



Corso di

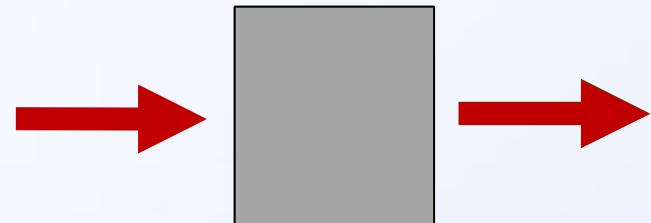
Architettura degli Elaboratori

a.a. 2021/2022

Il livello logico digitale: Circuiti sequenziali e memoria

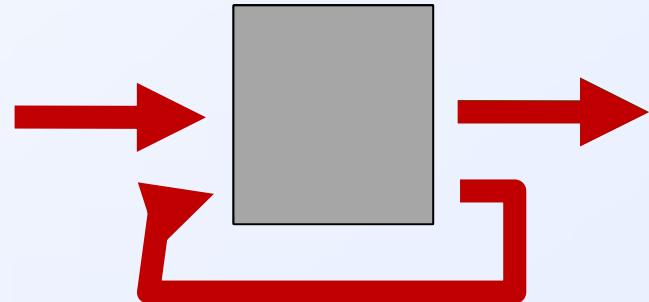
Circuiti combinatori e sequenziali

I circuiti **combinatori** sono in grado di calcolare funzioni che dipendono solo dai dati in input



I circuiti **sequenziali** sono invece in grado di calcolare funzioni che dipendono anche da uno *stato interno*, quindi dipendono anche da informazioni *memorizzate in elementi di memoria interni*

In generale, la funzione calcolata dal circuito ad un dato istante dipende dalla **sequenza temporale dei valori in input al circuito** (storia)



Macchine a Stati Finiti

Macchina (o *automa*):

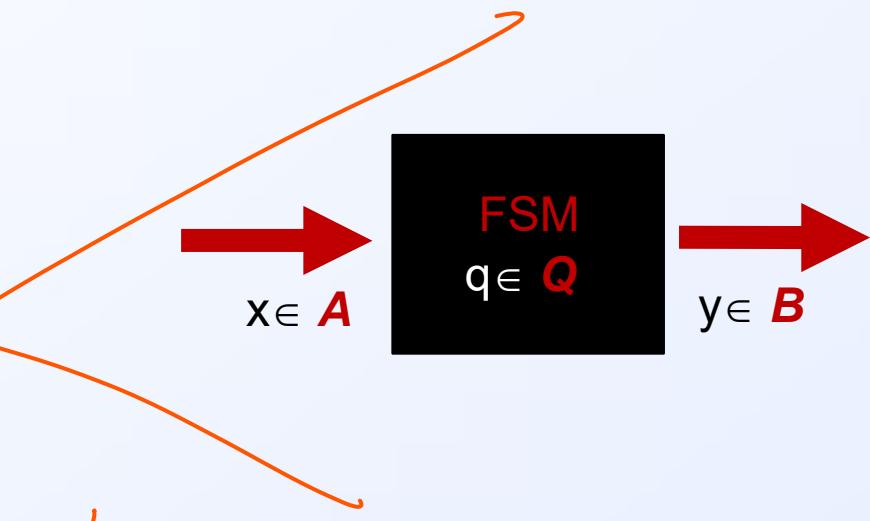
- dispositivo automatico in grado di interagire con l'ambiente esterno
- a fronte di uno stimolo in ingresso (*input*), esibisce un comportamento in uscita (*output*) che dipende anche da **informazioni memorizzate in elementi interni (stati)**.

Ci limitiamo a macchine con **memoria finita** (per macchine più potenti si ammette memoria illimitata ad. es. le *Macchine di Turing*)

Macchine a stati finiti

Una **macchina a stati finiti (FSM)**, è vista come una scatola “nera”, che possiamo descrivere mostrando cosa succede ad ogni passo:

- legge un simbolo in ingresso (che appartiene ad un insieme finito A)
- produce un simbolo in uscita (che appartiene ad un insieme finito B)
- cambia il proprio stato interno (la memoria è finita quindi l'insieme Q degli stati interni è finito)



Un esempio di Macchina a Stati Finiti

Un **distributore automatico** di biglietti



- accetta solo monete grandi (*MG*) e monete piccole (*MP*)
- un biglietto viene emesso quando vengono ricevute una *MP* ed una *MG*
- l'ordine di immissione delle monete non è rilevante
- non viene dato resto, è necessario sempre introdurre *MP* e *MG*
- *vengono restituite le monete “non adeguate”*

Un esempio di Macchina a Stati Finiti

- L'insieme (finito) dei simboli di ingresso è :

$$\mathbf{A} = \{MP, MG\}$$

- L'insieme (finito) dei simboli di uscita è

*(in-
es-
ito)* *q₁ + q₂ → emetti*

$$\mathbf{B} = \{ancora, restituisci, emetti\}$$

- L'insieme (finito) degli stati **Q** è formato da

- q0: non è stata inserita nessuna moneta
- q1: è stata inserita una MP
- q2: è stata inserita una MG



Un esempio di Macchina a Stati Finiti

distributore automatico di biglietti
Tabella di Transizione o Tabella di Stato



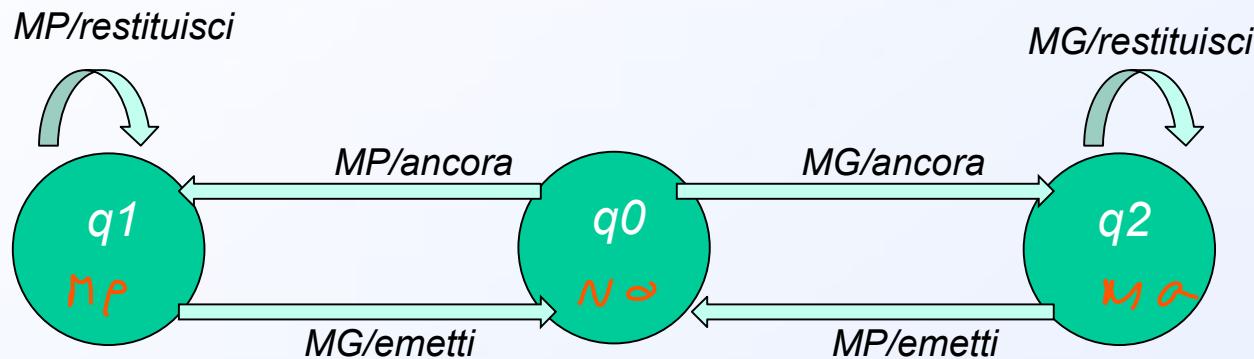
| | <i>MP</i> | <i>MG</i> |
|--------------------------------|-----------------------|-----------------------|
| <i>q0</i> <i>no moneta</i> | <i>q1/ancora</i> | <i>q2/ancora</i> |
| <i>q1</i> <i>M+</i> | <i>q1/restituisci</i> | <i>q0/emetti</i> |
| <i>q2</i> <i>MG</i> | <i>q0/emetti</i> | <i>q2/restituisci</i> |

L'insieme (finito) degli stati **Q** è formato da
q0: non è stata inserita nessuna moneta
q1: è stata inserita una MP
q2: è stata inserita una MG

Un esempio di Macchina a Stati Finiti

distributore automatico di biglietti

Grafo di Transizione (diagramma di stato)



L'insieme (finito) degli stati **Q** è formato da
q0: non è stata inserita nessuna moneta
q1: è stata inserita una MP
q2: è stata inserita una MG

Macchine a Stati Finiti

Una macchina (automa) a stati finiti \mathcal{M} è descritta da una quintupla

$$\mathcal{M} = \langle A, B, Q, o, s \rangle$$

- A : insieme finito di simboli di ingresso
- B : insieme finito di simboli di uscita
- Q : insieme finito di simboli di stato
- o : $A \times Q \rightarrow B$ funzione di uscita (per la macchina di Mealy) [o : $Q \rightarrow B$ (per la macchina di Moore)]
- s : $A \times Q \rightarrow Q$ funzione di cambiamento di stato

N.B. le funzioni o e s sono realizzate da reti logiche combinatorie

Esempio realizzazione distributore di biglietti

Codifica delle **tre** possibili uscite (2 bit)

| o_0 | o_1 | significato |
|-------------------------|-------------------------|--------------------|
| 0 | 0 | <i>ancora</i> |
| 0 | 1 | <i>emetti</i> |
| 1 | 0 | <i>restituisci</i> |
| 1 | 1 | -- |

Codifica dei **tre** possibili stati (2 bit)

| s_0 | s_1 | Significato |
|-------------------------|-------------------------|--------------------|
| 0 | 0 | q_0 |
| 0 | 1 | q_1 |
| 1 | 0 | q_2 |
| 1 | 1 | -- |

Codifica dei **due** possibili ingressi (1 bit)

| a | significato |
|----------|--------------------|
| 0 | <i>MP</i> |
| 1 | <i>MG</i> |

L'insieme (finito) degli stati **Q** è formato da
 q_0 : non è stata inserita nessuna moneta
 q_1 : è stata inserita una MP
 q_2 : è stata inserita una MG

Esempio realizzazione distributore di biglietti

Distributore di biglietti: funzioni o e s

divide solo in bit

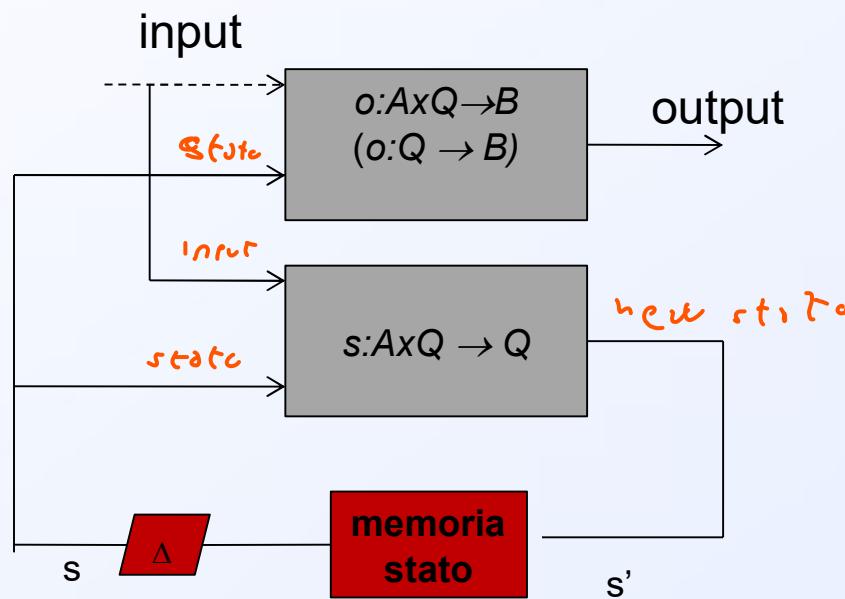
| ingressi | stato corrente | | | uscite | | stato futuro | | |
|--------------------------------|----------------|--------------|-------|--------|----------------|--------------|--------------|-------------|
| a <i>(clear monete)</i> | s_0 | <i>Scato</i> | s_1 | O_0 | <i>out put</i> | s'_0 | <i>Scato</i> | s'_1 |
| 0 | 0 | <i>void</i> | 0 | 0 | <i>sugars</i> | 0 | 0 | <i>H P</i> |
| 0 | 0 | <i>H P</i> | 1 | 1 | <i>resti</i> | 0 | 0 | <i>H P</i> |
| 0 | 1 | <i>H G</i> | 0 | 0 | <i>emett</i> | 1 | 0 | <i>void</i> |
| 0 | 1 | — | 1 | - | - | - | — | - |
| 1 | 0 | <i>void</i> | 0 | 0 | <i>suc</i> | 0 | 1 | <i>H a</i> |
| 1 | 0 | <i>H T</i> | 1 | 0 | <i>emett</i> | 1 | 0 | <i>void</i> |
| 1 | 1 | <i>H G</i> | 0 | 1 | <i>resti</i> | 0 | 1 | <i>H F</i> |
| 1 | 1 | — | 1 | - | - | - | — | - |

mf

MG

Realizzazione di Macchine a Stati Finiti:

Le Macchine a stati finiti sono implementate da reti logiche sequenziali in cui le funzioni o e s sono realizzate da reti logiche combinatorie, ma come memorizzare lo stato?



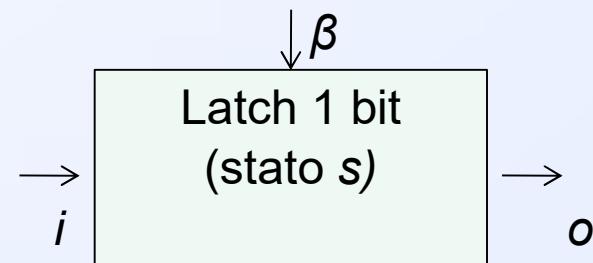
Macchine a Stati Finiti: come realizzare il dispositivo che memorizza lo stato?

Dispositivo a 1 bit (**latch** a 1 bit)

- due ingressi i e β ed un'uscita o
- mantiene uno stato interno s

se $\beta=1$ (*store*), $o \leftarrow s \leftarrow i$

se $\beta=0$ (*hold*), $o \leftarrow s$



Realizzazione di Macchine a Stati Finiti: latch a un bit

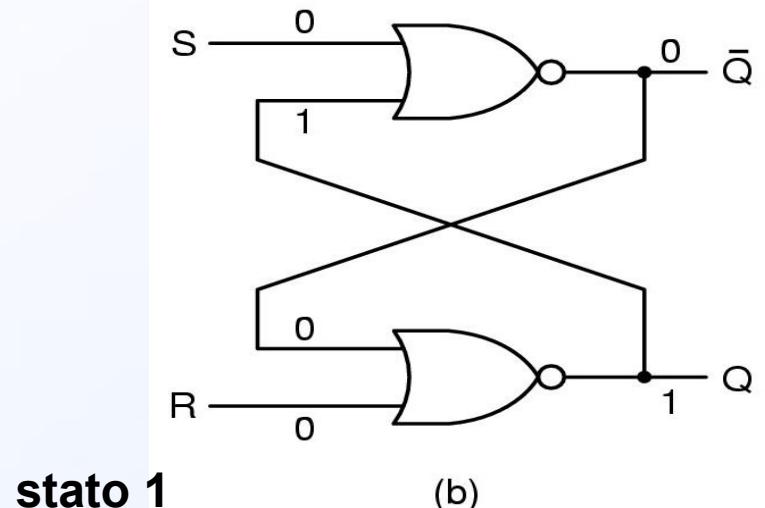
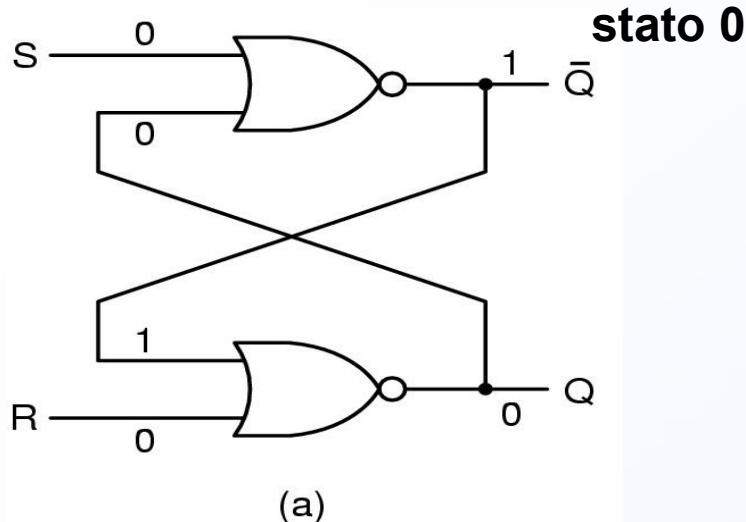
Le prime righe hanno $\beta=0$ (*hold*) per cui $s'=s$

Stato corrente Stato futuro

| β | i | s | $s'=o$ |
|---------|-----|-----|--------|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Le ultime righe hanno $\beta=1$ (*store*) per cui $s'=i$

Latch di tipo SR



Implementazione di un dispositivo di memoria ad 1 bit

Hold: $R = S = 0$ (due stati *stabili*)

Set (Store 1): $S = 1$ e $R = 0$ porta il latch allo stato 1

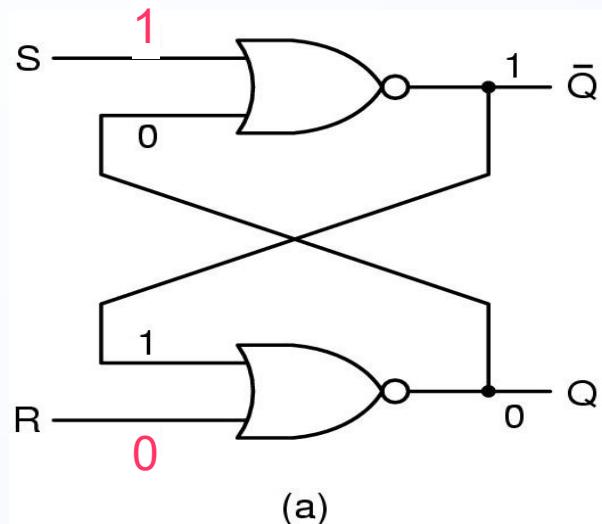
Reset (Store 0): $R = 1$ e $S = 0$ porta il latch allo stato 0

