Architettura degli Elaboratori 2025-2026

Circuiti Sequenziali

Prof. Elisabetta Fersini
elisabetta.fersini@unimib.it

Circuiti combinatori vs sequenziali

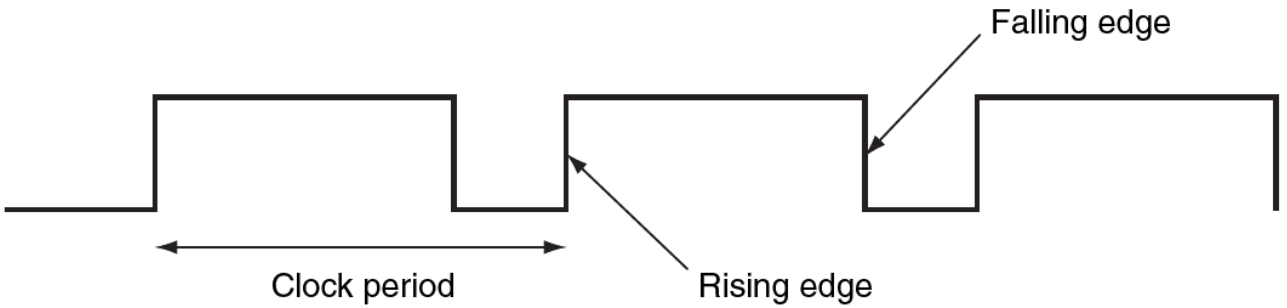
- I **circuiti combinatori** sono in grado di calcolare funzioni che dipendono **solo** dai dati in **input**
- I **circuiti sequenziali** sono invece in grado di calcolare funzioni che dipendono anche da uno **stato**
- => ovvero, che dipendono anche da informazioni memorizzate in elementi di memoria interni

Circuiti sequenziali: famiglie

- Esistono due famiglie di circuiti digitali sequenziali:
  - asincroni, non fanno uso di clock
  - sincroni, necessitano di clock
- Esempio di circuito sequenziale asincrono:
  - SR-latch
- Esempio di circuito sequenziale sincrono:
  - Flip-Flop

Clock

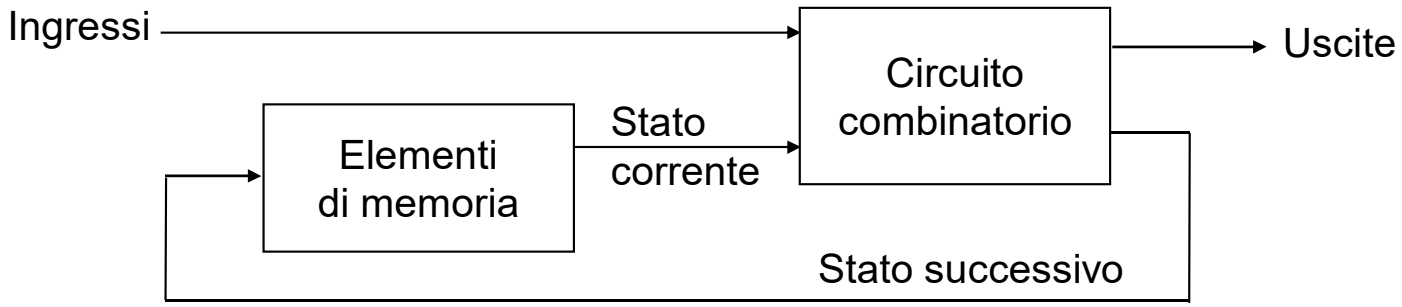
- Il segnale di **clock** è fondamentale per le reti sequanziali che sono caratterizzate da uno **stato**
- Clock – definito come un segnale (onda quadra) con un periodo predeterminato e costante



- Caratterizzato da un periodo T (oppure ciclo) di clock e dalla frequenza F(definita come 1/periodo) e misurata in Hertz F= 1/T

Circuiti sequenziali

- I circuiti sequenziali sono formati da:
  - Elementi di memoria (di vario tipo): **memorizzano** informazione
  - Reti combinatorie: **elaborano** informazione
- Un circuito sequenziale ha, in ogni dato istante, uno **stato** determinato dai bit memorizzati



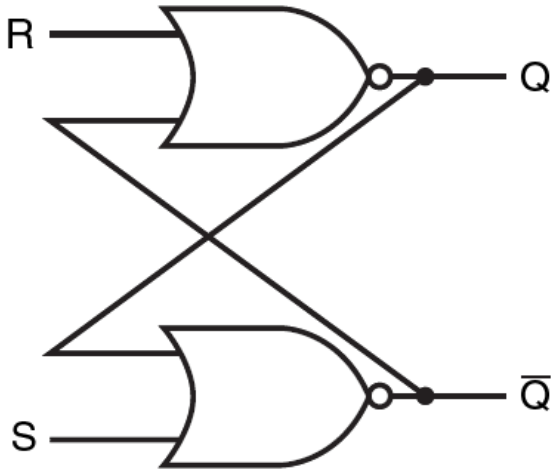
Circuiti sequenziali

- 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

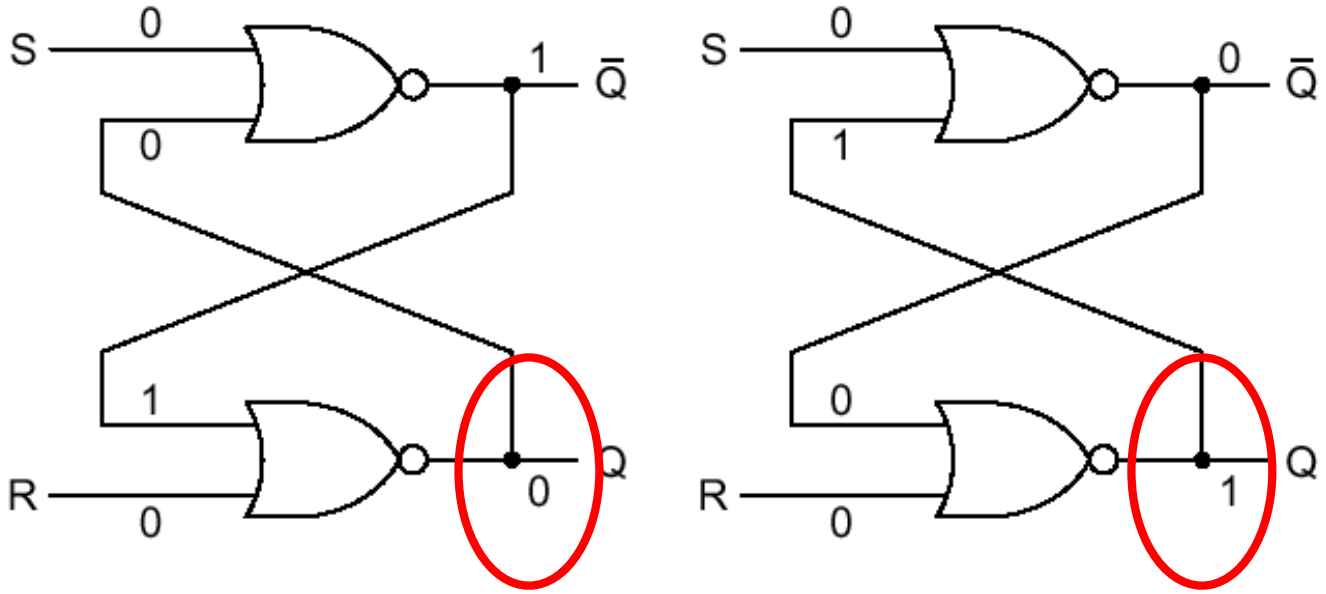
- S-R Latch è un circuito, composto da 2 porte NOR concatenate

S = Set e R = Reset.

