

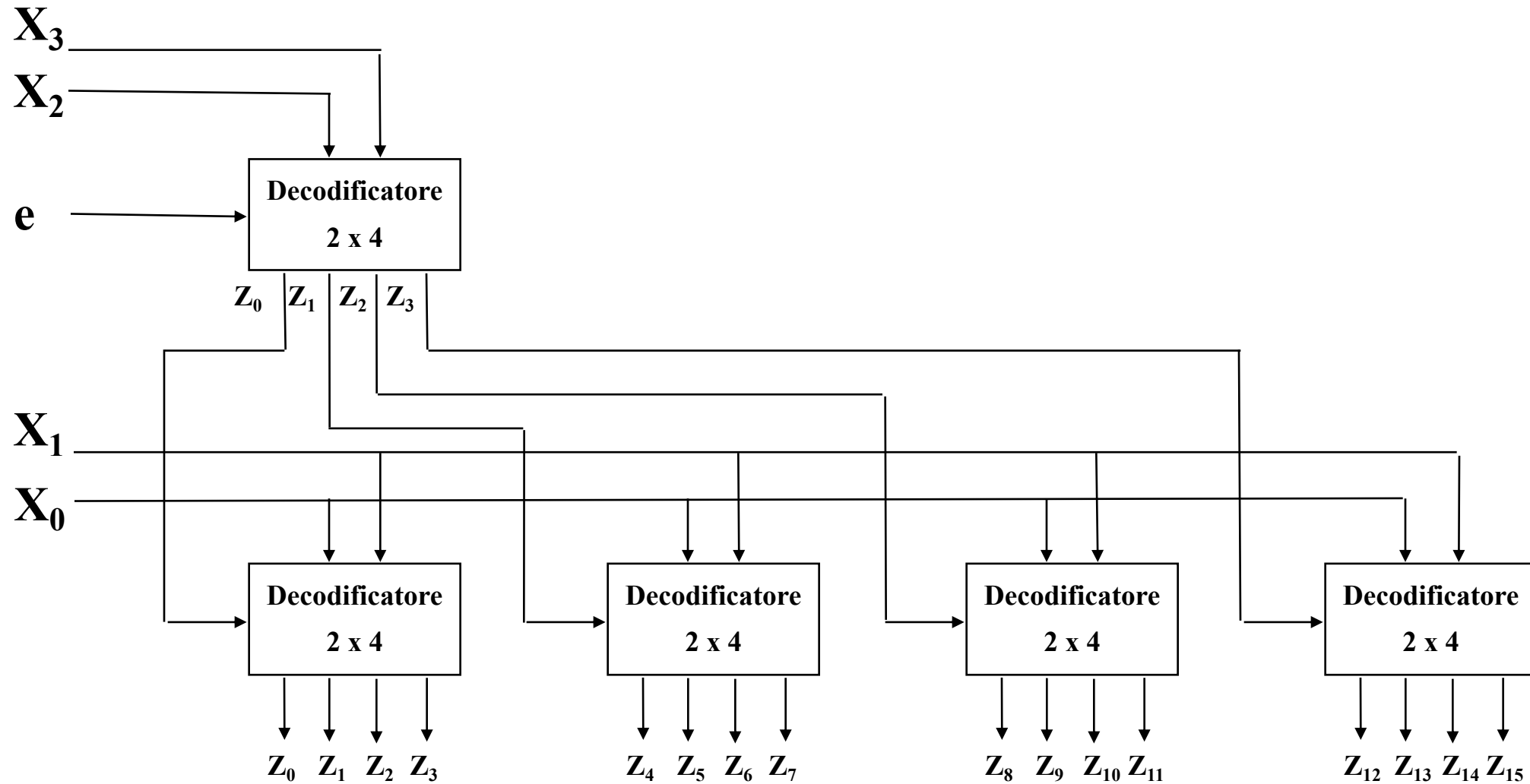
Decodificatore 2→4



Decodificatore 3 → 8



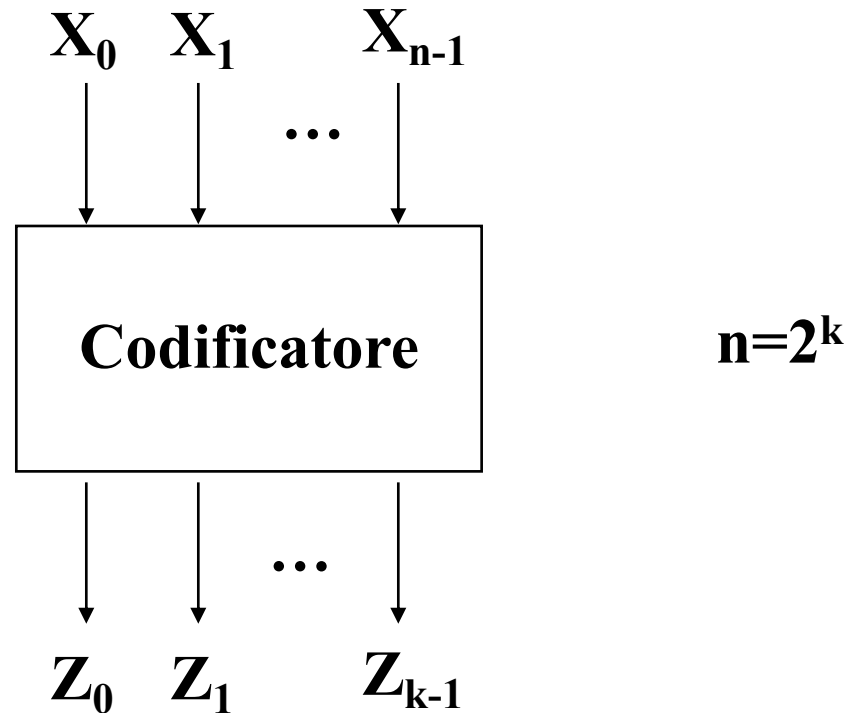
Decodificatore 4 → 16



Codificatori

Hanno 2^k linee di ingresso e k linee di uscita.

Sulle linee di uscita compare (codificato) il valore corrispondente all'indice della linea di ingresso attiva.



Codificatori: funzione

La tavola della verità di un codificatore $4 \rightarrow 2$ è la seguente:

X	Z
0001	00
0010	01
0100	10
1000	11

Codificatori prioritari

Se più di una linea di ingresso è attiva il risultato può essere scorretto, oppure può apparire il codice della linea attiva con priorità maggiore (*priority encoder*).

Codificatori prioritari: funzione

La tavola della verità di un codificatore prioritario 4→2 è la seguente:

X	Z
0001	00
0010	01
0011	01
0100	10
0101	10
0110	10
0111	10
1 - - -	11

ri prioritari: funzione

In alcuni casi si aggiungono due ulteriori segnali:

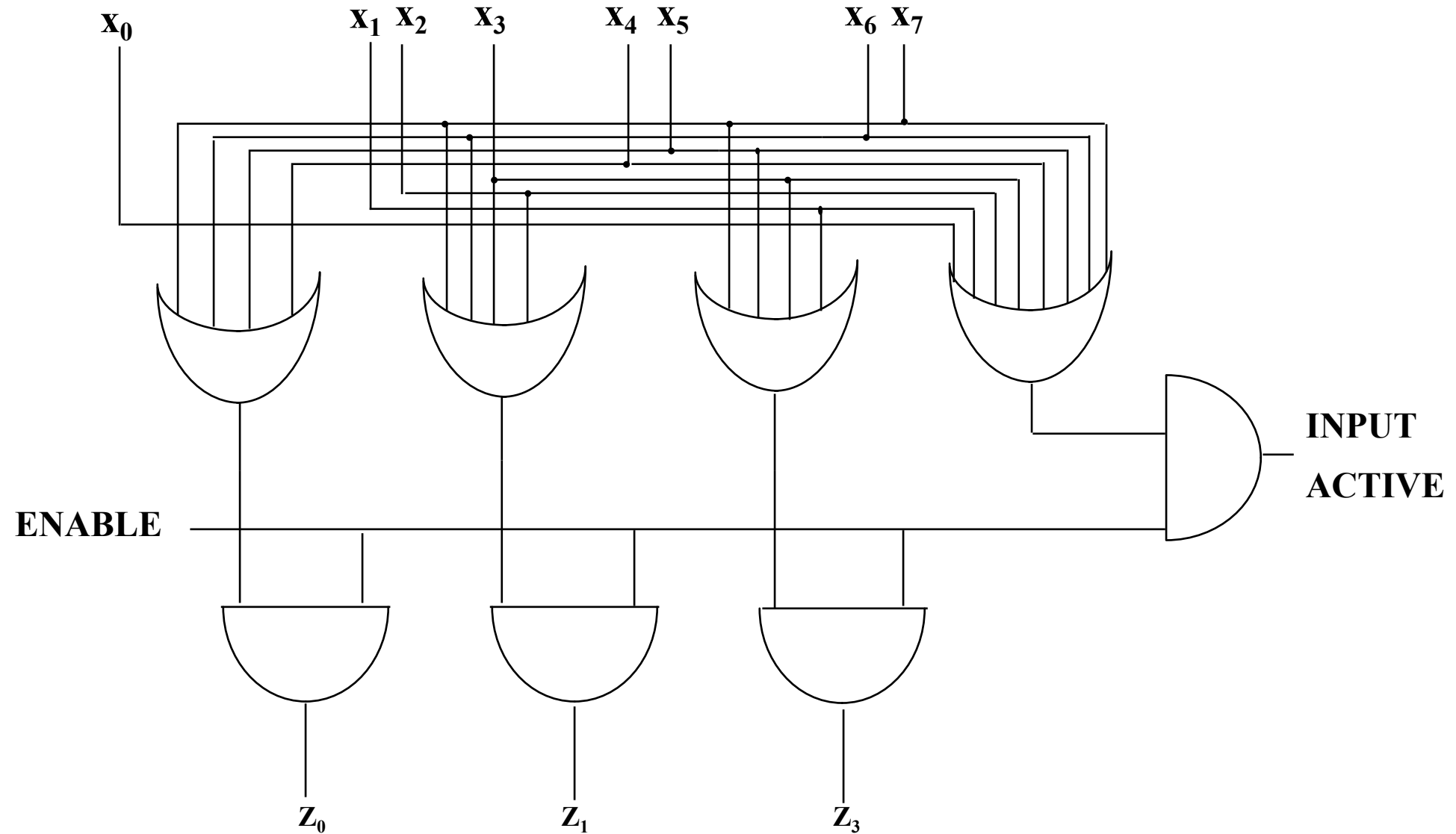
- Un segnale *enable*: quando è disattivo tutte le uscite sono 0
- Un segnale *input active*: va a 1 quando enable è attivo e almeno una linea è attiva

La ta

codificatore prioritario 4→2 è la seguente:

X	Z
0001	00
0010	01
0011	01
0100	10
0101	10
0110	10
0111	10
1 - - -	11

Codificatore 8→3 con enable



Moduli aritmetici

Possono avere complessità variabile a seconda di

- **tipo di dati supportati (interi, interi con segno, decimali)**
- **tipo di operazioni supportate (somma, sottrazione, moltiplicazione, divisione, operazioni trigonometriche)**
- **velocità (soluzioni combinatorie o sequenziali).**

Sommatori

Possono essere realizzati seguendo tre soluzioni alternative:

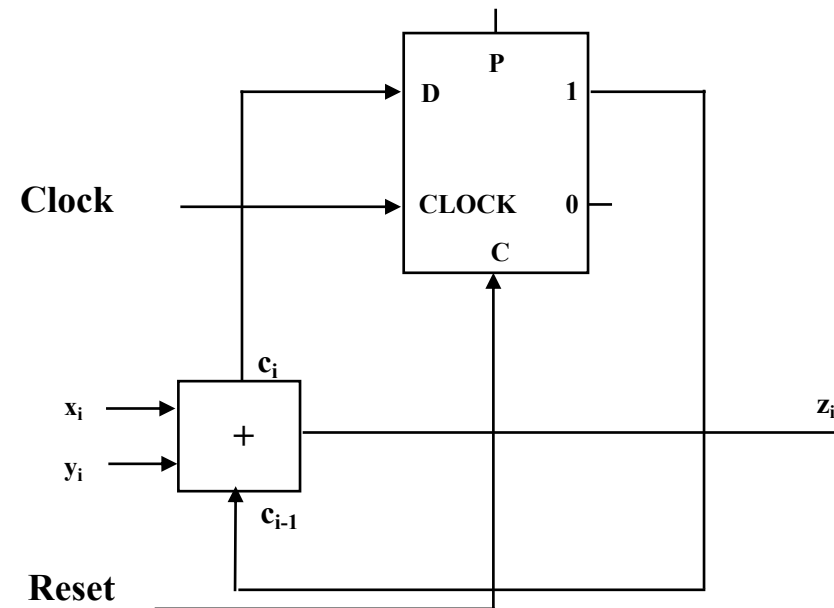
- **sommatori seriali**
- **sommatori combinatori**
- **sommatori combinatori modulari.**

Sommatore seriale

Il Flip Flop memorizza il Carry di un bit e lo riporta sul bit successivo durante il successivo periodo di clock.

All'inizio il FF deve venire azzerato.

È la soluzione che richiede in termini di porte logiche il costo minimo, indipendente da n (dimensione degli operandi); in compenso rappresenta la soluzione con il massimo tempo di risposta (pari a n periodi di clock).



Sommatore combinatorio

La soluzione più vantaggiosa in termini di tempo richiesto è quella rappresentata da un circuito a 2 livelli progettato ad hoc per sommare 2 numeri su n bit e produrre $n+1$ bit di uscita.

Un simile circuito può essere sintetizzato a partire dalla tavola di verità, oppure dall'espressione booleana corrispondente alla funzione somma.

Tale soluzione

- è quella ottima in termini di costo HW e di velocità**
- richiede di eseguire il progetto per ogni specifico valore di n .**

Sommatore combinatorio modulare

Più frequentemente si segue un approccio modulare:

- si suddividono i 2 numeri X e Y da sommare in bit (X_i, Y_i)
- si sommano tra loro le coppie X_i, Y_i , partendo dai bit meno significativi
- si combinano i risultati tenendo conto dei riporti.

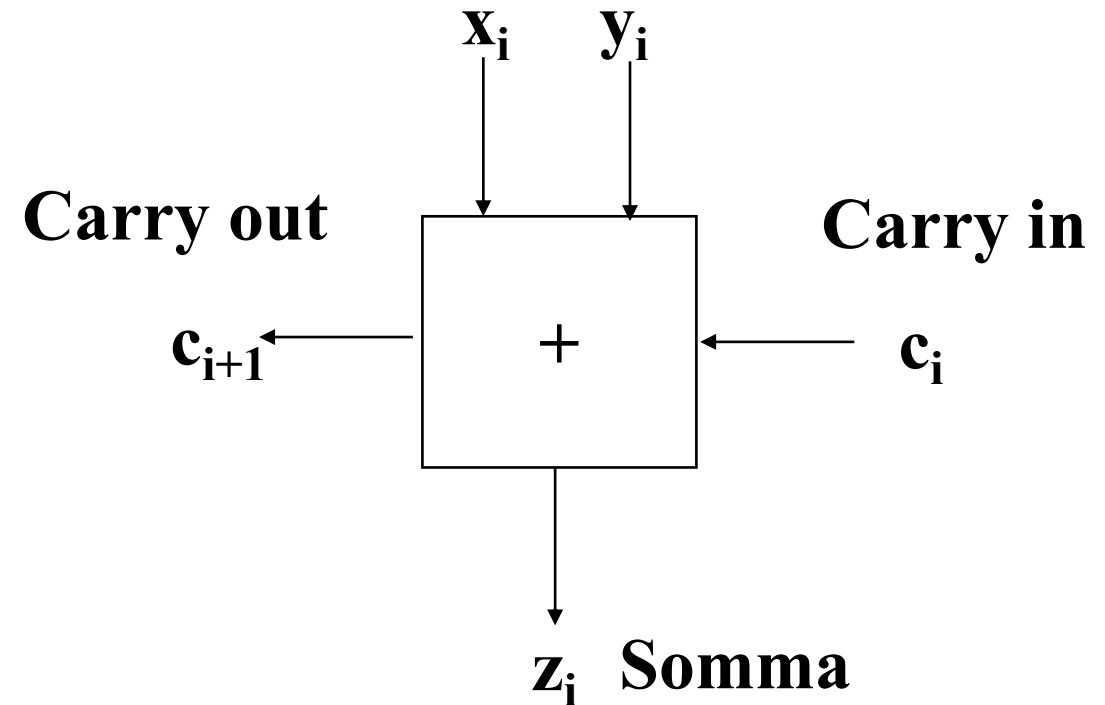
L'approccio modulare è basato su un circuito elementare denominato *full-adder*.