Il circuito memorizza qual è stato l'ultimo S o R

Rete corretta

Stato 0 -> Stato 1: $S = 1 \ R = 0$

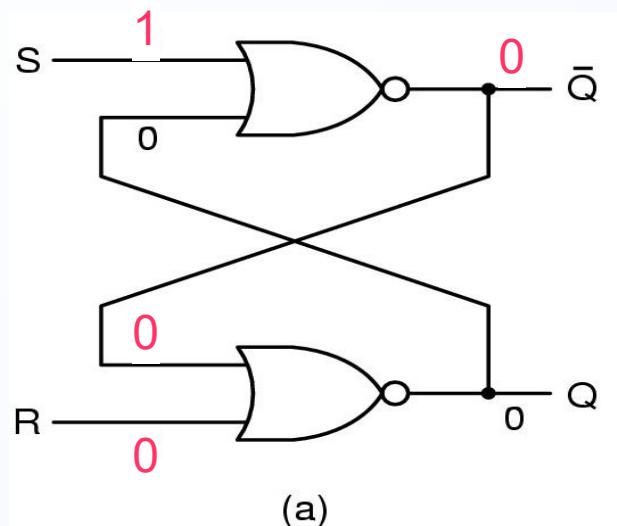
S (set) viene impostato ad uno, R lasciato a zero



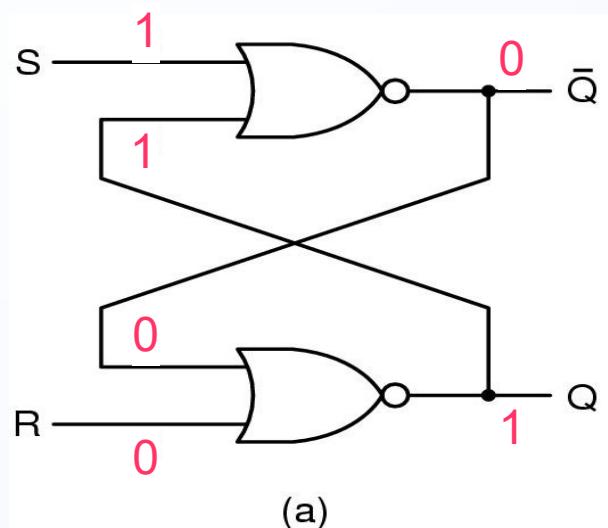
Stato 0

Stato 0 -> Stato 1: S = 1 R = 0

- Il valore in output del NOR in alto commuta, dal valore uno passa a zero
- Il valore zero, output per NOR in alto finisce come input del NOR in basso

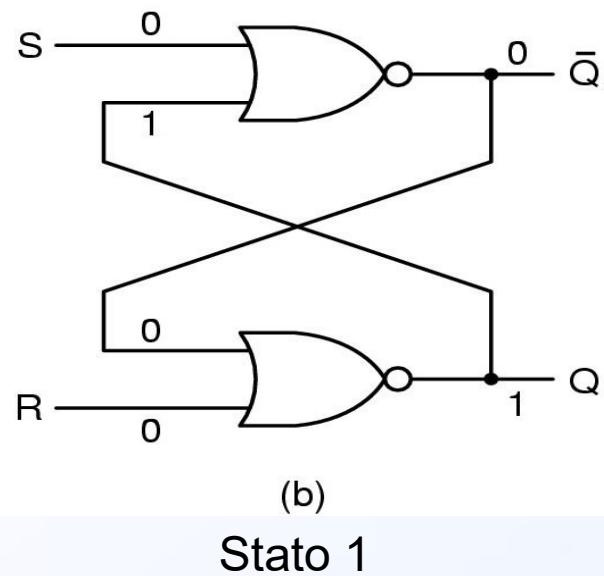


Stato 0 -> Stato 1: S = 1 R = 0



- Anche il valore del NOR in basso commuta, passando dal valore zero al valore uno
- Il nuovo valore del NOR in basso finisce anche come input del NOR in alto
- Il NOR in alto non commuta in presenza del nuovo input

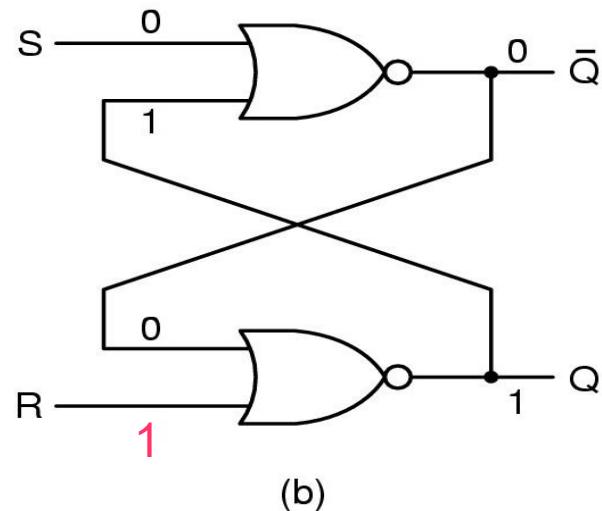
Stato 0 -> Stato 1: S = 0 R = 0



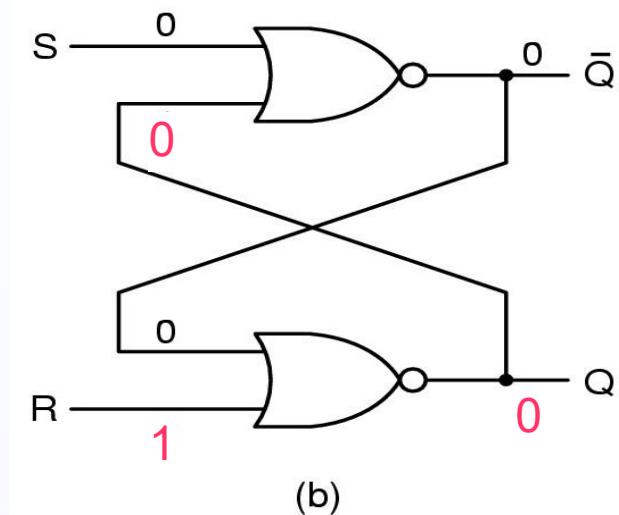
- Se ora il valore dell'input S torna a zero il circuito rimane stabile nel nuovo stato, che chiamiamo Stato 1 (l'output in Q è uno, in \bar{Q} è zero)
- Se S venisse riportato a uno lo stato rimarrebbe sempre lo stesso, è uno stato stabile

Stato 1 -> Stato 0: S = 0 R = 1

- Se vogliamo tornare nello Stato 0 ($Q = 0$ e $\bar{Q} = 1$) è necessario impostare R a uno



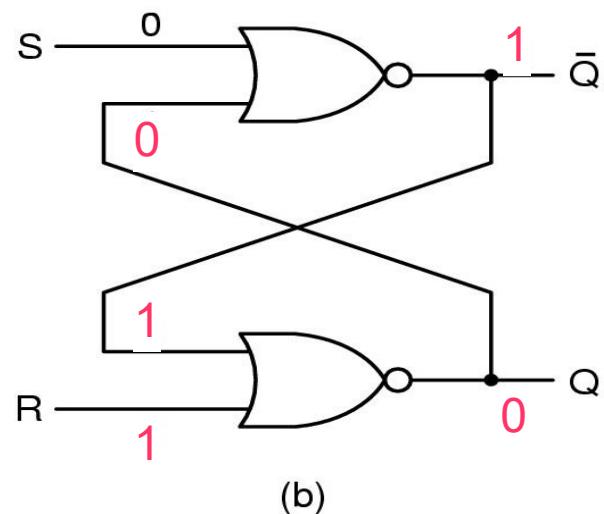
Stato 1 -> Stato 0: S = 0 R = 1



- Il NOR in basso commuta, l'output ora vale zero, questo finisce anche come input del NOR in alto

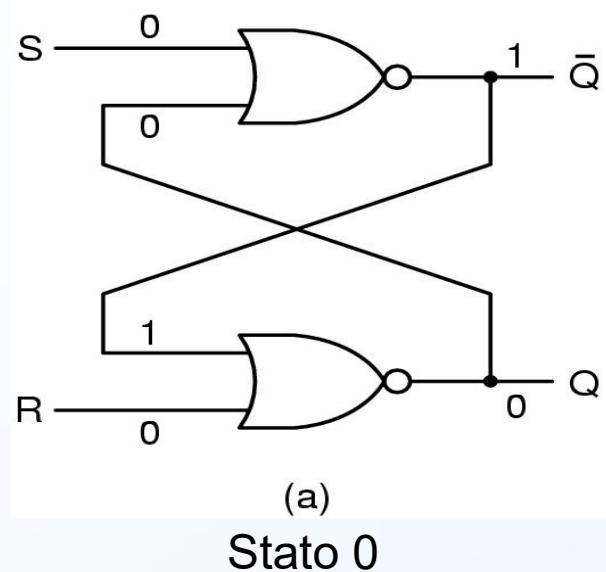
Stato 1 -> Stato 0: S = 0 R = 1

- L'output del NOR in alto vale ora uno, questo è anche uno degli input del NOR in basso che non cambia di output



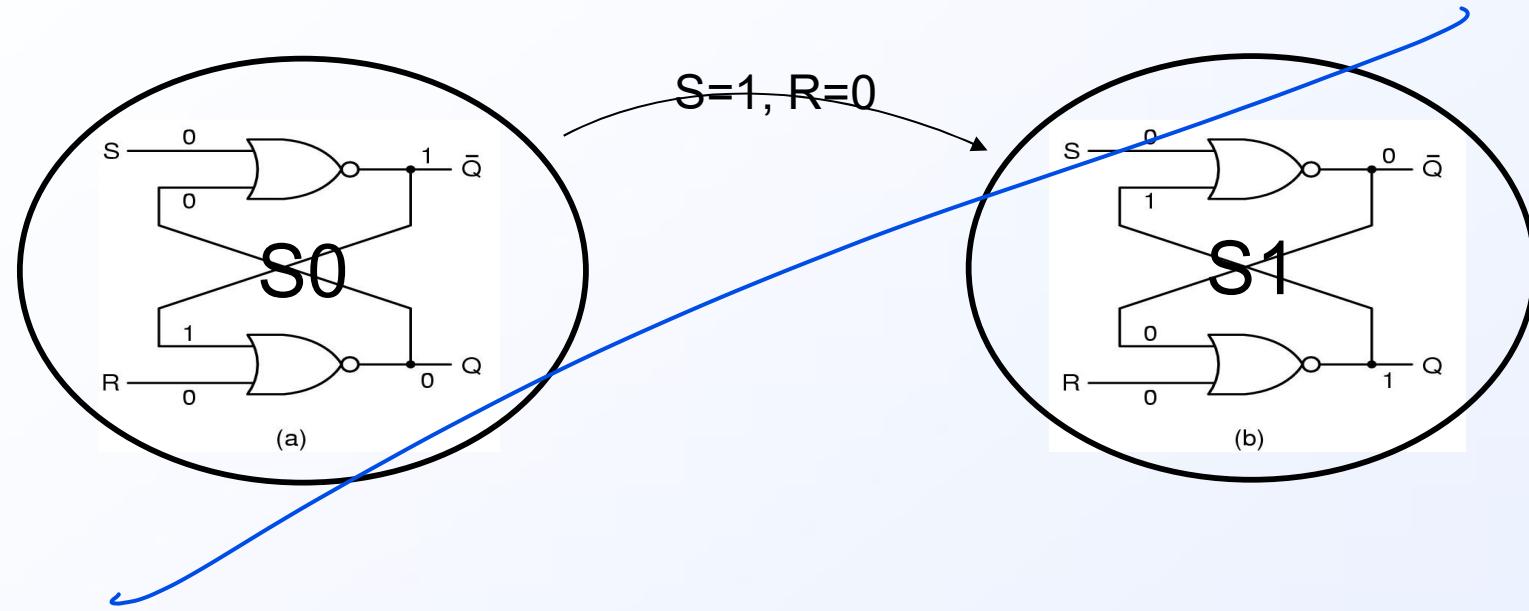
Stato 1 -> Stato 0: S = 0 R = 0

- Il circuito mantiene la sua impostazione anche se il valore di R torna ad essere zero, è uno stato stabile

