

# **Architettura degli Elaboratori e Laboratorio**

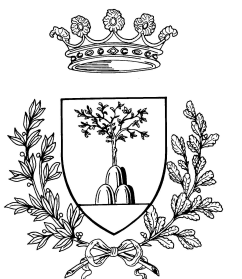
Matteo Manzali

Università degli Studi di Ferrara

Anno Accademico 2016 - 2017

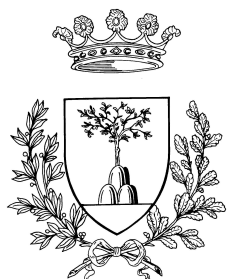
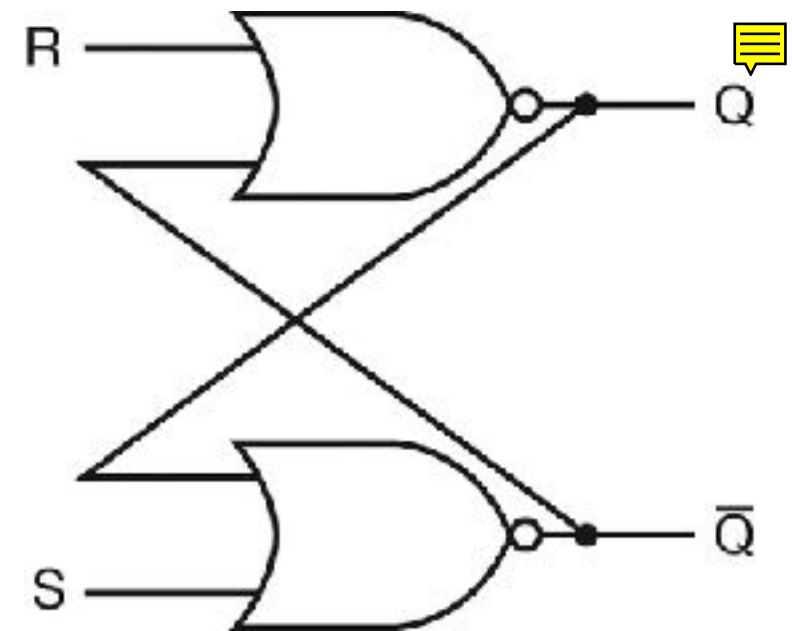
# Circuiti sequenziali

- I circuiti visti per la ALU sono definiti **combinatori**:
  - l'output dipende esclusivamente dai parametri di input
- Ai circuiti combinatori si contrappongono i circuiti **sequenziali**:
  - l'output dipende non solo dai parametri di input ma anche da uno "stato"
  - è il principio di base delle memorie
- E' possibile organizzare le porte logiche in modo tale da realizzare un circuito sequenziale?
  - Si... utilizzando la retroazione!



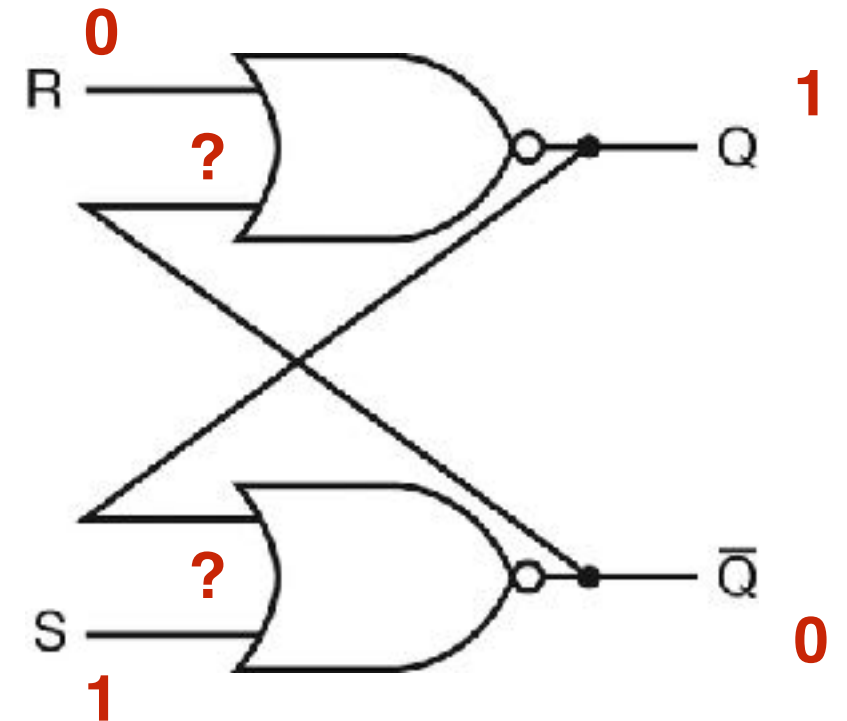
# II S-R latch

- E' un circuito bistabile (capace di mantenere 2 stati).
- Il cambio di stato è causato da segnale:
  - lo stato viene mantenuto fin tanto che non vengono inviati ulteriori segnali
- Bastano due porte NOR:
  - $S = 1, R = 0 \rightarrow Q = 1$  (set)
  - $S = 0, R = 1 \rightarrow Q = 0$  (reset)
  - $S = 0, R = 0 \rightarrow Q$  mantiene il suo valore
  - $S = 1, R = 1 \rightarrow$  **configurazione non valida**



# Set

- Poniamo  $(S, R) = (1, 0)$ :
  - qualsiasi valore abbia il secondo input, l'output del NOR di S sarà 0
  - essendo sia R sia il secondo input del NOR di R uguali a zero, l'output di questo NOR sarà uguale a 1
- Quindi avremo che:
  - $Q = 1$
  - $\bar{Q} = 0$

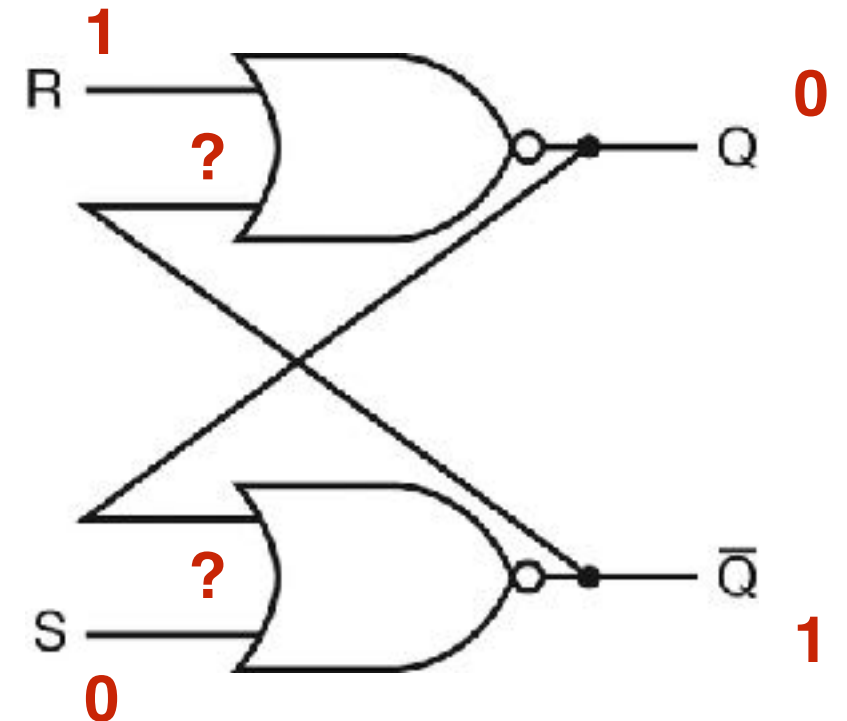


A	B	$\overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0



# Reset

- Poniamo  $(S, R) = (0, 1)$ :
  - qualsiasi valore abbia il secondo input, l'output del NOR di R sarà 0
  - essendo sia S sia il secondo input del NOR di R uguali a zero, l'output di questo NOR sarà uguale a 1
- Quindi avremo che:
  - $Q = 0$
  - $\overline{Q} = 1$



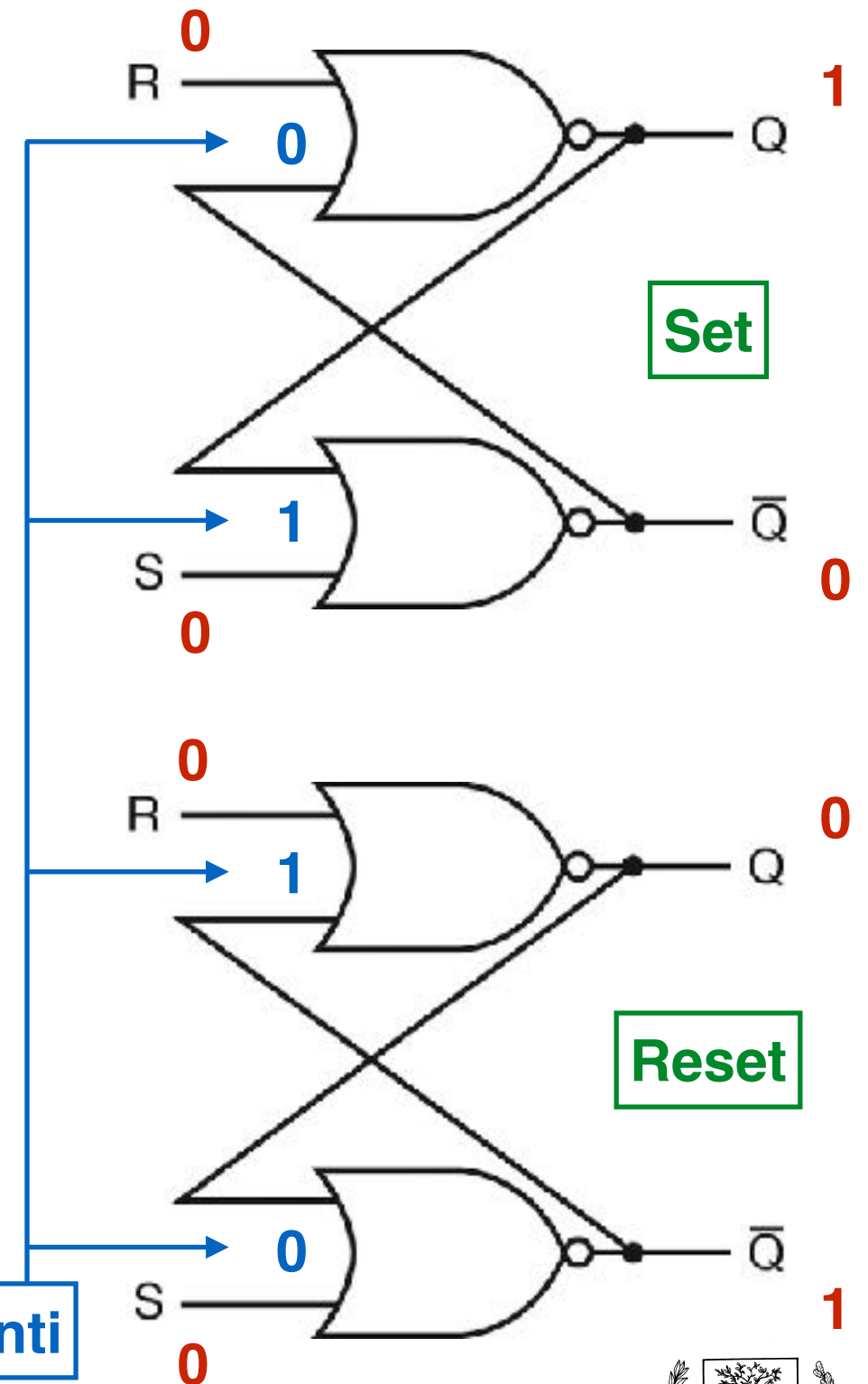
A	B	$\overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0



# Riposo

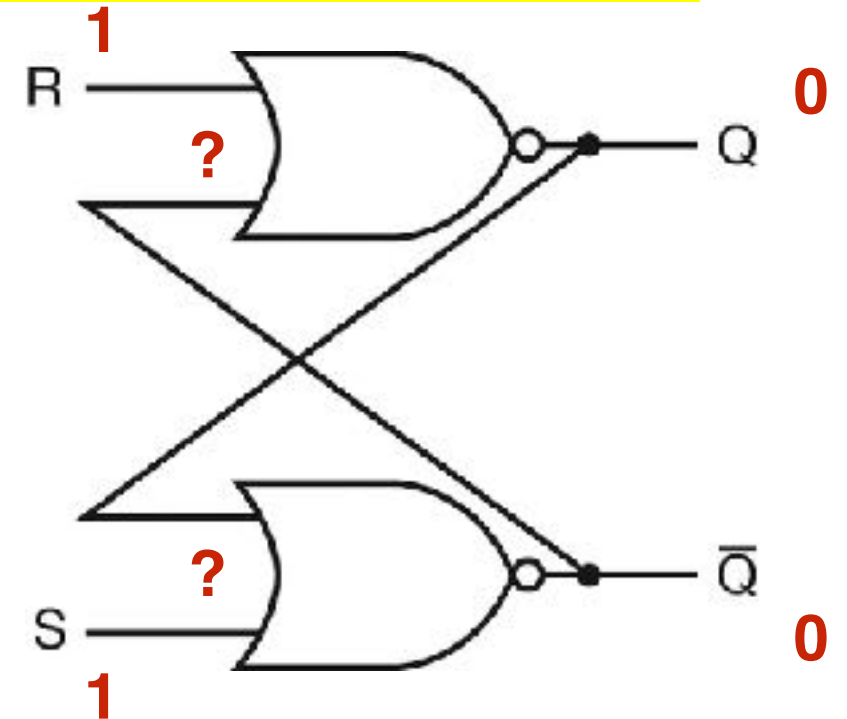
- Supponiamo che il latch sia partito da uno stato valido.
- Poniamo  $(S, R) = (0, 0)$ :
  - nel caso di uno stato di Set le uscite rimangono invariate
  - nel caso di uno stato di Reset le uscite rimangono invariate
- Nella configurazione “a riposo” il latch mantiene lo stato preesistente.

valori preesistenti



# Configurazione non valida

- Poniamo  $(S, R) = (1, 1)$ :
  - qualsiasi valore abbia il secondo input, l'output del NOR di R sarà 0
  - qualsiasi valore abbia il secondo input, l'output del NOR di S sarà 0
- Quindi avremo che:
  - $Q = \overline{Q}$
  - **Q non può essere uguale alla sua negazione!**

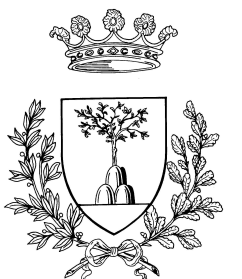
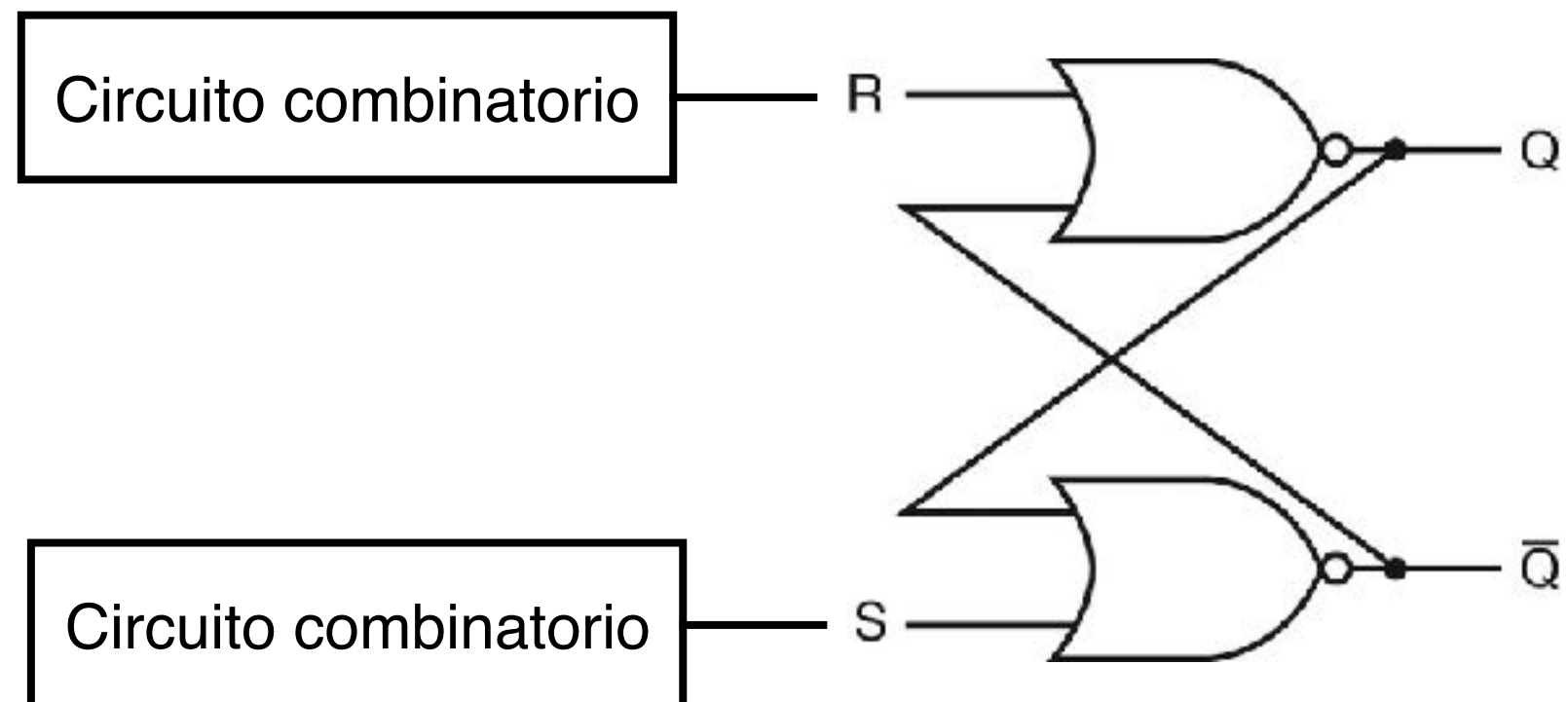


