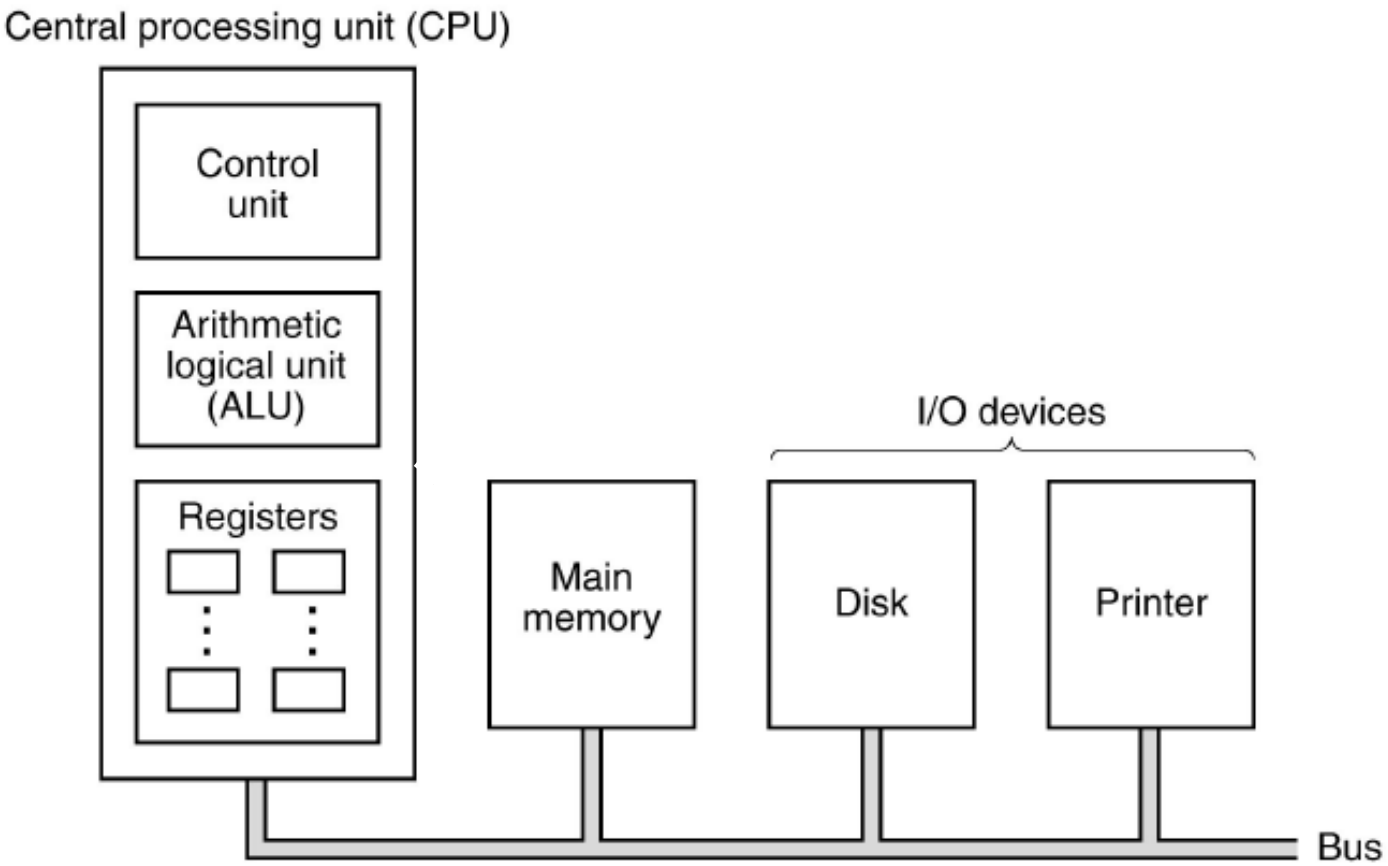
Architettura degli Elaboratori 2021-2022