Full-Adder

Calcola la somma di 2 bit x_i , y_i e di un carry in ingresso c_i producendo un bit di risultato z_i ed un bit di carry c_{i+1} :

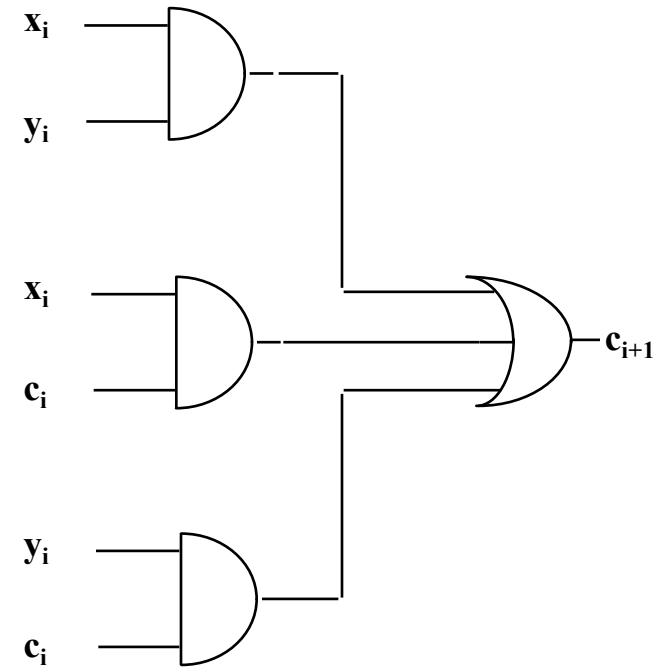
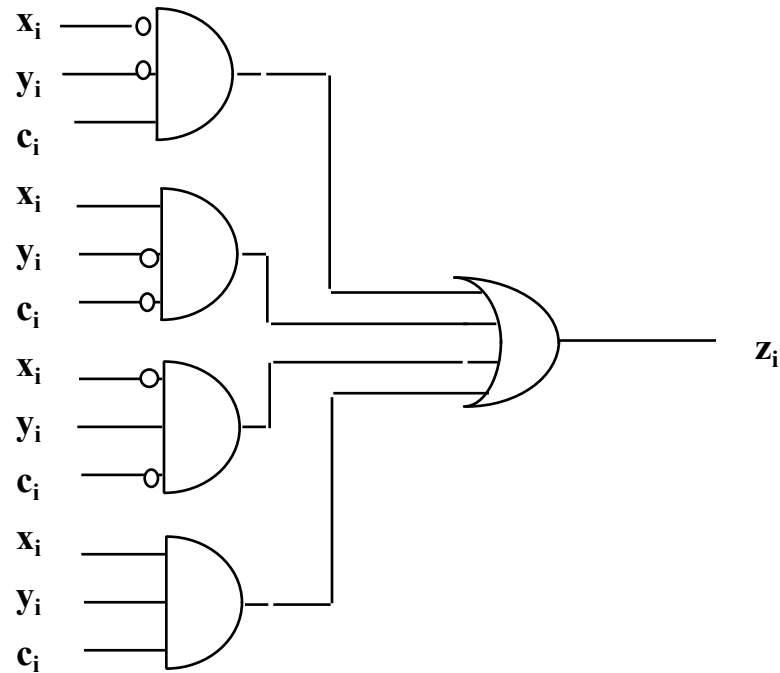
$$z_i = x_i \oplus y_i \oplus c_i$$
$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

A	B	Carry-In	Sum	Carry-Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Full-Adder:

possibile implementazione



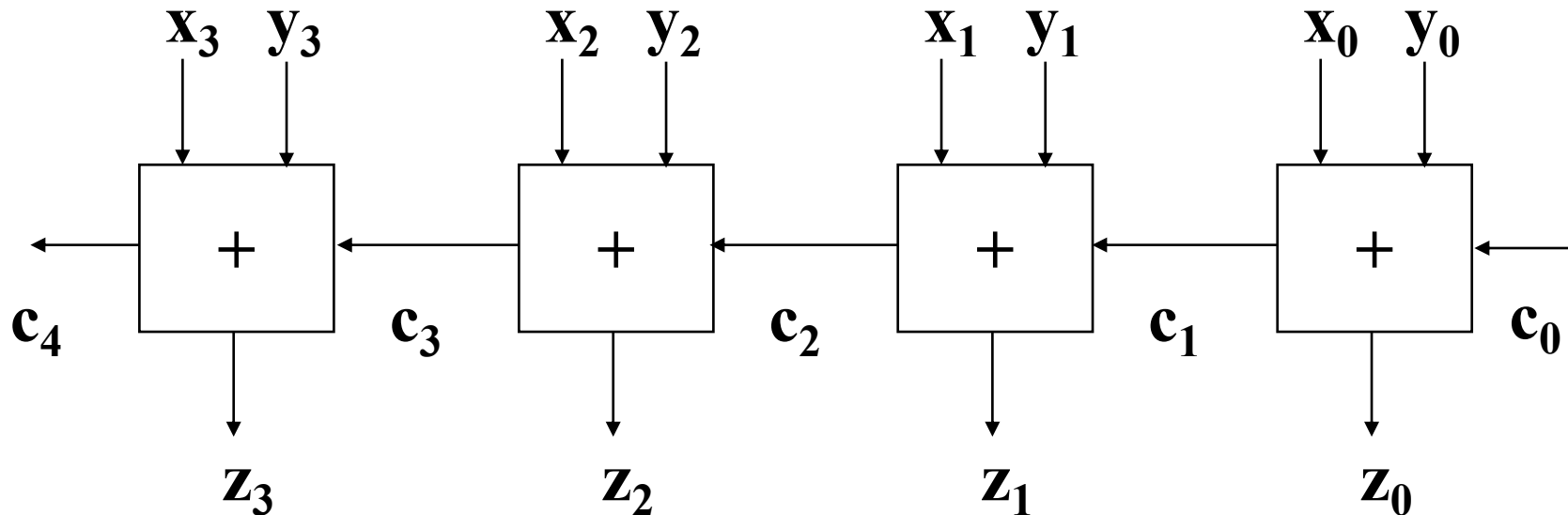
Ripple Carry Adder

Somma 2 numeri su n bit utilizzando una logica esclusivamente combinatoria.

È costruito connettendo in cascata n full-adder.

Il tempo richiesto per la generazione dell'ultimo carry è pari a nd , ove d è il ritardo del singolo modulo.

Il costo in termini di hardware è proporzionale a n .



Sommatore con carry-lookahead

Rappresenta ancora una soluzione combinatoria modulare, alternativa a quella basata sul ripple carry adder.

Permette di ridurre il ritardo nella generazione del risultato, dovuto al fatto che ogni modulo deve attendere il carry generato dal modulo precedente.

Il *Carry-Lookahead Generator* è un circuito in grado di generare il bit di carry di ogni modulo sulla base dei segnali che gli vengono in parallelo da tutti i moduli.

Questi sono una versione modificata di full-adder in cui vengono generati 2 segnali particolari:

$$g_i = x_i y_i$$

$$p_i = x_i + y_i$$

Carry-Lookahead Generator

Il carry out del singolo modulo è dato da

$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i = g_i + p_i c_i$$

Analogamente

$$c_i = g_{i-1} + p_{i-1} c_{i-1}$$

Sostituendo

$$c_{i+1} = g_i + p_i g_{i-1} + p_i p_{i-1} c_{i-1}$$

Ad esempio in un sommatore con carry-lookahead da 4 bit si ha:

$$c_1 = g_0 + p_0 c_{in}$$

$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_{in}$$

$$c_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_{in}$$

$$c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_{in}$$

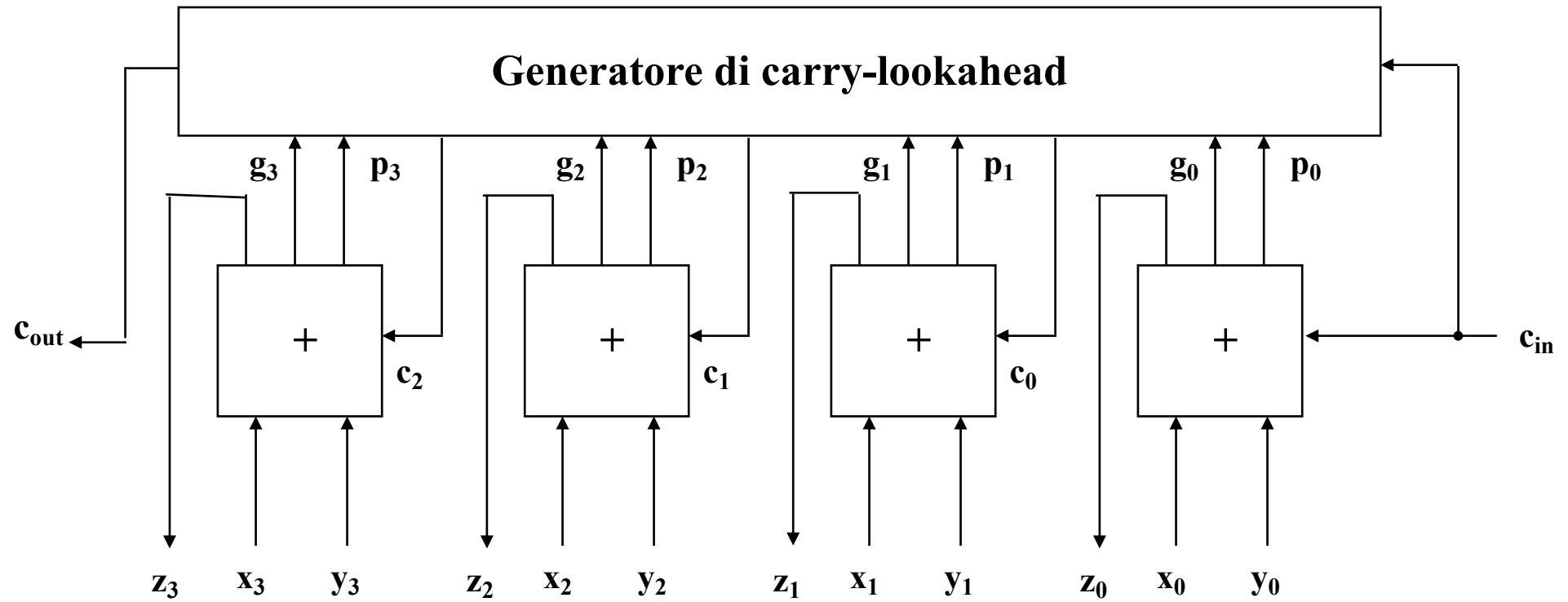
Significato di g_i e p_i

I due coefficienti g_i e p_i derivano il loro nome dal fatto che permettono

- la generazione (g_i)**
- la propagazione (p_i)**

dei segnali di carry relativi ai vari moduli.

Implementazione



Vantaggi

Detto d il ritardo introdotto da un circuito a 2 livelli, il ritardo di un sommatore con carry-lookahead è pari a $3d$.

Svantaggi

Il costo in termini hardware cresce in maniera esponenziale con n .

Per questo il metodo è in genere inapplicabile per valori di n elevati.

Approccio misto

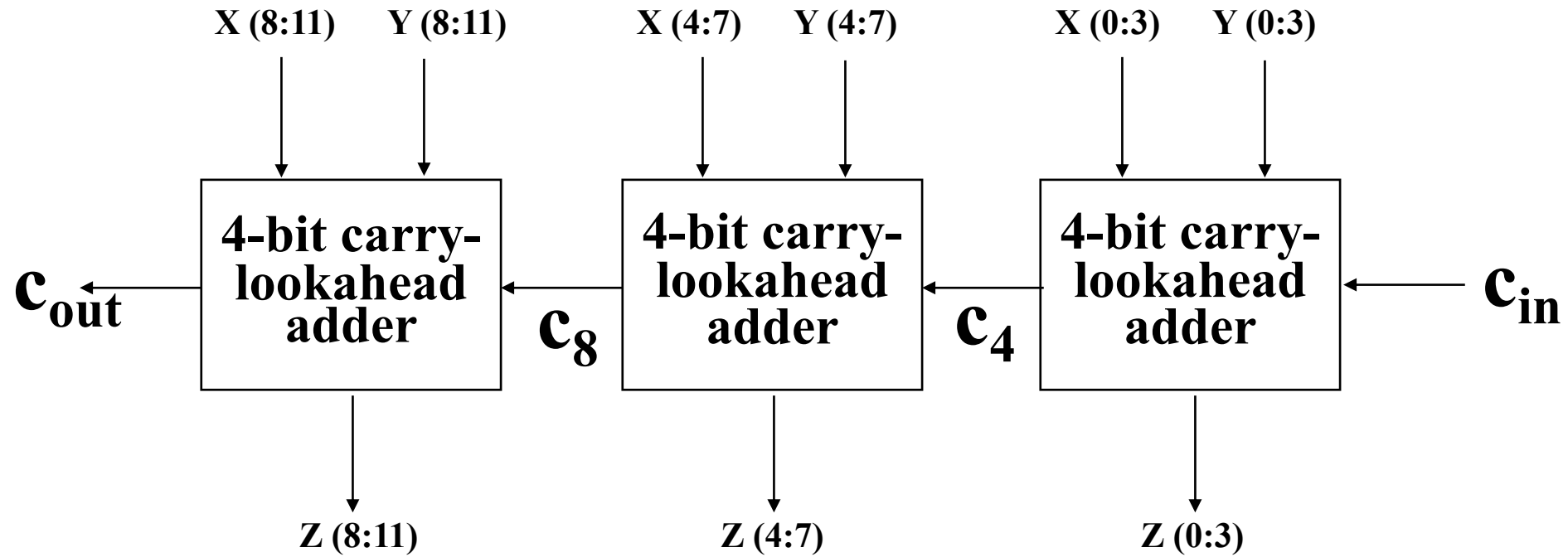
Può essere utilizzato per sommare numeri su n bit, con n grande.

Vengono utilizzati $m=n/k$ carry-lookahead adder, ognuno in grado di sommare un gruppi di k bit, e un ripple-adder per riportare i carry da un carry-lookahead adder all'altro.