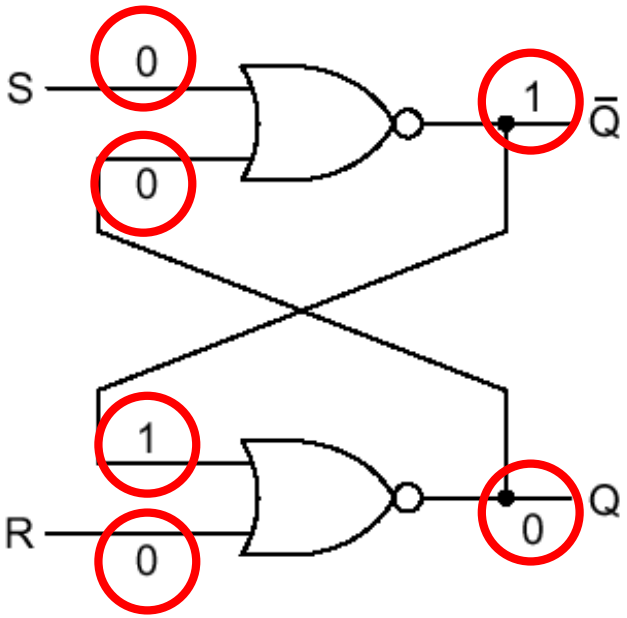


A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

S-R Latch

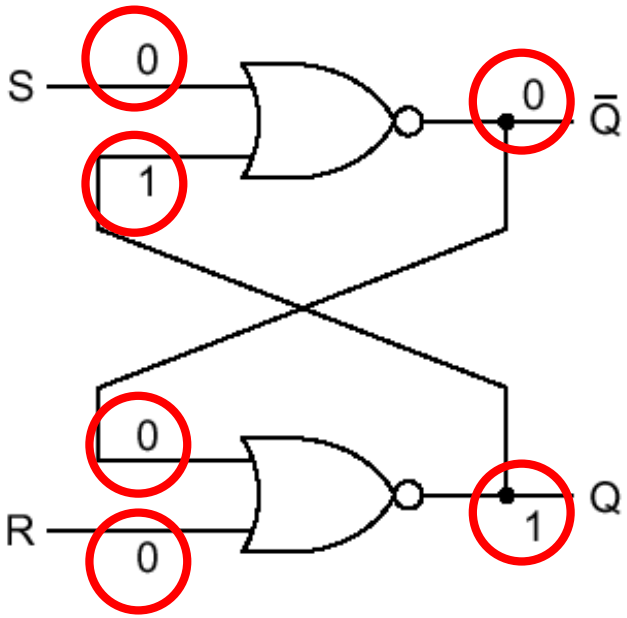


S-R Latch



Input S	Input R	Stato Interno Old Q	Output Q	Output $\bar{Q}$
0	0	0	0	1
0	0	1	1	0
0	1	1/0	0	1
1	0	1/0	1	0
1	1	1/0	0	0

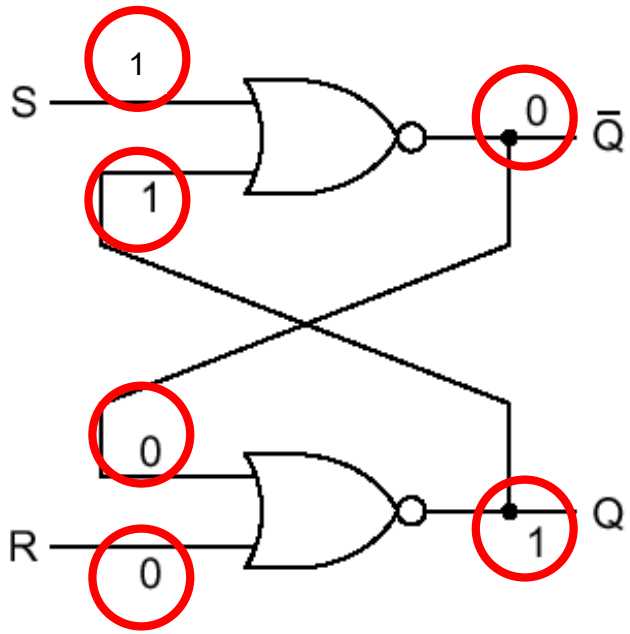
S-R Latch



Input S	Input R	Stato Interno Old Q	Output Q	Output $\bar{Q}$
0	0	0	0	1
0	0	1	1	0
0	1	1/0	0	1
1	0	1/0	1	0
1	1	1/0	0	0

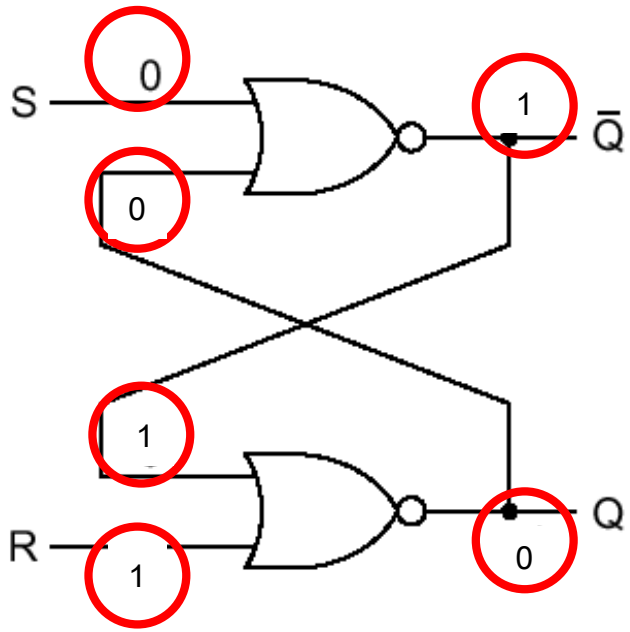
La combinazione (S,R) = (0,0) viene detta combinazione di riposo, perché semplicemente mantiene il valore memorizzato in precedenza

S-R Latch



Input S	Input R	Stato Interno Old Q	Output Q	Output $\bar{Q}$
0	0	0	0	1
0	0	1	1	0
0	1	1/0	0	1
1	0	1/0	1	0
1	1	1/0	0	0

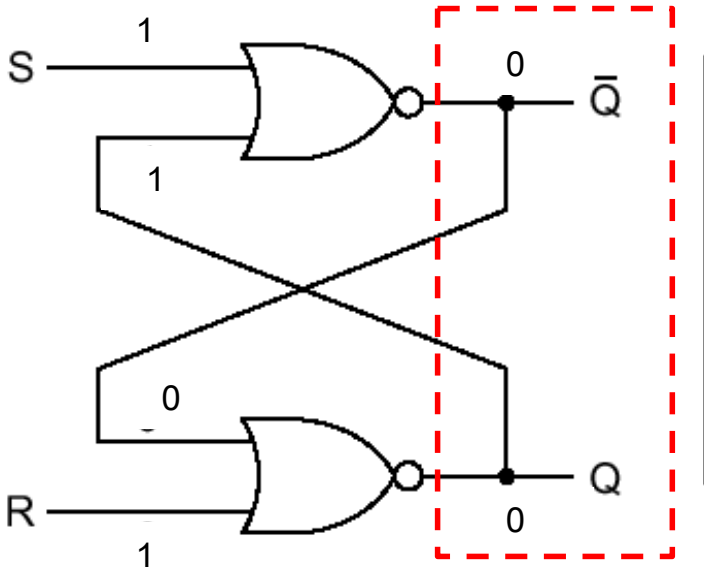
S-R Latch



Input S	Input R	Stato Interno Old Q	Output Q	Output $\bar{Q}$
0	0	0	0	1
0	0	1	1	0
0	1	1/0	0	1
1	0	1/0	1	0
1	1	1/0	0	0

S-R Latch

La configurazione di S=1 e R=1, viola la proprietà di complementarietà di Q e $\overline{Q}$, può portare ad una configurazione instabile.



Input S	Input R	Stato Interno Old Q	Output Q	Output $\overline{Q}$
0	0	0	0	1
0	0	1	1	0
0	1	1/0	0	1
1	0	1/0	1	0
1	1	1/0	0	0

S-R Latch

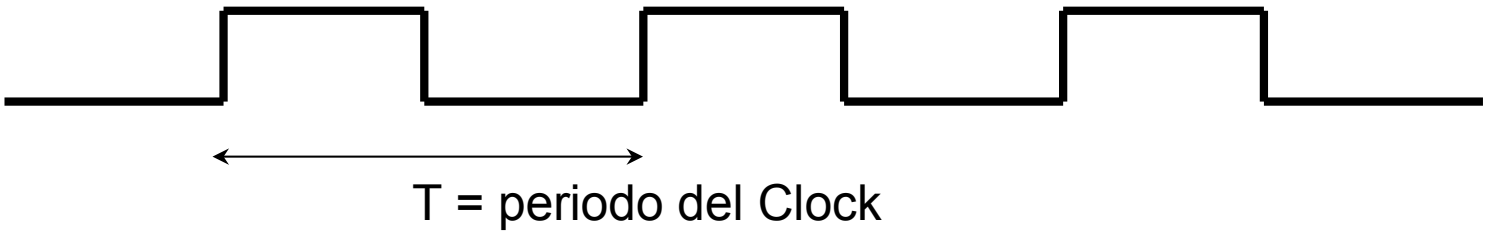
- In un SR Latch il valore di Q all'accensione dell'elaboratore è **indeterminato**, a meno che non ci sia un meccanismo di inizializzazione.
  - All'accensione, i valori interni dei nodi dipendono da condizioni casuali, come la carica residua nei transistor o nei condensatori.
  - Se non viene forzato un valore iniziale, il latch potrebbe assumere un qualsiasi stato in modo casuale.
    - **Reset Hardware**: Un segnale di reset globale può forzare Q=0 e NotQ=1 subito dopo l'accensione.
    - **SR con Priorità di Reset**: Se il circuito include un reset attivo all'avvio, Q sarà forzato a un valore noto.