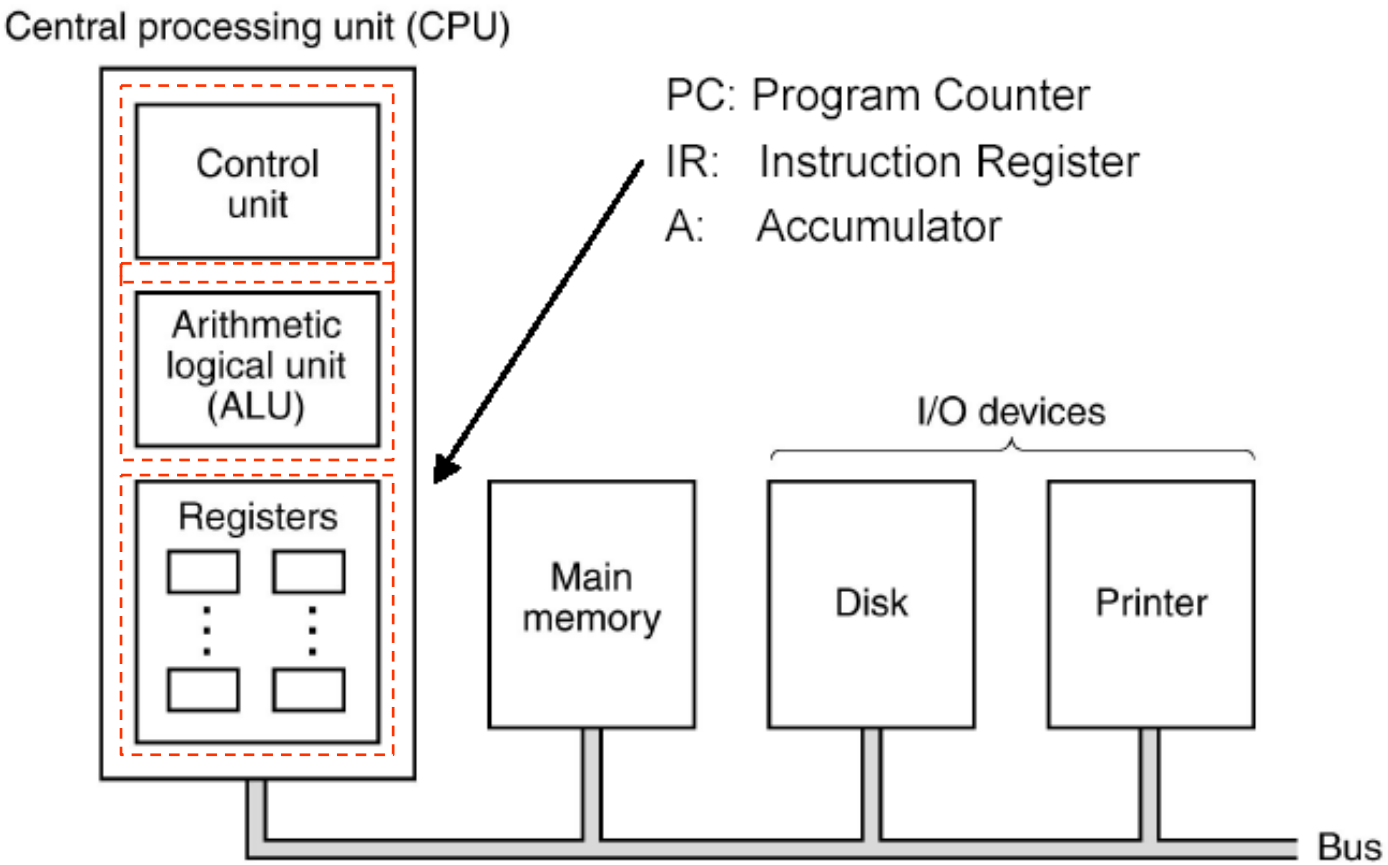
Circuiti Sequenziali

Prof. Elisabetta Fersini
elisabetta.fersini@unimib.it

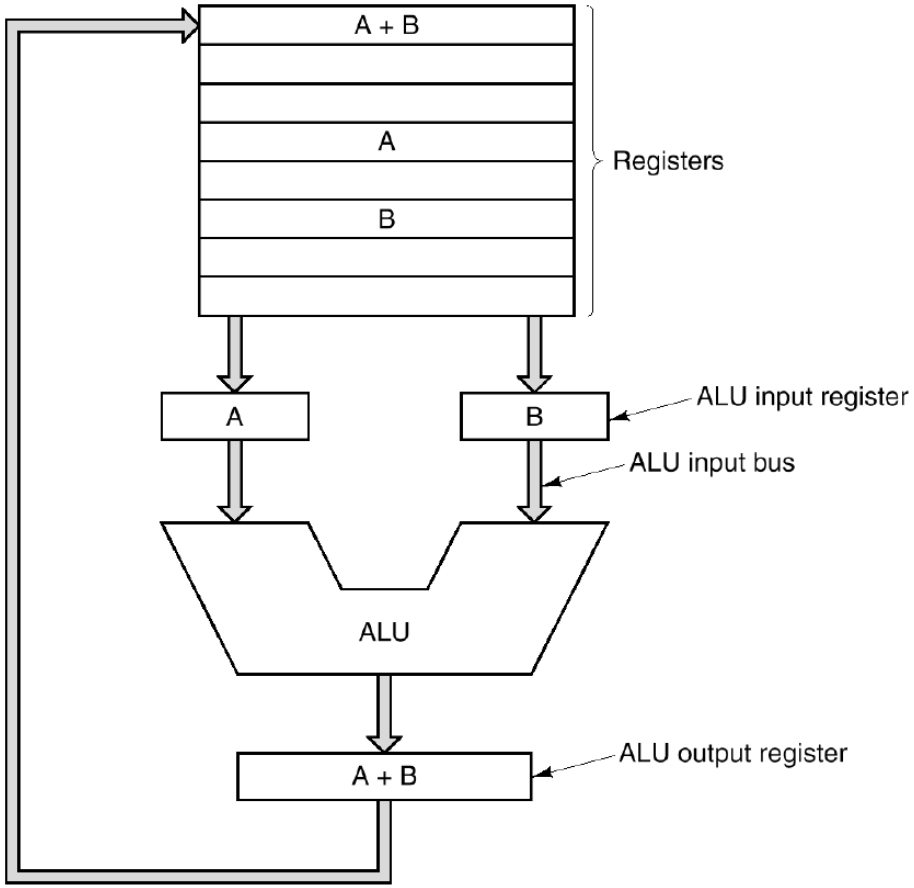
Organizzazione di un calcolatore: richiami



Organizzazione di un calcolatore: richiami



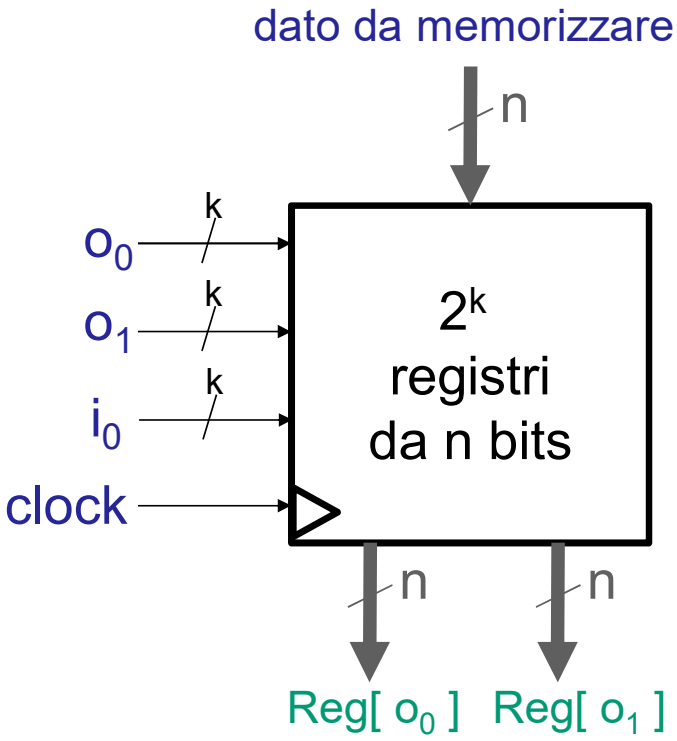
Datapath e Register File



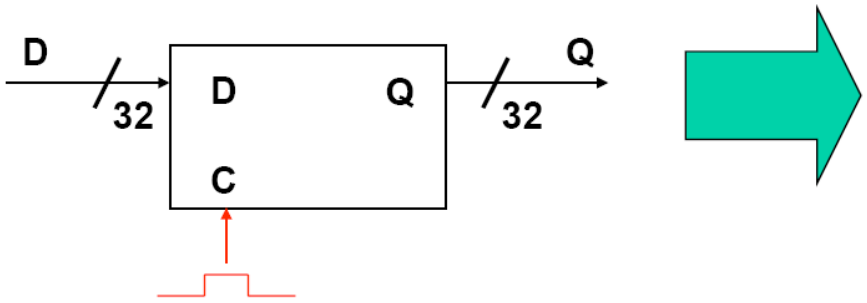
- Un **registro** è costituito da **n flip-flop**
  - Nel MIPS ogni registro è di 1 word = 4 byte = 32 bit
- I registri sono organizzati in un **Register File**
  - Il Register File del MIPS ha 32 registri (32 x 32 = 1024 flip-flop)
  - Il Register File permette la lettura di 2 registri e la scrittura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

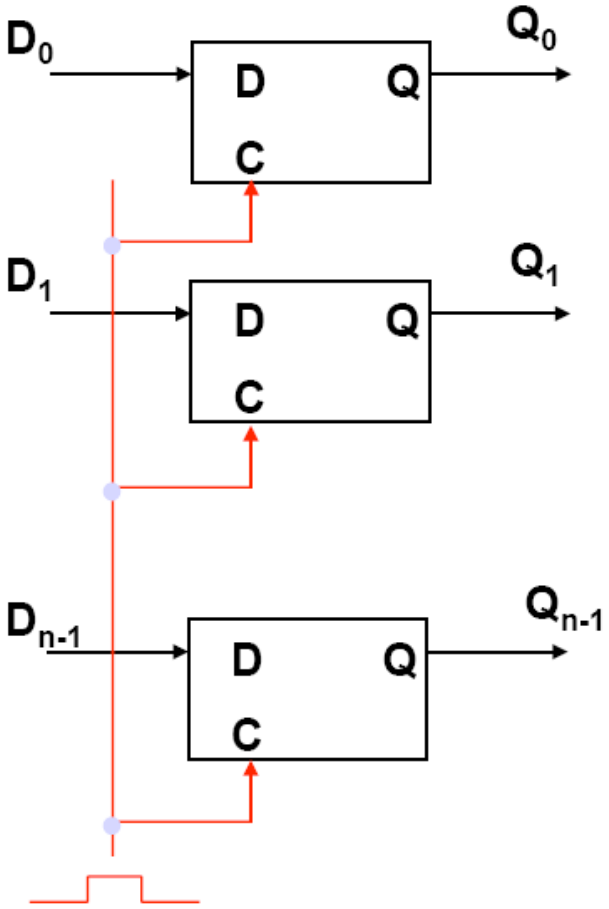
Register File - funzionamento

- Register File con 2 uscite e 1 entrata
  - 2^k registri da n bit ciascuno
- Output: valore attuale di due dei registri
- Input: la parola che verrà memorizzata e clock