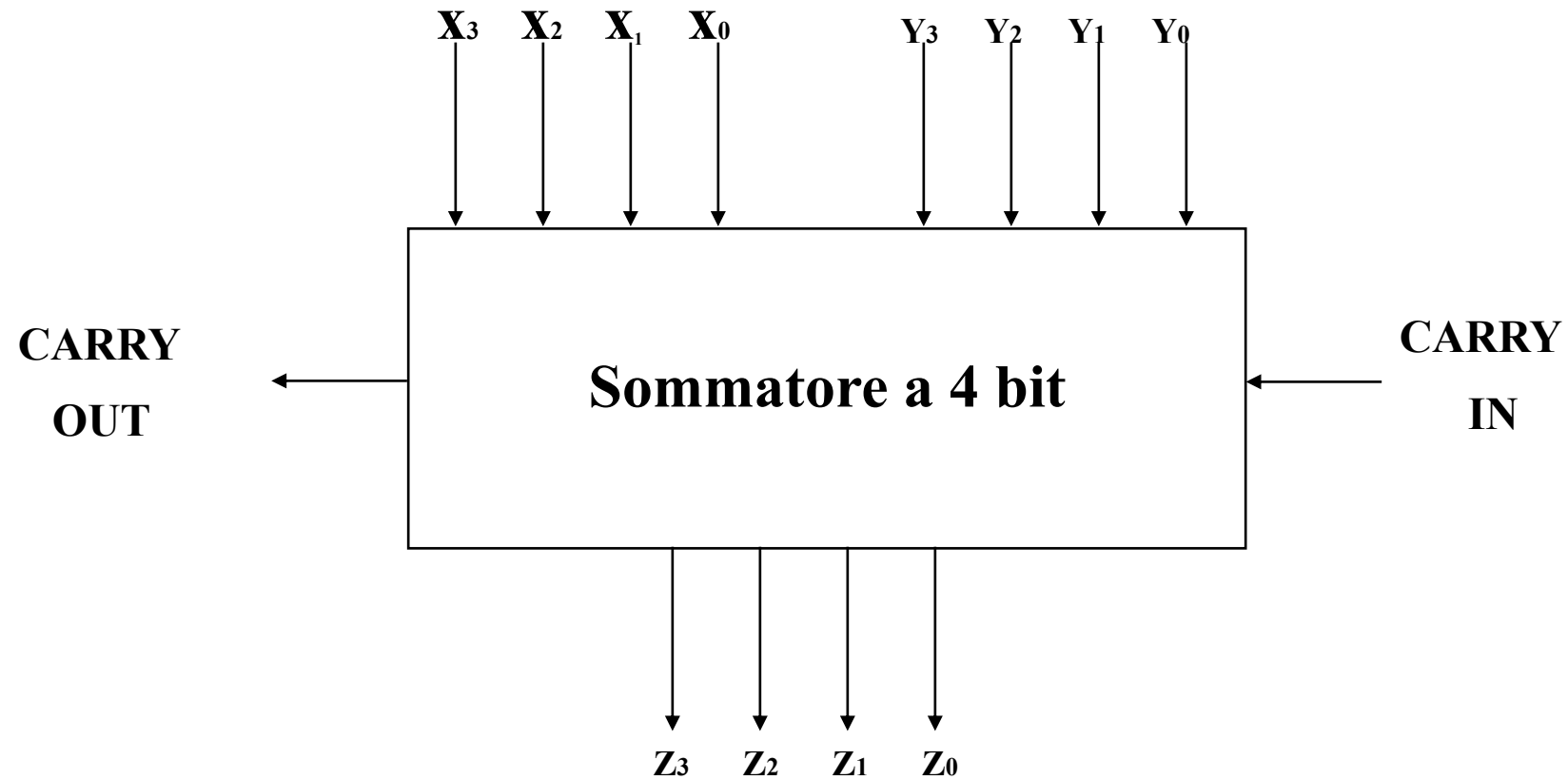
Se ci sono m carry-lookahead adder il ritardo complessivo sarà $m \times 3 \times d$.

Ad esempio se $n=12$, e $m=3$, il ritardo è pari a $9d$ con l'approccio misto, mentre è $12d$ con un ripple-carry-adder.

Implementazione (esempio)

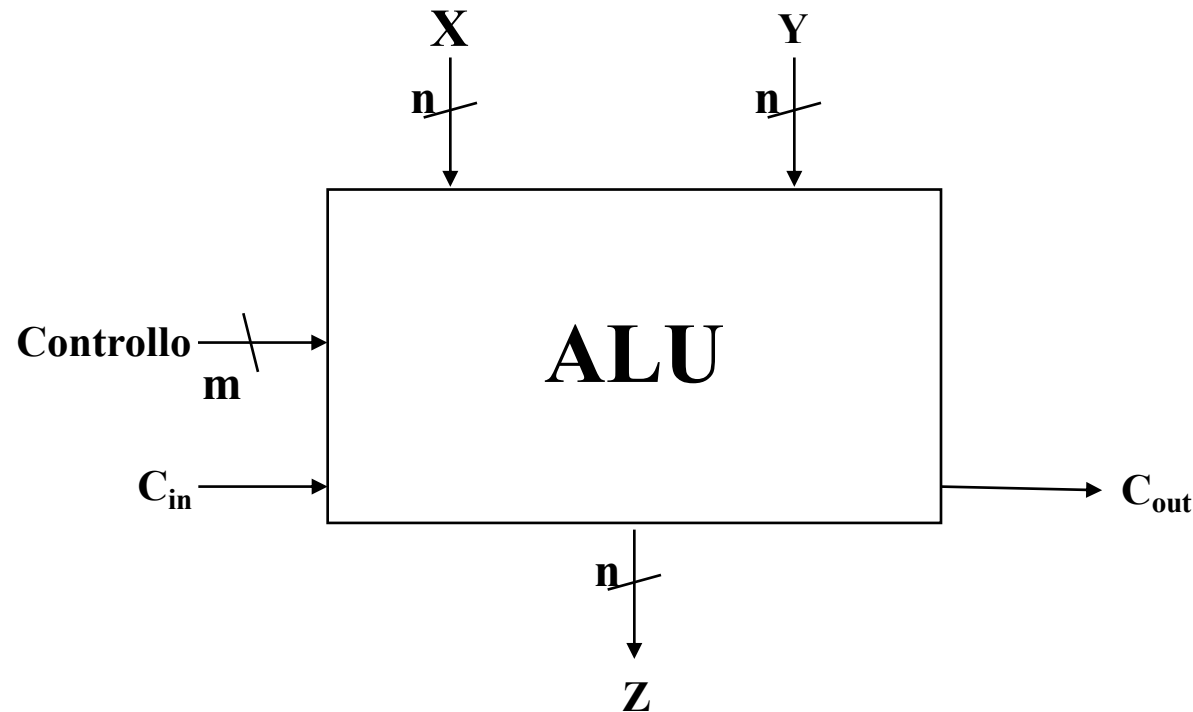


Sommatore a 4 bit



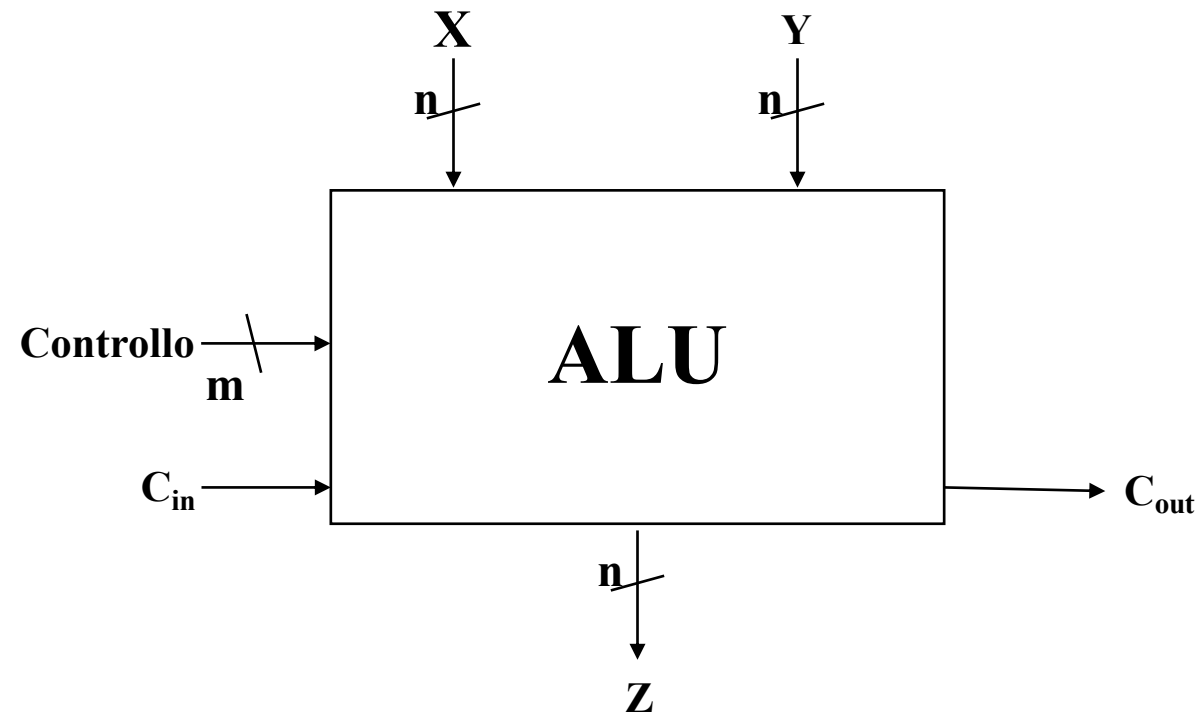
ALU

Le Unità Aritmetico Logiche (ALU) sono moduli combinatori che integrano in un unico blocco le principali funzioni aritmetiche e logiche (tipicamente somma, sottrazione, negazione, and, or, not, exor).



ALU

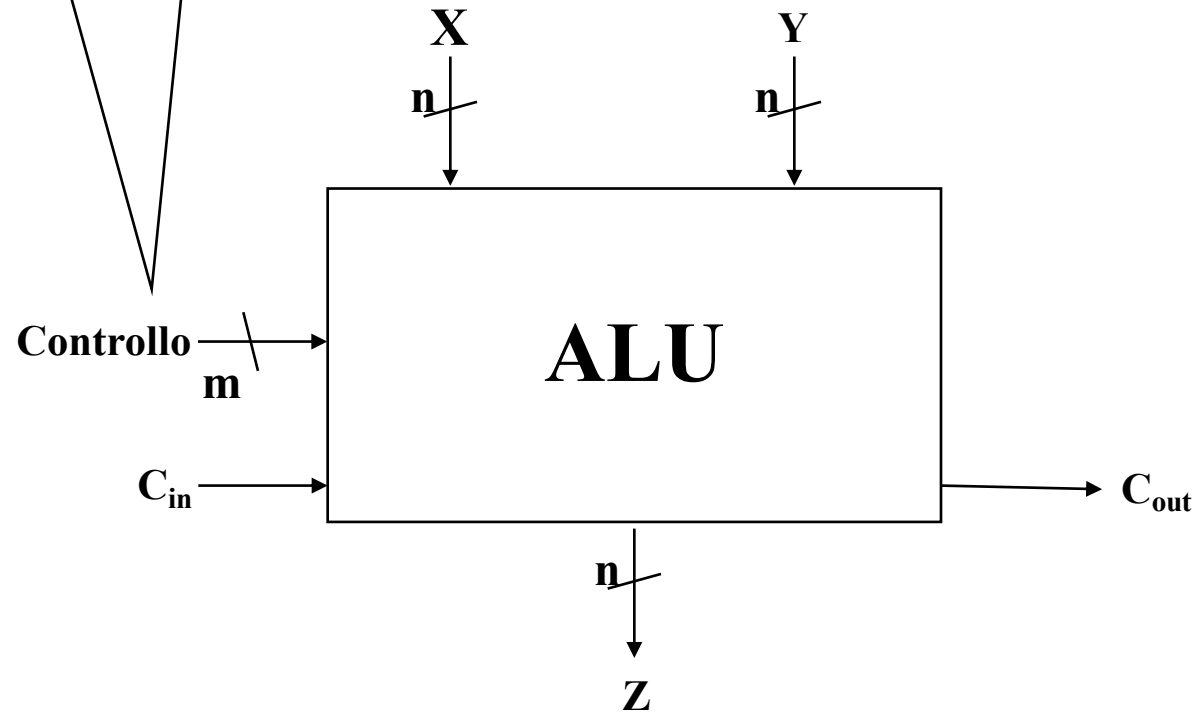
Le Unità Aritmetico Logiche (ALU) sono moduli combinatori che integrano in un unico blocco le principali funzioni aritmetiche e logiche (tipicamente somma, sottrazione, negazione, and, or, not, exor).



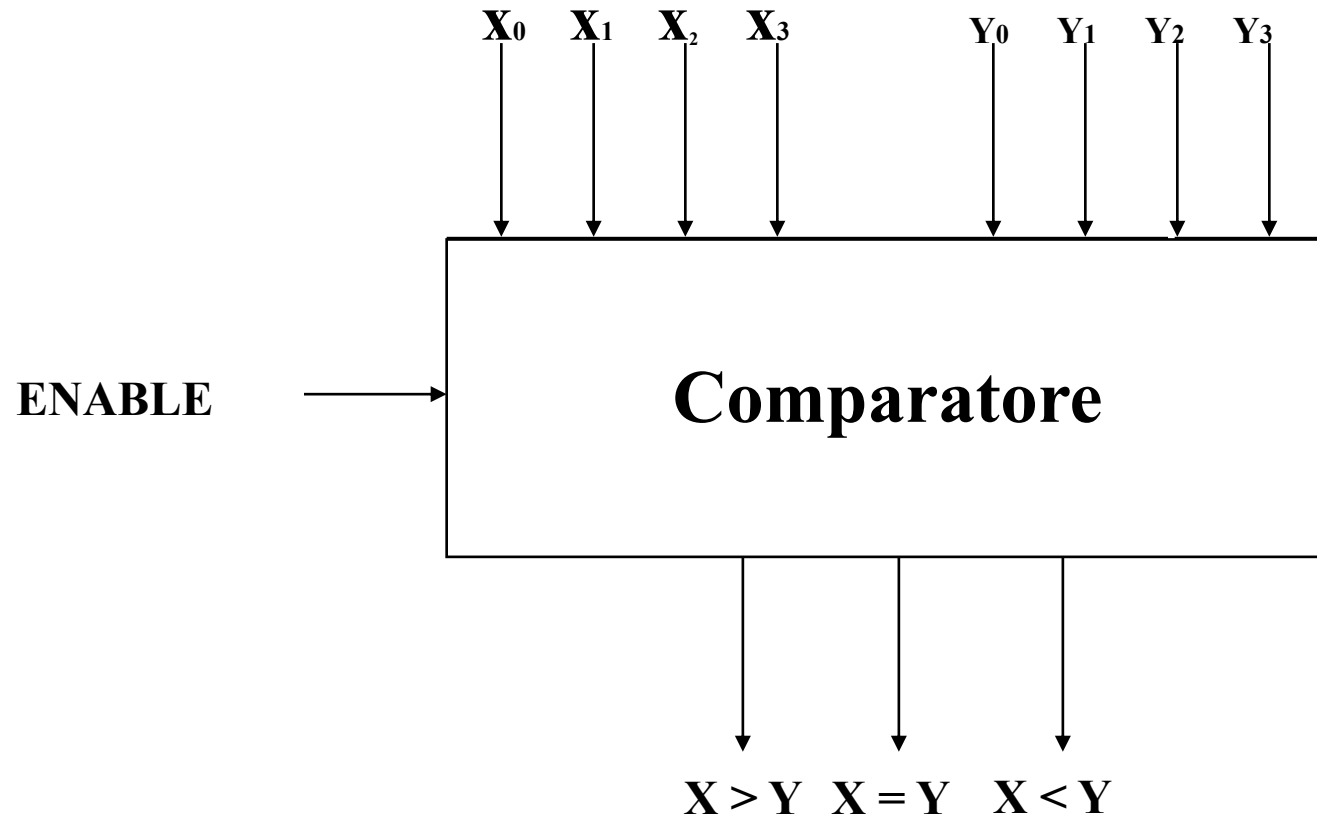
ALU

I segnali di controllo
determinano la funzione
che deve essere svolta dalla
ALU.

Le Unità Aritmetiche e Logiche (ALU) sono moduli combinatori che integrano in un unico blocco le funzioni aritmetiche e logiche (tipicamente somma, sottrazione, moltiplicazione, and, or, not, exor).