S-R Latch

- (S,R) sono di solito calcolati da un circuito combinatorio
  - l'output del circuito diventa stabile dopo un certo intervallo di tempo che dipende dal numero di porte attraversate
  - bisogna evitare che durante questo intervallo, gli output intermedi del circuito vengano memorizzati

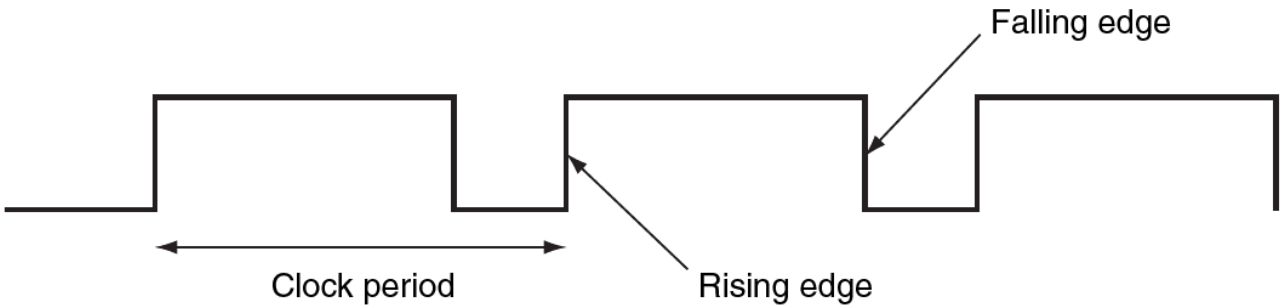
Latch e clock

- Soluzione: clock
- Usiamo un segnale a gradino, il cui periodo viene scelto abbastanza grande da assicurare la stabilità degli output
- 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



Clock

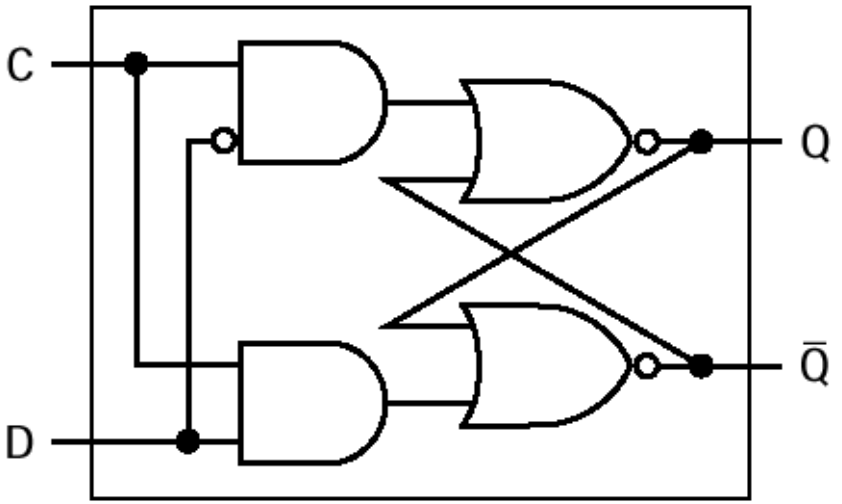
- Il segnale di **clock** è fondamentale per le reti sequanziali che sono caratterizzate da uno **stato**
- Clock – definito come un segnale (onda quadra) con un periodo predeterminato e costante



- Caratterizzato da un periodo T (oppure ciclo) di clock e dalla frequenza F(definita come 1/periodo) e misurata in Hertz F= 1/T

D Latch

- Latch sincronizzato con il clock
- Il clock garantisce che il latch cambi stato solo in certi momenti



D=1 corrisponde al *setting*

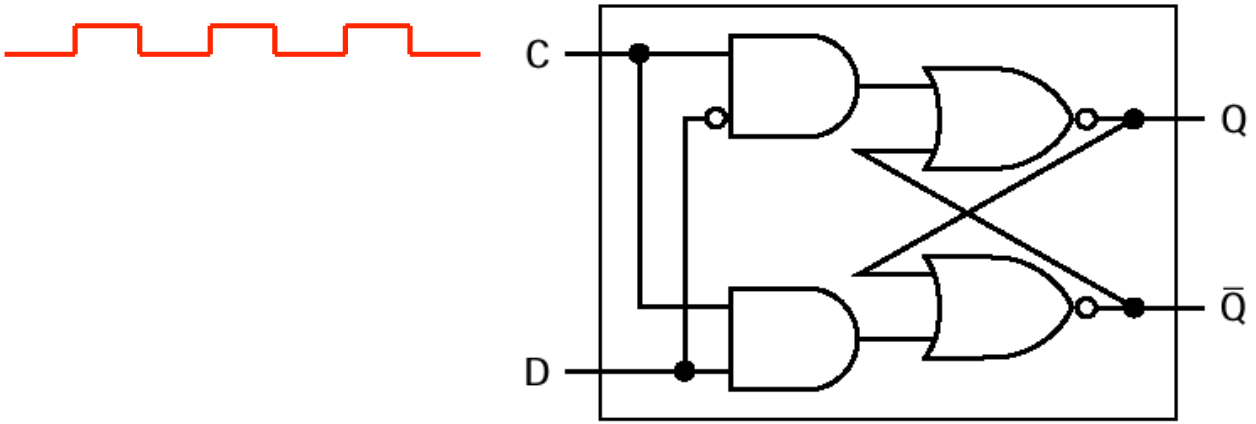
- S=1 e R=0

D=0 corrisponde al *resetting*

- S=0 e R=1

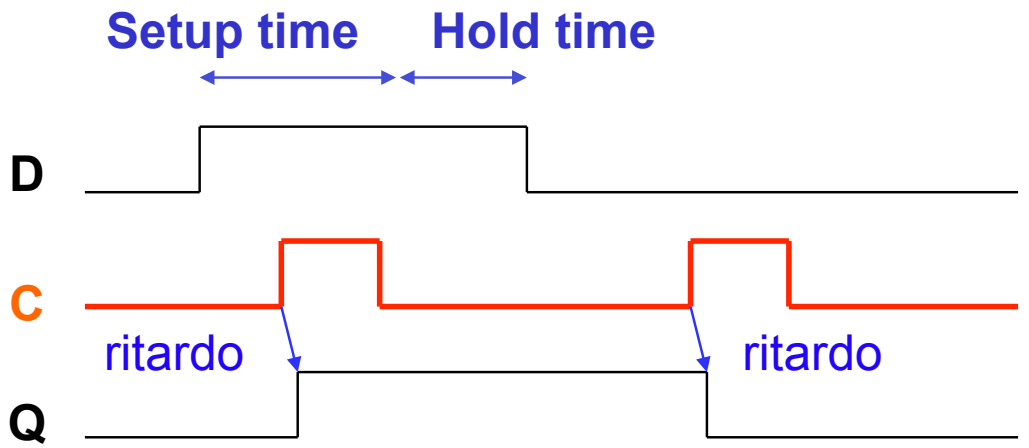
D Latch

- Quando il **clock è deasserted** non viene memorizzato nessuna valore:
  - S=0 e R=0 (viene mantenuto il valore precedentemente memorizzato)
- Quando il **clock è asserted** viene memorizzato un valore (in funzione del valore di D)