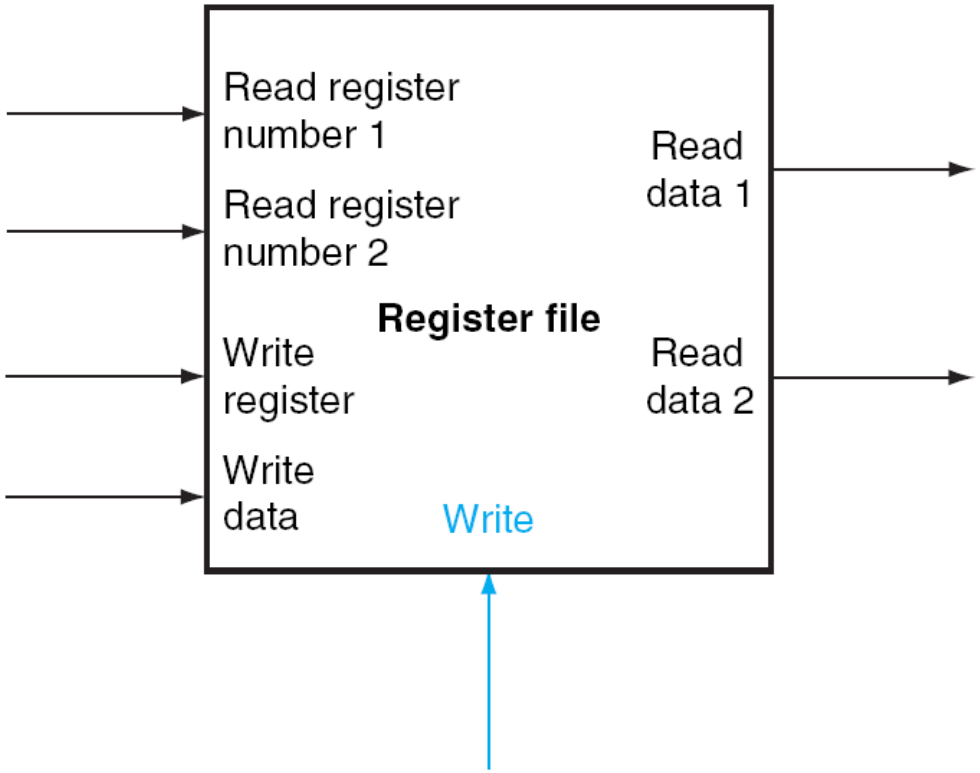
Datapath e Register File





- 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



Register File

Read Reg1 # (5 bit)

- numero del 1° registro da leggere

Read Reg2 # (5 bit)

- numero del 2° registro da leggere

Read data 1 (32 bit)

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

Read data 2 (32 bit)

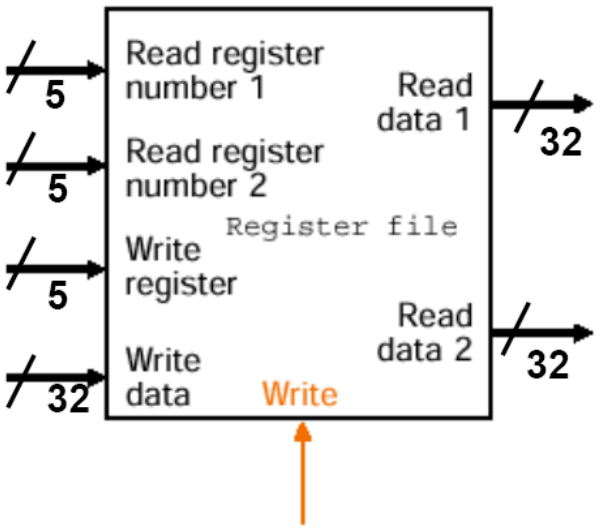
- valore del 2° registro, letto sulla base di Read Reg2 #

Write Reg # (5 bit)

- numero del registro da scrivere

Write data (32 bit)

