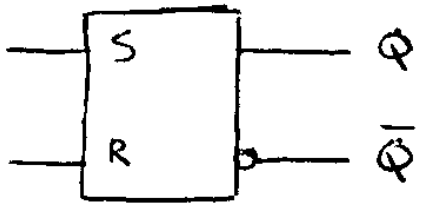
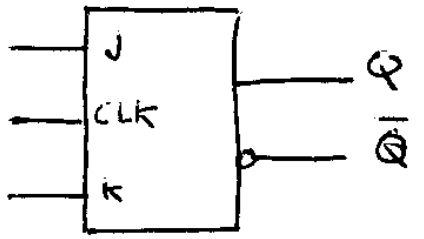
- FLIP-FLOP SR



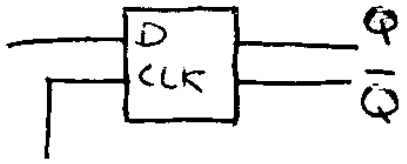
S	R	Q
0	0	MEMORIA
1	0	1
0	1	0
1	1	NO

- FLIP-FLOP JK



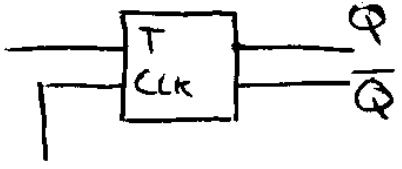
J	K	Q
0	0	MEMORIA
1	0	1
0	1	0
1	1	$\bar{Q}$

- FLIP-FLOP D



D	Q
0	0
1	1

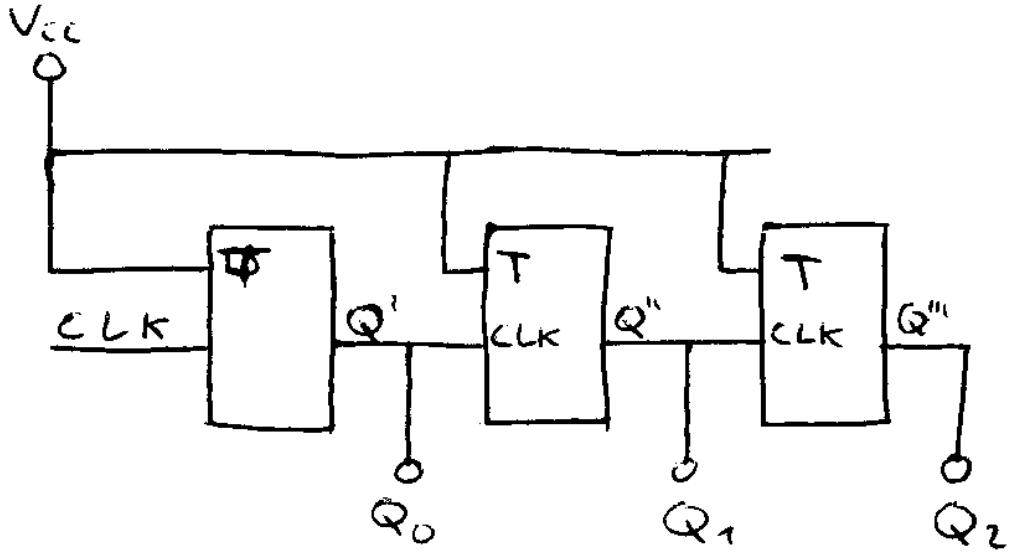
- FLIP-FLOP T



T	Q
0	Q
1	$\bar{Q}$

se T=1, la frequenza di Q risulta la metà di quella di CLK

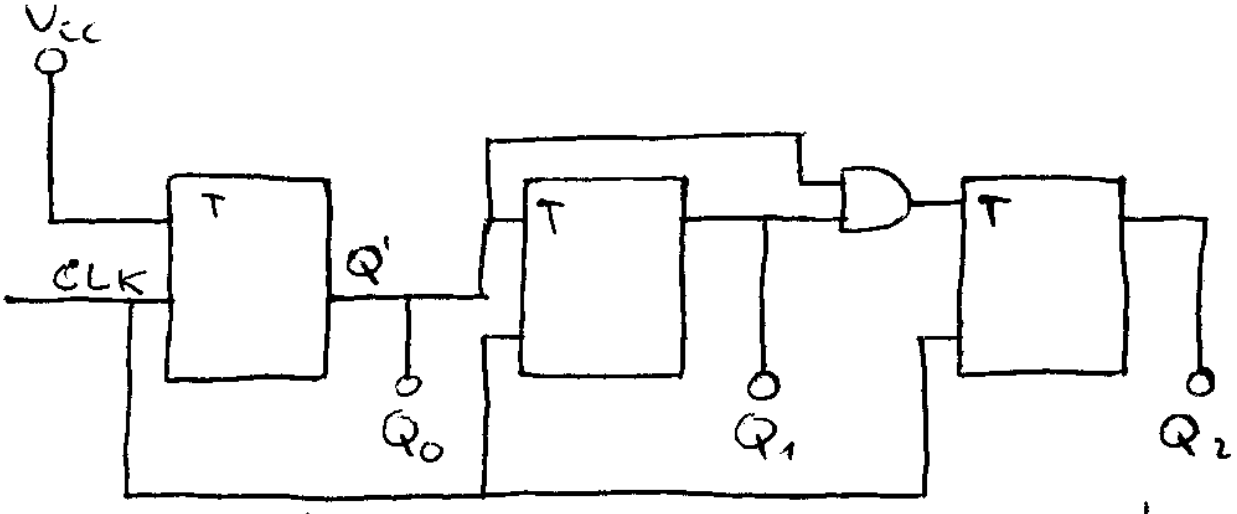
CONTATORI ASINCRONI



a ogni impulso di clock il valore di Q_0 commuta; Q_1 commuta sul fronte di discesa di Q_0 ecc. quindi la frequenza di Q_1 è la metà di quella di Q_0.
Si realizza quindi un contatore che incrementa di 1 per ogni impulso di clock ricevuto.

Il valore massimo è 2^n $n \rightarrow$ numero di flip-flop

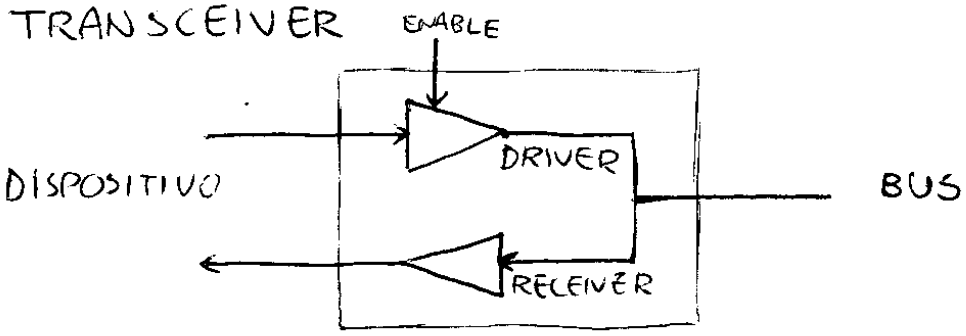
CONTATORI SINCRONI



Q_0 commuta a ogni impulso di CLK, mentre Q_1 a ogni colpo di CLK per cui Q_0 è 1, quindi ogni due.
Q_3 commuta quando $(Q_0 \text{ AND } Q_1) = 1$.