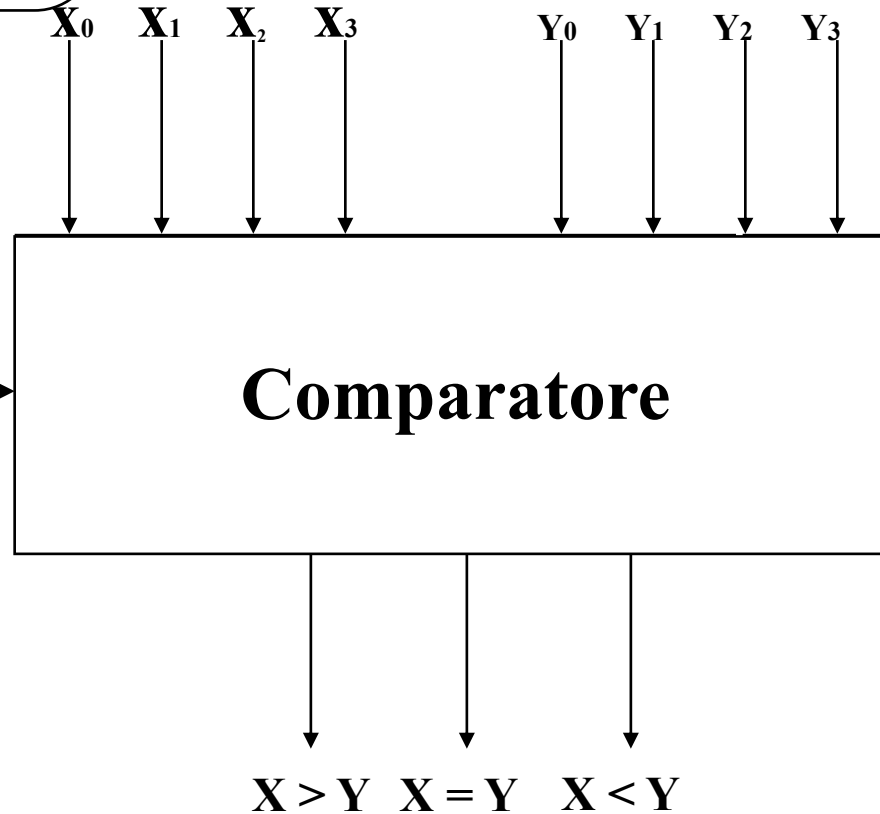
Comparatore a 4 bit



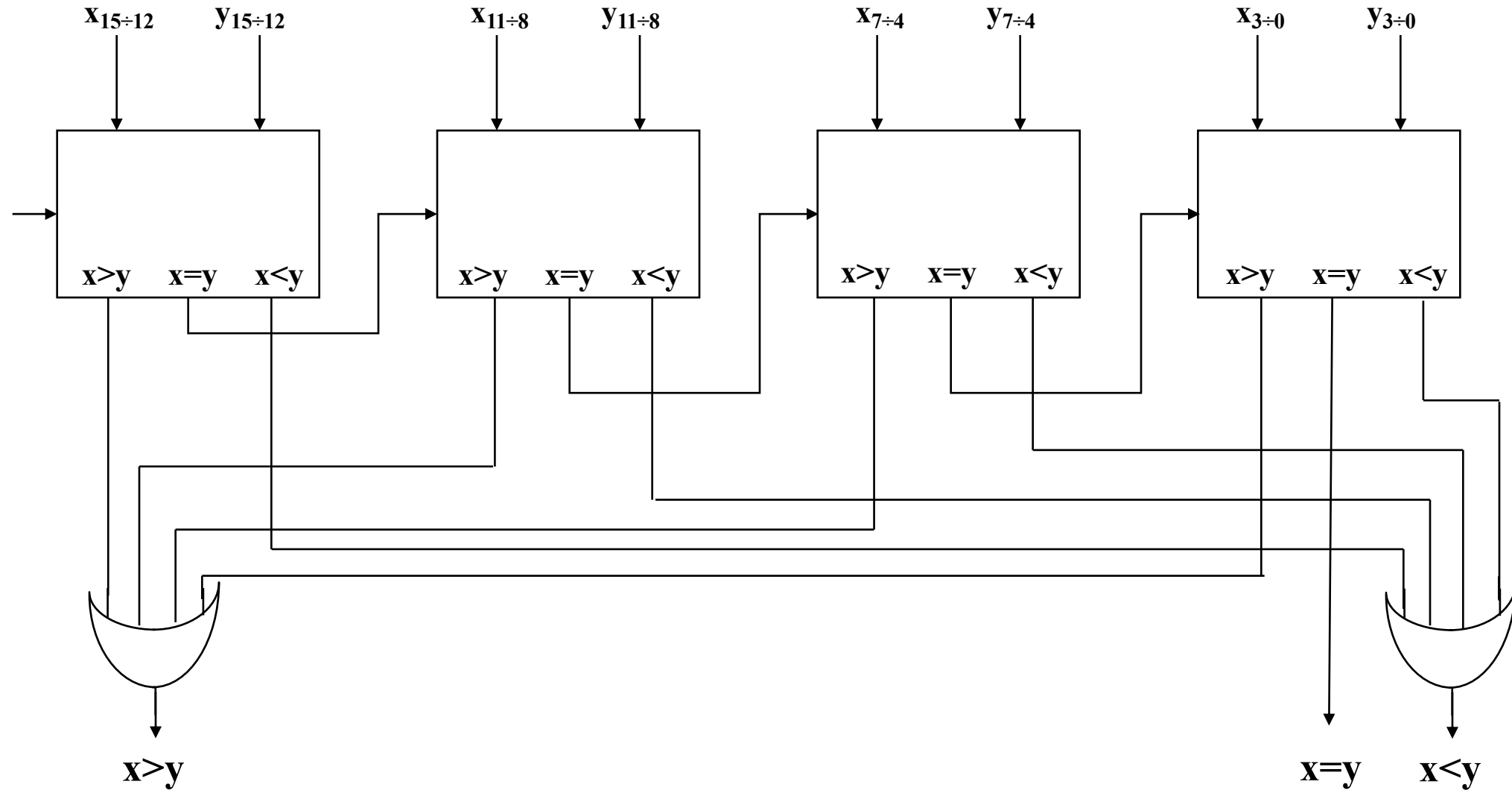
Comparatore a 4 bit

Quando il segnale di enable non è attivo nessuna uscita è attiva (qualunque sia il valore degli altri ingressi).

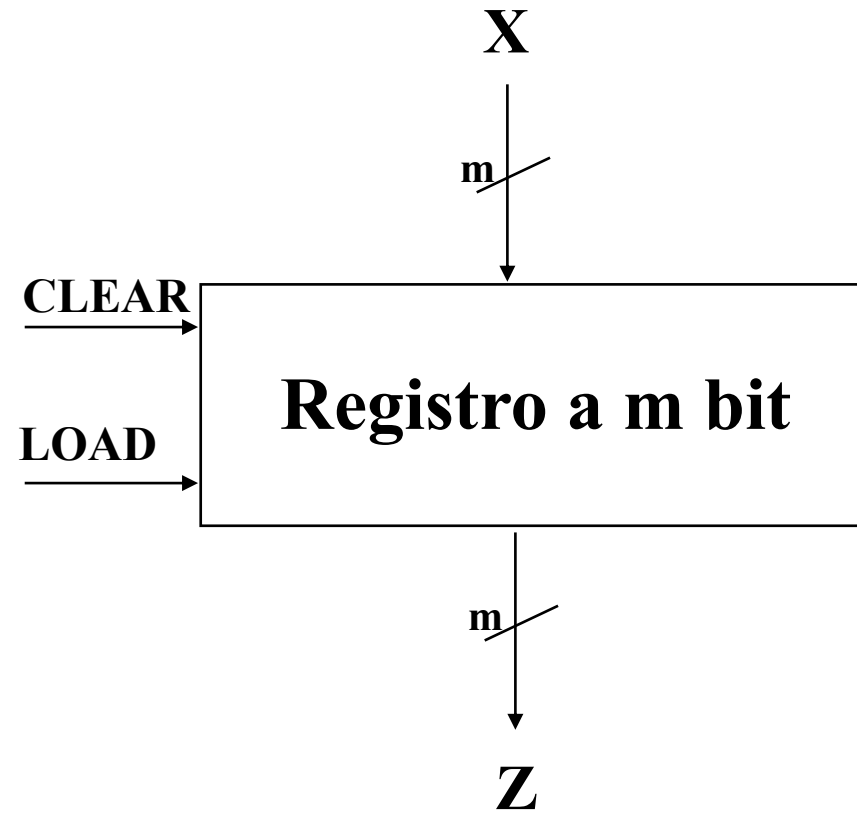
ENABLE



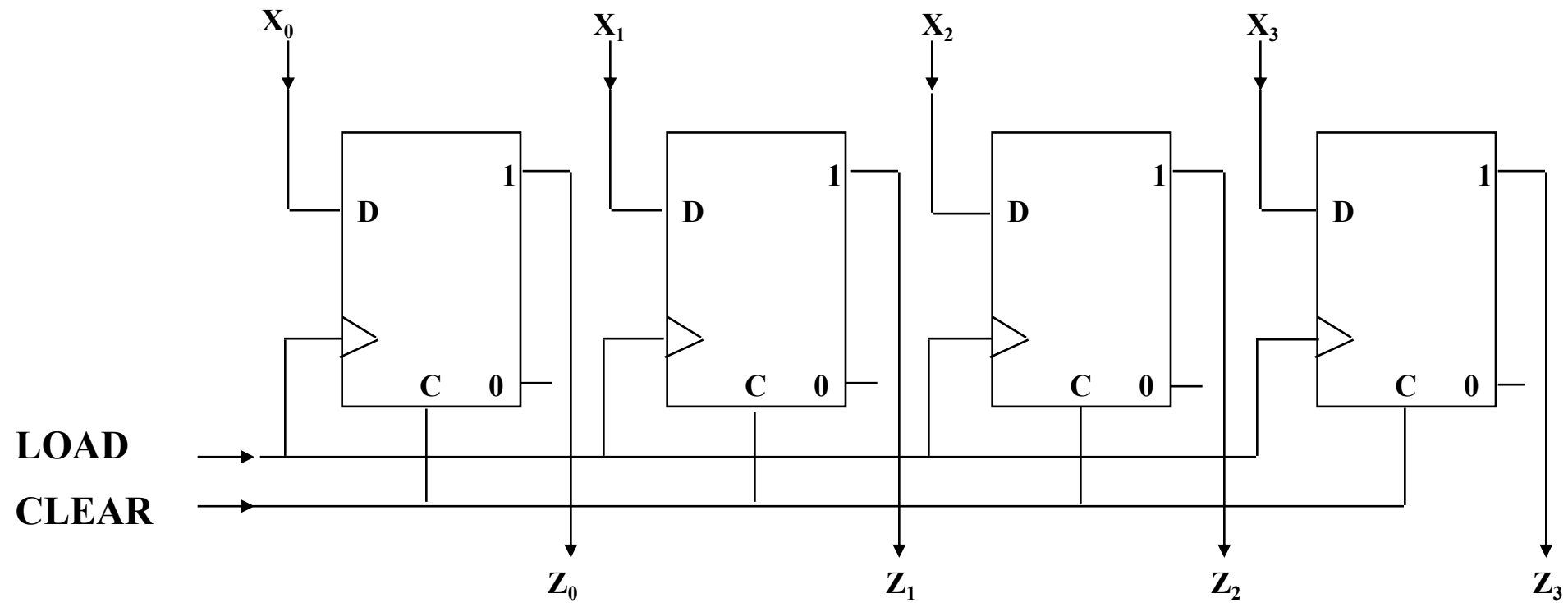
Connessione di comparatori



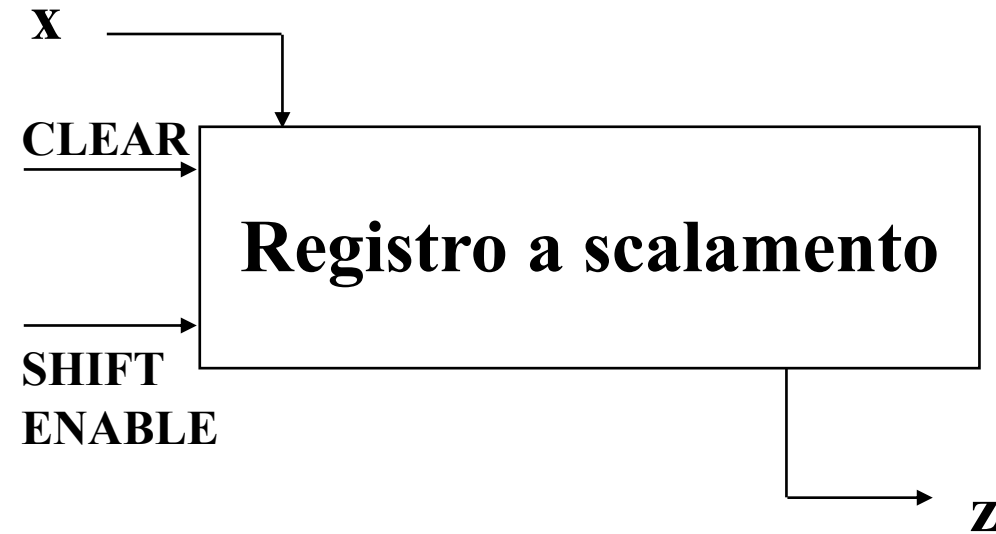
Registro a m bit



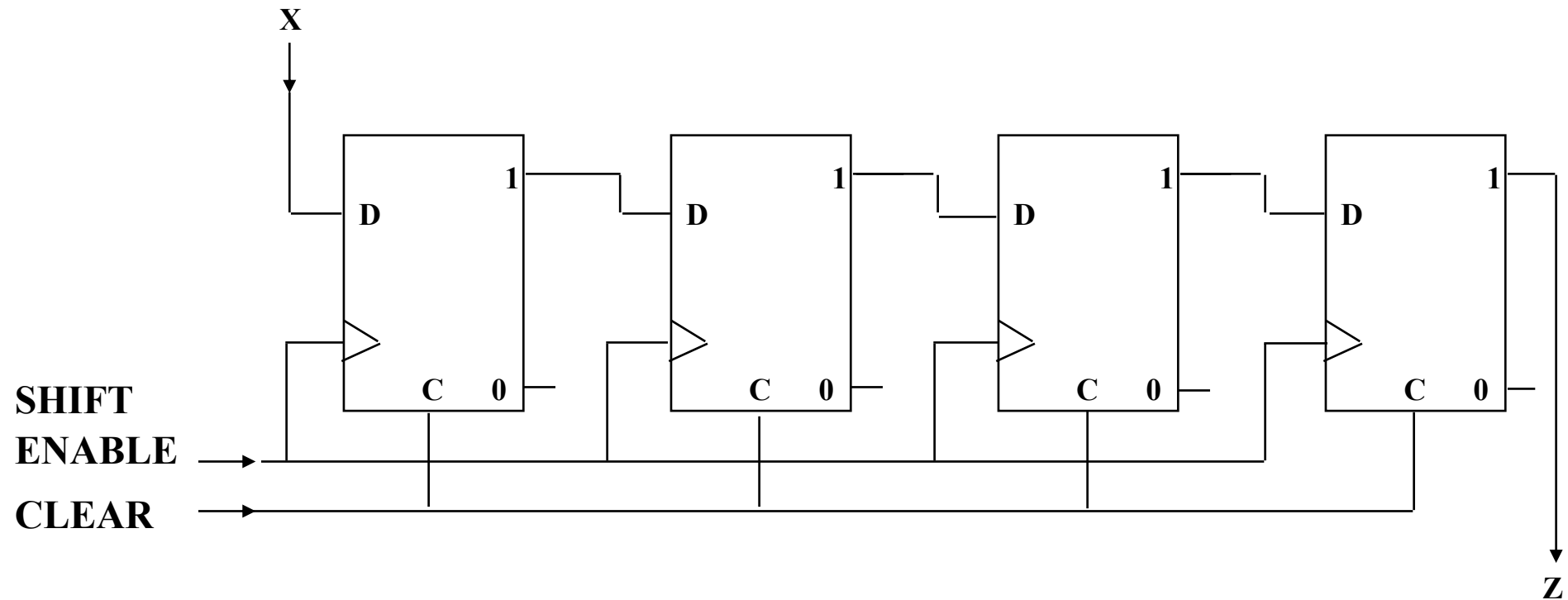
Registro a 4 bit (II)