- valore da scrivere nel registro Write Reg #

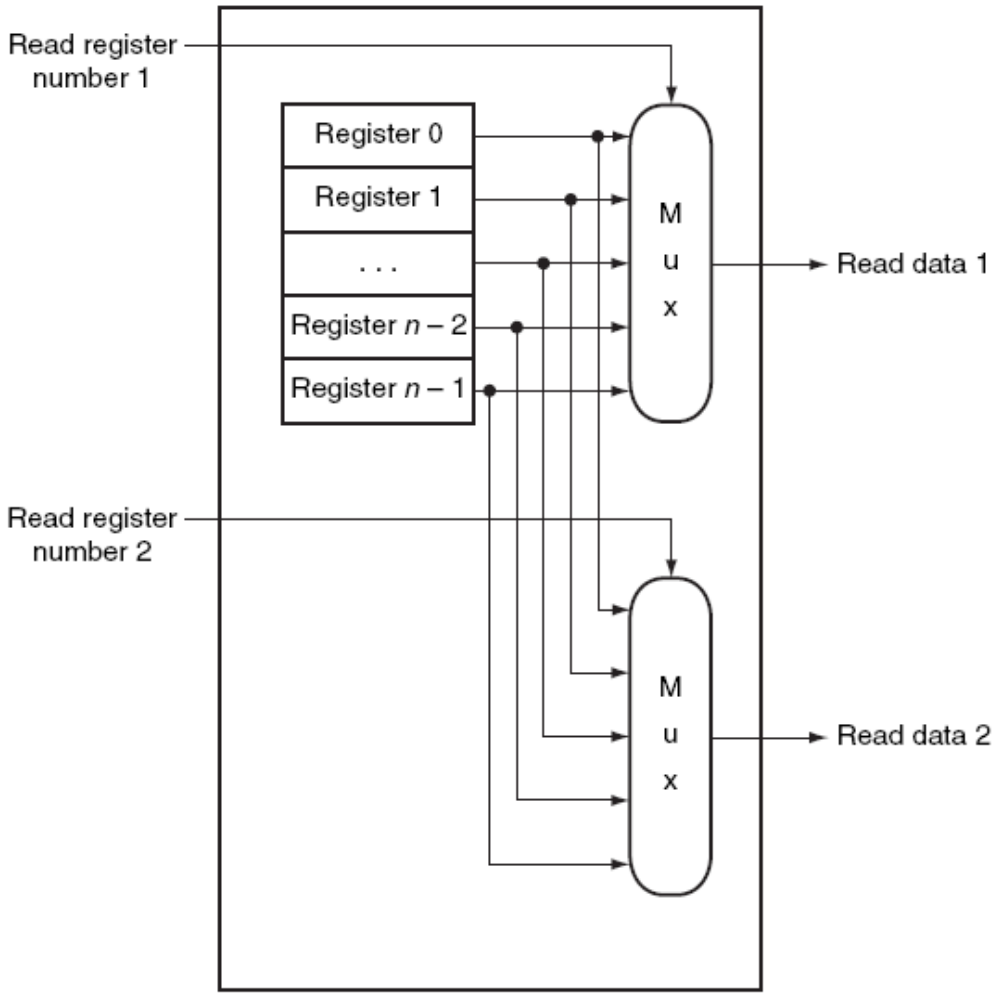


Write

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

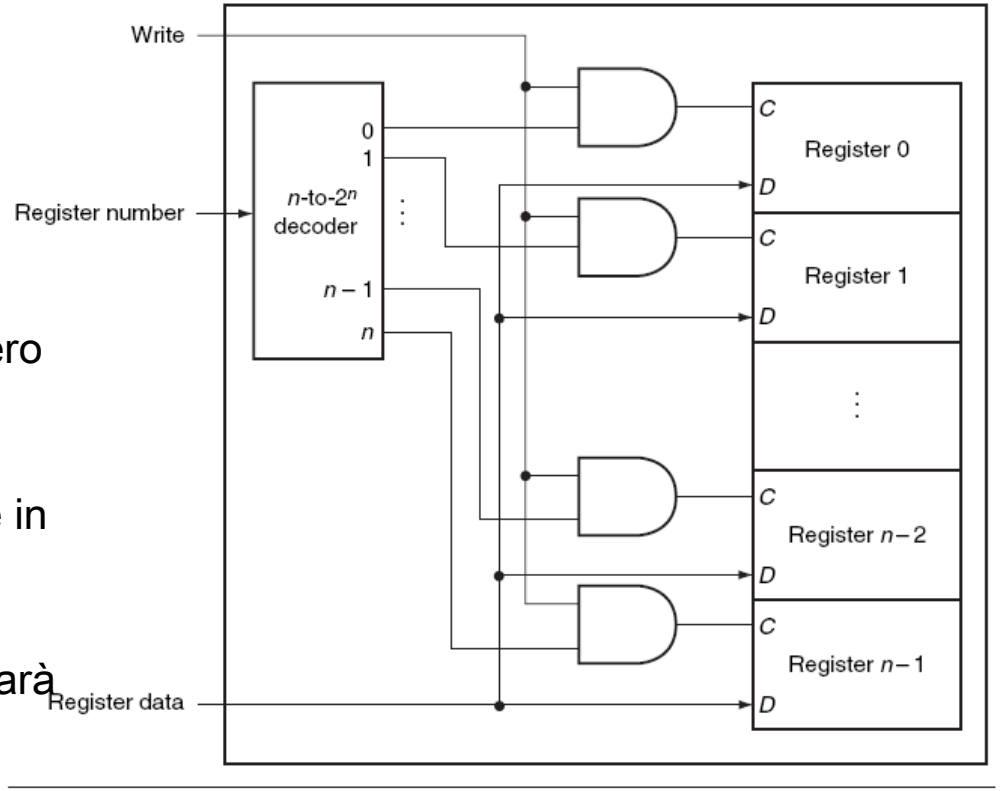
Lettura dal Register File

- 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 da 5 bit
- 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



Register File: alcune note

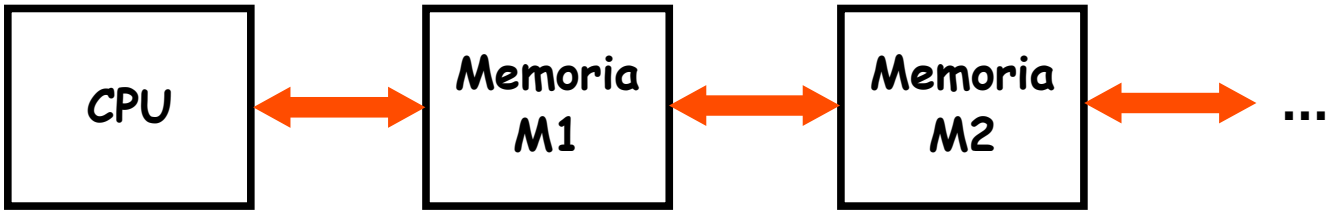
- Ogni registro può contenere valori di qualsiasi tipo purchè usino il numero di bit a disposizione nel registro
- Spetterà a chi definisce il programma in binario usare i dati memorizzati nei registri in modo coerente con il loro tipo
- Il ciclo di clock deve essere sufficientemente lungo da permettere al segnale di propagarsi dall'output del Register File al suo input (compreso il computo della ALU)

Memoria

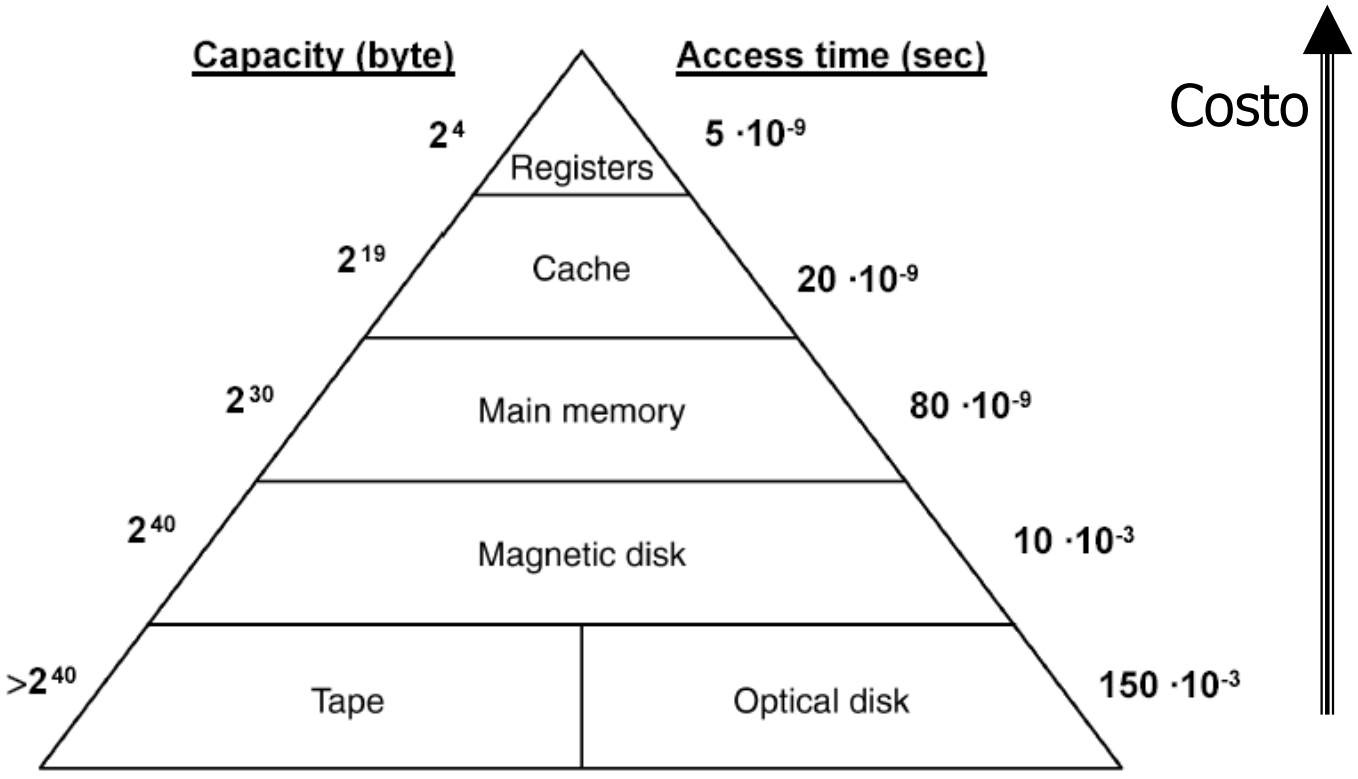
- Oltre alle piccole memorie implementate per mezzo di registri e file di registro, esistono altri tipi di memorie che possiamo distinguere in base a diversi **parametri**:
  1. **Dimensione**: quantità di dati memorizzabili
  2. **Velocità**: l'intervallo di tempo tra la richiesta del dato e il momento 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.