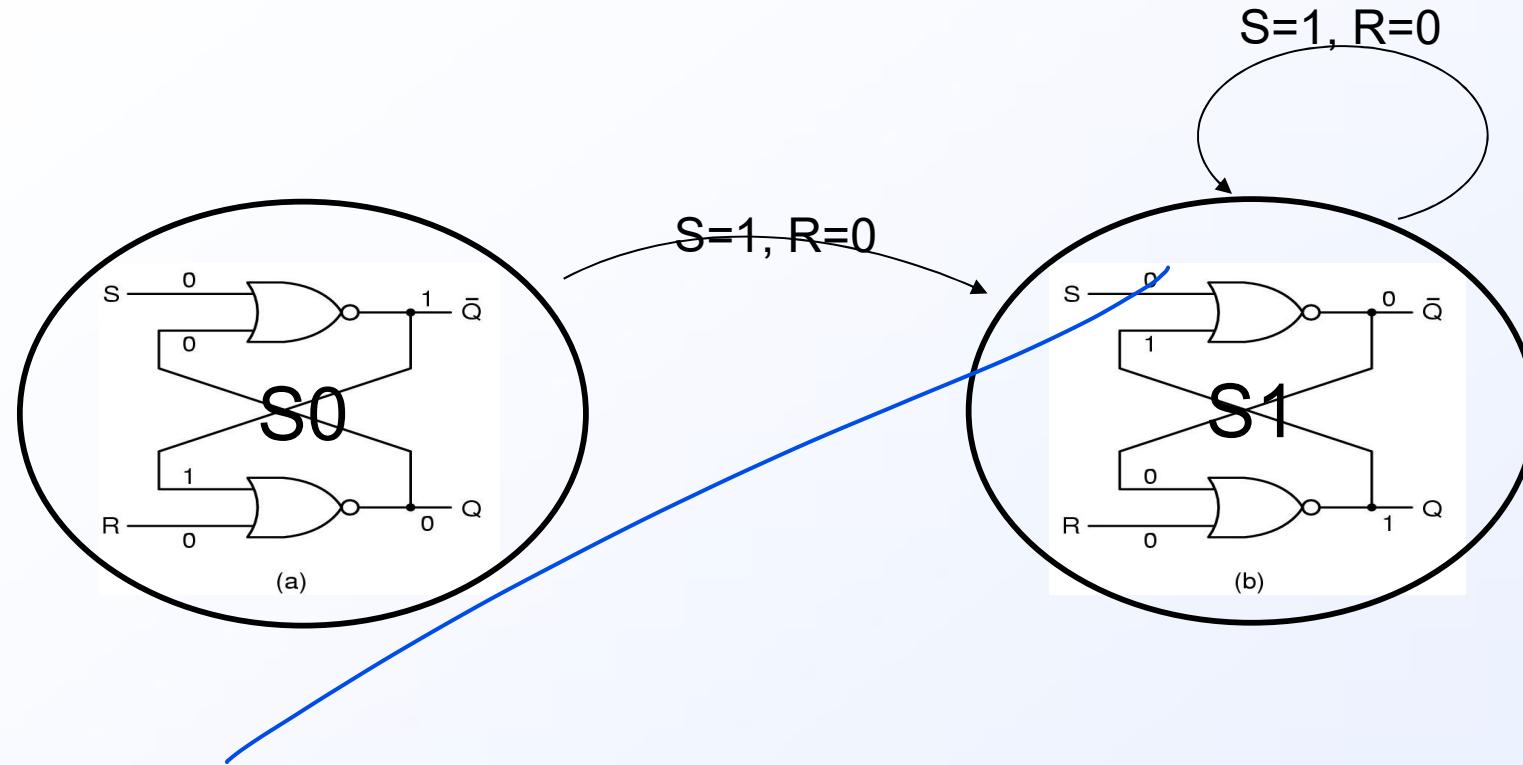


L

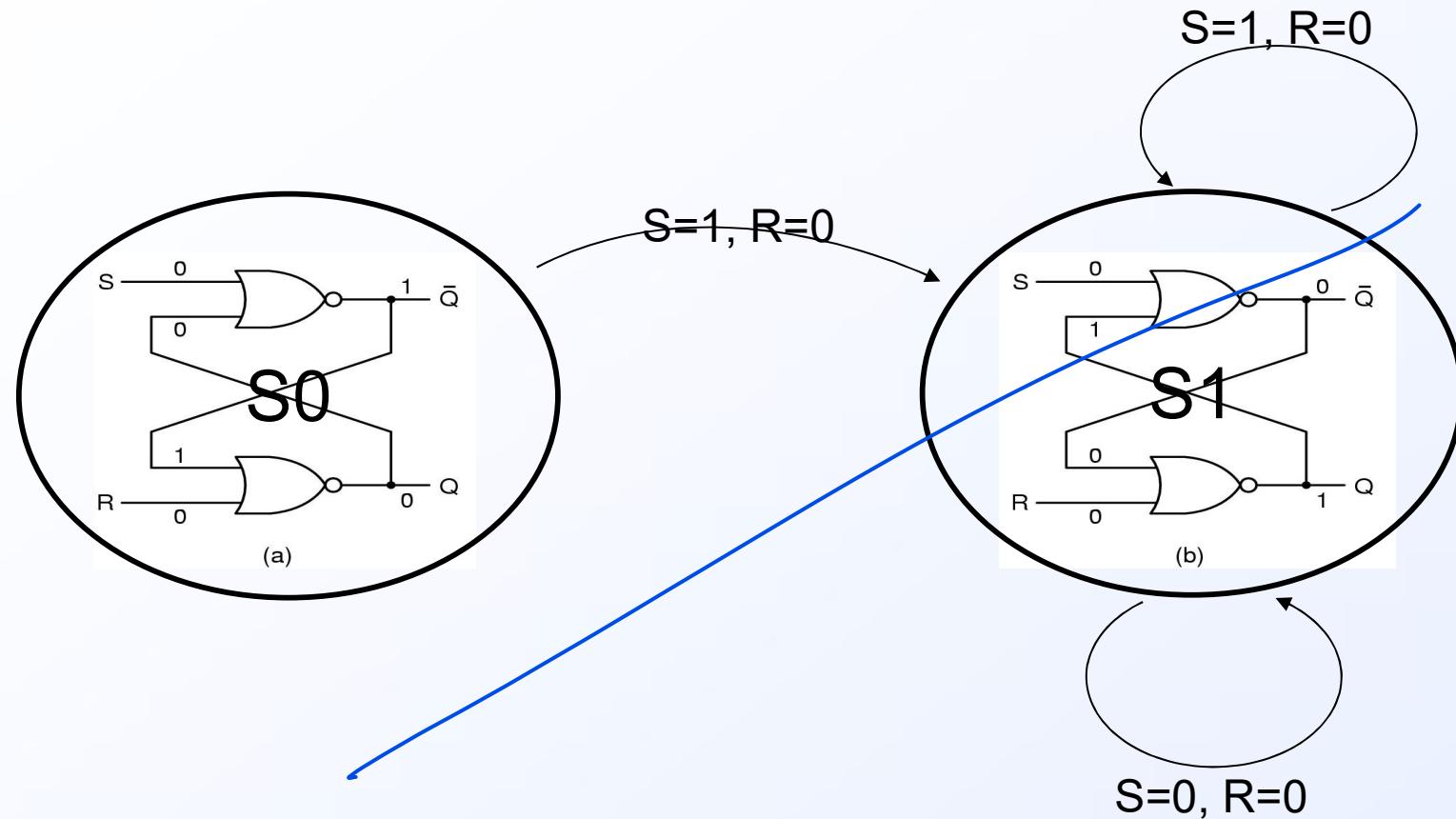
Latch SR: automa di transizione



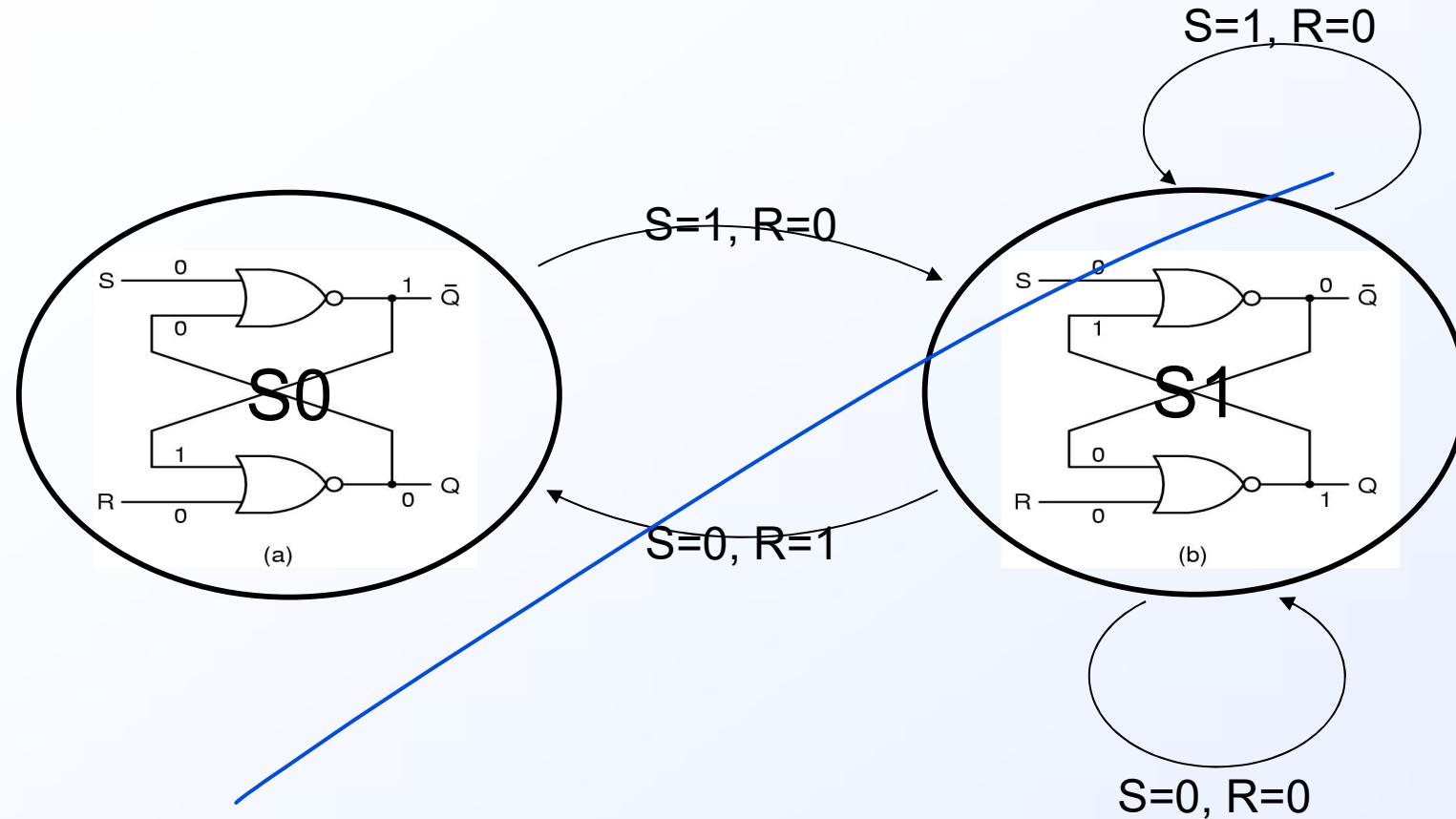
Latch SR: automa di transizione



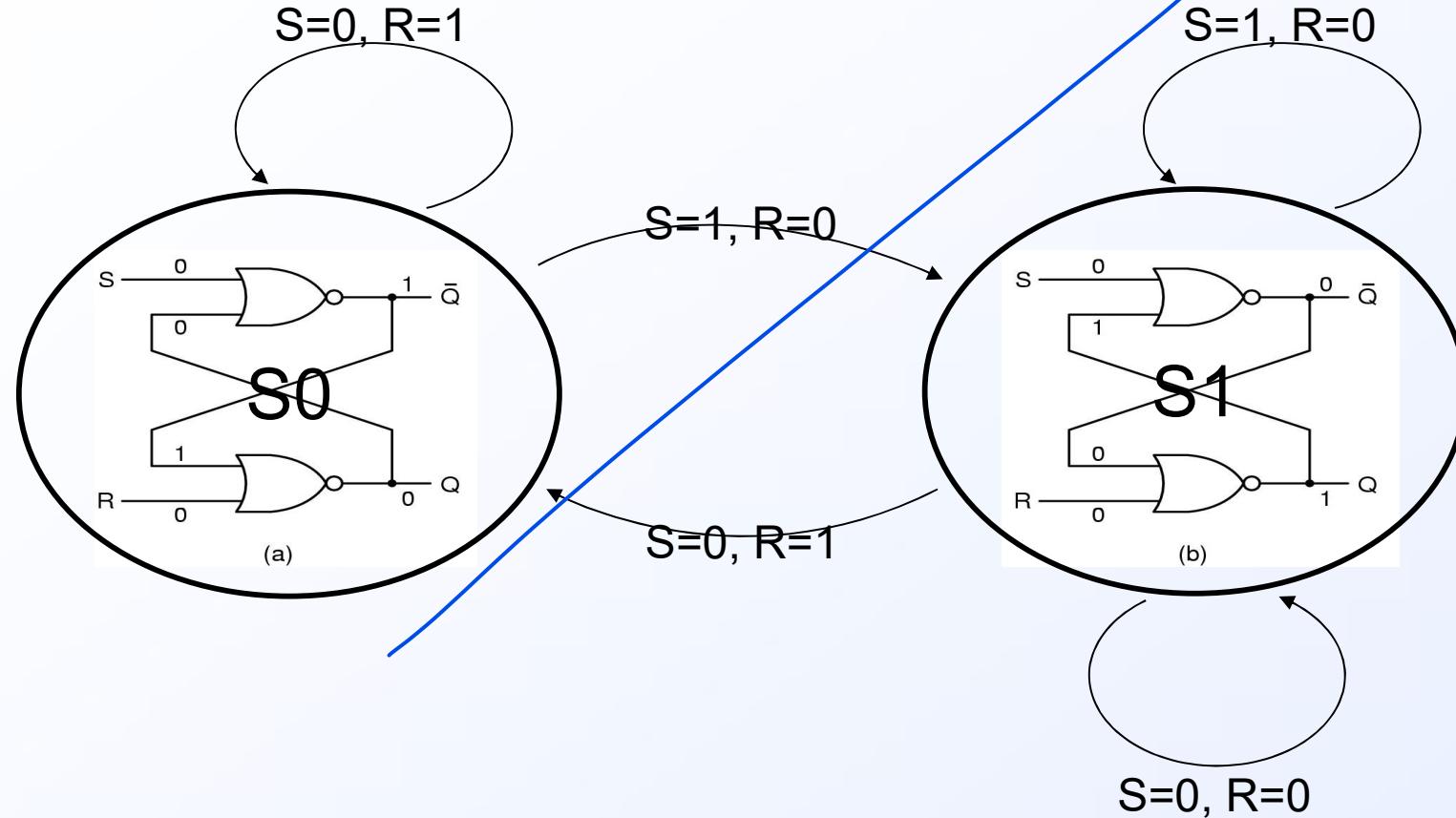
Latch SR: automa di transizione



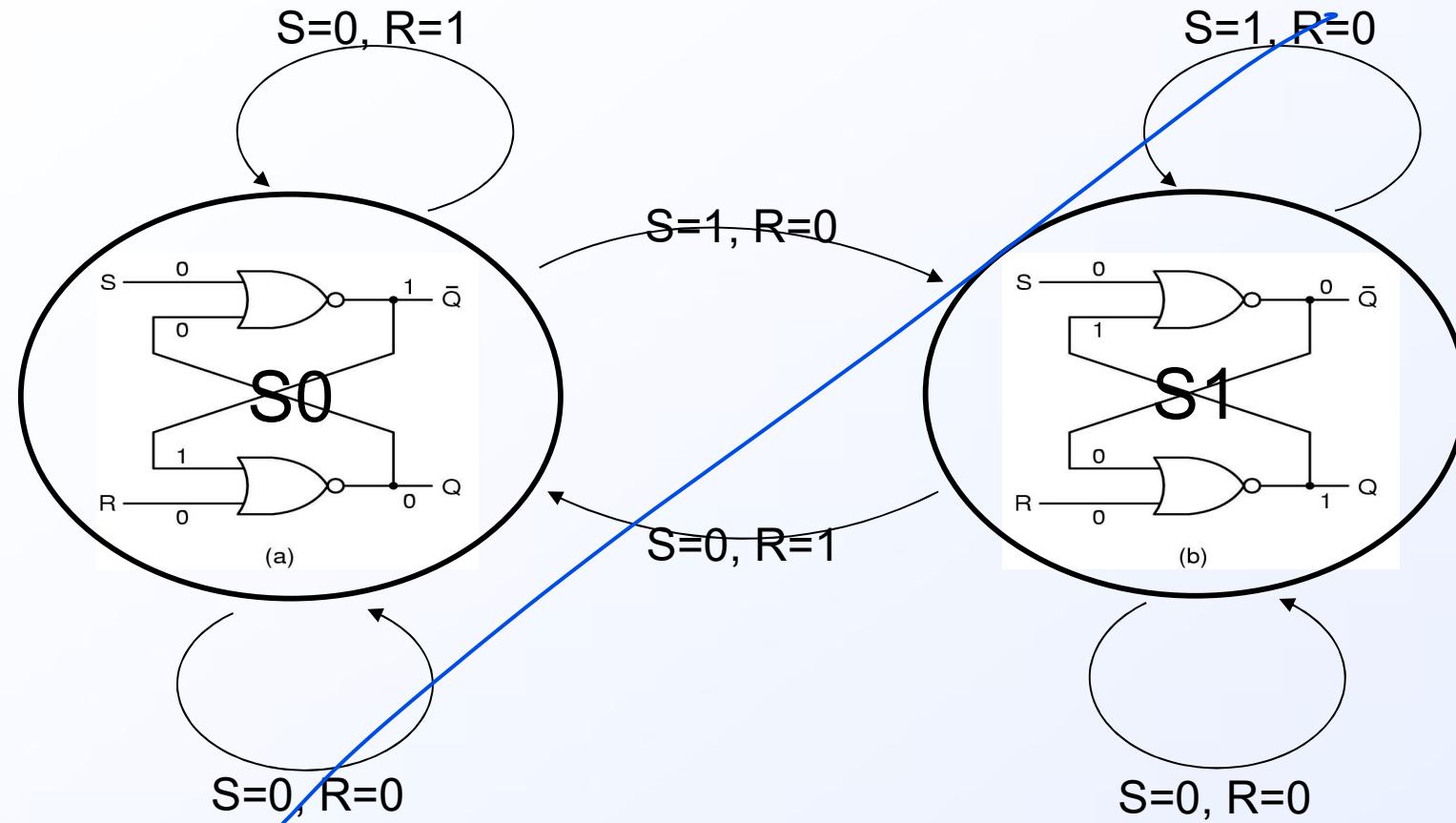
Latch SR: automa di transizione



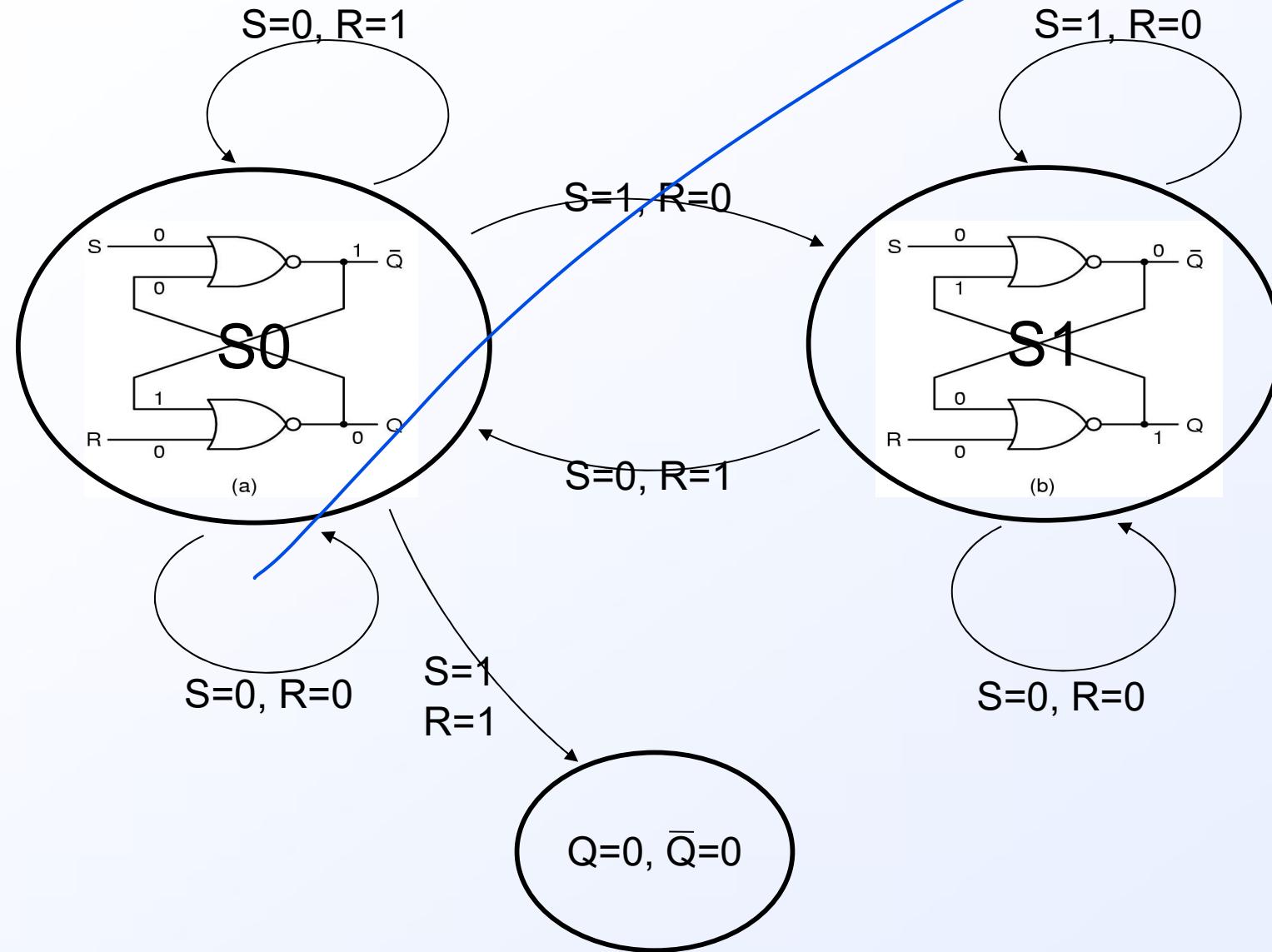
Latch SR: automa di transizione



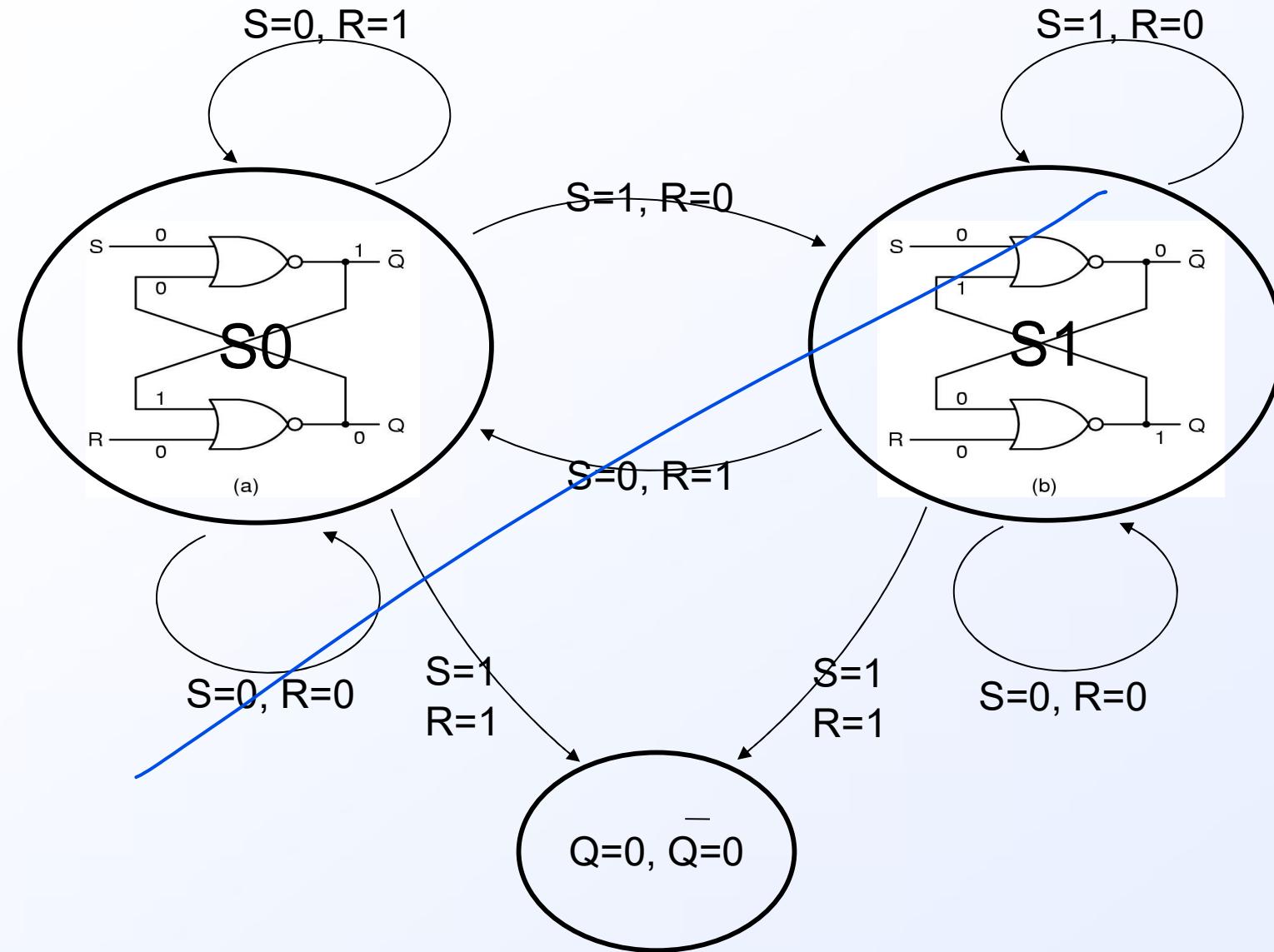
Latch SR: automa di transizione



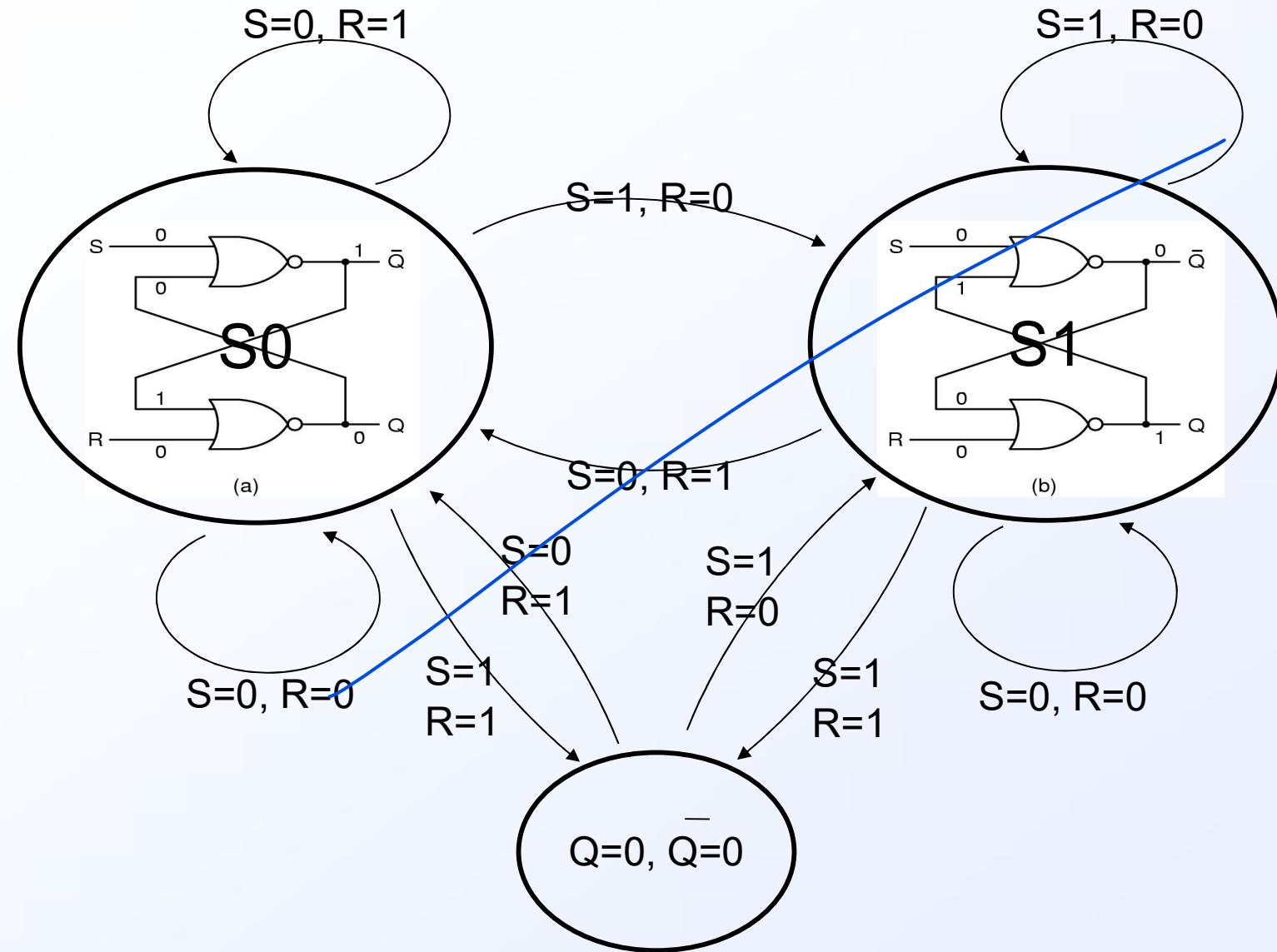
Latch SR: automa di transizione



Latch SR: automa di transizione

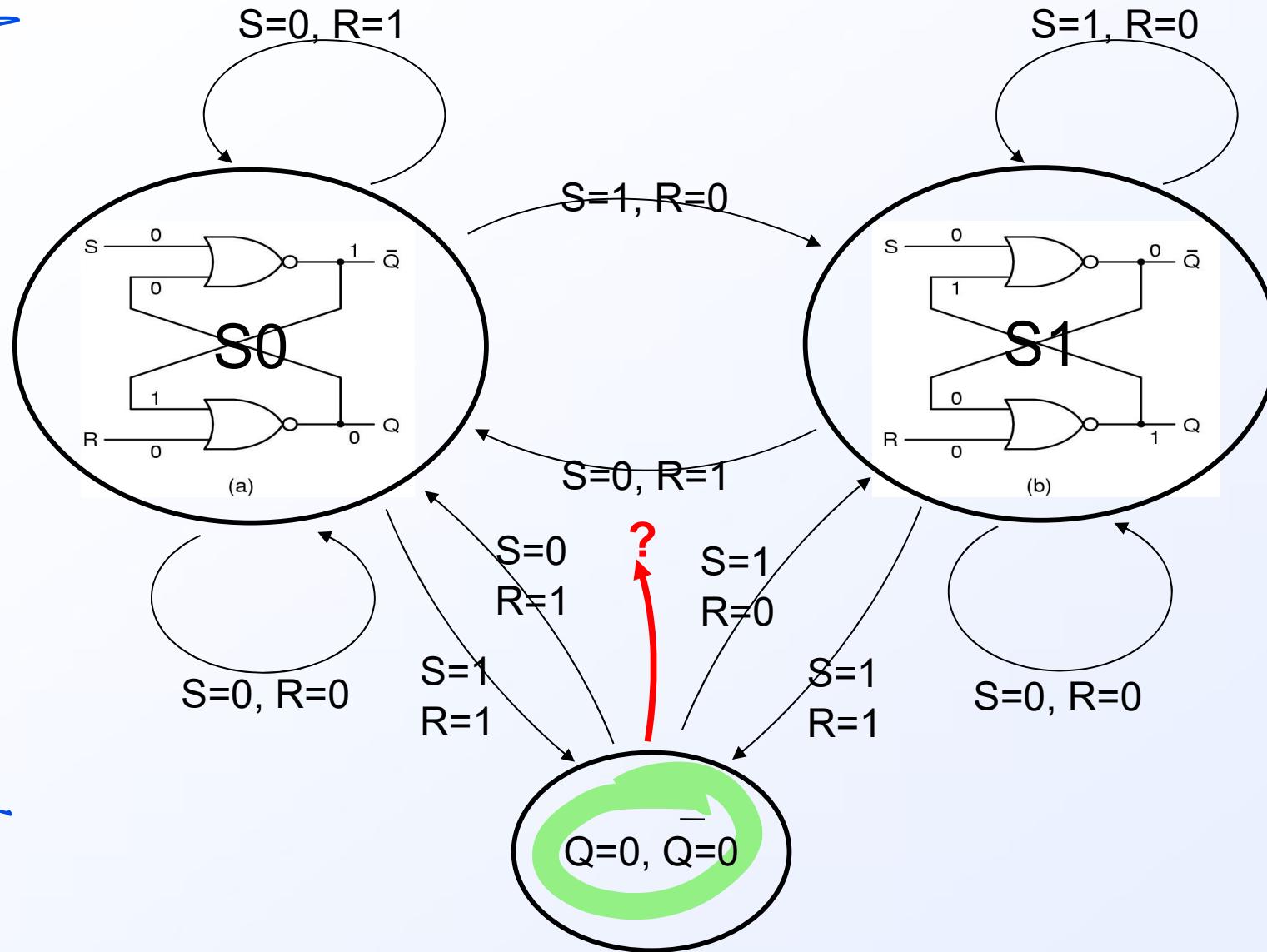


Latch SR: automa di transizione