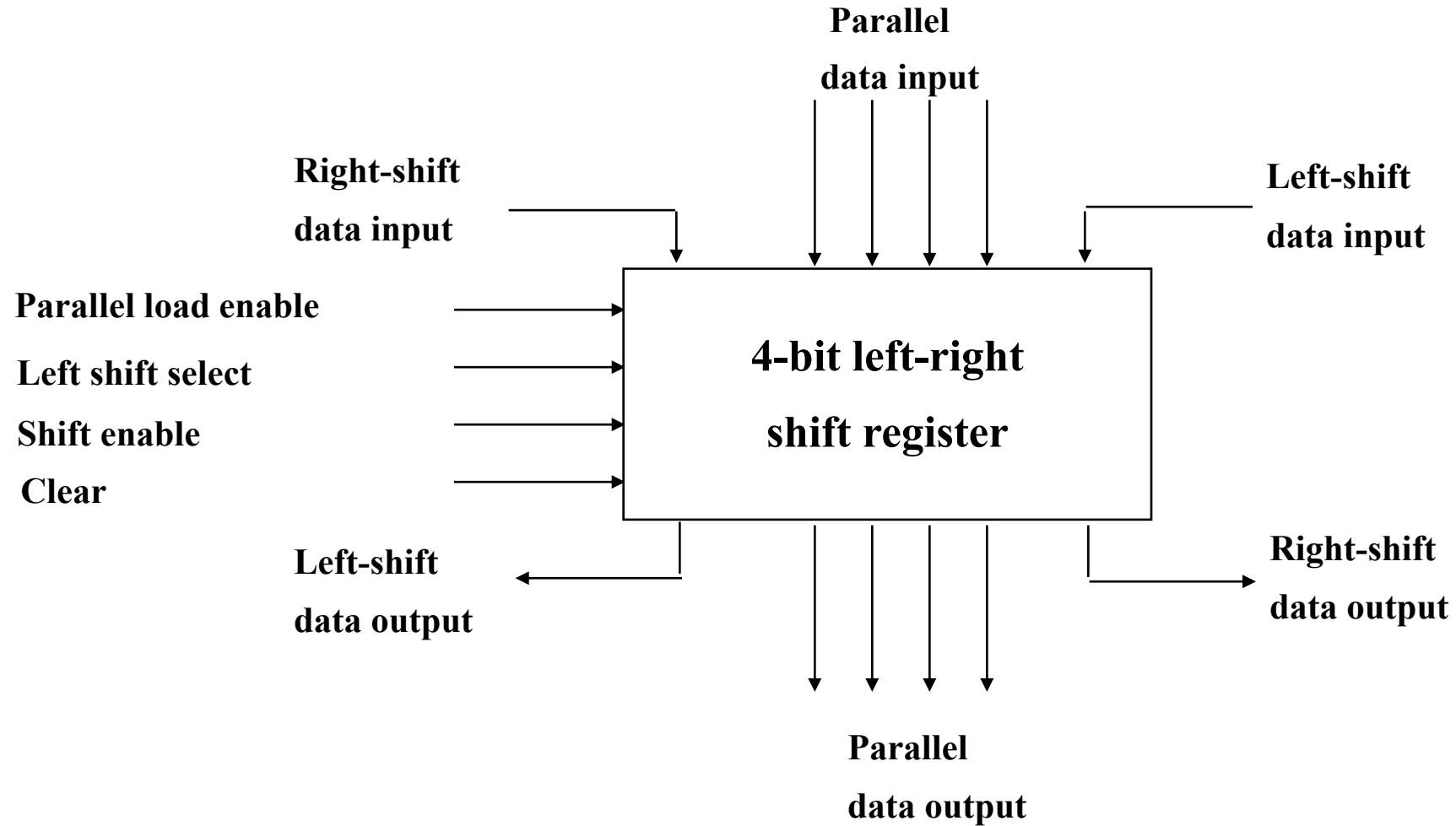
Registri a scalamento (*Shift Register*)



Registro a scalamento a 4 bit: realizzazione



Registro a scalamento universale



Usi dei registri a scalamento

Esempi

- memorizzazione di dati seriali (FIFO)

Usi dei registri a scalamento

Esempi

- **conversione seriale-parallelo e parallelo-seriale**

Usi dei registri a scalamento

Esempi

- **moltiplicazione e divisione di numeri in fixed-point senza segno.**

Contatori

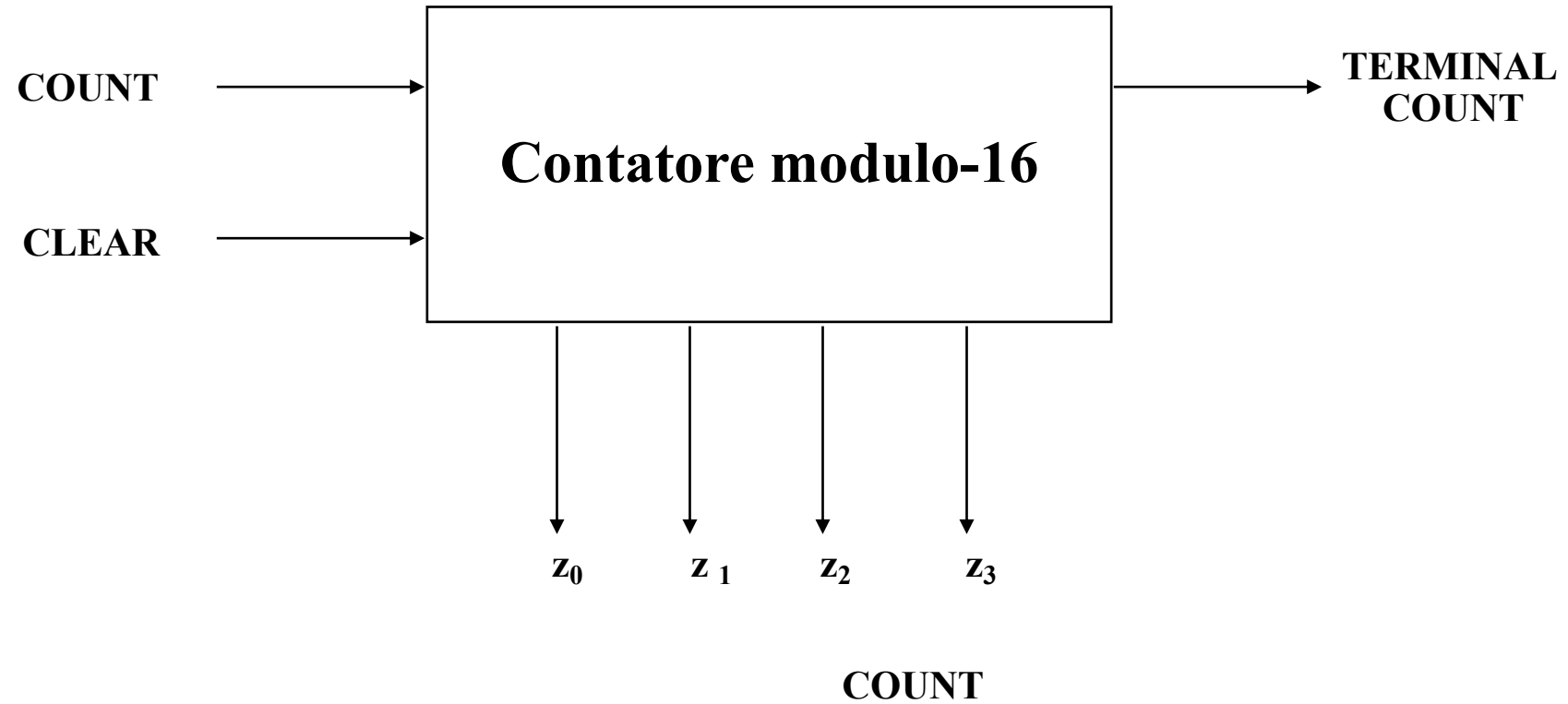
Evolgono attraverso k stati in risposta a k impulsi in ingresso

La codifica degli stati permette di contare il numero di impulsi

Tipologie:

- ***up-down counter*: possono contare avanti e indietro**
- ***programmable counter*: il valore del modulo può essere modificato**
- ***binary/decimal/gray/BCD counter*: possono contare in binario/decimale/gray/BCD.**

Contatore semplice



Usi dei contatori

Esempi

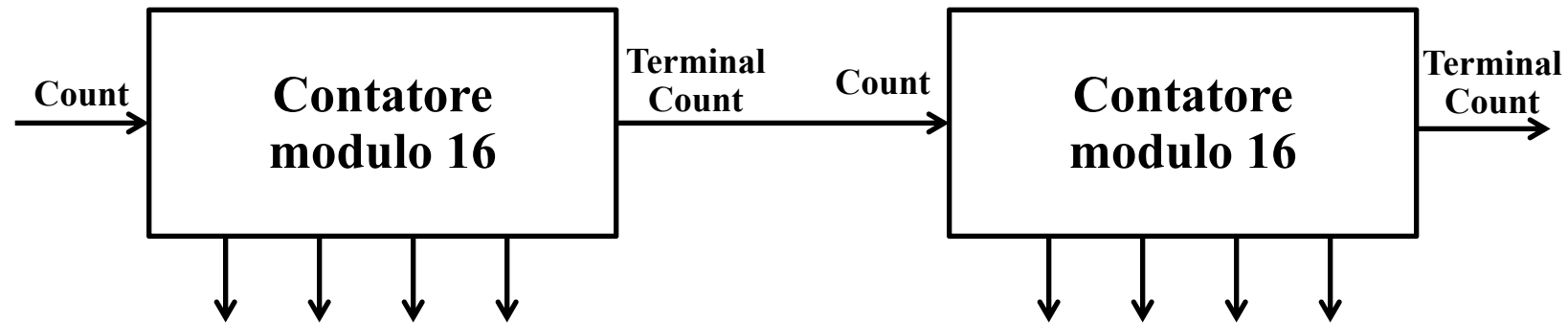
- come *Program Counter* in una macchina a stati
- per la generazione di segnali di temporizzazione (*divisori di frequenza*)
- come contatori di eventi.

Collegamento in cascata

Se si hanno a disposizione contatori di una certa dimensione è possibile costruire contatori di dimensione maggiore.

Esempio

Si vuole costruire un contatore modulo 256 utilizzando due contatori modulo 16.



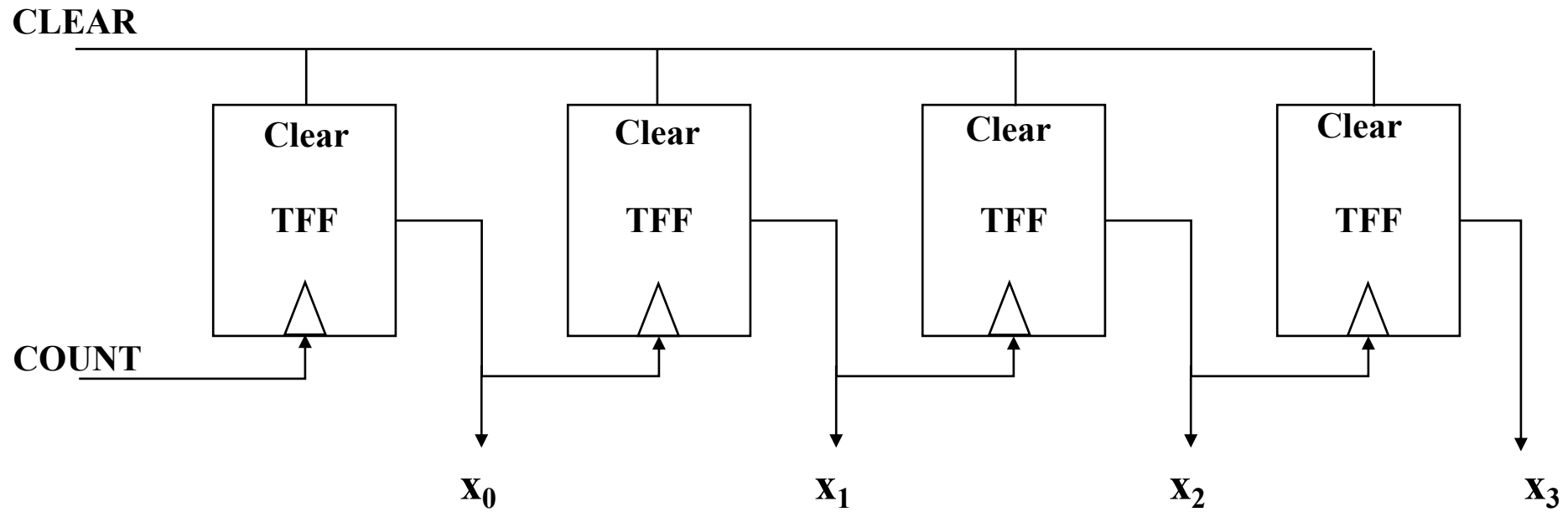
Contatori:

modalità di realizzazione

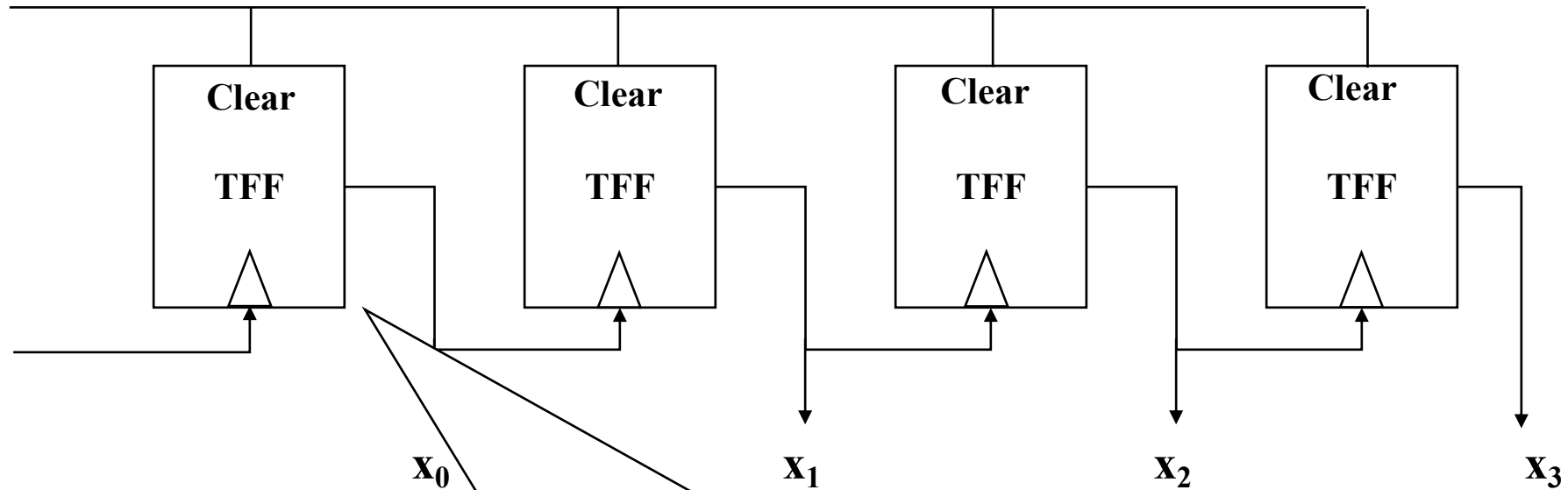
Esistono due principali modalità di realizzazione:

- *ripple counter (o asincroni)*
 - hanno un ritardo dipendente dalla lunghezza
- *contatori sincroni*
 - tutte le uscite assumono contemporaneamente il valore corrente.

