# Primo semestre

**CIRCUITO COMBINATORIO**: sono usati come blocchi costruttivi per la realizzazione di microarchitetture (Multiplexer, Demultiplexer, Sommatori, ALU, circuiti che comparano, circuiti sommatori con le cifre di riporto, circuiti di moltiplicazioni). I valori di output in uscita di questi circuiti dipendono solo dai valori in entrata. Tramite la mappa di Karnaugh possiamo semplificare la formula del circuito e questo ti permette di utilizzare meno numeri di componenti HW nel circuito.

**CIRCUITO SEQUENZIALE SINCRONO / ASINCRONO(SET-RESET):** sono principalmente utilizzati per memorizzare e mantenere dati, I valori di output in uscita di questi circuiti NON dipendono solo dai valori in entrata ma anche di valori in uscita di inserimenti precedenti. Un problema dei circuiti sequenziali **asincroni** è sono le alee di commutazione che può generare errori di sincronizzazione dati da dei ritardi di propagazione, questo problema viene risolti con i circuiti sequenziali sincroni data la presenza del CLOCK che per mette la sincronizzazione totale del risultato del circuito.

Esempi di circuiti sequenziali sincroni sono:

Master/Slave Edge-triggered tipo D, utilizzato nelle RAM statiche(cache), registro a scorrimento e registri processore (Quando Ck assume il valore 0, il primo flip flop a sinistra ossia il master risulta essere bloccato sulla, quando il CK assume il valore 1 il comportamento è inverso)

memorizzazione del valore precedente, mentre il secondo flip flop a destra ossia lo slave riproduce tale

valore in uscita).

Tipo T: permettono di mantenere invariata oppure invertire l'uscita, a seconda del valore presente sull'ingresso T al momento della variazione del Clock e viene utilizzato nei registri contatori(PC ?)

Tipo J-K: è la versione sincrona del flip-flop Set/Reset per le prime tre combinazioni di ingressi, in più si aggiunge il comportamento di inversione del valore precedentemente memorizzato

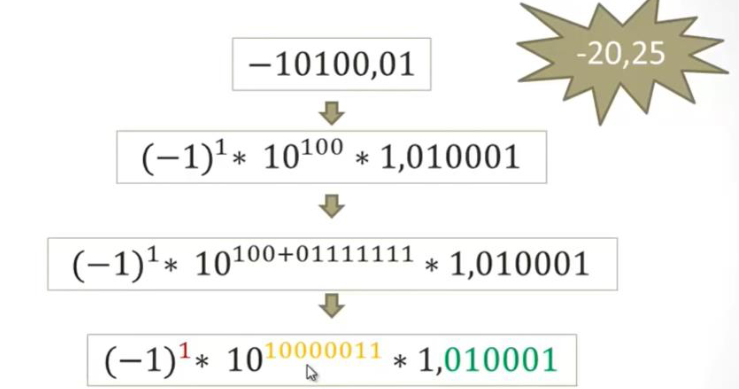
**IEEE 754**: in virgola mobile esistono diversi modi per rappresentare uno stesso valori, quindi nasce l’esigenza di uno standard. Ci sono 2 formati 32 bit (precisione semplice) e 64(precisione doppia).

32 bit: esponente viene rappresentato in eccesso 127, l’intervallo è da -128 a 127.

64 bit: esponente viene rappresentato in eccesso 1023, l’intervallo è da -1024 a 1023.

La convenzione è che la prima e unica cifra della mantissa si trovi immediatamente a sinistra del

punto decimale. Invece per quanto riguarda l’esponete se sono **tutti i bit a 0** è demoralizzato (0 ≤ 𝑀 < 1) se invece i bit sono diversi da tutti 0 sono normalizzati (1 ≤ 𝑀 < 2).

 = (-1)^S \* 2^E \*M



**CODICE AD ESPANSIONE**: sono quei codici che utilizzano le loro codifiche base ma hanno delle codifiche speciali per rappresentare l’espansione del codice che viene utilizzata quando devi utilizzare delle codifiche meno importanti.