BUS

TRANSCEIVER



Comprende i dispositivi per scrivere sul bus (driver) e per leggere dal bus (receiver).
Collega i dispositivi al bus.

TIPI DI SEGNALI:
- DATO
- INDIRIZZO indicano a chi è indirizzato il dato
- CONTROLLO informazioni di stato, temporizzazione, tipo.

I segnali possono anche passare sulla stessa linea grazie a un multiplexer.

MASTER

Inizia la procedura di trasferimento e decide con chi comunicare

SLAVE

Risponde al master

BUS SINCRONI

I dispositivi lavorano alla stessa frequenza e non è necessario nessun metodo di sincronizzazione. La frequenza è stabilita dal più lento.

BUS ASINCRONI

La velocità è variabile e definita da operazioni di handshaking.

① il master attiva ABUS ponendo l'indirizzo
② il master attiva il segnale di controllo
③ lo slave mette i dati richiesti su DBUS
④ lo slave attiva xACK
⑤ dopo aver letto, il master disattiva il segnale di controllo e libera il ABUS
⑥ lo slave disattiva xACK e libera il DBUS

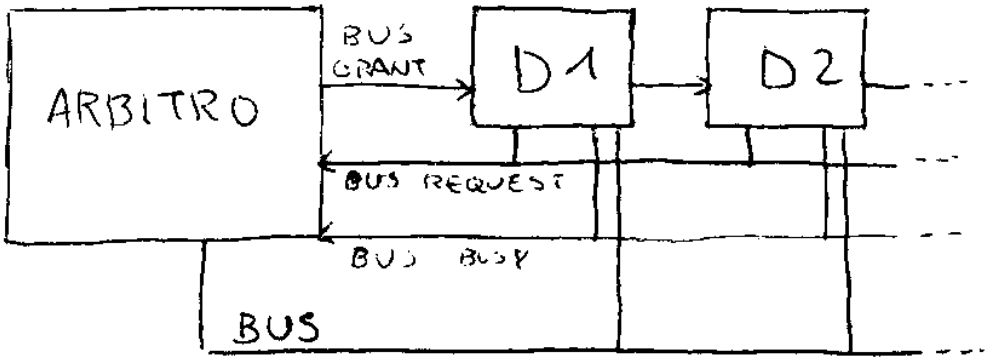
BUS

ARBITRAGGIO

dato che solo un dispositivo alla volta può essere master, devono essere definite delle regole di priorità.

- DISTRIBUITO: ogni modulo conosce le priorità e da solo il master prioritario
- CENTRALIZZATO: c'è un dispositivo (arbitro) che decide chi è il master

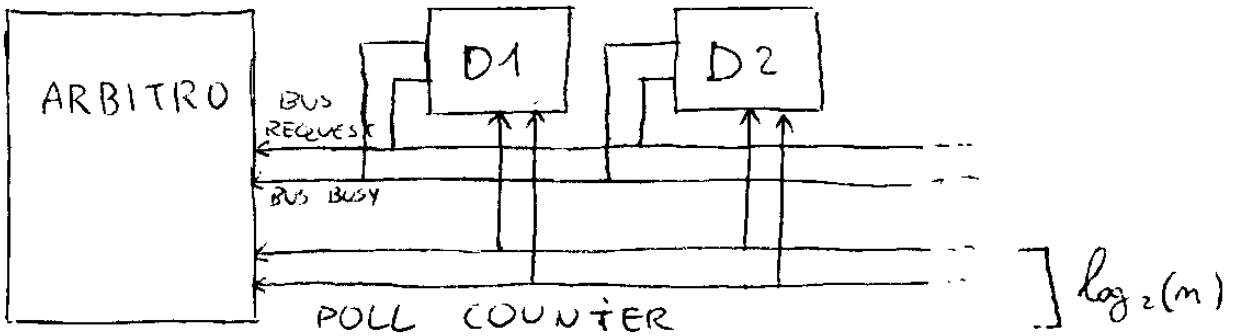
- DAISY CHAINING



① una unità (Dn) manda BUS REQUEST all'arbitro
② l'arbitro attiva BUS GRANT su D1
③ se D1 ha inviato la sua BUS REQ, attiva BUS BUSY, altrimenti passa BUS GRANT a D2 ecc.

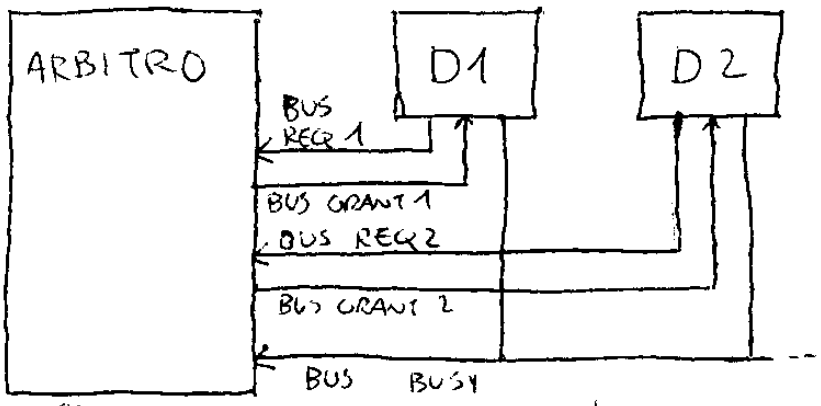
+ solo 3 segnali di controllo
- priorità fisse (dipende solo da come sono messe le unità)
- non tollera i guasti
- lenta per molte unità

- POLLING



① l'unità attiva BUS REQUEST
② l'arbitro pone su POLL COUNTER gli indirizzi delle unità in ordine di priorità
③ se l'unità il cui indirizzo è sul POLL COUNTER ha fatta richiesta, attiva BUS BUSY e l'altra interrompe la scansione

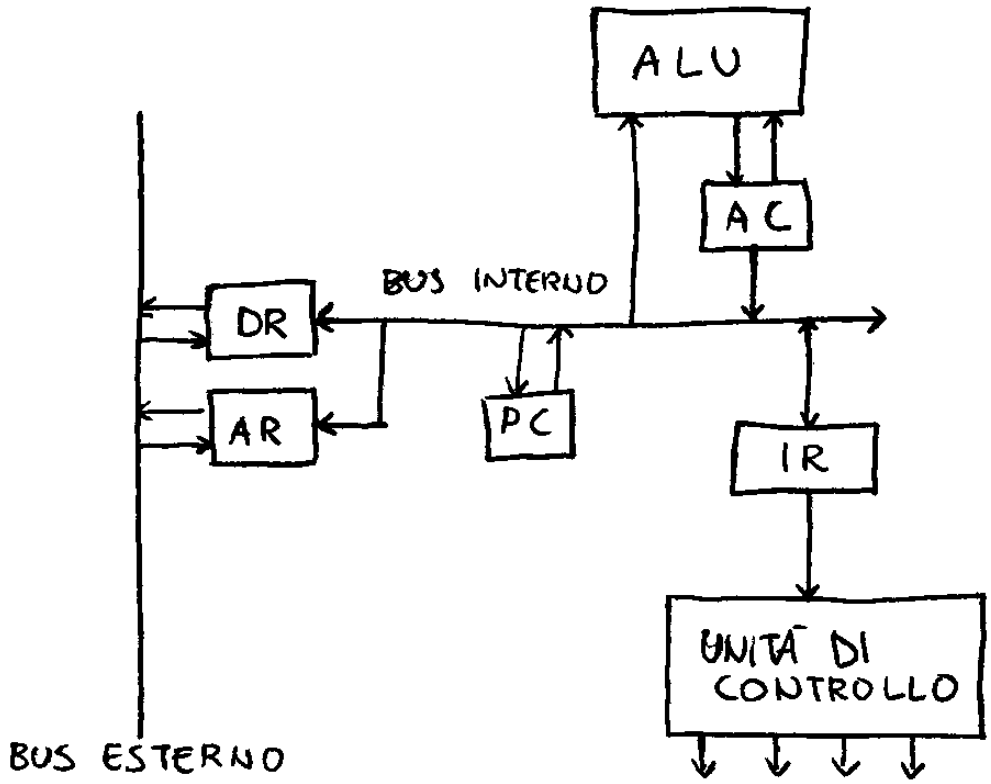
\+ facilmente a gruppi di unità
\+ le priorità sono modificabili
\- richiede $\log_2 n + 2$ segnali di controllo
\- RICHIESTE INDIPENDENTI