Contatore asincrono



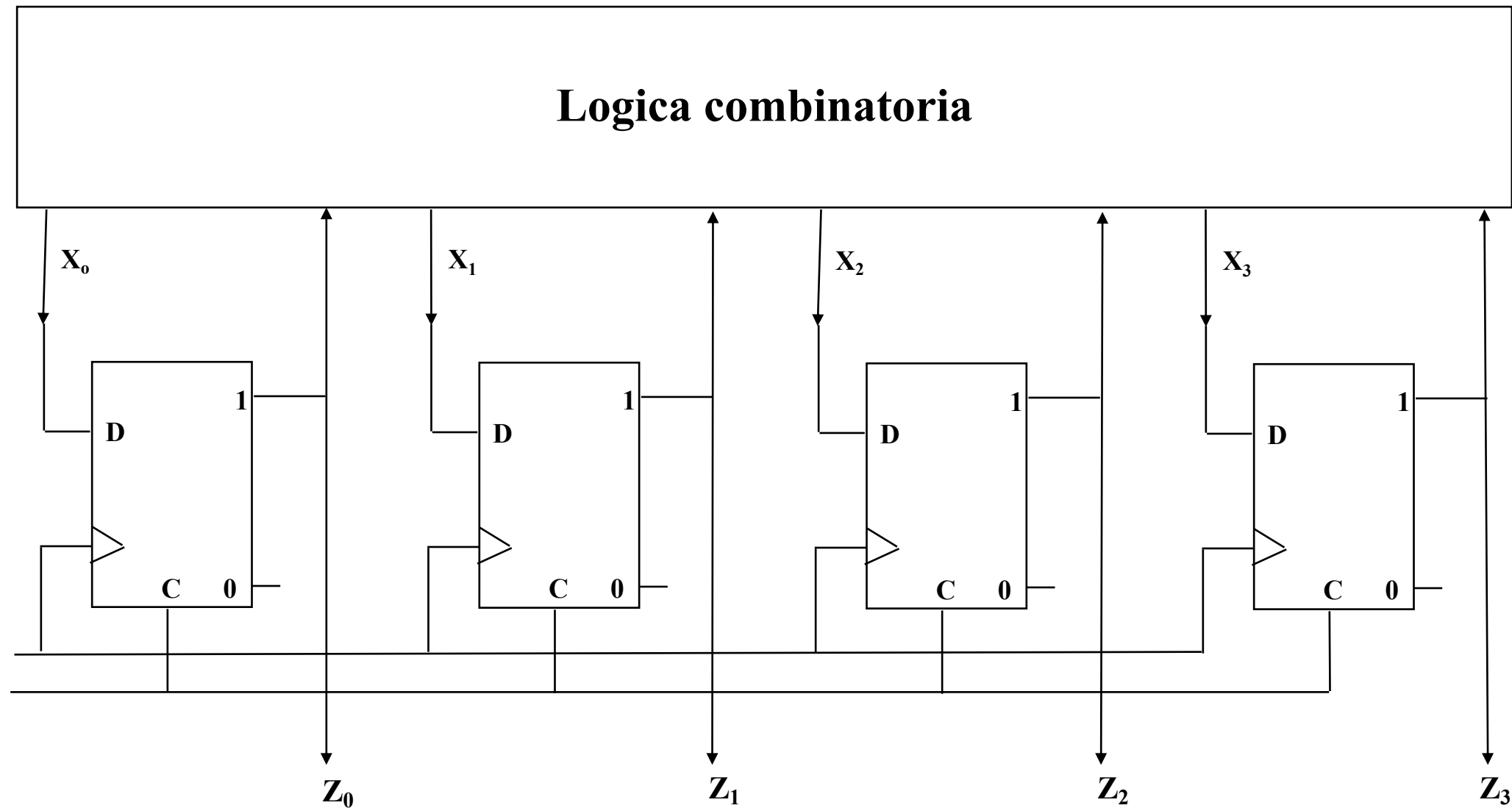
Contatore asincrono



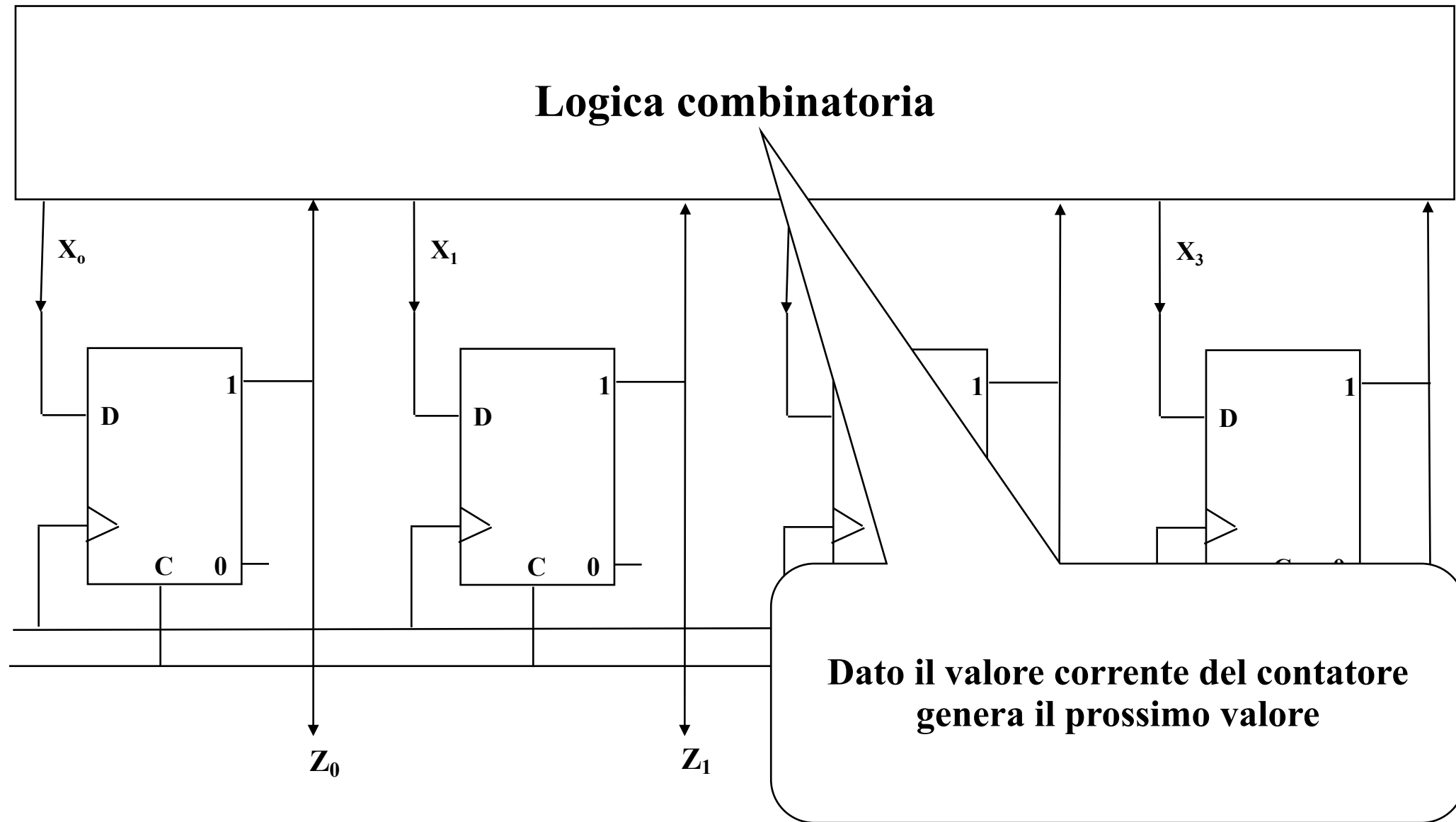
Il flip flop di tipo T si complementa ad ogni impulso di clock.

Può essere ottenuto dal flip flop di tipo D connettendo l'uscita all'ingresso attraverso un inverter.

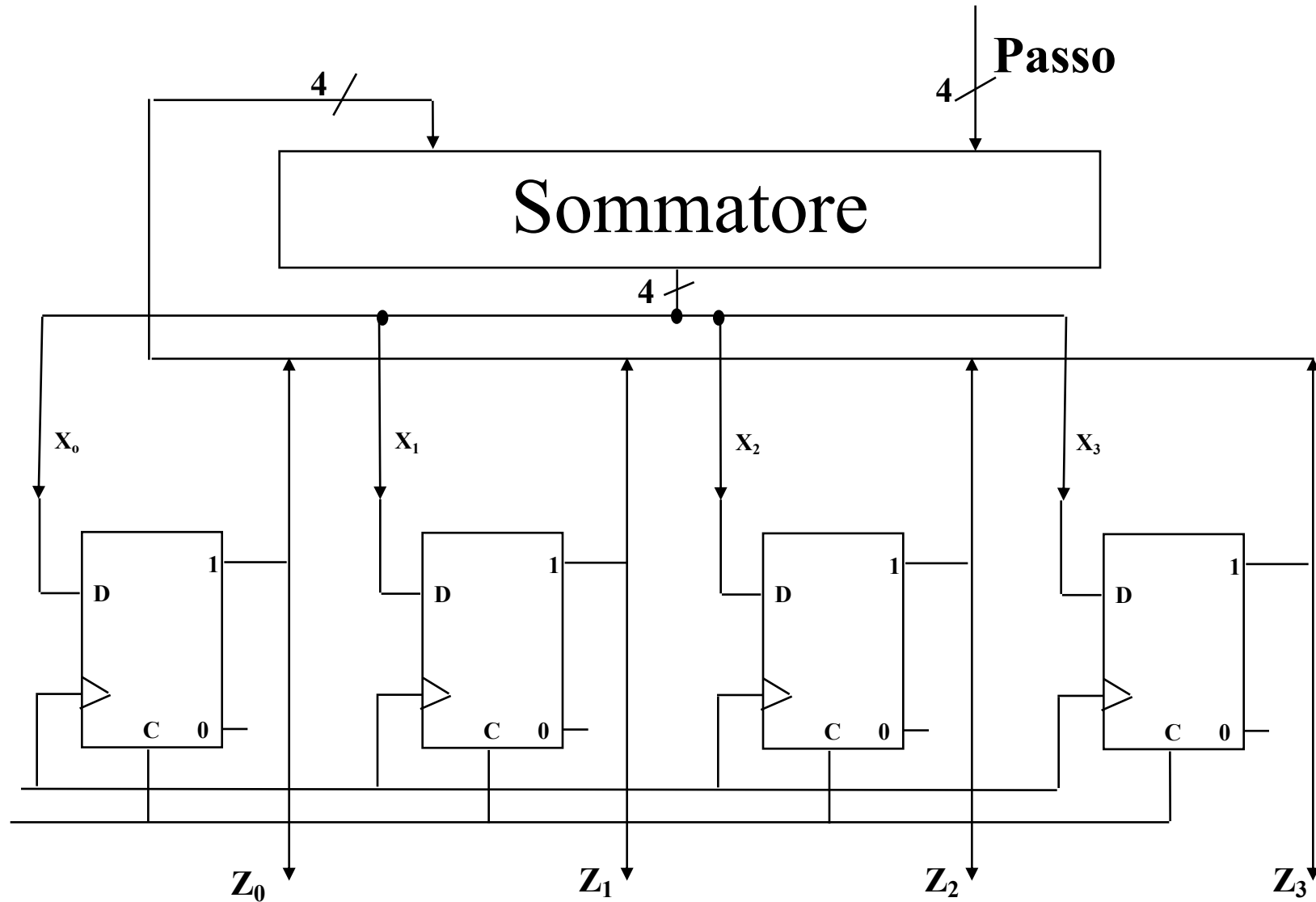
Contatore sincrono



Contatore sincrono



Contatore sincrono modulo 16 con passo programmabile



FPGA

I *Field Programmable Gate Array* (FPGA) sono dispositivi programmabili che permettono di realizzare circuiti con funzionalità complesse, sia combinatorie sia sequenziali.

Rappresentano una soluzione alternativa alla realizzazione di un ASIC (Application Specific IC).

ASIC

- **Elevato costo di progettazione**
- **Basso costo unitario**

FPGA

- **Basso costo di progettazione**
- **Elevato costo unitario.**

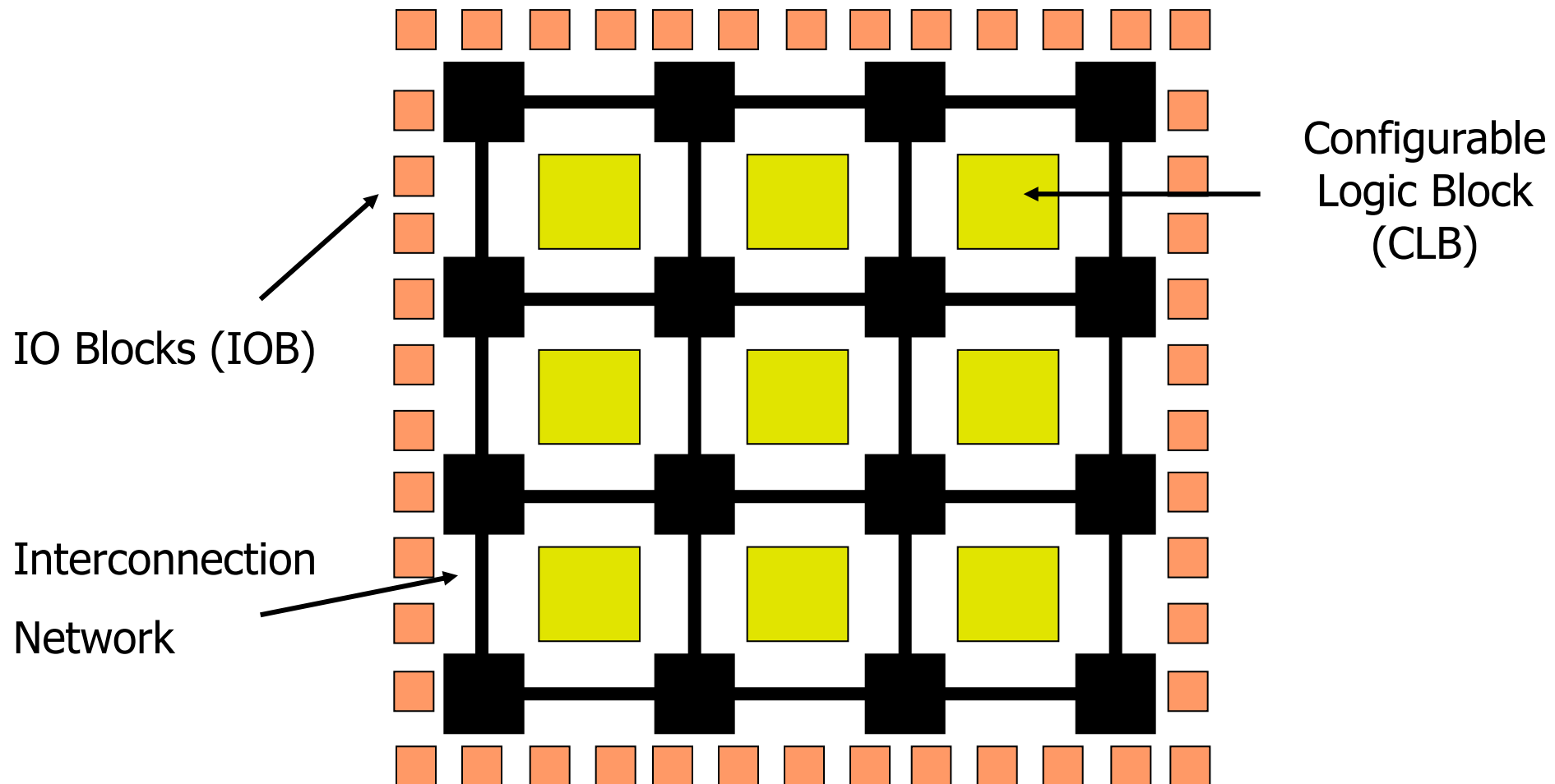
FPGA

Un dispositivo FPGA è composto da una griglia di blocchi logici (CLB) connessi da una rete di interconnessione e accessibili da un certo numero di segnali di I/O.