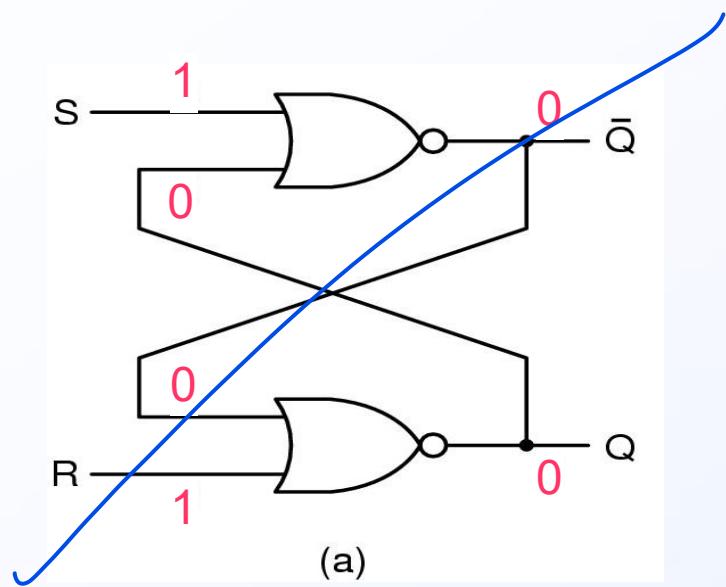


Latch SR: automa di transizione

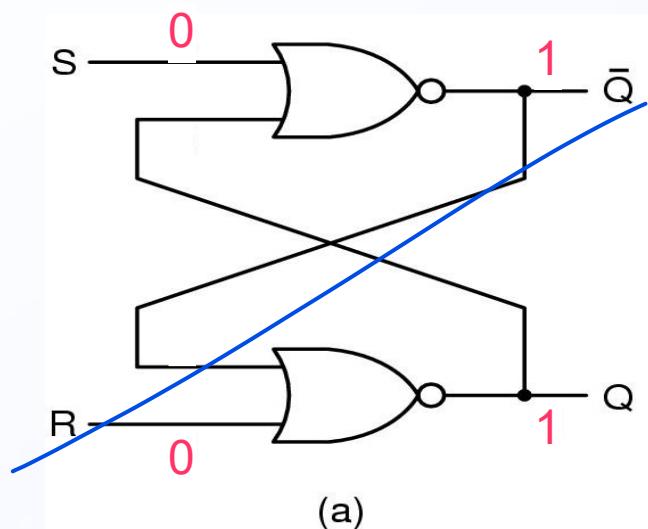
diagr. imp



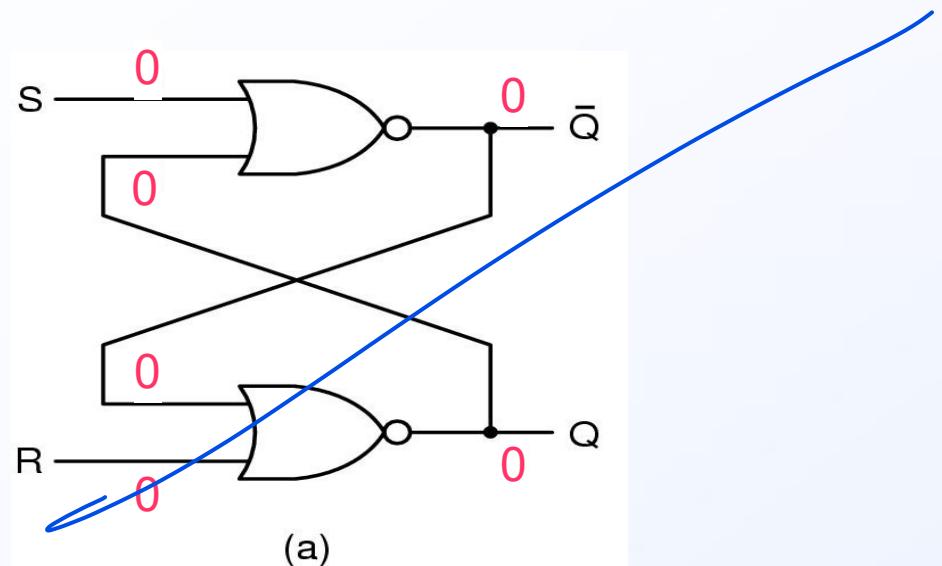
$$S = 1 \ R = 1$$



$$S = 1 \ R = 1$$

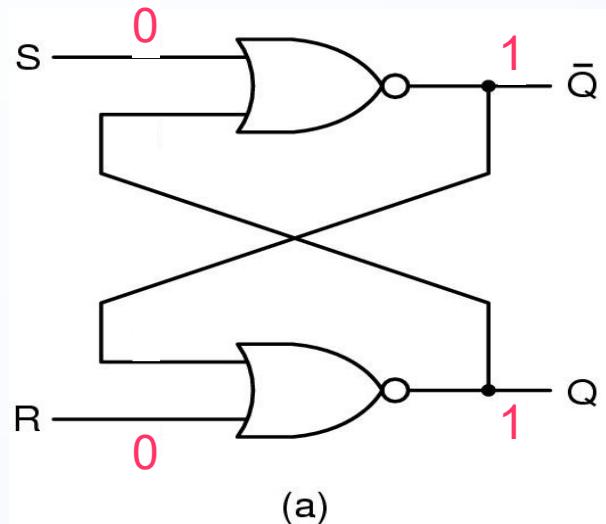


$$S = 1 \ R = 1$$



P1

$$S = 1 \ R = 1$$



→ $\bar{Q} = Q = 0$

er ricasiva

1 h mondo ideale

$$S = 1 \ R = 1$$

S=R=1 lo stato non è stabile:

- Q impredicibile
- Possibile oscillazione

S=R=1 non deve mai accadere



- Anche in fenomeni di transitorio
- Si potrebbe innescare un'oscillazione

$$S = 1 \quad R = 1$$

I segnali **S** e **R** non possono essere contemporaneamente uguali a 1 per poter memorizzare un valore corretto

