Architettura degli Elaboratori

Memorie e circuiti sequenziali

slide a cura di Salvatore Orlando, Andrea Torsello, Marta Simeoni

Circuiti combinatori vs 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**

=> ovvero, che dipendono anche da informazioni memorizzate in **elementi di memoria** interni

- in generale, la funzione calcolata dal circuito sequenziale ad un dato istante dipende dalla *sequenza temporale* dei valori in input al circuito
- la *sequenza temporale* determina infatti il valore memorizzato nello stato

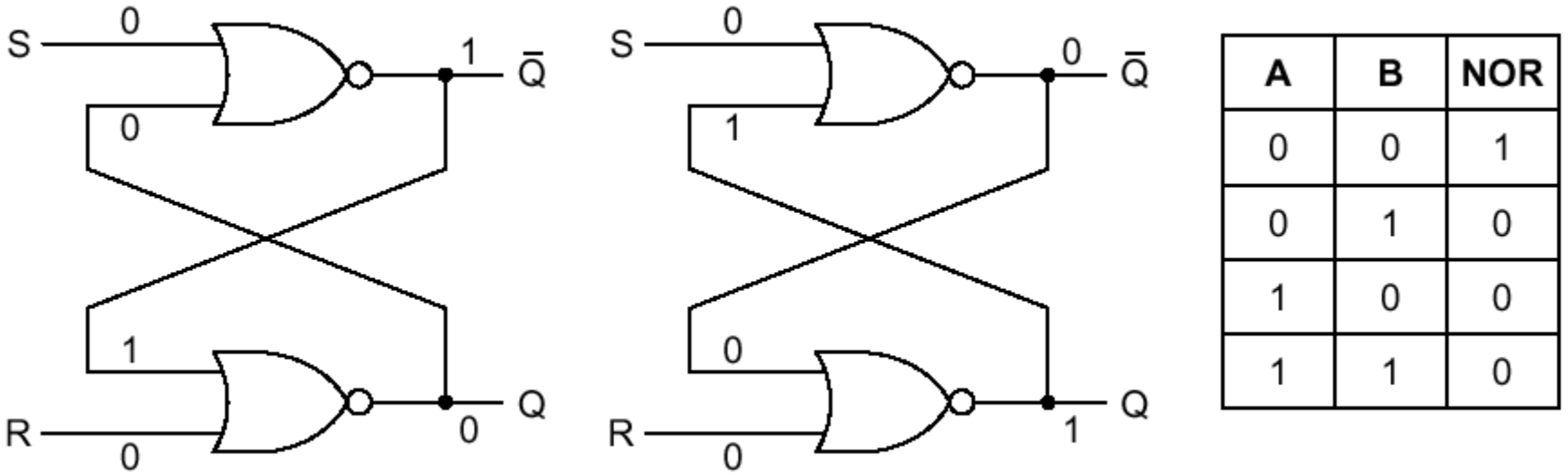
Elementi di memoria

Per realizzare circuiti sequenziali è necessario un elemento di memoria per memorizzare lo stato

Possiamo organizzare le porte logiche in modo da realizzare un elemento di memoria ?

- Sì, un elemento in grado di memorizzare un singolo bit è il latch

S-R Latch



A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

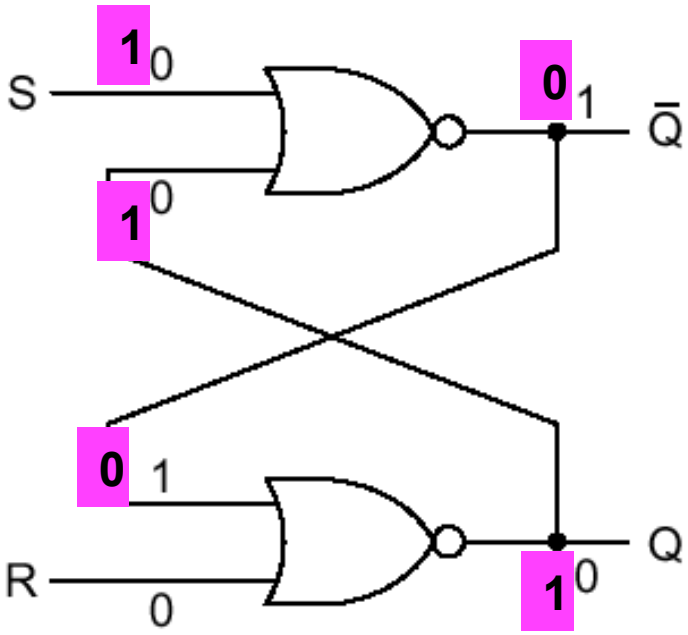
L'S-R Latch è un circuito, composto da 2 porte NOR concatenate, che costituisce l'elemento base per costruire elementi di memoria più complessi

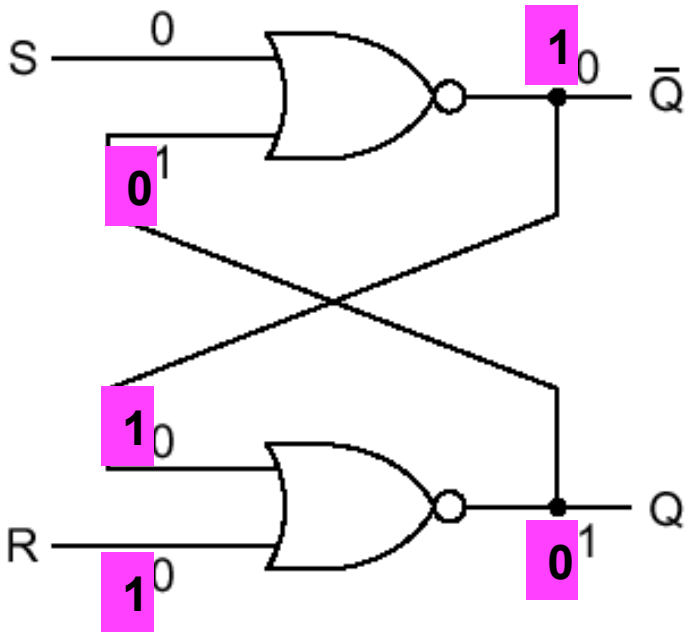
- S sta per Set e R sta per Reset

Anche se (S,R)=(0,0), gli output del latch possono comunque essere diversi

- l'output è infatti il valore memorizzato nel latch
- verificare infatti che il latch a sinistra memorizza il valore 0, mentre quello a destra memorizza il valore 1

Set / Reset del latch





A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

Poniamo (S,R)=(1,0) per effettuare il setting del latch a sinistra

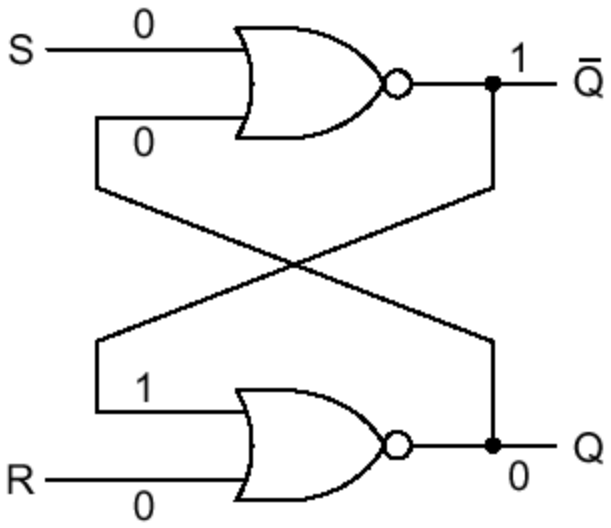
- il valore memorizzato passa da 0 a 1 (e viene poi mantenuto riportando S=0)

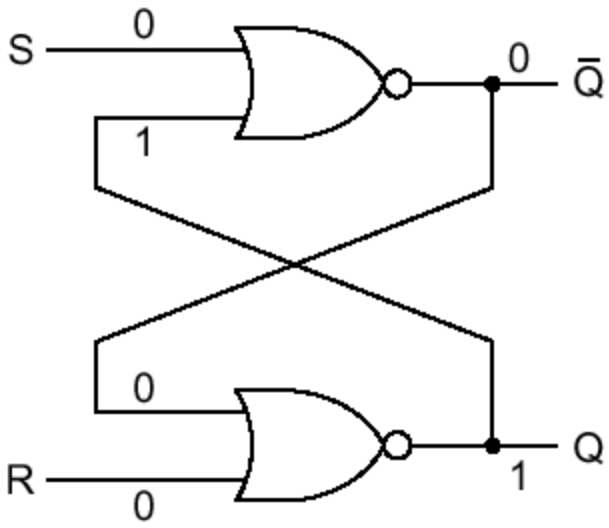
Poniamo (S,R)=(0,1) per effettuare il resetting del latch a destra

- il valore memorizzato passa da 1 a 0 (e viene poi mantenuto riportando R=0)

La combinazione (S,R) = (0,0) viene detta combinazione di riposo, perché semplicemente mantiene il valore memorizzato in precedenza

S-R Latch





A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

La combinazione di valori (S,R)=(1,1) non deve mai esser presentata al latch

- succede che Q=~Q=0, ma il valore memorizzato può essere arbitrario quando resettiamo S e R (dipende dall'ordine del resetting).

Clock

I segnali S e R devono essere stabili, e valere (1,0) o (0,1) per poter memorizzare un valore corretto

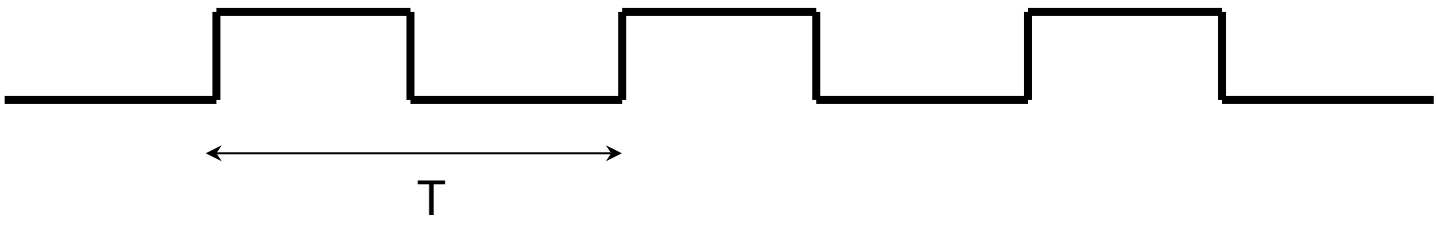
Ma (S,R) sono di solito calcolati da un circuito combinatorio

- l'output del circuito diventa stabile dopo un certo intervallo di tempo
- è possibile calcolare questo intervallo di 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 potrebbero essere memorizzati valori errati

Clock

Soluzione $\Rightarrow$ usiamo un segnale a gradino, detto clock, il cui periodo (ovvero l'intervallo di tempo tra l'inizio di un gradino e il successivo) viene scelto abbastanza *grande* 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)



Unità di misura del clock

Se il periodo T è espresso in *sec*

- Frequenza di clock: Freq = 1/T Hz (numero di cicli al secondo)

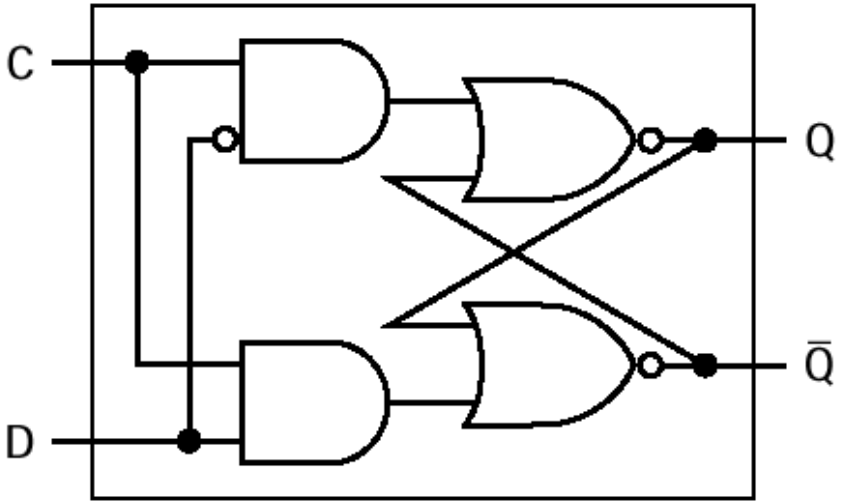
Se T = 10 nsec, qual è la frequenza del clock?

- 1 nsec = 10^{-9} sec 1 μsec = 10^{-6} sec 1 msec = 10^{-3} sec
- Freq = 1/T = 1 / ($10*10^{-9}$) = 10^{8} Hz = 10^{8} Hz = $10^{2} * 10^{6}$ Hz = 100 MHz

Se T = 1 nsec, qual è la frequenza del clock?

- 1 nsec = 10^{-9} sec
- Freq = 1/T = $1/10^{-9}$ = 10^{9} Hz = 1 GHz

Latch clockato (D-latch)



NOTA: **latch disegnato *sottosopra***
rispetto alla figura precedente
=> posizioni invertite di (S, R)
e di (Q, ~Q)

D=1 corrisponde al *setting*

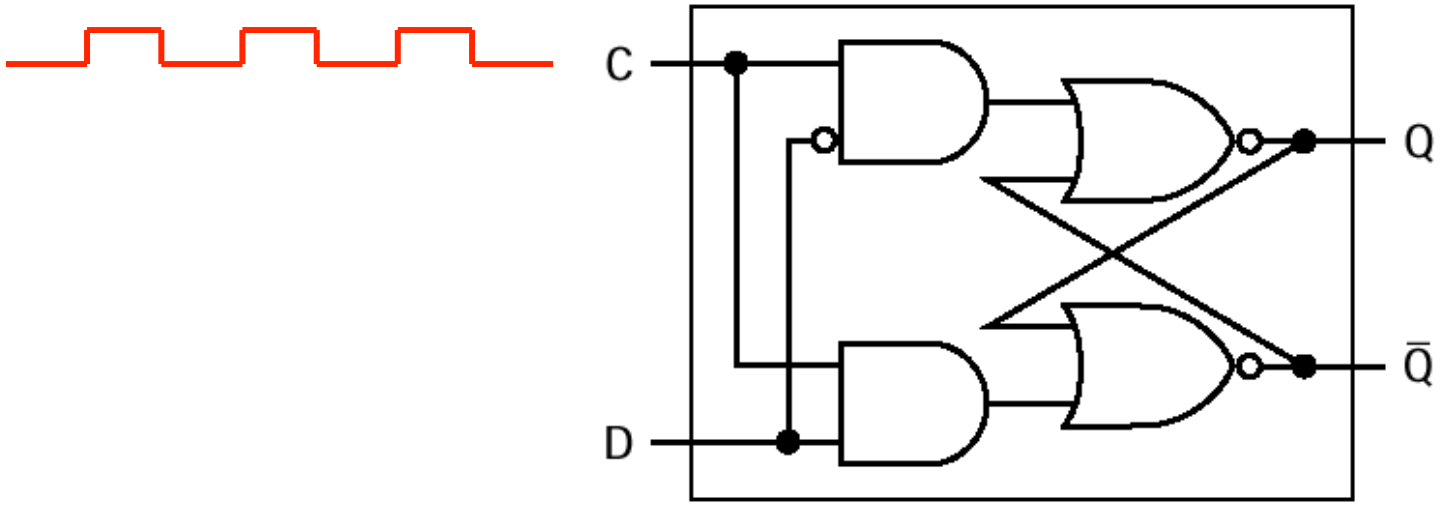
- S=1 e R=0

D=0 corrisponde al *resetting*

- S=0 e R=1

La combinazione S=1 e R=1 non può mai verificarsi

Latch clockato (D-latch)

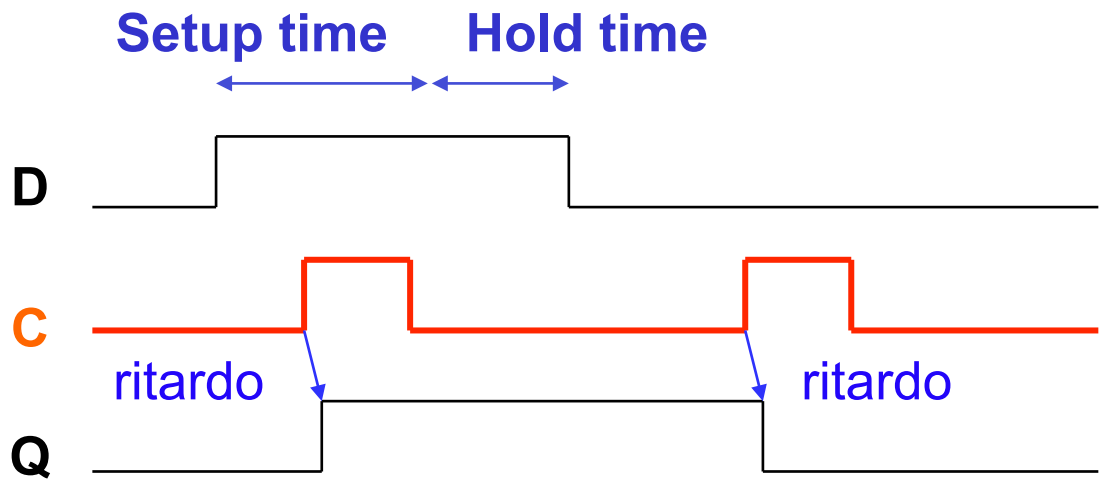


A causa delle 2 porte AND, quando il clock è *deasserted* abbiamo che nel latch non viene memorizzato alcun valore: S=0 e R=0 (viene mantenuto il valore precedentemente memorizzato)

Viene memorizzato un valore (in dipendenza del valore di D) solo quando il clock è *asserted*

Se il segnale C fosse invertito (porta NOT aggiuntiva), il latch memorizzerebbe sul valore *basso* del clock (cioè quando il clock è *deasserted*)

Diagramma temporale del D-latch

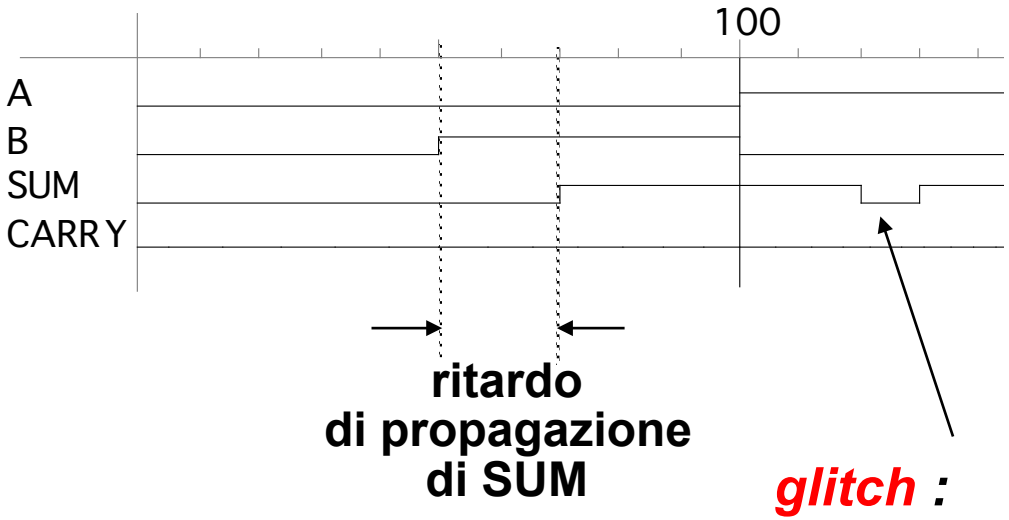


Il segnale D, ottenuto solitamente come output di un circuito combinatorio

- deve essere già stabile quando C diventa asserted
- 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)

Ritardi nella propagazione dei segnali nei circuiti

Diagramma temporale di 1-bit adder (half-adder)



glitch : **1+0=1, non 0 !**

I circuiti reali hanno ritardi non-nulli, che dipendono dai cammini, ovvero dalle porte e dai fili, attraversati dai segnali

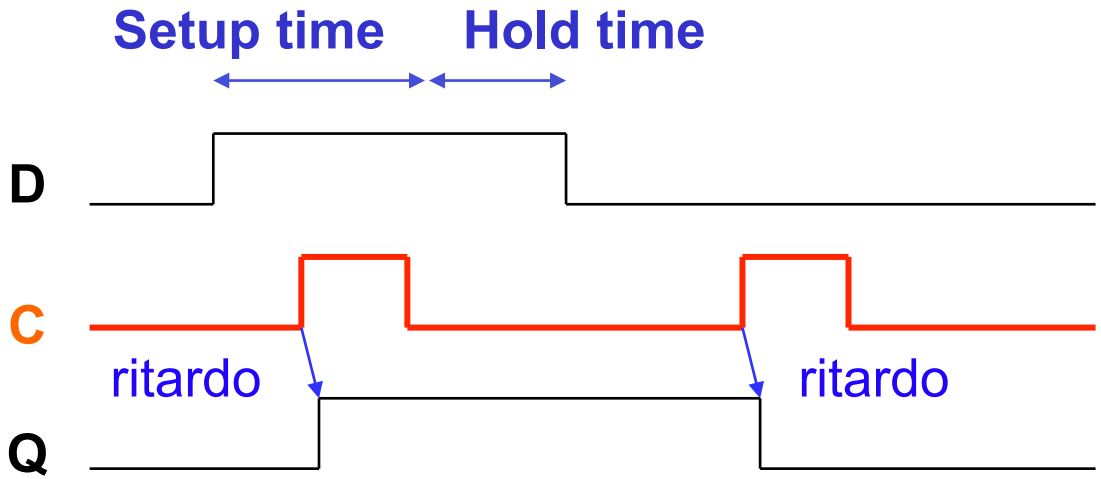
Gli output possono temporaneamente cambiare da valori corretti a valori errati, e ancora a valori corretti

- questo fenomeno è noto come glitch
- dopo un certo intervallo, con alta probabilità i segnali si *stabilizzano*

Periodo del ciclo di clock

Il periodo T deve essere scelto abbastanza lungo affinché l'output del circuito combinatorio si stabilizzi

- deve essere stabile un po' prima del periodo di apertura del latch (setup time), e lo deve rimanere per un certo tempo (hold time)

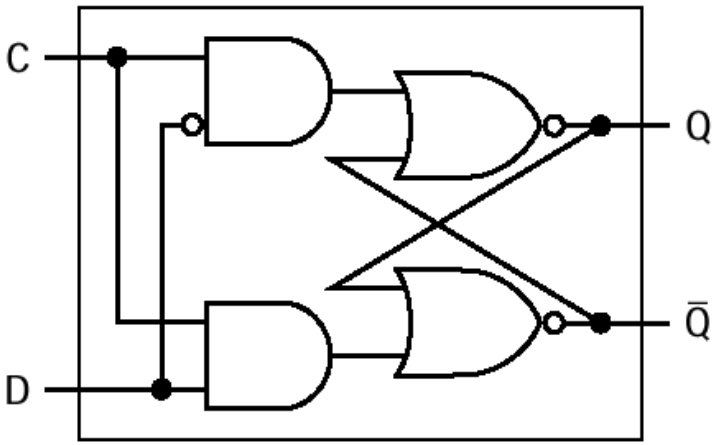


D-latch clockato: fenomeno della trasparenza

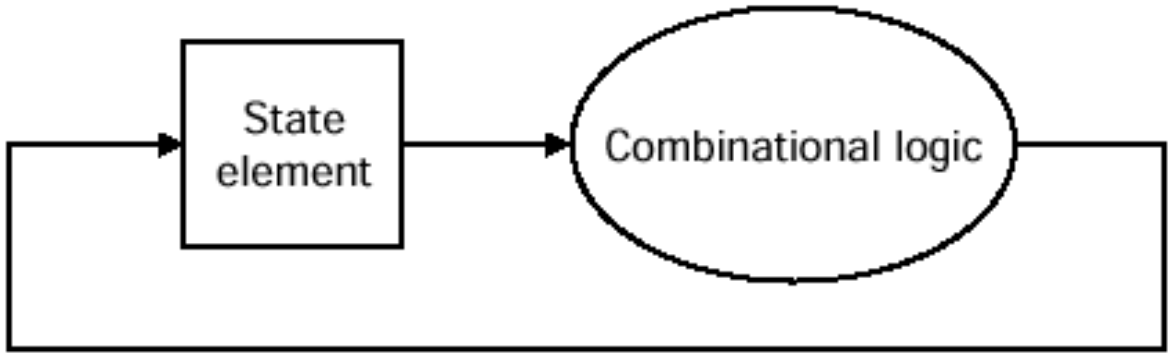
Osserviamo il seguente comportamento del D-latch clockato:

- durante l'intervallo alto del clock il valore del segnale di ingresso D viene memorizzato nel latch
- il valore di D si propaga immediatamente (o quasi) all'uscita Q
- ma anche le eventuali variazioni di D si propagano immediatamente, col risultato che Q può variare più volte durante l'intervallo alto del clock
- solo quando il clock torna a zero Q si stabilizza
- possiamo concludere che durante l'intervallo alto del clock il latch non esercita di fatto alcuna funzione di memorizzazione.

Questo comportamento è noto come trasparenza del latch.



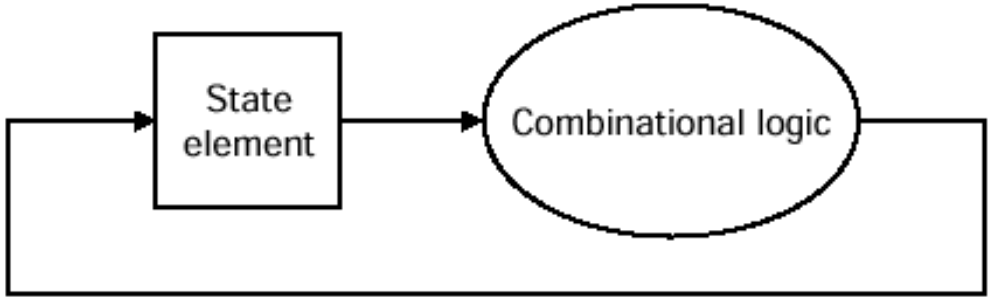
Elemento di memoria usato come input e output



Durante ogni periodo di clock

- il circuito combinatorio di sopra 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

Elemento di memoria usato come input e output



⇒ l'elemento di memoria deve essere usato sia come input che come output durante lo stesso ciclo di clock

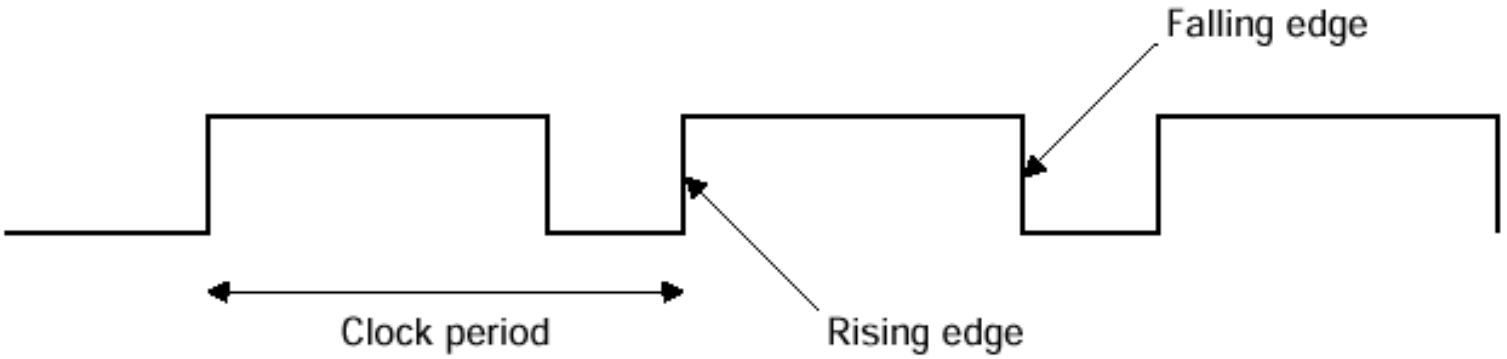
Il D-latch precedente funzionerebbe in questo caso ?

- Purtroppo no, perché se il clock rimane alto per molto tempo, allora il valore memorizzato nel latch potrebbe nel frattempo fluire fuori, entrare nel circuito, e un valore scorretto potrebbe finire per essere memorizzato nel latch

Metodologia di timing

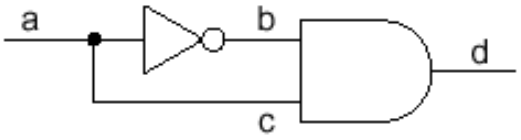
Si possono progettare componenti di memoria, in cui la memorizzazione può avvenire in vari istanti rispetto al segnale a gradino del clock

- level-triggered methodology
  - avviene sul livello alto (o basso) del clock
  - il D-latch precedente era level-triggered (rispetto al *livello alto*)
- edge-triggered methodology
  - 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 !!**
  - gli elementi di memoria edge-triggered si chiamano flip-flop

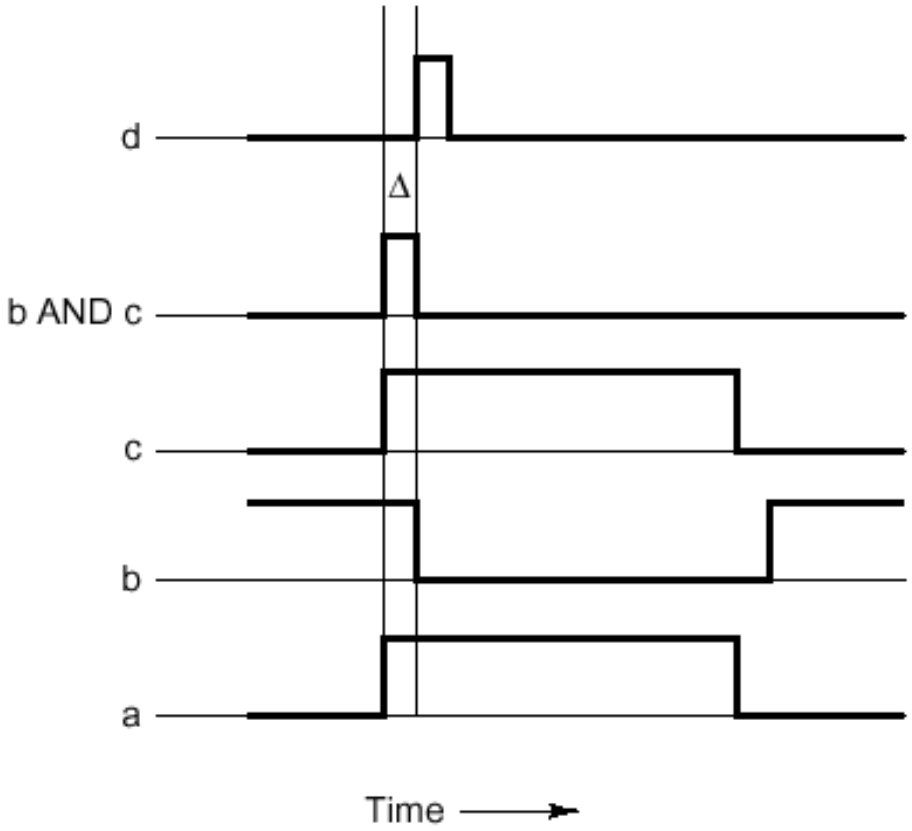


Generatore di implusi

Il generatore di impulsi permette appunto di generare impulsi (brevissimi) in corrispondenza del fronte di salita di un segnale a gradino (a)



Generatori di impulsi

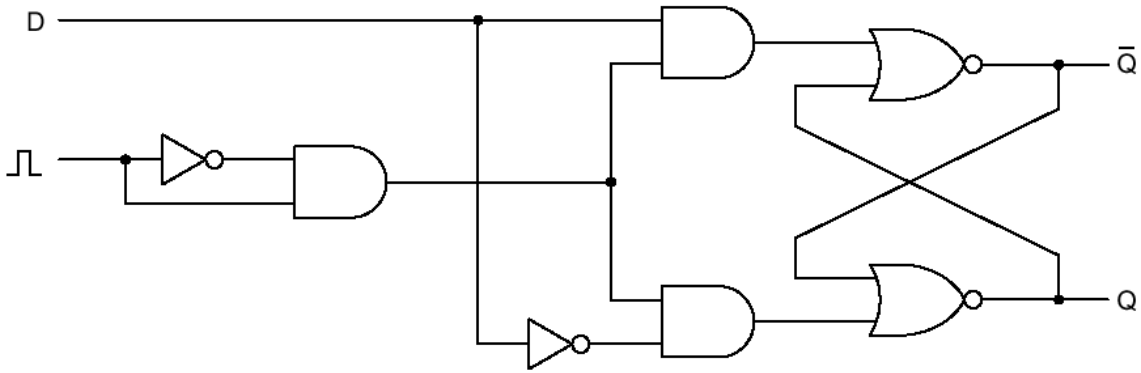


Porta NOT e AND con ritardo Δ

Solo durante l'intervallo Δ, quando *a* sale e diventa affermato

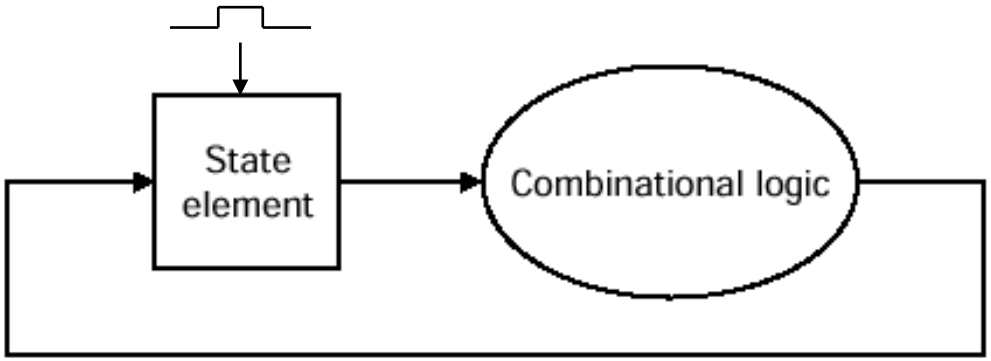
- i valori corrispondenti a *b* e *c* sono entrambi affermati
- l'impulso *b AND c* diventa anch'esso affermato

Flip-flop semplice (con generatore di impulsi)



Il flip - flop semplice memorizza "istantaneamente" il valore di D sul fronte di salita del clock: in corrispondenza dell'impulso

- metodologia edge-triggered di tipo rising triggered



Il segnale memorizzato comincia a fluire subito fuori dal flip-flop

Causa brevità dell'impulso, il segnale "non fa però in tempo" a entrare/uscire nel/dal circuito combinatorio a valle, e a modificare l'input del flip-flop

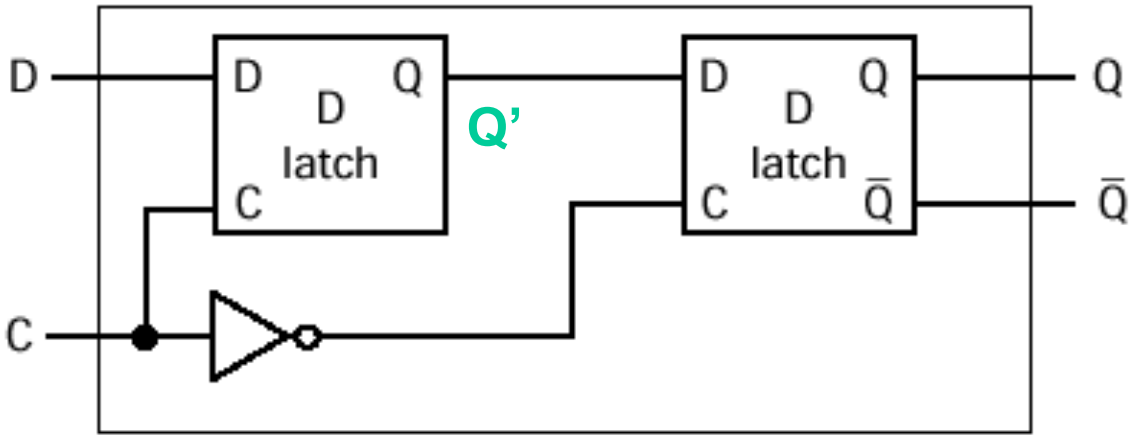
Flip-flop più complesso (D flip-flop)

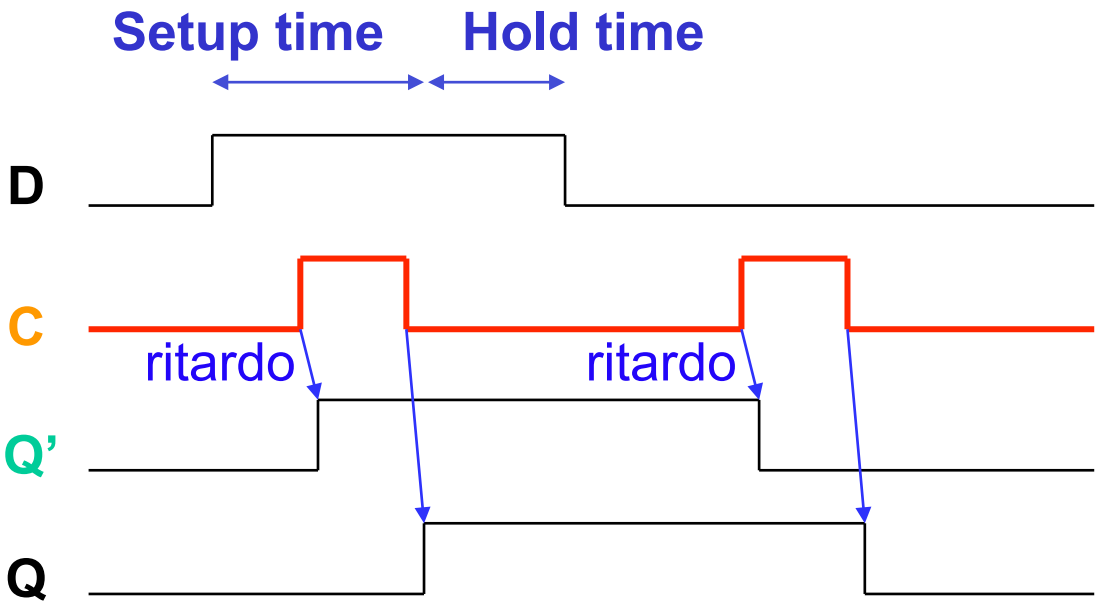
Il Flip-flop di tipo D usabile come input e output durante uno stesso ciclo di clock

- realizzato ponendo in serie 2 D-latch: il primo viene detto *master* e il secondo *slave*

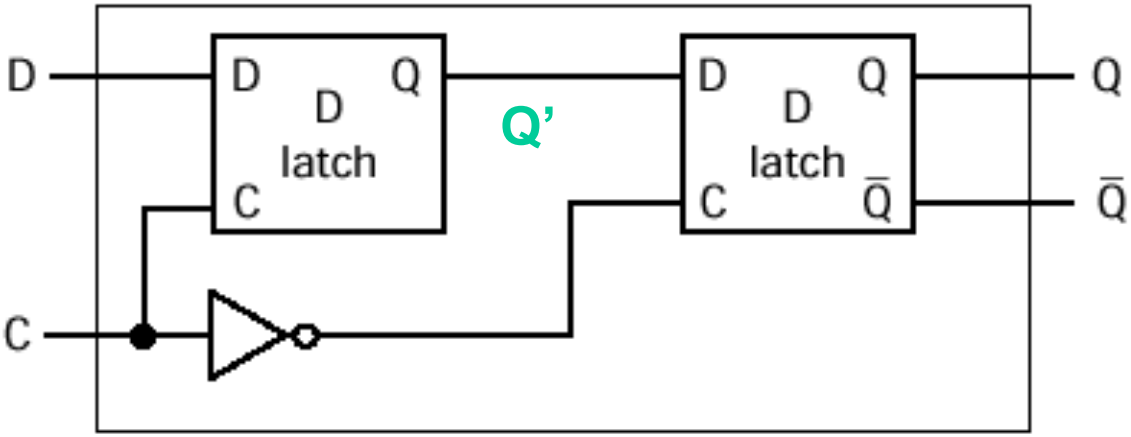
Metodologia edge-triggered

- rispetto al flip-flop precedente, questo è di tipo falling triggered
- per semplicità, possiamo pensare che la memorizzazione avvenga in maniera *istantanea* su fronte di discesa del clock C



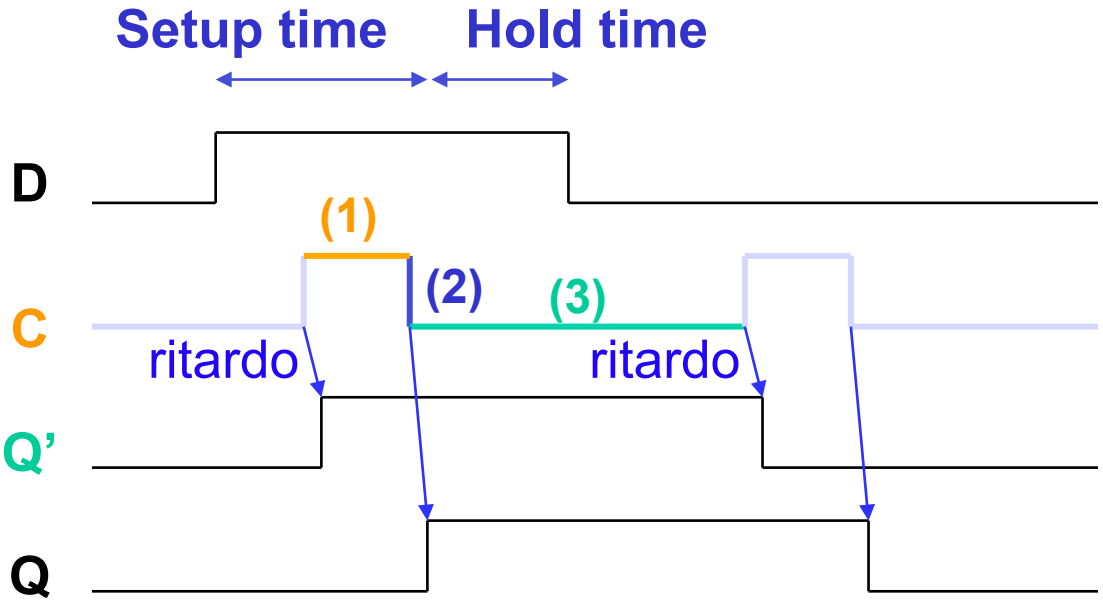


D Flip-flop



(1) Il primo latch è aperto e pronto per memorizzare D. Il valore memorizzato Q' fluisce fuori, ma il secondo latch è chiuso

- => nel circuito combinatorio a valle entra ancora il vecchio valore di Q



(2) Il segnale del clock scende, e in questo istante il secondo latch viene aperto per memorizzare il valore di Q'

(3) Il secondo latch è aperto, memorizza D (Q'), e fa fluire il nuovo valore Q nel circuito a valle. Il primo latch è invece chiuso, e non memorizza niente

Uso degli elementi di memoria

Ora sappiamo come costruire gli elementi di memoria.

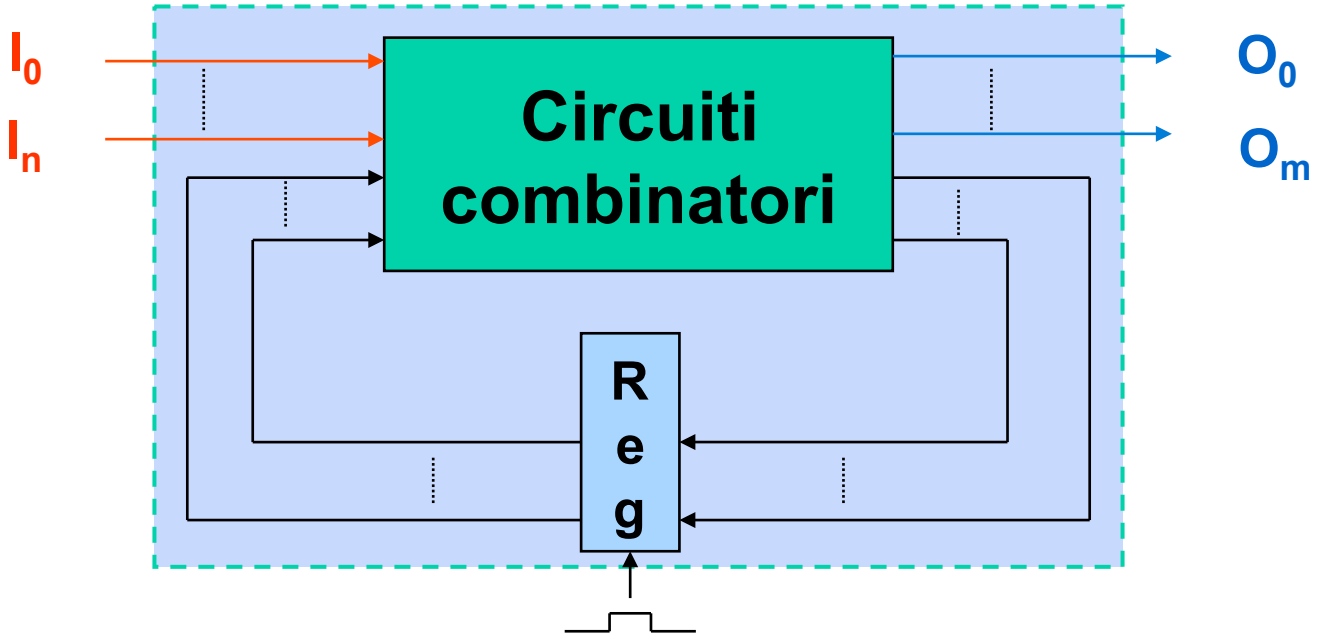
Vediamo come utilizzarli per realizzare:

- Circuiti sequenziali
- Register File
- Memorie principali

Circuito sequenziale sincrono

Blocco logico con linee in input e output composto da

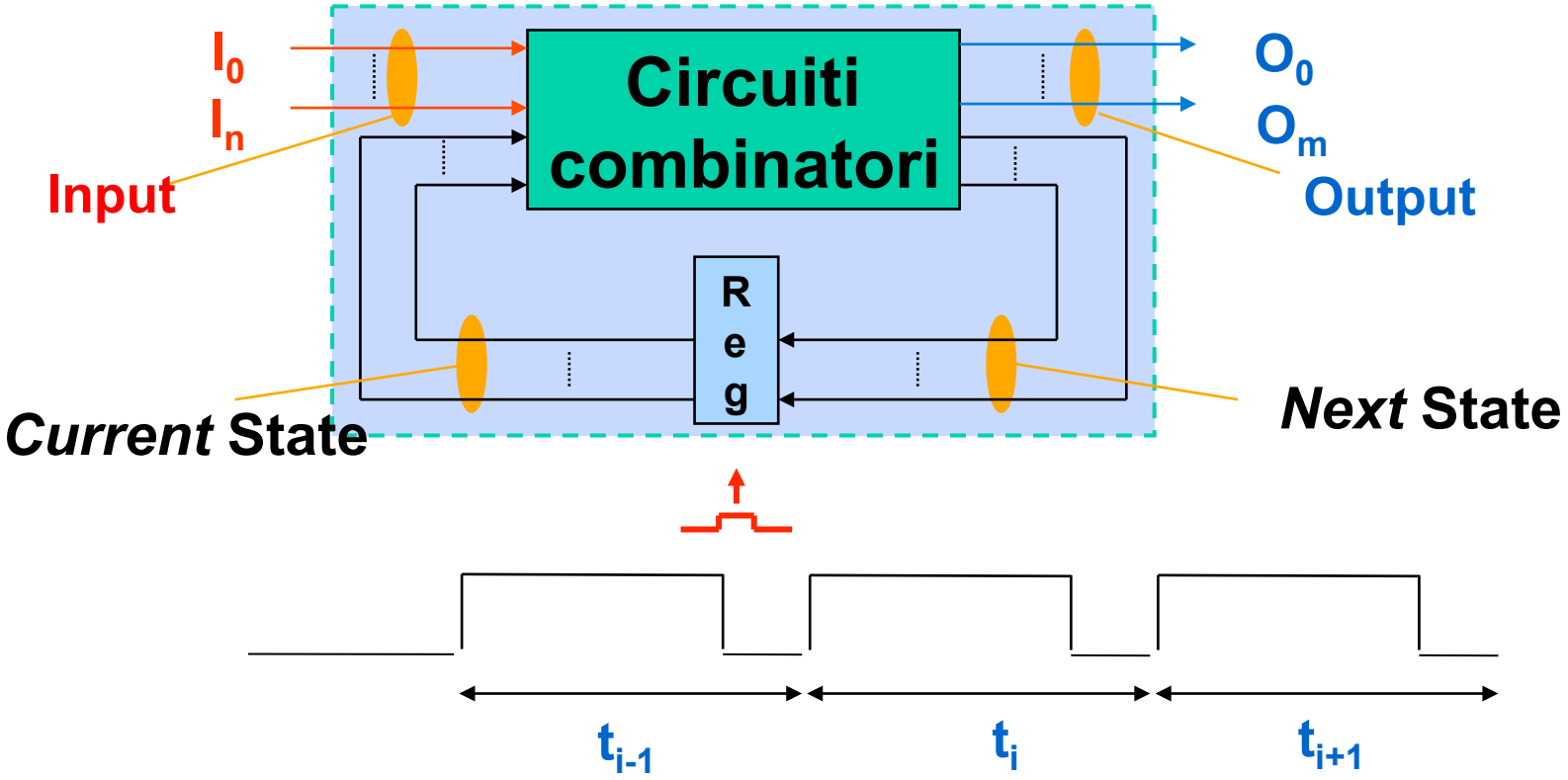
- circuiti combinatori
- elementi di memoria clockati (registri realizzati tramite flip-flop), che mantengono lo *stato* del circuito e che possono essere letti/scritti nello stesso periodo di clock



I circuiti combinatori sono le componenti che calcolano funzioni che generano

- i valori in output
- i valori da memorizzare negli elementi di memoria

Circuiti sequenziali



Registro di stato realizzato con flip-flop che impiegano una metodologia *falling edge triggered*

- durante il periodo t_i, il prossimo stato viene calcolato (ovvero lo stato al tempo t_{i+1}), ma viene memorizzato *solo* in corrispondenza del fronte di discesa del clock

Tipi di circuito (Mealy vs Moore)

Definiamo

- INPUT(t_i) e OUTPUT(t_i) i valori presenti, rispettivamente, sugli input e gli output dei circuiti combinatori al tempo t_i
- STATE(t_i) i valori presenti nei registri di stato al tempo t_i

Circuito sequenziale di Mealy

- OUTPUT(t_i) = δ(INPUT(t_i), STATE(t_i))
- NEXT_STATE(t_{i+1}) = λ(INPUT(t_i), STATE(t_i))

Circuito sequenziale di Moore

- OUTPUT(t_i) = δ(STATE(t_i))
- NEXT_STATE(t_{i+1}) = λ(INPUT(t_i), STATE(t_i))

Tipi di circuito (Mealy vs Moore)

Circuito sequenziale di Mealy

- $OUTPUT(t_i) = \delta(INPUT(t_i), STATE(t_i))$

Circuito sequenziale di Moore

- $OUTPUT(t_i) = \delta(STATE(t_i))$
- valore dell'output al tempo t_i dipende solo dal valore dei registri di stato

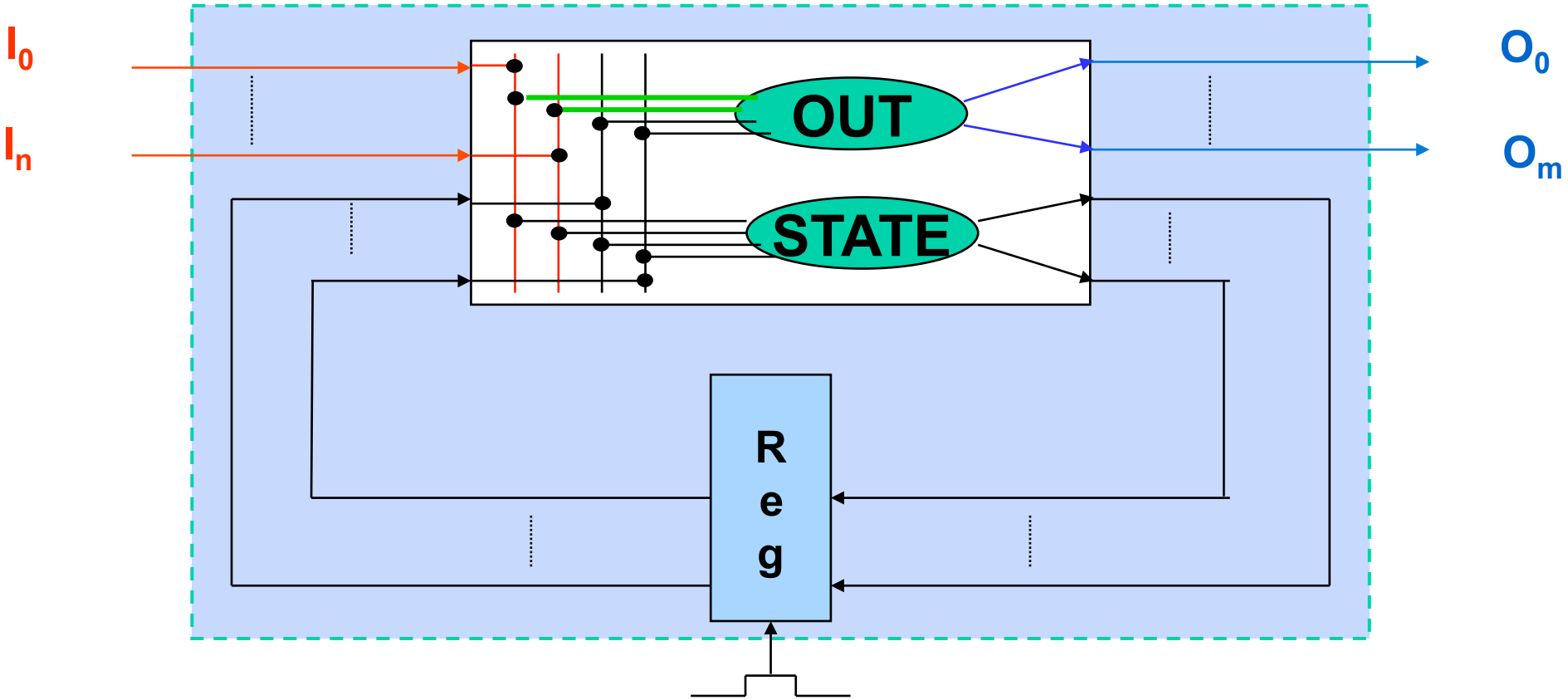
$STATE(t_i)$ modificato alla fine del ciclo di clock precedente (t_{i-1}) in base a:

- input a quel tempo presenti in ingresso al circuito: $INPUT(t_{i-1})$
- stato a quel tempo memorizzato nei registri: $STATE(t_{i-1})$

Mealy vs. Moore

Abbiamo quindi bisogno di 2 circuiti combinatori, che in linea di principio sono distinti

I collegamenti in verde verso il circuito OUT **non** sono necessari per realizzare circuiti di Moore



Sintesi di reti sequenziali

Per sintetizzare il circuito sequenziale in maniera *diretta* basta conoscere le tabelle di verità delle funzioni

- OUTPUT e NEXT_STATE

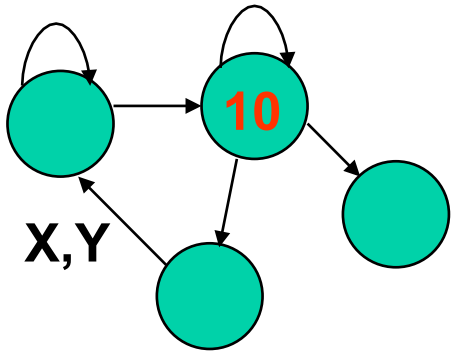
Dalle tabelle siamo poi in grado di determinare le equazioni booleane e i corrispondenti circuiti

Sintesi di reti sequenziali

Ma come si possono specificare e derivare le tabelle di verità?

Vedremo in seguito come è possibile specificare il comportamento di un circuito sequenziale tramite un particolare programma ad eventi, espresso graficamente tramite un *automa a stati finiti*

- grafo diretto
- nodi (stati) + archi (transizioni di stato)
- etichette sui nodi e sugli archi (input/output)



Register file

La Parte Operativa (Datapath) della CPU contiene, oltre all'ALU, alcuni registri che memorizzano, all'interno della CPU, gli operandi delle istruzioni aritmetico/logiche.

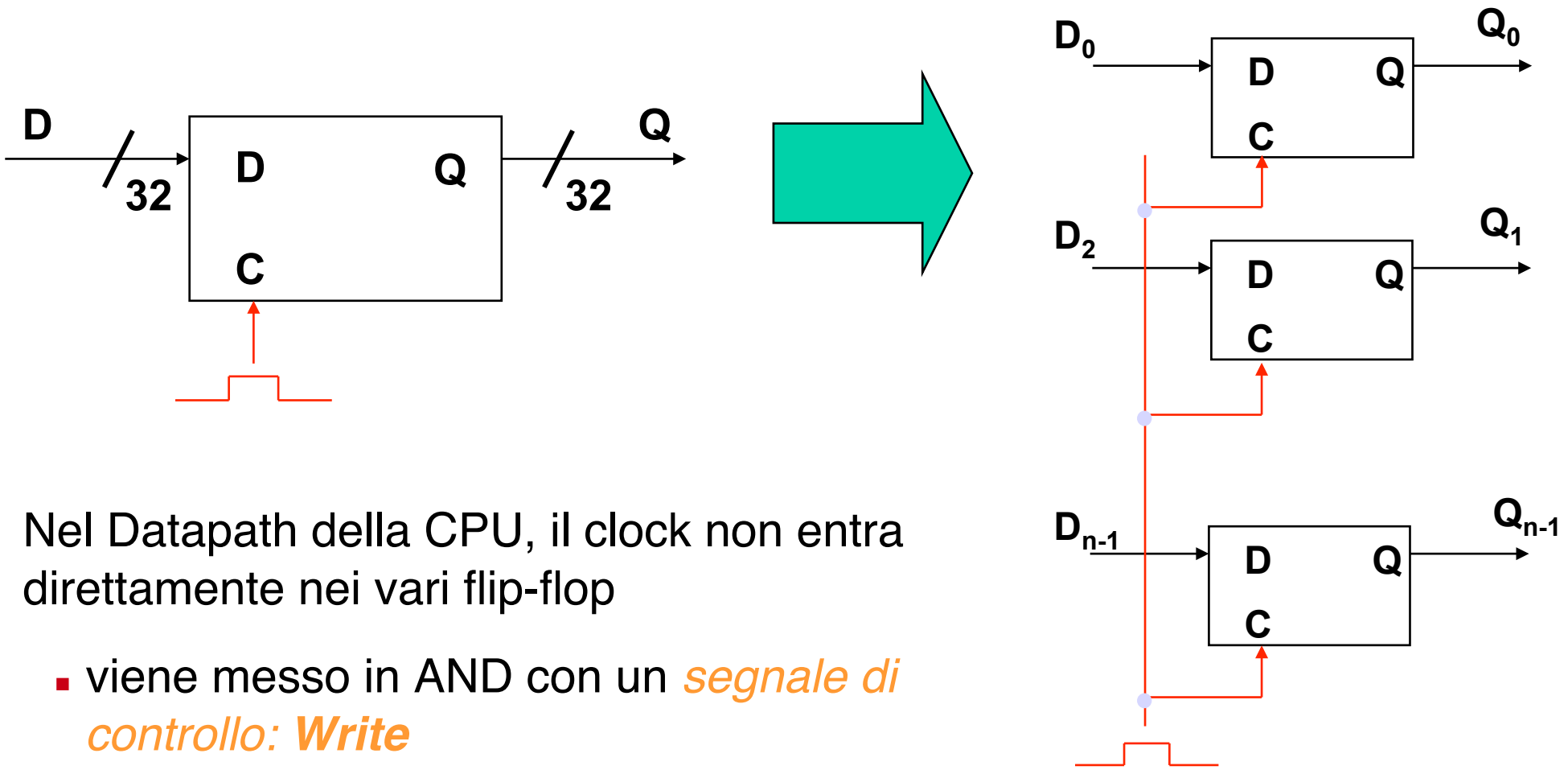
Ogni registro è costituito da *n* *flip-flop*, dove *n* è il numero bit che costituiscono una Word

- nel MIPS ogni registro è di 1 Word = 4 B = 32 b

Più registri sono organizzati in una componente nota come ***Register file***

- il Register File del MIPS contiene 32 registri (32x32=1024 flip-flop)
- deve permettere: lettura di 2 registri, e scrittura di 1 registro

Singolo registro

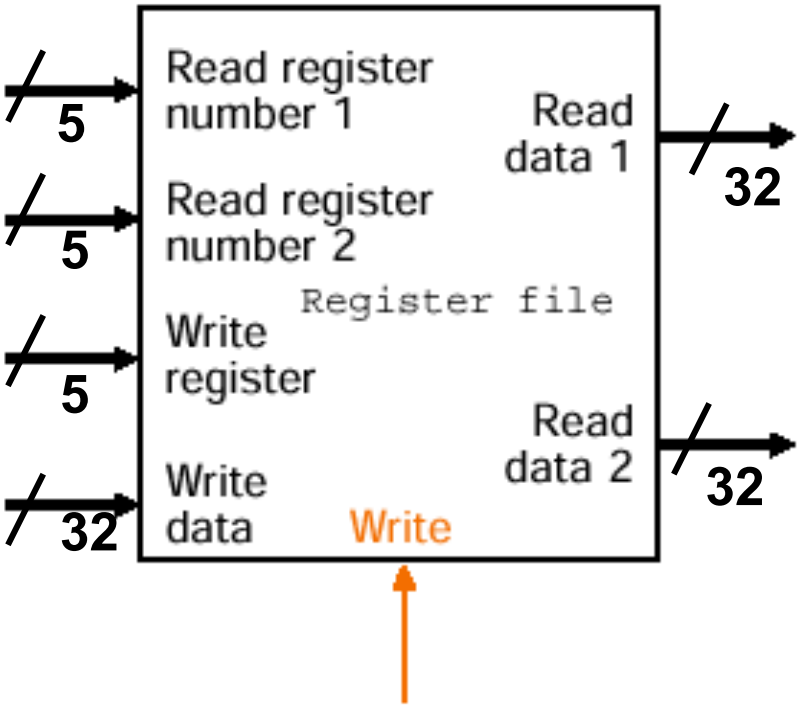


Nel Datapath della CPU, il clock non entra direttamente nei vari flip-flop

- viene messo in AND con un *segnale di controllo:* ***Write***

il segnale determina se, in corrispondenza del fronte di discesa del clock, il valore di D debba (o meno) essere memorizzato nel registro

Register file del Datapath



Read Reg1 # (5 bit)

- numero del 1° registro da leggere

Read Reg2 # (5 bit)

- numero del 2° registro da leggere

Read data 1 (32 bit)

- valore del 1° registro, letto sulla base di Read Reg1 #

Read data 2 (32 bit)

- valore del 2° registro, letto sulla base di Read Reg2 #

Write Reg # (5 bit)

- numero del registro da scrivere

Write data (32 bit)

- valore del registro da scrivere in base a Write Reg #

Write

- segnale di controllo messo in AND con il *clock*
- solo se Write=1, il valore di Write data viene scritto in uno dei registri

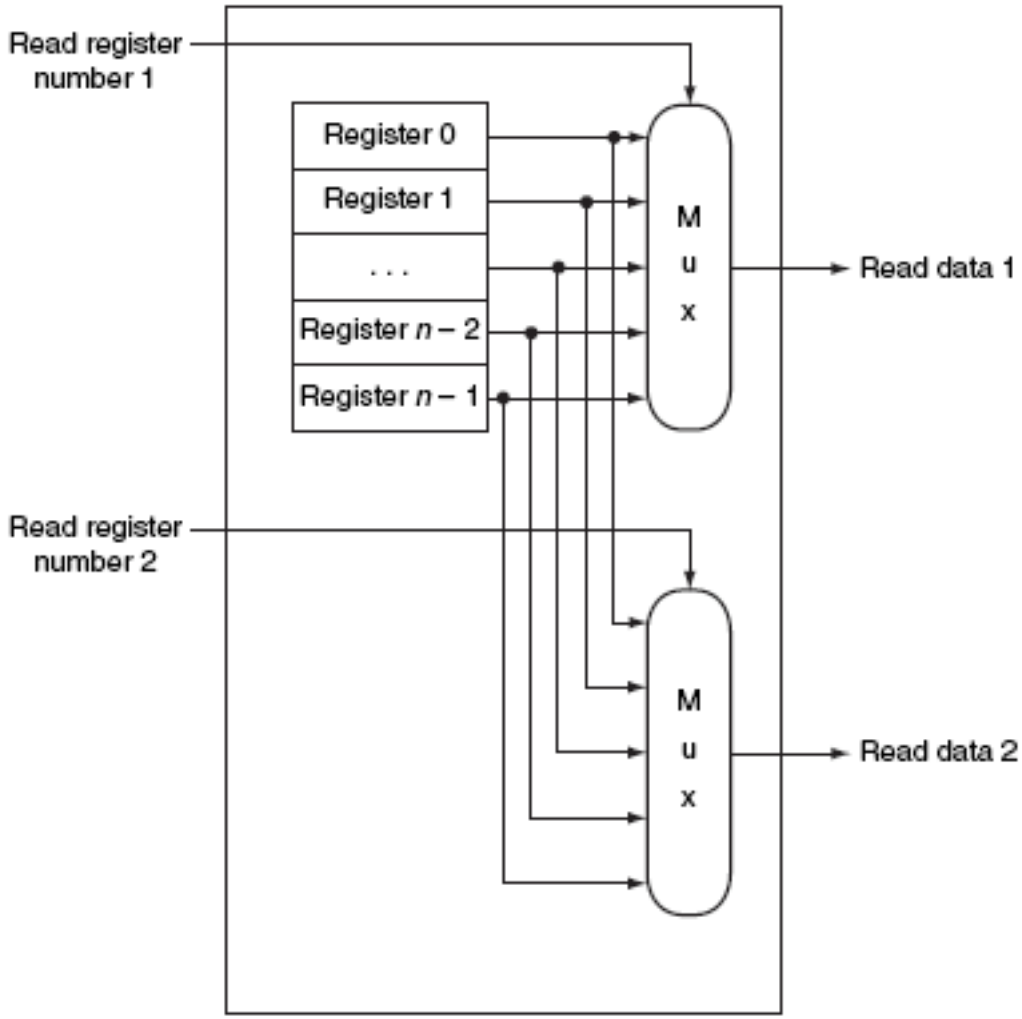
Lettura dal Register file

2 MUX 32:1

- larghi 32 bit

I controlli dei 2 MUX sono

- Read Reg1 #
- Read Reg2 #



Nota che il Register file fornisce sempre in output una coppia di registri

- *non significativi,* se i controlli Read Reg1 # e Read Reg2 # non lo sono
- in tal caso, i circuiti che potenzialmente potrebbero usarli, devono ignorali

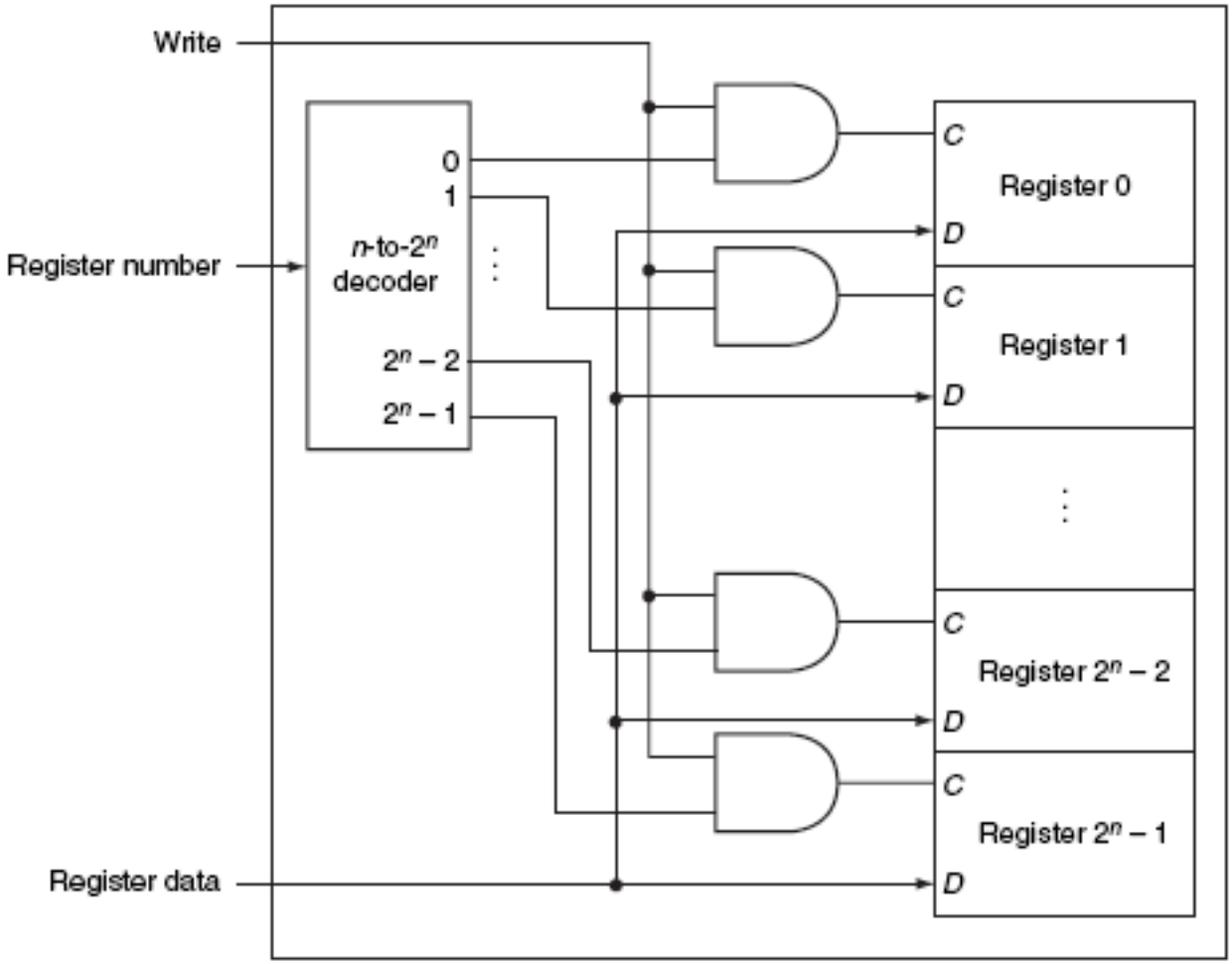
Scrittura nel Register file

Decoder che decodifica il segnale di controllo Write
Reg # (5 bit)

- 32 bit in output

Il segnale di Write (a sua volta in *AND* con il clock) abilita solo uno dei 32 registri

- il segnale di Write è infatti in *AND* con l'output del Decoder



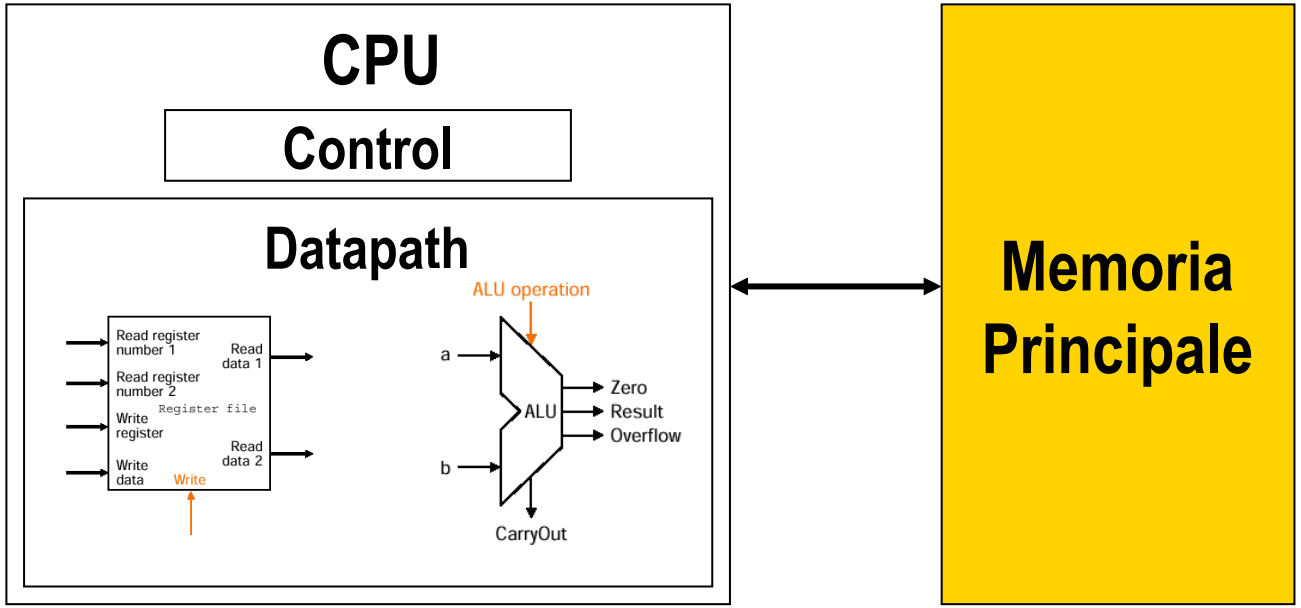
Se il segnale di Write è non affermato, i possibili valori spuri in input non verranno memorizzati nel Register file

Lettura/Scrittura nel Register file

Domanda: cosa succede se uno stesso registro del Register file viene acceduto in lettura e scrittura durante uno stesso ciclo di clock?

Risposta: poiché la scrittura sul registro avviene sul fronte di discesa del clock, il valore ritornato dalla lettura sarà quello memorizzato in un ciclo di clock precedente.

Memoria principale



La dimensione del Register File è piccola

- registri usati per memorizzare singole variabili di tipo semplice
- purtroppo per memorizzare dati strutturati e codice di programma, sono tipicamente necessari diversi KB o MB

Memoria principale (RAM - Random Access Memory)

- meno veloce della memoria dei registri, ma molto più capiente
- è detta *Random Access Memory* perché i tempi di accesso sono indipendenti dal valore dell'indirizzo della cella di memoria acceduta

SRAM e DRAM

La SRAM (Static RAM) è più veloce

- per la sua realizzazione vengono usati dei latch
- è usata per realizzare memorie veloci, come le *memorie cache*
- tempi di accesso (2008) intorno a 0,5 – 2,5 ns
  - SRAM a basso consumo (5-10 volte più lente)

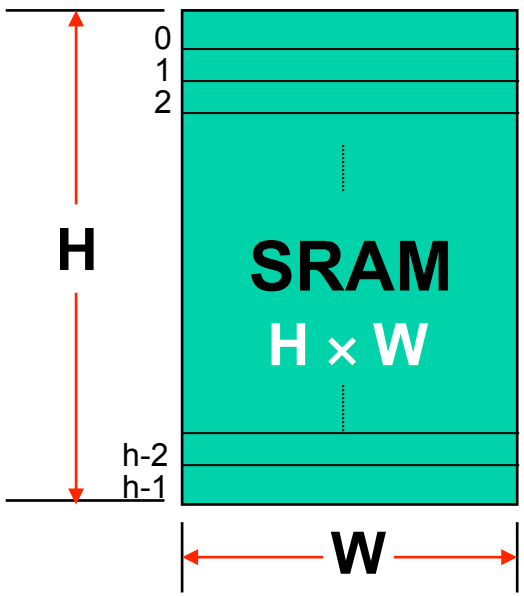
La DRAM (Dynamic RAM) è più capiente ma più lenta

- tempi di accesso (2008) intorno a 50-70 ns
- non è realizzata tramite latch
- ogni bit è memorizzato tramite un condensatore
- è necessario “rinfrescare” il contenuto delle DRAM a intervalli di tempo prefissati
- è usata per realizzare memorie capienti come quella principale

SRAM

SRAM realizzata come matrice di latch HxW con

- larghezza o ampiezza W (numero di latch per ogni cella)
- altezza H (numero di celle 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: $\log_2 H$

Esempio 1

- 256K × 1 (256K celle da 1 bit = 256Kb)
- 18 linee di indirizzo (2^{18} = 256K), 1 linea in output

Esempio 2

- 32K × 8 (32K celle da 8 bit = 256Kb)
- 15 linee di indirizzo (2^{15} = 32K), 8 linee in output

SRAM

Esempio di chip 32K × 8

Din e Dout

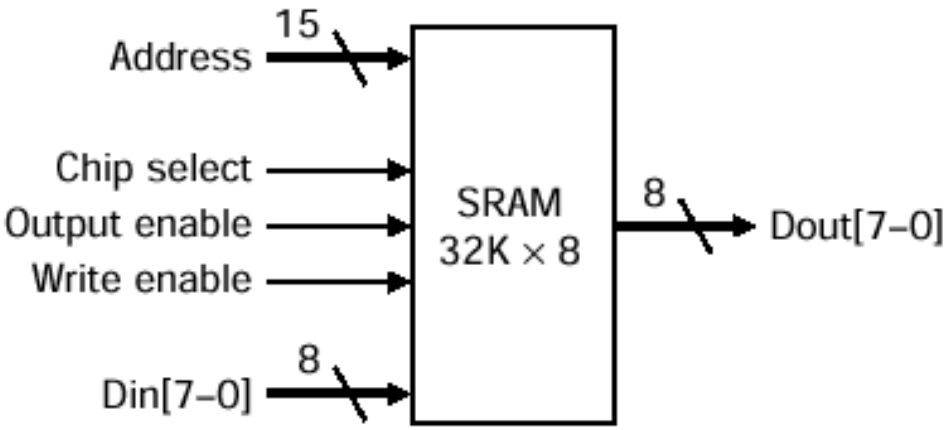
Address

Chip select

- da affermare per poter leggere o scrivere

Output enable

- da affermare per poter abilitare l'uscita del chip su un bus condiviso
- serve a poter collegare molti chip di memoria ad un singolo bus



Write enable

- impulso che, quando attivato, registra nella linea di latch individuati da Address il valore presentato in Din

SRAM

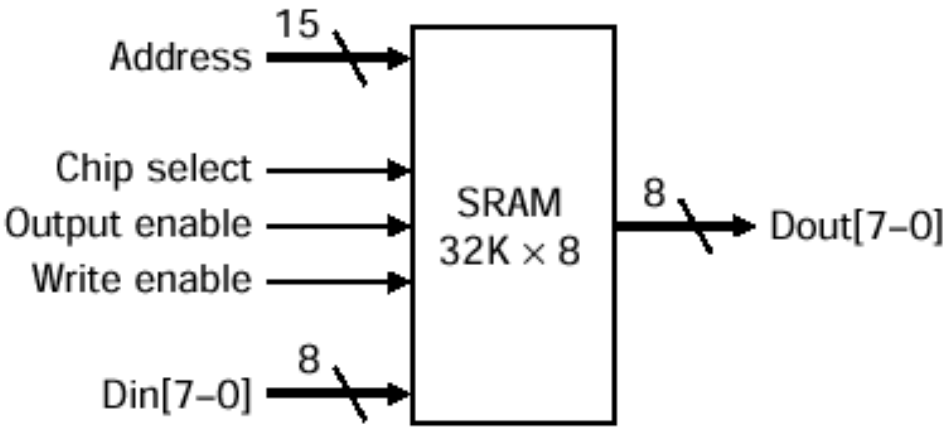
Esempio di chip 32K × 8

Per scrivere

- Chip select affermato
- Indirizzo in Address
- Write enable affermato
- Dato in input in Din

Per leggere

- Chip select affermato
- Indirizzo in Address
- Output enable affermato
- Dato in output in Dout



Realizzazione SRAM

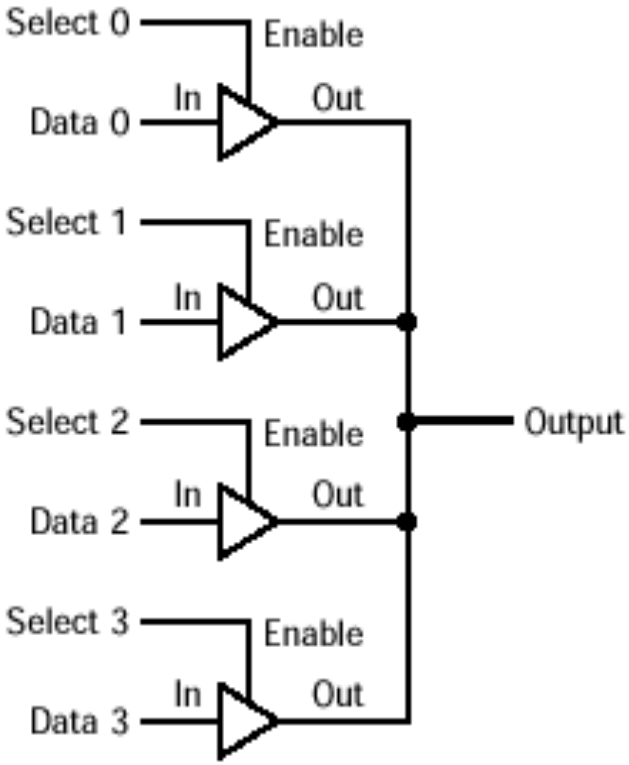
Tecniche realizzative diverse rispetto a quelle del register file

- il register file impiega decoder (per selezionare il registro da scrivere) e multiplexer (per selezionale il registro da leggere)
- con un numero elevato di celle di memoria avremmo bisogno di enormi decoder o multiplexer
  - avremmo bisogno di porte AND con fan-in troppo elevato
  - necessari livelli multipli di porte AND, con conseguente introduzione di ritardi negli accessi alla memoria

Realizzazione SRAM

Per evitare il multiplexer in uscita

- possiamo usare una linea di bit condivisa su cui i vari elementi di memoria sono tutti collegati (*or*-ed)
- il collegamento alla linea condivisa avviene tramite *buffer a tre stati,* che aprono o chiudono i collegamenti (se il controllo è affermato o meno). In particolare:
- Il *buffer* ha due ingressi (dato e segnale di Output Enable) e una uscita:
  - l'uscita è uguale al dato (zero o uno) se Output Enable è affermato
  - l'uscita viene impedita (high-impedance state) se Output Enable non è affermato

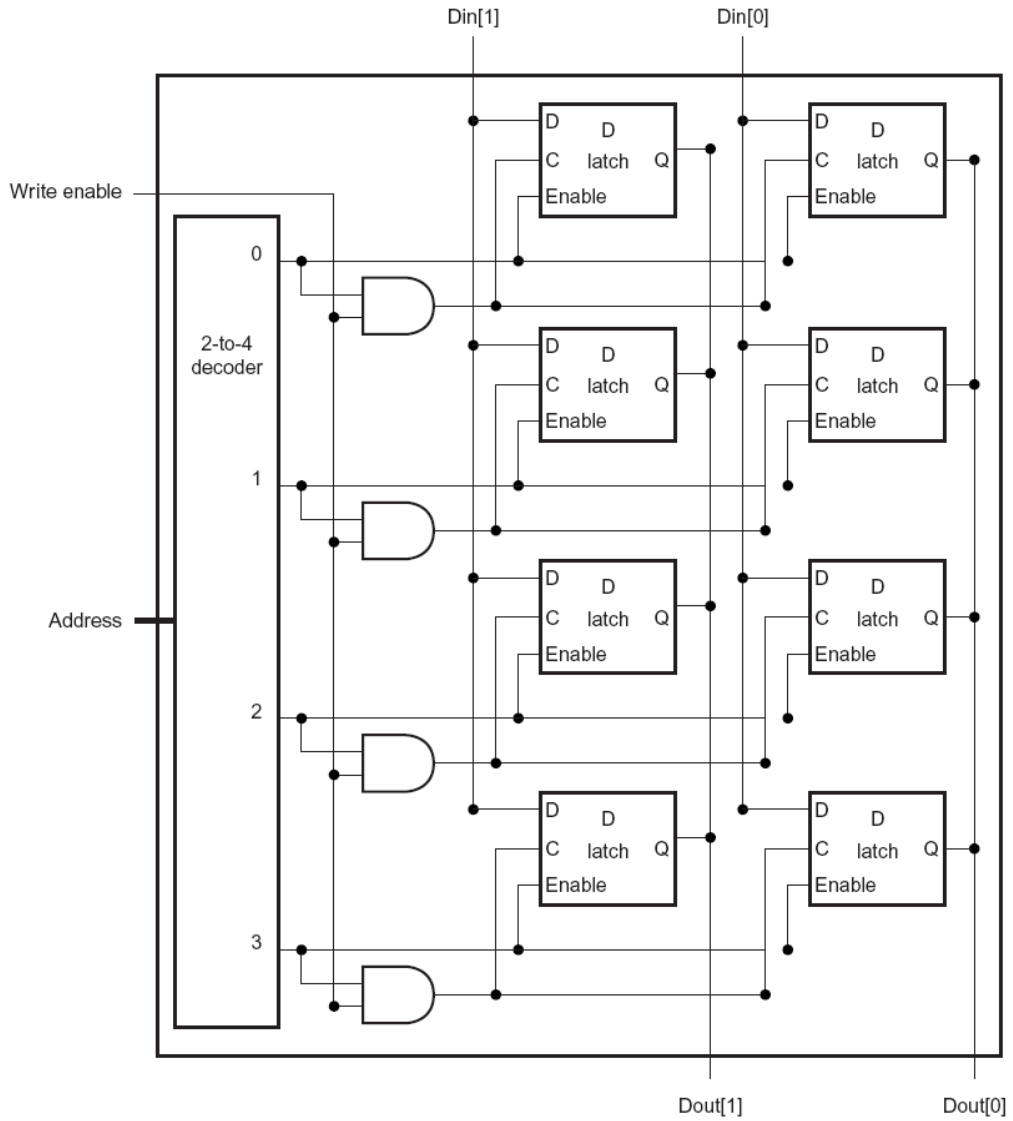


Multiplexer 4:1 realizzato con 4 buffer a 3 stati

Esempio di SRAM 4x2

I latch di una certa colonna sono collegati alla stessa linea in output (Dout[0] e Dout[1])

- nell'esempio ogni elemento di memoria (D-latch) ha un segnale di *Enable* che abilita il *three-state buffer* interno



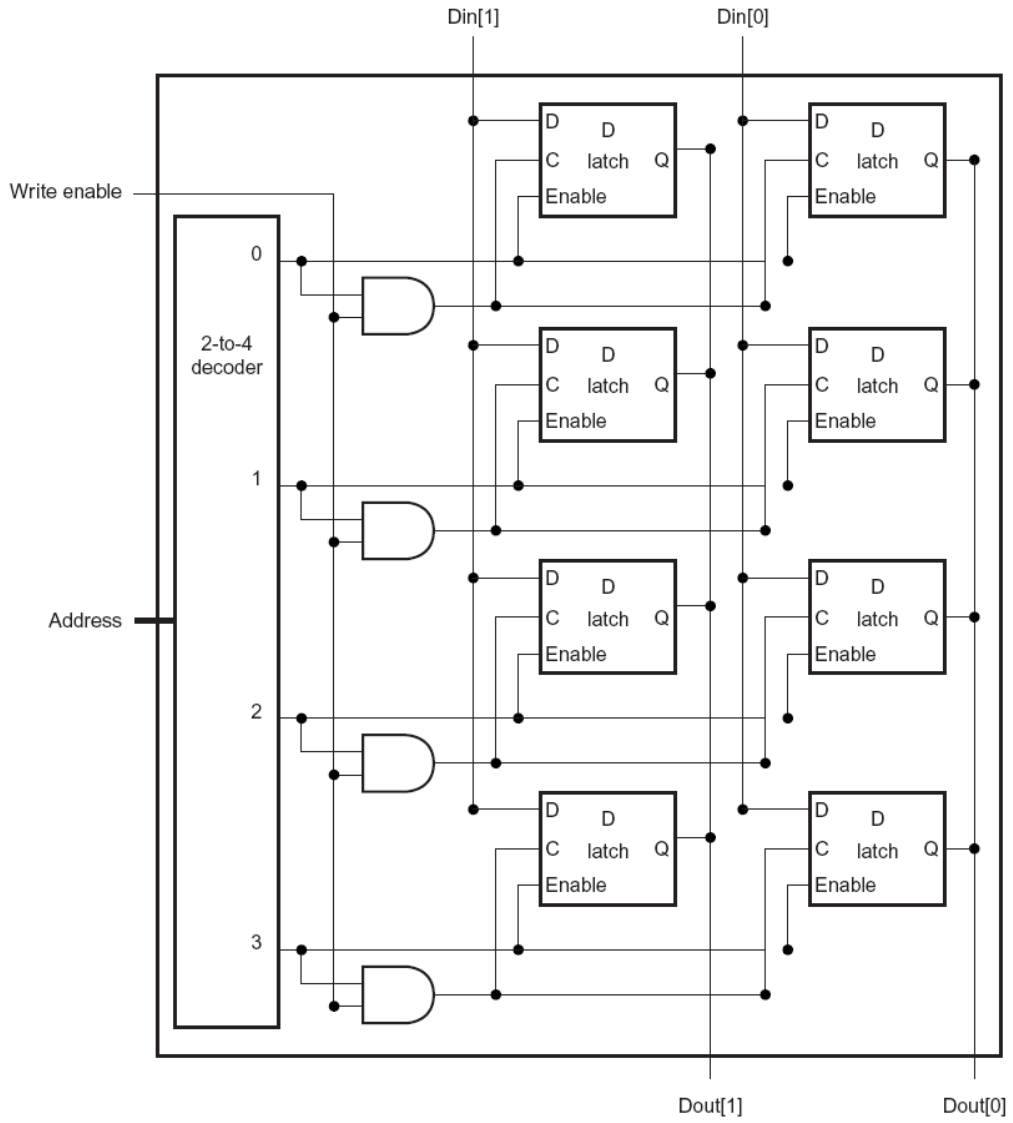
Il Decoder serve ad abilitare in lettura / scrittura una certa linea della memoria

- Entrambi i segnali Write enable e Enable vengono abilitati su una sola linea di memoria (2 bit)

Esempio di SRAM 4x2

Chip Enable e Output Enable sono stati omessi per semplicità ma possono essere aggiunti con qualche porta AND

- Solo se Chip Enable è abilitato i segnali Write Enable e Output Enable sono significativi (servono porte AND aggiuntive)
- Solo se Output Enable è affermato, la coppia Dout [0-1] dovrebbe essere abilitata ad uscire (*Output Enable* in AND con *Enable*, a sua volta determinato dal decoder)



SRAM a due livelli

Nel caso precedente abbiamo evitato l'uso dei Multiplexer, ma rimane il problema del Decoder grande

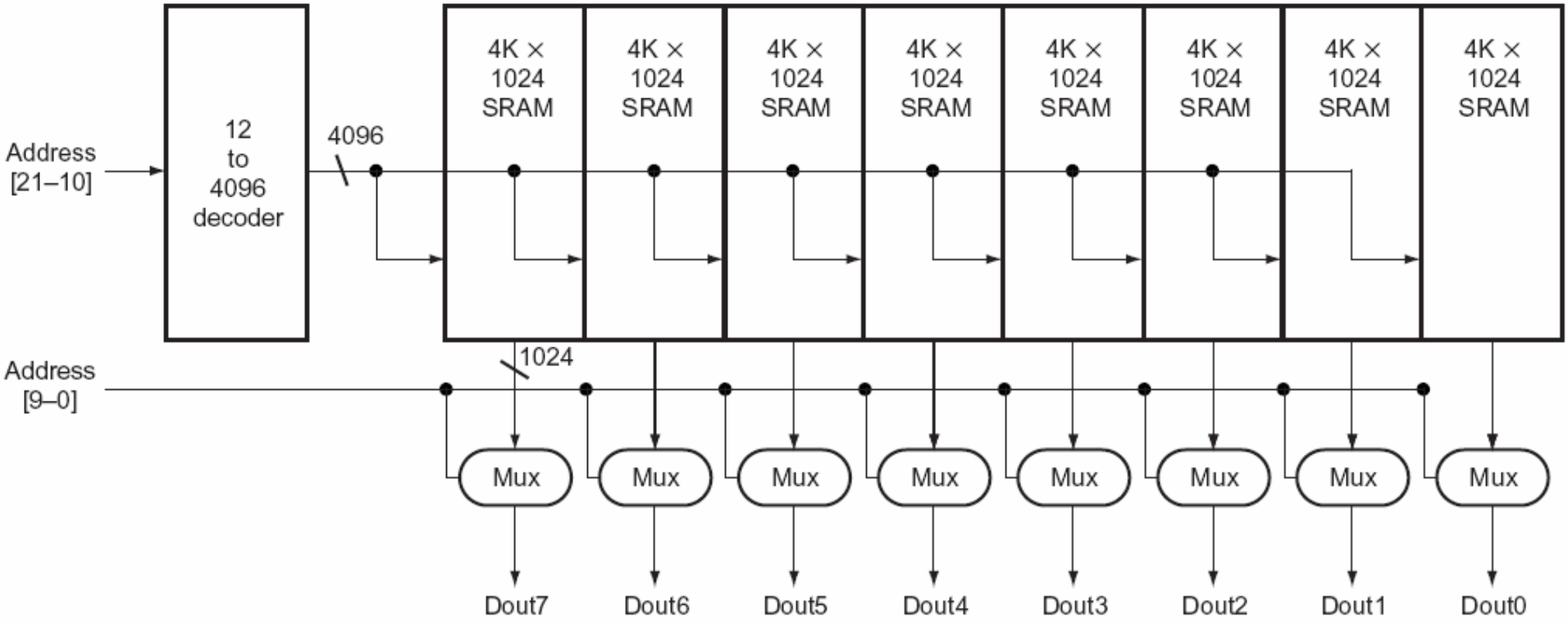
Per ovviare, decodifica degli indirizzi a due livelli

Usiamo un decoder più piccolo e una batteria di piccoli Multiplexer

SRAM a due livelli

Esempio di SRAM: 4M × 8 => servono 22 bit di indirizzo

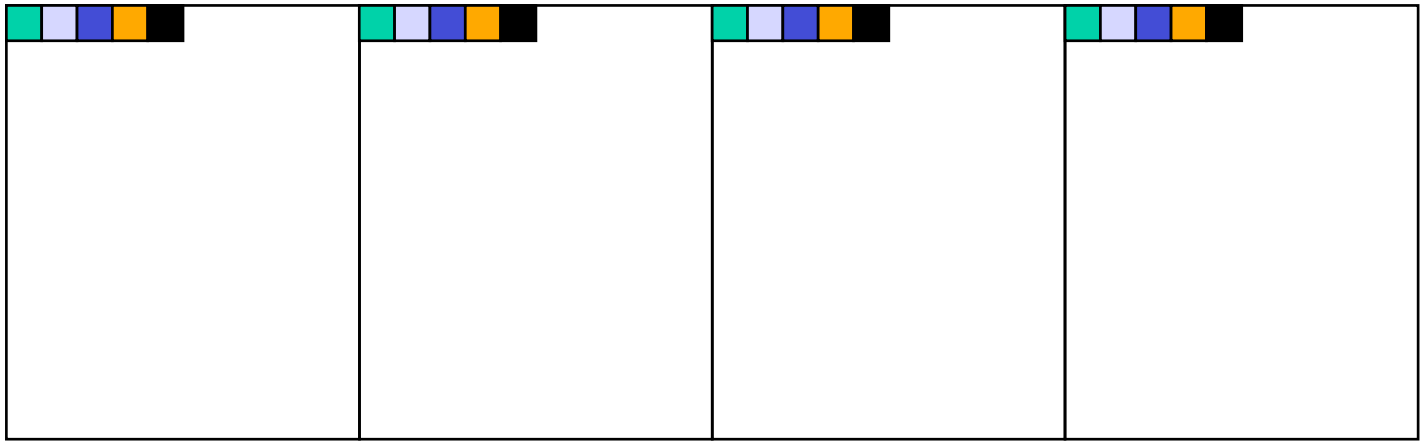
- suddiviso in 8 blocchi da 4Mb (blocchi rettang. 4K × 1024 bit)
- parte alta dell'indirizzo [21-10] seleziona la medesima riga di ogni blocco da 4K × 1024 bit attraverso un Decoder
- parte bassa dell'indirizzo [9-0] seleziona singoli bit dei 1024 bit in output dai vari blocchi, attraverso una batteria di 8 Multiplexer



Esempio: SRAM a due livelli con ampiezza 4

Bit di una singola cella sparsi nei 4 moduli della SRAM

- i bit appartenenti alla stessa cella di memoria rappresentati da quadratini di colore identico



Cella di 4 bit di indirizzo 000…0000

Cella di 4 bit di indirizzo 000…0001

Cella di 4 bit di indirizzo 000…0010

Cella di 4 bit di indirizzo 000…0011

Cella di 4 bit di indirizzo 000…0100

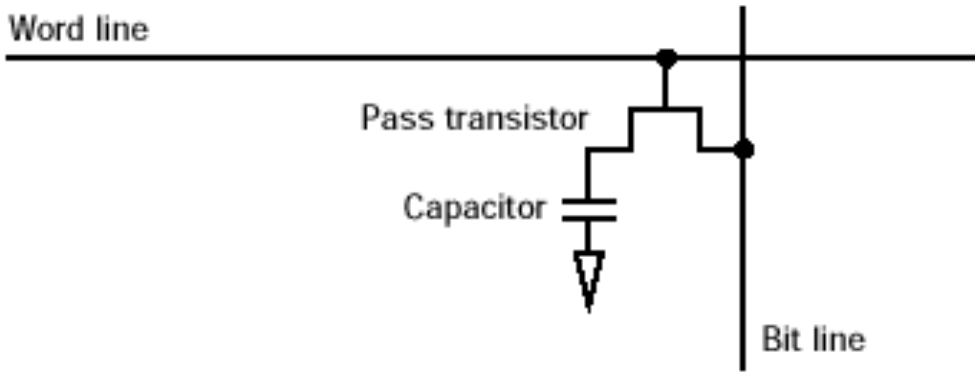
DRAM

Gli elementi di memoria di tipo DRAM sono meno costosi e più capienti rispetto al tipo SRAM, ma sono *più lenti*

- DRAM sono da 5 a 10 volte meno veloci delle SRAM

La DRAM è meno costosa, perché è realizzata con un solo transistor per bit, e un condensatore

- il condensatore possiede la carica (0/1)
- il transistor viene chiuso, trasferendo il potenziale elettrico del condensatore sulla Bit line (output), grazie al segnale affermato della Word line
- la specifica Word line è attivata sulla base dell'indirizzo di memoria richiesto

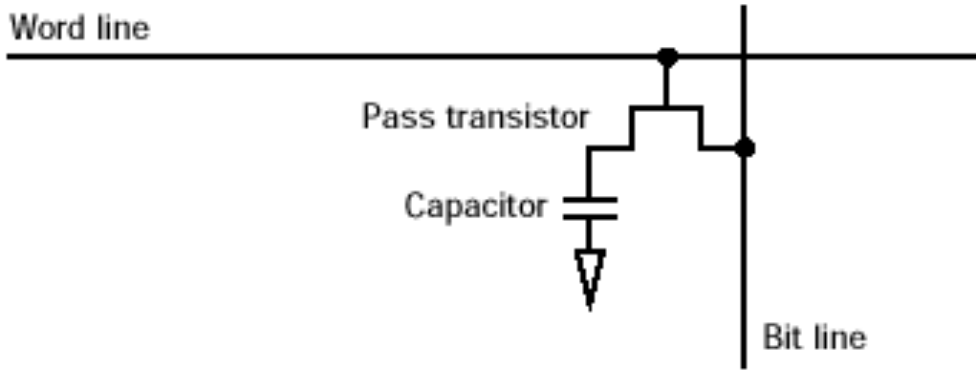


DRAM

I condensatori mantengono i valori memorizzati solo per alcuni *ms*

Necessario il *refresh dinamico* delle DRAM, effettuato leggendo, e subito riscrivendo i valori appena letti

Il refresh avviene ad intervalli fissi, occupa circa il 2% del tempo totale, e avviene per righe

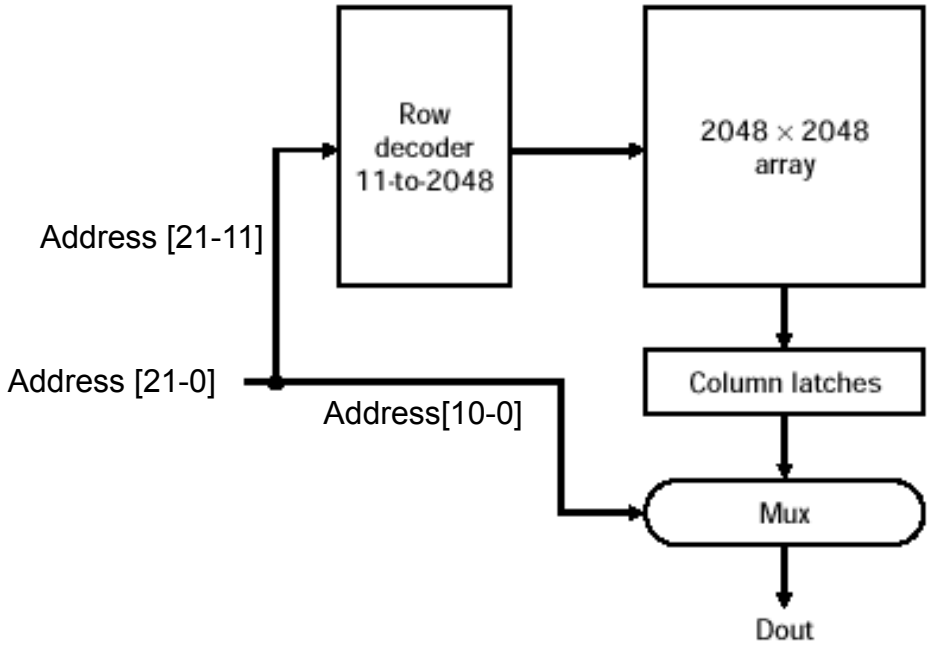


DRAM

DRAM (come SRAM) realizzate con decodifica a 2 livelli

Esempio di chip DRAM di 2^{22} = 4 Mb, con organizzazione 4M×1

- indirizzo totale di 22 bit
- indirizzo spezzato in due pezzi da 11 bit ciascuno



- parte alta e bassa dell'indirizzo considerate come indirizzo di RIGA o di COLONNA, rispettivamente
  - indirizzo di RIGA ha effetto sul *Decoder*
  - indirizzo di COLONNA ha effetto sul *Mux*
- possibile accesso *ottimizzato* a bit di memoria consecutivi
  - contenuti nei *column latches* dopo un accesso di RIGA

Caratteristiche DRAM

Size: x 4 ogni 3 anni fino 1998

Size: x 2 ogni 2 anni dopo 1998

Costo non tiene conto dell'inflazione

Anno	Chip Size	$ per MB	Total access time	Soltanto Column Access
1980	64 Kbit	$1500	250 ns	150 ns
1983	256 Kbit	$500	185 ns	100 ns
1985	1 Mbit	$200	135 ns	40 ns
1989	4 Mbit	$50	110 ns	40 ns
1992	16 Mbit	$15	90 ns	30 ns
1996	64 Mbit	$10	60 ns	12 ns
1998	128 Mbit	$4	60 ns	10 ns
2000	256 Mbit	$1	55 ns	7 ns
2002	512 Mbit	$0.25	50 ns	5 ns
2004	1 Gbit	$0.10	45 ns	3 ns

SSRAM e SDRAM

Abbiamo visto che per diminuire la complessità dei decoder è opportuno suddividere gli indirizzi in 2 blocchi

- parte alta per accedere una riga
- parte bassa per accedere una specifica colonna

Nota che celle consecutive hanno indirizzi che solitamente differiscono solo per la parte bassa dell'indirizzo

- quindi sono contenuti all'interno di una stessa riga selezionata con la parte alta dell'indirizzo

Le Synchronous SRAM e DRAM (SSRAM e SDRAM) permettono di aumentare la banda di trasferimento della memoria sfruttando questa proprietà

SSRAM e SDRAM

- Memoria sincrona (col segnale di clock)
- E' possibile specificare che vogliamo trasferire dalla memoria un *burst* di dati (ovvero una sequenza di celle consecutive)
- Ogni *burst* specificato da un *indirizzo di partenza*, e da una *lunghezza*
- Le celle del burst sono contenute all'interno di una stessa Riga, selezionata una volta per tutte tramite decoder
- La memoria fornisce una delle celle del *burst* a ogni ciclo di clock

$\Rightarrow$ Migliora banda di trasferimento (numero di trasf. al sec)

- non è necessario ripresentare l'indirizzo per ottenere ogni cella del *burst*
- costo del decoder pagato una sola volta, all'inizio