Sia i blocchi logici sia quelli di interconnessione sono programmabili:

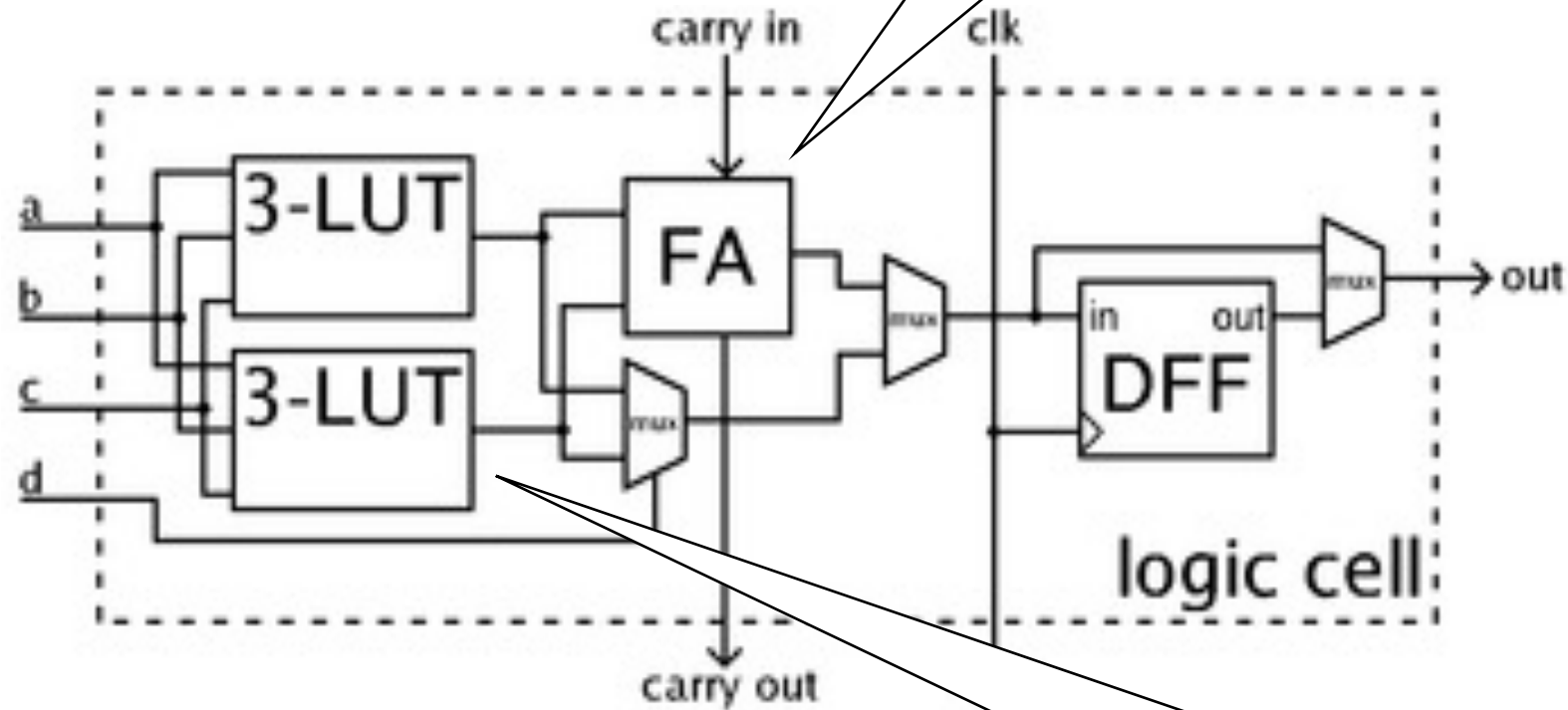
- **programmando i primi si definisce la funzione che questi svolgono**
- **programmando i secondi si definiscono le connessioni che devono essere realizzate.**

FPGA: architettura



CLB

Full Adder



Una LUT implementa una qualsiasi funzione logica a 3 ingressi, definita in fase di programmazione

FPGA: programmazione

La programmazione di un dispositivo FPGA definisce

- il comportamento di ciascun CLB**
- il comportamento delle interconnessioni**
- il comportamento dei segnali di I/O.**

FPGA: programmazione

La programmazione di un dispositivo FPGA può avvenire in vari modi:

- **al momento del primo utilizzo, «bruciando» alcuni elementi del dispositivo; la programmazione viene eseguita una sola volta (*One Time Programmable* o *OTP*)**
- **caricando in una memoria Flash interna una serie di bit (*bitstream*) che determinano il comportamento del dispositivo; la programmazione viene eseguita ogni volta che si vuole cambiare il circuito implementato**
- **caricando in una memoria RAM interna il bitstream; la programmazione viene eseguita ogni volta che il dispositivo viene alimentato.**

FPGA: programmazione

Il successo dei dispositivi FPGA dipende fortemente dalla disponibilità di strumenti CAD a basso costo che generano automaticamente i file di programmazione (*bitstream*) per il dispositivo.

FPGA: usi

Le FPGA sono spesso utilizzate per realizzare a basso costo dispositivi altrimenti realizzabili tramite ASIC (*Application Specific IC*).

Rispetto agli ASIC le FPGA sono una soluzione

- **più economica da progettare**
- **più costosa da produrre (in termini di costo di produzione)**
- **più lenta**
- **più costosa in termini di spazio.**

Le FPGA sono quindi usate principalmente per

- **prodotti realizzati in numero limitato**
- **applicazioni riconfigurabili**
- **prototipi.**

FPGA: prodotti

I prodotti più diffusi basati sulla tecnologia FPGA sono

- **AMD Xilinx**
- **Intel Altera**
- **Microchip**

AMD 
XILINX



Bus

Permettono lo scambio di dati tra diversi componenti.

Possono essere realizzati

- **all'interno del singolo IC (*Integrated Circuit*)**
- **su una piastra (per connettere dispositivi diversi)**
- **come interconnessione tra piastre (*backplane*).**

Al bus possono essere associati:

- **dispositivi di amplificazione**
- **buffer.**

Bus: costo

Il costo di un bus può essere elevato in termini di area di silicio (se connette moduli distanti sullo stesso IC) e di pin (se è accessibile dall'esterno del chip).

Connessione al bus

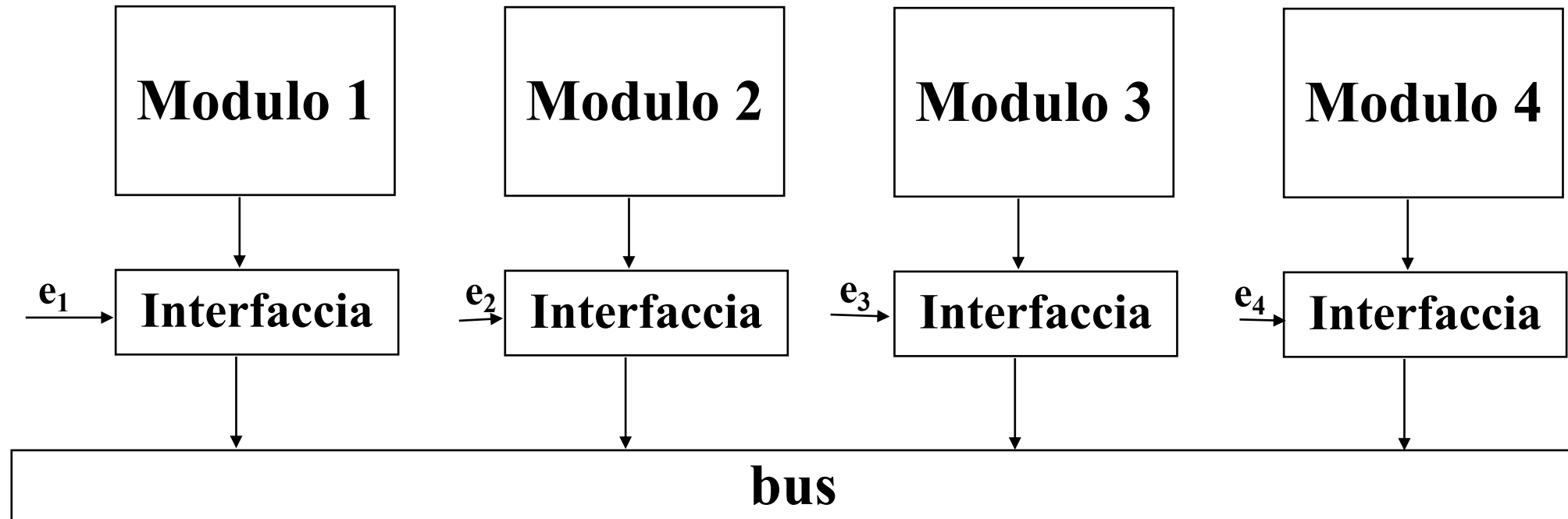
Il generico dispositivo connesso a un bus può

- **leggere i valori presenti sul bus**
- **scrivere sul bus**
- **leggere o scrivere a seconda dei momenti.**

Connessione al bus

Ad ogni istante uno e un solo dispositivo tra quelli connessi in scrittura al bus deve pilotarne il valore.

Gli altri devono assumere un valore particolare noto come Z (*alta impedenza*).



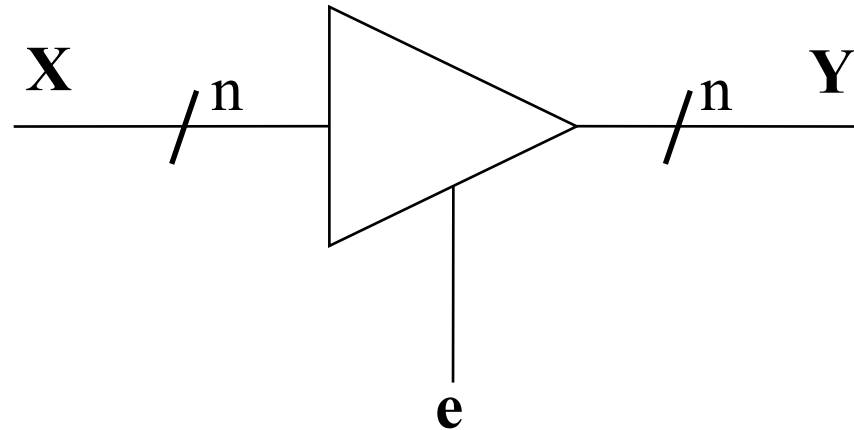
Buffer Tri-State

I buffer che fungono da interfaccia verso il bus sono detti *tri-state*.

Possiedono n ingressi di dato X , 1 ingresso di controllo e , e n uscite di dato Y .

Il valore di Y è

- quello di X , se $e=1$
- Z , se $e=0$.



Transceiver

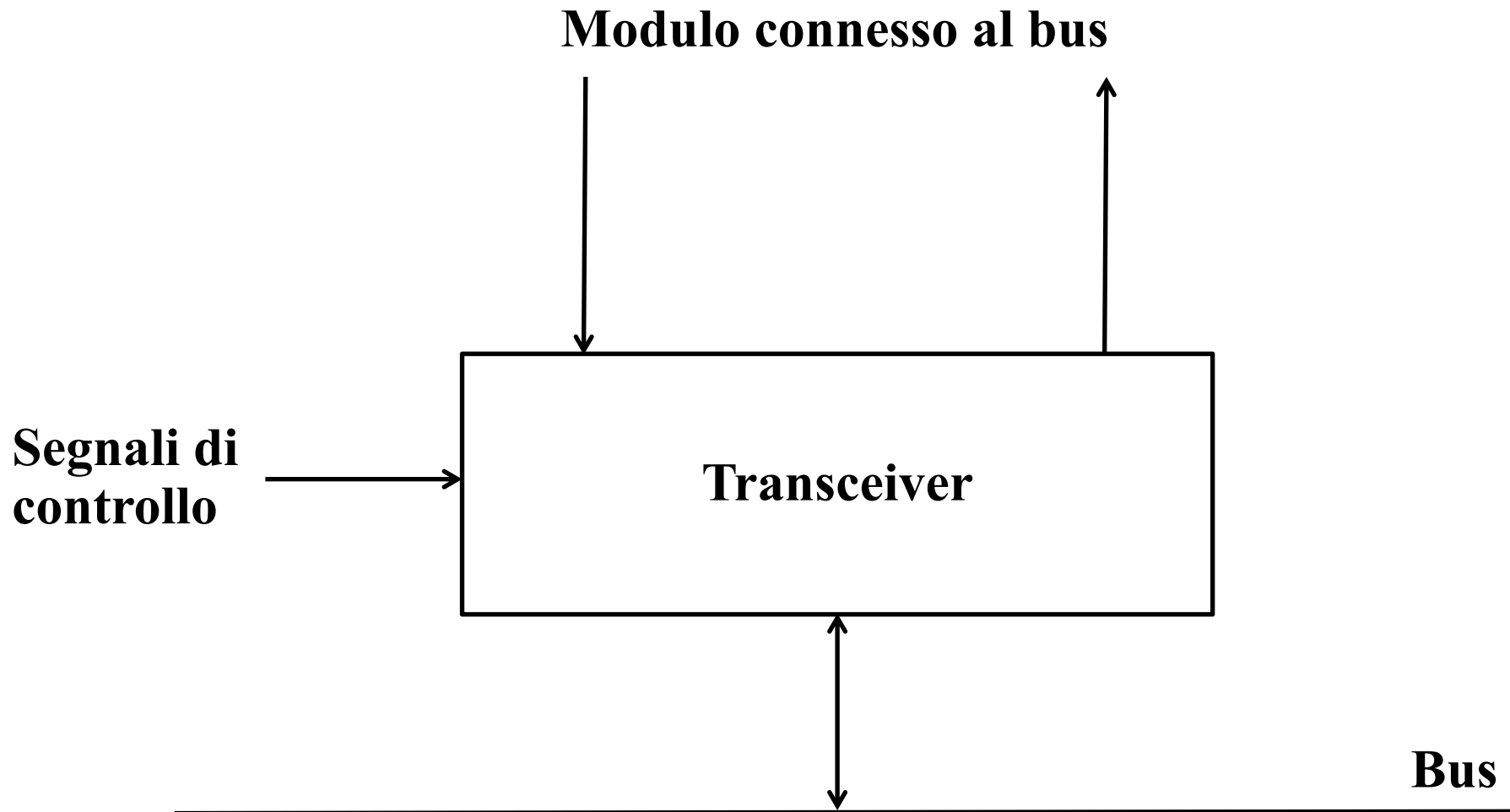
Alcuni moduli (ad esempio i processori e le memorie) possono prevedere segnali bidirezionali che a seconda dei momenti si configurano in entrata o in uscita.

In tal caso la connessione al bus deve avvenire attraverso un dispositivo (denominato *transceiver*) che risolve i problemi esistenti dal punto di vista elettrico.

A seconda del valore dei segnali di controllo, i segnali del trasceiver connessi al bus

- leggono i valori provenienti dal bus e li trasferiscono al modulo**
- scrivono sul bus i valori provenienti dal modulo**
- assumono il valore Z.**

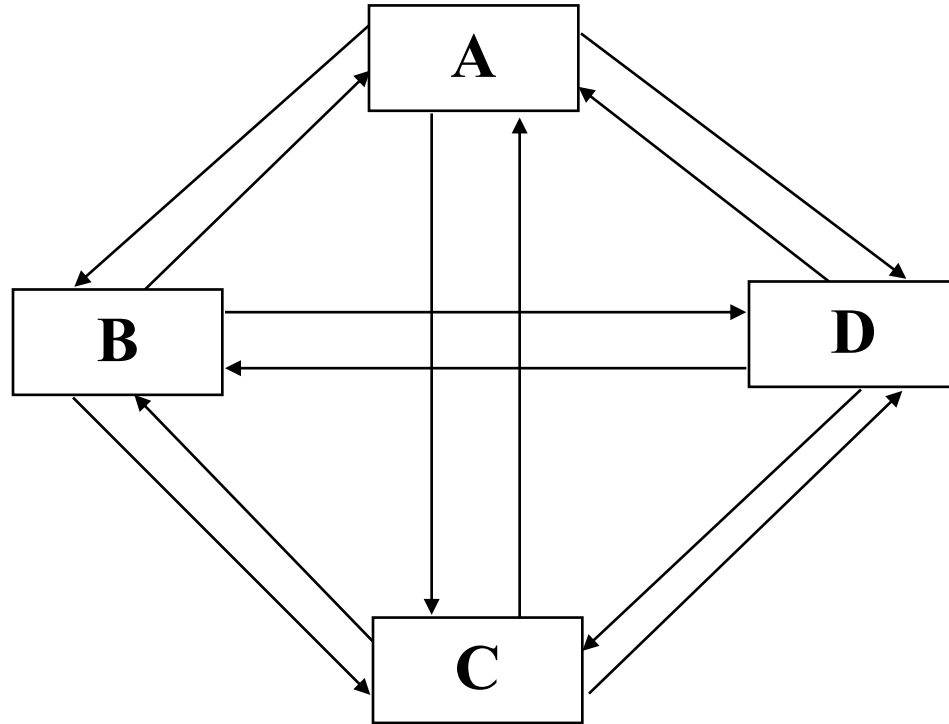
Transceiver



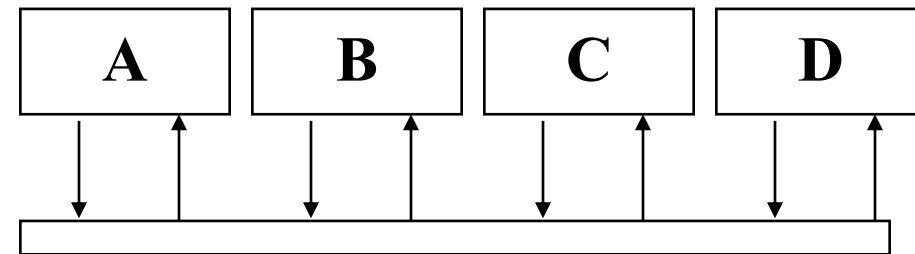
Bus: tipologie

Possono essere:

- *dedicati* (alto costo, elevate prestazioni)
- *condivisi* (basso costo, basse prestazioni): richiedono un meccanismo di regolamentazione degli accessi.