A	B	$\overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0



# Sincronizzazione

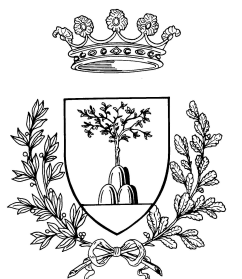
- I segnali S e R devono essere stabili e con valori validi (Set, Reset, Riposo).
- Spesso sono però calcolati da un circuiti combinatorio:
  - ogni porta logica introduce un ritardo
  - l'output del circuito diventa stabile solo **dopo** un certo intervallo di tempo



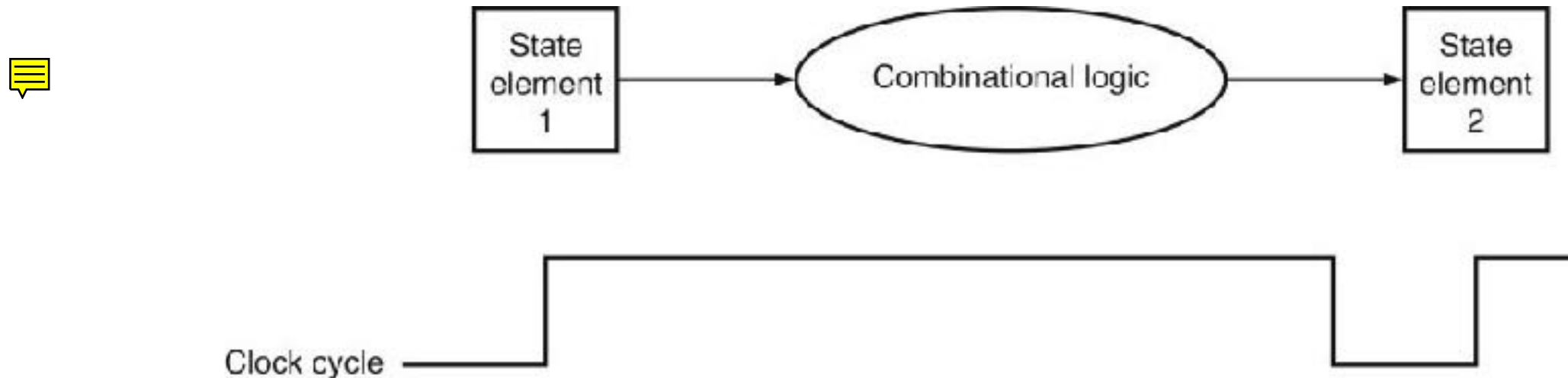


# Clock

- Bisogna “abilitare” il latch solo quando si è sicuri che tutte le operazioni precedenti siano state completate.
- Soluzione:
  - si usa un segnale a gradino, detto **clock**, che oscilla tra due stati
  - il periodo (o ciclo) di clock viene scelto grande abbastanza da assicurare la stabilità degli output del circuito
  - usiamo il clock per abilitare la scrittura nei latch
  - il clock determina il *ritmo* dei calcoli e delle relative operazioni di memorizzazione
  - Il circuito diventa *sincrono* (rispetto al segnale di clock)



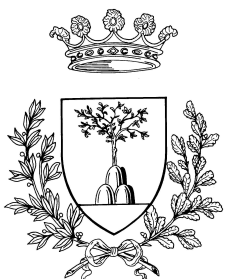
# Clock



- Il primo elemento di stato fornisce gli input al circuito logico combinatorio.
- La durata di un ciclo di clock deve essere sufficientemente lungo affinché i segnali in input riescano a propagarsi all'interno del circuito logico e generare un segnale di output stabile.
- il clock è unico per tutto il sistema → il periodo deve essere sufficiente affinché ogni circuito logico abbia il tempo di stabilizzare il proprio segnale di output.

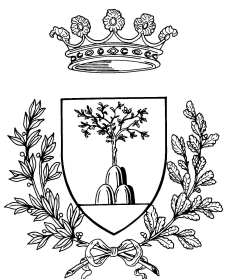
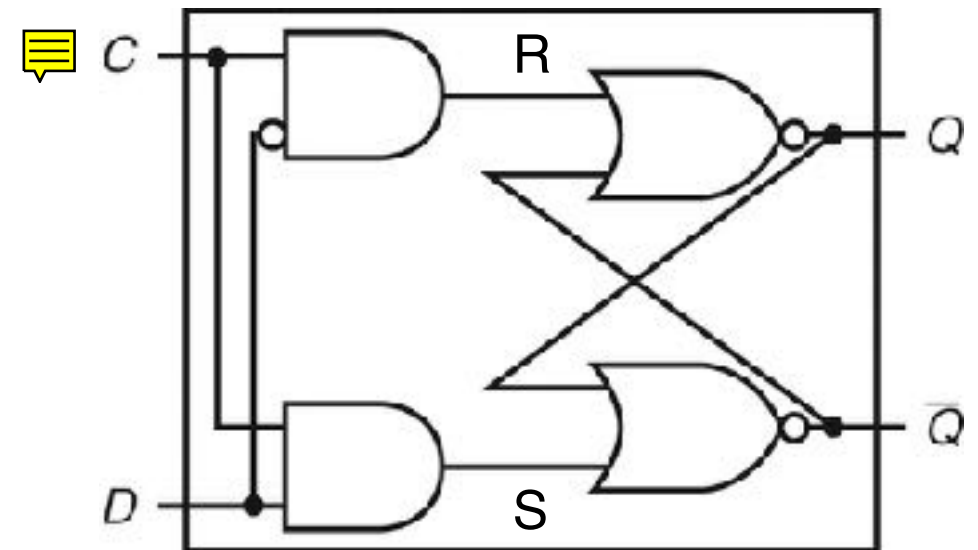
# Unità di misura

- Dato un periodo di clock  $T$  espresso in secondi:
  - frequenza di clock ( $F$ ):  **$1 / T$  Hz** (numero di clici al secondo)
- Esempio:
  - $T = 10 \text{ us} \rightarrow T = 10 \cdot 10^{-6} \text{ s}$   
 $F = 1 / T = 1 / (10 \cdot 10^{-6}) = 1 / 10^{-5} = 10^5 \text{ Hz} = 100 \text{ KHz}$
  - $T = 1 \text{ ns} \rightarrow T = 1 \cdot 10^{-9} \text{ s}$   
 $F = 1 / T = 1 / (1 \cdot 10^{-9}) = 1 / 10^{-9} = 10^9 \text{ Hz} = 1 \text{ GHz}$




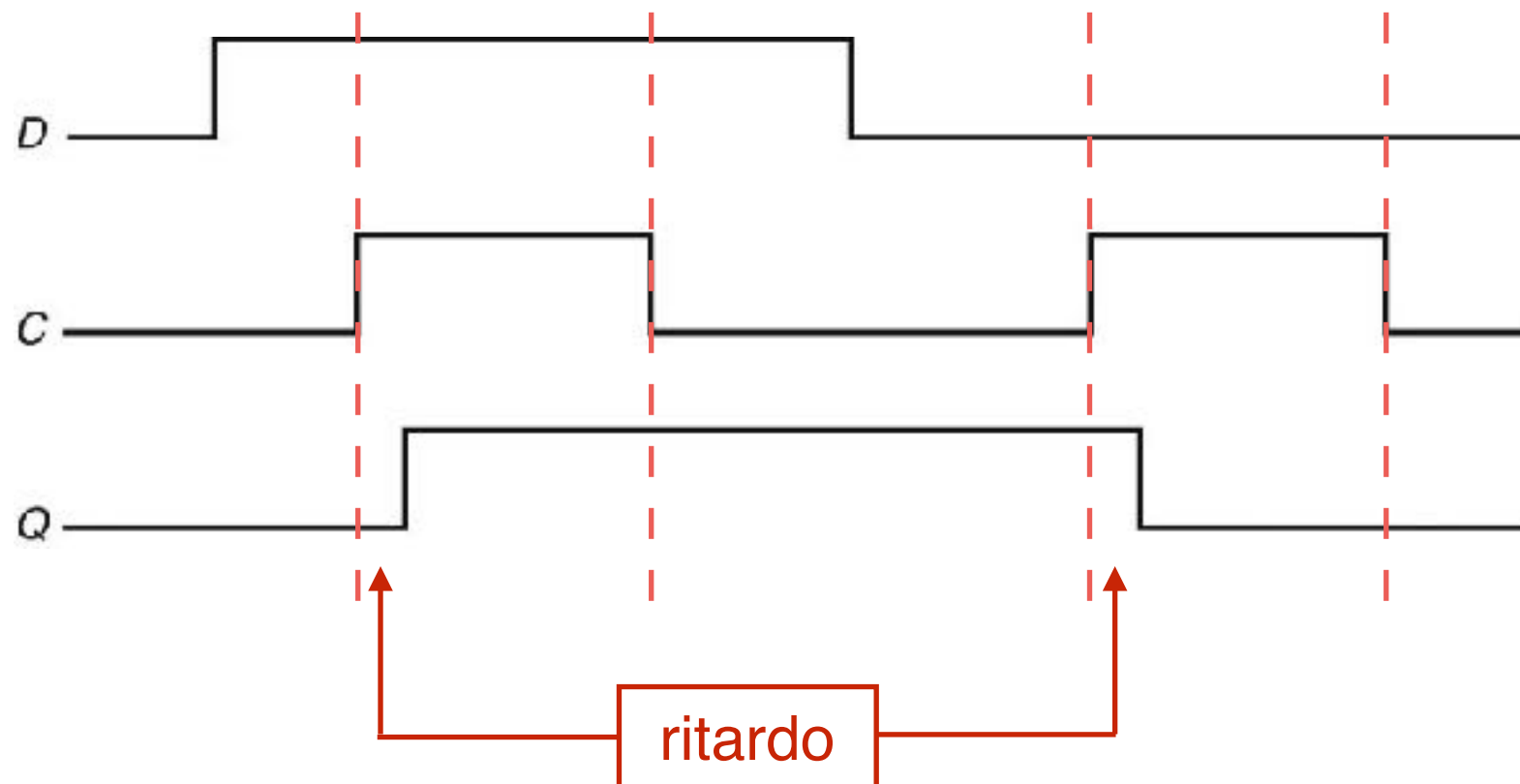
# D latch

- Combinazione di clock e latch:
  - $C \rightarrow$  segnale di clock
  - $D \rightarrow$  input
  - $Q \rightarrow$  output
- Quando il clock è alto (a 1) viene memorizzato il segnale D:
  - $D = 1$  corrisponde al Set
  - $D = 0$  corrisponde al Reset
- Quando il clock è basso (a 0) viene mantenuto il vecchio stato.
- **La configurazione non valida non può mai verificarsi.**



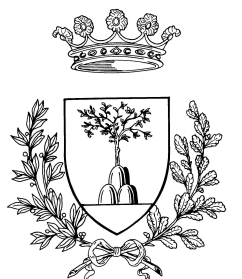
# D latch

- Il D latch è “trasparente”: 
- se C è alto, Q assume “immediatamente” (a meno di ritardi dovuti alla propagazione del segnale) il valore di D

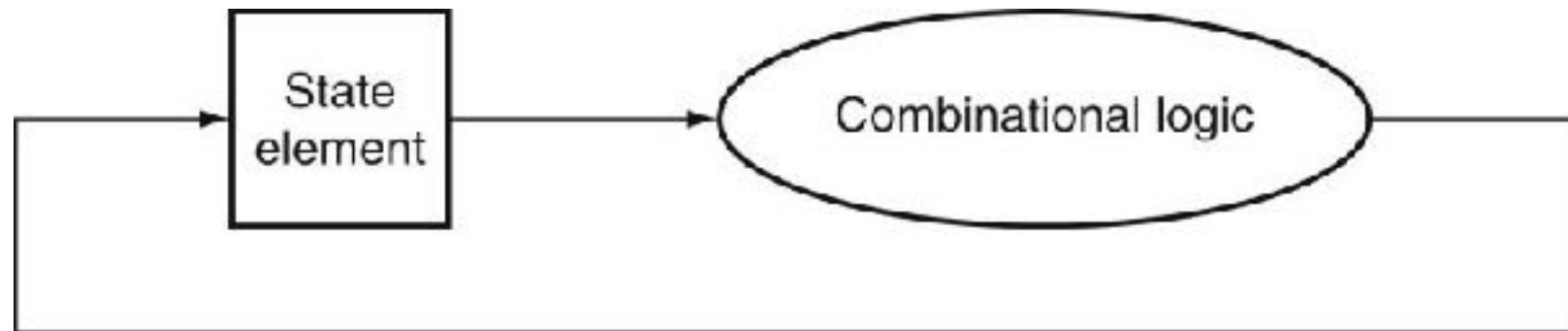


# D latch

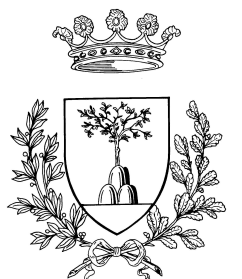
- Fin tanto che C è alto, **ogni cambio di stato di D porta ad un cambio di Q.**
- Di fatto il D latch non memorizza tutti i cambiamenti di stato durante la fase alta del clock:
  - solo quando il clock diventa basso, Q mantiene il vecchio stato
- Non è il comportamento che vogliamo!



# Memorizzazione

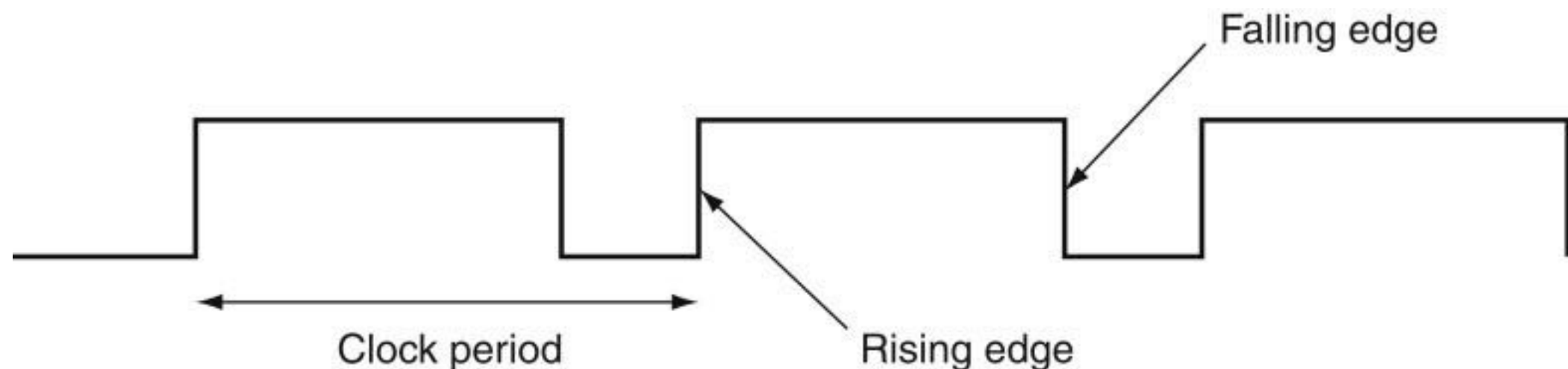


- **Durante ogni periodo di clock:**
  - il circuito combinatorio dovrebbe calcolare una funzione sulla base dell'attuale valore dell'elemento di memoria (stato del circuito)
  - l'output calcolato dovrebbe diventare il nuovo valore da memorizzare nell'elemento di memoria (nuovo stato del circuito)
  - il nuovo valore memorizzato dovrebbe essere usato come input del circuito durante il ciclo di clock successivo



# Metodologia di timing

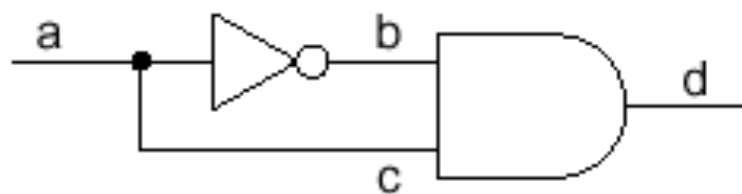
- La memorizzazione può avvenire in vari istanti rispetto al clock:
- **level-triggered**: avviene sul livello alto (o basso) del clock  
→ il D latch ne è un esempio
- **edge-triggered**: avviene sul fronte di salita (o discesa) del clock  
→ **è quello che ci serve!**
- Possiamo immaginare che la memorizzazione 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.



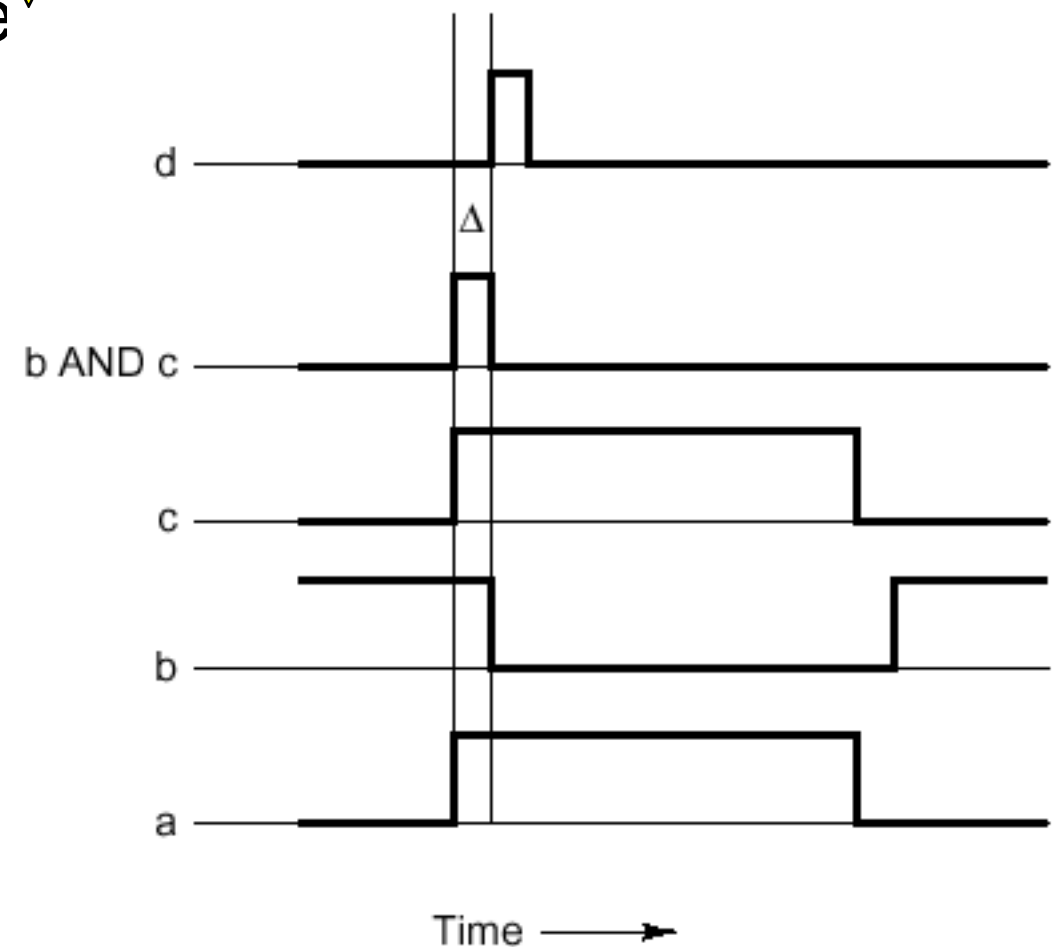


# Generatore di impulsi

- Il generatore di impulsi permette di generare impulsi *brevissimi* in corrispondenza del fronte di salita di un segnale a gradino:
  - sfrutta i ritardi delle porte logiche
- Porta NOT e AND con ritardo:

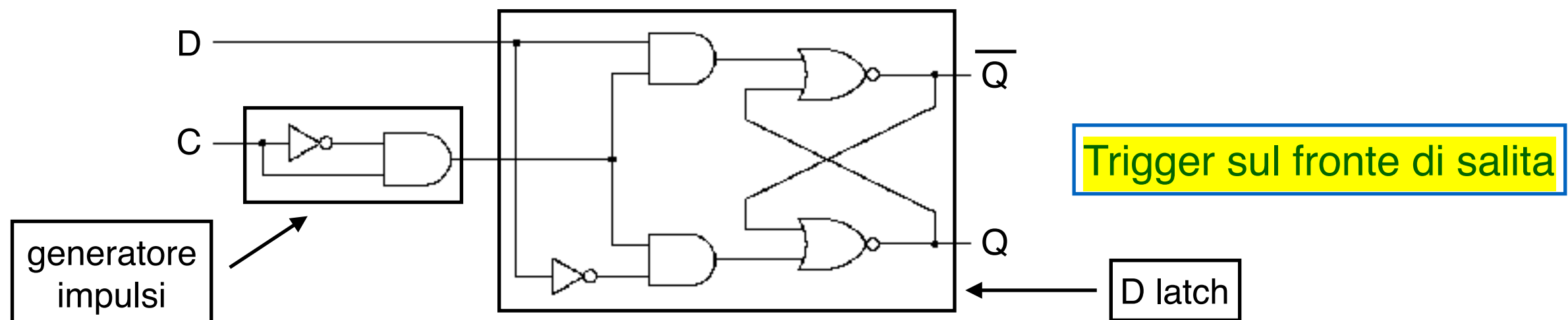


- Solo durante l'intervallo  $\Delta$ :
  - i valori b e c sono entrambi a 1
  - l'impulso b AND c vale 1

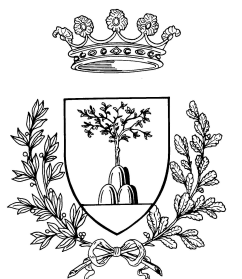
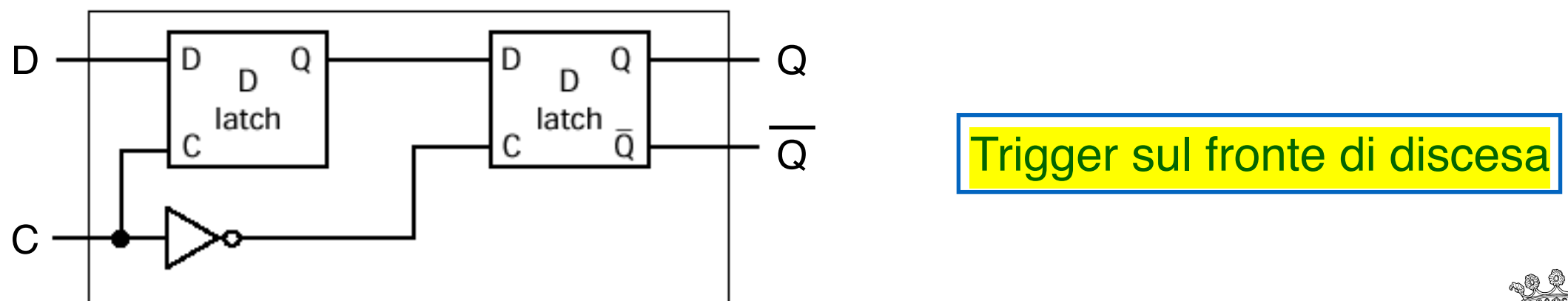


# D flip-flop

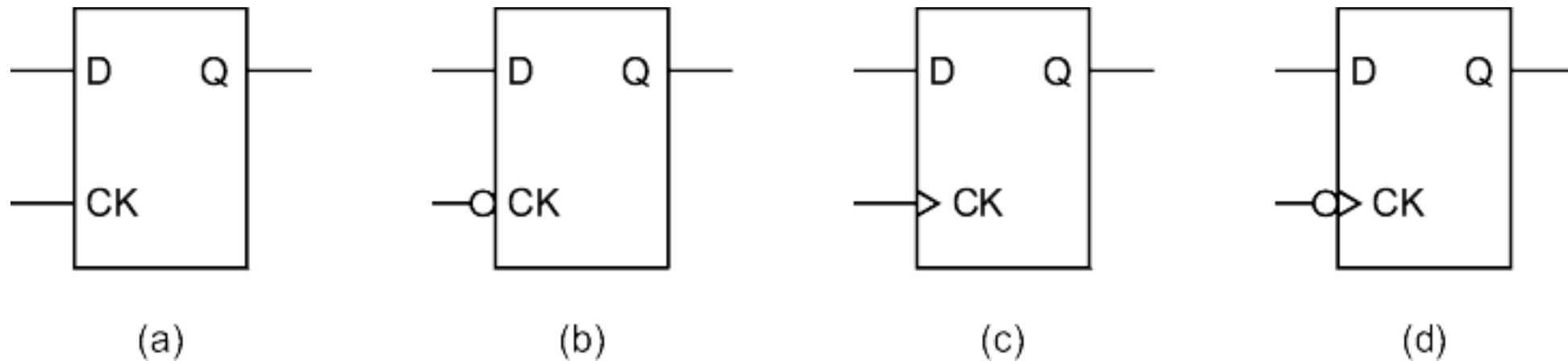
- I flip-flop sono dei circuiti che permettono di memorizzare i valori in ingresso durante il fronte di salita (o discesa) del clock.
- Possono essere creati con un D latch e un generatore di impulsi:



- Oppure con due D latch in serie ed un segnale di clock negato:



# Rappresentazioni grafiche



(a) Latch attivo alto

(b) Latch attivo basso

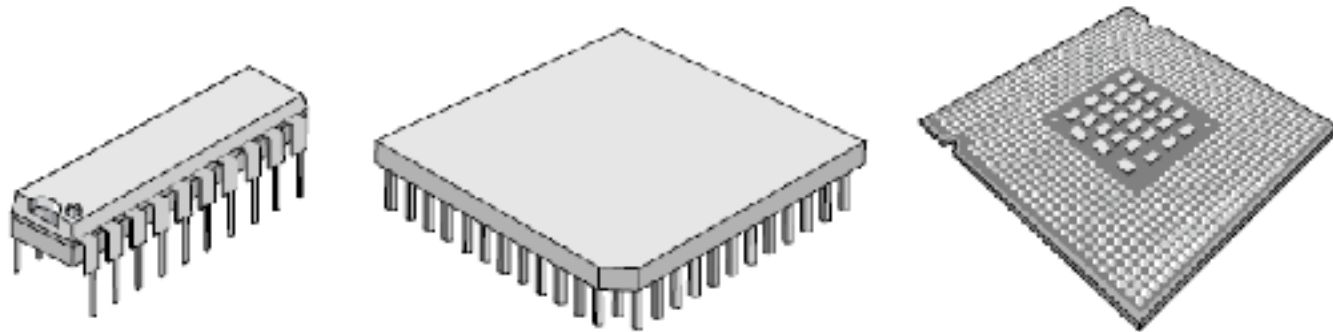
(c) Flip-flop attivo sul fronte di salita

(d) Flip-flop attivo sul fronte di discesa

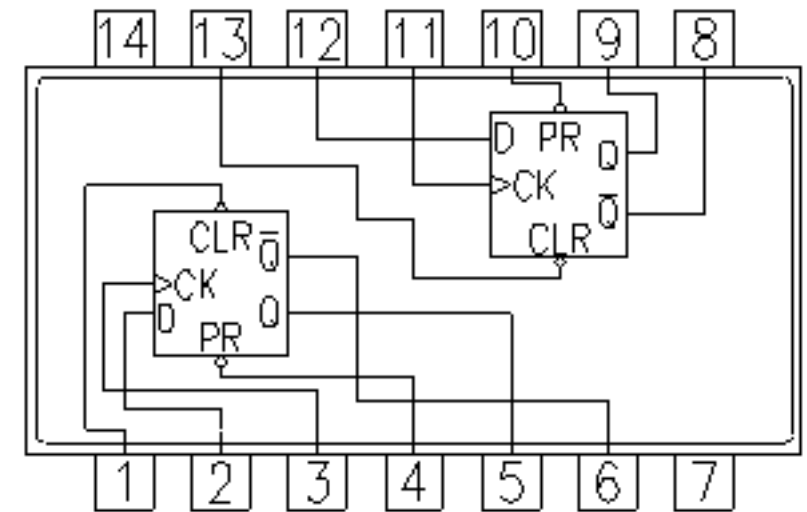
P.S.: l'uscita negata è omessa nella rappresentazione grafica.

# Nella realtà

- Nella realtà le porte logiche sono implementate attraverso transistor, resistenze e collegamenti.
- Questi vengono creati su fogli di silicio, trattandoli opportunamente secondo determinati schemi.
- I fogli di silicio trattati vengono inseriti all'interno di appositi contenitori, detti **packages**.



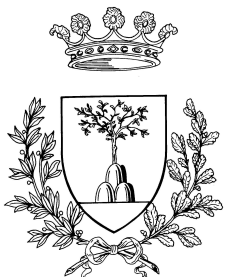
(a) diversi esempi di packages



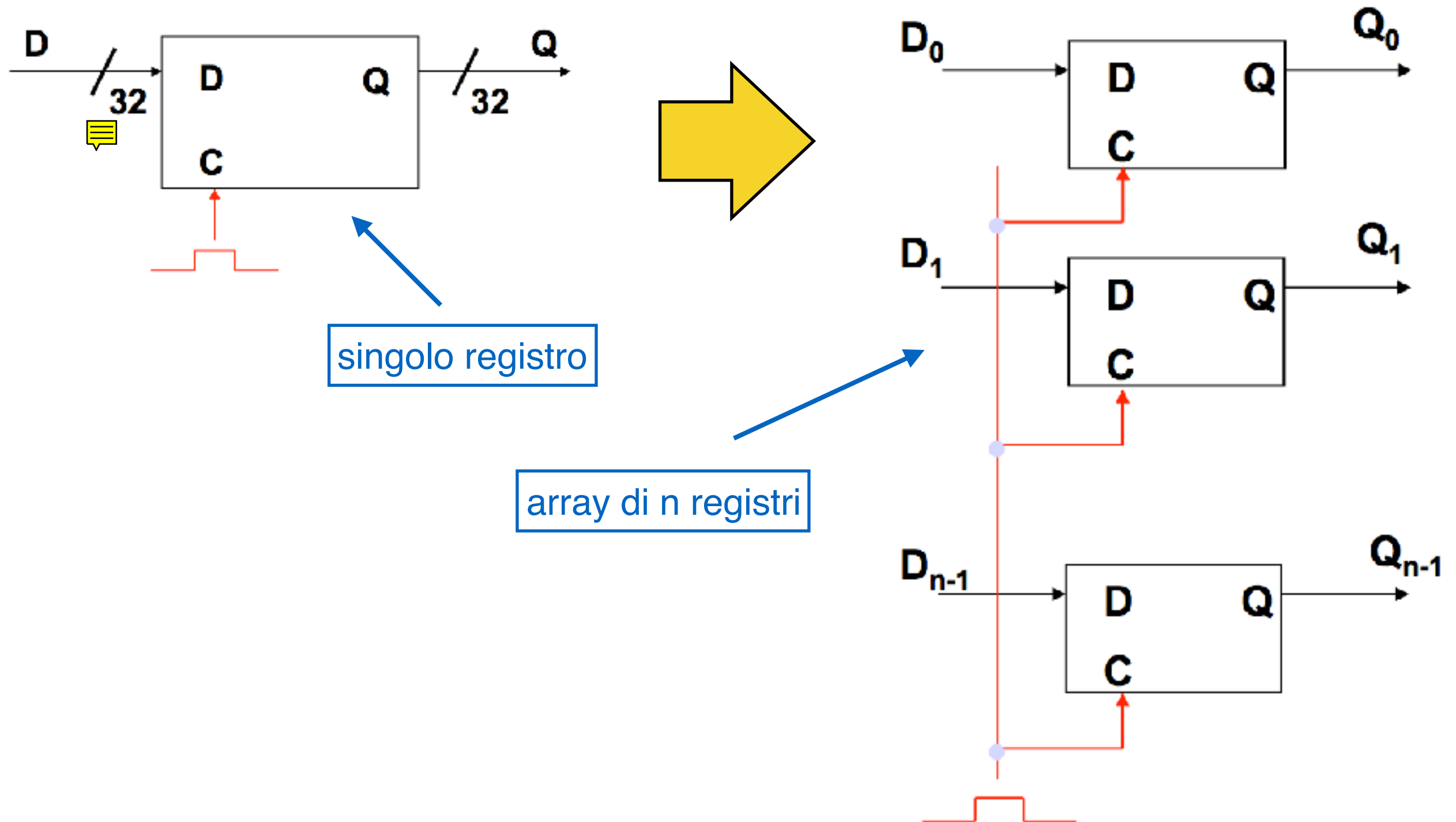
(b) schema logico del IC 7474  
contenente 2 “D flip-flop”

# Register File

- La parte operativa della CPU (datapath) contiene alcuni registri che memorizzano gli operandi delle istruzioni aritmetico/logiche:
  - ogni registro è costituito da  $n$  flip-flop, dove  $n$  è il numero bit che costituiscono una parola
- Più registri sono organizzati in una componente nota come **Register File**:
  - il Register File del MIPS contiene 32 registri ( $32 \cdot 32 \text{ bit} = 1024$  flip-flop)
- Il Register File deve permettere:
  - lettura di 2 registri
  - scrittura di 1 registro

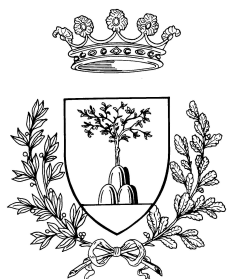
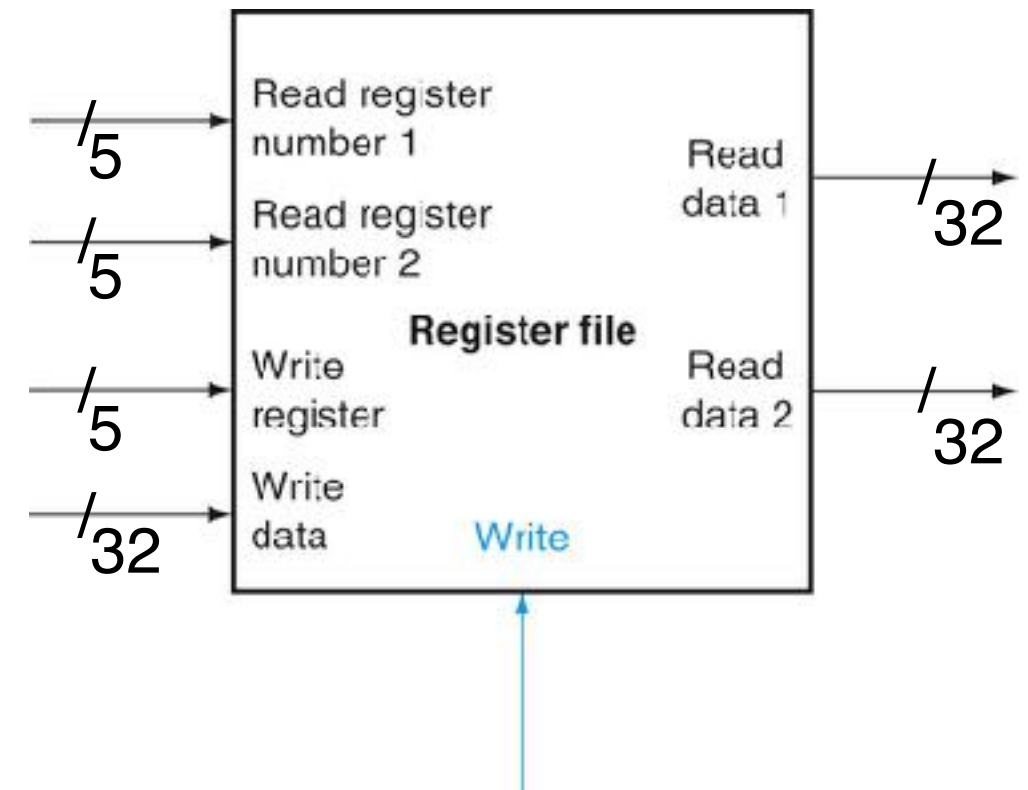


# Register File



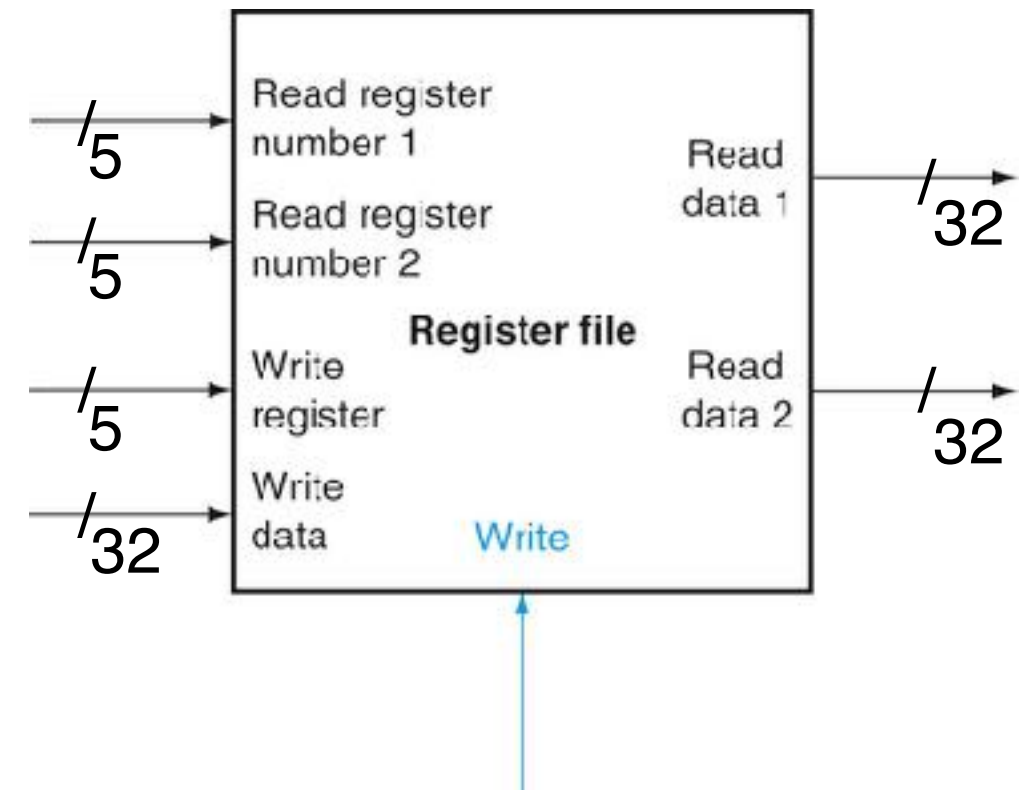
# Register File


- **Read reg. # 1** (5 bit):
  - numero del 1° registro da leggere
- **Read reg. # 2** (5 bit):
  - numero del 2° registro da leggere
- **Read data 1** (32 bit):
  - valore del 1° registro, letto sulla base di **Read reg. # 1**
- **Read data 2** (32 bit):
  - valore del 2° registro, letto sulla base di **Read reg. # 2**

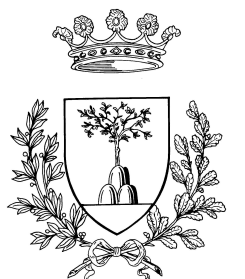


# Register File

- **Write register** (5 bit):
  - numero del registro da scrivere
- **Write data** (32 bit):
  - valore da scrivere nel registro



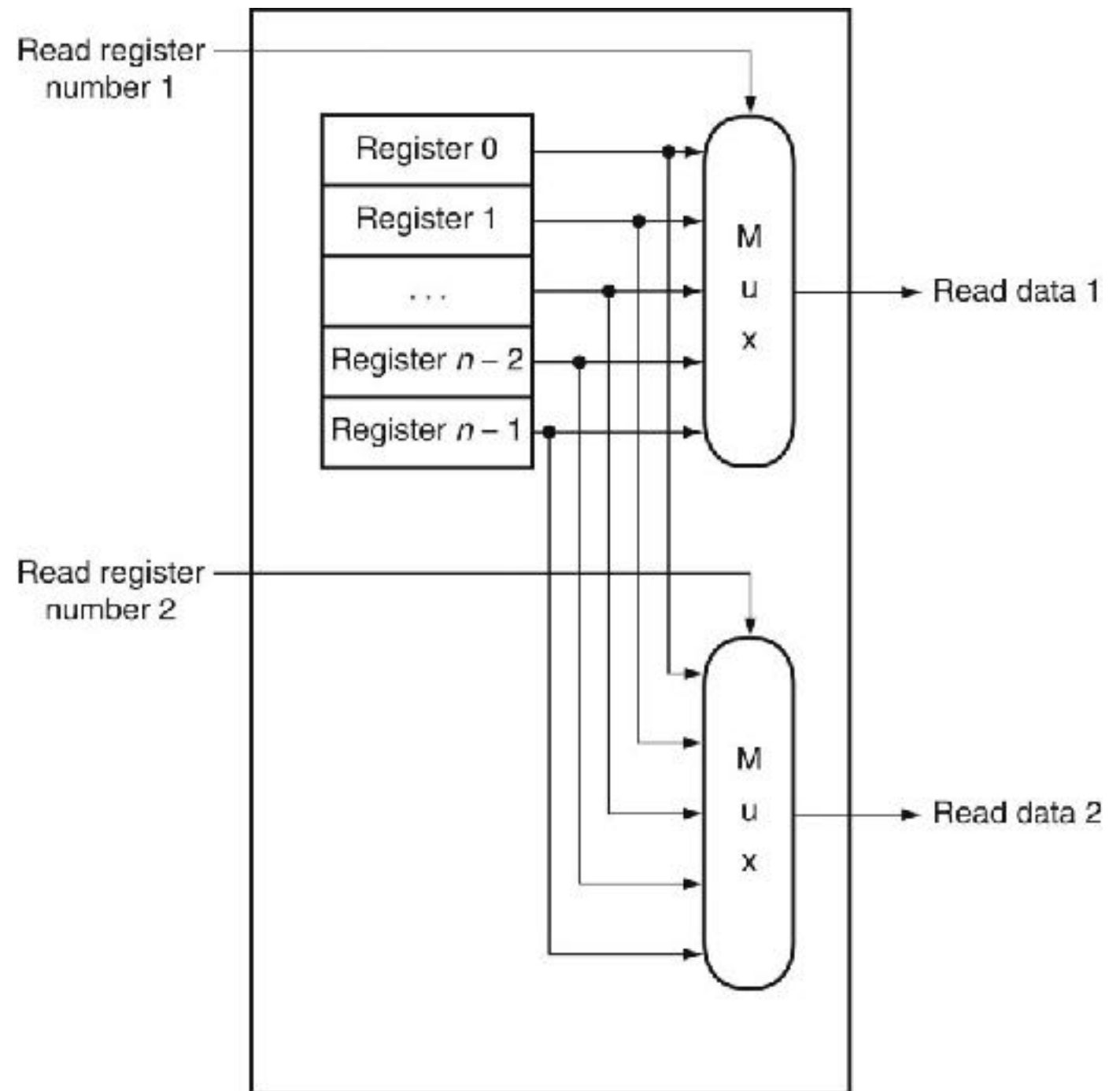
- **Write:** 
  - segnale di controllo messo in AND **con il clock**
  - il segnale determina se, in corrispondenza del fronte di discesa del clock, il valore debba (o meno) essere memorizzato
  - solo se **Write** = 1, il valore di **Write data** viene scritto in uno dei registri





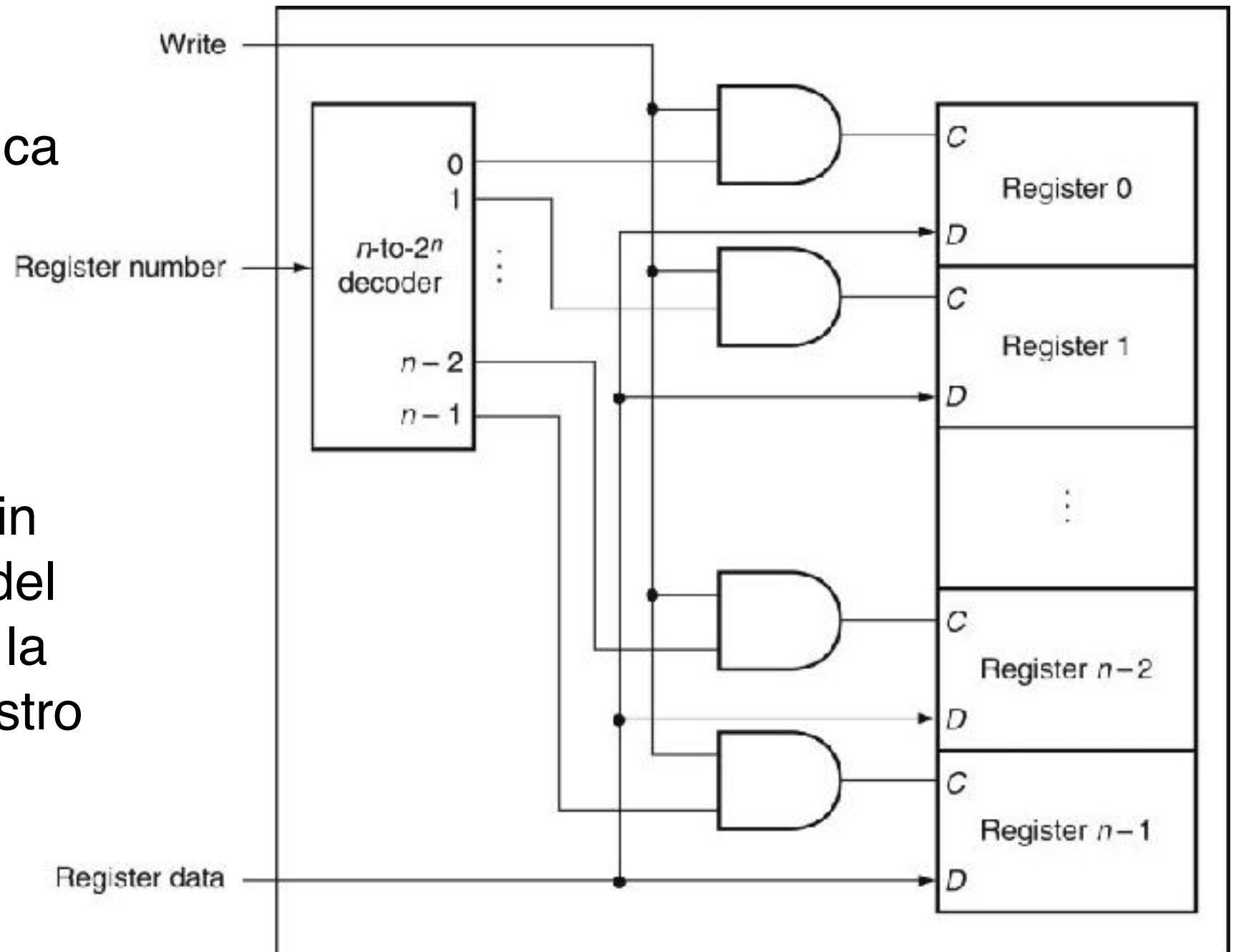
# Lettura dal Register File

- MUX a 32 bit.
- Le operazioni richiedono solitamente 2 operandi.
- Ogni input indica il numero del registro da leggere.
- Ogni output contiene il valore del registro letto.
- Il Register File fornisce **sempre** in output dei valori, semplicemente vengono ignorati se non si è in una operazione di lettura.



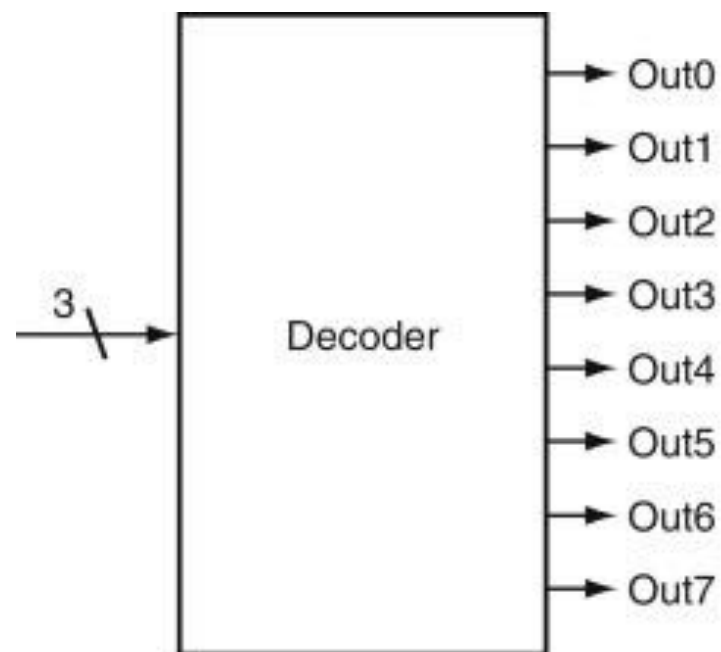
# Scrittura sul Register File

- Viene scritto un solo registro.
- Il decoder decodifica il segnale a 5 bit nel numero di registro scelto (da 0 a 31).
- Il segnale **Write** è in AND con l'output del decoder ed abilita la scrittura per il registro selezionato.