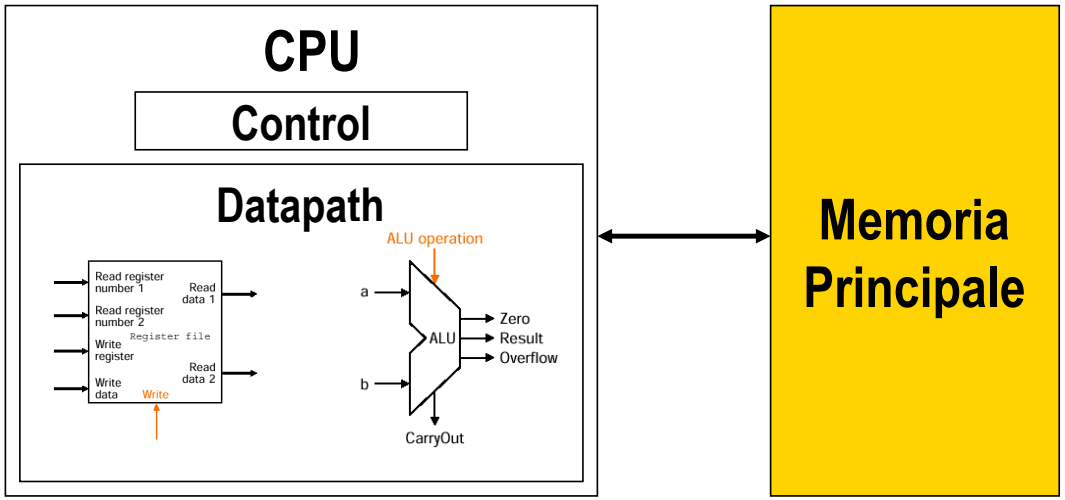


Gerarchie di memoria



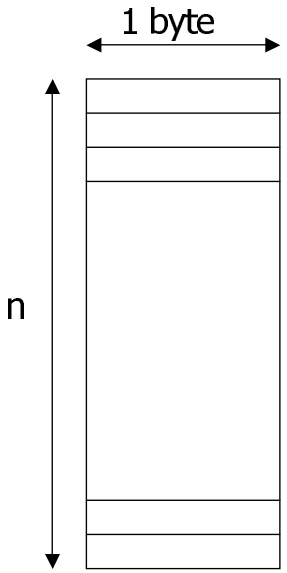
Memoria Principale

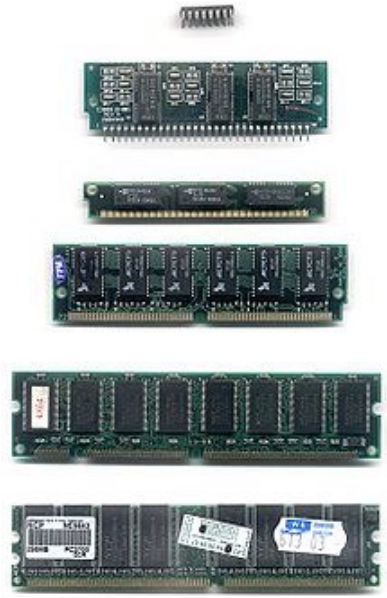
- La dimensione del Register File è piccola (singole variabili di tipo semplice)
- Per memorizzare dati strutturati e codice di programma, abbiamo bisogno di memorie più grandi
- **Memoria principale** (RAM - Random Access Memory)
  - meno veloce della memoria dei registri, ma molto più capiente



RAM – nozioni di base

- RAM = Insieme di celle (1byte) ognuna individuata da un indirizzo
- Operazioni: lettura e scrittura
- Indirizzamento: attività con cui l'elaboratore seleziona una particolare cella di memoria
  - Registro Indirizzi: con n bit si indirizzano 2^n celle di memoria

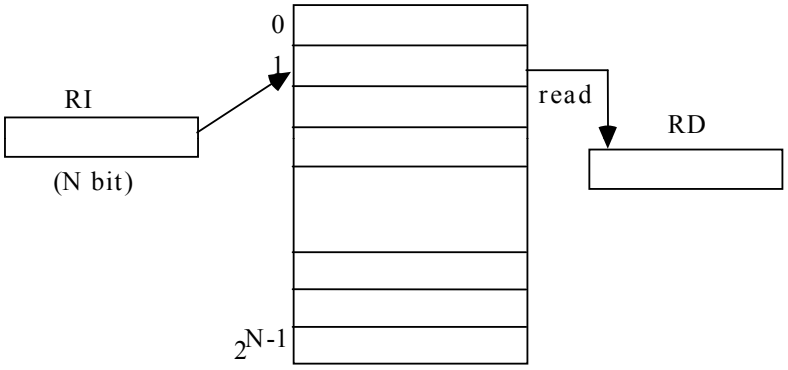


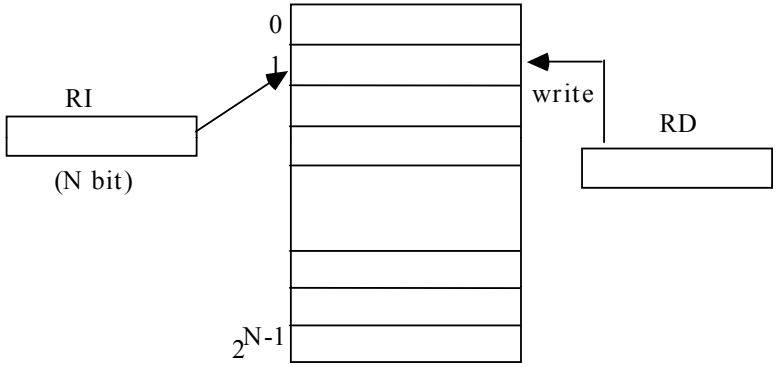


RAM: lettura e scrittura

Lettura (Read): il contenuto della cella di memoria indirizzata dal Registro Indirizzi è copiato nel Registro Dati

Scrittura (Write): il contenuto del Registro Dati è copiato nella cella di memoria indirizzata dal Registro Indirizzi



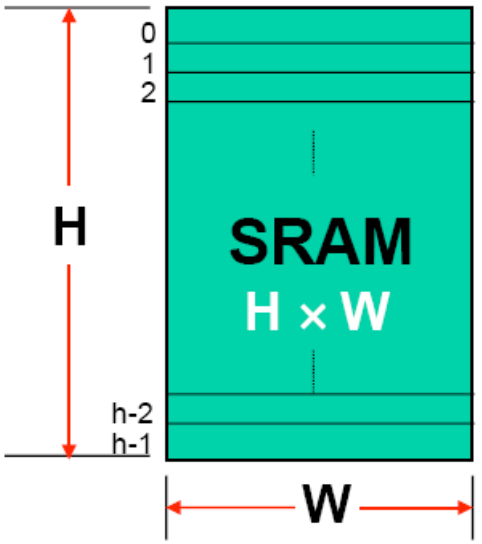


SRAM e DRAM

- La SRAM (Static RAM) è più veloce
  - per la sua realizzazione vengono usati dei latch
  - tempi di accesso intorno a 0,5 – 2,5 ns
  - SRAM a basso consumo (5-10 volte più lente)
- La DRAM (Dynamic RAM) è più capiente ma più lenta
  - ogni bit è memorizzato tramite un condensatore
  - tempi di accesso intorno a 50-70 ns
  - è necessario “rinfrescare” il contenuto delle DRAM a intervalli di tempo prefissati

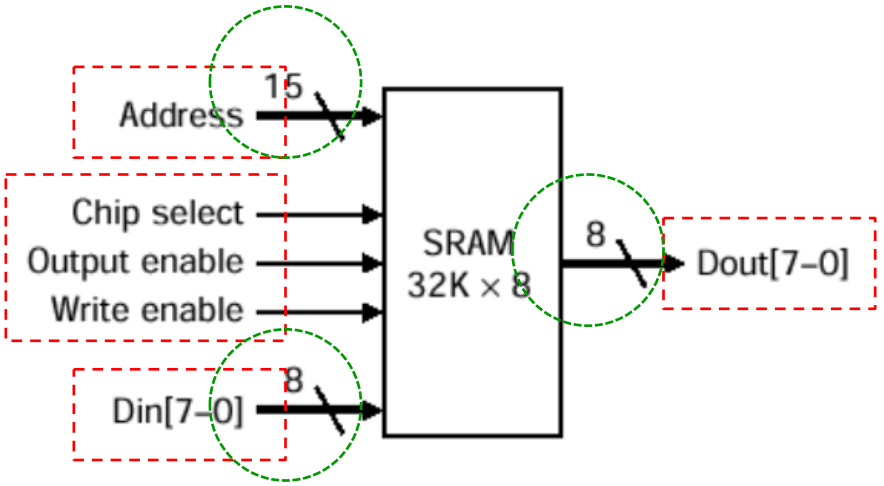
SRAM

- Realizzata come **matrice di latch** H x W:
  - Larghezza o ampiezza W (numero di latch per ogni cella)
  - Altezza H (numero di celle indirizzabili)
  - Singolo indirizzo usato sia per lettura che per scrittura
  - Non è possibile leggere e scrivere contemporaneamente (come per Register File)



SRAM

- Din e Dout
- Address
- Segnali di controllo:
  - Chip select: deve essere *asserito* per poter leggere o scrivere
  - Output enable: deve essere *asserito* per poter leggere
  - Write enable: deve essere *asserito* per poter scrivere



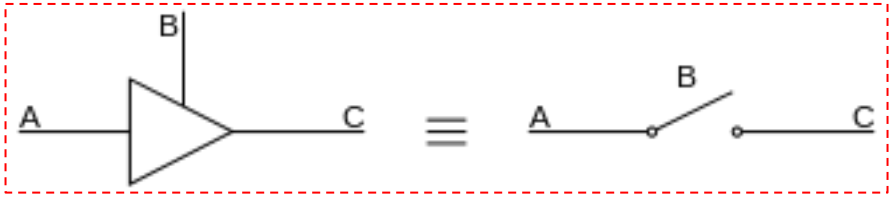
Esempio:

- 32K x 8 (32K celle da 8 bit =>256Kb)
- 15 linee di indirizzo (2^15=32K)
- 8 linee in output

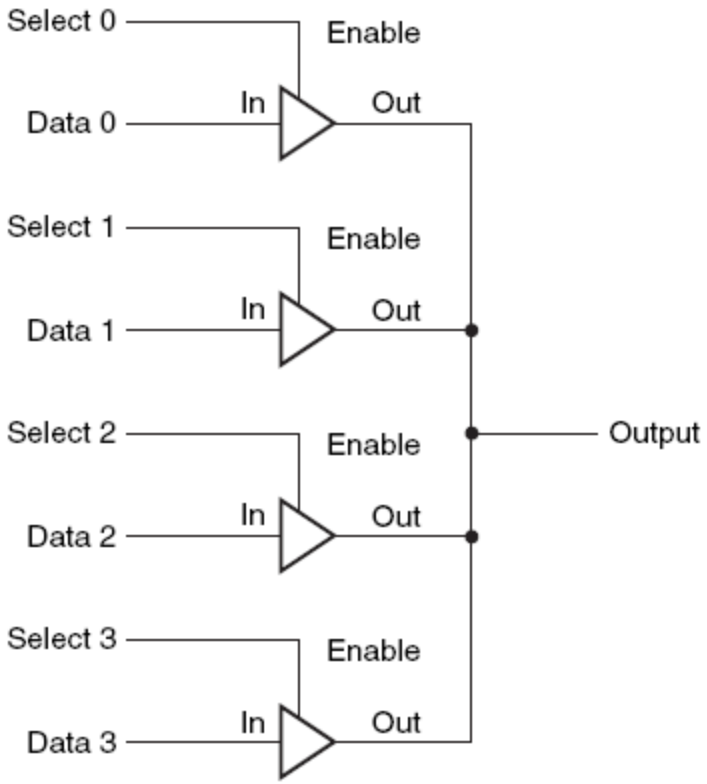
Realizzazione SRAM

- Tecniche realizzative diverse rispetto a quelle del register file:
  - il register file impiega decoder (per selezionare il registro da scrivere) e multiplexer (per selezionale il registro da leggere)
  - con un numero elevato di celle di memoria avremmo bisogno di enormi decoder o multiplexer
    - Avremmo bisogno di porte AND con numero di input accettabili di una porta logica troppo elevato
    - Sarebbero necessari livelli multipli di porte AND, con conseguente introduzione di ritardi negli accessi alla memoria

Realizzazione SRAM

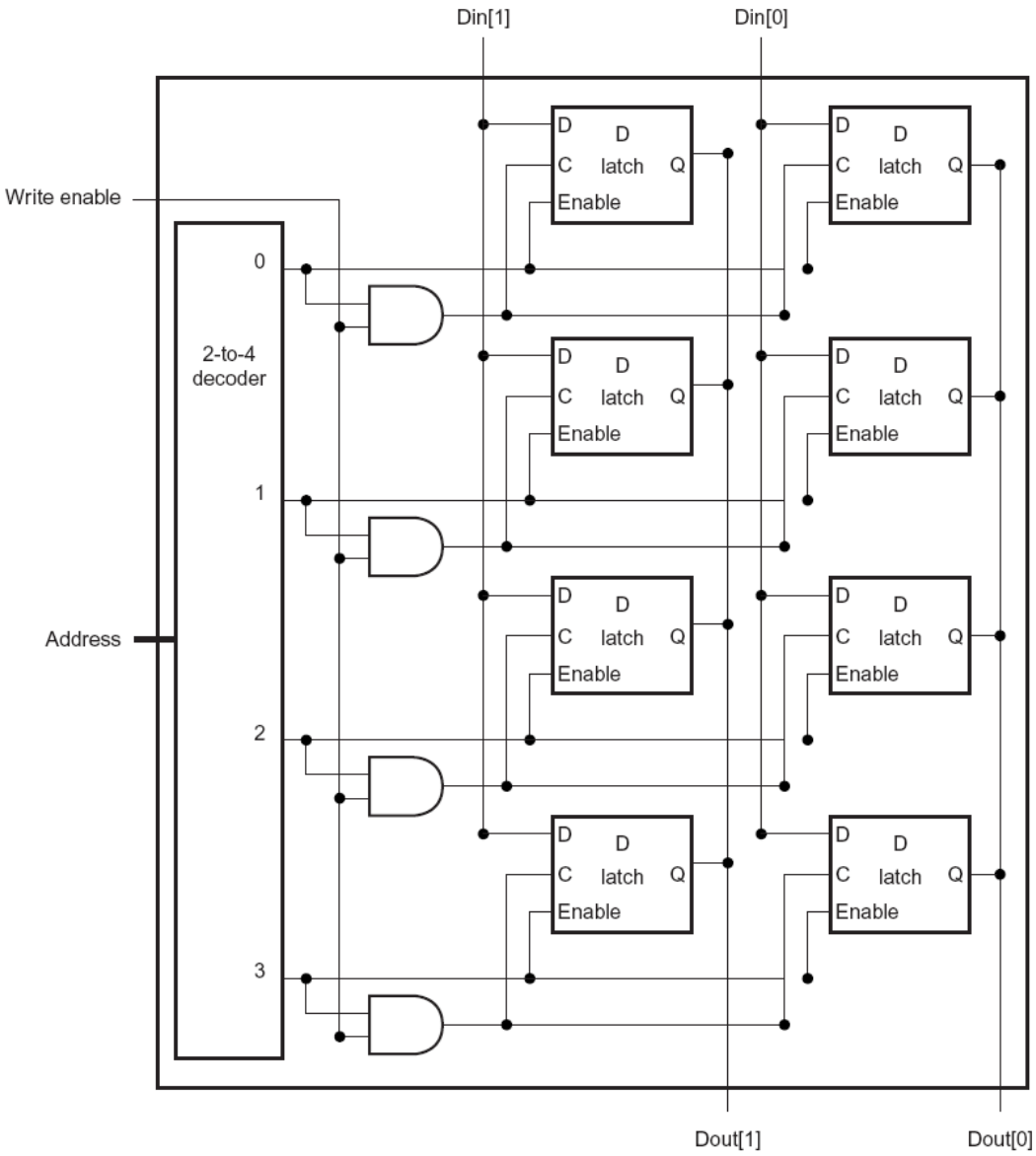


- Per evitare il multiplexer in uscita
  - possiamo usare una linea di bit condivisa su cui i vari elementi di memoria sono tutti collegati (OR)
  - Il collegamento alla linea condivisa avviene tramite **buffer a tre stati**, che aprono o chiudono i collegamenti (se il controllo è asserito o meno):
    - Il buffer ha due ingressi (dato e segnale di Enable) e una uscita:
      - l'uscita è uguale al dato (0 o 1) se Enable è asserito
      - l'uscita viene impedita se Enable non è asseritto



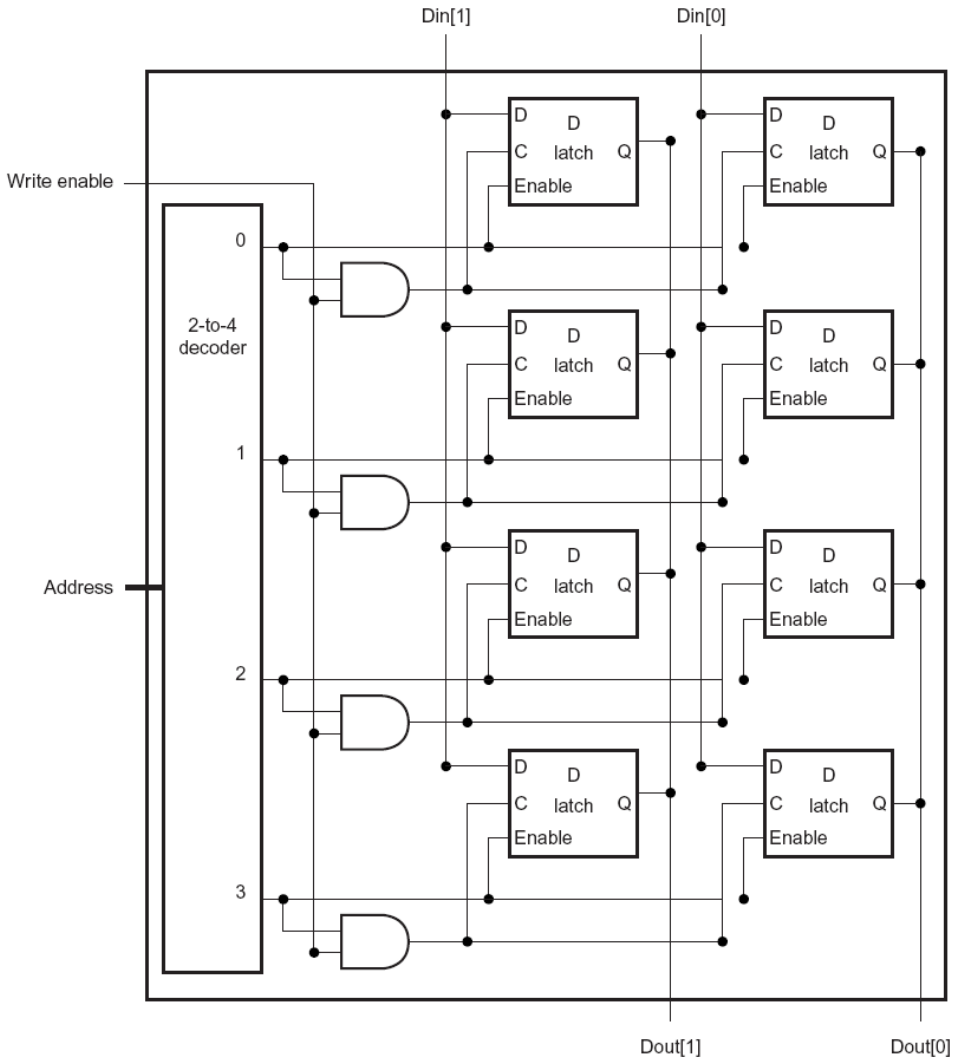
Realizzazione SRAM

- *Esempio*: SRAM 4 x 2
- I latch di una certa colonna sono collegati alla stessa linea in output (Dout[0] e Dout[1])
  - nell'esempio ogni D-latch ha un segnale di Enable che abilita il three-state buffer interno
- Il decoder serve ad abilitare in lettura/scrittura una certa linea di memoria
  - Entrambi i segnali Write enable e Enable vengono abilitati su una sola linea di memoria (2 bit)



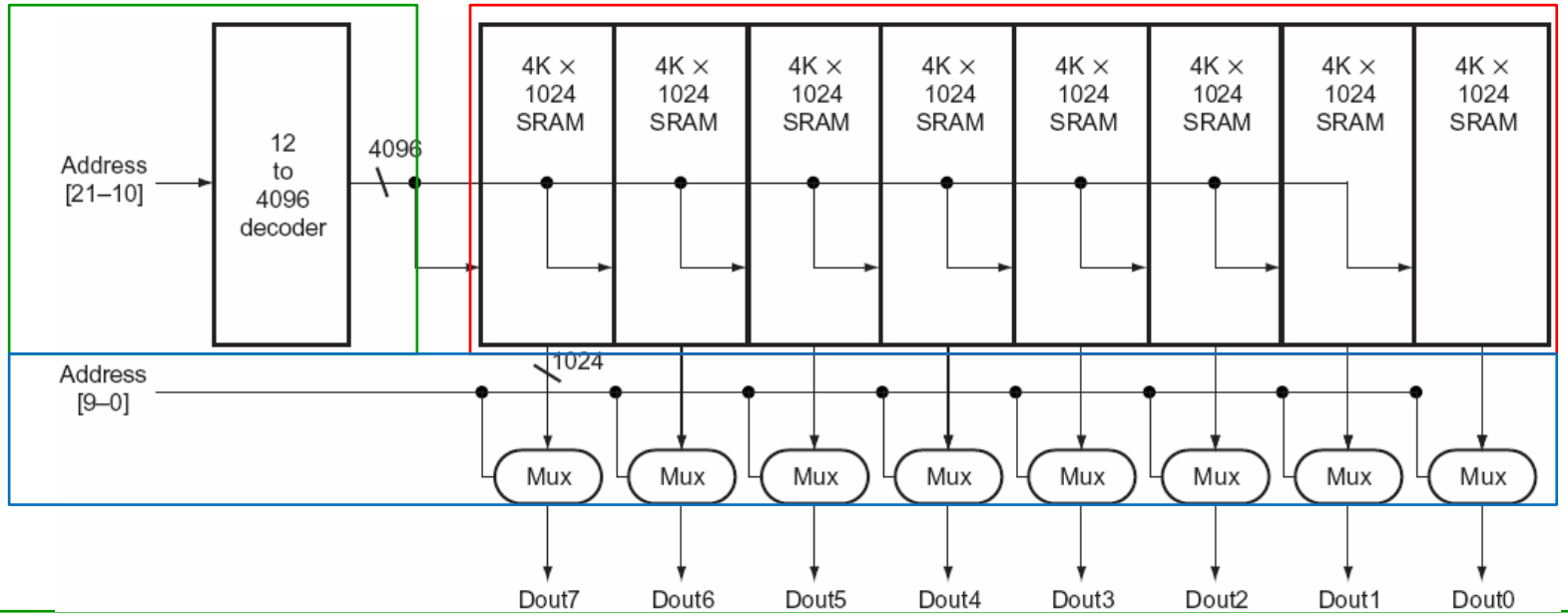
Realizzazione SRAM

- Chip Select e Output Enable sono stati omessi per semplicità ma possono essere aggiunti con qualche porta AND
- Solo se Chip Select è abilitato i segnali Write Enable e Output Enable sono significativi (servono porte AND aggiuntive)
- Solo se Output Enable è affermato, la coppia Dout[0-1] dovrebbe essere abilitata ad uscire

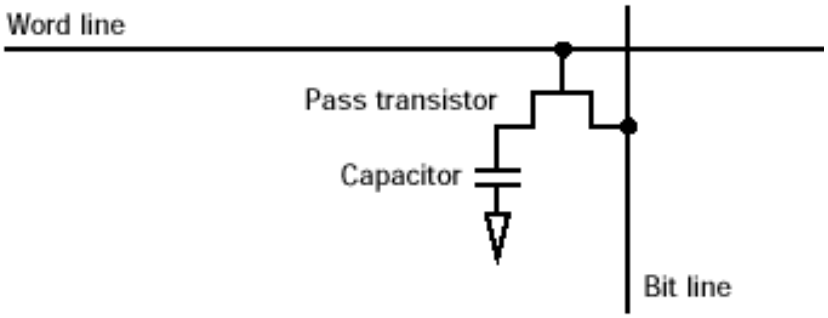


SRAM a due livelli

- Esempio di SRAM: 4M × 8 => servono 22 linee di indirizzo ($4M=2^{22}$)
  - suddiviso in 8 blocchi da 4Mb
  - parte alta dell'indirizzo [21-10] seleziona la medesima riga di ogni blocco attraverso un Decoder
  - parte bassa dell'indirizzo [9-0] seleziona singoli bit dei 1024 bit in output dai vari blocchi attraverso una serie di 8 Multiplexer



- Gli elementi di memoria di tipo DRAM sono meno costosi e più capienti rispetto al tipo SRAM, ma sono più lenti
  - DRAM sono da 5 a 10 volte meno veloci delle SRAM
- La DRAM è meno costosa, perché è realizzata con un solo transistor per bit, e un condensatore
  - il condensatore possiede la carica (0/1)
  - il transistor viene chiuso, trasferendo il potenziale elettrico del condensatore sulla Bit line (output), grazie al segnale affermato della Word line
  - la specifica Word line è attivata sulla base dell'indirizzo di memoria richiesto



SSRAM e SDRAM

- Per diminuire la complessità dei decoder è opportuno suddividere gli indirizzi in 2 blocchi
  - parte alta per accedere una riga
  - parte bassa per accedere una specifica colonna
- Nota che celle consecutive hanno indirizzi che solitamente differiscono solo per la parte bassa dell'indirizzo
  - quindi sono contenuti all'interno di una stessa riga selezionata con la parte alta dell'indirizzo
- Le **Synchronous SRAM e DRAM** (SSRAM e SDRAM) permettono di aumentare la banda di trasferimento della memoria sfruttando questa proprietà

SSRAM e SDRAM

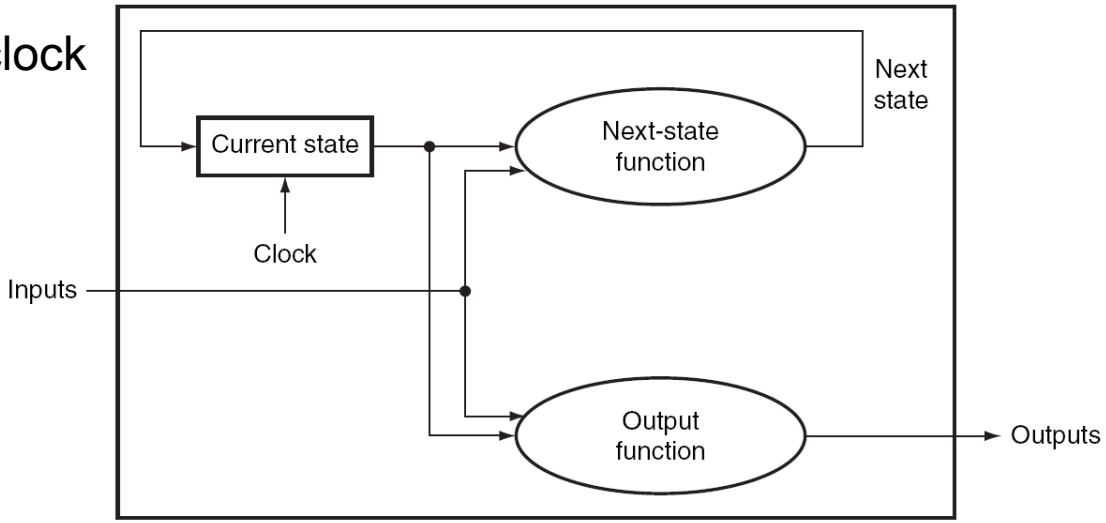
- Memorie sincrone con segnale di clock
- E' possibile specificare che vogliamo trasferire dalla memoria una una sequenza di celle consecutive (**burst**)
- Ogni burst specificato da un indirizzo di partenza, e da una lunghezza
- Le celle del burst sono contenute all'interno di una stessa Riga, selezionata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

⇒ Migliora banda di trasferimento (numero di trasf. al sec)

- Non è necessario ripresentare l'indirizzo per ottenere ogni cella del burst
- Costo del decoder pagato una sola volta all'inizio

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

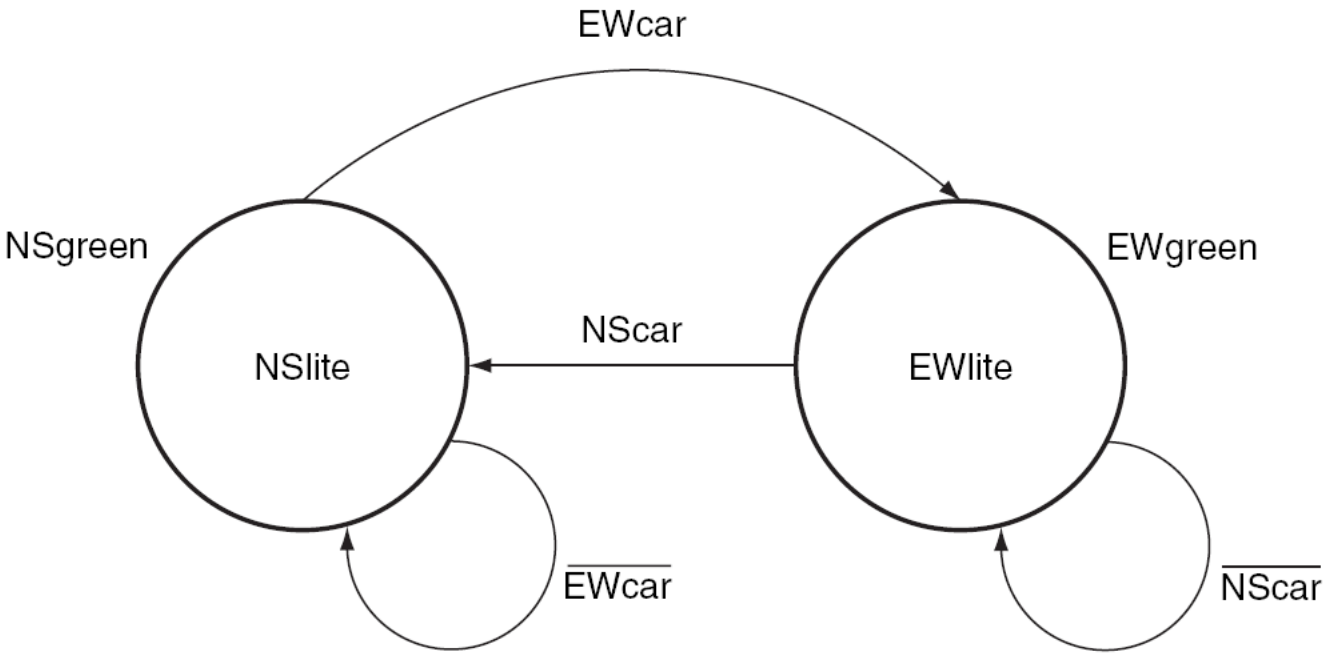
- Esempio: semaforo – con rosso e verde
- Segnali di input: NScar & EWcar
- Segnali di output: NSlite & EWlite
- Possibili stati: NSgreen, EWgreen
- Lo stato cambia quando arrivano macchine al semaforo

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



Da leggere

- "Basics of Logic Design", appendice C del testo 4th ed. (nel testo 5th ed. questo argomento è trattato nella appendice B)
  pag. C-55 – pag. C-70