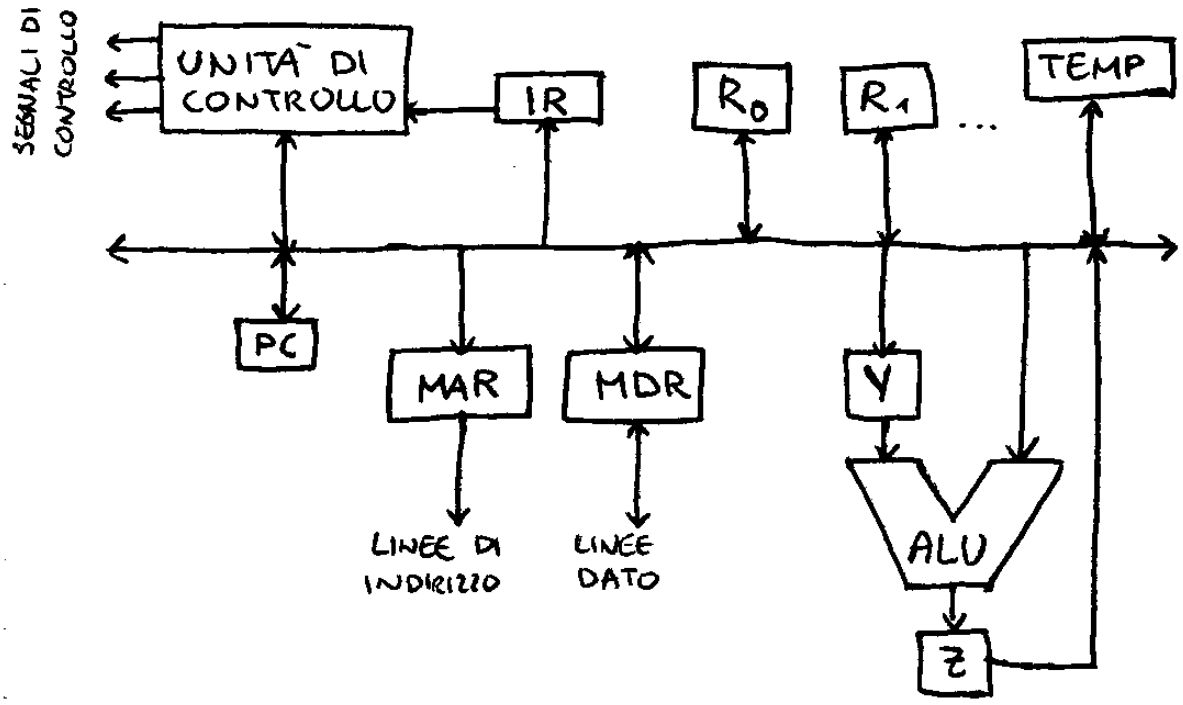
① l'unità n fa richiesta attivando BUS REQ n
② l'arbitro concede il BUS GRANT all'unità con priorità maggiore
③ l'unità attiva BUS BUSY

\+ facilmente a gruppi di unità
\+ priorità stabilite dall'arbitro
\- richiede $2n+1$ segnali di controllo

PROCESSORE



FETCH: viene letto il codice dell'istruzione dalla memoria
EXECUTE: il codice viene decodificato ed eseguito.



STACK: struttura LIFO, modificabile con PUSH e POP
lo stack pointer (SP) punta all'elemento più in alto
Lo stack permette di annidare procedure: al ritorno da una procedura si ottengono i dati nell'ordine opposto a quello in cui sono stati messi

8086

REGISTRI DATO: memorizzano operandi e risultati
- AX (accumulatore)
- BX (base) può anche essere usato per gli indirizzi
- CX (count) usato come contatore da LOOP
- DX (data) contiene indirizzo di I-O per alcune istruzioni

REGISTRI PUNTATORE:
- IP (instruction pointer) punta l'istruzione da eseguire (incrementato al fetch)
- SP (stack pointer)
- BP (base pointer) per accedere all'interno dello stack
- SI e DI (source e destination index) sono usati come indici

REGISTRI DI SEGMENTO: usati per costruire gli indirizzi fisici
- CS inizio del segmento di codice
- DS inizio del segmento di dato
- ES di dato supplementare
- SS di stack

gli indirizzi sono costruiti sommando il contenuto di un registro con il segmento corrispondente (traslato di 4 posizioni).

FLAG DI CONDIZIONE (PAROLA DI STATO)
- SF indica il segno del risultato dell'operazione (è l'MSB del risultato)
- ZF se il risultato è nullo, vale 0
- PF vale 1 se il numero di 1 negli 8 bit meno significativi è pari
- CF vale 1 se c'è un riporto nella somma o sottrazione
- AF come il CF, sul bit 3 (aritmetica BCD)
- OF vale 1 se c'è stato overflow

FLAG DI CONTROLLO
- DF indica la direzione di manipolazione di una stringa
- IF se vale 0, impedisce gli interrupt (solo quelli mascherabili)
- TF abilita il trap (utile nel debug)

PIPELINE

è un meccanismo composto da più stati la cui esecuzione richiede un tempo costante e uguale; gli stadi lavorano in cascata.

Il tempo di elaborazione è ridotto di un fattore n dove n è il numero di stadi.