# Decoder

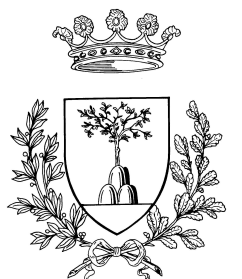
- Un circuito logico con  $n$  segnali di input e  $2^n$  segnali di output, dove solo un segnale è alto per ogni combinazione di input.
- Es. decoder a 3 bit ( $2^3$  bit di output):



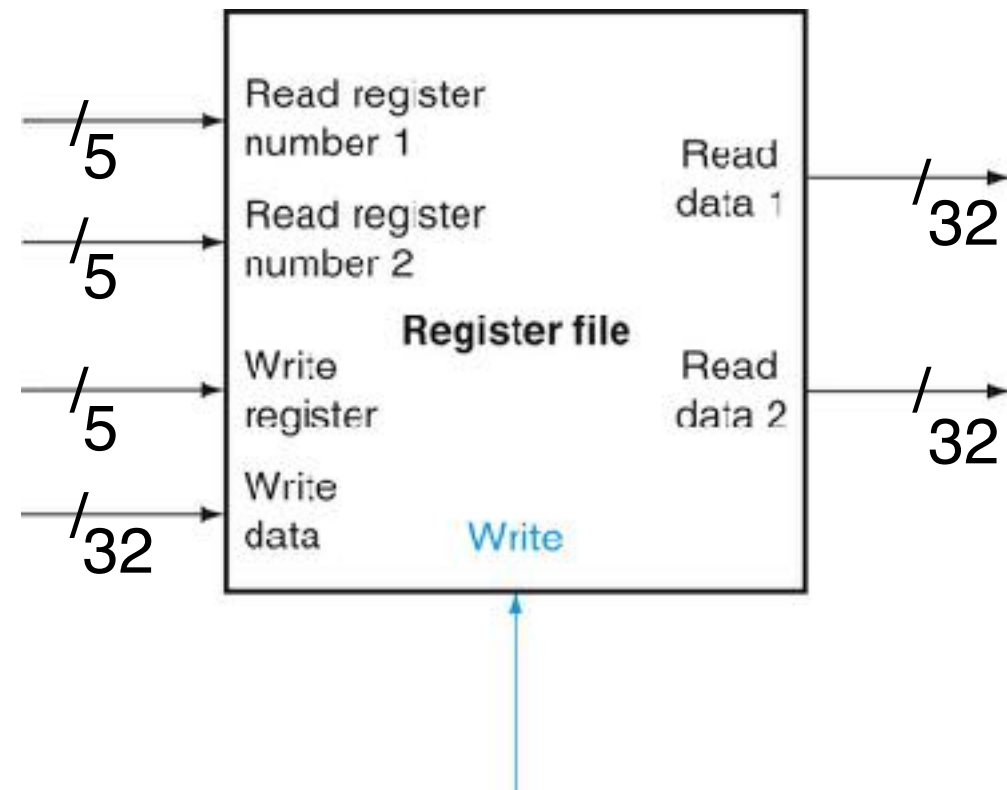
a. A 3-bit decoder

Inputs			Outputs							
12	11	10	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

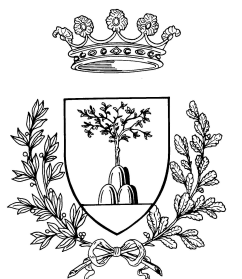
b. The truth table for a 3-bit decoder



# Lettura / Scrittura

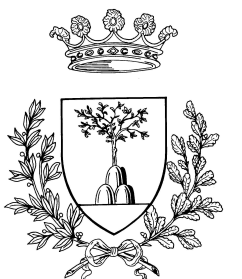


- Cosa succede se uno stesso registro del Register file viene acceduto in lettura e scrittura **durante uno stesso ciclo di clock?**
- Il valore ritornato dalla lettura è quello memorizzato in un ciclo di clock precedente!

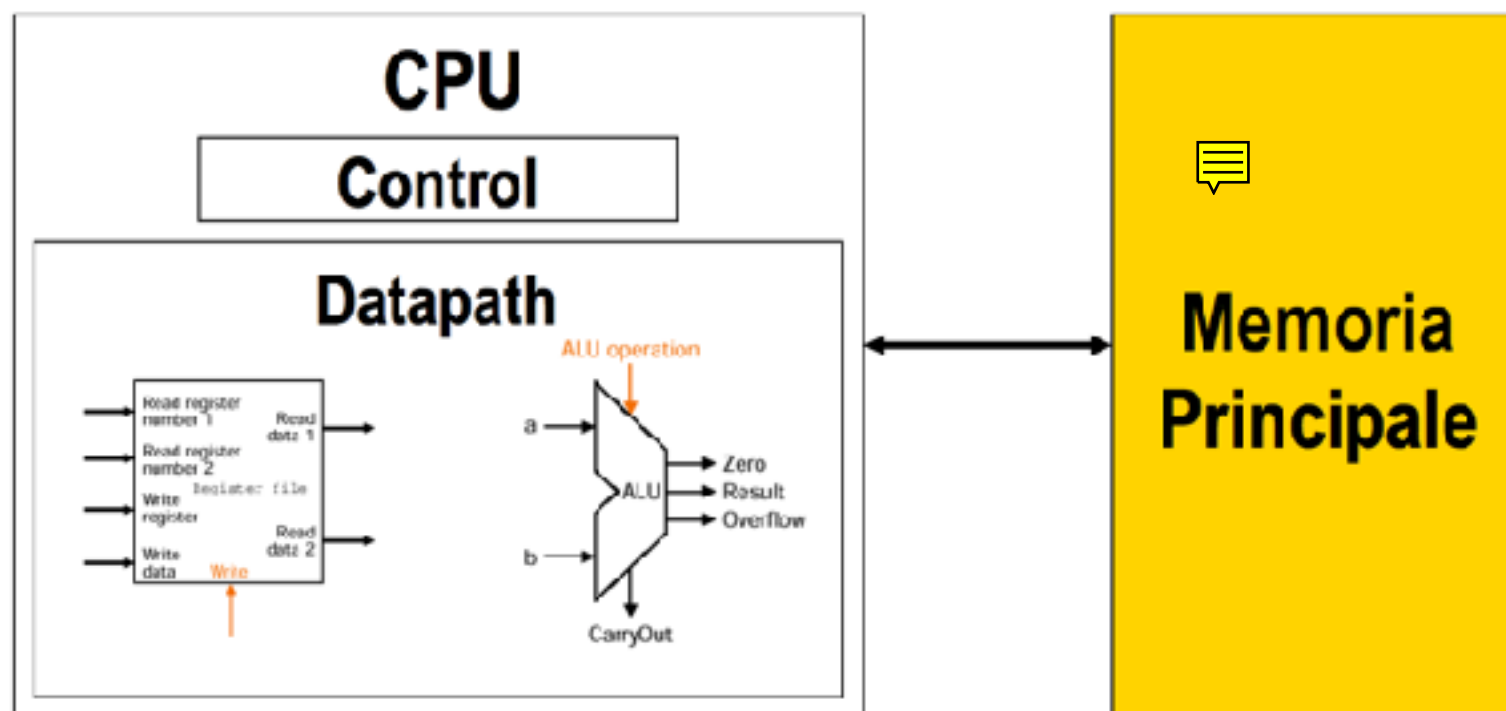


# Altre memorie

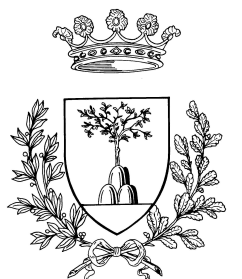
- Non è economicamente sostenibile avere memorie implementate come il Register File ma di dimensioni molto superiori:
  - multiplexer e decoder enormi
  - troppi transistor per bit
- Esistono tecnologie alternative che permettono di avere densità maggiori di bit:
  - costi inferiori
  - prestazioni inferiori



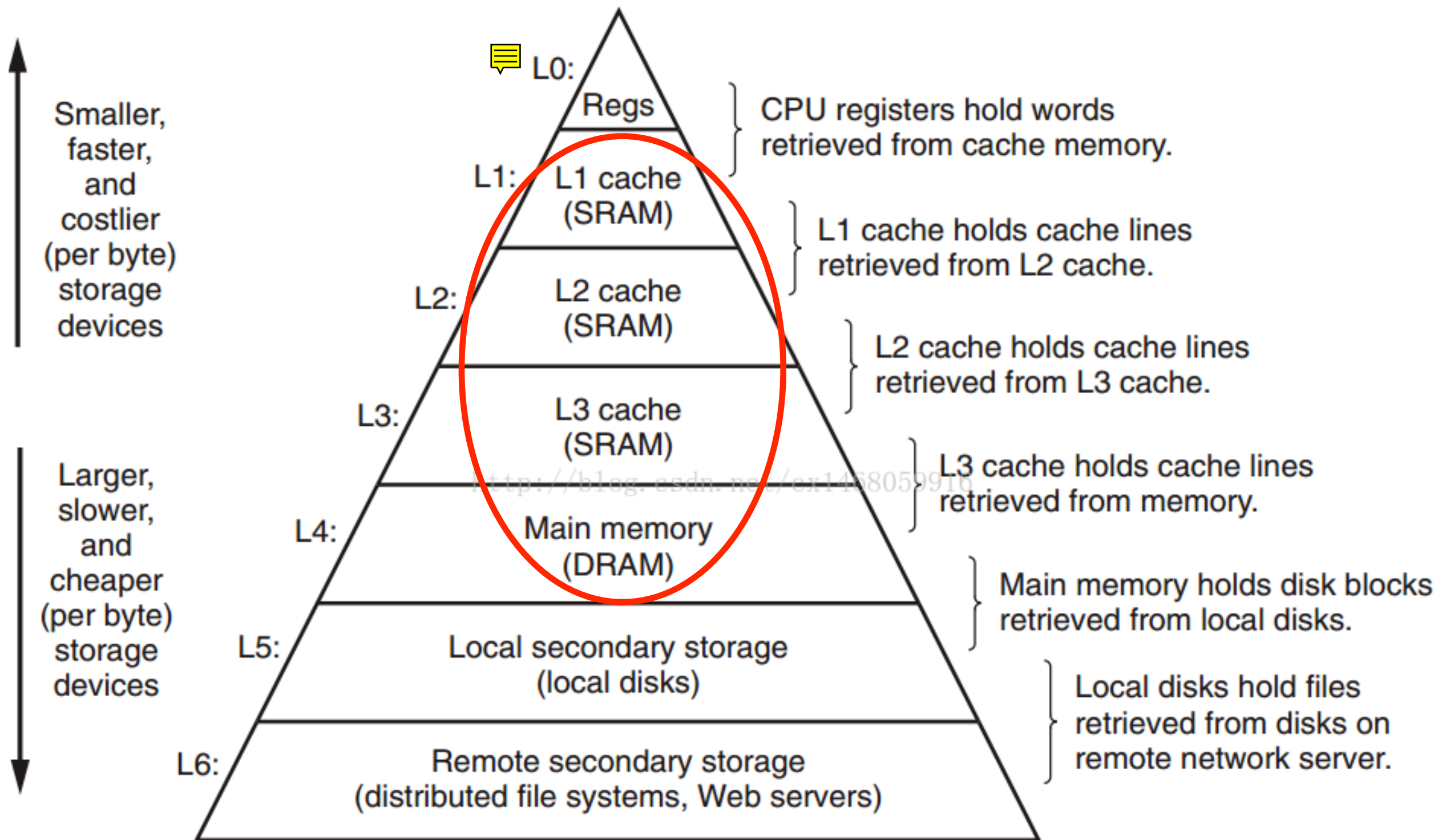
# Memoria principale



- Memoria principale (RAM - Random Access Memory):
  - meno veloce della memoria dei registri, ma molto più capiente
  - è detta RAM perché i tempi di accesso sono indipendenti dal valore dell'indirizzo della cella di memoria acceduta
  - è composta da più livelli (**gerarchie di memoria**)
  - è un tipo di memoria **volatile** (perde i dati se non alimentata)




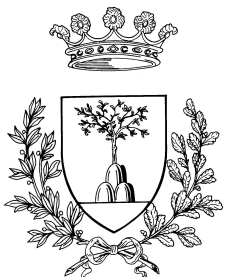
# Gerarchia di memoria





# SRAM

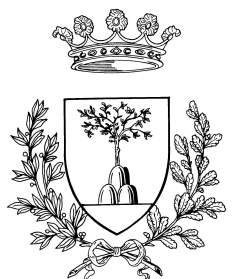
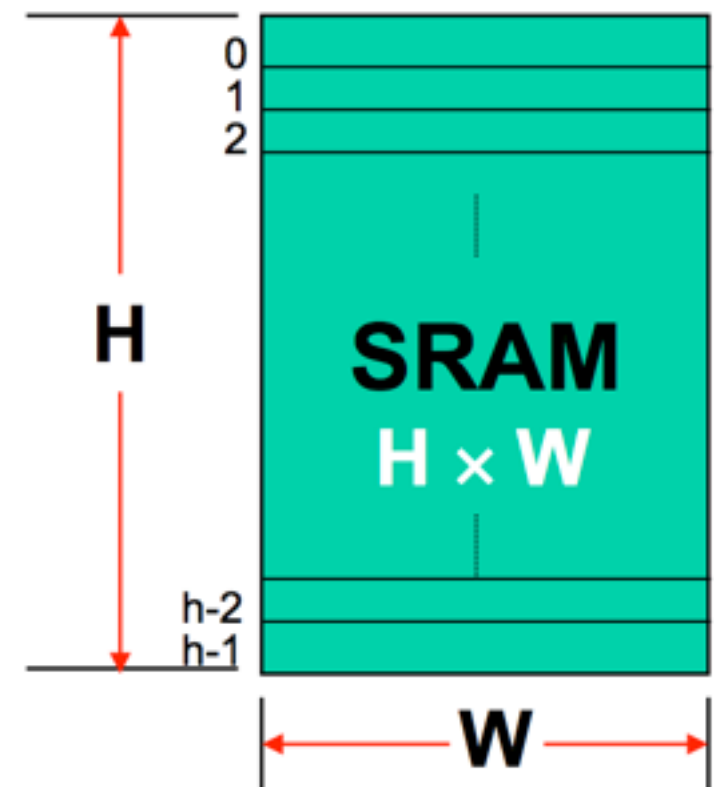
- La SRAM (Static RAM) è più veloce:
  - per la sua realizzazione vengono usati dei flip-flop (~6 transistors per bit)
  - è usata per realizzare memorie veloci, come le **memorie cache**
  - tempi di accesso intorno al nanosecondo
  - dimensioni fino a qualche MB
- Nel passato erano in packages separati, oggi giorno sono integrate nel processore (cache L1 e L2):
  - in realtà il discorso è un pò più complesso (multicore, architetture NUMA, etc...) 