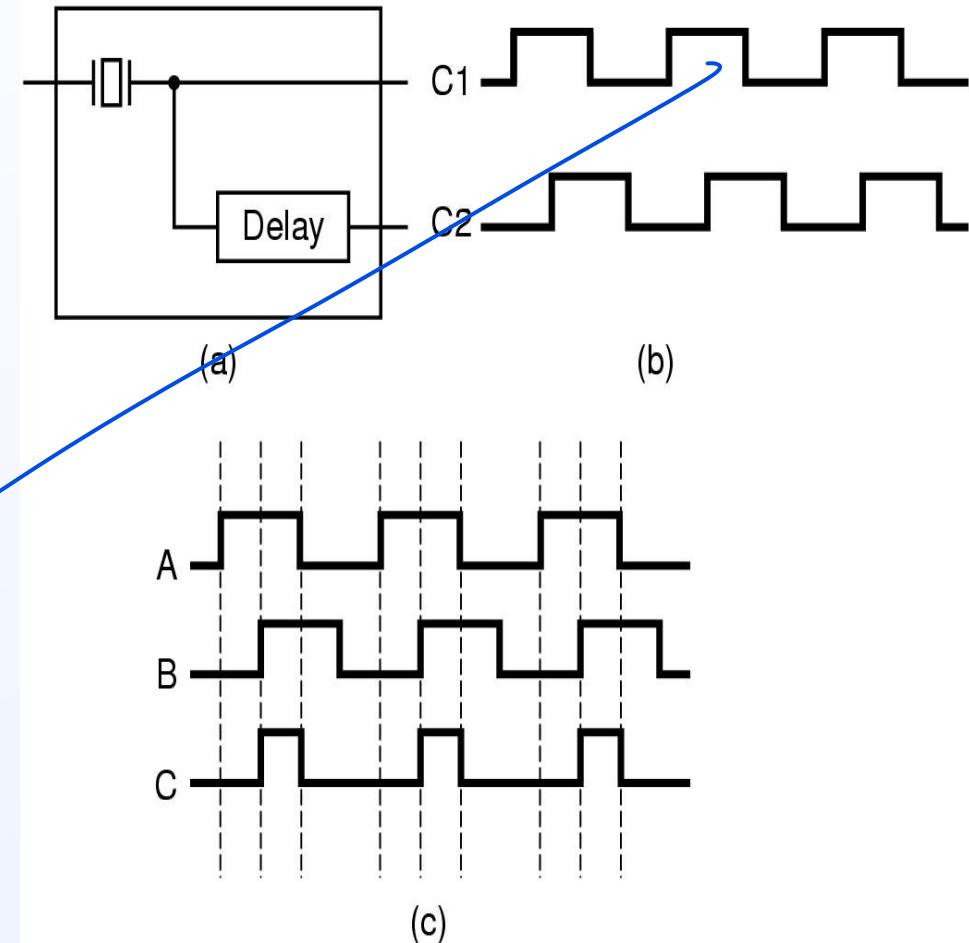
Ma (**S**, **R**) sono di solito calcolati da un *circuito combinatorio*

- l'output del circuito diventa **stabile** dopo un certo intervallo di tempo
- è possibile calcolare questo intervallo tempo, che dipende dal numero di porte attraversate e dal ritardo delle porte
- bisogna evitare che durante questo intervallo gli output intermedi del circuito vengano presentati al latch per la memorizzazione (altrimenti possono essere memorizzati valori errati)

Soluzione : usiamo un segnale di **clock** !

Clock

- **Clock:** un circuito che emette una serie di impulsi con una specifica larghezza e intermittenza
- **Tempo di ciclo di clock:** intervallo fra i fronti corrispondenti di due impulsi consecutivi
- Fronte di salita di C1, fronte di discesa di C1, fronte di salita di C2, fronte di discesa di C2

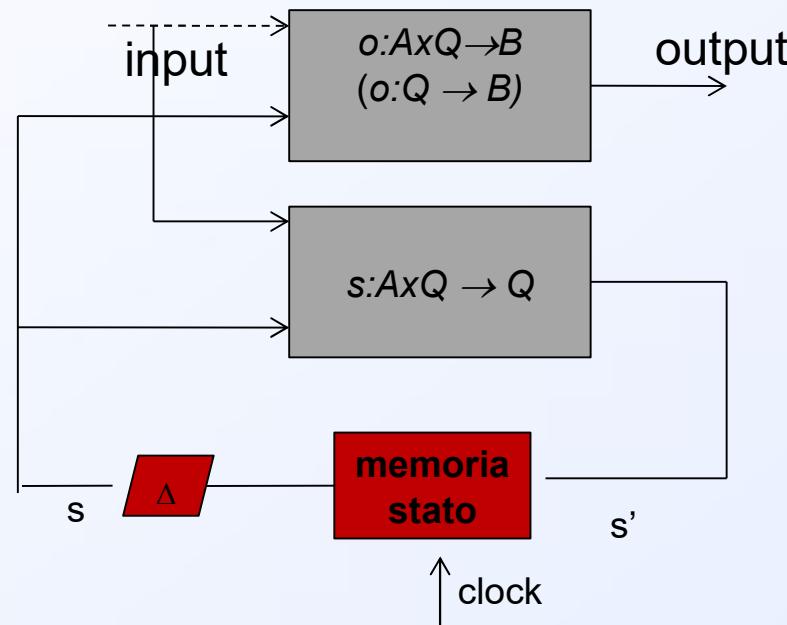


500 MHz = 2 nsec di tempo
di ciclo di clock

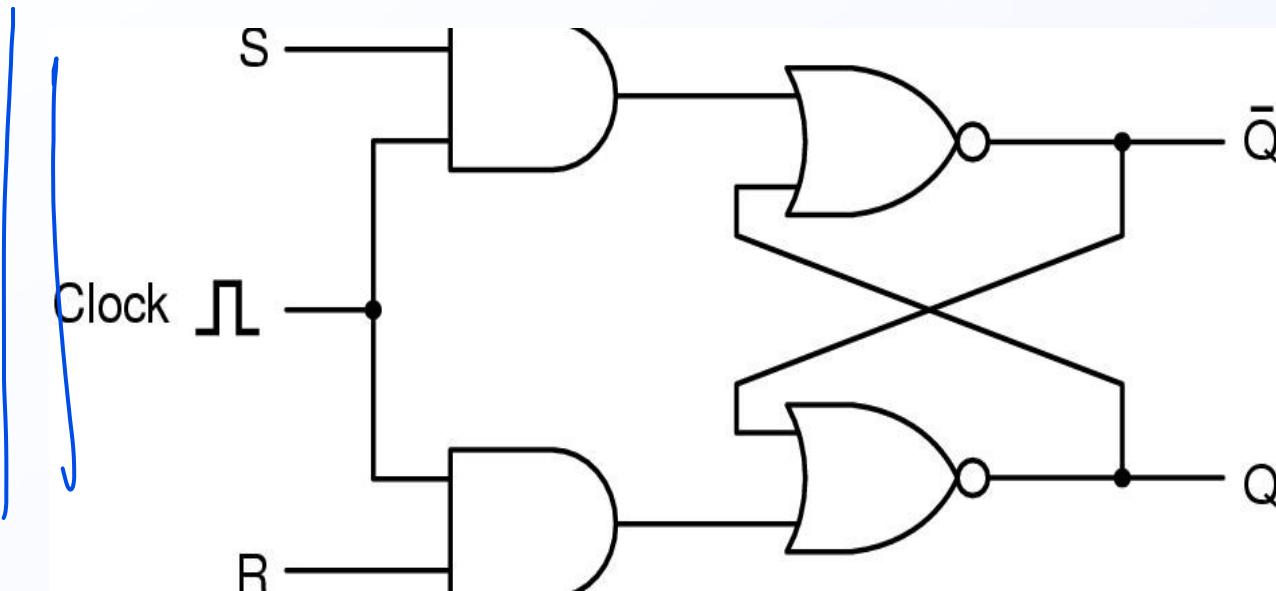
Circuiti sequenziali sincroni

Soluzione : scegliamo il periodo del segnale di **clock** in modo che sia abbastanza grande da assicurare la stabilità degli output del circuito

- usiamo il clock per abilitare la scrittura nei latch (cambio di stato)
- il clock determina il *ritmo dei calcoli e delle relative operazioni di memorizzazione*: il circuito sequenziale viene detto **sincrono**

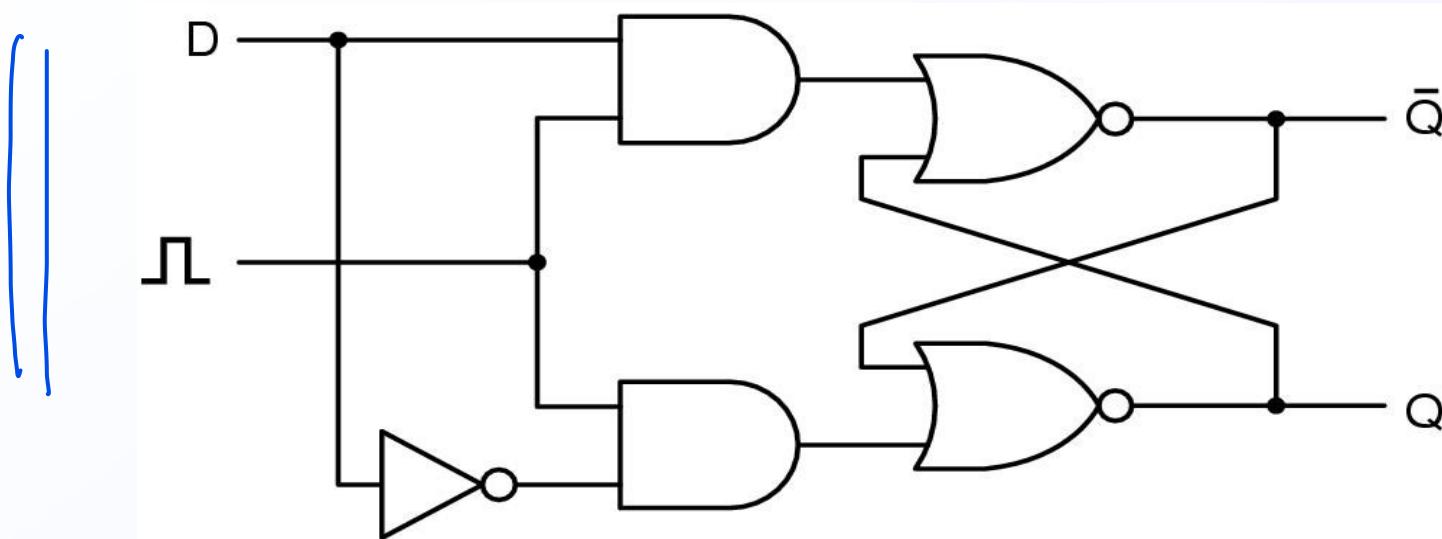


Latch di tipo SR sincronizzato



- Un clock garantisce che il latch cambi stato solo in certi momenti specifici
- Le porte AND abilitano gli input S e R solo quando il clock è a 1

Latch di tipo D sincronizzato

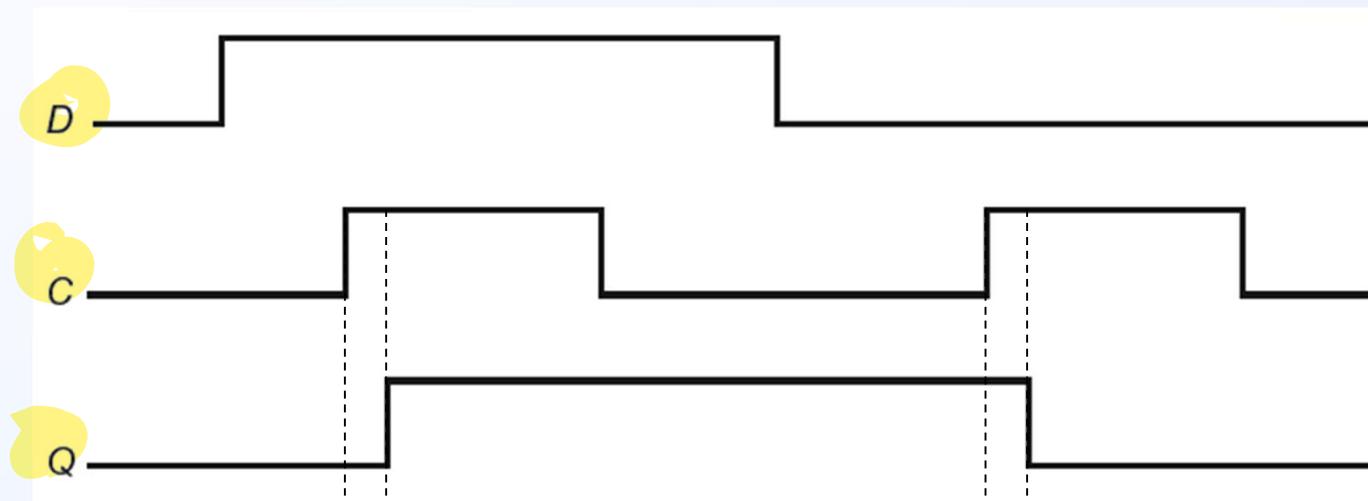
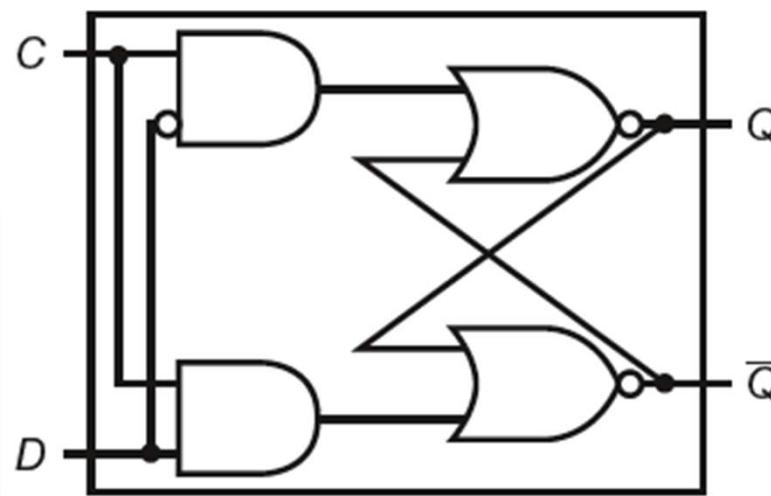


Memorizza il valore che D assume mentre il clock è 1:
se D varia mentre il clock è 1, varia anche lo stato

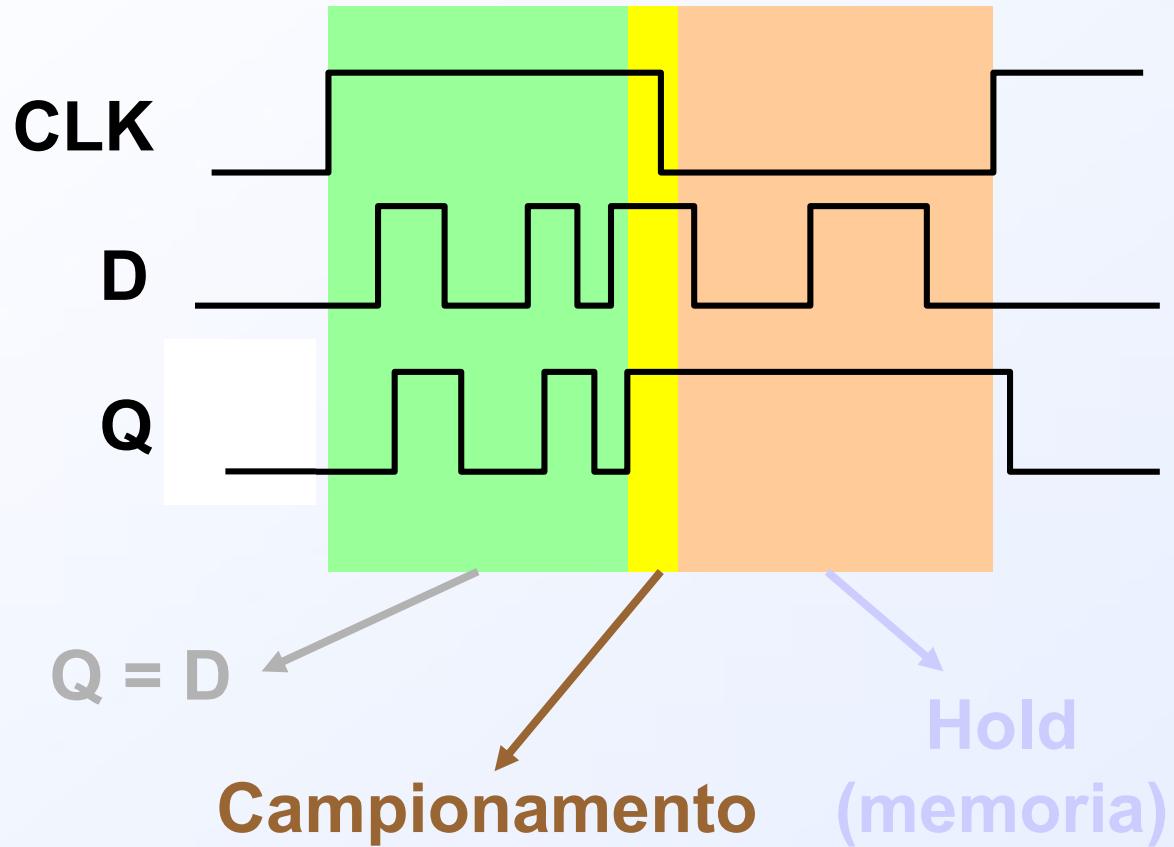
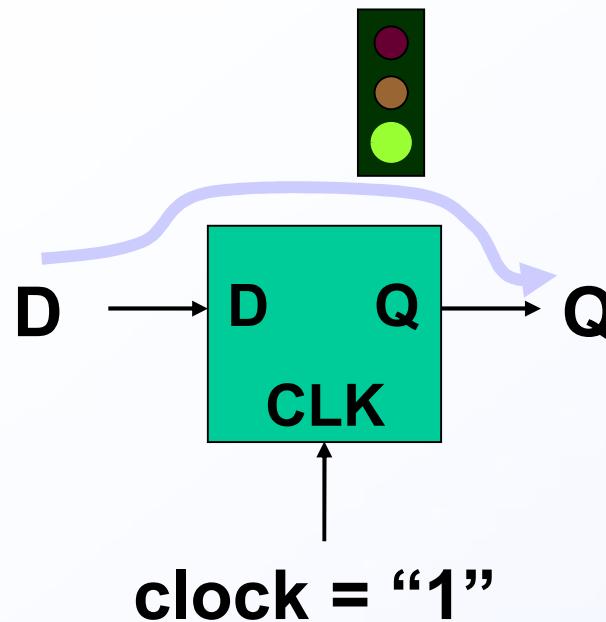
Il latch D evita l'ambiguità dello stato S=1, R=1:

- $D = 1$ e $\text{clock} = 1 \Rightarrow$ “stato 1”,
- $D = 0$ e $\text{clock} = 1 \Rightarrow$ “stato 0”

Latch di tipo D sincronizzato

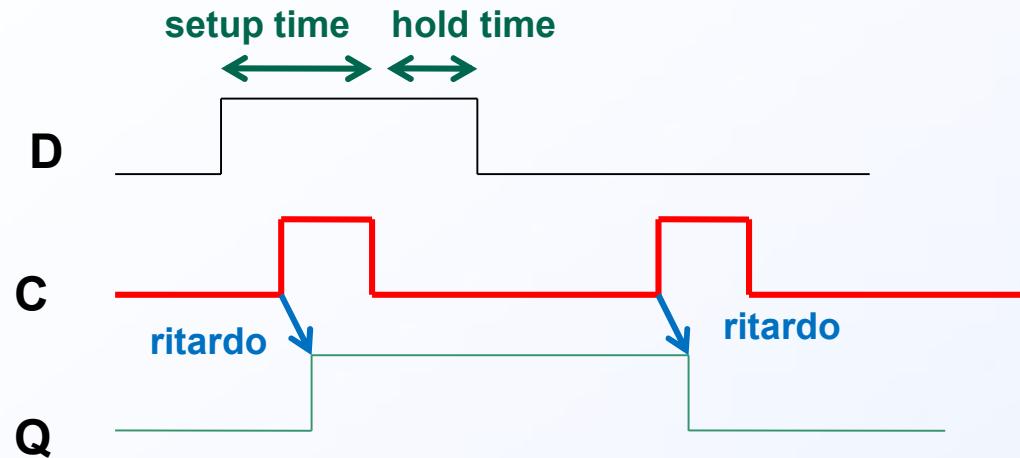


Il problema della “trasparenza”



Con $CLK=1$ il latch è “trasparente”, ossia l’uscita segue l’ingresso istante per istante (anche se con un ritardo dovuto al tempo di propagazione attraverso il latch)

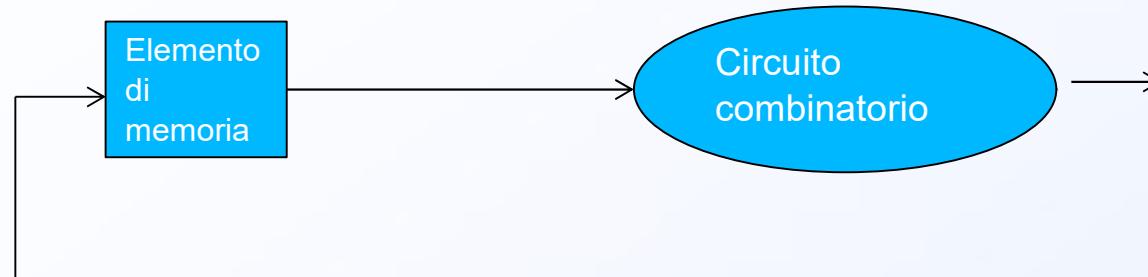
Diagramma temporale del D-latch



Il segnale D, ottenuto solitamente come output di un circuito combinatorio

- deve essere già stabile quando C diventa alto
- deve rimanere stabile per tutta la durata del livello alto di C (**setup time**)
- deve infine rimanere stabile per un altro periodo di tempo per evitare malfunzionamenti (**hold time**)

Elemento di memoria usato come input e output



Durante ogni periodo di clock il circuito combinatorio di sopra dovrebbe calcolare una **funzione sulla base del vecchio valore** dell'elemento di memoria (stato del circuito):

- l'output calcolato dovrebbe diventare il **nuovo valore** da memorizzare nell'elemento di memoria (**nuovo stato** del circuito)
- il **nuovo stato** dovrebbe essere usato come **input** del circuito durante il ciclo di clock successivo

Quindi l'elemento di memoria deve essere usato sia come input e sia come output durante lo stesso ciclo di clock.

Il D-latch funzionerebbe in questo caso ?

No!! , perché se il clock rimane alto per molto tempo, allora un eventuale **segnale di ritorno sporco**, proveniente dal circuito combinatorio potrebbe essere memorizzato nel latch.

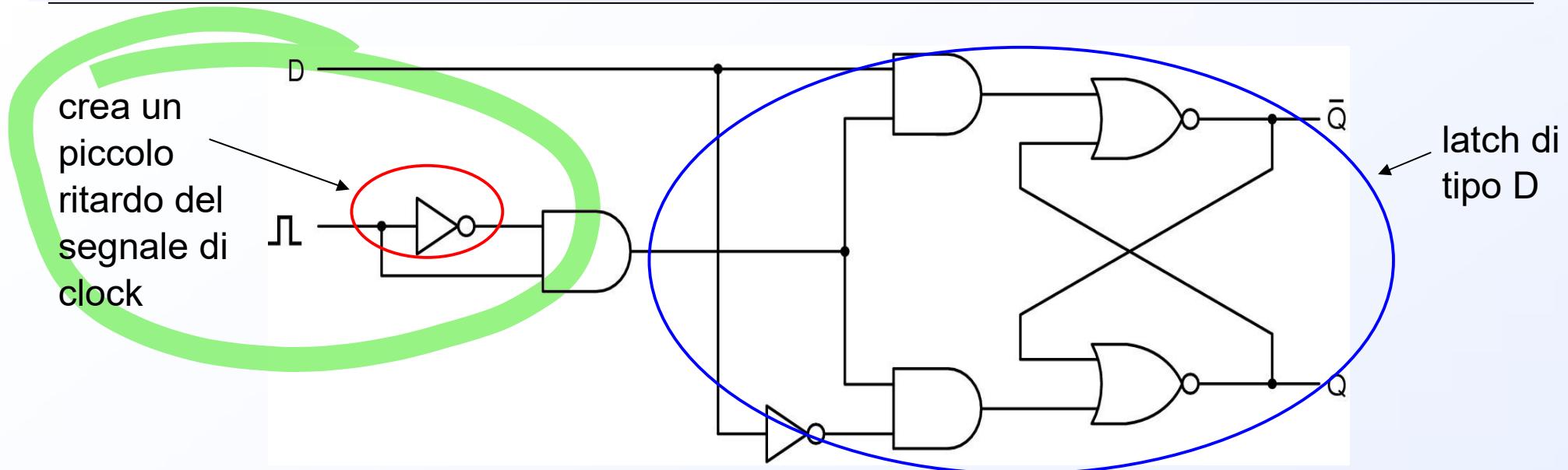
Diagramma temporale del D-latch



Si possono progettare componenti di memoria, in cui la memorizzazione può avvenire a diversi istanti rispetto al segnale a gradino del clock:

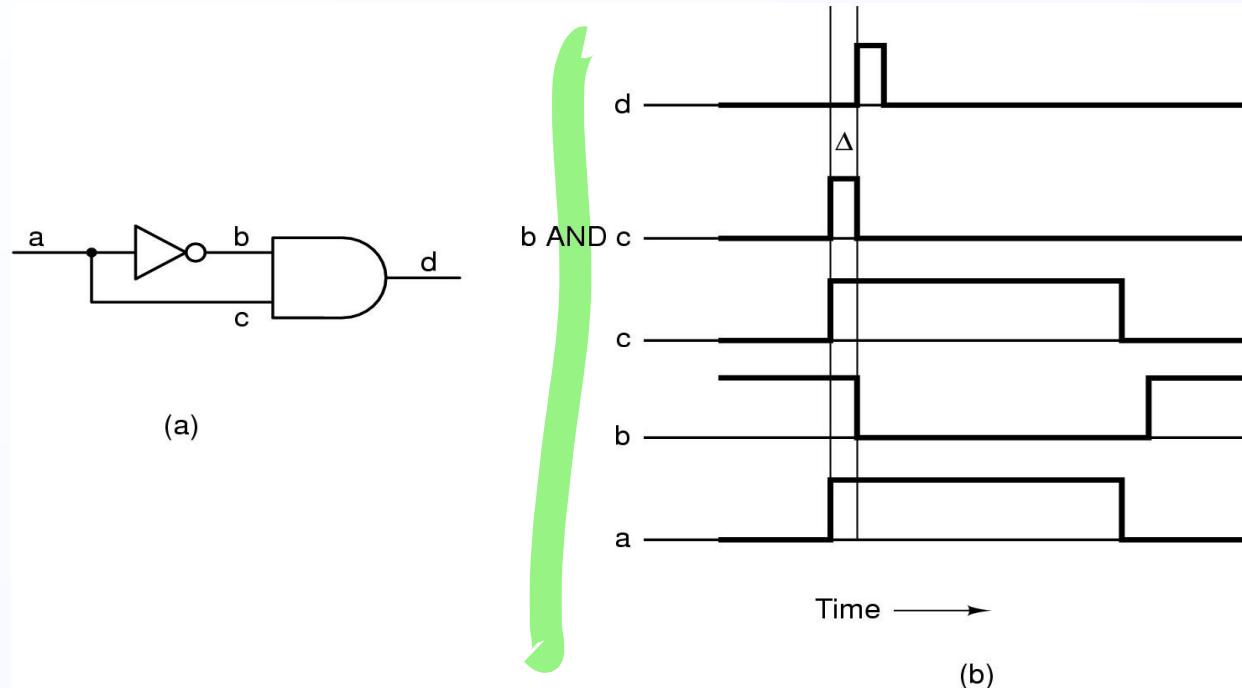
- metodologia con **commutazione a livello**: la commutazione avviene sul livello alto (o basso) del clock. Il D-latch è di questo tipo.
- metodologia con **commutazione sul fronte**: la commutazione avviene sul fronte di salita (o di discesa) del clock. Possiamo immaginare che la memorizzazione avvenga *istantaneamente*, e che l'eventuale *segnale di ritorno sporco, proveniente dal circuito combinatorio*, non faccia in tempo ad arrivare a causa dell'istantaneità della memorizzazione: è quello che ci serve !!

Flip-flop di tipo D



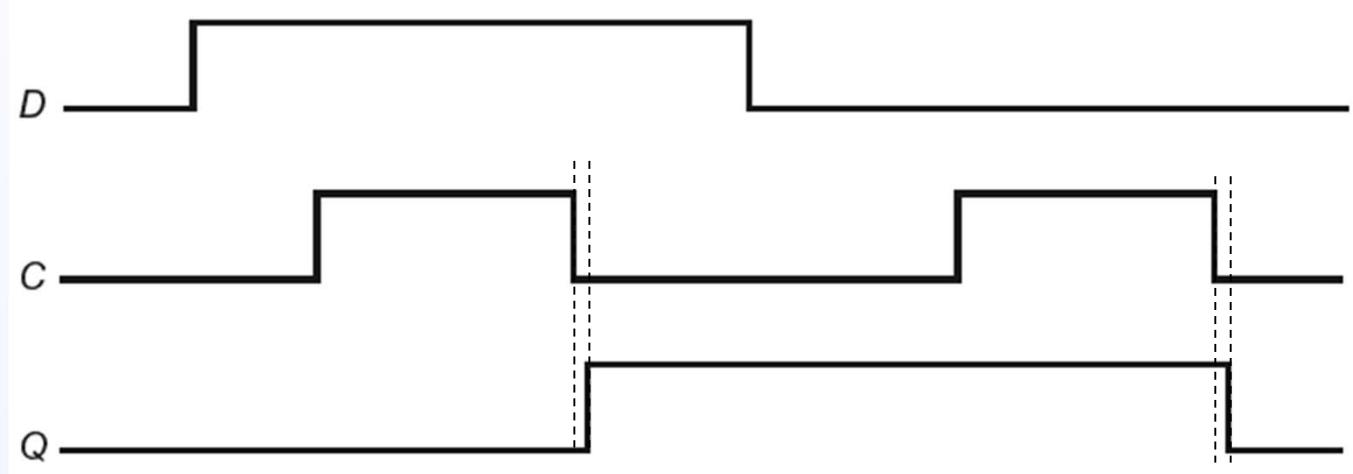
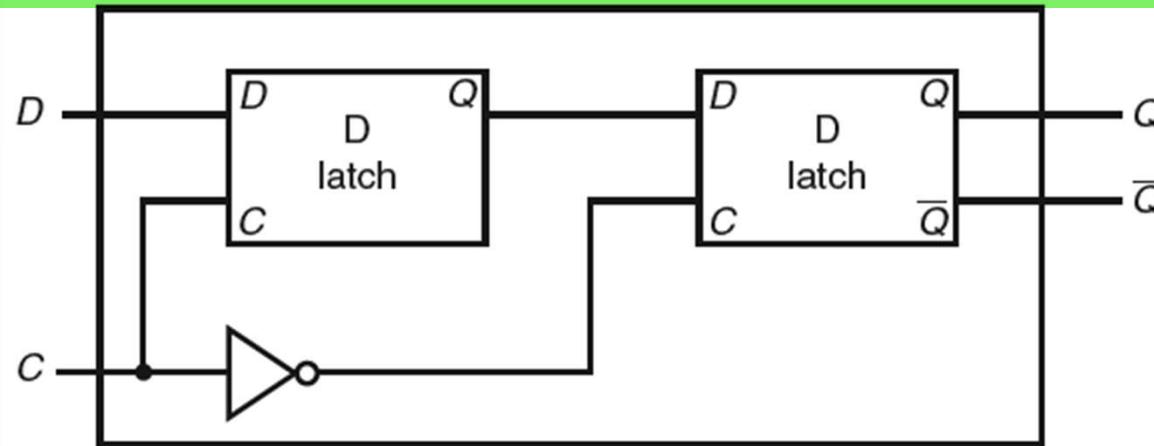
- Un latch è azionato dal livello (commutazione a livello)
- Un flip-flop è azionato dal fronte (commutazione sul fronte)
- La lunghezza dell'impulso di clock non è importante

Flip-flop di tipo D

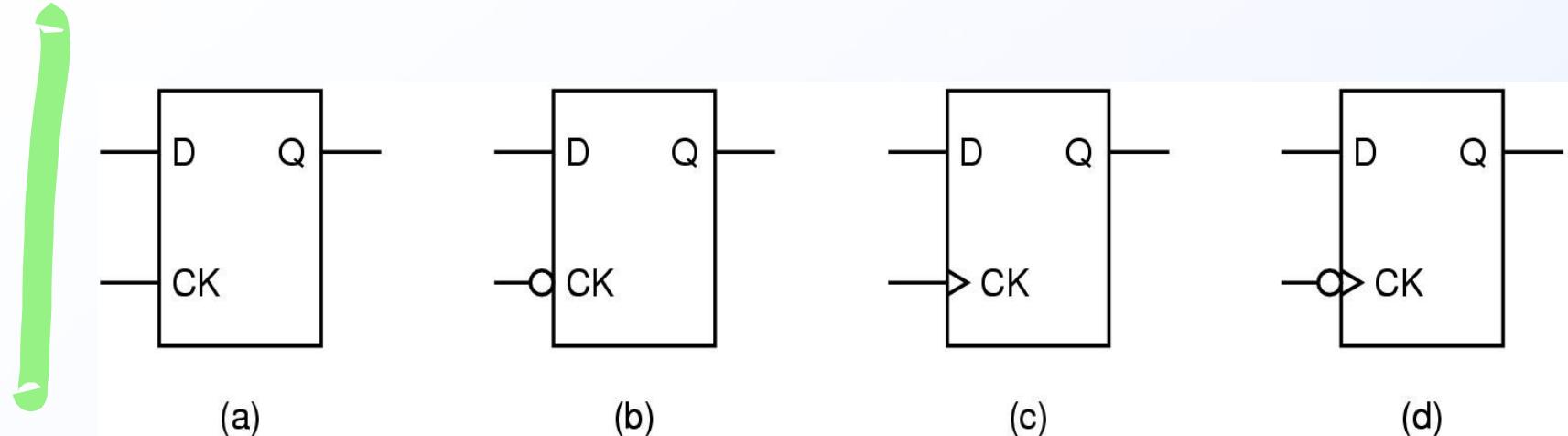


- L'invertitore crea un piccolo ritardo alla propagazione del segnale **a** verso **b**
- Il latch D verrà attivato ad un ritardo fisso dopo il fronte di salita del clock (per l'attraversamento dell'AND)