Bus dedicato



Bus condiviso

Memorie

Sono componenti corrispondenti funzionalmente ad insiemi di celle di memorizzazione, ciascuna in grado di memorizzare un bit.

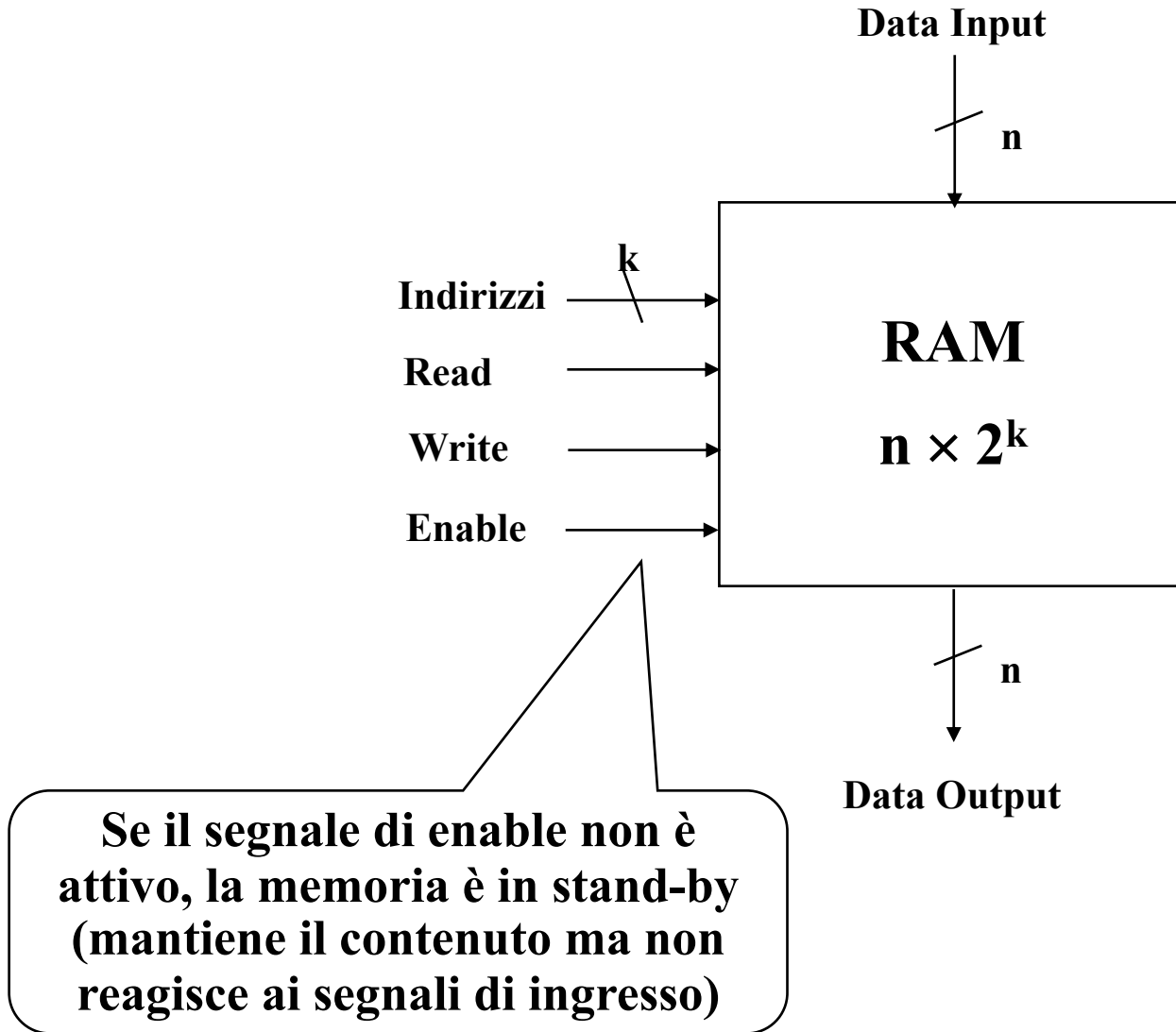
Le celle sono organizzate in *parole*, ciascuna composta da n bit.

Tutti i bit di una parola sono letti e scritti insieme durante la generica operazione di I/O da/verso la memoria.

Ogni parola è caratterizzata da un *indirizzo*, corrispondente ad un intero tra 0 e $m-1$ (dimensione della memoria) rappresentato su k bit ($m=2^k$).

La lettura o scrittura di una parola avviene applicando il corrispondente indirizzo agli ingressi della memoria.

RAM



RAM: funzionamento

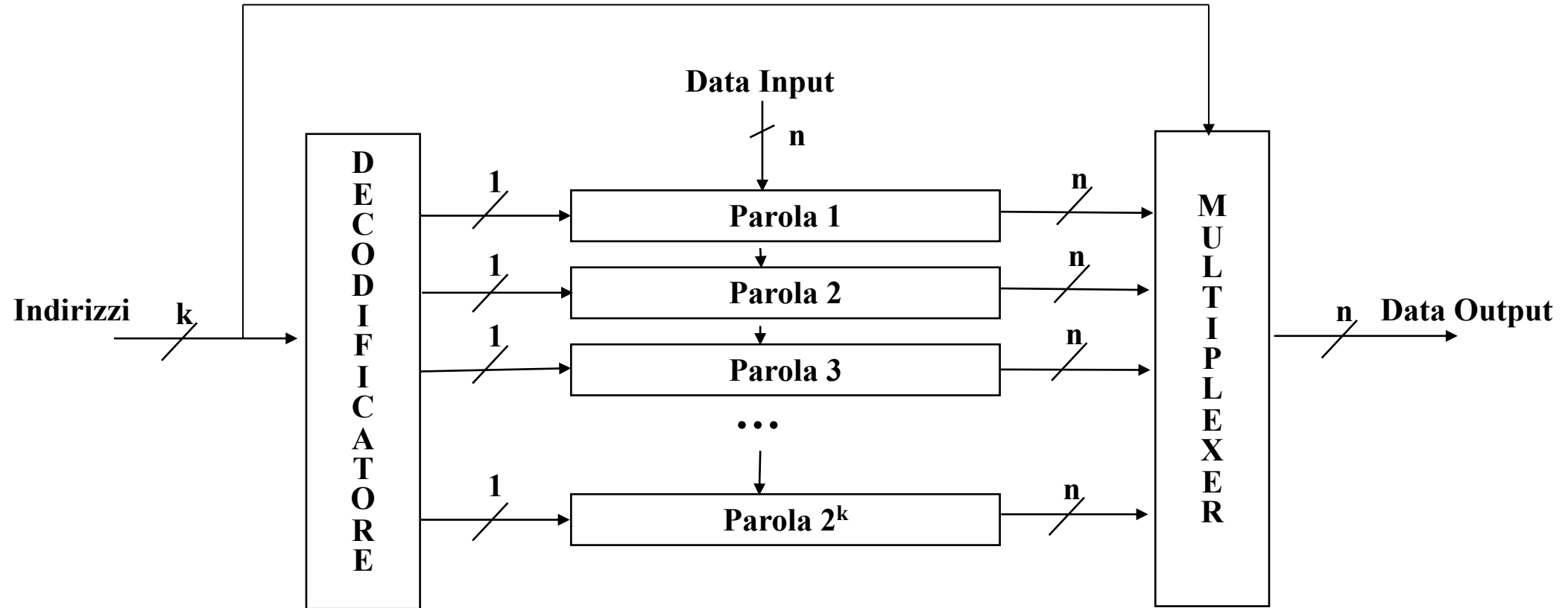
Ciclo di scrittura:

- si fornisce l'indirizzo
- si fornisce il dato
- si attiva il segnale di *write*.

Ciclo di lettura:

- si fornisce l'indirizzo
- si attiva il segnale di *read*
- si legge il dato.

RAM: struttura interna



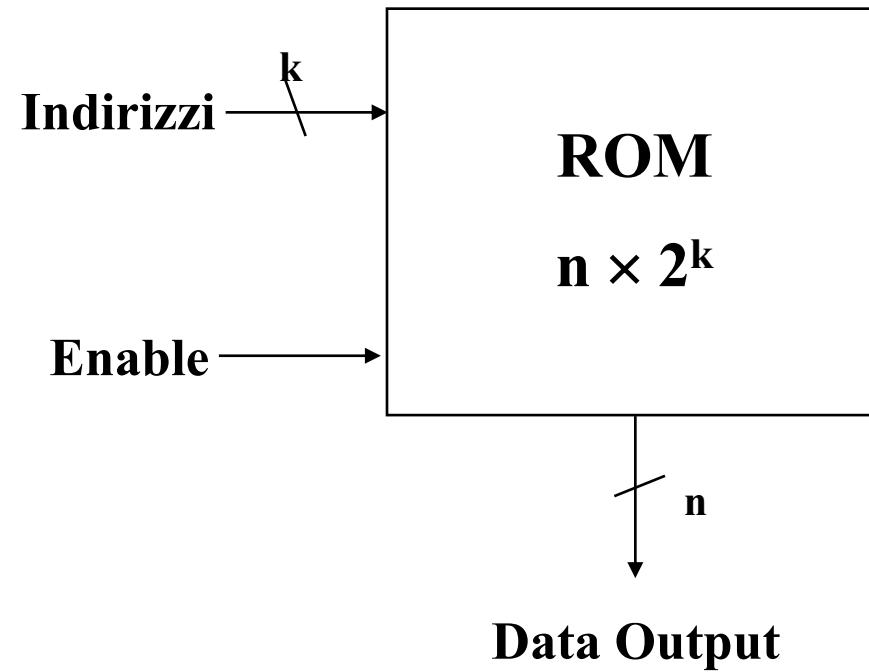
ROM

Sono memorie a sola lettura.

Implementano una funzione combinatoria.

Il contenuto viene determinato all'atto della fabbricazione, e non può in nessun modo essere modificato successivamente.

ROM

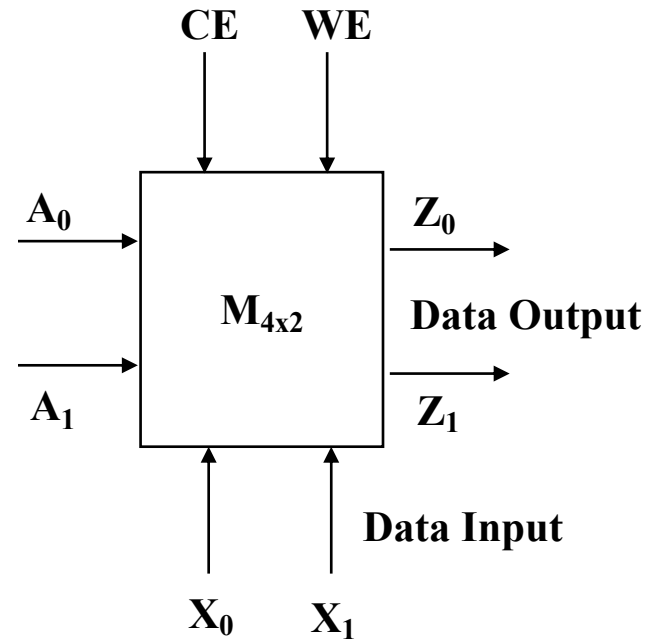


Banchi di memoria

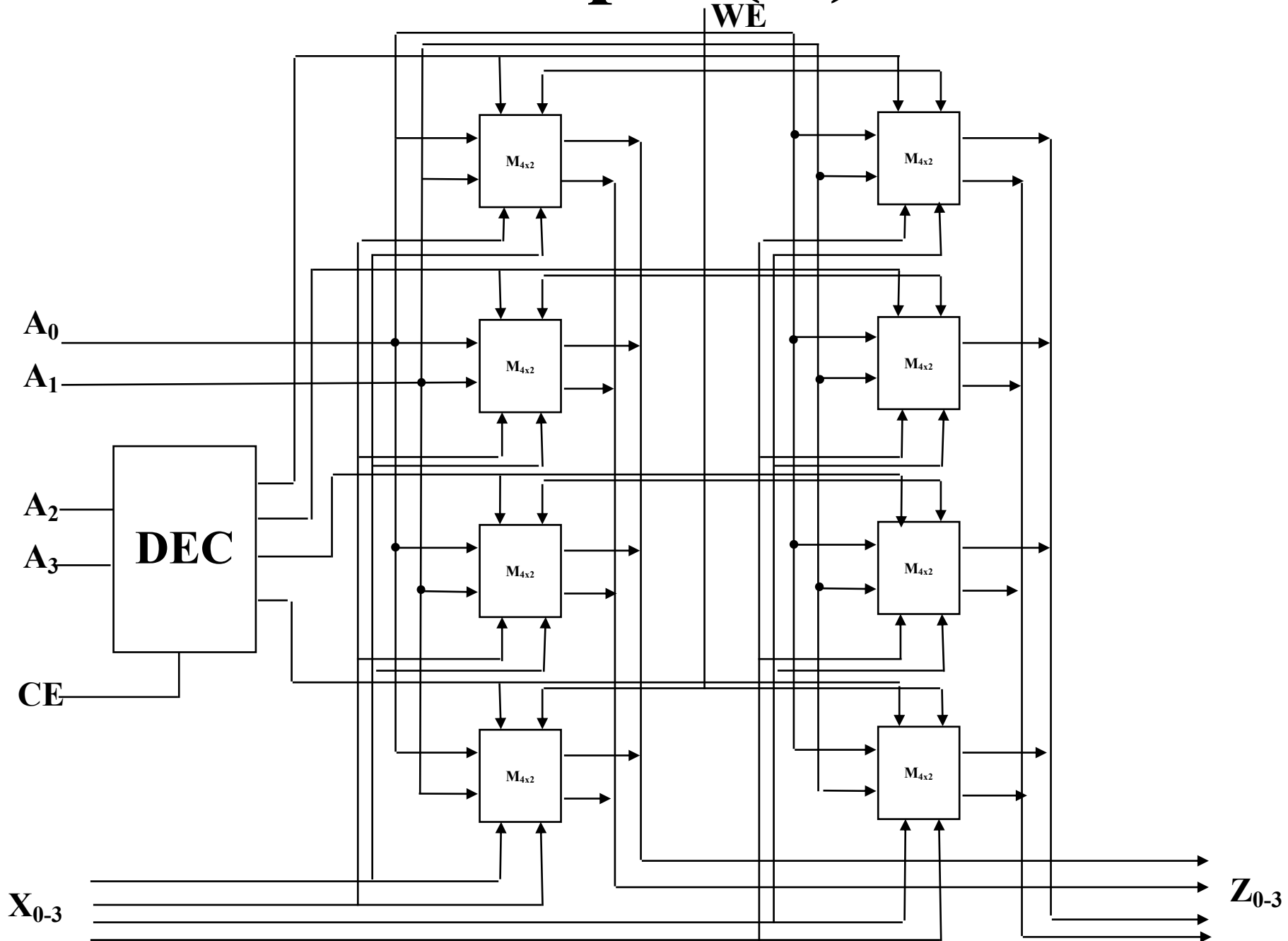
Un *banco* di memoria è un insieme di moduli di memoria opportunamente interconnessi che si comportano come un'unica memoria (ROM, RAM o mista) di dimensioni complessive pari alla somma delle dimensioni dei moduli componenti.

Esempio

Si vuole progettare una RAM composta da 16x4 bit utilizzando il seguente modulo, contenente 4 parole da 2 bit ciascuna:



Esempio (II)



Regole generali per il progetto

- **Tutti i chip sono collegati in parallelo al data bus**
- **Gli ingressi di indirizzo dei vari chip sono pilotati in parallelo dai bit meno significativi dell'indirizzo**
- **I bit di indirizzo più significativi vanno ad un decoder, che pilota i segnali di enable (CE) dei vari chip di memoria.**