BIU gestisce le operazioni con l'esterno (BUS INTERFACE UNIT)
- fetch delle istruzioni
- lettura e scrittura di operandi e risultati
- generazione degli indirizzi
- accodamento delle istruzioni

EU decodifica e esegue le istruzioni (EXECUTION UNIT)

ESECUZIONE DI ISTRUZIONI

- FETCH
  1. viene caricata dalla memoria il codice macchina dell'istruzione il cui indirizzo è nel PC
  2. incremento del PC

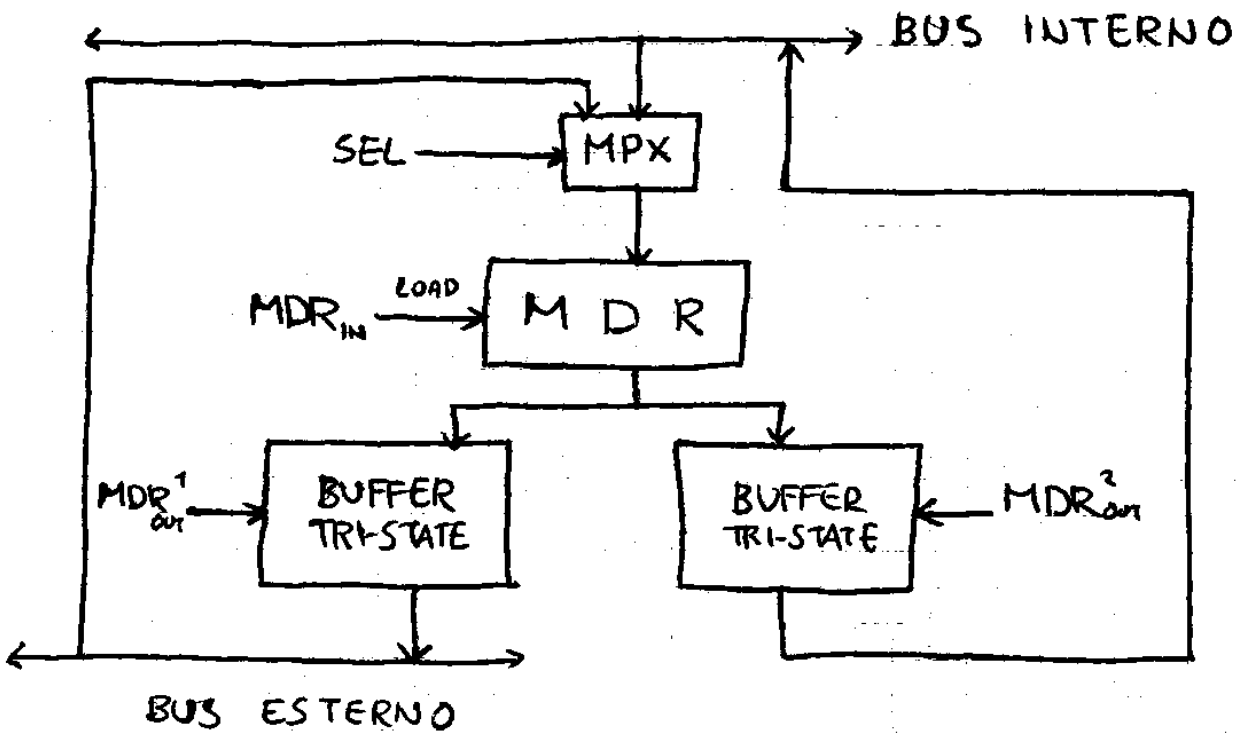
- ESECUZIONE

La CPU esegue ~~pochi~~ 4 tipi di istruzioni che compongono operazioni più complesse:
- prelievo di dati o istruzioni e caricamento in un registro
- scrittura di un dato da registro a memoria
- trasferimento di dati tra registri
- esecuzione di operazioni aritmetiche o logiche e memorizzazione del risultato

Ogni registro ha un'interfaccia controllata da due segnali, R_{in} e R_{out}, verso il bus. Solo un registro alla volta può dettare R_{out}

MDR

registro che contiene dati in ingresso/uscita dalla CPU



PRELIEVO DI UNA PAROLA DALLA MEMORIA

- MAR ← R_1
- attiva i segnali di lettura
- attendi MFC (dato pronto sul bus esterno)
- MDR ← bus esterno
- R2 ← MDR

MEMORIZZAZIONE DI UNA PAROLA IN MEMORIA

- MAR ← R1
- MDR ← R_2
- attiva i segnali di scrittura
- aspetta MFC

ESECUZIONE DI OPERAZIONI

somma tra R_1 e R_2 in R3

- setta $R1_{out}$ e Y_{IN}
- setta $R2_{out}$ e Z_{IN}
- setta Z_{out} e $R3_{IN}$

GESTIONE DISPOSITIVI

- I/O PROGRAMMATO

il processore gestisce tutte le periferiche, con continue interrogazioni (polling)

+ poco costoso da realizzare
- poco efficiente

- INTERRUPT

il dispositivo manda al processore un segnale quando ha bisogno di attenzione.

+ il processore può svolgere altri compiti nell'attesa di periferiche lente
+ la periferica ottiene l'attenzione quando necessario

All'arrivo di una richiesta di interrupt:
- vengono salvati il PC e il registro di stato nello stack
- viene eseguita una procedura apposita

GESTIONE DELL'INTERRUPT:

- LINEE DI INTERRUPT MULTIPLE

ogni dispositivo è gestito separatamente e ha un proprio piedino di interrupt

- POLLING

un solo piedino di interrupt, quando viene attivato la CPU analizza tutti i dispositivi

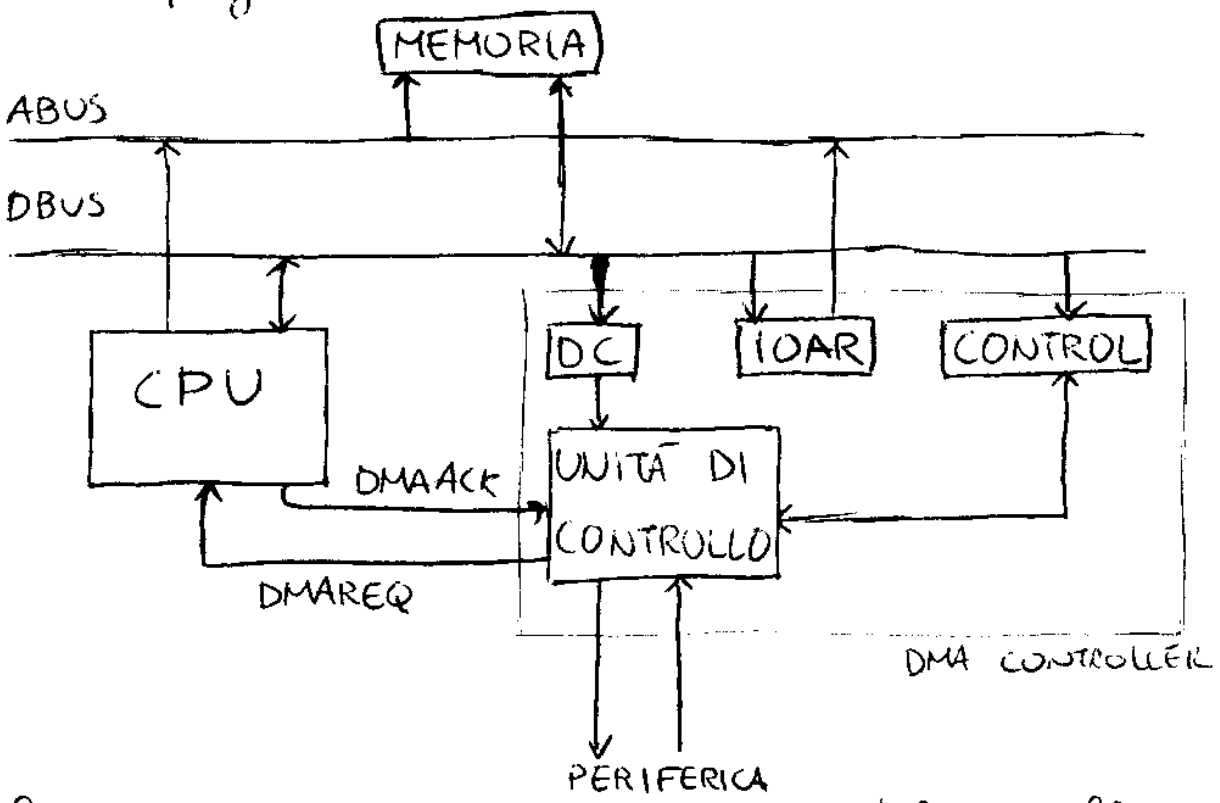
- INTERRUPT VETTORIZZATO

quando è attivato il piedino di interrupt e la CPU è pronta a servirla, manda un segnale di INTERRUPT ACK, il dispositivo risponde con un indirizzo (sul bus dati). L'indirizzo indica la procedura di servizio dell'interrupt (il collegamento tra codice periferica e indirizzo della procedura è fatto appunto dalla tabella)

DMA

metodo per trasferire dati senza occupare la CPU

Un DMA controller trasferisce dati da una periferica alla memoria e viceversa fungendo da master del BUS



① la CPU carica in IOAR l'indirizzo del primo blocco di memoria e in DC il numero di dati (consecutivi),

② la periferica richiede al DMA controller il trasferimento

③ il DMAC invia un DMA REQUEST alla CPU

④ quando la CPU la accoglie, rilascia il bus e risponde con DMA ACK

⑤ il DMAC inizia a trasferire aggiornando DC e IOAR ad ogni parola

⑥ in base a come è gestito il trasferimento, possono esserci delle pause (si disattiva DMAREQ e quindi DMAACK e la CPU riprende il controllo del bus)

⑦ quando DC = 0, il trasferimento è terminato e invia un interrupt alla CPU

MEMORIE

Memorie veloci → prezzo più alto → dimensione minore

IN ORDINE DI VELOCITÀ

- REGISTRI interni alla CPU, accesso: ns, al più qualche Mb realizzati con SRAM
- MEMORIA PRINCIPALE, fino a qualche Gb, accesso: decine di ns accessibili alla CPU con indirizzi, DRAM
- MEMORIA SECONDARIA dimensioni molto maggiori, accesso: decine di ms accesso attraverso interfaccia, realizzati con dischi magnetici
- MEMORIA OFF-LINE grandi dimensioni (Gb, anche Tb), accesso: decine di s

VELOCITÀ

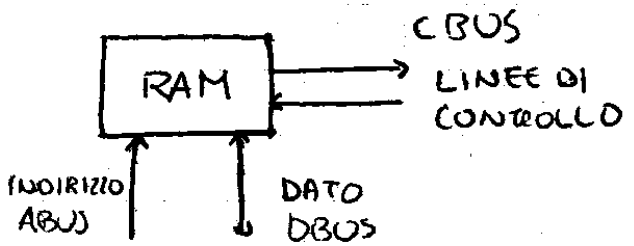
- TEMPO DI ACCESSO (LATENZA) tempo tra il momento in cui un dato è richiesto e questo viene fornito
- TEMPO DI CICLO tempo tra l'inizio di un accesso e quello successivo (>= a LATENZA)
- TASSO DI TRASFERIMENTO velocità con la quale i dati sono trasferiti; nelle memorie ad accesso casuale è $\frac{1}{\text{T.CICLO}}$

ACCESSO

- SEQUENZIALE informazioni lette e scritte in un ordine definito
- DIRETTO molti blocchi, ogni blocco è separabile ma si può decidere a quale blocco accedere con un indirizzo
- CASUALE ogni unità ha un indirizzo
- ASSOCIATIVO accesso in base alla corrispondenza del contenuto

REFRESH: necessario per mantenere un valore in alcune memorie a condensatore

RAM



- ogni cella può essere indirizzata
- tempi di accesso costanti

LETTURA

① su ABUS si scrive l'indirizzo
② su CBUS si specifica che si vuole leggere
③ su DBUS troviamo il dato che ci da la memoria

SCRITTURA

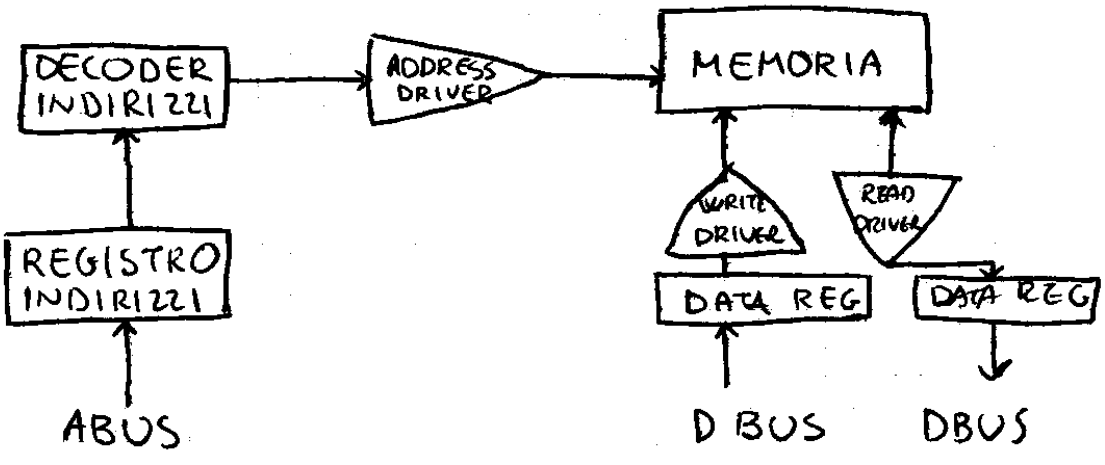
① su ABUS si scrive l'indirizzo
② su CBUS si dice di scrivere
③ su DBUS si scrive il valore da memorizzare

CONTROLLO

- tipo di operazione
- quando ABUS è pronto
- quando DBUS è pronto (sia per lettura che scrittura)
- quando è possibile predere a un nuova accesso

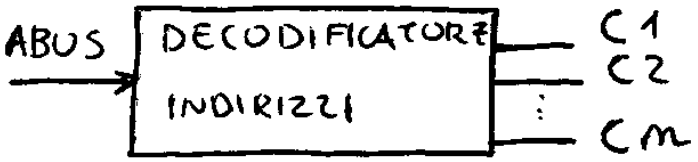
DIMENSIONI

ogni memoria ha n parole e m bit per parola
abbiamo quindi
- $\log_2 n$ segnali per gli indirizzi
- m segnali di dato



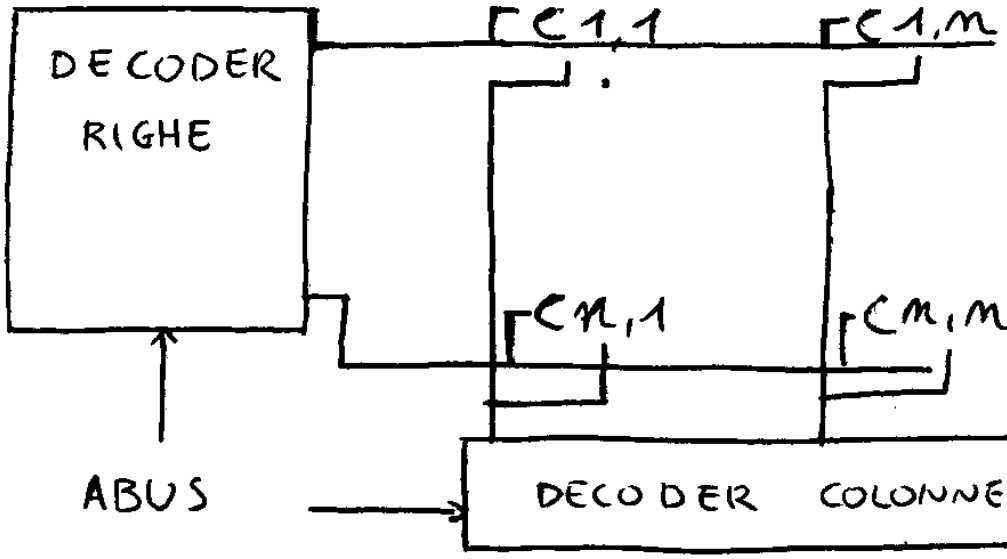
RAM

_VETTORE



1 decoder da $\log_2 n$ a n
n driver

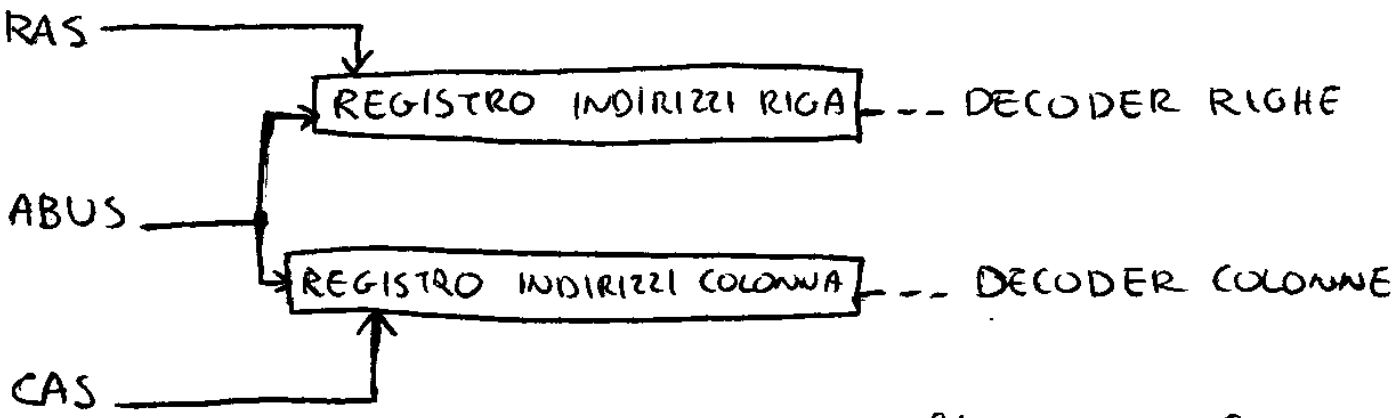
_MATRICE



2 decoder da $\log_2 \sqrt{n}$ a $\sqrt{n}$

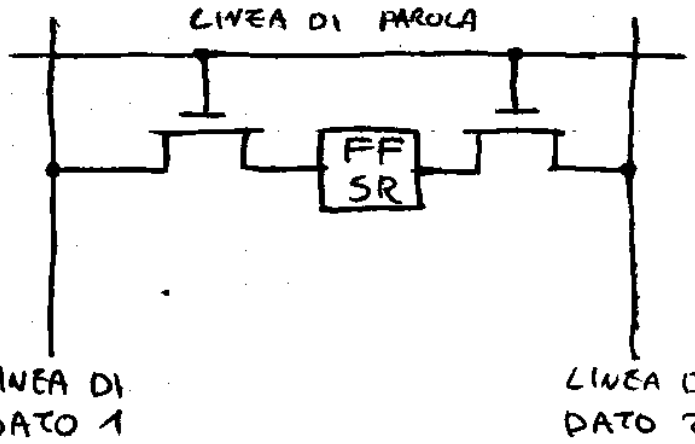
$2\sqrt{n}$ driver

per usare un solo ABUS, è possibile che sia la riga che la colonna giungano sullo stesso ingresso in tempi diversi abbiamo quindi due segnali RAS e CAS che abilitano i registri rispettivi



è inoltre possibile la PAGE MODE, ovvero l'accesso a celle nella stessa riga senza mandare ogni volta l'indirizzo della riga, che resta nel registro dato che RAS non viene attivato

- STATICA
  - ogni cella è un flip-flop
  - più veloci
  - più costose (e meno dense)
  - più affidabili

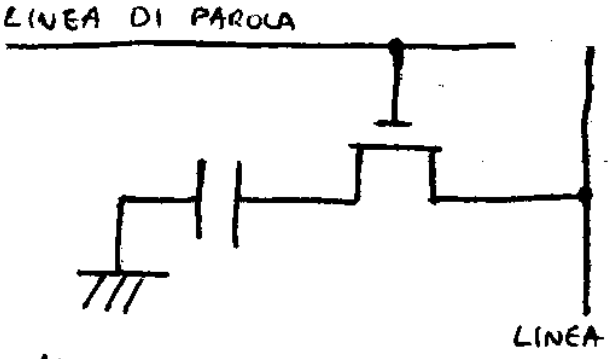


① si attiva la linea della parola (quando non è attiva il FF mantiene il valore).

② si possono leggere i due valori o scrivere forzando le linee dato

le linee dato 1 e 2 contengono sempre valori opposti

- DINAMICA



① attivando la linea di parola il condensatore è collegato alla linea di dato che legge o scrive il valore

Necessita di REFRESH perché il condensatore tende a scaricarsi

CODICE DI PROTEZIONE permette di rilevare e correggere eventuali errori nella memoria es. CODICE DI PARITÀ rileva errori di numero dispari; i CODICI DI HAMMING correggono errori singoli

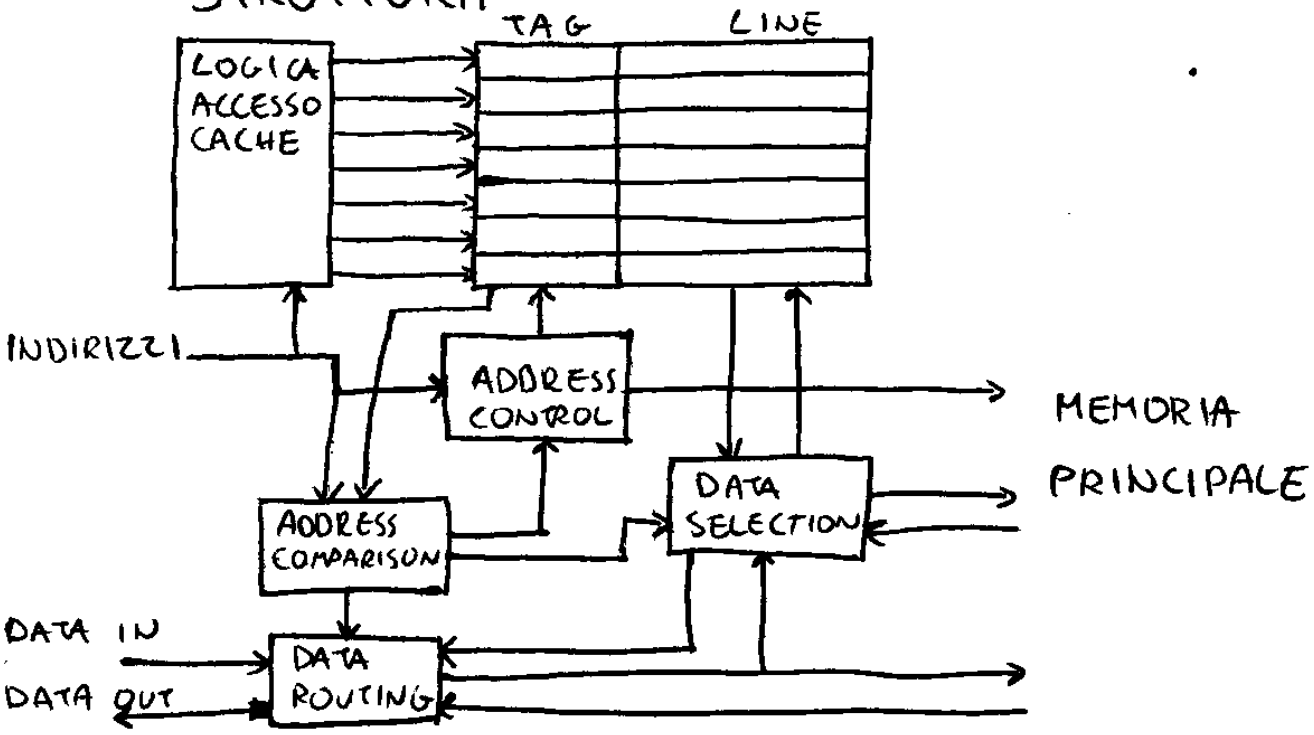
CACHE

LOCALITÁ DEI RIFERIMENTI

- TEMPORALE: se all'istante t accede alla cella C, è probabile che ci dovrà riaccedere entro t+D
- SPAZIALE: se all'istante t accede alla cella di indirizzo l, è probabile che a t+D accede a l+e

quindi, se a t carichiamo il blocco l-a, l+b, per un certo periodo di tempo è probabile che il programma troverà tutto nella cache.

STRUTTURA



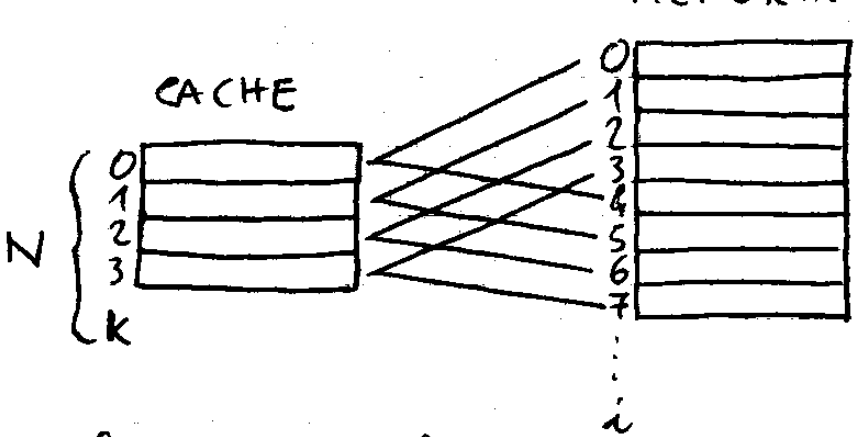
quando la CPU accede alla memoria, la cache

① intercetta l'indirizzo
② verifica se il blocco corrispondente è in cache
③ se sì, la manda alla CPU HIT
④ se no, carica il blocco della memoria di cui la parola fa parte MISS

il caricamento del blocco può avvenire nel momento (rallenta l'esecuzione) o dopo aver fornito la parola (LOAD THROUGH)

DIRECT MAPPING

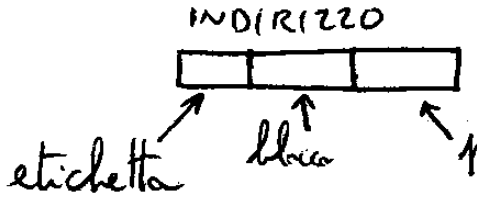
a ogni blocco della memoria corrisponde una linea di cache



$K = i \bmod N$

\+ facile da implementare (si usano i bit meno significativi dell'indirizzo)

\- se il programma accede spesso a due blocchi assegnati alla stessa linea, si ha un MISS per ogni accesso



la parte dell'indirizzo che indica il blocco identifica la linea della cache

l'etichetta è il comp TAG e serve a verificare se si tratta del dato corrispondente all'indirizzo che serve

ASSOCIATIVE MAPPING

ogni blocco di memoria principale può stare in qualsiasi blocco di cache

\+ massima flessibilità

\- molto difficile da implementare

SET ASSOCIATIVE MAPPING

le linee sono divise in S insiemi, ogni blocco di memoria principale è associato all'insieme K se $K = i \bmod S$, ma il blocco i può stare in una qualsiasi linea dell'insieme K.

ALGORITMO DI RIMPIAZZAMENTO indica come passare i blocchi nell'insieme (quali blocchi eliminare)

- LRU (il meno recente)
- LFU (il meno usato)
- FIFO (il primo entrato)
- random

AGGIORNAMENTO NELLA MEMORIA PRINCIPALE

le modifiche ai dati in cache devono essere riportate alla memoria principale per non perdere i risultati dell'elaborazione

- WRITE-BACK

per ogni blocco un DIRTY-BIT dice se il blocco è stato modificato. Quando il blocco viene eliminato, se il dirty bit è settato, questo viene copiato nella memoria principale

Può rallentare la lettura di alcuni blocchi quando c'è un MISS e un blocco deve essere copiato in memoria principale, nei sistemi multi-CPU varie cache possono avere valori incoerenti

- WRITE-THROUGH

a ogni modifica viene aggiornata sia la cache che la memoria principale, ma si considera che le operazioni di scrittura sono molto meno frequenti di quelle di lettura

MEMORIE AD ACCESSO SERIALE

TEMPO DI ACCESSO = $t_S + t_L + t_D$

t_S → seek time, tempo necessario a posizionare correttamente la testina sulla traccia

t_L → tempo per far passare il settore sulla testina

t_D → tempo per leggere i dati

Dato che t_S e t_L sono molto maggiori di t_D, i dati sono raggruppati in blocchi e la loro lettura/scrittura avviene in blocchi.

RAID insieme di dischi per aumentare affidabilità o prestazioni

0 → i dati sono divisi tra i dischi, aumenta la velocità

1 → i dati sono duplicati tra i dischi

2 → alcuni dischi sono addetti al contenimento di codici di controllo, a gruppi di bit

3 → come il 2, ma con un solo disco che contiene la parità

4 → i dischi lavorano indipendentemente, ma è per la parità

5 → come il 4, ma la parità è distribuita su tutti i dischi

CAV: velocità angolare costante, la densità dipende dalla traccia e le tracce sono concentriche. Si può accedere a ogni settore specificando traccia e settore

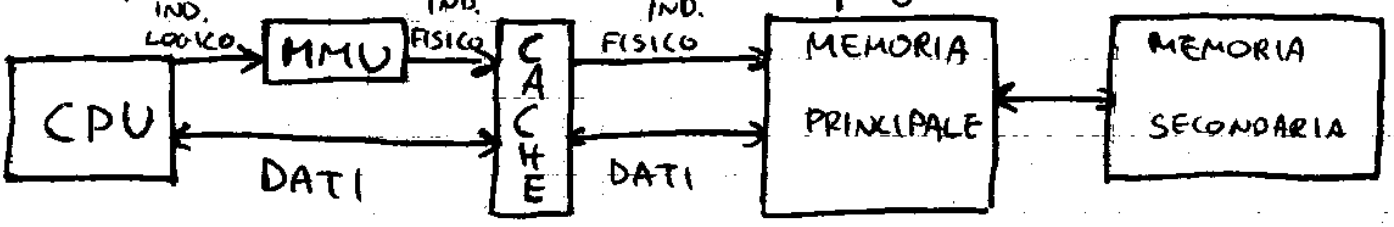
CLV: velocità lineare costante, il laser che legge segue una traiettoria costante.