**TRAP(sincrona) / INTERRUZIONI(asincrona)**: sono dei meccanismi adottati dalle macchine moderne, le **interruzioni** sono causate da sistemi esterni, che lanciano una interruzione alla CPU, che smette di fare quello che sta facendo e si prepara a gestire l’interruzione eseguendo L’INTERRUPT HANDLER, che ha le istruzioni che gestiscono la seguente interruzione scatenata. L’interruzione può essere gestita in 2 modi, in maniera immediata come detto in precedenza o non viene gestita immediatamente ma vengono messe le istruzioni dell’INTERRUPT HANDLER all’interno delle istruzioni che deve fare il processore a tempo debito.

Quando si sta già svolgendo una interruzione, il bit di maschera viene messo ad 1, così se viene scatenata un’altra interruzione è costretta ad aspettare. (in caso arrivasse una interruzione con i bit di maschera con una priorità superiore, interrompe l’interruzione che si sta svolgendo e svolge quella nuova)

**Trap**: questo meccanismo è simile all’interruzione ma questa volta viene scatenata da un programma interno in esecuzione, viene considerata sincrona, viene gestita chiamando subito il TRAP HANDLER che gestisce la nostra trap. Esistono 2 tipi di Trap: **implicita**🡪viene generata da un programma, **esplicita** 🡪viene utilizzata per entrare in modalità privilegiata.

**RAM DINAMICA / STATICA**:

STATICA: è formata da registri di tipo D, ci accedi in maniera diretta, è molto veloce e costosa e ci puoi accedere sia in lettura che in scrittura.

DINAMICA: è formata da registri di tipo D con transistor, che salvano temporaneamente il dato in memoria (sfrutta elettromagnetismo), così si rende necessario un refresh dei dati in memoria ad intervalli regolari. Questo la rende più lenta rispetto a quella statica, perché durante la frase di Refresh non ci si può accedere e bisogna aspettare la sua terminazione.

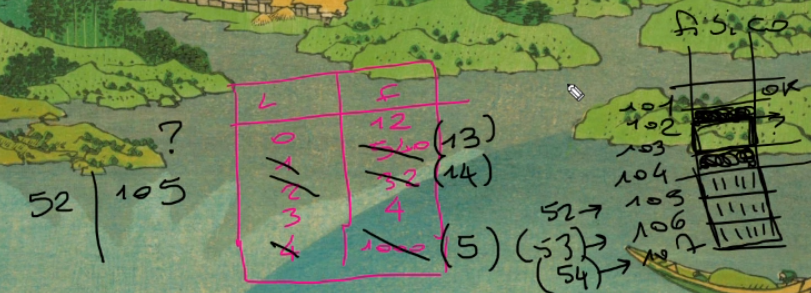
**MEMORIA VIRTUALE:** è una tecnica che permette, a livello logico, di strutturare in maniera ordinata la RAM e di non permettere ai programmi di accedere direttamente agli indirizzi fisici per evitare errori. (e permette anche lo SWAP🡪 in caso la memoria primaria si piana permette di utilizzare la memoria secondaria come RAM d’appoggio).

**SEGMENTAZIONE**: è utilizzata per suddividere la RAM in pezzi (segmenti), per un’organizzazione strutturata della memoria. Esistono 2 tipi di segmenti: **IMPLICITA**🡪il tipo di indirizzamento è gestito in modo automatica dalla macchina sapendo quanti segmenti ci sono e in base all’indirizzo in qualche segmento ci siamo trovando.

**ESPLICITA🡪** è il tipo di indirizzamento dove all’indirizzo virtuale viene aggiunto in testa il numero di segmento trattato, in caso in cui il segmento deve essere specificato perché non è definito in maniera standard a livello macchia.