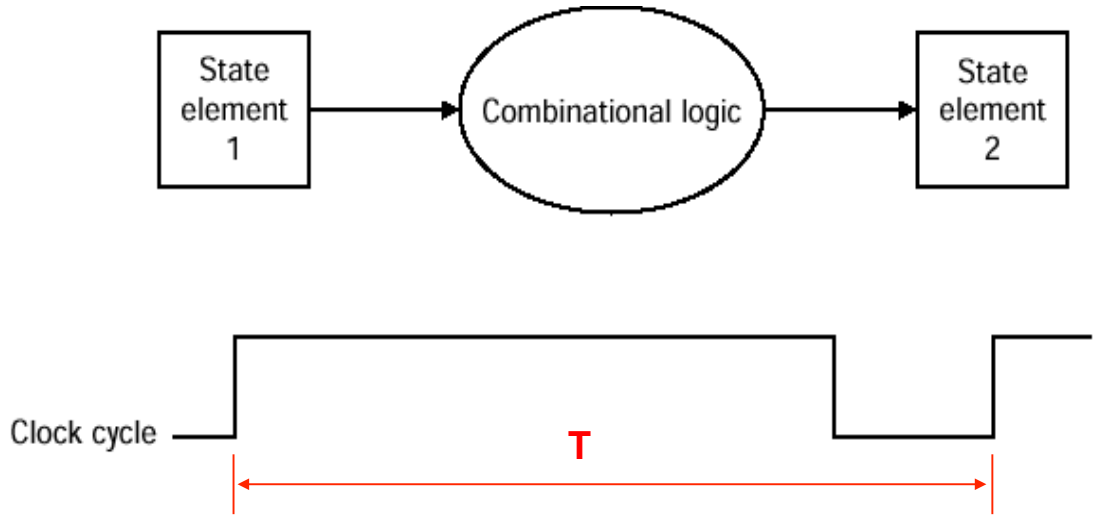
D Latch

- Il segnale D, ottenuto come output di un circuito combinatorio deve:
  - essere già **stabile** quando C diventa *asserted*
  - rimanere **stabile** per tutta la durata del livello alto di C (Setup time)
  - rimanere **stabile** per un altro periodo di tempo per evitare malfunzionamenti (Hold time)



D Latch

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



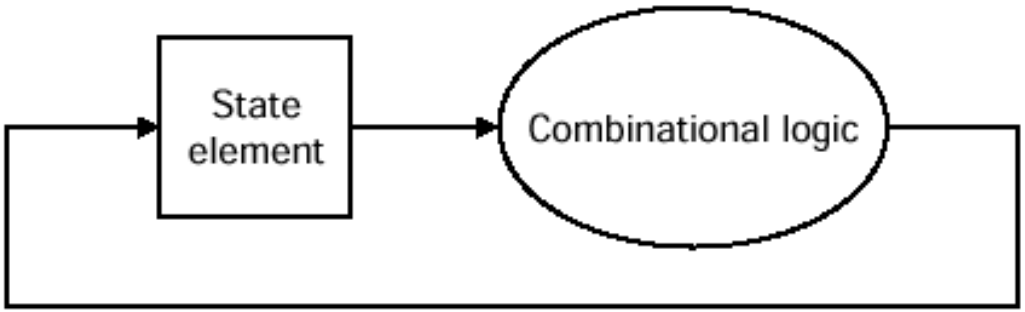
D Latch

- Il D-latch è caratterizzato dal seguente comportamento:
  - durante l'intervallo alto del clock il valore del segnale di ingresso D viene memorizzato nel latch
  - il valore di D si propaga quasi immediatamente all'uscita Q
  - anche le eventuali variazioni di D si propagano quasi immediatamente, con il risultato che Q può variare più volte durante l'intervallo alto del clock
  - solo quando il clock torna a zero Q si stabilizza
  - durante l'intervallo basso del clock il latch non memorizza

trasparenza del latch

D Latch – input/output

- Supponiamo che l'elemento di memoria debba essere usato sia come input che come output durante lo stesso ciclo di clock. Funzionerebbe il D Latch?



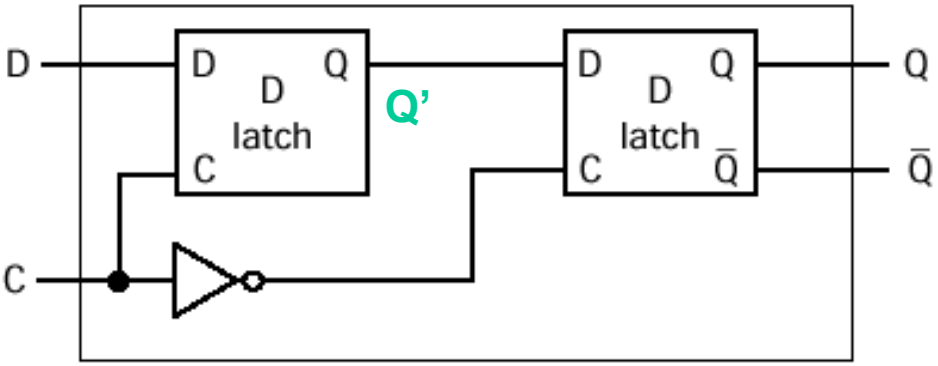
Metodologie di timing

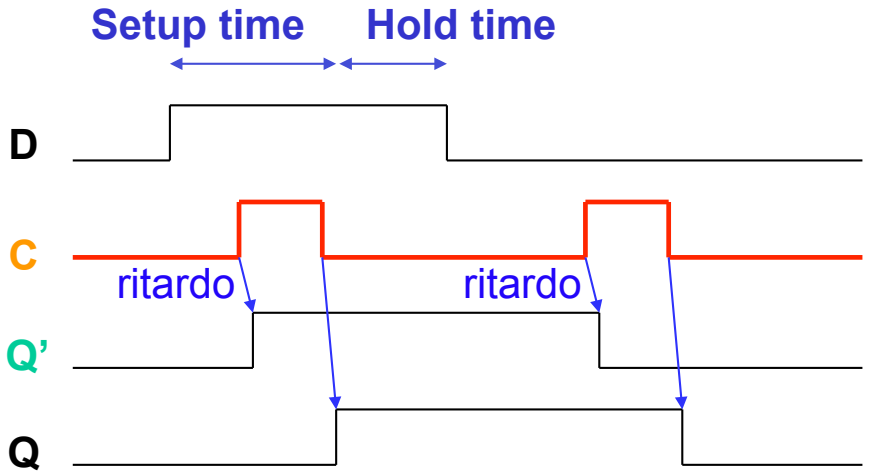
- Fino ad ora abbiamo visto una metodologia di timing detta **level-triggered** che avviene **sul livello alto** (o basso) del clock
- Esiste una metodologia di timing chiamata **edge-triggered** che avviene **sul fronte di salita** (o di discesa) del clock
  - la memorizzazione avviene istantaneamente
  - l'eventuale segnale di ritorno «sporco» non fa in tempo ad arrivare a causa dell'istantaneità della memorizzazione

flip-flop

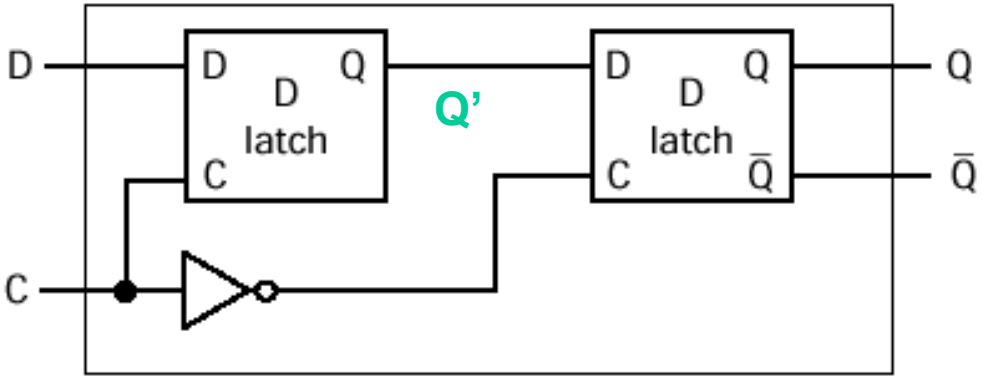
D Flip-Flop

- Il D Flip-flop è usabile come input e output durante lo stesso ciclo di clock
- Realizzato ponendo in serie 2 D-latch: il primo viene detto **master** e il secondo **slave**