MEMORIA VIRTUALE

Si tratta di una memoria "finta" composta da indirizzi logici attraverso i quali si passa a quelli fisici della memoria primaria o alla memoria secondaria. La traduzione di indirizzo è compito dell'MMU (memory managment unit)

La MMU, per ogni indirizzo logico, verifica se la parola è nella memoria principale; se si produce l'indirizzo fisico corrispondente, se no, dice al sistema operativo di caricare il blocco nella memoria principale. (PAGE FAULT)

La memoria è suddivisa in pagine, quando c'è un page fault viene copiata nella memoria l'intera pagina.



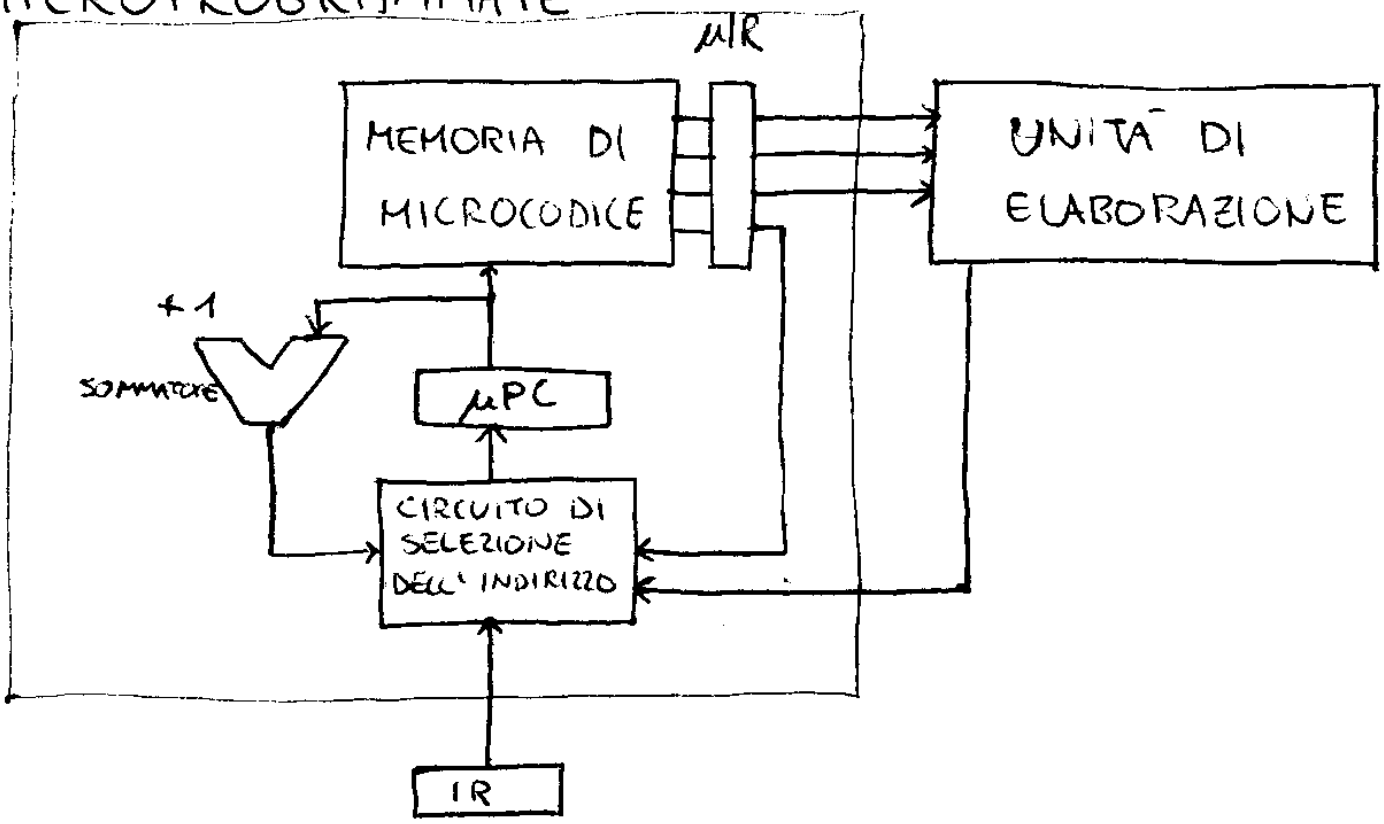
UNITÀ DI CONTROLLO

CABLATE (HARDWIRED)

si tratta di circuiti sequenziali.

- impossibile da modificare
- complessa per tabelle molto ampie
+ molto veloce

MICROPROGRAMMATE



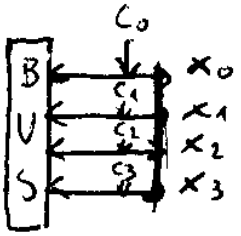
1) si usa l'indirizzo contenuto in μPC per leggere nella MEMORIA DI MICROCODICE
2) la parola letta è memorizzata nel μIR
3) il contenuto del μIR modifica i segnali di controllo per l'UNITÀ DI ELABORAZIONE e per il CIRCUITO DI SELEZIONE DELL'INDIRIZZO
4) questa logica, in base anche all'IR, genera l'indirizzo della prossima istruzione, che finirà nel μPC

+ flessibili e facili da modificare
- più lenti
- più costose

MICROPROGRAMMAZIONE ORIZZONTALE

la microistruzione contiene un bit per ogni segnale di controllo; i segnali sono quindi pilotati direttamente dal valore

- alcune combinazioni di valori potrebbero non verificarsi mai
- la memoria risulta maggiore



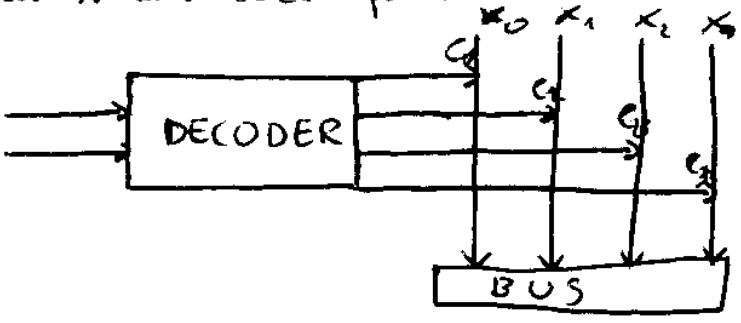
CONFIGURAZIONI POSSIBILI

0 0 0 1
0 0 1 0
0 1 0 0
1 0 0 0

→ si usano 4 bit per 4 combinazioni, quando ne potrebbero bastare 2 ($\log_2 4$)

MICROPROGRAMMAZIONE VERTICALE

le microistruzioni sono codificate, i segnali di controllo sono pilotati dai valori decodificati



si usano 2 bit per 4 segnali di controllo