Flip-flop di tipo D sul fronte di discesa

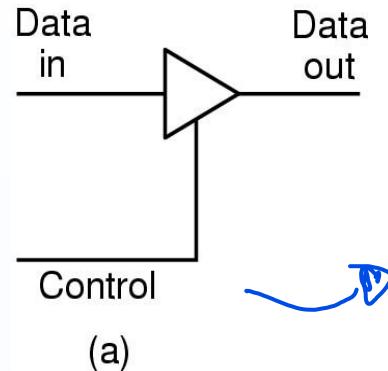


Flip-Flop



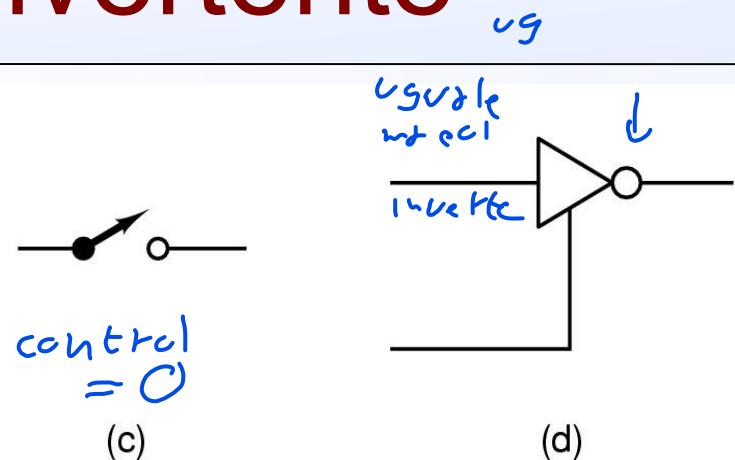
- (a) latch di tipo D attivato con livello 1 del clock
- (b) latch di tipo D attivato con livello 0 del clock
- (c) flip-flop di tipo D attivato sul fronte di salita del clock
- (d) flip-flop di tipo D attivato sul fronte di discesa del clock

Buffer (non) invertente



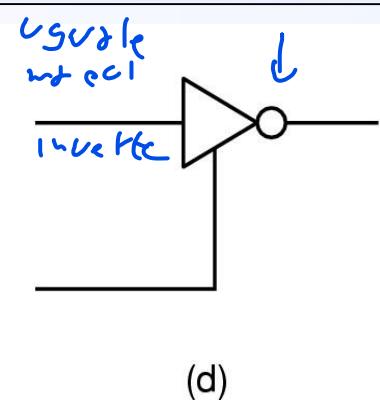
control
= 1

(b)



control
= 0

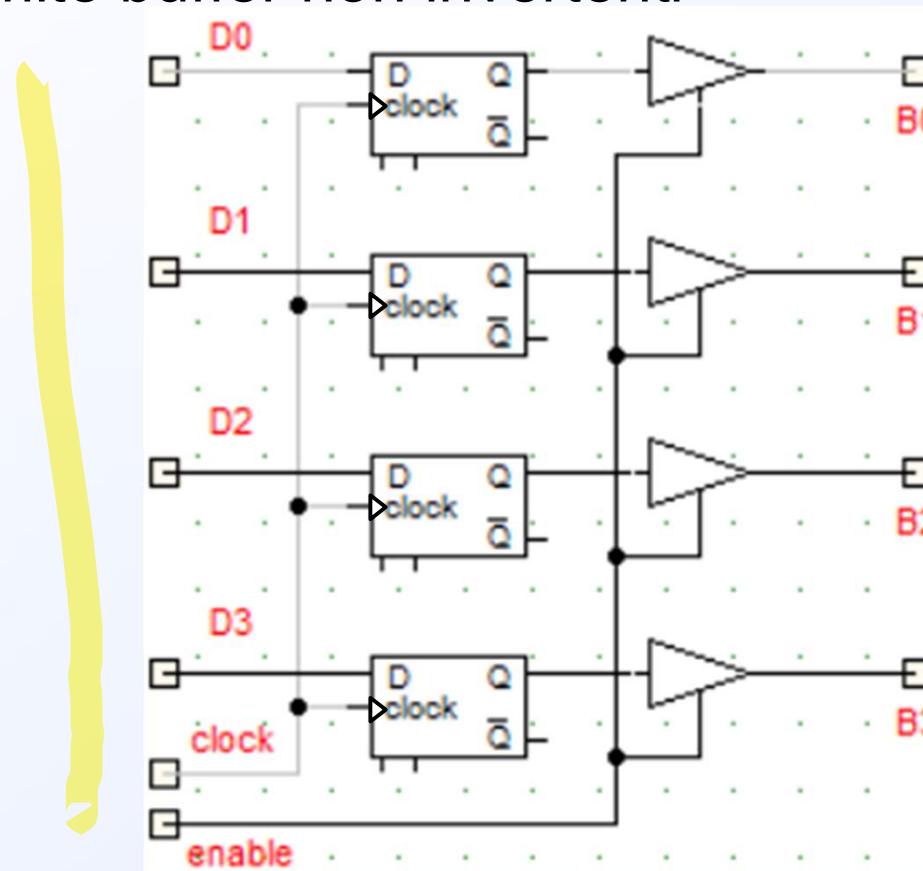
(c)



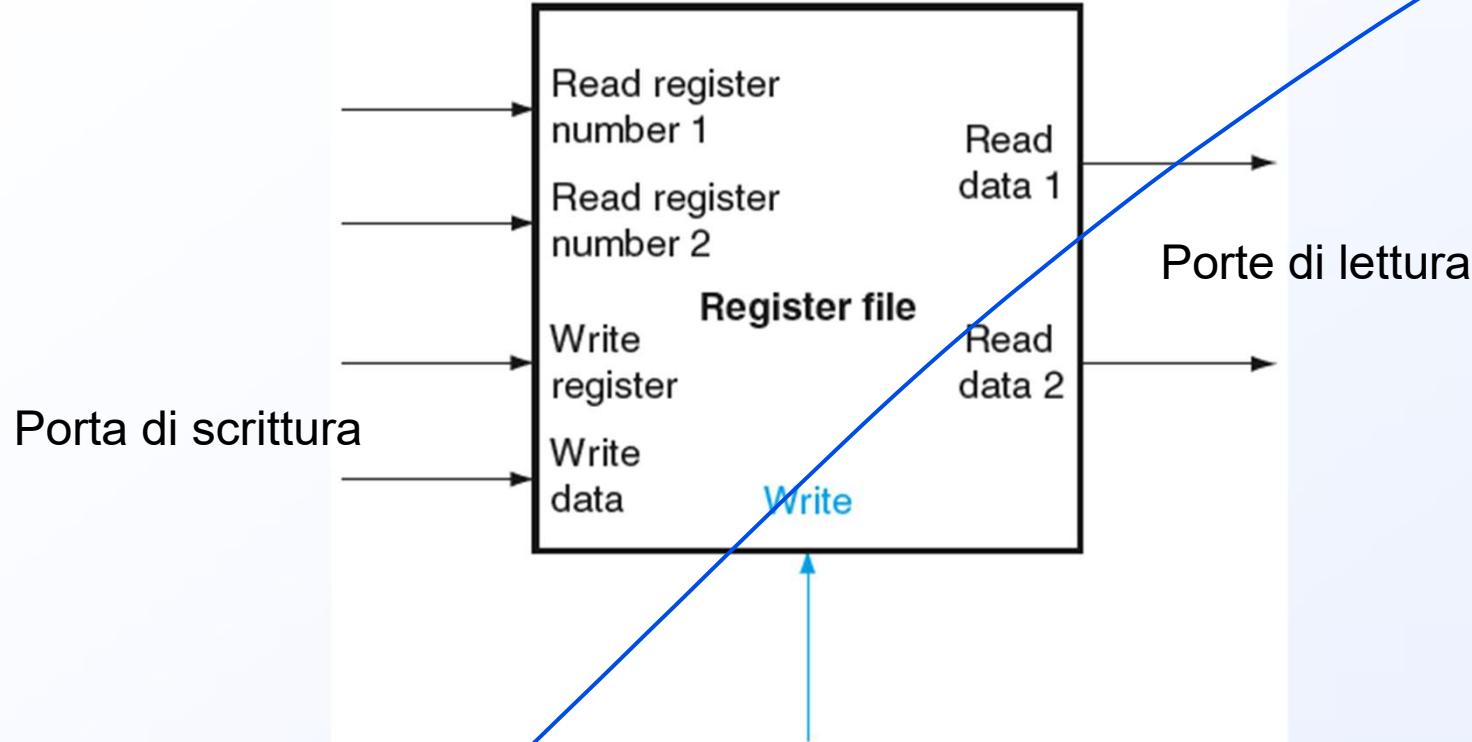
- Il buffer (non) invertente (detto anche tri-state) si comporta come un filo quando l'ingresso Control è alto (caso b)
- Il buffer (non) invertente disconnette Datain e Dataout quando l'ingresso Control è basso (caso c)
- Buffer invertente (caso d)

Registri

- Insieme di flip-flop D raggruppati insieme per formare un **registro**
- Lo stesso clock è in ingresso a tutti i **flip-flop** del registro
- L'ingresso *enable* permette di (dis)connettere il registro dal bus di output tramite buffer non invertenti

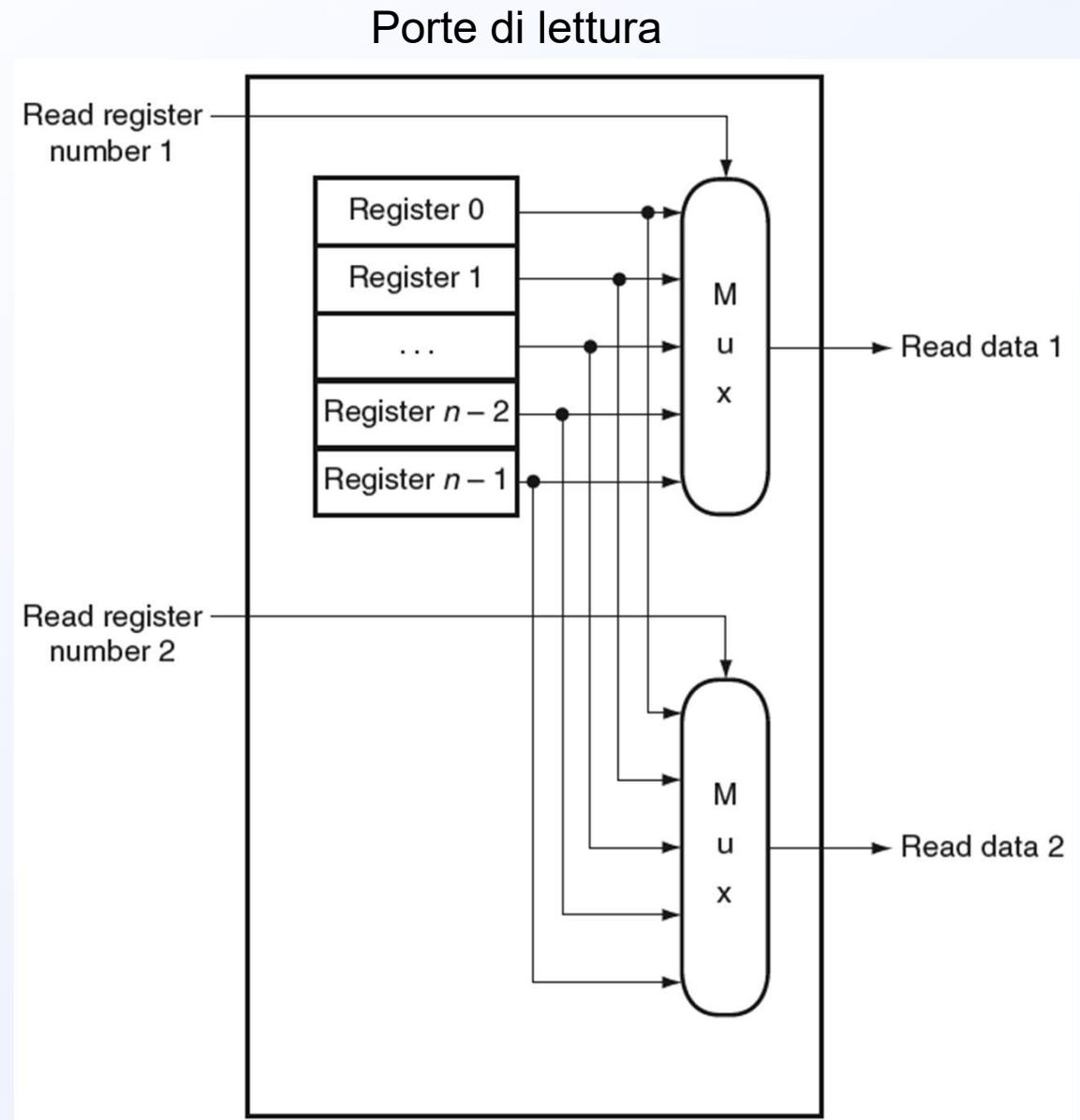
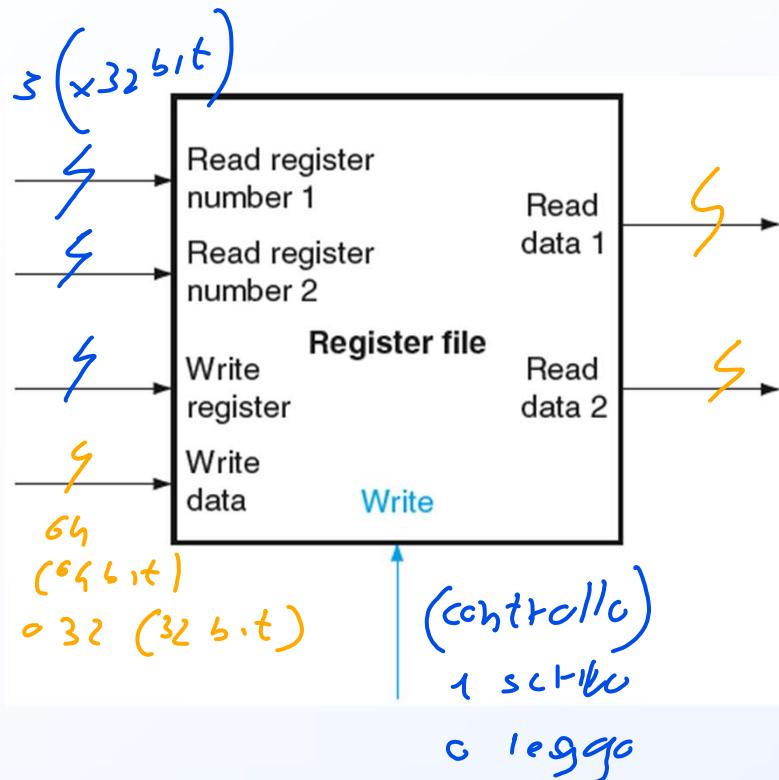


Blocco di registry (register file)

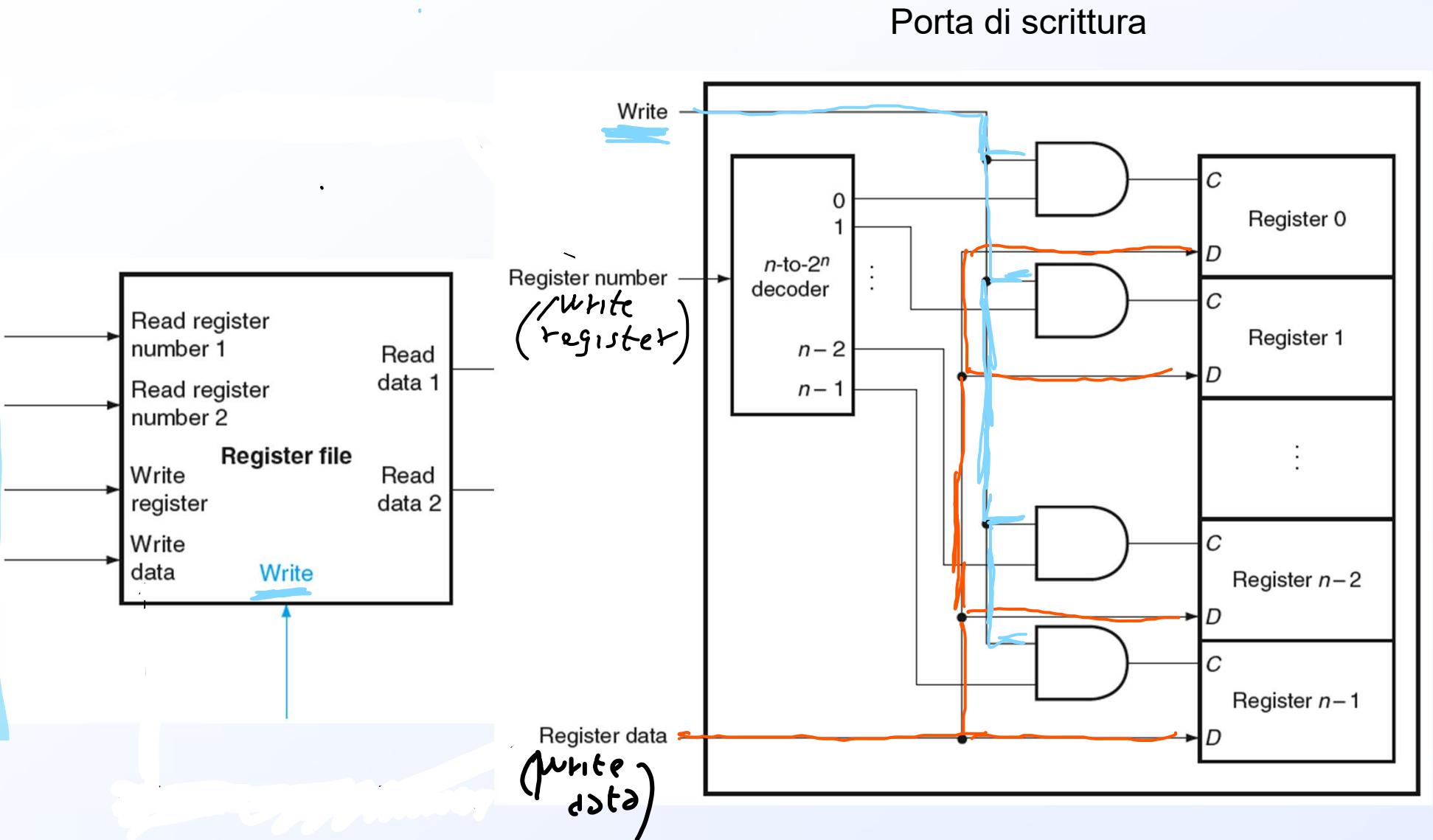


- Insieme di registri leggibili e scrivibili
- Si deve fornire in ingresso il numero del registro sul quale compiere l'operazione
- Si implementa con un decoder per ogni porta di lettura o scrittura ed un insieme di registri basati su flip-flop di tipo D

Blocco di registry (register file)



Blocco di registry (register file)

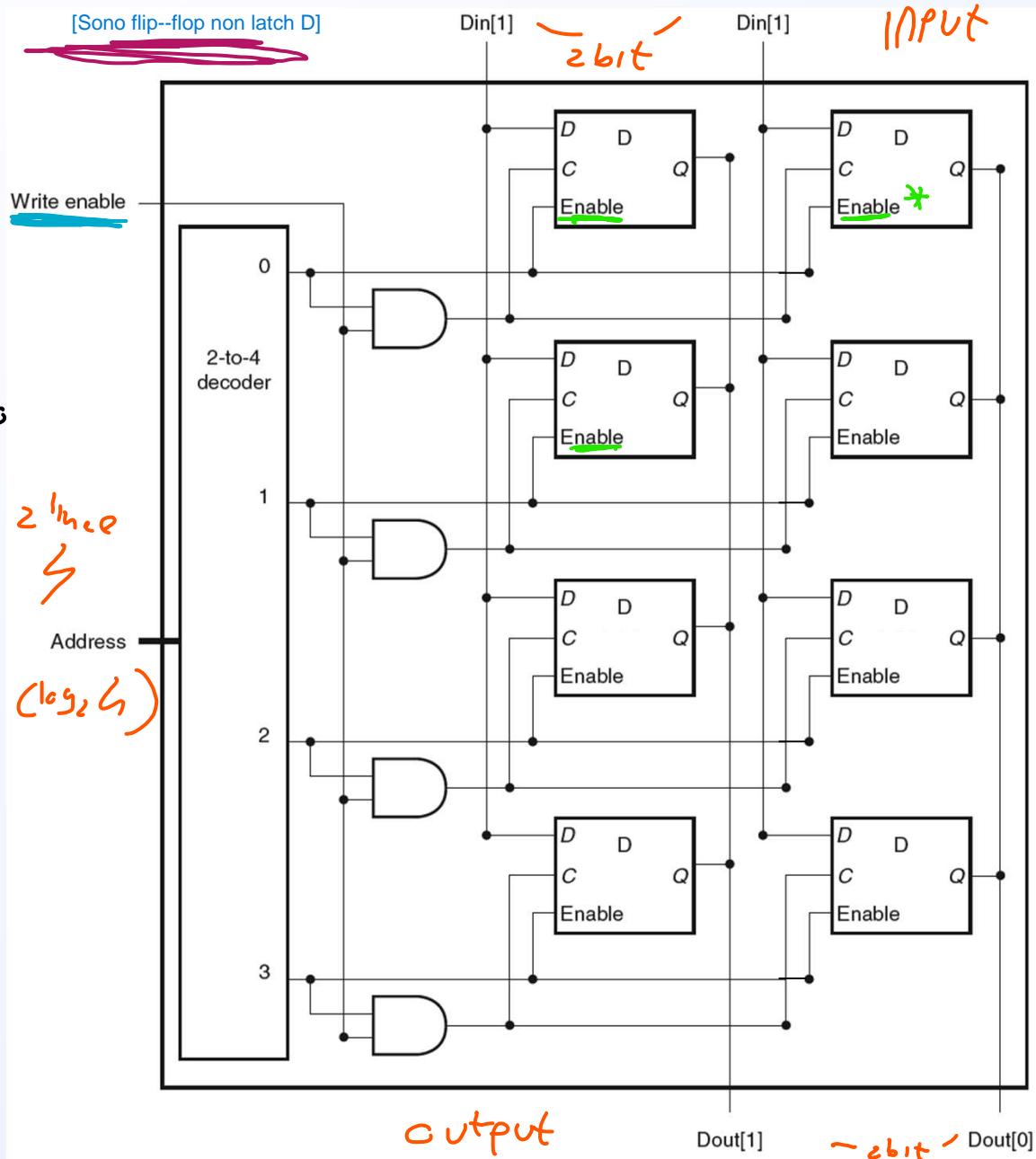


Organizzazione della memoria

- memoria 4×2 bit
- $\overset{\text{word}}{\text{5 linee di ingresso:}}$

 - 2 per i dati di input
 $\log_2(4)$
 - 2 per l'indirizzo $\log_2(4)$ *sceglie*
 - 1 per i bit di controllo:
 - Write enable

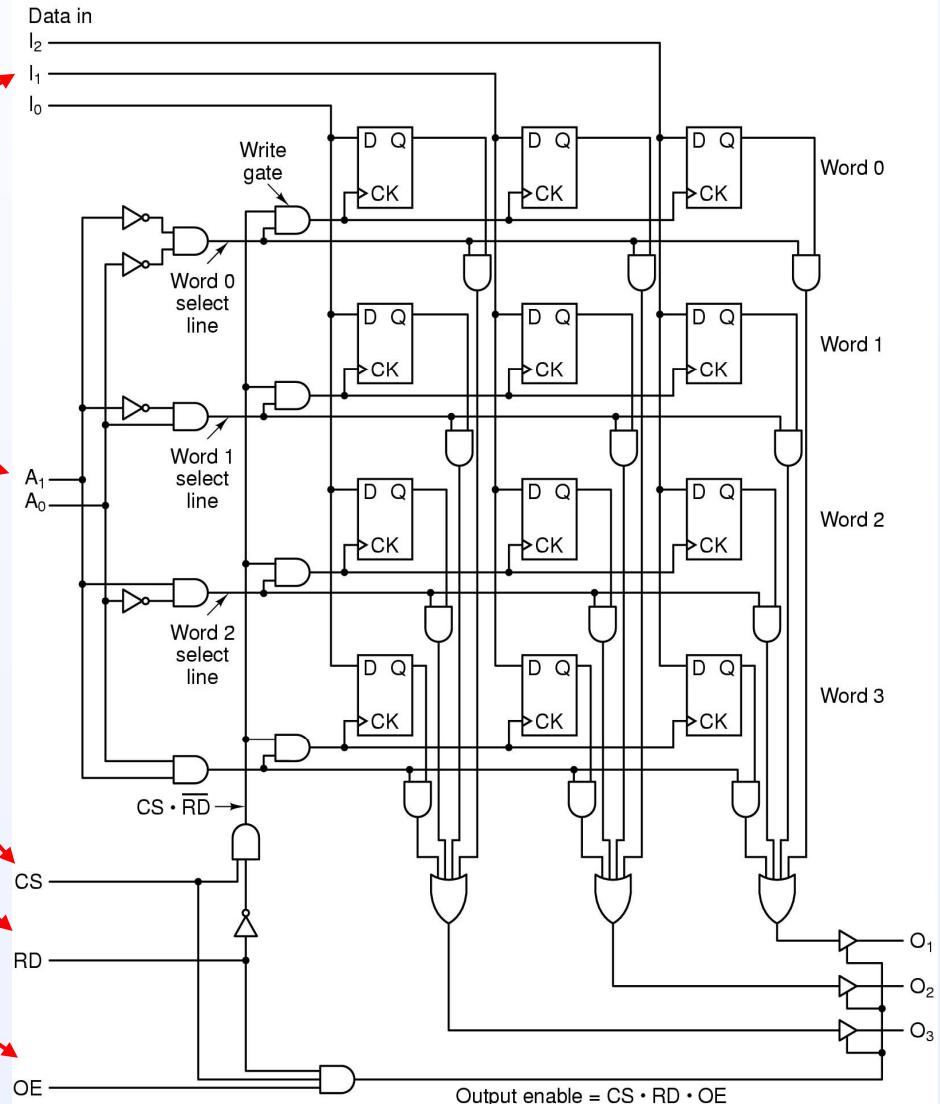
- 2 linee di uscita



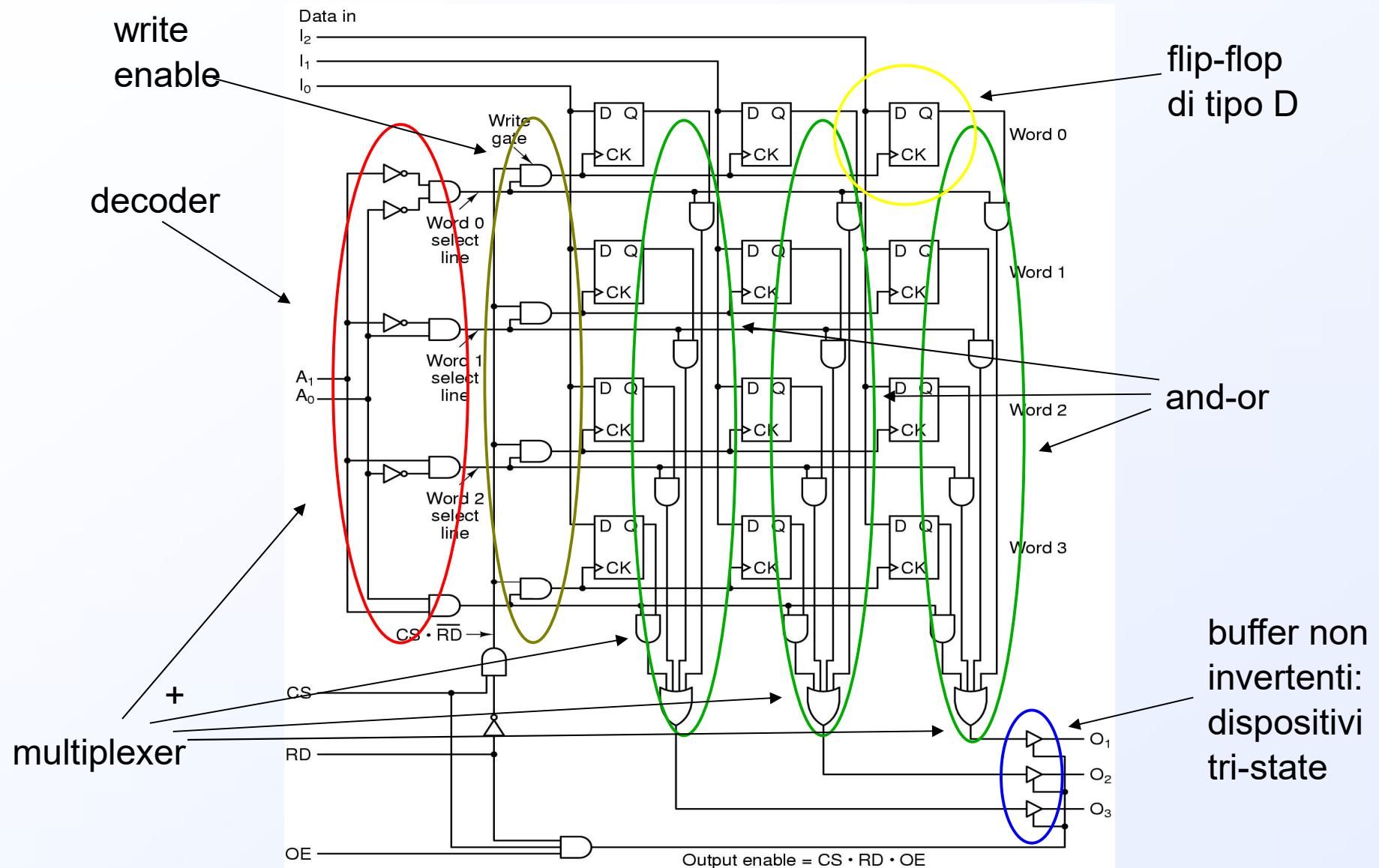
Organizzazione della memoria

es.
extra

- memoria 4×3
- 8 linee di input:
 - ✓ 3 per i dati di input
 - ✓ 2 per l'indirizzo
 - ✓ 3 per i bit di controllo:
 - ◆ CS per Chip Select
 - ◆ RD per distinguere tra read e write
 - ◆ OE per abilitare l'output
 - ✓ 3 per output



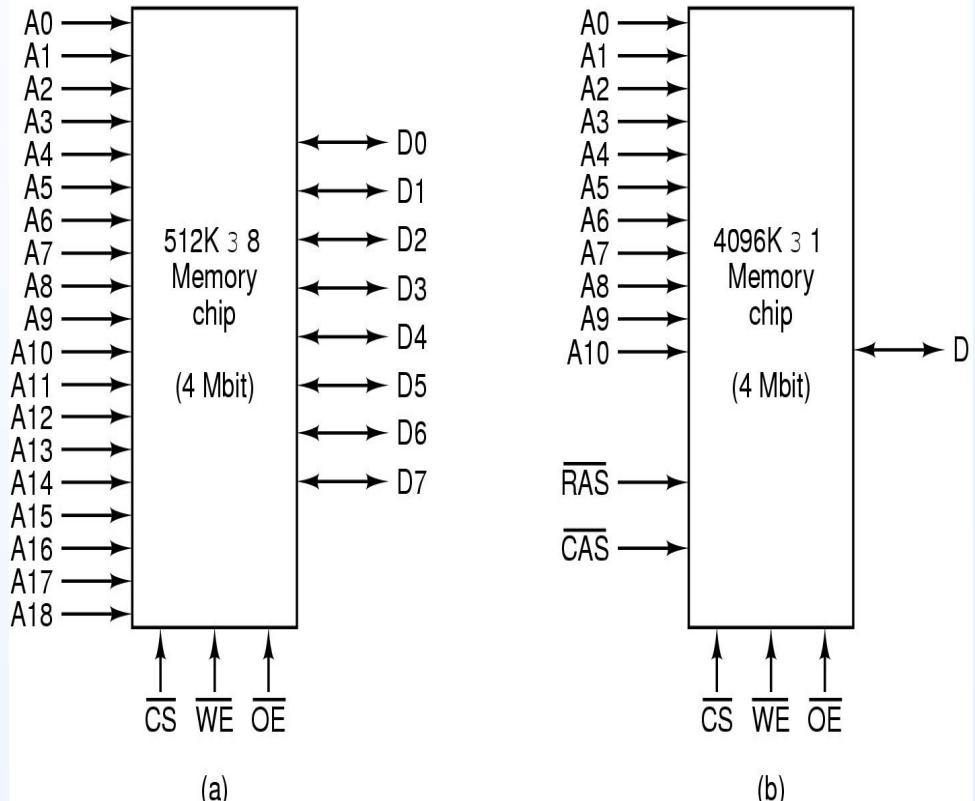
Organizzazione della memoria



bch
in
solociate

Chip di memoria

- n linee di indirizzo corrispondono a 2^n righe di flip-flop
- b linee di output corrispondono a b colonne di flip-flop
- I segnali possono essere attivi quando il livello è basso o alto (specificato nel chip)
- Matrici $2^n \times b$: selezione della colonna e della riga



Tipi di memoria: RAM

RAM: random access memory

- ✓ **SRAM: RAM statiche** (flip-flop tipo D), **estremamente veloci**, utilizzate per realizzare le cache
- ✓ **DRAM: RAM dinamiche** (transistor con condensatore), vanno rinfrescati, offrono grandi capacità ma più lente
 - DRAM FPM (Fast Page Mode): organizzate in matrici
 - DRAM EDO (Extended Data Output): con semplice pipeline per l'output
 - SDRAM (Synchronous DRAM): usata inizialmente su cache e memorie centrali, dati e indirizzi controllati dallo stesso clock (133 MHz per le SDR, Single Data Rate)
 - SDRAM-DDR (Double Data Rate): leggono sia nel fronte di salita che in quello di discesa (333/400 MHz), comparse inizialmente sulle schede video

Non
penso
siano
di fare

Tipi di memoria: ROM

ROM: Read-Only Memory, utilizzati per dati che non devono essere modificati

- ✓ PROM (Programmable ROM)
- ✓ EPROM (Erasable PROM): memoria cancellabile mediante esposizione alla luce ultravioletta
- ✓ EEPROM (Electrically Erasable PROM): memoria cancellabile per mezzo di impulsi elettrici (ma molto più lente)
- ✓ FLASH (EEPROM cancellabile a blocchi)