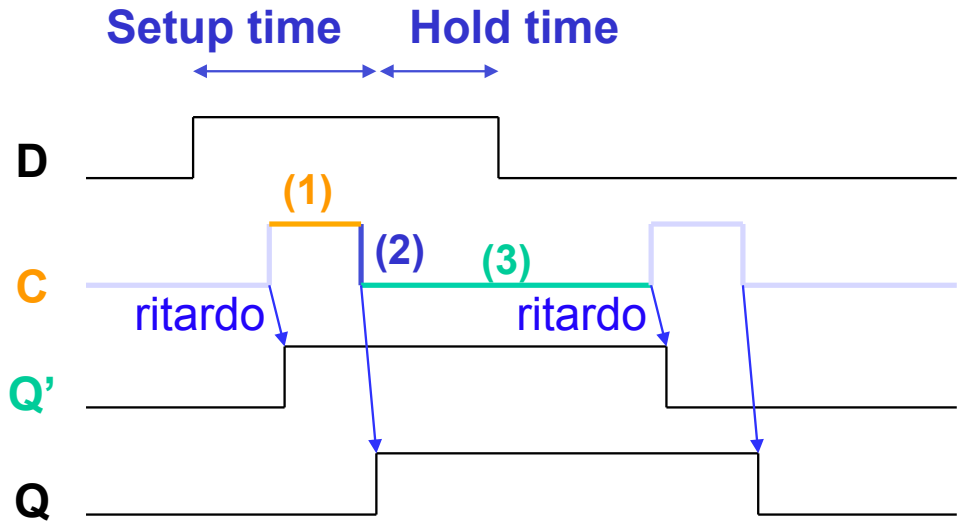


D Flip-Flop



(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

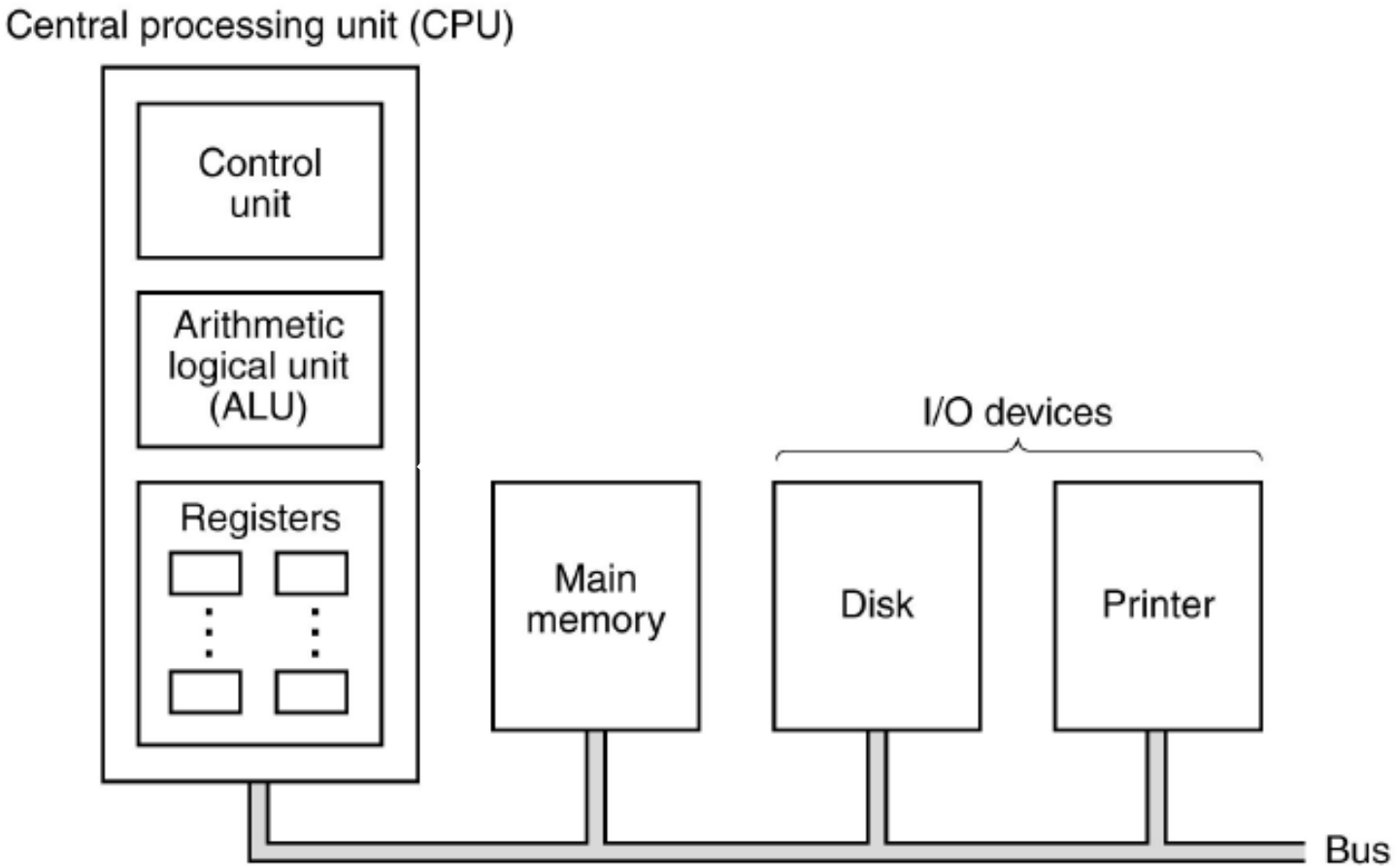


(2) Il segnale del clock scende, e in questo istante il secondo latch viene aperto per memorizzare il valore di Q'

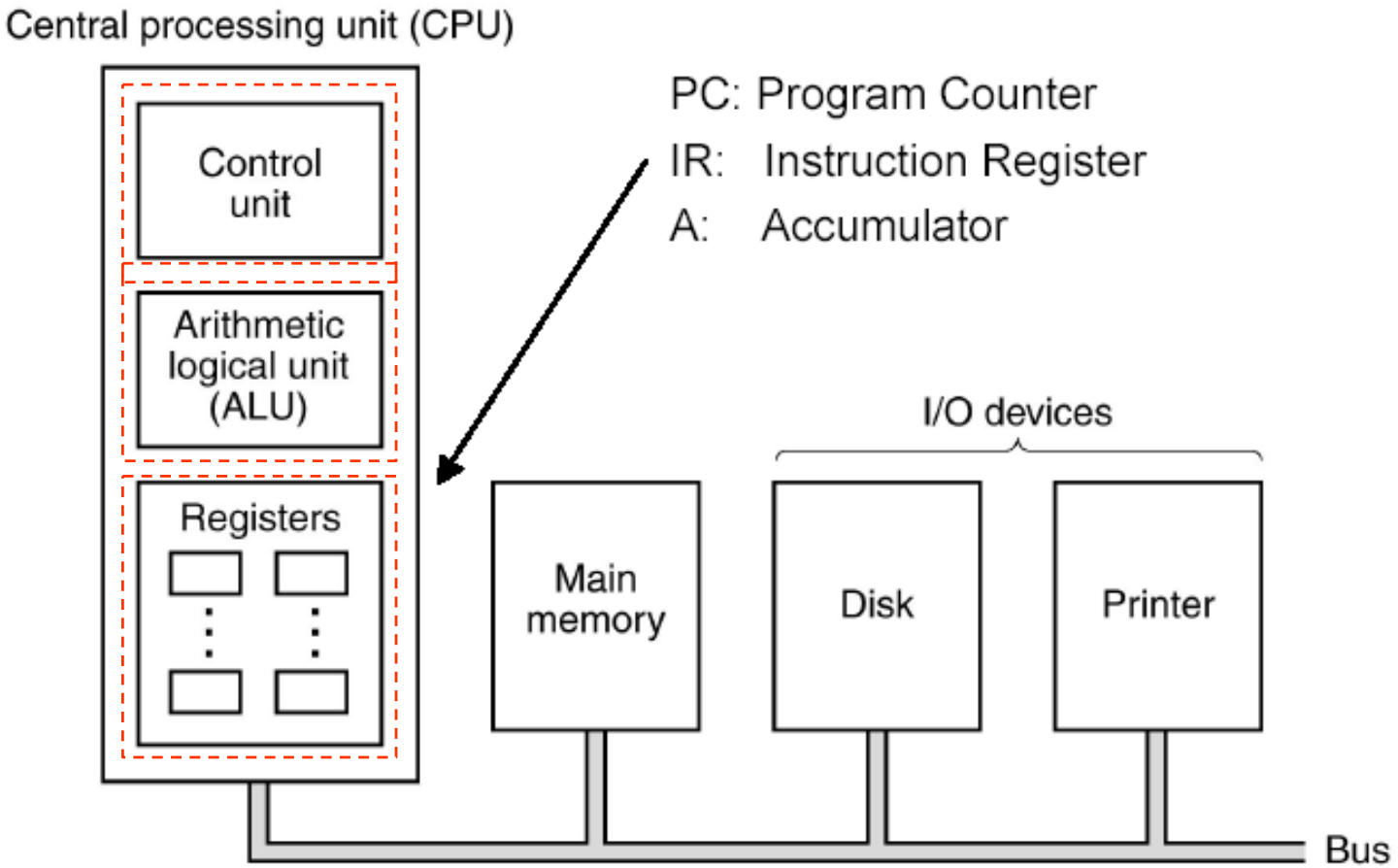
(3) Il secondo latch è aperto, memorizza D (Q'), e fa fluire il nuovo valore Q nel circuito a valle. Il primo latch è invece chiuso, e non memorizza niente

Da circuiti sequenziali a register file

Organizzazione di un calcolatore: richiami



Organizzazione di un calcolatore: richiami



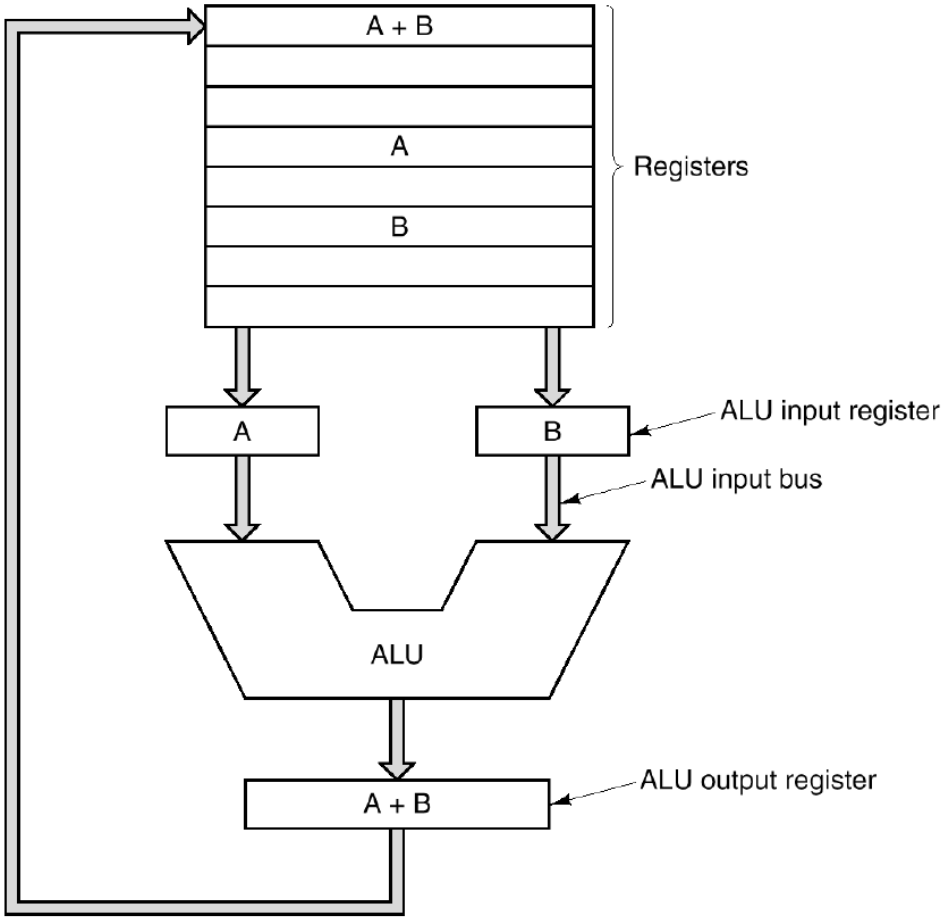
Register File

Il **Register File** è una memoria veloce all'interno della CPU che contiene **registri** utilizzati per memorizzare dati temporanei durante l'esecuzione delle istruzioni.

Ogni **registro** è costituito da:

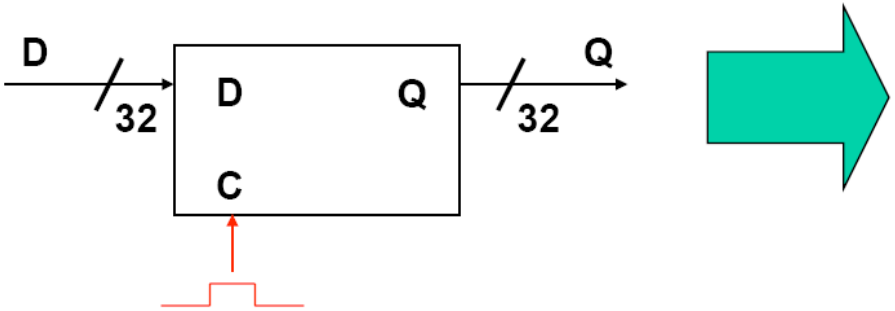
1. **Flip-Flop D (D Flip-Flop)** → Per memorizzare ogni singolo bit.
2. **Multiplexer (MUX)** → Per selezionare quale registro leggere o scrivere.
3. **Decoder** → Per indirizzare correttamente i registri.
4. **Circuiti di controllo** → Per gestire lettura/scrittura sincronizzate con il clock.

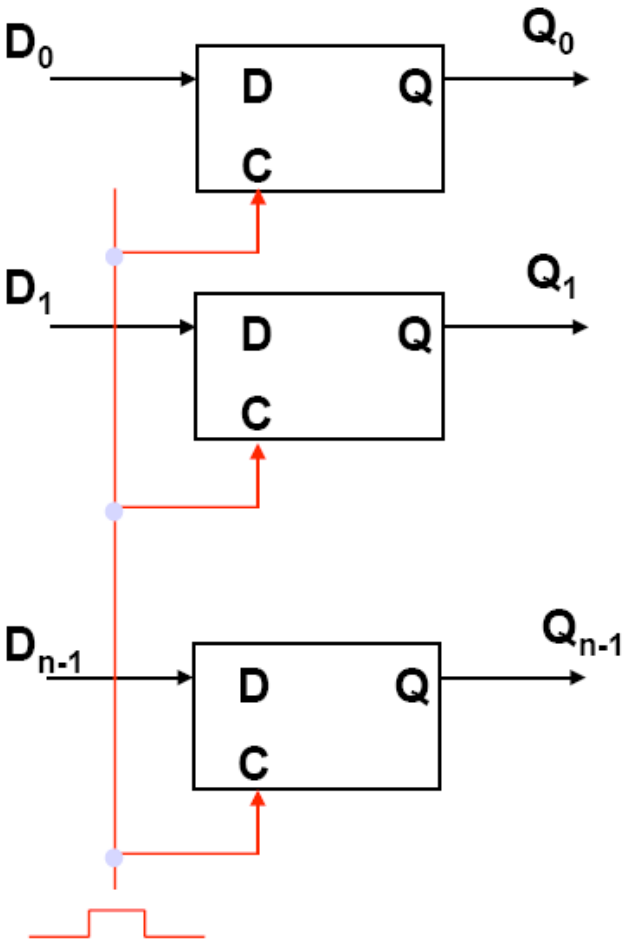
Datapath e Register File



- Un **registro** è costituito da **n flip-flop**
  - Nel MIPS ogni registro è di 1 word = 4 byte = 32 bit
- I registri sono organizzati in un **Register File**
  - Il Register File del MIPS ha 32 registri (32 x 32 = 1024 flip-flop)
  - Il Register File permette la lettura di 2 registri e la scrittura di 1 registro

Datapath e Register File





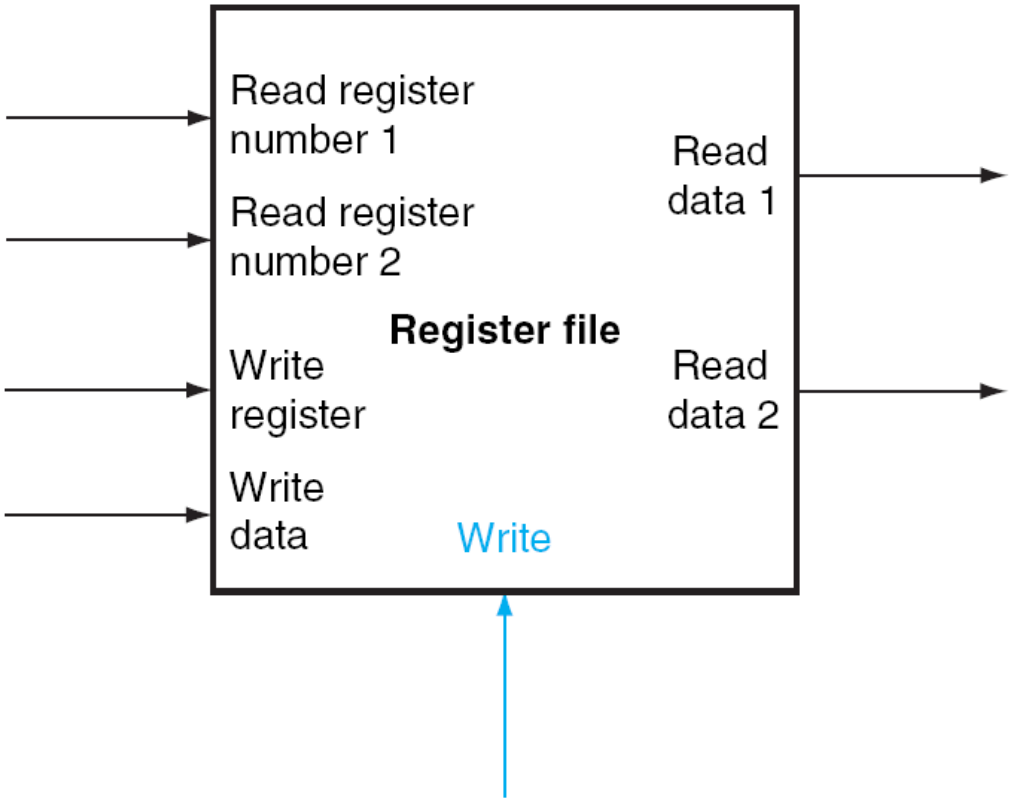
- Nel Datapath della CPU il clock non entra direttamente nei vari flip-flop; viene messo in AND con un segnale di controllo “Write”
- Il segnale Write determina se, in corrispondenza del fronte di discesa del clock, il valore D debba (o meno) essere memorizzato nel registro

Register File

Il register file è utilizzato in ogni ciclo istruzione:

1. **Fetch**: L'istruzione viene letta dalla memoria.
2. **Decode**: Si identificano gli operandi nei registri (es. somma tra due valori).
3. **Execute**: L'ALU esegue l'operazione.
4. **Write**: Il risultato viene scritto nel registro di destinazione (se necessario).

Register File



Register File

Read Reg1 # (5 bit)

- numero del 1º registro da leggere

Read Reg2 # (5 bit)

- numero del 2º registro da leggere

Read data 1 (32 bit)

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

Read data 2 (32 bit)

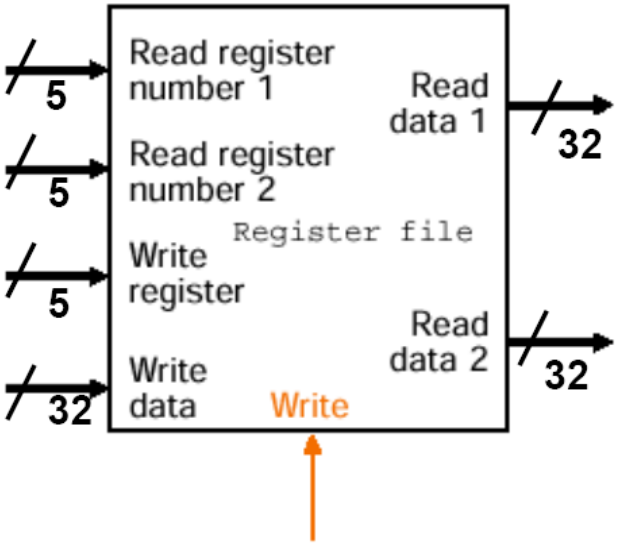
- valore del 2º registro, letto sulla base di Read Reg2 #

Write Reg # (5 bit)

- numero del registro da scrivere

Write data (32 bit)

- valore da scrivere nel registro Write Reg #

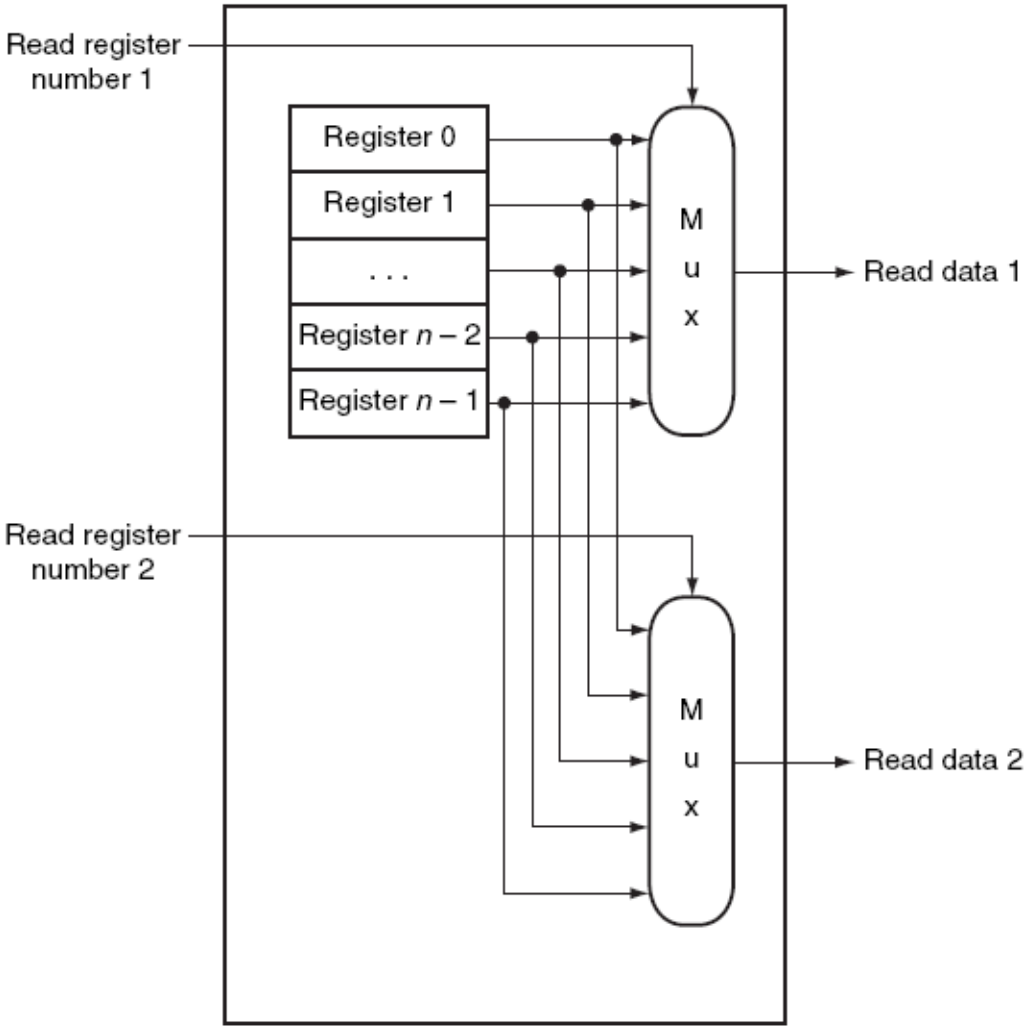


Write

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

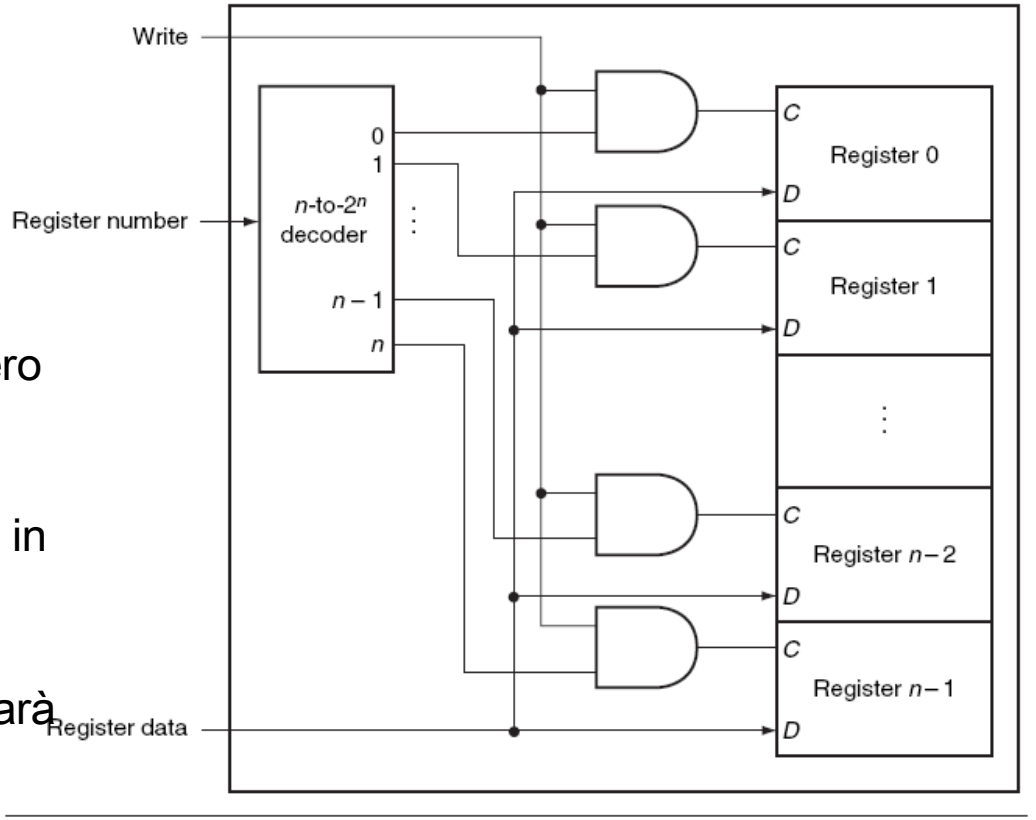
Lettura dal Register File

- Utilizza 2 segnali che indicano i registri da leggere (Read Reg1, Read Reg2)
- Utlizza 2 multiplexer: ognuno con 32 ingressi e un segnale di controllo
- Il register file fornisce *sempre* in output una coppia di registri



Scrittura nel Register File

- Utilizza 3 segnali:
  - Il registro da scrivere (Register Number)
  - Il valore da scrivere (Register Data)
  - Il segnale di controllo (Write)
- Utilizza un decoder che decodifica il numero del registro da scrivere (Write Register)
- Il segnale Write (già in AND con il clock) è in AND con l'output del decoder
- Se Write non è affermato nessun valore sarà scritto nel registro

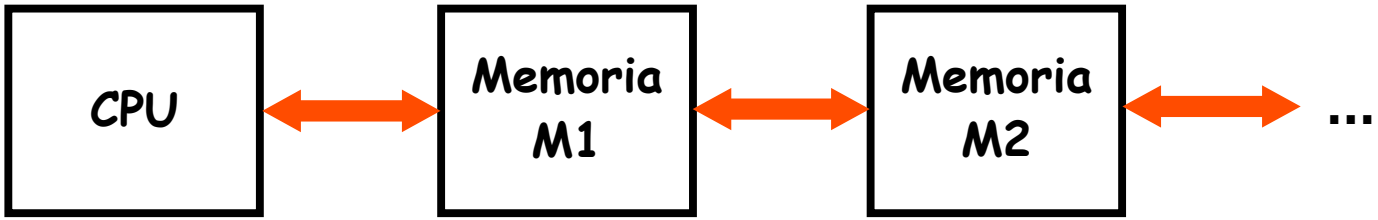


Memoria

- Oltre alle piccole memorie implementate per mezzo di registri e file di registro, esistono altri tipi di memorie che possiamo distinguere in base a diversi **parametri**:
  1. **Dimensione**: quantità di dati memorizzabili
  2. **Velocità**: l'intervallo di tempo tra la richiesta del dato e il momento in cui è disponibile
  3. **Consumo**: potenza assorbita
  4. **Costo**: costo per bit
- Idealmente un calcolatore dovrebbe avere quanta **più memoria** possibile, ad alta velocità, basso consumo e minimo costo.

Gerarchie di memoria

- Non è possibile avere un'unica memoria con tutte le caratteristiche ideali.
- Possiamo però organizzare in una sorta di gerarchia:
  - memorie piccole, più veloci (e costose) sono poste ai livelli alti
  - le memorie ampie, più lente (e meno costose) sono poste ai livelli più bassi.

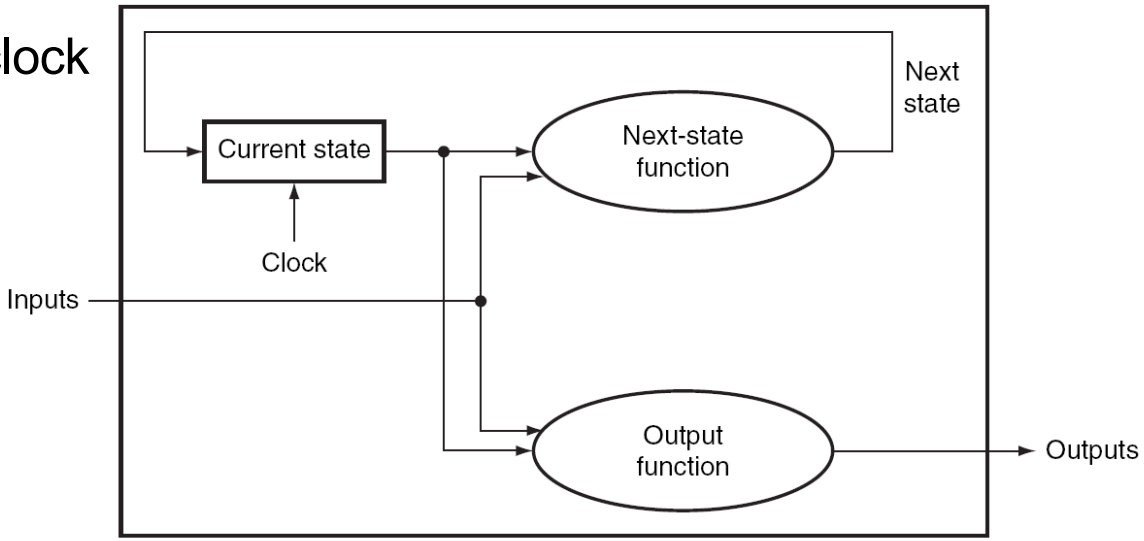


Memorie

- **IN AUTONOMIA**
  - RAM
  - SRAM
  - DRAM
  - SSRAM
  - SDRAM

Macchine a stati finiti

- Finite State Machine (FSM): usate per descrivere i circuiti sequenziali
- Composte da un set di stati e 2 funzioni:
  - **Next state function** – determina lo stato successivo partendo dallo stato corrente e dai valori in ingresso
  - **Output function** – produce un insieme di risultati partendo dallo stato corrente e dai valori in ingresso
- Sono sincronizzate con il clock



FSM – Moore vs Mealy

- FSM – next – state dipende sia dagli input che dallo stato corrente
- FSM – output:
  - Se dipende solo dallo stato corrente: Moore – usato come controller
  - Se dipende dallo stato corrente e dagli input: Mealy
  - Moore and Mealy sono equivalenti e si possono trasformare autonamticamente tra di loro
  - In questo corso usiamo FSM Moore
- Esempio: semaforo – con rosso e verde

FSM – Semaforo

- Esempio: semaforo – con rosso e verde
- Segnali di input: NScar & EWcar
- Segnali di output: NSlite & EWlite
- Possibili stati: NSgreen, EWgreen
- Lo stato cambia quando arrivano macchine al semaforo

FSM – Semaforo

	Inputs		Next state
	NScar	EWcar	
NSgreen	0	0	NSgreen
NSgreen	0	1	EWgreen
NSgreen	1	0	NSgreen
NSgreen	1	1	EWgreen
EWgreen	0	0	EWgreen
EWgreen	0	1	EWgreen
EWgreen	1	0	NSgreen
EWgreen	1	1	NSgreen

	Outputs	
	NSlite	EWlite
NSgreen	1	0
EWgreen	0	1

FSM – Semaforo