Un problema della segmentazione consiste nel fatto che i segmenti sono definiti in un certo modo, ed è quindi difficile una modifica della loro dimensione in quanto sarebbe necessaria la contiguità del segmento.

**PAGINAZIONE**: questa tecnica risolve il problema della contiguità, per mette di associare ad ogni indirizzo fisico un indirizzo logico (gli indirizzi logici sono contigui fra di loro ma non è detto che gli indirizzi fisici che corrispondono lo siano). Questo sicuramente aiuta i programmi con la contiguità della memoria ma questa realizzazione richiede una tabella delle pagine molto grande, che occuperebbe esattamente metà della memoria. Per ridurre la dimensione della tabella sono state utilizzate delle tecniche di contiguità, che fanno corrispondere a N indirizzo logici contigui lo stesso N indirizzi fisici contigui.



**TECNICA MISTA**:

alla fine nelle macchine moderne si utilizza la combinazione di queste 2 tecnica. Si combina la strutura ordinata della segmentazione con la possibilità di non avere per forza indirizzi contigui da parte della paginazione.

Così si ottiene una singola tabella di segmenti, ogni segmento viene indirizzato alla sua tabella delle pagine

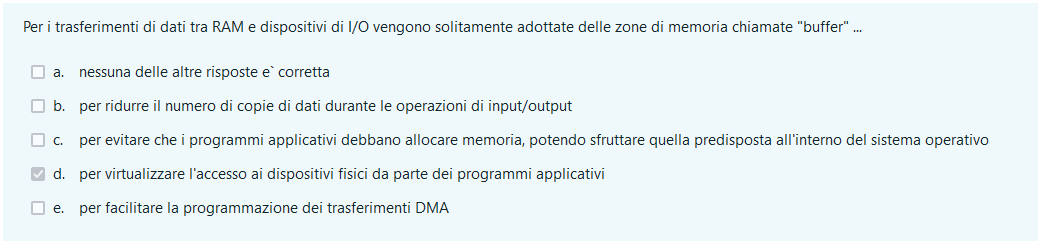
**INDIRIZZI FISICI / LOGICI**: con la tecnica mista l’indirizzo logico è suddiviso in 3 parti (segmento, pagina, offset). La parte del segmento vene utilizzata per capire all’interno della tabella dei segmenti a quale segmento ci stiamo riferendo. Successivamente la parte della pagina ci fa capire a quale pagina ci stiamo riferendo all’interno della tabella delle pagine (se la pagina supera il massimo della tabella ->lancio TRAP)

L’offset serve per capire di quanto ti devi spostare una volta tradotto l’indirizzo logico in fisico.

L’indirizzo fisico sarebbe l’indirizzo univoco in memoria che corrisponde ad unica singola cella della RAM

**MMU**: è quella dispositivo che effettua la traduzione degli indirizzi e verifica la correttezza. Il problema di questo dispositivo è temporale, quando deve verificare la correttezza degli indirizzi deve accedere in RAM e questo richiede tempo. Per velocizzare questa procedura viene utilizzata una memoria associativa (tipo CACHE’) la TLB, che contiene l’associazione fra pagina logica(tag) e pagina fisica(dati) {località spazio e tempo}

**FUNZIONAMENTO BUFFER I/O**:





D,C, E

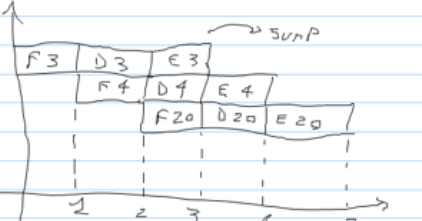
Esempio tastiera.

**PIPELINE**: è una tecnica che viene utilizzata per ottimizzare l’esecuzione di istruzioni, precedentemente i 3 stadi di esecuzione (FETCH+DECODE+EXECUTE) venivano eseguiti in maniera sequenziale, e venivano svolti in cicli di CK differenti. Nelle macchine moderne con la PIPELINE, attraverso 3 moduli differenti che si occupano dei corrispettivi stadi di FETCH+DECODE+EXECUTE, permette una esecuzione quasi in parallelo dei 3 dispostivi.

