02 - LOGICA - LIVELLO LOGICO-DIGITALE

RETI COMBINATORIE

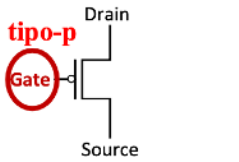
slide 1

Cenni alla tecnologia CMOS e ai transitori - *approfondimento*

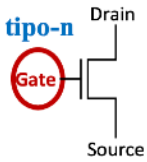
CMOS = **C**omplementary, **M**etal (conduttori), **O**xide (isolanti), **S**emiconductors

I **transitori** sono usati per progettare le porte logiche elementari.

Un transistor funziona come fosse un interruttore **on/off.**
Ci sono **2 tipi** di transitori: un transistor di **tipo-p** funziona in modo complementare rispetto ad uno di **tipo-n.**



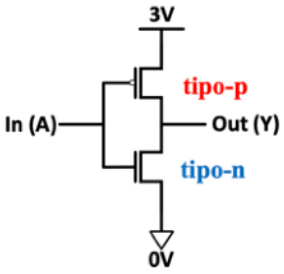
tipo-p: Il circuito drain-source si chiude quando il gate è alimentato a **0V**



tipo-n: il circuito drain-source si chiude quando il gate è alimentato a **3V**

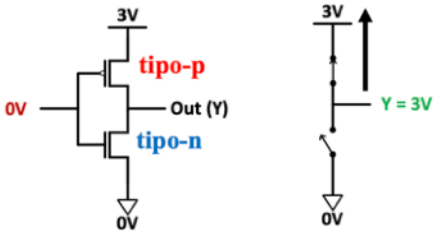
Tecnologia CMOS per progettare circuiti logici

Tecnologia complementary MOS **(CMOS)** = **pMOS + nMOS**
Il più semplice circuito logico presente in un microprocessore ha un ingresso **A** ed un'uscita **Y** ed è composto da due transitori: uno di **tipo-p** e uno di **tipo-n** collegati in serie:



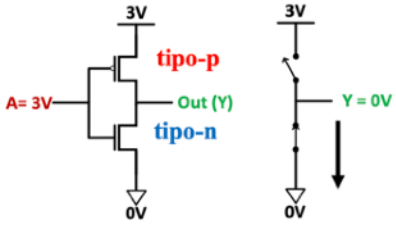
Funzionalità del circuito CMOS

Cosa succede quando il segnale d'ingrasso A è collegato a 0V?



il transistor di **tipo-p** porta a 3V il segnale di uscita, cioè ha funzione **pull up**

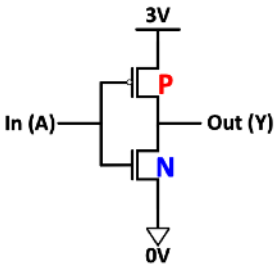
cosa succede quando il segnale d'ingresso A è collegato a 3V?



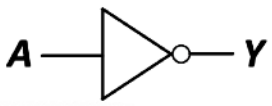
il transistor di tipo-n porta a 0V il segnale di uscita, cioè ha funzione **pull down**

Porta logica NOT in tecnologia CMOS

- Porta logica **NOT** (invertitore) **Y = !A**
- Circuito logico: possibile interpretazione dei segnati
  - **0V** come valore logico (binario) **0**
  - **3V** come valore logico (binario) **1**



A	P	N	Y
0	ON	OFF	1
1	OFF	ON	0



ALGEBRA BOOLEANA E PORTE LOGICHE ELEMENTARI

Algebra di Boole (matematico inglese George Boole, 1815-1864)

- serve a descrivere le operazioni logiche
- Componenti dell'algebra di Boole:
  - L'insieme B di elemento (operandi booleani) sui quali vengono eseguite le operazioni
  - Operatori booleani che agiscono sugli elementi dell'insieme B
  - Elementi neutri di B necessari a definire le operazioni

Algebra di commutazione

- l'insieme B contiene solo 2 elementi a cui vengono assegnati per convenzione i valori 0 (falso) e 1 (vero): B = {0, 1}.
- Gli elementi neutri corrispondono ai due soli valori dell'insieme B.
- Il valore di un'operazione logica elementare (e più in generale di una funzione logica) può essere espresso da una Tabella di Verità che elenca tutte le combinazioni dei valori di ingresso e il corrispondente valore del risultato dell'operazione logica.
- Operatori logici elementari: not, and, or

A	not A
0	1
1	0

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

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

Porte logiche (circuiti combinatori elementari)

- I circuiti digitali che elaborano segnali binari (0,1) sono formati da componenti elementari, chiamate **porte logiche**, che forniscono in uscita il risultato di operazioni logiche sui valori delle **n** variabili in ingresso.
- Le **porte logiche elementari**: porta **NOT**, porta **AND**, porta **OR**.
- La funzione calcolata da un circuito con **n** ingressi è descritta dalla corrispondente tabella di verità per ciascuna delle **2^n** combinazioni degli ingressi.

Operatori e porte logiche

Buffer

A	Z
0	0
1	1

AND

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

OR

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

XOR

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

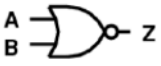
Inverter

A	Z
0	1
1	0

NAND

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

NOR



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

XNOR

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

NOT	X = !A	inverte il valore di 0 o 1
AND	X = A • B	vale 1 sse entrambi gli ingressi valgono 1
OR	X = A + B	vale 1 sse almeno un ingresso vale 1
NAND	X = ! (A • B)	vale 0 sse entrambi gli ingressi valgono 0
NOR	X = ! (A + B)	vale 0 sse almeno un ingresso vale 1
XOR	X = A ⊕ B	vale 1 sse una sola variabile vale 1 (riconosce la disuguaglianza)
XNOR	X = ! (A ⊕ B)	vale 1 sse entrambe e variabili valgono 0 o 1 (riconosce l'uguaglianza)

PROPRIETÀ DELL'ALGEBRA DI COMMUTAZIONE

- Deriva dall'**algebra di Boole** e consente di descrivere matematicamente i circuiti digitali (o logici)
- Definisce le **espressioni logiche** che descrivono il **comportamento** di un circuiti nella forma **U = f (I)**
- A partire dalle eq. logiche è possibile derivare la **realizzazione circuitale** (rete logica)
- I componenti dell'algebra di Boole sono: le **variabili di commutazione**, gli operatori fondamentali e le proprietà degli operatori logici trami le quali è possibile trasformare le espressioni logiche

Proprietà	AND (•)	OR (+)
Identità	1 • A = A	0 + A = A
Elemento nullo	0 • A = 0	1 + A = 1
Idempotenza	A • A = A	A + A = A
Inverso	A • !A = 0	A + !A = 1
Commutativa	A • B = B • A	A + B = B + A
Associativa	(A • B) • C = A • (B • C)	(A + B) + C = A + (B + C)
Distributiva	A • (B + C) = (A • B) + (A • C)	A + (B • C) = (A + B) • (A + C)
Assorbimento	A • (A + B) = A	A + A B = A
De Morgan	!(A • B) = !A + !B	!(A + B) = !A • !B

Vale il **principio di dualità** → il duale di una funzione booleana si ottiene sostituendo:
- **and** a **or** oppure **or** a **and**
- **0** a **1** oppure **1** a **0**

Osservazioni

- **precedenza degli operatori**: inversione, prodotto, somma
- le prop. commutativa, distributiva, identità e invertono sono **assiomi**
- le altre prop. sono **teoremi**: dimostrabili per induzione matematica completa attribuendo alle variabili di agresto tutti i 2^n valori disponibili
- **concetto di equivalenza**: due espressioni booleane sono **equivalenti** se hanno la stessa tabella di verità

Prop. Assorbimento - dimostrazione

$$A + A \cdot B = A$$

- per induzione matematica completa, attribuiamo alle variabili di ingresso A e B tutti i 2^n valori disponibili
- a partire dall'espressione iniziale, usando le proprietà dell'algebra di Boole, semplifichiamo la funzione in un'espressione equivalente:

A	B	A • B	A + A • B
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

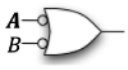
Espressione	Proprietà
A + A • B	Identità
A • 1 + A • B	Distributiva
A • (1 + B)	Elemento nullo
A • 1	Identità
A	

Teorema di De Morgan

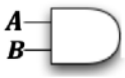
!(A • B) = !A + !B

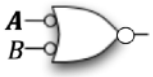
A	B	! (A • B)	!A	!B	!A + !B
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

- per induzione matematica completa, attribuiamo ad A e B tutti i 2^n valori possibili
- La negazione dell'**and** di due variabili equivale all'**or** delle variabili negate



- negando entrambi i membri delle due precedenti espressioni si ricava: **A • B = !(!A + !B)**
- l'**and** di 2 variabili equivale al **nor** delle variabili negate



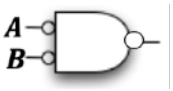


!(A + B) = !A • !B

A	B	! (A • B)	!A	!B	!A + !B
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

- per induzione matematica completa, attribuiamo ad A e B tutti i 2^n valori possibili
- la negazione dell'**or** di due variabili equivale all'**and** delle variabili negate

- negando entrambi i membri delle due precedenti espressioni si ricava: **A + B = !(!A • !B)**
- l'**or** di due variabili equivale al **nand** delle variabili negate.



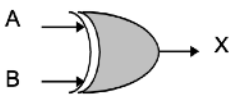
ESEMPI

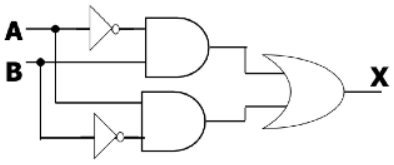
OR esclusivo - X = A ⊕ B

Dimostriamo che A ⊕ B = !A•B + A•!B

A	B	A • B	!A • B	A • !B	!A • B + A • !B
0	0	0	0	0	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

Circuiti logici corrispondenti



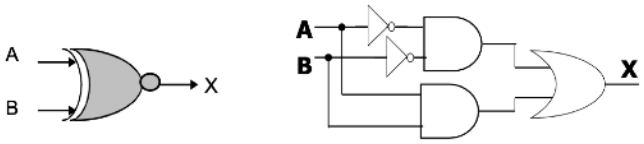


XNOR - X = !(A ⊕ B)

Dimostriamo che !A ⊕ B = !A•!B + A•B

A	B	! (A + B)	!A • !B	A • B	!A • !B + A • B
0	0	1	1	0	1
0	1	0	0	0	0
1	0	0	0	0	0
1	1	1	0	1	1

Circuiti logici corrispondenti



Esempio 1 di semplificazione

f(A,B,C)=!A!BC + !ABC + !AB!C

Espressione	Proprietà
!A!BC + !ABC + !AB!C	Idempotenza x + x = x
!A!BC + **!ABC** + !ABC + !AB!C	Distributiva xy+xz = x(y+z)
!AC (!B + B) + !AB (C + !C)	inverso x + !x = 1
!AC1 + !AB1	Identità x1=x
!AC + !AB	Distributiva xy+xz = x(y+z)
!A (C + B)	

Esempio 2 di semplificazione

f(A,B,C) = !AB!C + AB!C + ABC

Espressione	Proprietà
!AB!C + AB!C + ABC	Idempotenza x + x = x
!AB!C + **AB!C** + **AB!C** + ABC	Distributiva xy+xz = x(y+z)
B!C (!A + A) + AB (!C + C)	inverso x + !x = 1
B!C1 + AB1	Identità x1=x
B!C + AB	

INSIEME DI OPERATORI FUNZIONALMENTE COMPLETI

I 3 operatori logici elementari **(and, or, not)** costituiscono un **insieme funzionalmente completo** cioè consentono di rappresentare una qualunque funzione di commutazione.
Problema: Esistono insiemi di operatori più ridotti, che siano ancora un insieme funzionalmente completo?

Consideriamo l'insieme composto dai due operatori (not, and)

- Per capire se è funzionalmente completo dobbiamo verificare se, mediante applicazioni anche ripetute di not e and, è possibile realizzare l'operatore **or**
- Dal teorema di De Morgan: **!(!A • !B) = A + B**
- Poichè è possibile realizzare l'operatore or componendo gli operatori not e and Þ l'insieme (not, and) è funzionalmente completo.

Consideriamo l'insieme composto dai due operatori (not, or)

- Per capire se è funzionalmente completo dobbiamo verificare se, mediante applicazioni anche ripetute di not e or, è possibile realizzare l'operatore **and**
- Dal teorema di De Morgan: **!(!A + !B) = A • B**
- Poiché è possibile realizzare l'operatore and componendo gli operatori not e or Þ l'insieme (not, or) è funzionalmente completo.

Consideriamo l'insieme composto dai due operatori (and, or)

Si dimostra che non è funzionalmente completo, perché la loro composizione non permette di ricavare in alcun modo l'operatore **not**.

Operatori universali: NOR

- **Problema**: L'insieme composto dal solo operatore **(nor)** è un insieme funzionalmente completo?
- Per vedere se l'operatore **nor** costituisce un insieme funzionalmente completo dobbiamo verificare se mediante il **nor** è possibile realizzare gli operatori not, and, e or:
  - 0 nor A = !A oppure (A nor A) = !A
  - (A nor B) nor 0 = A or B

    - (A nor 0) nor (B nor 0) = A and B
- L'operatore **NOR è funzionalmente completo** (**operatore universale**).

Operatori universali: NAND

- **Problema**: L'insieme composto dal solo operatore (**nand**) è un insieme funzionalmente completo?
- Per vedere se l'operatore **nand** costituisce un insieme funzionalmente completo dobbiamo verificare se mediante il **nand** è possibile realizzare gli operatori not, and, e or.
    - 1 nand A = !A oppure (A nand A) = !A
    - (A nand B) nand 1 = A and B
    - (A nand 1) nand (B nand 1) = A or B
- L'operatore **NAND è funzionalmente completo** (**operatore universale**).

GENERALIZZAZIONE: PORTE LOGICHE A PIÙ INGRESSI

Alcuni tipi di porte a 2 ingressi posso essere generalizzate a 3, 4 o più ingressi
Le porte più usate sono **AND** e **OR**, tipicamente a **2, 4 o 8 ingressi**

Nota: non tutti i tipi di porte a più di 2 ingressi si possono realizzare come generalizzazioni di porte a 2 ingressi (funziona sempre con AND, ORD, XOR, XNOR)

Attenzione:
- XOR generalizzato a n variabili di ingresso: l'uscita vale 1 se e solo se il numero di 1 è dispari
- XNOR generalizzato a n variabili di ingresso: l'uscita vale 1 se e solo se il numero di 1 è pari

Porta AND a 3 ingressi + cascata

Simbolo funzionale

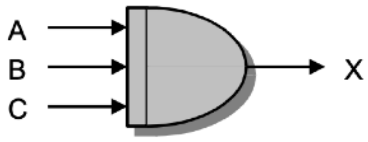
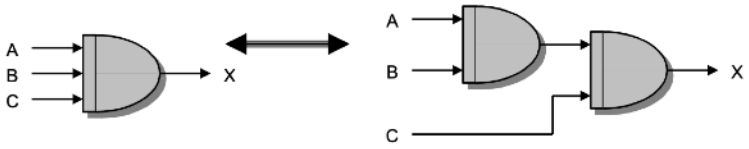


Tabella di verità → l'uscita vale 1 sse tutti e 3 gli ingressi valgono 1

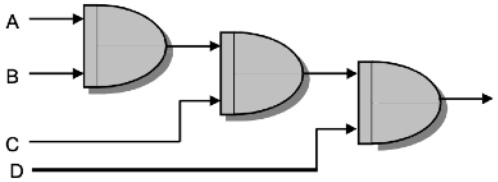
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

La porta AND a 3 ingressi si realizza come circuito a cascata di porte AND a 2 ingressi

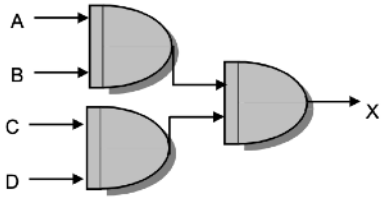


Esempio - AND a 4 ingressi

Circuito a cascata

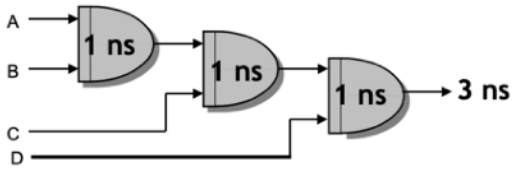


Circuito ad albero

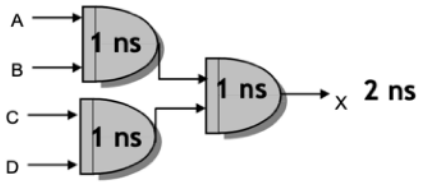


Cascata vs Albero - costo e tempo di propagazione

Circuito a cascata:
Costo = 3 porte AND
Ritardo = 3 ns



Circuito ad albero:
Costo = 3 porte AND
Ritardo = 2 ns



ANALISI DI RETI COMBINATORIE

Ad ogni **funzione combinatoria**, data come espressione booleana, si può sempre associare un **circuito digitale**, formato da **porte logiche**, che viene chiamato **rete combinatoria**:

- gli ingressi n >= 1 della rete combinatoria sono le variabili della funzione
- la rete è formata dall'interconnessione di porte logiche elementari AND, OR , NOT, ecc.
- l'uscita della rete combinatoria ha il valore assunto dalla funzione combinatoria per ciascun valore degli ingressi

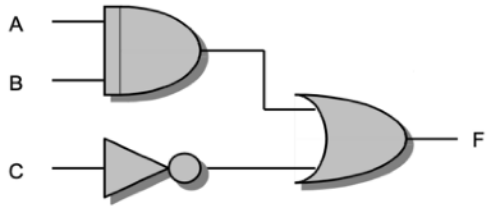
Una rete combinatoria è un **circuito logico digitale** in cui le decisioni logiche dipendono solo da una **combinazione degli ingressi** (il risultato non dipende dalla storia passata del circuito).
Il valore delle uscite è aggiornato immediatamente dopo il cambiamento degli ingressi.

I circuiti/reti combinatori possono essere descritti in modo equivalente come:

- **tabella di verità a più ingressi**
- **espressioni booleane**
- **circuiti logici**

Esempio

f (A, B, C) = A B + !C

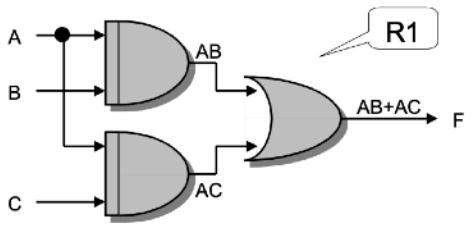


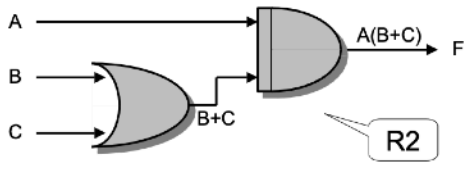
A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

La tabella di verità di una rete combinatoria può anche essere ricavata per **simulazione del funzionamento circuitale della rete combinatoria** stessa → si applicano **tutte le possibili combinazioni** dei valori agli ingressi, e li si propaga lungo la rete fino all'uscita.

Es: per una rete a 3 ingressi bisogna svolgere 2^3 = 8 simulazioni

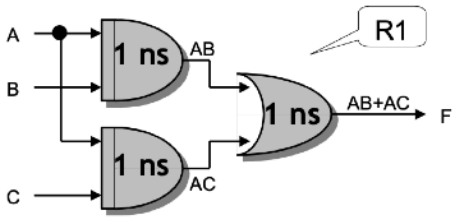
Due reti equivalenti



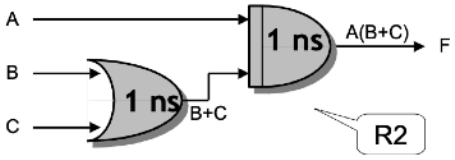


F1 = AB + AC
Trasformazione:
F1 = AB + AC =
= A(B + C) = F2
(prop. distributiva)

F2 = A(B + C)



F1 = AB + AC
Costo = 3 porte elem.
Ritardo totale = 2 ns
Frequenza = 1 / 2 ns = 500 MHz



F2 = A(B + C)
Costo = 2 porte elem.
Ritardo totale = 2 ns (percorso critico)
Frequenza = 1 / 2 ns = 500 MHz

SINTESI DI RETI COMBINATORIE: FORME CANONICHE

La **sintesi di una rete combinatoria** espressa come tabella delle verità, consiste nel ricavare lo schema logico (il circuito digitale) che calcola la funzione combinatoria.
In generale, **per una data tabella di verità possono esistere più reti combinatorie** (soluzione non unica)

Esistono svariate procedure di sintesi di reti combinatorie, che differiscono per:
- **Complessità** della procedura di sintesi
- **Ottimalità** della rete combinatoria risultante, per dimensioni e velocità

Data una funzione booleana, la soluzione iniziale al problema di determinare una sua espressione consiste nel ricorso alle forme canoniche.

Le forme canoniche sono due:
- **Somma di prodotti (SoP)** (sintesi in 1a forma canonica)
- **Prodotto di somme (PoS)** (sintesi in 2a forma canonica).

Data una funzione booleana esiste **una ed una sola** forma canonica SoP e PoS che la rappresenta.

Sintesi in 1° forma canonica (SoP)

Data una tabella di verità, a n>=1 ingressi, della funzione da sintetizzare, la funzione **F** che la realizza può essere specificata come **Somma di Prodotti (SoP)**: somma logica (OR) di tutti (e soli) i termini prodotto (AND) delle variabili di ingresso corrispondenti agli 1 della funzione.

Ogni **termine prodotto** (o **mintermine**) è costituito dal prodotto logico delle variabili di ingresso (letterale) prese in forma naturale se valgono 1, in forma complementata se valgono 0.

Esempio

a b	f(a,b)
0 0	0
0 1	1
1 0	0
1 1	1

È intuitivo osservare che la funzione possa essere ottenuta dall' OR delle seguenti funzioni

a b	f(a,b)
0 0	0
0 1	1
1 0	0
1 1	1

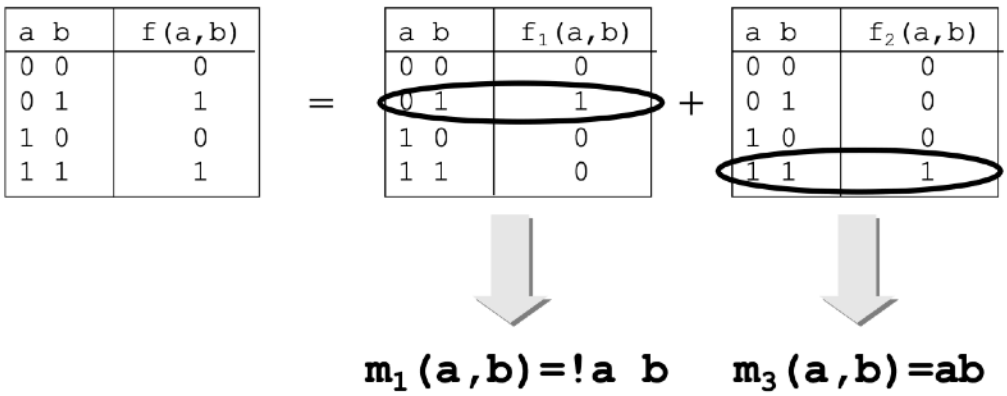
=

a b	$f_1(a,b)$
0 0	0
0 1	1
1 0	0
1 1	0

+

a b	$f_2(a,b)$
0 0	0
0 1	0
1 0	0
1 1	1

Per cui intuitivamente di ottiene:



→ **f (a, b) = !ab + ab**
Mettendo in OR i **mintermini** della funzione si ottiene l'espressione booleana della funzione stessa espressa come somma di prodotti

Mintermine

- Un **mintermine** m_i è una funzione booleana a n ingressi che vale 1 in corrispondenza della sola configurazione di ingresso che vale i.
- Esempio: **m_1(a,b)= !ab**
- Un **mintermine** m_i corrisponde al **prodotto** di **n letterali**, ciascuno dei quali compare nella forma **x** se nella corrispondente configurazione di ingresso ha **valore 1**, nella forma **!x** se ha **valore 0**.
- Data una funzione ad **n** ingressi, possiamo avere fino a **2^n** mintermini.
- Ogni **mintermine** è rappresentabile con un **AND** a **n** ingressi.

Costo di una rete logica

Sintesi in 1° forma canonica: la funzione logica è costituita da:

- **n mintermini pari agli 1 della funzione**
- per ogni mintermine, **n letterali pari al numero delle variabili di ingresso**

Il costo della rete è funzione sia del numero di mintermini che del numero di letterali (variabili o variabili negate).

Sintesi in 2° forma canonica (PoS) - dualità

Data una tabella delle verità, a n>=1 ingressi, della funzione da sintetizzare, la **funzione F** che la realizza può essere specificata come **Prodotto di Somme (PoS)**: prodotto logico (AND) di tutti (e soli) i termini somma (OR) delle variabili di ingresso corrispondenti agli 0 della funzione.

Ogni **termine somma** (o **maxtermine**) è costituito dalla somma logica delle variabili di ingresso (letterale) prese in forma naturale se valgono 0, in forma complementata se valgono 1.

Esempio

a b	f(a,b)
0 0	0
0 1	1
1 0	0
1 1	1

È intuitivo osservare che la funzione possa essere ottenuta dall' AND delle seguenti funzioni

a b	f(a,b)
0 0	0
0 1	1
1 0	0
1 1	1

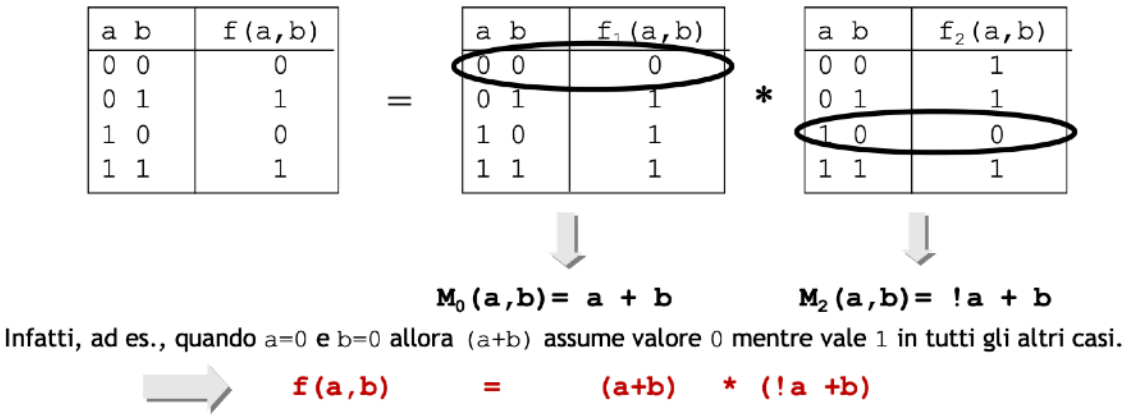
=

a b	f1(a,b)
0 0	0
0 1	1
1 0	1
1 1	1

*

a b	f2(a,b)
0 0	1
0 1	1
1 0	0
1 1	1

Per cui intuitivamente, si ottiene:

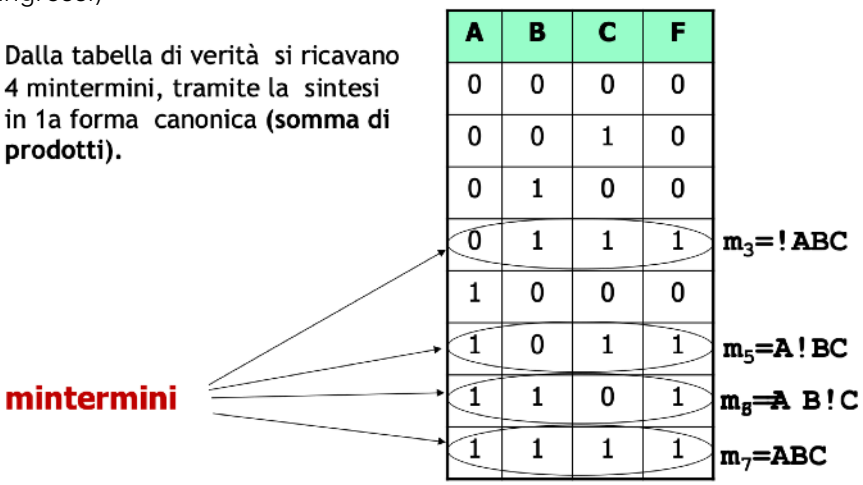


Mettendo in **AND** i ***maxtermini*** della funzione si ottiene l'espressione booleana della funzione stessa espressa come prodotto di somme.
Nel **maxtermine** (somma) una variabile compare nella forma naturale **(x)** se nella corrispondente configurazione di ingresso ha **valore 0,** nella forma complementata **(x')** se ha **valore 1.**

Esempio: Funzione di maggioranza

Si consideri una funzione combinatoria dotata di 3 ingressi (A, B e C) e di un'uscita F, funzionante come segue: l'uscita **vale 1 se e solo se 2 o tutti e 3 gli ingressi valgono 1** (cioè sse il valore 1 è presente in maggioranza sugli ingressi)

- Dalla tabella di verità si ricavano 4 mintermini, tramite la sintesi in 1a forma canonica **(somma di prodotti).**



F(A,B,C) = !ABC + A!BC + AB!C + ABC

Minimizzata:

Espressione	Proprietà utilizzata
!A B C + A !B C + A B !C + A B C	Idempotenza X + X = X
!A B C + A B C + A !B C + A B C + A B !C + A B C	Distributiva XY + X Z = X (Y+Z)
B C (!A + A) + A C (!B + B) + A B (!C + C)	Inverso (!X + X) = 1
BC 1 + AC 1 + AB 1	Identità X 1 = X
BC + AC + AB	

BLOCCHI FUNZIONALI COMBINATORI

slide 3

Esiste una **libreria di blocchi funzionali** predefiniti di tipo combinatorio. I tipici blocchi funzionali combinatori sono:

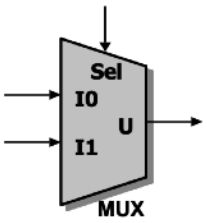
- **Multiplexer**
- **Demultiplexer**
- **Decoder** (decodificatore)
- **Confrontatore**
- **Shifter combinatorio**
- **Half adder** e **Full adder**
- **Addizionatore a n bit**
- **ALU or, and, not e somma**

Multiplexer (o selettore)

- n >= 1 ingressi di selezione
- 2^n >= 2 ingressi dati
- 1 uscita
- Gli ingressi sono numerati a partire da 0: k = 0, 1, 2, ... 2^n -1
- Se sugli inglesi di selezione è presente il valore k, il k-esimo ingresso dati viene inviato in uscita

Multiplexer 2:1

- 1 ingresso di selezione SEL
- 2 ingressi dati I0 e I1
- 1 uscita U



SEL	U
0	I0
1	I1

U = !SEL I0 + SEL I1

Tabella di verità

SEL	I0	I1	U
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Multiplexer 4:1

- 2 ingresso di selezione S0, S1
- 4 ingressi dati I0, I1, I2, I3
- 1 uscita U

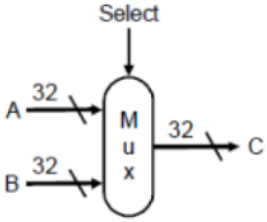
S1	S0	U
0	0	I0
0	1	I1
1	0	I2
1	1	I3

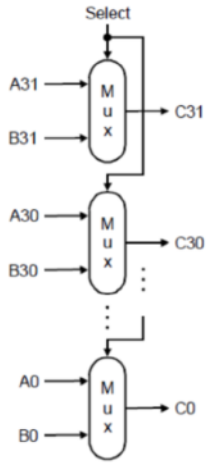
Tabella di verità
(ridotta nel num. di righe)

# riga	S1	S0	I0	I1	I2	I3	U
0	0	0	0	X	X	X	0
1	0	0	1	X	X	X	1
2	0	1	X	0	X	X	0
3	0	1	X	1	X	X	1
4	1	0	X	X	0	X	0
5	1	0	X	X	1	X	1
6	1	1	X	X	X	0	0
7	1	1	X	X	X	1	1

Multiplexer 2:1 con ingressi dati da k bit

- 1 ingresso di selezione **Select**
- 2 ingressi dati **A** e **B** da 32bit
- 1 uscita **C** a 32bit





Demultiplexer

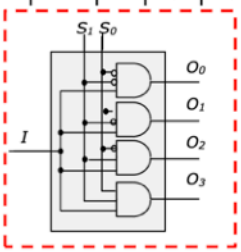
- n >= 1 ingressi di selezione
- 1 ingresso dati
- 2^n >= 2 uscite

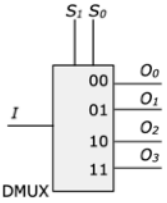
- Le uscite sono numerate a partire da 0: k = 0, 1, 2, … , 2^n -1
- Se sugli ingressi di selezione è presente il numero binario k, l'ingresso dati viene inviato alla k-esima uscita, le rimanenti sono a 0

Ingressi	Selezione		Uscite			
I	S_1	S_0	O_0	O_1	O_2	O_3
D	0	0	D	0	0	0
D	0	1	0	D	0	0
D	1	0	0	0	D	0
D	1	1	0	0	0	D

$O_0 = !S_1!S_0\,I \quad O_2 = S_1!S_0I$

$O_1 = !S_1S_0I \quad O_3 = S_1S_0I$

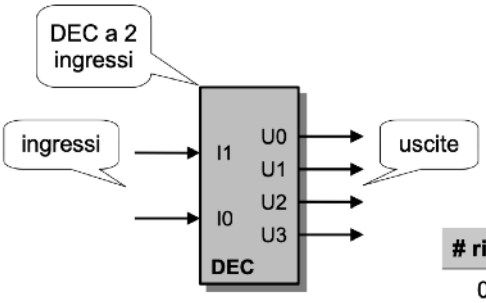




Decodificatore (decoder)

- n>=1 ingressi
- 2^n >= 2 uscite

- Le uscite sono numerate a partire da 0; k = 0, 1, 2, …, 2^n -1
- Se sugli ingressi è presente il numero binario k, la k-esima uscita assume il valore 1 e le restanti uscite assumono il valore 0



U0 = ! I1 !I0
U1 = ! I1 I0
U2 = I1 !I0
U3 = I1 I0

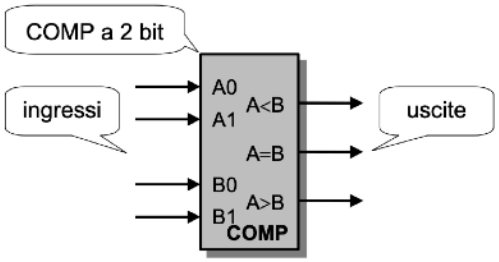
Tabella di verità

# riga	I1	I0	U0	U1	U2	U3
0	0	0	1	0	0	0
1	0	1	0	1	0	0
2	1	0	0	0	1	0
3	1	1	0	0	0	1

Confrontatore (comparator)

- 2 gruppi A e B di ingressi da n>=1 bit ciascuno
- 3 uscite: A<B, A=B, A>B

- Il blocco confronta i due numeri binari A e B da n bit presenti sui due gruppi di ingressi, e attiva a 1 l'uscita corrispondente all'esito del confronto



Confrontatore di numeri a 2 bit

Tabella delle verità

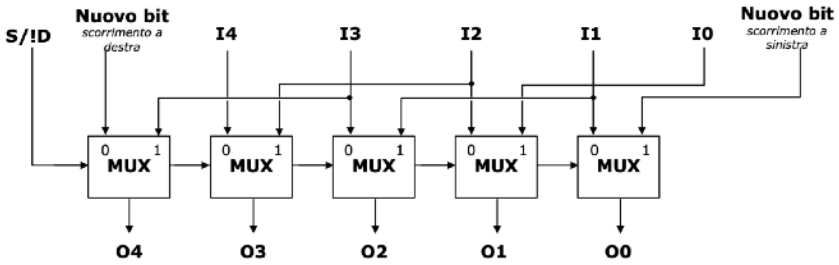
# riga	A1	A0	B1	B0	A<B	A=B	A>B
0	0	0	0	0	0	1	0
1	0	0	0	1	1	0	0
2	0	0	1	0	1	0	0
3	0	0	1	1	1	0	0
4	0	1	0	0	0	0	1
5	0	1	0	1	0	1	0
6	0	1	1	0	1	0	0
7	0	1	1	1	1	0	0
8	1	0	0	0	0	0	1
9	1	0	0	1	0	0	1
10	1	0	1	0	0	1	0
11	1	0	1	1	1	0	0
12	1	1	0	0	0	0	1
13	1	1	0	1	0	0	1
14	1	1	1	0	0	0	1
15	1	1	1	1	0	1	0

Shifter combinatorio

- n >= 1 ingressi
- 1 ingresso per il bit aggiunto a dx (scorrimento a sinistra)
- 1 ingresso per il bit aggiunto a sx (scorrimento a destra)
- 1 ingresso di controllo che comanda lo scorrimento a destra o a sinistra
- n >= 1 uscite
- Uscite:
  - scorrimento a dx: bit aggiunto a sx + ingressi shiftati di una posizione a dx (viene "perso" il bit meno significativo degli ingressi)
  - scorrimento a sx: bit aggiunto a dx + ingressi shiftati di una posizione a sx (viene "perso" il bit più significativo degli ingressi)

- Si noti che se si considerano gli ingressi come un valore numerico espresso in binario naturale
  - lo **scorrimento a dx** (con bit aggiunto a sx = 0) equivale ad una **divisione per 2**
  - lo **scorrimento a sx** (con bit aggiunto a dx = 0) equivale ad una **moltiplicazione per 2**

Shifter combinatorio 5 ingressi



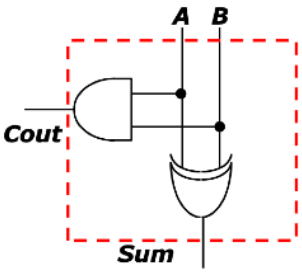
	O4	O3	O2	O1	O0
Shift DX (SEL = 0)	Nuovo bit DX	I4	I3	I2	I1
Shift SX (SEL = 1)	I3	I2	I1	I0	Nuovo bit SX

Half Adder

Sum = A ⊕ B

Cout = AB

A	B	Cout	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

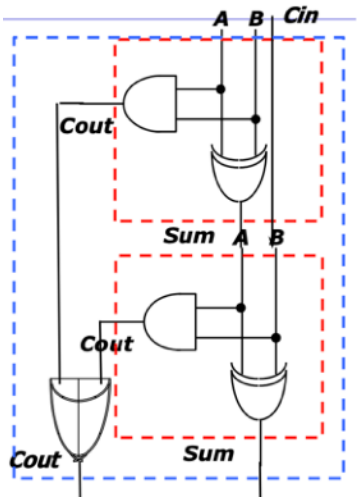


Full Adder

Sum = (A xor B) xor C_{in}

C_{out} =AB+C_{in}(A xor B)

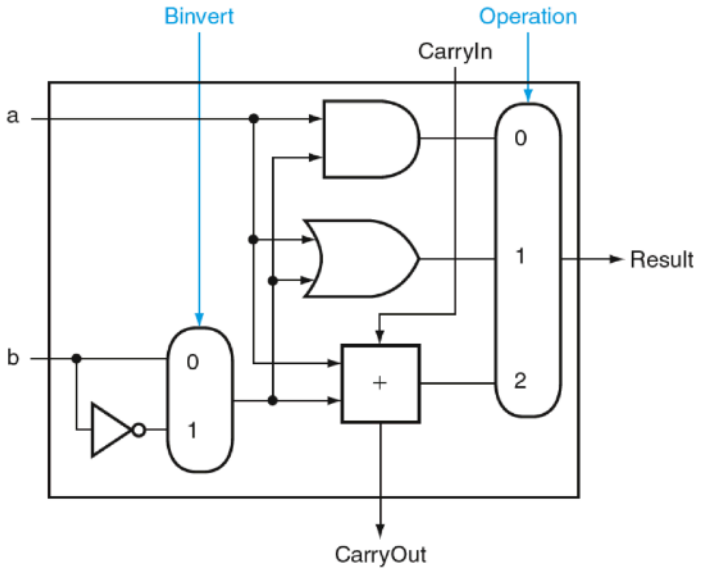
A	B	Carry In	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



ALU RISC-V

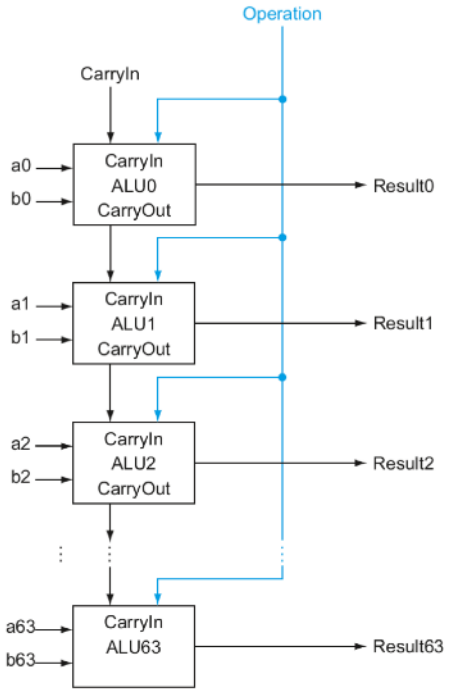
ALU a 1bit : AND, OR, ADD/SUB di a e b in cp2

Selezionando !b (Binvert = 1) and impostando CarryIn al bit meno significativo dell'ALU, otteniamo 2 sottrazioni in complemento di b rispetto all'addizione di b ad a



ALU a 64bit

CarryOut del bit meno significativo è connesso al **CarryIn** del bit più significativo. Questa organizzazione è chiamata **ripple carry**

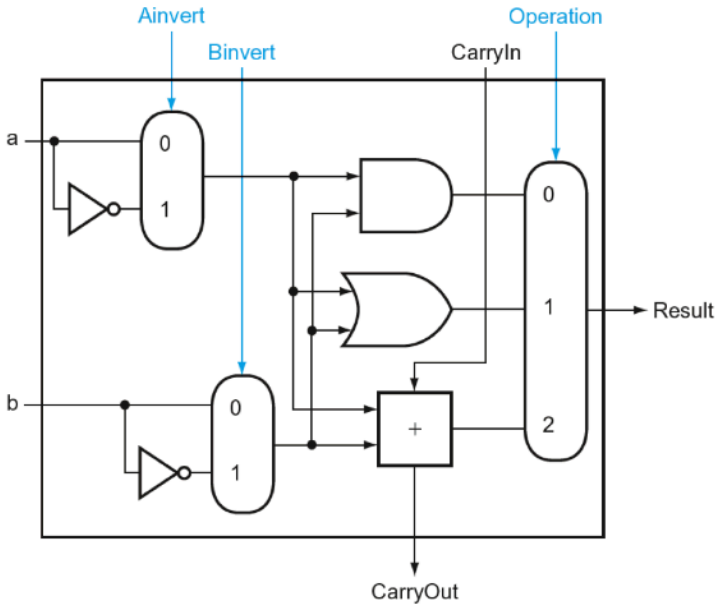


ALU a 64 bit

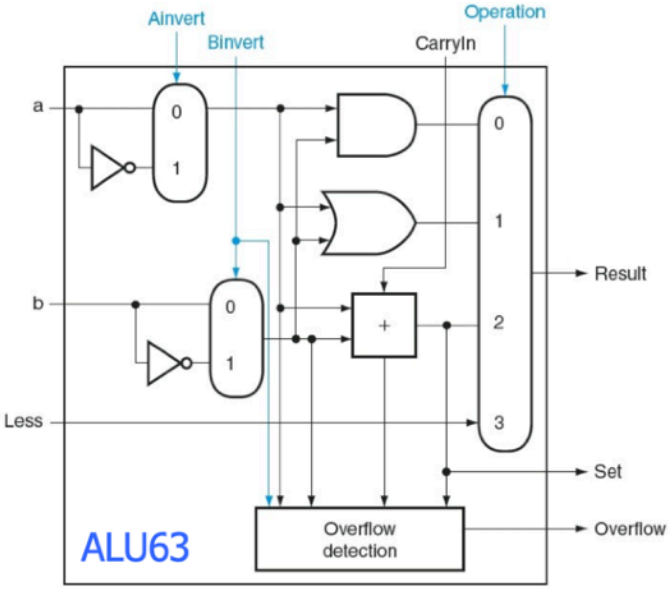
Aggiungiamo il NOR di a,b

Per De Morgan, il **NOR di a,b** si ottiene calcolando:
!(a OR b) = !a AND !b

Schema ALU a 1 bit valido ad eccezione del bit più significativo a cui è necessario aggiungere la gestione del segnale di **overflow** ↓



Overflow detection: ALU63 usata per il bit più significativo



Per numeri interi relativi in CMPL2, **l'overflow** si verifica quando si sommano 2 numeri dello stesso segno e il segno della somma non è concorde con il segno dei 2 numeri:
Overflow = a_{63} b_{63} !R_{63}+ !a_{63} !$b_{63}R_{63}$

Il segnale di overflow va a 1 se

Msbit oper_1	MSbit oper_2	Msbit risultato
0	0	1
1	1	0

Dove: `oper_1 = a` per somma e sottrazione,
`oper_2 = b` per somma e
`oper_2 = b_negato` per sottrazione

ALU63: Aggiungiamo slt (set less than)

`slt s1, s2, s3`

- se **s2 < s3** allora **s1 = 1** altrimenti **s1 = 0:** 0000 0000 ..... 0000 000**S**
- **s2 < s3** equivale a **s2 – s3<0**
- se **s2 – s3<0** allora il bit di segno del **sommatore è 1 altrimenti è 0**

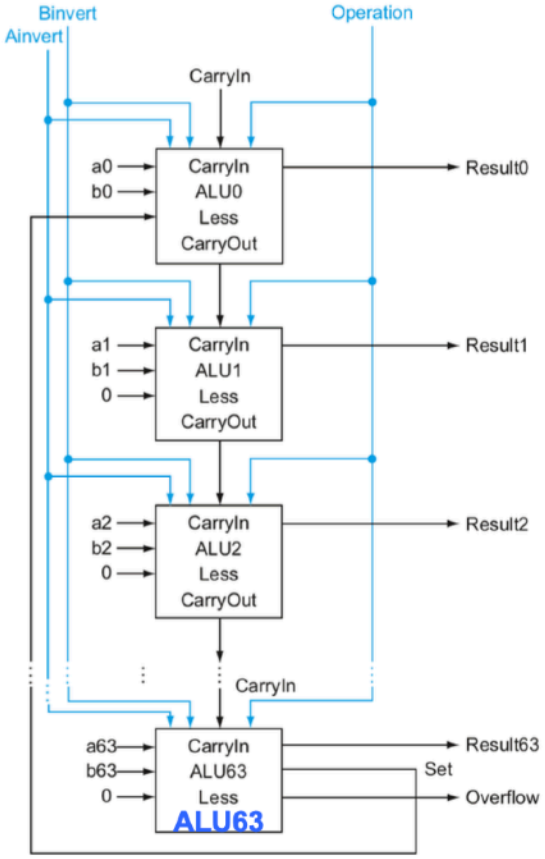
L'ingresso ***Less*** viene portato direttamente in uscita tramite il MUX: deve **valere 0 per i 63 bit più significativi di s1** e **vale 1 oppure 0 per il bit meno significativo** (ALU0)
=> Il bit ***Set*** deve essere portato in uscita e sarà collegato direttamente al bit ***Less*** di **ALU0**

ALU a 64bit

Set e Overflow
in uscita solo da **ALU63** (bit più significativo)

Il bit ***Set*** deve essere portato in uscita da **ALU63** e sarà collegato direttamente al bit ***Less*** di **ALU0** (bit meno significativo)

Slt (generato con i valori di tutti i bit)

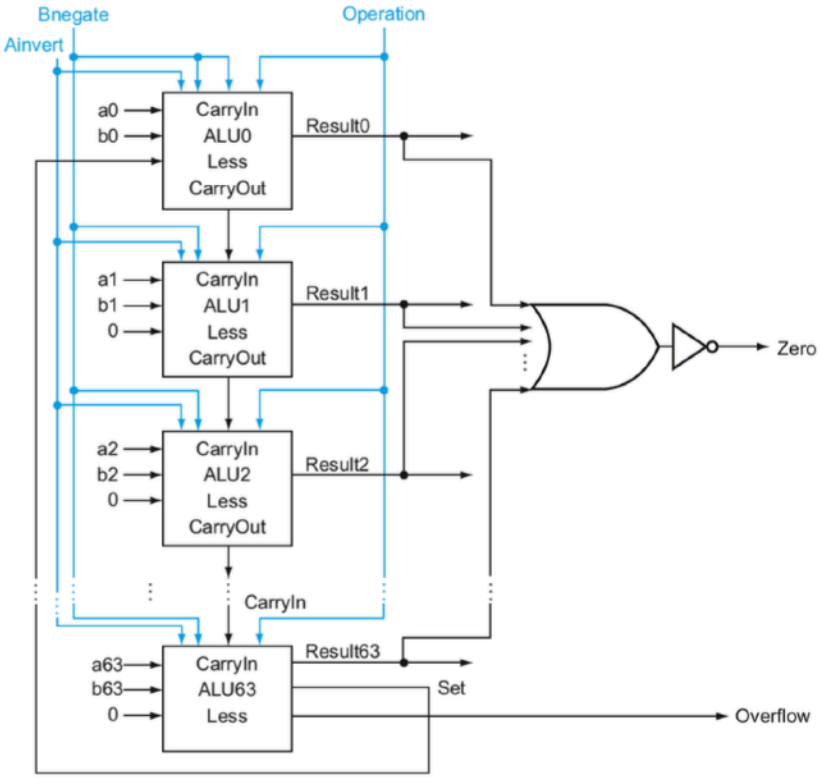


Bit Zero e Bnegate

Zero
- vale **1** se il risultato dell'ALU vale zero su 64-bit
- altrimenti vale **0**

Bnegate
Per la sottrazione in cpl2 è necessario avere i segnali ***Binvert*** e ***CarryIn*** a 1 mentre per le altre operazioni considerate (eccetto il NOR) sono entrambi a 0

Vengono sostituiti dall'unico segnale ***Bnegate***

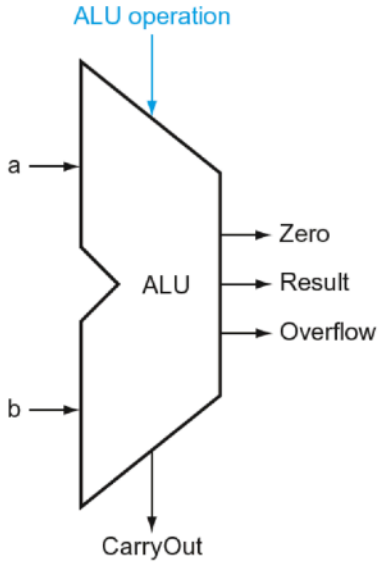


ALU del processore RISC-V e linee di controllo

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set less than
1100	NOR

Le 4 linee di controllo dell'ALU corrispondono ai bit:

Ainvert, Bnegate, Operation (2bit)



... da usare come blocco combinatorio

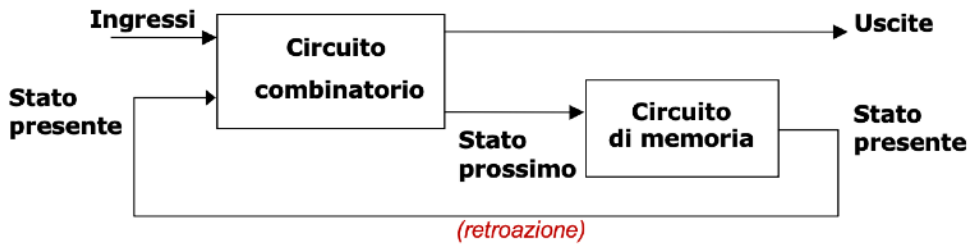
I CIRCUITI SEQUENZIALI

slide 2

- Un **circuito digitale è di tipo sequenziale** se le sue uscite dipendono non solo dai **valori correnti degli ingressi**, ma anche da (alcuni dei) **valori passati.**
- Un circuito digitale sequenziale (o rete sequenziale) è pertanto dotato, in ogni istante di tempo, di uno **stato presente** che, insieme ai valori degli ingressi, **ne determina lo stato prossimo.**
    - Lo **stato** di un circuito sequenziale rappresenta una forma di **memoria** e contiene una sorta di descrizione dell'**evoluzione** del circuito stesso.
- L'elemento funzionale elementare per la realizzazione di circuiti sequenziali è il **bistabile (elemento di memoria)**, che è in grado di memorizzare un bit di informazione.

I circuiti digitali sequenziali sono formati da:
- **bistabili** → hanno la funzione di **memorizzare** i bit di informazione
- **porte logiche** → organizzate in reti **combinatorie**, che hanno funzioni di elaborazione di informazioni

Il circuito sequenziale ha, in ogni istante, **uno stato**: il valore dei bit memorizzati nei **bistabili** facenti parte del circuito.

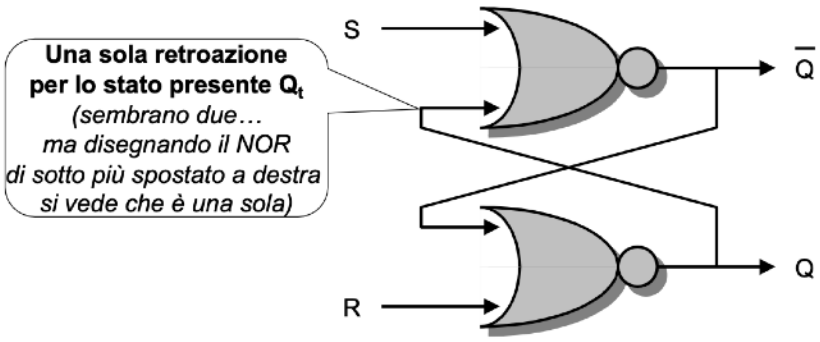


Bistabili

Sono caratterizzati da **due stati (0 e 1) stabili**. Mantengono lo stato memorizzato finché uno o più segnali di ingresso **forzano** il cambiamento dello stato.

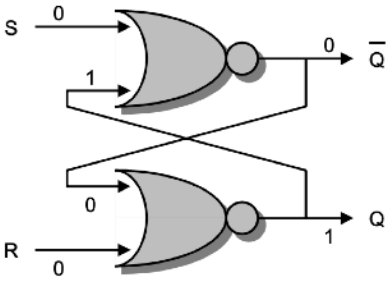
Esistono **2 famiglie** di bistabili:
1. **Asincroni** → privi di un segnale di sincronizzazione e modificano lo stato rispondendo direttamente a eventi sui segnali di ingresso
2. **Sincroni** → sensibili ad un segnale di controllo (o di sincronizzazione detto **clock**): la transizione da uno stato all'altro può avvenire solo in corrispondenza di eventi del segnale di clock
    - si può dire che il comportamento di un circuito sincrono viene osservato in **istanti discreti** di tempo

BISTABILE SR ASINCRONO

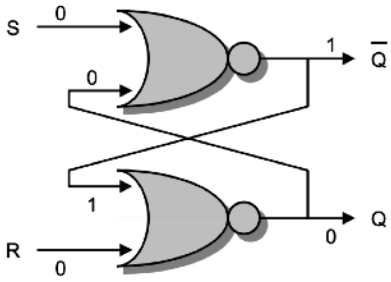


- dotato di **2 ingressi**: **S (set)** e **R (reset)**
- dotato di **2 uscite**: **Q** e **!Q**
- l'uscita **Q** rappresenta lo **stato memorizzato** e **!Q** è il suo **negato**

Come funziona il bistabile SR



S = R = 0 e Q = 1
memorizza il valore **1**
(lo stato prossimo Q_{t+1} è uguale
allo stato presente Q_t



S = R = 0 e Q = 0
memorizza il valore **0**
(lo stato prossimo Q_{t+1} è uguale
allo stato presente Q_t

NOR

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

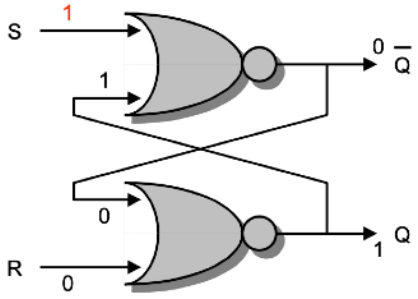
Ha due stati stabili (da qui il nome bistabile) → se S = R = 0, l'uscita Q mantiene memorizzato il valore logico perchè lo stato prossimo Qt+1 è uguale allo stato presente Qt

Transizioni di stato: SET

SET → se S = 1 e R = 0, qualunque sia il valore dello stato presente Qt → lo stato Qt+1 viene portato a 1

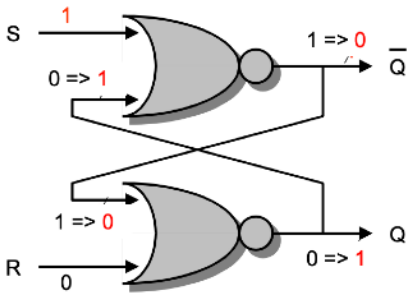
Ingresso SET = 1 e Q_t = 1

S=1 e R=0, con Q_t =1 → Q_{t+1} rimane a 1



Ingresso SET = 1 con transizione da Qt = 0 a Qt+1 = 1

S=1 e R=0, con Q_t=0 → Q_{t+1} commuta a 1

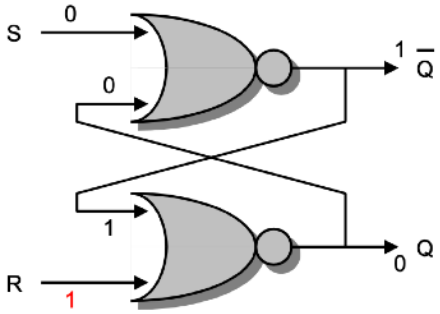


Transizioni di stato: RESET

RESET → se S = 0 e R = 1, qualunque sia il valore dello stato presente Q_t → stato prossimo Q_{t+1} viene portato a 0

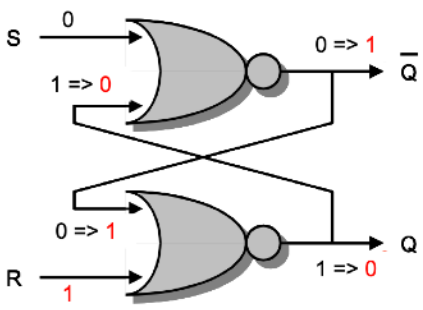
Ingresso RESET con Q_t = 0

S=0 e R=1 → Q_{t+1} rimane a 0



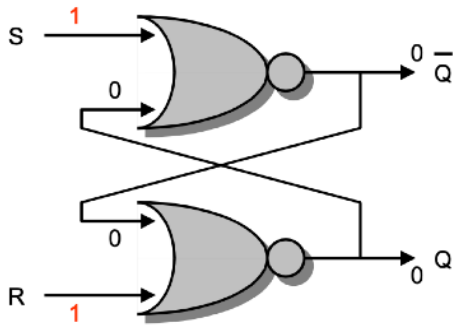
Ingresso RESET con transizione da Qt = 1 a Qt+1 = 0

S=0 e R=1, con Q_t=1 → Q_{t+1} commuta a 0



Anomalia di funzionamenti S = R = 1

S = 1 e R = 1, allora idealmente Q e !Q a 0



Se applicassimo la configurazione di ingresso S = R = 1 avremmo una condizione anomala con Q e !Q entrambi a 0!!!
E poi lo stato prossimo non sarebbe determinabile: dipenderebbe da chi tra S e R rimarrà a 1 più a lungo. Per questi motivi si preferisce vietare la configurazione d'ingresso S = R = 1

Funzionamento del bistabile SR asincrono

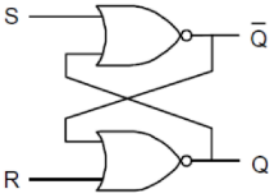
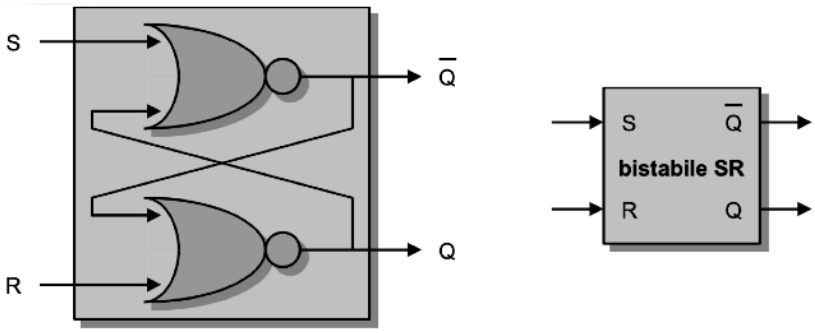


Tabella di verità

Stato presente | Stato prossimo

Q_t	S_t	R_t	Q_{t+1}	
0	0	0	0	**Stato stabile**
0	0	1	0	**Stato di reset**
0	1	0	1	**Stato di set**
0	1	1	(vietato)	**Stato indefinito**
1	0	0	1	**Stato stabile**
1	0	1	0	**Stato di reset**
1	1	0	1	**Stato di set**
1	1	1	(vietato)	**Stato indefinito**

Rappresentazione bistabile SR asincrono



Simbolo del bistabile SR (set-reset)

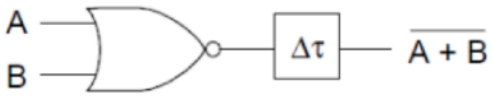
Diagramma temporale

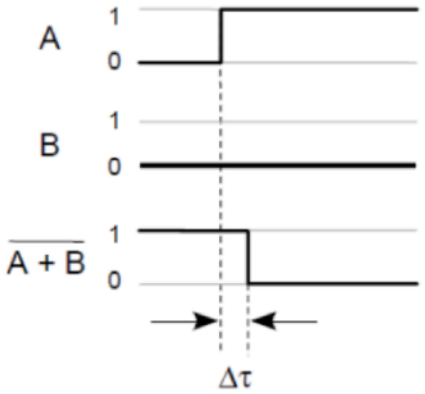
- Per visualizzare il comportamento di circuiti digitali che dipendono dal tempo e da eventi passati (per circuiti sequenziali) usiamo il **diagramma temporale**
- **Diagramma temporale**: sistema di assi cartesiani con
  - in ascissa il **tempo** (in istanti discreti);
  - in ordinata i **valori logici** (0, 1) dei segnali che evolvono nel tempo.

Ritardo di commutazione

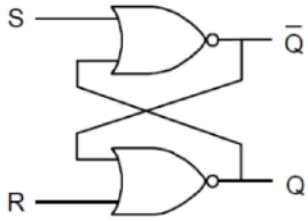
I bistabili (sincronizzati o no) e in generale le porte logiche presentano un **ritardo di commutazione dell'uscita rispetto all'ingresso.**

La commutazione dell'uscita avviene con **un certo ritardo Δτ** rispetto alla variazione degli ingressi o al fronte di clock che hanno indotto la transizione di stato.

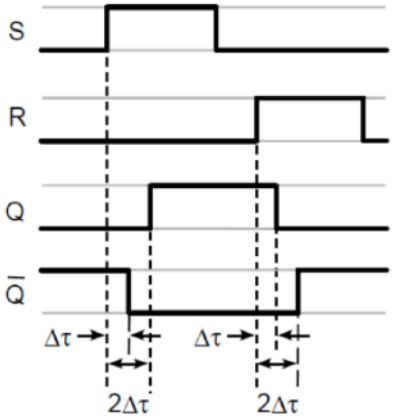




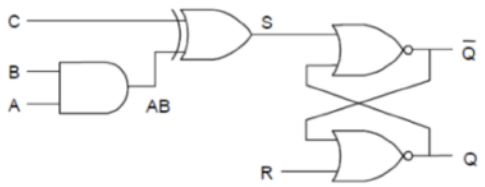
Nota: se trascuriamo il ritardo di commutazione, cioè se assumiamo che valga 0 → Q e !Q commutano istantaneamente alla variazione degli ingressi



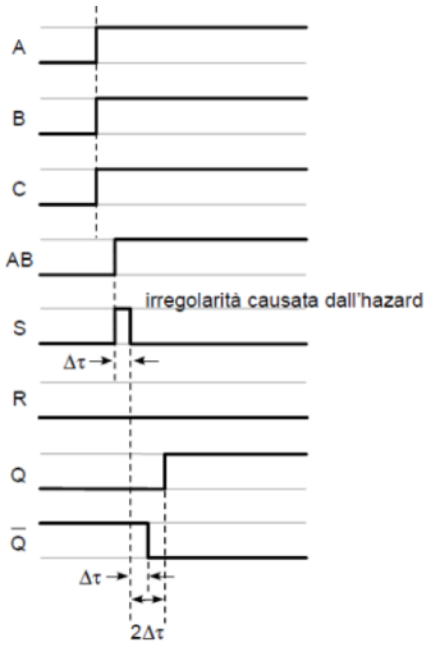
Q_t	S_t	R_t	Q_{t+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	(vietato)
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	(vietato)



A volte succede che ci sia un hazard / glitch in ingresso, che per esempio in questo caso è dovuto ad un diverso tempo di ritardo tra AB e C



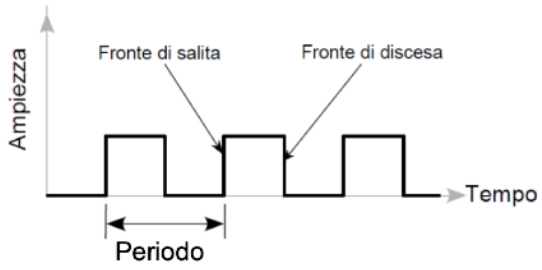
E' meglio essere in grado di "disattivare" il latch in modo che non sia sensibile a questi *hazard*.



SEGNALE DI CLOCK

In molte situazioni, è necessario che lo stato di un bistabile possa cambiare solo in determinati intervalli di tempo. Per ottenere questo occorre disporre di un segnale di clock (o di sincronizzazione) che scandisca gli istanti o intervalli di tempo in cui le transizioni di stato possono avvenire.
Il segnale di clock è un segnale binario con andamento periodico nel tempo
Il segnale di clock è una successione di impulsi: ogni impulso ha una larghezza costante e due impulsi consecutivi stanno a una distanza costante

Temporizzazione del segnale di clock



Periodo = 1 ns (ad esempio)

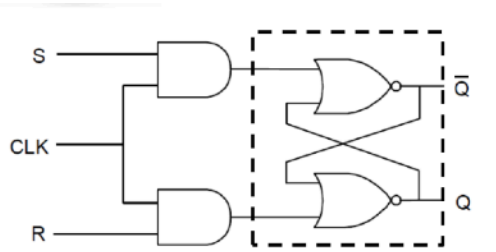
Frequenza di clock = 1 / periodo di clock = 1 / 1 ns = 1 GHz

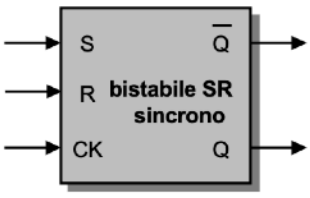
BISTABILE SR SINCRONO

- **2 ingressi S e R** (che costituiscono i segnali di Set e Reset)
- **1 ingresso di sincronizzazione (clock)**
- **2 uscite**: **Q** rappresenta lo stato del bistabile, e l'uscita negata **!Q**

Comportamento

- se **clock = 0** → gli ingressi S e R sono **inefficaci** (**latch SR opaco**), e il bistabile mantiene memorizzato lo stato corrente
- se **clock = 1** → gli ingressi S e R sono **efficaci** (**latch SR trasparente**), e il comportamento è lo stesso descritto per il bistabile SR asincrono



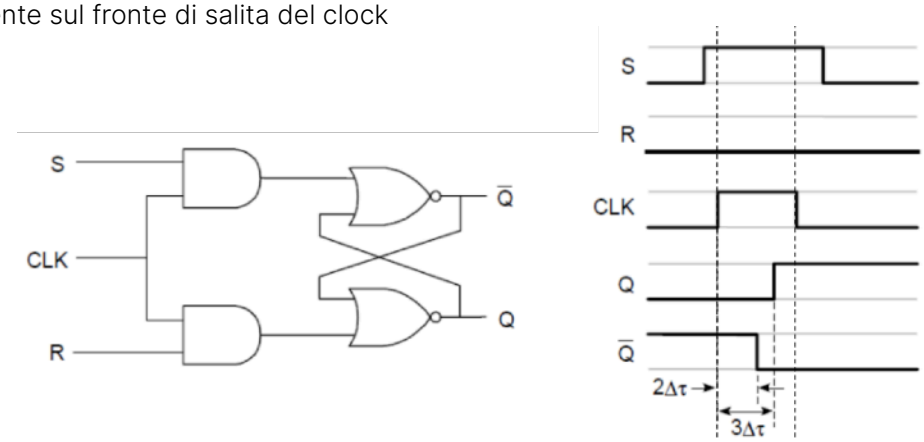


- ❑ Se il clock vale **0**, gli ingressi sono inefficaci (opaco) e il bistabile mantiene lo stato corrente
- ❑ **Se il clock vale 1, gli ingressi sono efficaci (latch trasparente) e funziona come un SR asincrono**

C	S	R	Q_{t+1}	
0	X	X	Q_t	Ingressi inefficaci e stato stabile
1	0	0	Q_t	Stato stabile
1	0	1	0	Stato di Reset
1	1	0	1	Stato di Set
1	1	1	-	Stato indefinito

Temporizzazione

Se trascuriamo il ritardo di commutazione, cioè se assumiamo il ritardo uguale a 0, Q e !Q commutano istantaneamente sul fronte di salita del clock

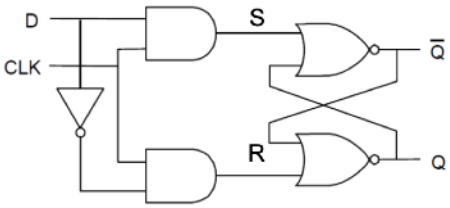


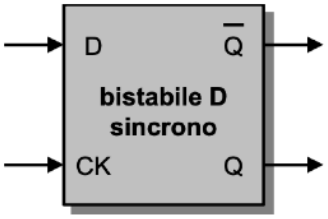
BISTABILE D SINCRONO

- **1 ingresso D** (che rappresenta il dato che verrà memorizzato)
- **1 ingresso di sincronizzazione (clock)**
- **2 uscite**: l'uscita **Q** che rappresenta lo stato del bistabile, e l'uscita negata **!Q**

Comportamento

- se **clock = 0** → l'ingresso D è **inefficace** (**latch D opaco**), e il bistabile mantiene memorizzato il suo stato corrente
- Se **clock = 1** → l'ingresso D è **efficace** (**latch D trasparente**), e il bistabile memorizza il valore logico (0 oppure 1) presente sull'ingresso D

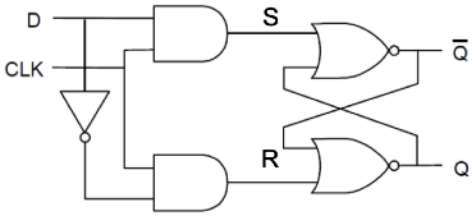




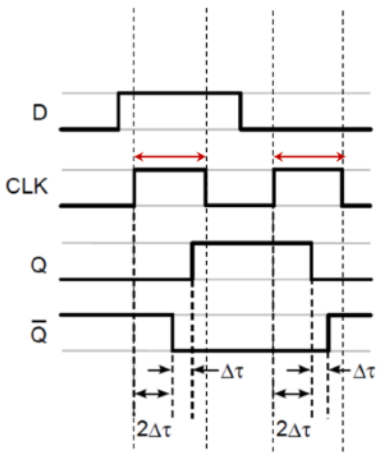
C	D	Q_{t+1}
0	X	Q_t
1	0	0
1	1	1

Temporizzazione

Se trascuriamo il ritardo di commutazione, cioè se assumiamo il ritardo uguale a 0, Q e !Q commutano istantaneamente sul fronte di salita del clock



Intervalli di trasparenza: quando il segnale di clock CLK è attivo (linee rosse)



Trasparenza

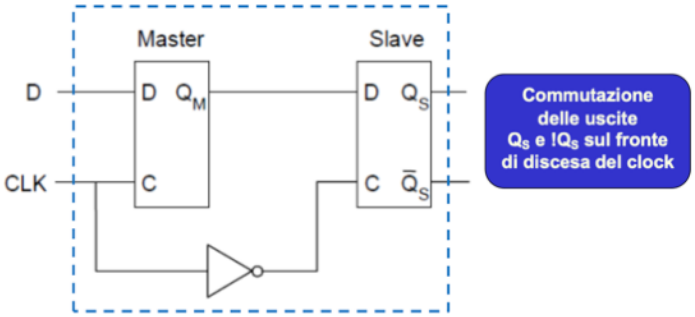
- Durante l'intervallo di tempo in cui il clock è attivo, i latch sincroni (SR o D) presentano il fenomeno della trasparenza delle uscite.
- In questo intervallo, se gli ingressi si modificano => le uscite seguono questa modifica e possono dunque commutare più volte.
- Durante l'intervallo di trasparenza, bisogna mantenere stabili gli ingressi

---

FLIP-FLOP D MASTER-SLAVE

Per evitare il fenomeno della trasparenza delle uscite si usano i **flip-flop (D o SR)** che sono costituiti da **2 bistabili D collegati in cascata** (nominati **Master** e **Slave**) con clock invertiti in modo che lo stato possa modificare le uscite solo in corrispondenza del **fronte** del segnale di clock.

Coppia di bistabili sincroni D trasparenti collegati incassata con clock invertiti per eliminare il fenomeno della trasparenza →



Funzionamento

- Il bistabile **Master** campiona l'ingresso **D** durante l'intervallo alto del clock, lo emette sull'uscita **Q_M** collegata all'ingresso D dello Slave.
- Il bistabile **Slave** campiona l'ingresso **D** durante l'intervallo basso del clock e lo emette sull'uscita QS

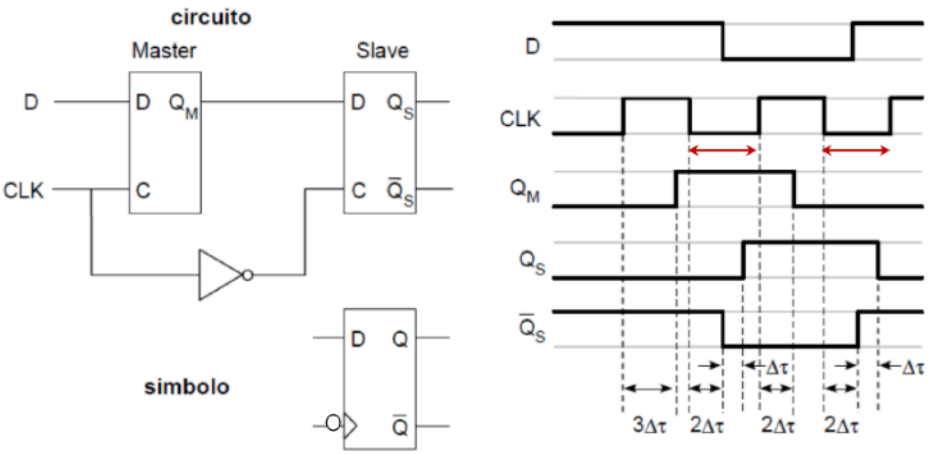
=> **L'uscita QS può variare solo sul fronte di discesa del clock**

- **Trasparenza dei bistabili Master e Slave:**
  - Nell'intervallo **alto** del clock, il bistabile **Master è trasparente**.
  - Nell'intervallo **basso** del clock, il bistabile **Slave è trasparente.**

- In pratica Q_M **non** cambia durante l'intervallo basso del clock (quando lo Slave sarebbe trasparente) **in modo da eliminare il fenomeno della trasparenza verso le uscite.**

Temporizzazione

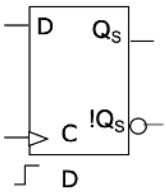
La fase crescente del clock determina l'arrivo di nuovi dati al Master, mentre lo Slave mantiene i dati precedenti. La base discendente fa si che i nuovi dati del master vengano passato allo slave. Dunque il F/F cambia stato alla discesa del clock.



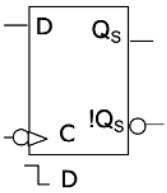
Q_M non cambia durante l'intervallo basso del clock (quando lo Slave sarebbe trasparente) in modo da eliminare il fenomeno della trasparenza verso le uscite

Convenzioni sul clock

L'evoluzione delle uscite del FF Master Slave è controllata dal fronte (di salita o di discesa) del clock denominato fronte attivo



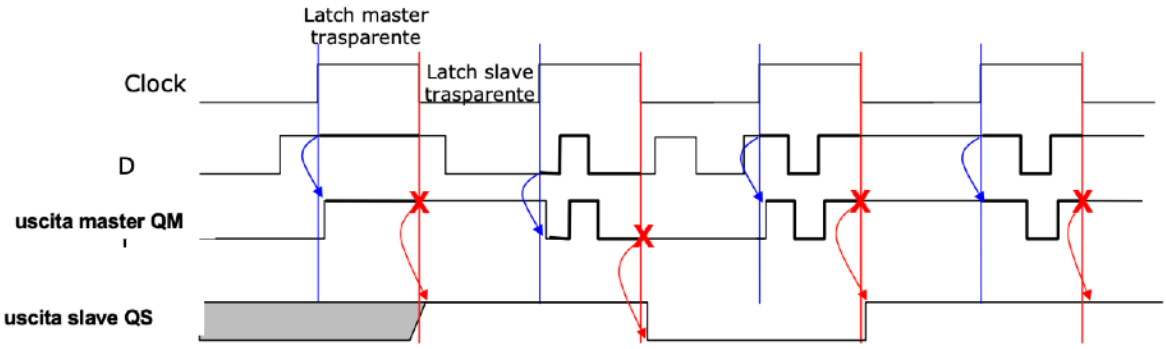
Il FF master-slave commuta le uscite Q_S e $!Q_S$ sul fronte di ***salita*** del clock



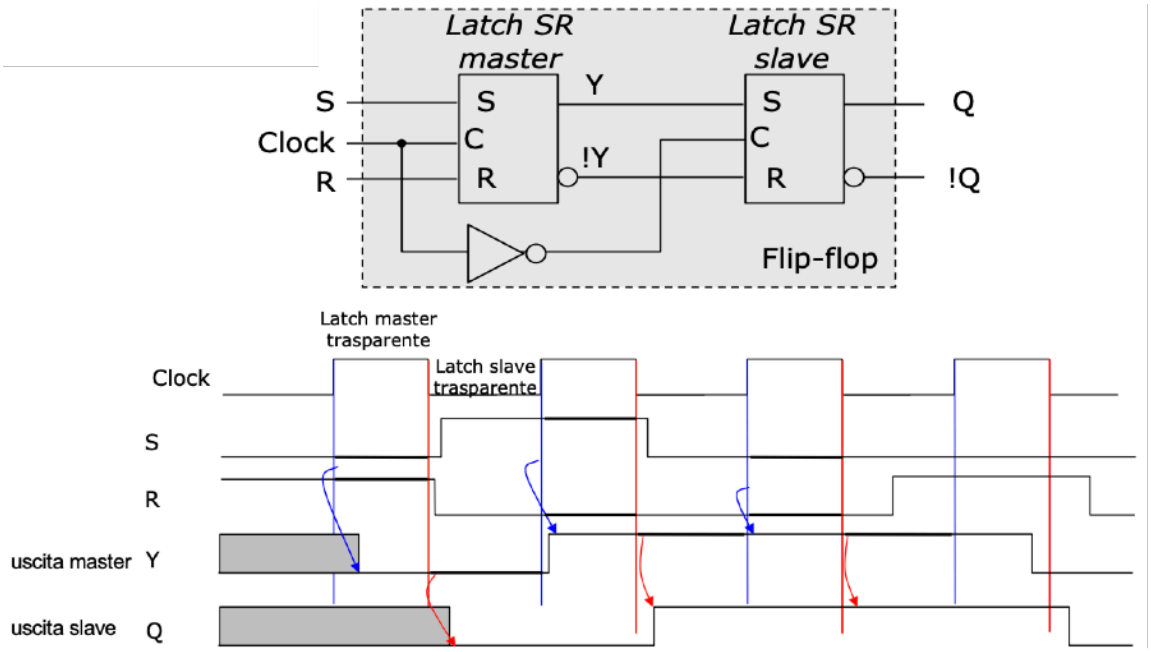
Si usa il simbolo ***NOT sul clock*** per indicare che il FF master-slave commuta le uscite Q_S e $!Q_S$ sul fronte di ***discesa*** del clock

Filtraggio dei disturbi o glitch presenti sul segnale di ingresso D

Funzionamento **«da specifica»**: durante tutto il livello alto del clock l'ingresso D deve essere stabile e rappresenta **il valore del dato che si vuole memorizzare**.
Eventuali disturbi sull'ingresso D vengono filtrati dall'uscita Q_S perché Q_M non cambia durante l'intervallo basso del clock (quando lo Slave sarebbe trasparente) **in modo da eliminare il fenomeno della trasparenza verso le uscite**



FLIP-FLOP SR MASTER-SLAVE



SOMMARIO

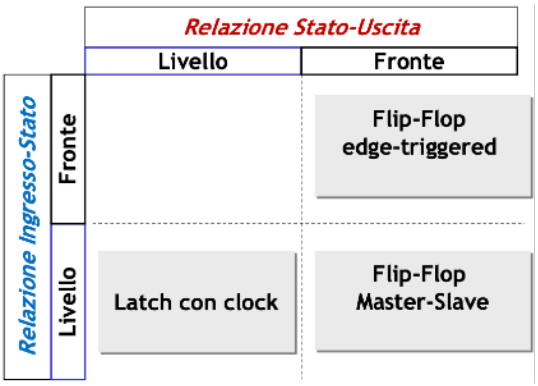
BIstabili sincroni e temporizzazione

I fattori che differenziano i bistabili riguardano 2 aspetti:

1. La ***relazione ingresso-stato*** (tipo di **temporizzazione**) definisce quando gli ingressi modificano lo stato interno del bistabile
    - Commutazione basata sul **livello del segnale di sincronizzazione**
        - Durante tutto l'intervallo di tempo in cui il segnale di controllo è attivo, qualsiasi variazione sui segnali di ingresso influenza il valore dello stato interno del bistabile (bistabili con commutazione a livello)
    - Commutazione basata sul **fronte del segnale di controllo**
        - Il valore dello stato interno del bistabile viene aggiornato solamente in corrispondenza di un fronte del segnale di controllo (bistabili con commutazione sul fronte - di salita oppure di discesa).
2. La ***relazione stato-uscita*** definisce quando lo stato aggiorna le uscite.
    - Commutazione basata sul **livello del segnale di controllo**
        - Durante tutto l'intervallo di tempo in cui il segnale di controllo è attivo un cambiamento dei segnali di ingresso modifica oltre allo stato interno anche le uscite.
        - Bistabili con questa relazione stato-uscita sono denominati **LATCH**
    - Commutazione basata sul **fronte del segnale di controllo**
        - Le uscite vengono aggiornate su di un fronte del segnale di sincronismo.
        - Bistabili con questa relazione stato-uscita sono denominati **FLIP-FLOP**
        - Le uscite cambiano in corrispondenza di un evento del clock

Trasparenza

- per evitare la trasparenza delle uscite si utilizzano i flip-flop (D o SR) che sono costituiti da 2 latch in cascata in modo che lo stato possa modificare le uscite solo in corrispondenza del fonte del segnale di clock
- Nei flip-flop (D o SR)
    - Relazione stato-uscita (aggiornamento della uscita): sul fronte
    - Relazione ingresso-stato (aggiornamento dello stato): a livello (flip-flop master-slave), a fronte (flip-flop edge-triggered)



Flip Flop edge-triggered

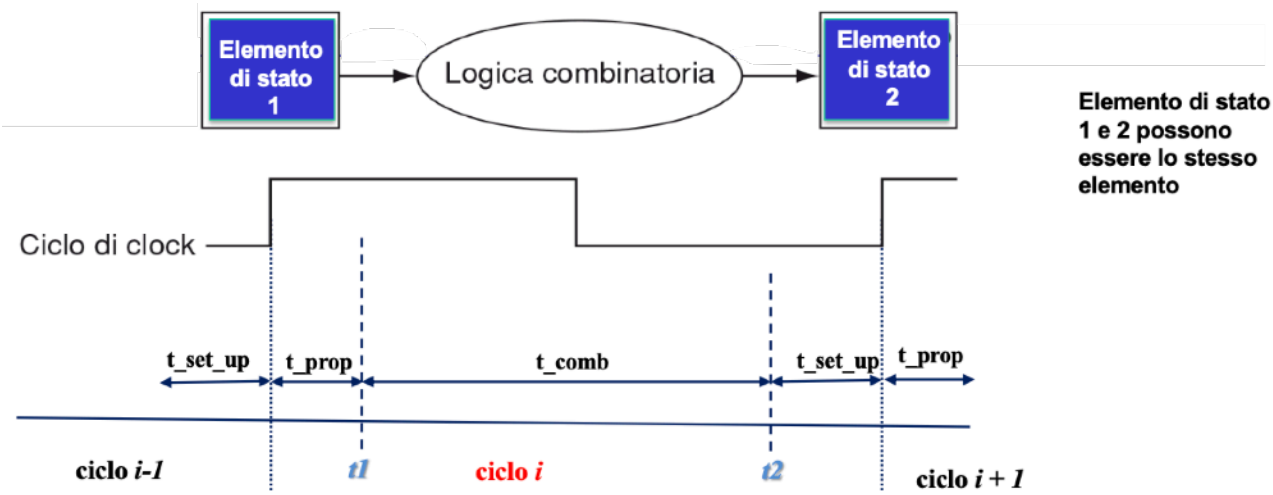
- Hanno una struttura interna più complicata rispetto ai Master Slave e per questo **non** la vediamo nei dettagli, ma studiamo il loro comportamento.
- **Limitano il problema di dover mantenere stabili gli ingressi al FF per tutto il tempo in cui il livello del clock è alto (o basso).**
- L'evoluzione del FF è controllata dal *fronte* (di salita o di discesa) del clock: **fronte attivo**
- Il fronte attivo agisce come un **"segnale di campionamento"** e in corrispondenza del fronte attivo i valori agli ingressi dell'elemento di stato vengono campionati, memorizzati e quindi presentati in uscita

Temporizzazione sensibile ai fronti (edge-triggered)

Il fronte di salita è il fronte attivo in cui gli ingressi vengono campionati e memorizzati.
Requisiti per corretto funzionamento dei singoli elementi di stato (è possibile trascurare ***T_hold***):

- ***T_set_up*** : intervallo di tempo in cui gli ingressi devono essere stabili prima del fronte attivo
- ***T_prop***: intervallo di tempo necessario perché il nuovo stato memorizzato si presenti stabile in uscita

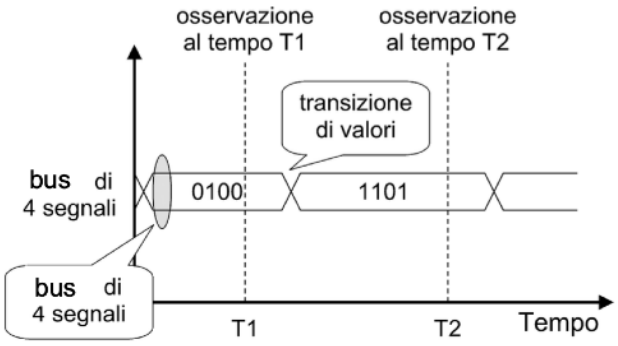
Nel circuito sopra, il periodo di clock deve essere sufficientemente lungo per consentire il corretto funzionamento della logica combinatoria ***T_clock = T_prop + T_set_up + T_comb***



BLOCCHI SEQUENZIALI

Principali blocchi sequenziali di libreria:
- Registro parallelo
- Registro a scorrimento
- Register File
- Memoria

Come rappresentare un **insieme di segnali (bus)**



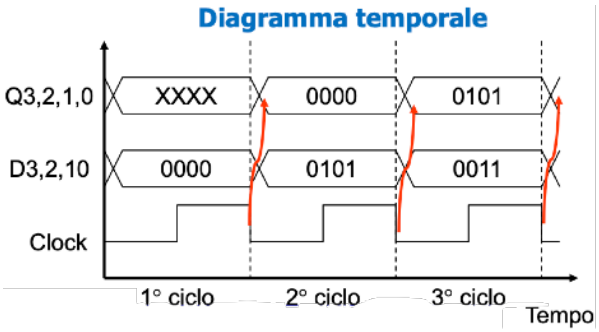
Al tempo T1 il bus vale 0100
Al tempo T2 il bus vale 1101

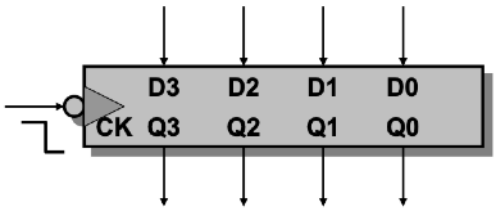
REGISTRO PARALLELO

Il registro parallelo è composto da n>=1 flip-flop D master-slave con:
- **n >= 1 ingressi D0, … , Dn-1**
- **n >= 1 uscita Q0, …, Qn-1**
- l'ingresso di **clock CK**

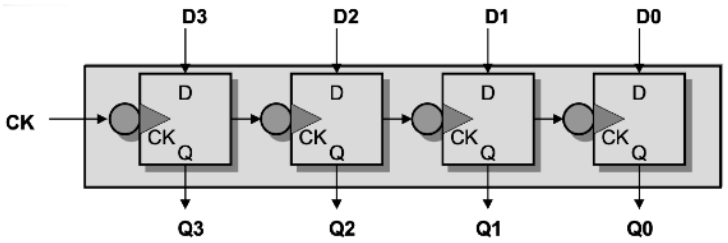
Ad ogni ciclo di clock, il registro legge e memorizza nel suo stato presente la parola di n bit di ingresso, e la presenta sugli n bit in uscita nel ciclo successivo (stato prossimo).

Registro parallelo a 4 bit: simbolo e funzionamento





Registro parallelo a 4 bit: progetto in stile funzionale



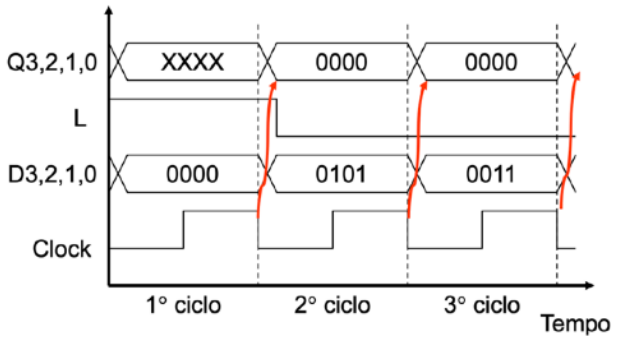
Registro parallelo progettato in stile funzionale, usando 4 flip-flop D master-slave sincroni sul fronte di discesa del clock

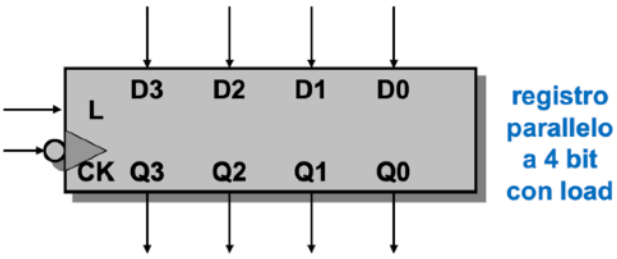
Nota: Se invece si usassero dei bistabili D trasparenti sul livello, durante il livello alto del clock il registro sarebbe del tutto trasparente, e dunque non si comporterebbe come dovuto …

Registro parallelo con load

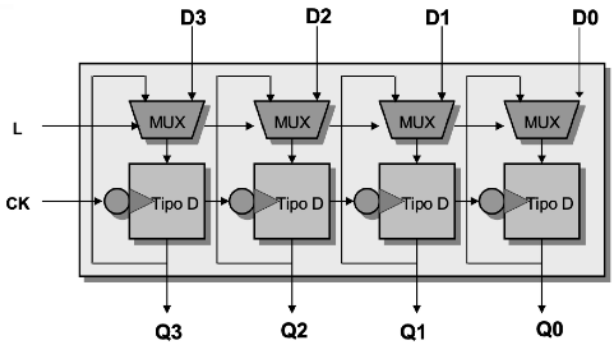
Registro parallelo a n bit con un comando in ingresso di **caricamento (Load):**

- Se il segnale di **L** è attivo **(L = 1)**, la parola in ingresso al registro viene memorizzata nel registro stesso e presentata in uscita nel ciclo successivo
- Altrimenti **(cioè L = 0)**, il registro mantiene il suo stato corrente di memorizzazione





Registro parallelo con load: progetto in stile funzionale



registro parallelo a 4 bit con load

Registro parallelo progettato in stile funzionale, usando 4 flip-flop D master- slave sincroni sul fronte (di discesa) e 4 multiplexer ad un ingresso di selezione e 2 ingressi dati con funzione di caricamento di ingresso seriale (Load) oppure parallelo (D3, D2, D1, D0)

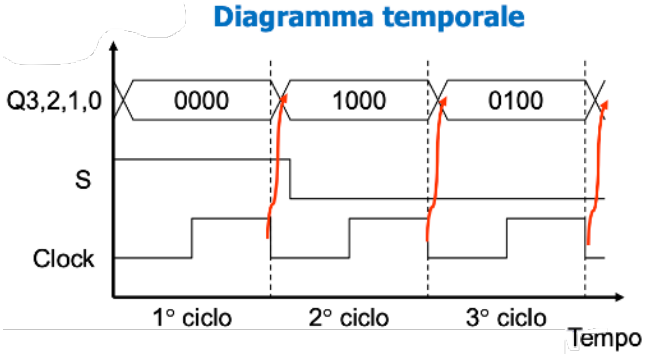
REGISTRO A SCORRIMENTO

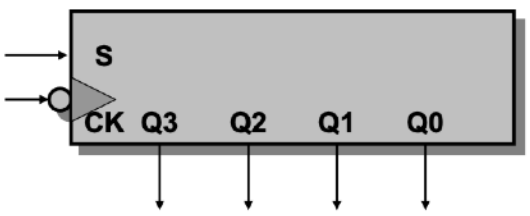
Il registro a scorrimento è realizzato da **n >= 1 flip-flop di tipo D** collegati in **cascata**:

- **un ingresso seriale S**
- **n >= 1 uscite parallele Q0, ..., Qn-1**
- **l'ingresso di clock**

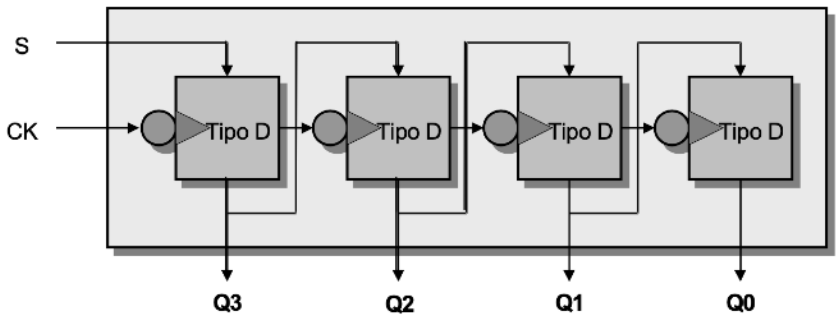
A ogni ciclo di clock, fa scorrere di un bit verso DX la parola memorizzata, perdendo il bit più a DX (meno significativo) e aggiungendo a sinistra il bit presente sull'ingresso seriale

Registro a 4 bit con scorrimento a dx: simbolo e funzionamento





Registro a 4 bit con scorrimento a dx: progetto in stile funzionale



Registro a scorrimento a DX progettato in stile funzionale, usando 4 flip-flop D master-slave sincroni sul fronte (di discesa) collegati in cascata

Altre varianti

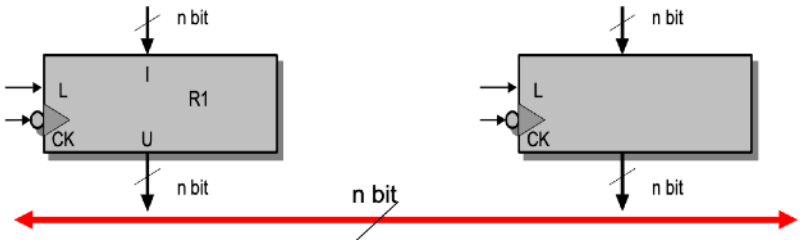
- Registro a scorrimento a SX
- Registro a scorrimento universale: DX / SX (è dotato di un comando di scelta del verso di scorrimento)
- Registro a scorrimento (DX / SX) con funzione di caricamento parallelo
- Registro parallelo / scorrimento universale: riunisce le funzioni dei registri parallelo e a scorrimento universali

- Registro IN seriale / OUT seriale
- Registro IN parallelo / OUT seriale
- Registro IN parallelo/seriale OUT parallelo/seriale

REGISTER FILE

Linee di uscita condivise

- L'organizzazione interna della memoria e la struttura dei banchi di memoria e dei banchi di registri prevedono che le uscite di 2 o più componenti siano collegate alle stesse linee di uscita (**bus**)
- Servono opportuni "elementi funzionali" **(circuiti di pilotaggio delle uscite del componente)** che garantiscano la **NON** interferenza (dei segnali) tra i moduli che condividono le stesse linee di uscita

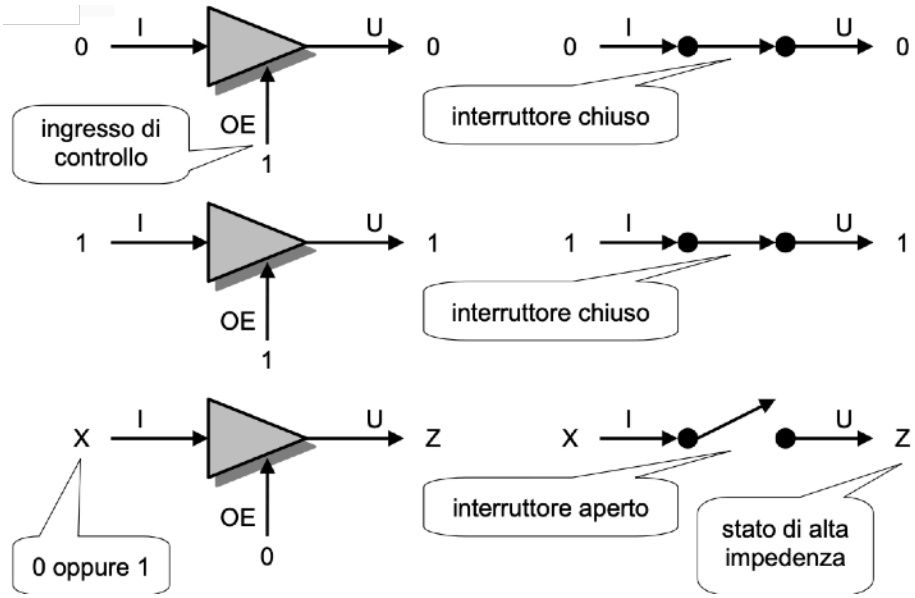


Circuiti di pilotaggio delle uscite: buffer tri-state

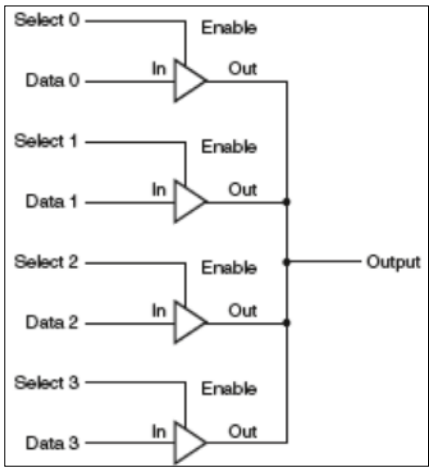
- Circuito elementare modellabile come un contatto a **3 posizioni**:
  - In stato di **bassa impedenza** consente di avere in uscita o il **livello logico alto (1)** o **basso (0)**
  - In stato di **alta impedenza (Z)** isola elettricamente l'uscita

- L'uscita tri-state viene gestita da un apposito ingresso di controllo **(output enable)** che, se vale 0 (non attivo), forza lo stato di alta impedenza

Funzionamento del buffer tri-state

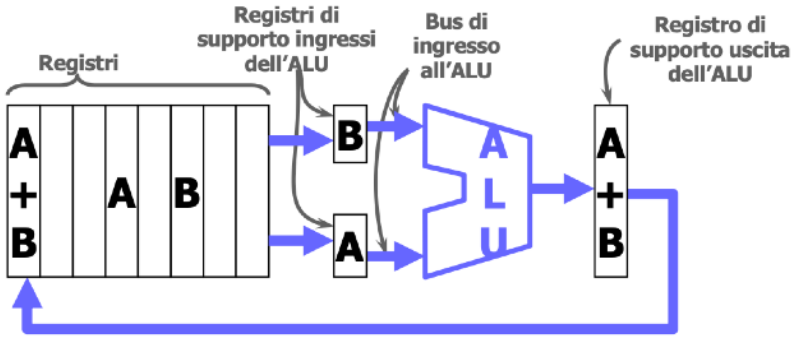


Multiplexer sulle 4 linee di uscita



Register File

Spesso occorre utilizzare un gruppo di registri paralleli con funzione di memorizzazione, tutti aventi le stesse dimensioni e le stesse funzioni (ad esempio nel data-patch della CPU).



I registri vengono organizzati in una struttura chiamata **Banco di Registri** o **Register File (RF).**

Caratteristiche

- Consideriamo come esempio di riferimento il **Register File del processore RISC-V** composto da **32 registri da 64-bit ciascuno**

- Ogni registro è identificato da un **indirizzo (numero di registro)** specificato su **5 bit**.

- Le operazioni eseguibili sul Register File sono:
  - **lettura**: si presentano in uscita dal RF i **64 bit** memorizzati nel registro indirizzato come sorgente;
  - **scrittura**: si memorizzano i 64 bit presenti all'ingresso del RF nel registro indirizzato come destinazione.

- ***Porte di lettura/scrittura***: consentono la lettura e la scrittura dei registri: nel RISC-V abbiamo **2 porte di lettura e 1 porta di scrittura** (servono per gestire le istruzioni assembly con 2 registri sorgente e 1 registro destinazione)

Register File del RISC-V

Con opportuni comandi e temporizzazioni è possibile accedere in // a 3 registri distinti attraverso:

- **2 porte di lettura**
- **1 porta di scrittura**

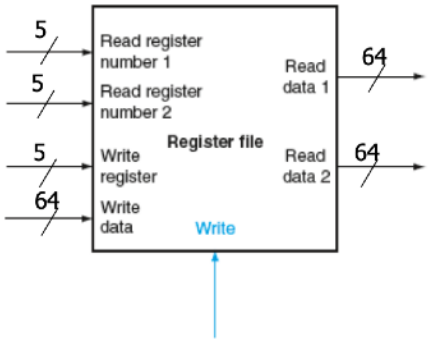
Accesso in scrittura

ingressi:

- indirizzo su 5 bit del registro da scrivere
- dati du 64 bit da scrivere all'interno del registro indirizzato
- un comando di scrittura esplicito (**Write**)

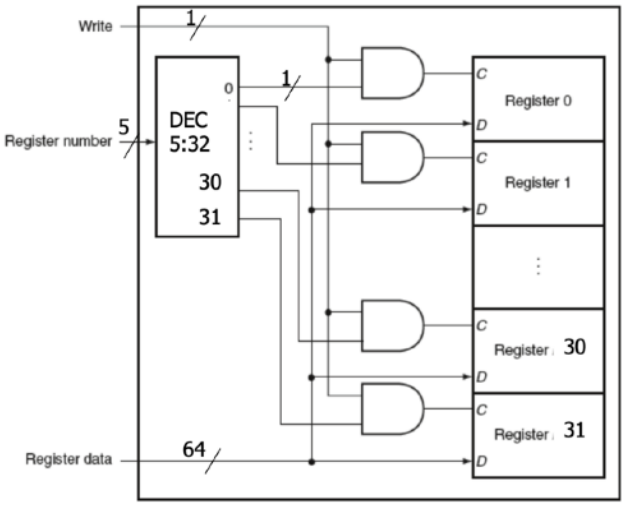
Accesso in lettura

- **ingressi**: 2 indirizzi su 5 bit dei 2 registri da leggere
- (non è necessario un comando di lettura esplicito)
- **uscite**: 2 dati da 64bit (lo stato interno del registro non viene modificato)



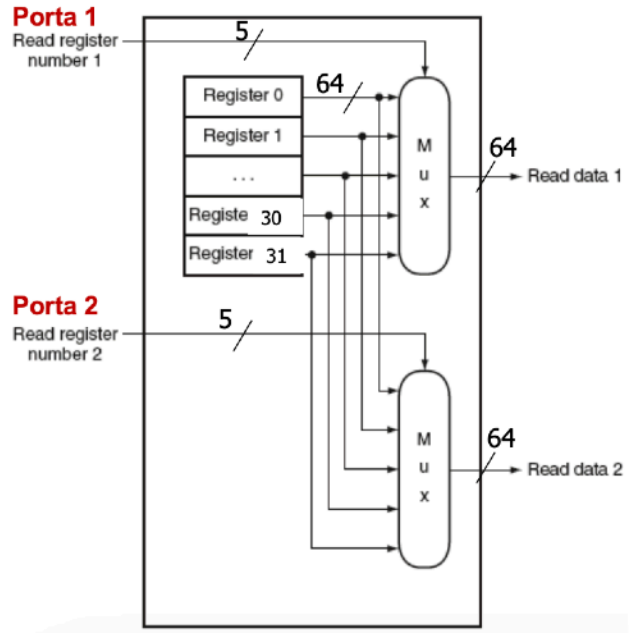
Il segnale di clock è sottointeso

Register File del RISC-V: 1 porta di scrittura



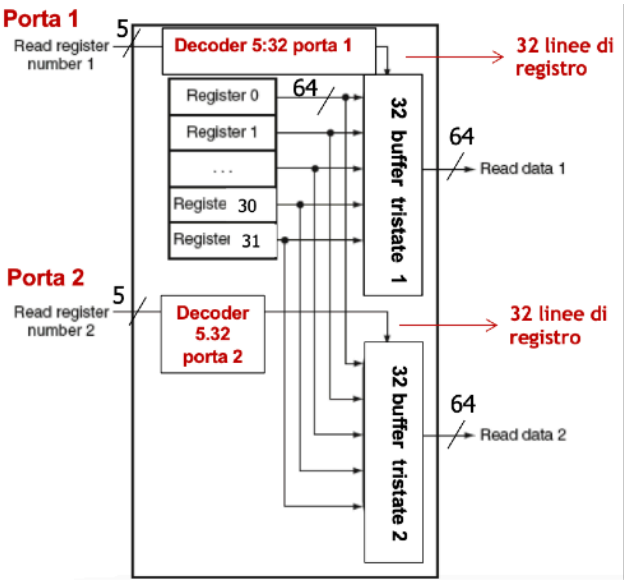
- L'indirizzo su 5 bit del registro su 64-bit da scrivere è fornito come ingresso al decodificatore 5:32.
- La ***linea «di registro»*** in uscita dal DEC e il segnale di ***write*** abilitano un solo registro in scrittura.
- Sul fronte attivo del clock (non mostrato) il dato ***D*** su 64-bit da scrivere viene memorizzato nel registro indirizzato.

Register File del RISC-V: 2 porte di lettura



- **Porta di lettura 1:** L'indirizzo su 5-bit del registro su 64-bit da leggere è usato come segnale di controllo del multiplexer 32:1 della porta 1
- **Porta di lettura 2**: L'indirizzo su 5-bit del registro 2 su 64-bit da leggere è usato come segnale di controllo del multiplexer 32:1 della porta 2
- Le due porte di lettura sono **indipendenti** e i due multiplexer vengono controllati in modo indipendente

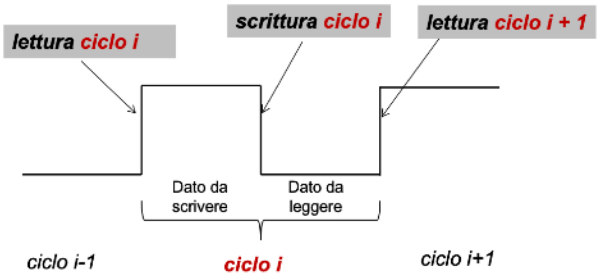
Register File del RISC-V: porte di lettura (versione con buffer tristate)



- la realizzazione richiede buffer tri-state se le uscite devono essere scollegate elettricamente
- Tutte le uscite dei registri condividono le stesse linee di uscita (di una porta)
- L'indirizzo del registro da leggere è usato come segnale di ingresso al decodificatore (5:32)
- Le 32 linee di registro (uscite del decoder) sono attive in mutua esclusione es Ono usate come linee di ***Enable*** per il buffer tristate della porta di lettura considerata
- I decodificatori delle 2 porte sono controllati in modo indipendente

Lettura e scrittura di un registro nello stesso ciclo di clock

La scrittura al ***ciclo i*** avviene sul fronte di discesa del cloche quindi la lettura deve avvenire sul fronte di salita al ***ciclo (i+1)*** e fornisce il valore scritto al ***ciclo i*** precedente.

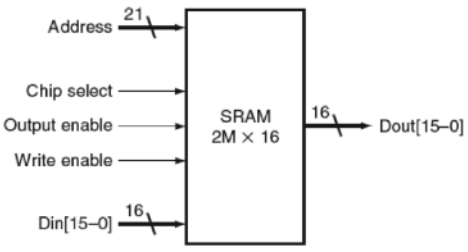


MEMORIA PRINCIPALE (MAIN MEMORY)

- La memoria è un **blocco funzionale** di tipo sequenziale complesso
  - **Memoria volatile** (solo se alimentata mantiene le informazioni memorizzate) e permette l'accesso in lettura o in scrittura: Memoria ad accesso diretto con tecnologia **RAM (Random Access Memory)** di tipo: **Statico (SRAM)** o **Dinamico (DRAM)**.
- Ha una **struttura a matrice** ad accesso diretto le cui righe sono le parole di memoria.
- Un **componente integrato** (**chip**) di memoria si caratterizza specificando:
  - la capacità, misurata in numero totale di Byte;
  - le funzioni: lettura e scrittura, solo lettura;
  - il numero di porte di accesso;
  - il tempo di accesso.

Interfaccia di memoria

- il contenuto della memoria viene letto/scritto una parola per volta in un ciclo di clock: es con parole da 16bit (half word)
- Si accede ad una parola di memoria tramite un indirizzo fornito alla **porta di acceso alla memoria**
- La porta di accesso alla memoria può funzionare in lettura e scrittura (è il caso più frequente (oppure solo in lettura)



Segnali dell'interfaccia di memoria

- Gli **ingressi di indirizzo**, codificano l'indirizzo della parola su cui scrivere /leggere.
- Le **uscite/ingressi di dato**, che servono per leggere/scrivere una parola
- Per le linee di dato e indirizzo sono da rispettare i tempi di ***set_up*** e ***hold***, quindi questi segnali vengono forniti per primi in modo che siano stabili quando le linee di comando sono attivate a seconda dell'operazione
- Il comando di scrittura, **Write enable WE** attivo a livello 1;
- Il comando di abilitazione delle uscite dati, **OE (output enable)** usato per comandare i buffer tristate: Se OE = 1 le uscite sono abilitate; Se OE = 0 le uscite sono isolate;
- Il comando di abilitazione del componente, **CS (chip select):** Se CS = 1 chip attivo, si può accedere al contenuto; CS = 0 chip non attivo.

Cicli di lettura e scrittura

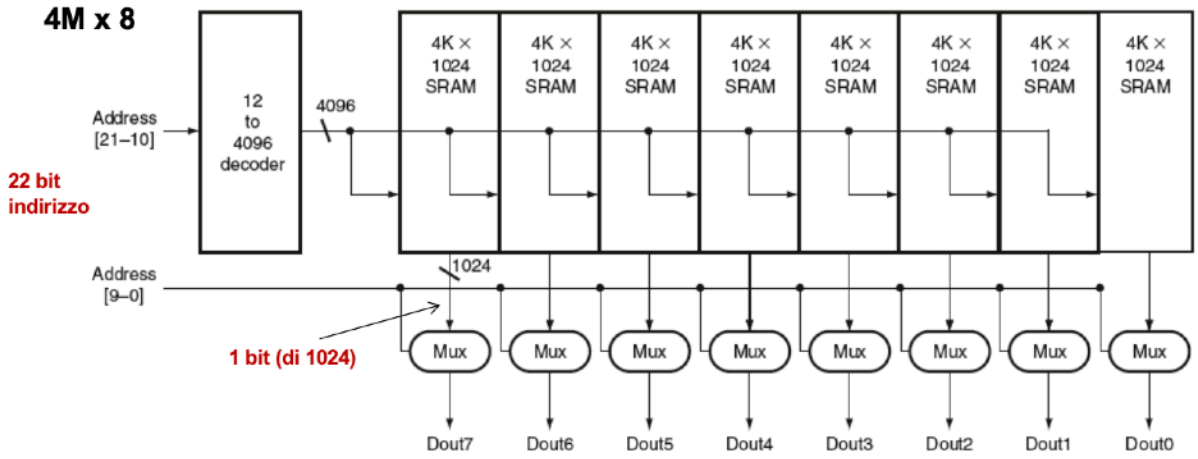
Ciclo di lettura

- indirizzo della parola da leggere
- Comando di lettura (WE a livello 0)
- Non isolare le uscite dati (OE = 1)
- Abilitare il componente (CS = 1)
- Contenuto della parola disponibile sulle uscite. Ritardo di lettura: 8-20ns.

Ciclo di scrittura

- indirizzo della parola da scrivere
- Dato da scrivere in ingresso
- Comando di scrittura (WE a livello 1)
- Isolare le uscite dati (OE = 0)
- Abilitare il componente (CS = 1). Ritardo di scrittura: 8-20ns.

Organizzazione a matrice di componenti: decodifica dell'indirizzo a 2 livelli



DRAM: qualche considerazione

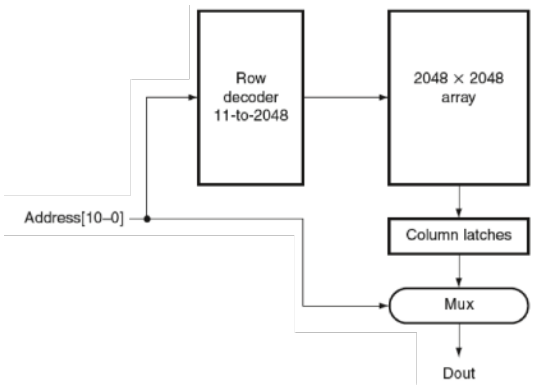
- **RAM dinamica o DRAM**: il singolo bit è «memorizzato» nella carica di un condensatore (capacità parassita) il cui accesso avviene tramite un transistor che può «leggere» o «scrivere» il suo valore (durata della carica alcuni millisecondi)
- **Vantaggi**: DRAM usa **un singolo transistor per memorizzare 1 bit di informazione** contro i 6 transistor per bit delle SRAM, quindi richiede meno area di silicio e quindi ha un costo inferiore per bit delle SRAM.
- **Svantaggi**:
  - Richiede tempi di accesso maggiori delle SRAM
  - Necessario il segnale di ***refresh*** delle parole di memoria (lettura e riscrittura del contenuto) per sopperire al problema della degradazione della carica del condensatore. Si usa decodifica dell'indirizzo a due livelli per diminuire i tempi di refresh. Le operazioni di refresh consumano meno del 2% dei cicli di lettura/scrittura utili.

Struttura di DRAM

Linee di indirizzo suddivise in: segnali **RAS (Row Address Strobe) e CAS (Column Address Strobe).**

Latch di colonna: contiene il contenuto di una riga letta.

La singola operazione di refresh riscrive un'intera riga.



RAM Dinamica (DRAM)

- La tecnologia **DRAM** usa 1 transistor per bit memorizzato
    - Sfrutta il fenomeno dell'accumulo temporaneo di carica sul transistor (capacità parassita)
    - Internamente contiene un circuito di rinfresco che rigenera le cariche
- Memoria RAM (matrice di transistor) ad altissima densità
- Capacità grande-grandissima
- Tempo di accesso medio
- Funziona in lettura e scrittura
- **Volatile**: senza alimentazione il contenuto della memoria svanisce
- DRAM usate per la Memoria principale (Main Memory) dei calcolatori

RAM Statica (SRAM)

- **Memoria RAM (Random Access Memory) Statica**: per ogni bit memorizzato servono 6 transistor (maggiore consumo di area e costo rispetto alla DRAM)
- Capacità medio-piccola
- Tempo di accesso molto veloce
- Funziona in lettura e scrittura
- **Volatile**: senza alimentazione il contenuto della memoria svanisce
- SRAM si utilizzano on-chip come memorie cache con tempo di accesso più veloce rispetto alla DRAM.

ROM (Read Only Memory)

- **Memoria ROM (Read Only Memory)**, realizzata come matrice di transistor
- Capacità grande
- Tempo di accesso medio
- Funziona in sola lettura
- **Persistente**: il contenuto permane anche in assenza di alimentazione
- Usi: per memorizzare programmi permanenti, non modificabili; grandi volumi di produzione

PROM, EPROM, EEPROM

- Capacità e tempo simili alla ROM
- Sola lettura e persistenti
- Sono programmabili sul campo, tramite un apposito programmatore:
    - PROM: programmabile una volta sola
    - EPROM: cancellabile con raggi UV
    - EEPROM: cancellabile elettricamente (si può anche scrivere un solo byte per volta)
- Usi: piccoli volumi di produzione, prototipi

Memoria FLASH

- Capacità e tempo simili alla DRAM (o solo di poco inferiori)
- Funziona in lettura e scrittura (la scrittura però è a blocchi di byte)
- **Persistente**: il contenuto permane anche in assenza di alimentazione

- Usi: dati multimediali (p. es. immagini statiche, sequenze video), programmi periodicamente aggiornabili

Tabella riassuntiva

Tipo	Categoria	Modalità di cancellazione	Scrittura byte	Volatile	Usi specifici
SRAM	lett/scritt	elettrica	si	si	cache
DRAM	lett/scritt	elettrica	si	si	mem. centrale
ROM	sola lett	nessuna	no	no	grandi vol.
PROM	sola lett*	nessuna	no	no	piccoli vol.
EPROM	sola lett*	luce UV	no	no	prototipi
EEPROM	sola lett*	elettrica	si (lenta)	no	prototipi
FLASH	lett/scritt	elettrica	a blocchi	no	multimedia

*Le memorie cancellabili vengono talvolta qualificate come “memorie prevalentemente a sola lettura” (read-mostly), invece che “a sola lettura” (read-only)