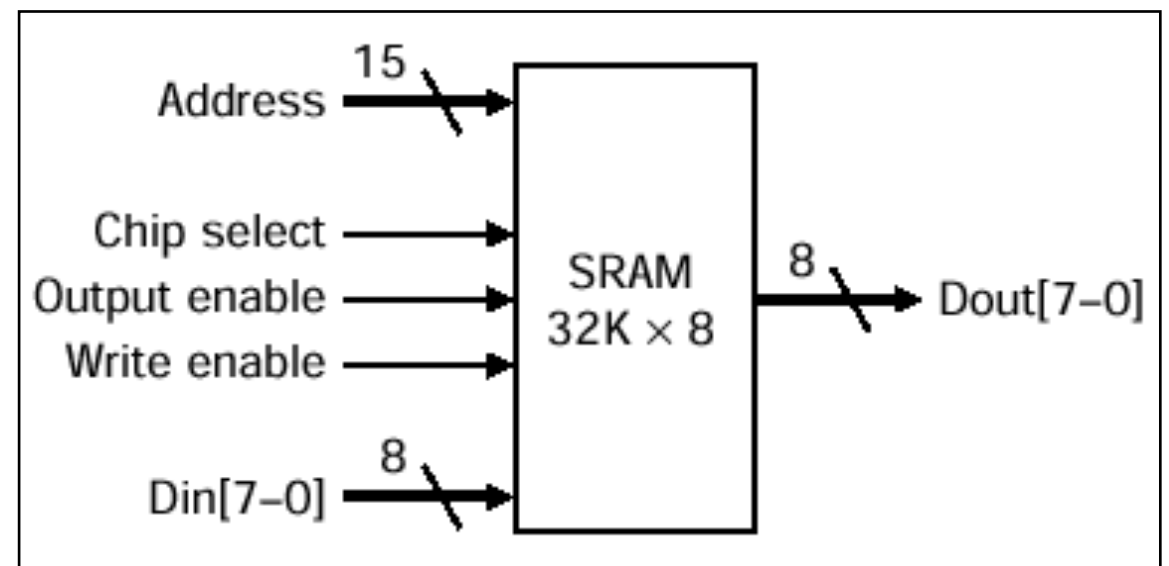
# SRAM

- Le SRAM sono realizzate come matrici di flip-flop **H x W** con:
  - larghezza o ampiezza **W** (numero di flip-flop per ogni riga)
  - altezza **H** (numero di linee indirizzabili)
  - per ragioni costruttive **W** è spesso piccolo
  - **singolo indirizzo** per lettura o scrittura
  - non è possibile scrivere e leggere contemporaneamente, a differenza del Register File
- Numero di bit dell'indirizzo:  
 **$\log_2 H$**   $\rightarrow$  con **N** bit si possono indirizzare fino a  **$2^N$**  linee

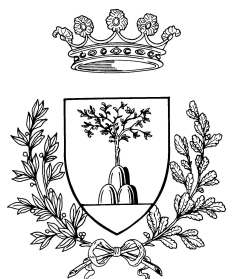


# SRAM

- Esempio di chip 32K x 8:
- **Din** → data in (8 bit)
- **Dout** → data out (8 bit)
- **Address** → indirizzo (15 bit):



- seleziona la linea: 15 bit →  $2^{15}$  righe → 32K righe
- **Chip select** → segnale booleano (attivo alto): abilita scrittura o lettura
- **Output enabled** → segnale booleano (attivo alto): abilita l'uscita del chip su un bus condiviso (per collegare più chip in serie)
- **Write enabled** → segnale booleano (attivo alto): registra nella linea individuata da Address il valore presentato da Din

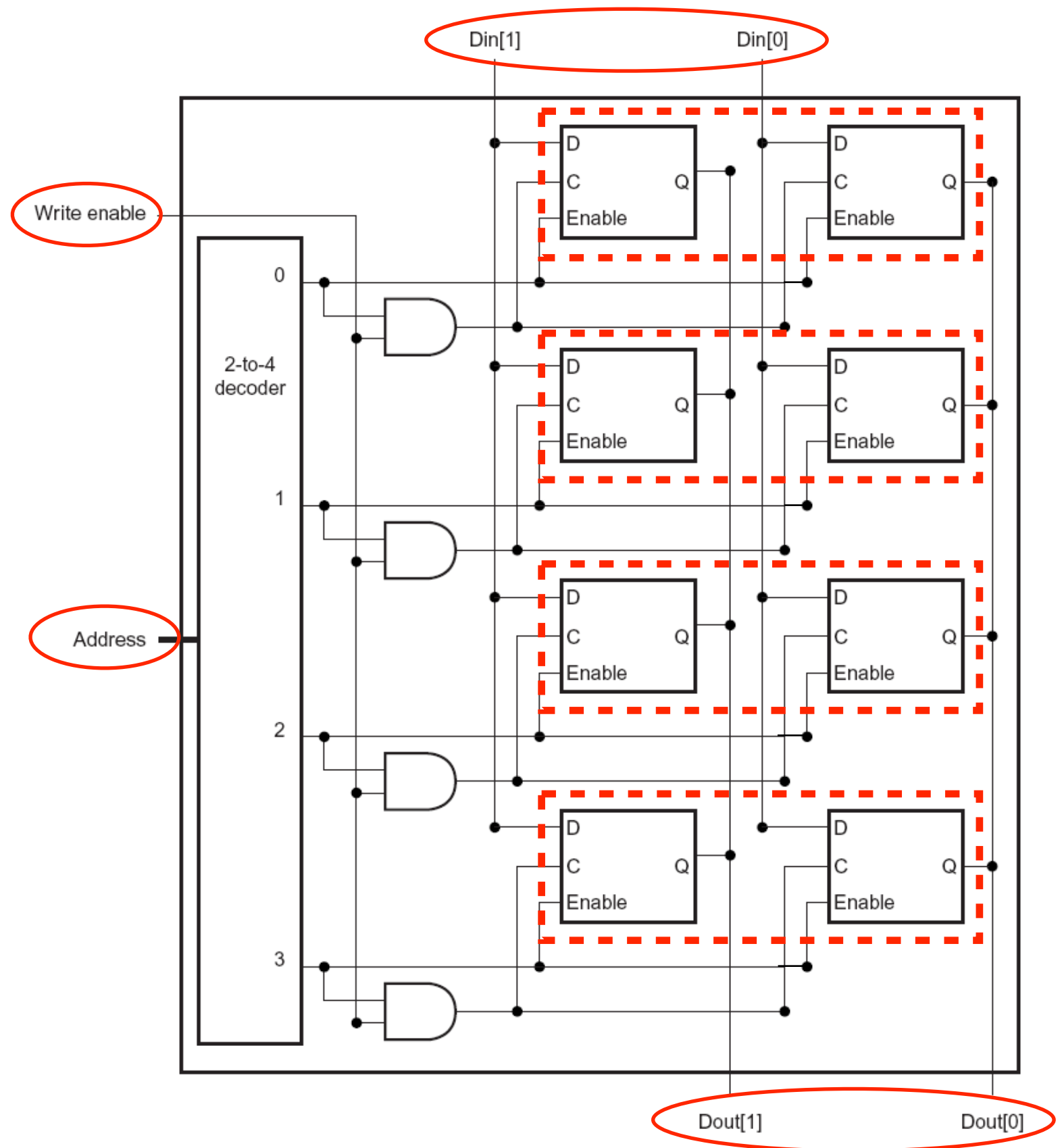



## Esempio di SRAM 4 x 2:

- 4 linee
- 2 bit per linea
- 2 bit di indirizzo
- 2 bit di “dato”

Esempio molto semplice, ma che segue le stesse logiche di SRAM più grandi.

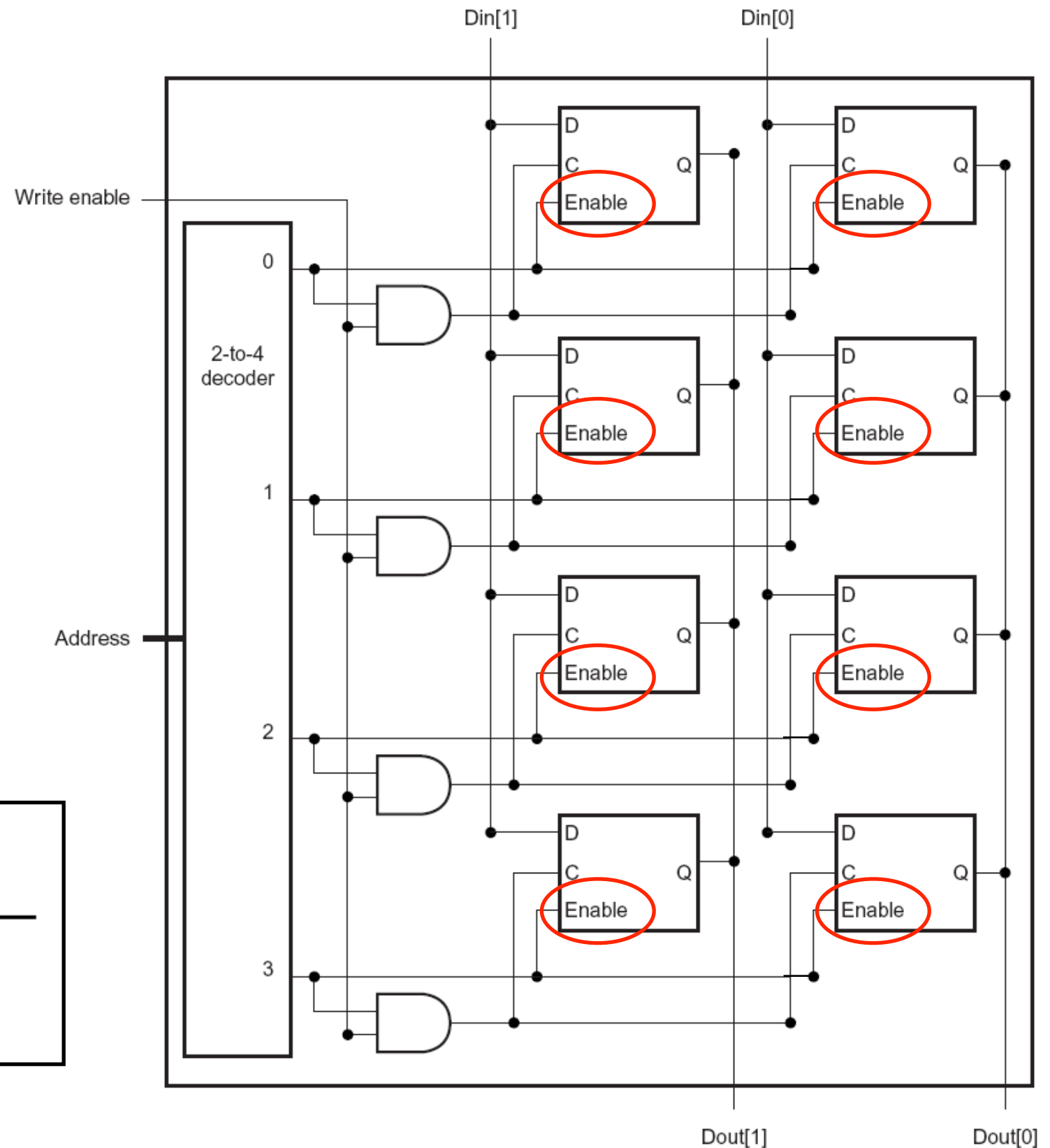
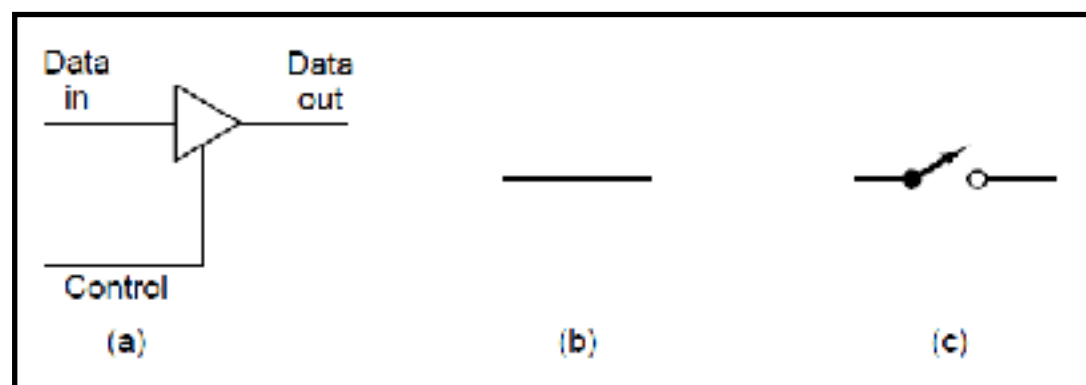
N.B. I flag Output enabled e Chip select sono stati omessi per comodità.



Il multiplexer in uscita (presente nel Register File) è stato rimosso. 

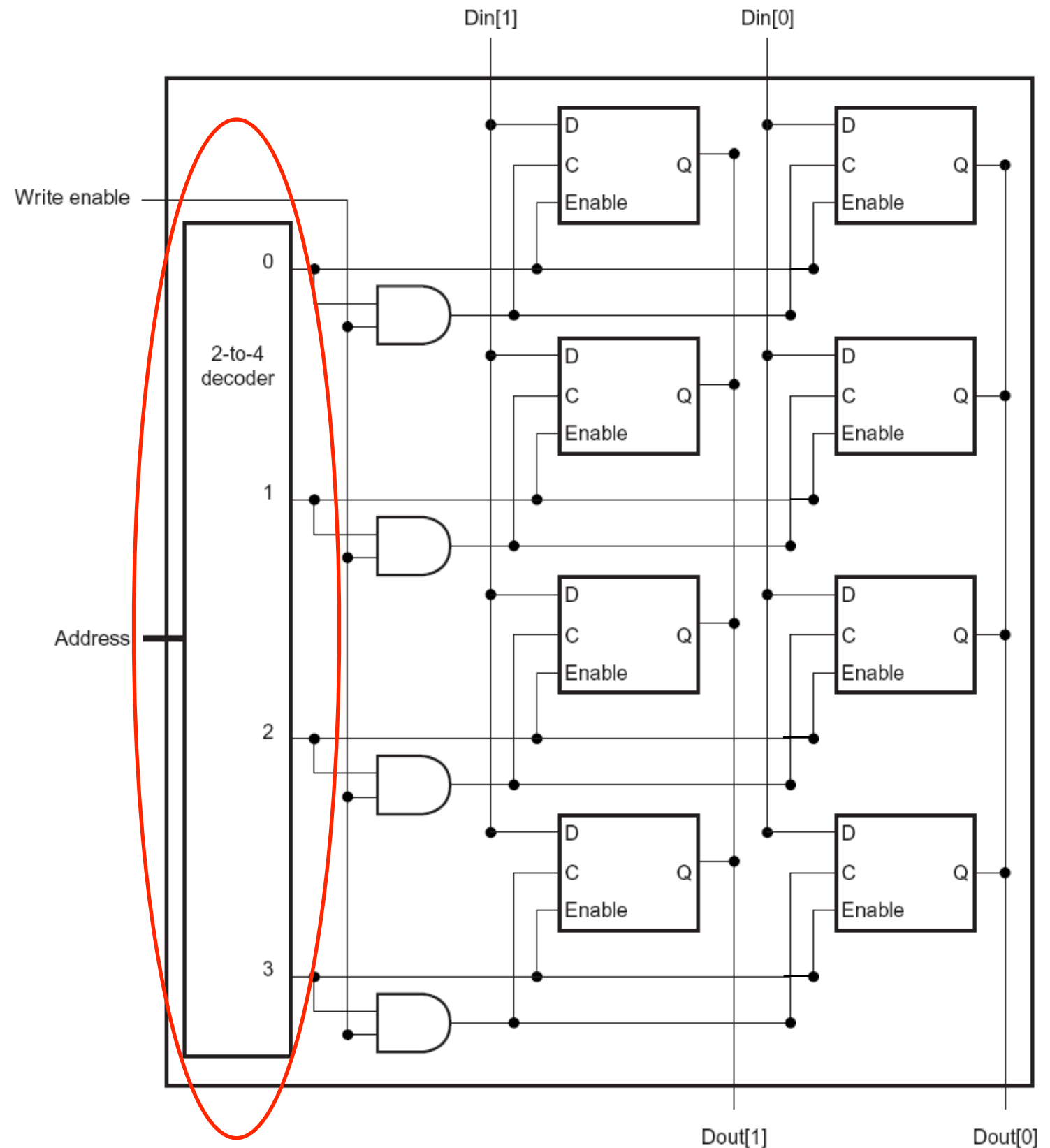
I flip-flop hanno un'ulteriore ingresso "Enable" collegato ad un **tri-state buffer** (a):

- se Control = 1, il circuito è chiuso (b)
- se Control = 0, il circuito è aperto (c)



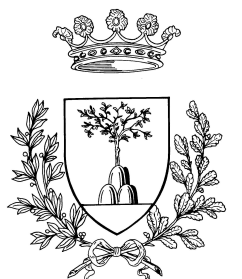
Rimane il problema del decoder iniziale:

- per un alto numero di linee diventa un componente molto costoso
- ci si aspetta che le memorie possano contenere migliaia (o milioni) di linee.



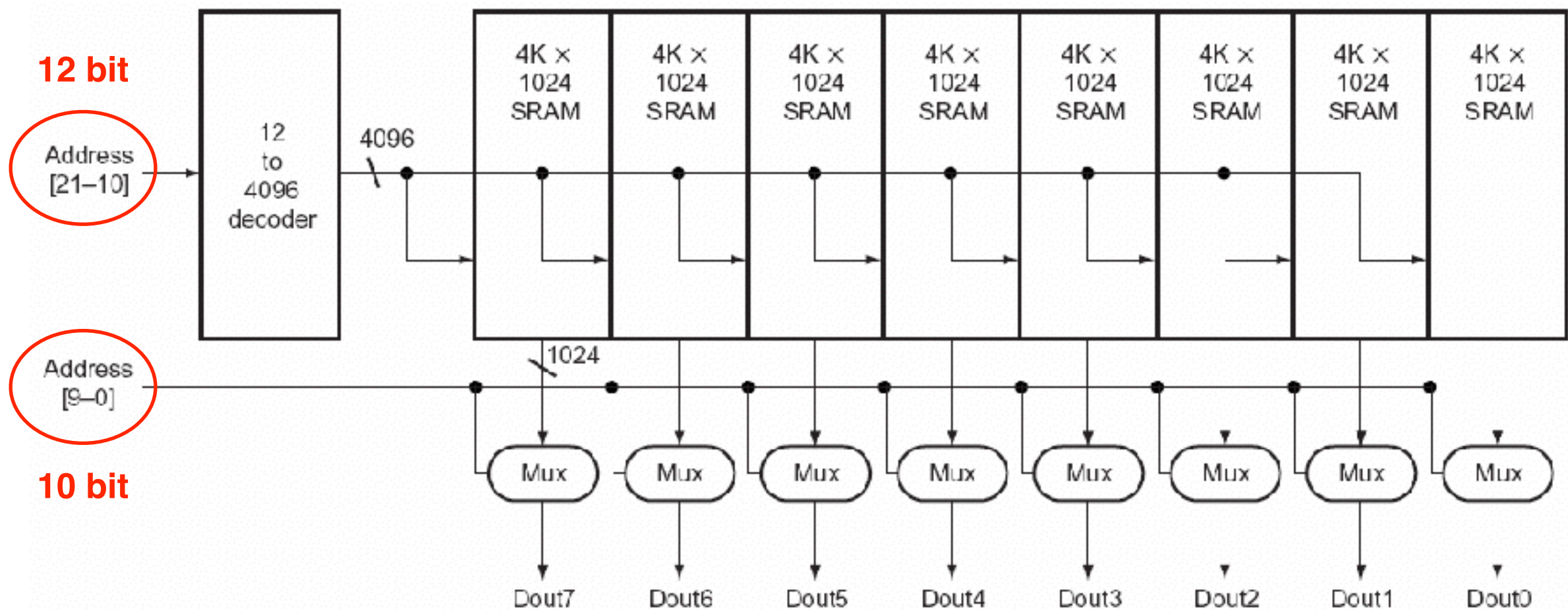
# SRAM a due livelli

- Consideriamo una SRAM di **4M x 8** → servono 22 bit di indirizzamento.
- Il decoder deve avere quindi 22 ingressi (accettabile) e 4 milioni di uscite (non accettabile).
- **Soluzione:** tipicamente le SRAM sono organizzate su due livelli di codifica, che eliminano la necessità di un decoder “enorme”.
- Suddivisione in **8** blocchi da **4K x 1024**:
  - il numero di blocchi corrisponde al numero di colonne (8)
  - la capacità di un singolo blocco corrisponde al numero di celle di memoria ( $4k \times 1024 = 4M$ )



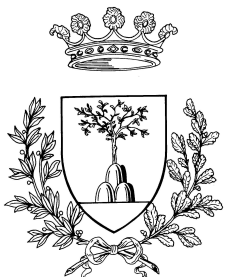
# SRAM a due livelli

- Parte **alta** dell'indirizzo [21-10] seleziona la medesima linea di ogni blocco da  $4K \times 1024$  bit attraverso un decoder.
- Parte **bassa** dell'indirizzo [9-0] seleziona uno dei 1024 bit in output da ogni blocco, attraverso una batteria di 8 multiplexer.



# DRAM

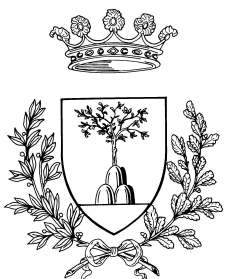
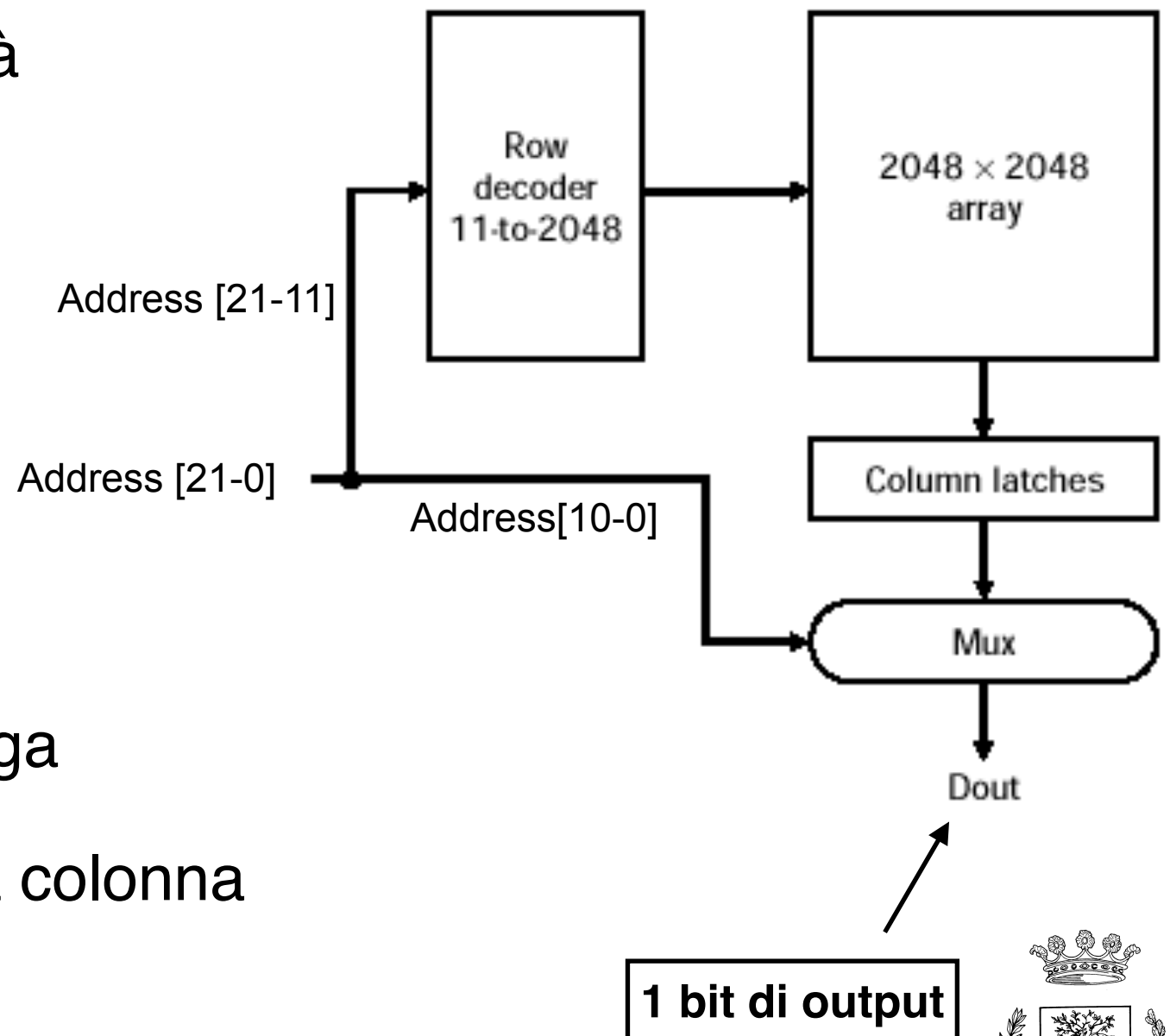
- La DRAM (Dynamic RAM) è più capiente ma più lenta:
  - tempi di accesso intorno al centinaio di nanosecondi
  - ogni bit è memorizzato tramite un **condensatore** (memoria più densa della SRAM)
  - i condensatori perdono la loro carica: è necessario **rinfrascare** il contenuto delle DRAM a intervalli di tempo prefissati (~1ms)
  - è usata per realizzare memorie capienti e di accesso meno frequente (come la **memoria principale**)





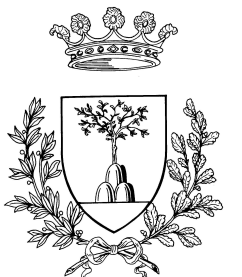
# DRAM

- La DRAM è organizzata su due livelli (come la SRAM).
- Segue una **struttura quadrata**:
  - $2^N \times 2^N \rightarrow 4^N$  bit di capacità
- Es. DRAM di  $2^{22}$  bit  $\rightarrow$  4Mb con organizzazione 4M x 1:
  - indirizzo totale 22 bit
  - indirizzo spezzato in 2 pezzi da 11 bit
  - la parte alta seleziona la riga
  - la parte bassa seleziona la colonna



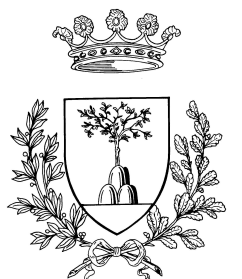
# Design della memoria

- Gli esempi visti nelle ultime slides hanno mostrato il contenuto di un singolo chip di memoria (SRAM o DRAM).
- Solitamente i chip forniscono in output 1, 4, 8 o 16 bit.
- Singoli chip possono essere combinati assieme per creare memorie più capienti e capaci di leggere/scrivere una quantità maggiore di bit.
  - ad esempio per poter leggere e scrivere delle word (32 bit in MIPS)
- Per estendere la quantità di bit che si possono gestire con una singola operazione di lettura/scrittura, si raggruppano i chip di memoria in **set**:
  - ogni chip fornisce una parte dei bit del dato in uscita



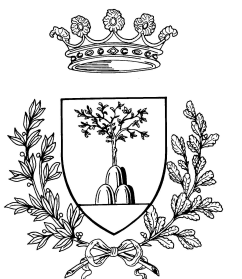
# Design della memoria

- Per aumentare la capacità della memoria, si aumenta il numero di set:
  - il set corretto viene selezionato attraverso un decoder
- Le specifiche di una memoria sono:
  - **capacità** (solitamente espressa in byte)
  - **unità di indirizzamento** (quanti bit mi identifica un indirizzo, solitamente un byte)
  - **unità di lettura/scrittura** (quanti bit accedo con una singola lettura/scrittura, solitamente pari ad una parola dell'architettura)



# Esempio 1

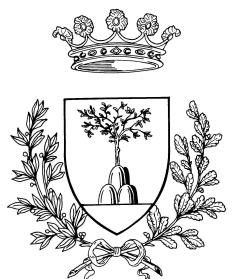
- Realizzare una memoria di 2 MB con 32 bit di lettura/scrittura e unità di indirizzamento di 32 bit, utilizzando chip da 512K x 8.
- Quanti chip mi servono?
  - 2 MB  $\rightarrow$  16 Mb (bit totali della memoria)
  - 512K x 8  $\rightarrow$  4 Mb (bit del singolo chip)
  - abbiamo bisogno di **4 chip**
- Quanti chip per set?
  - 32 bit di lettura/scrittura
  - ogni chip ha 8 bit di lettura/scrittura (512K x **8**)
  - ogni set contiene  $32 / 8 = \mathbf{4 \text{ chip}}$   $\rightarrow$  avremo quindi 1 set



# Esempio 1

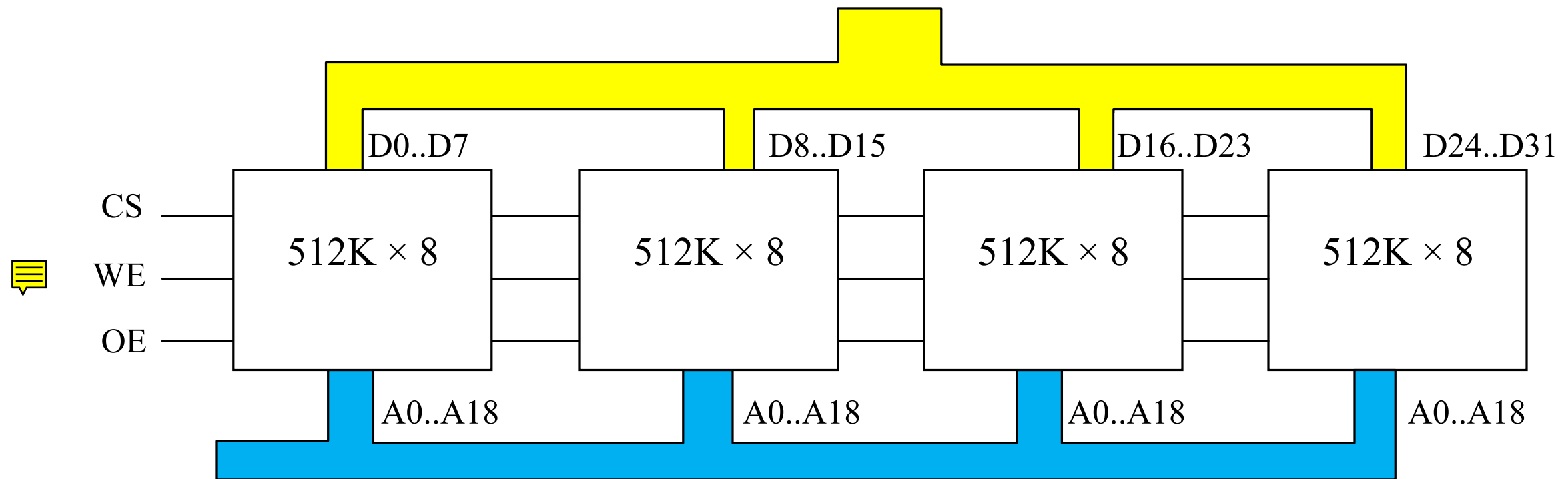
- Da quanti bit è formato ogni indirizzo?
  - 2 MB → 16 Mb (bit totali della memoria)
  - 16 Mb / 32 b (unità di indirizzamento) = 512K linee
  - $\log_2 512K = 19 \rightarrow$  ogni indirizzo è composto da **19 bit**

$512 \times 2^{10}$   
e non  
 $512 \times 10^3$



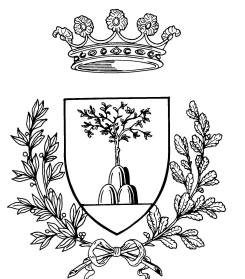
# Esempio 1

Bus Dati (D0..D31) – unione dei bus dati



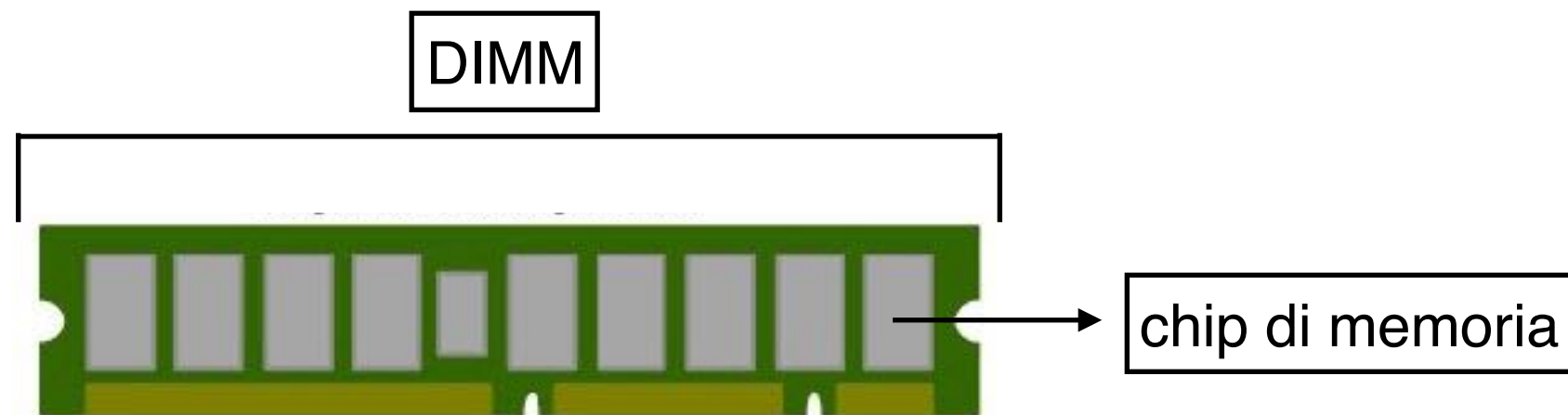
Bus Indirizzi (A0..A18) – comune a tutti

- I 19 bit dell'indirizzo e i bit CS, WE e OE sono comuni a tutti i chip.
- Tutti i chip lavorano sempre insieme.
- I gruppi di 8-bit dei singoli chip vengono affiancati a costituire i 32 bit richiesti in uscita.



# Esempio 1 - DIMM

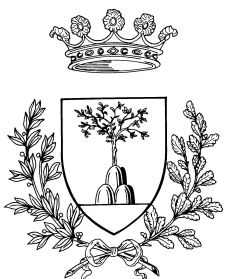
- L'esempio appena descritto rappresenta il normale schema di collegamento di singoli chip di memoria per la realizzazione di un modulo **DIMM**.
- Ad esempio per realizzare un modulo DIMM di capacità 1 GB (organizzato 128M×64) possiamo usare 8 chip con capacità 1024 Mb (organizzazione 128M×8).



- Nell'esempio si può notare che ci sono 9 chip (quello piccolo centrale è il controller, non un chip di memoria):
- $8 \text{ bit} \times 9 = 72 \text{ bit} \rightarrow 64 \text{ bit} + 8 \text{ per controllo errori (lo vedremo più avanti nel corso)}$

# Esempio 2

- Realizzare una memoria di 2 MB con 8 bit di lettura/scrittura e unità di indirizzamento di 8 bit, utilizzando chip da 512K x 8.
- Quanti chip mi servono?
  - 2 MB  $\rightarrow$  16 Mb (bit totali della memoria)
  - 512K x 8  $\rightarrow$  4 Mb (bit del singolo chip)
  - abbiamo bisogno di **4 chip**
- Quanti chip per set?
  - 8 bit di lettura/scrittura
  - ogni chip ha 8 bit di lettura/scrittura (512K x **8**)
  - ogni set contiene  $8 / 8 = \mathbf{1 \text{ chip}}$   $\rightarrow$  avremo quindi 4 set

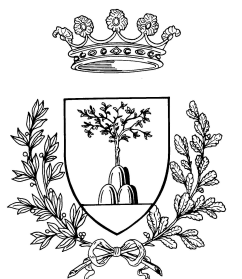




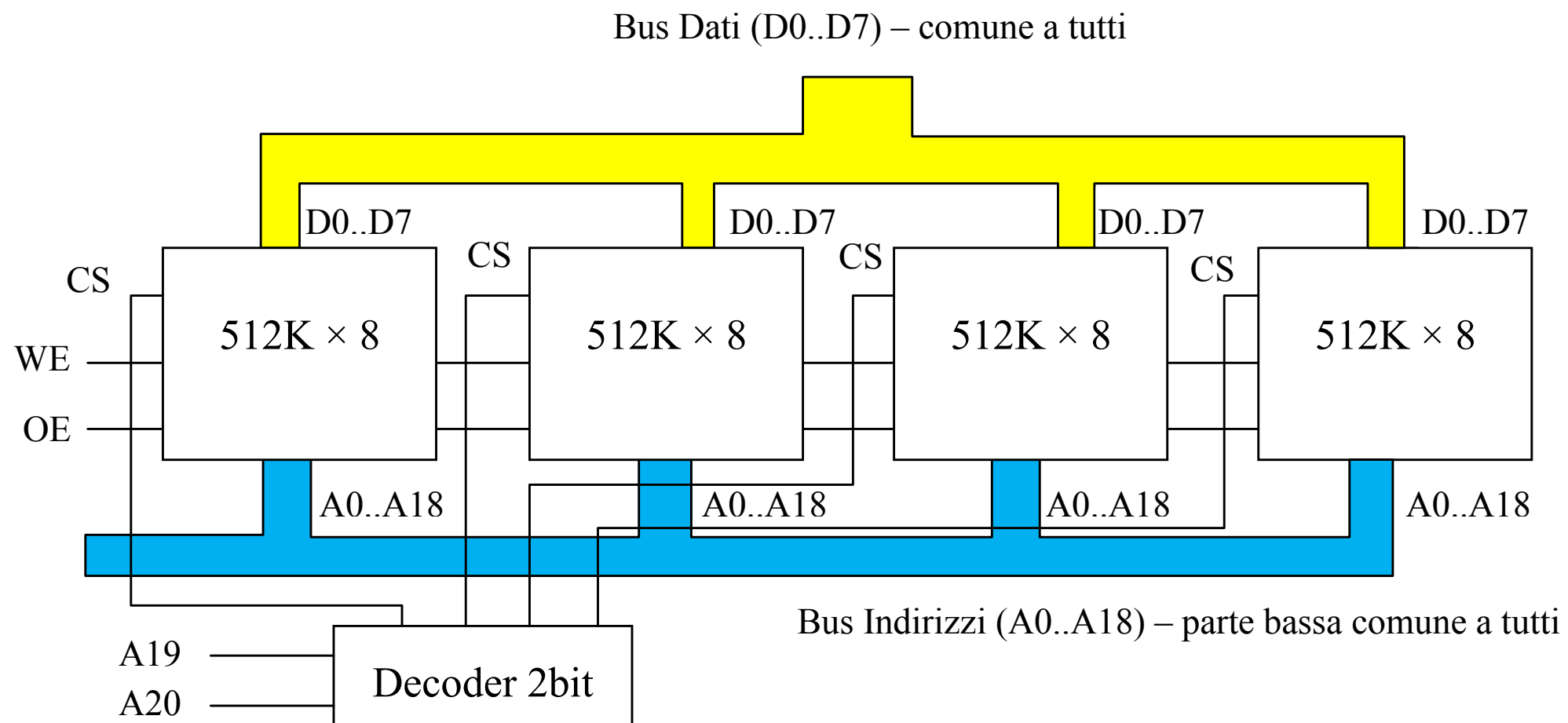
# Esempio 2

$2 \times 2^{20}$   
e non  
 $2 \times 10^6$

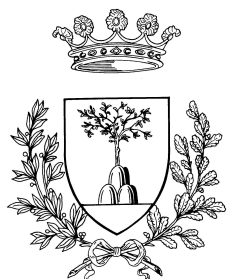
- Da quanti bit è formato ogni indirizzo?
- $2 \text{ MB} \rightarrow 16 \text{ Mb}$  (bit totali della memoria)
- $16 \text{ Mb} / 8 \text{ b}$  (unità di indirizzamento) = 2M linee
- $\log_2 2\text{M} = 21 \rightarrow$  ogni indirizzo è composto da **21** bit



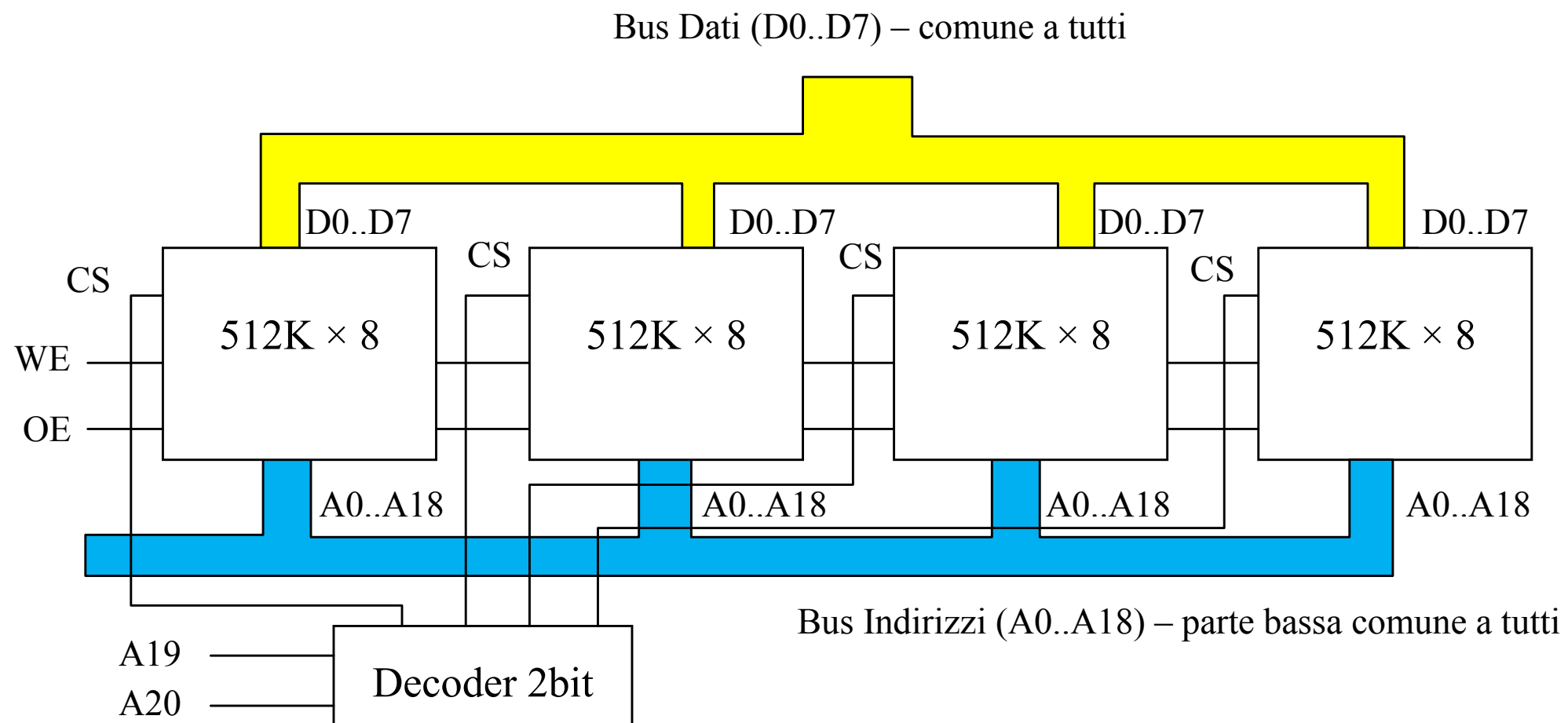
# Esempio 2



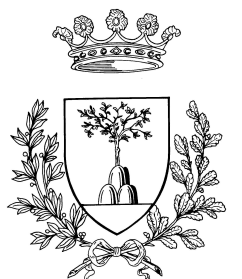
- I bus dati possono essere collegati insieme (uno solo chip per volta è attivo e quindi non ci sono conflitti).
- Stesso dicasi per i 2 bit di input WE e OE.



# Esempio 2

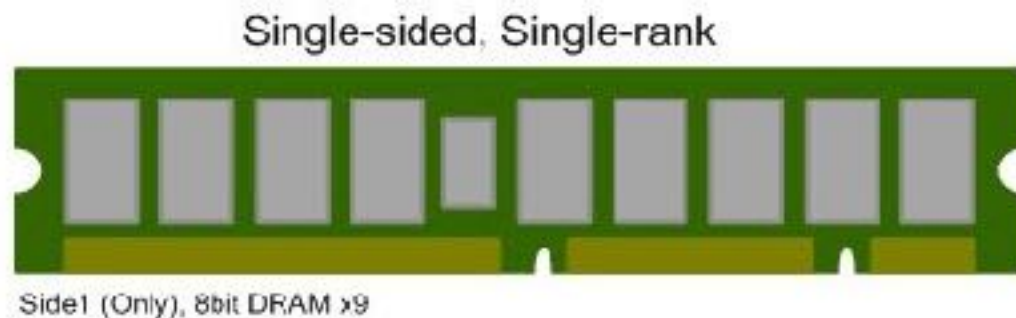


- Per poter indirizzare 2M parole occorre estendere il bus indirizzi da 19 a 21 bit:
- i due bit più significativi degli indirizzi (A19 e A20) sono utilizzati per selezionare (attraverso un decoder) quale chip attivare (**collegamento a CS**).

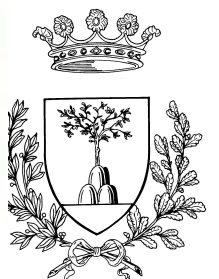
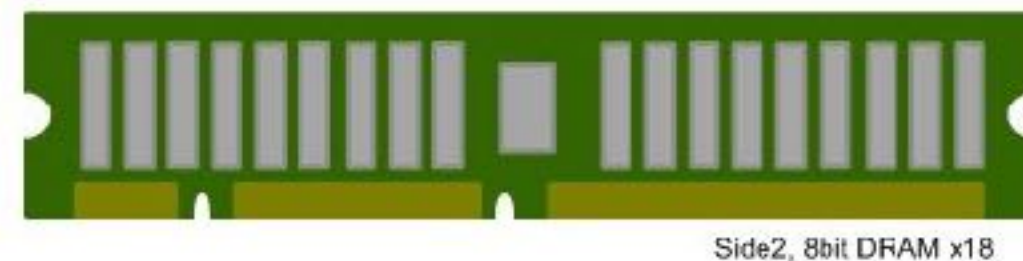
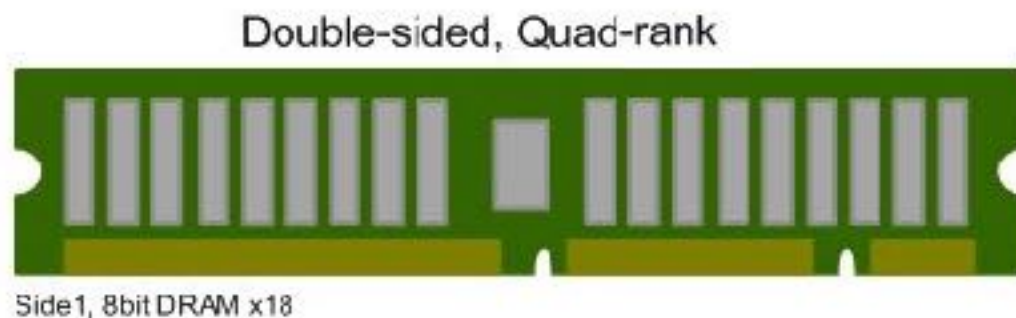
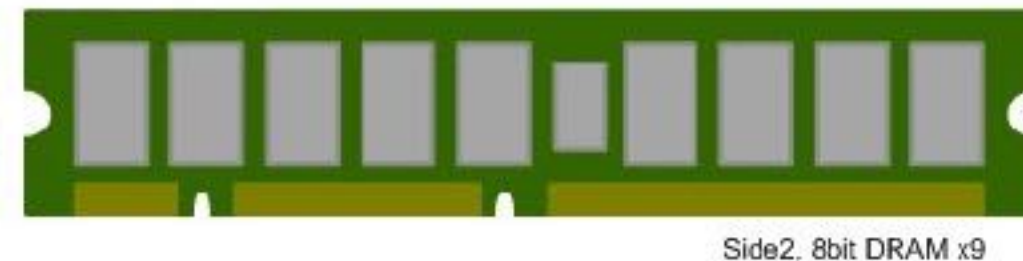
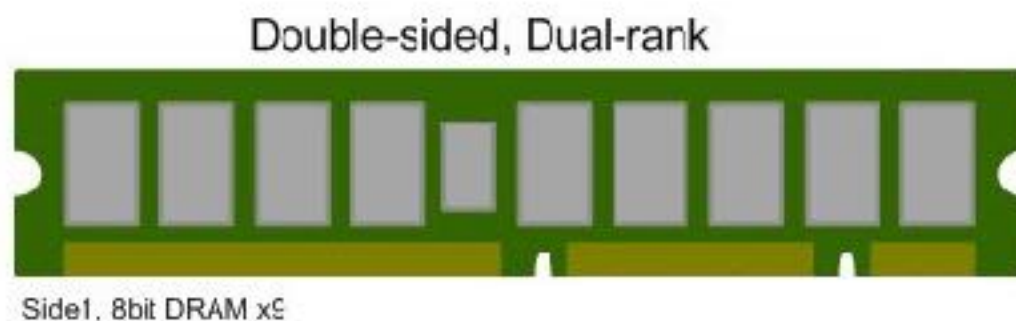
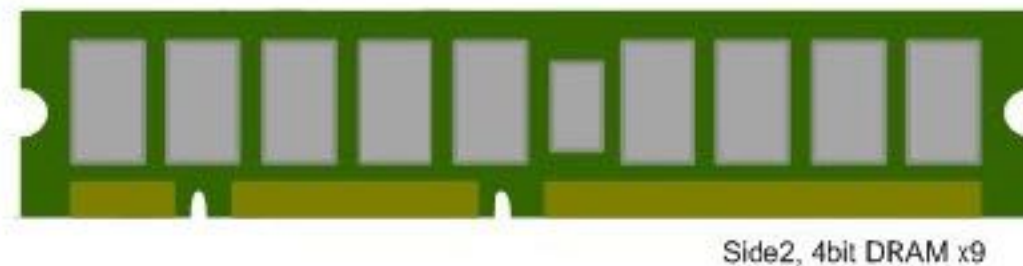
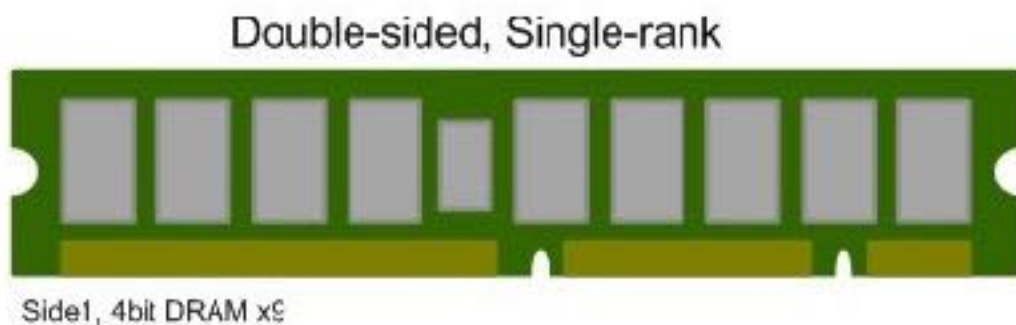


# Esempio 2 - Rank

- Questo tipo di schema (o meglio un caso ibrido tra questo e il precedente) è utilizzato per organizzazione di DIMM in **rank**.



Es. 1 rank = 9 chip x8  
(ogni chip ha parole da 8 bit)



# Synchronous RAM

- Le memorie SRAM e DRAM sono asincrone.
- Per diminuire il gap di prestazioni tra la CPU e le memorie sono state inventate le Synchronous SRAM (SSRAM) e Synchronous DRAM (SDRAM).
- Queste memorie seguono i cicli di clock del bus di comunicazione del calcolatore (comunque più lento del clock della CPU):
  - sono in grado di servire una richiesta dati per ciclo di clock
  - le Double Data Rate (DDR) invece sono in grado di servire due richieste dati per ciclo di clock.