Un eventuale problema emerge con le istruzioni di salto, perché queste vengono svolte solo nella fase di esecuzione, quindi per eseguire la nuova istruzione si dovrà svuotare la PIPELINE (putando cosi delle FETCH e DECODE).

Una possibile soluzione è il Delay Slot, che ha il compito di contenere la penultima istruzione utile primo del Jump al posto di perderla, così salvi un ciclo di CK.

Un’altra operazione introdotta è la NOP (no operation) che al compito di perdere un ciclo di CK se non ho bisogno del Delay Slot.



**FETCH+DECODE+EXECUTE**: F🡪 fa la Fetch dell’indirizzo contenuto all’interno del PC e dopo lo incrementa, D🡪decodifica l’situazione presa, E🡪esegue assieme alla ALU l’istruzione appena decodificata.

**CACHE**: memoria piccola, associativa, statica e nascosta, è molto veloce paragonabile al ciclo di clock. Lo scopo di questa memoria è quello di contenere dati che rispondono alle proprietà di località nello spazio / tempo. Cioè contiene indirizzi adiacenti rispetto a quelli richiesti dal programma (località spazio), e ovviamente l’indirizzo richiesto (località tempo). Il **cache controller** è quel modulo che si occupa di quali dati copiare all’interno della Cache.

La Cache è composta da “righe di Cache” ed esse sono composte da word. Ogni riga di Cache è suddivisa in 2 parti: il campo tag (ha lo scopo di inficiare la riga / contiene la parte più significativa dell’indirizzo) e il campo dati contiene le parti meno significative associate ai dati

**3 TIPI**:

**completamente associativa**: uguale al caso base della memoria Cache detta in precedenza. (più costosa per i comparatori, è più grossa)

**Corrispondenza diretta**: in questo caso non c’è più il capo tag perché accedi in maniera diretta alla riga di Cache, ma all’interno della riga tu dovrai andare a trovare il tuo dato. (meno costosa più piccola).

**Associativa ad insieme**: corrisponde alla via di messo dei 2 tipi di Cache precedenti, ogni riga è suddivisa 3 parti: la **prima** corrisponde al tag che i permette di capire la riga di Cache nelle varie tabelle, alla **seconda** parte corrisponde alla corrispondenza di retta che ti fa capire quale tabella è quella interessata, poi la 3 parte corrisponde al offset che ti permette di capire dove prendere il dato all’interno delle word.

**ALGORITMI DI CONSISTENZA**:

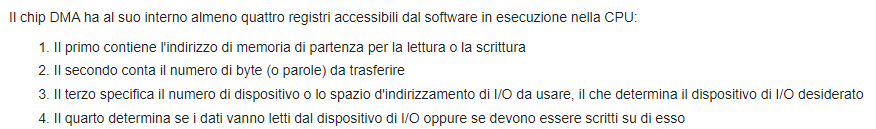
**Write-through**: appena viene modificato il dato in cache, viene subito rimodificato anche in RAM (mantenere la consistenza).

**Write-back**: il nuovo dato viene memorizzato solo nella Cache, non nella RAM. Quando il dato verrà tolto nella Cache, verrà successivamente modificato nella RAM, per eliminare l’inconsistenza in memoria (dato che in RAM c’era ancora il dato vecchio).

**TRASFERIMENTO DMA**:

è una tecnica che permette a dispositivi esterni di accedere alla RAM senza interpellare la CPU, sarà un dispositivo DMA che si occuperà del trasferimento di dati in RAM.

**BUS MASTERING**: sarebbe la tecnica di trasferimento del BUS che non interpella la CPU, e quando il trasferimento si conclude avverte la CPU

**AMBER-23**:

