Appunti di architetture degli elaboratori e sistemi operativi

Appunti

Prime lezioni

Il termine informatica trae la sua origine dal francese "*informatique*" ed è stato usato per la prima volta nel 1957 e vuold idre informazione automatica. Al concetto di informatica è strettamente collegato quello di **informazione** che vuol dire ridurre il grado di incertezza. Questa informazioni è misurabile e la sua unità è il **BIT** (Binary digit) che può assumere solo 2 valori, vero o falso.
L'informazione è tratta dalla raccolta e successivo sviluppo dei **dati**, unità fondamentale dell'informazione. Questi dati vengono trattati tramite **sistemi**, *ovvero un insieme di parti eterogenee ed interagenti volte a raggiungere un unico fine*, come disse il padre della teoria dei sistemi generali **Ludwing Von Bertalanffy**.
Il sistema che il nostro ramo della scienza informatica tratta è quello dei sistemi informativi, ovvero quel sistema che gestisce inforamzioni. Il sistema informatico invece è quel sistema che automatizza tutte le sequenza di raccolta e lavorazione dati in quello informativo.
Il calcolatore è un insieme hardware e software che permette a noi la gestione delle informazioni. Ma nel dettaglio questo calcolatore ha bisogno di un ponte che permetta alla parte software di parlare con quella hardware. Il **firmware** è proprio quella parte che ha questo lavoro di permettere alle 2 parti di *convivere*.

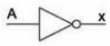
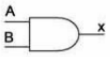
Un po' di storia

Il primo calcolatore venne inventato nel 1642 da Blaise Pascal per fare addizioni e sottrazioni, poi nel 1716 Gottfried Von Leibniz costruì una macchina in grado di fare anche moltiplicazioni e divisioni.Nel 1834, Charles Babbage, realizzò la analytical engine il primo dispositivo in grado di supportare funzioni diverse in base al tipo di programma che veniva caricato. Nel 1925 l'MIT progettò il primo calcolatore a valvole termoioniche e relè. Durante la seconda guerra mondiale spicca Alan Mathison Turing, uno dei primi e veri padri fondatori del computer. Negli anni successivi vediamo l'**ENIAC**, uno dei computer più grandi mai fatti fatto da **Von Neumann** e i suoi consulenti. Von Neumann ha progettato l'architettura della nostra CPU odierna e dei registri necessari al suo funzionamento.

Dopo le valvole i transistor presere il loro posto. Inventati nel 1948 alla Bell Labs il transistor preesnta il primo passo verso la miniaturizzazione. Sempre parlando di miniaturizzazione nel 1958 Robert Noyce inventa i circuiti integrati, che permise di inserire tanti transistor in un solo pezzo di silicio molto piccolo. tutta questa evoluzione è stata anche predetta dalla **legge di Moore** che ancora oggi ci rende partecipi di un concetto che è in continua espansione. La legge dice che ogni 18 mesi la quantità di transistor in un microprocessore sarebbe raddoppiata e infatti continuano ancora a crescere.

Logica cobinatorio e porte logiche

Queste sono le porte logiche su cui si basa la logica combinatoria. Per ora tutte le operazioni principali si fanno con la **XOR, NOT e AND**. La caratteristica principale della **XOR** è che il risultato è sempre vero se in input si hanno valori differenti, è falso se sono uguali. Questo è molto importante perchè nell'addizione binaria la XOR permette di addizionare e riportare il resto degli addendi.

Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A \oplus B$	$\overline{A \oplus B}$
Symbol							
Truth Table	A X 0 1 1 0	B A X 0 0 0 0 1 0 1 0 0 1 1 1	B A X 0 0 1 0 1 1 1 0 1 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 1	B A X 0 0 1 0 1 0 1 0 0 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 0	B A X 0 0 1 0 1 0 1 0 0 1 1 1

Operazioni

Quando porti i numeri da una qualsiasi base a binario devi prima di tutto ricordarti che essendo 1 byte devi avere sempre 8 bit come ad esempio 00100000

Conversioni

Ecco qui una tabella con tutto il necessario per la conversione tra basi.

Base	Base finale	Procedimento
Decimale	Binario	Divisione successiva prendendo il resto dal basso verso l'alto
Decimale	Ottale	Devi prima passare in binario e poi trasformare il numero binario in ottale dividendolo ogni 3 cifre da destra vestro sinistra
Decimale	Esadecimale	Prendere ogni cifra edel numero decimale e trasformala in binario con un massimo di 4 bit, quindi 4 cifre
Binario	Deciamle	Puoi usare la ***formaula*** 8 4 2 1 oppure moltiplicare ogni cifra per la base (2) ed elevarla alla sua posizione
Binario	Ottale	Devi dividere il numero binario in gruppi di 3 cifre partendo da destra e trasformare ogni singolo gruppo di bit in numero
Binario	Esadecimale	Devi dividere il numero binario in gruppi di 4 partendo da destra e rappresentare quel numero in formato decimale. Atetnzione perchè l'esadecimale parte da 0 e arrivo fino a F, quindi 15 valori
Ottale	Decimale	Devi passare priama al bianrio e poi al decimale
Ottale	Bianrio	Prendere la singola cifra e rappresentala con 3 bit in binario

Base	Base finale	Procedimento
Ottale	Esadecimale	Devi passare prima al binariio e poi puoi trasformare in Esadecimale
Esadecimale	Decimale	Devi passare prima in binario e poi poi trasformarlo in decimale
Esadecimale	Binario	Prendere la singola cifra e rappresentarla con 4 bit in binario
Esadecimale	Ottale	Devi passare prima in binario e poi trasformarlo in ottale

Addizioni binarie

Dati 2 numeri in input dobbiamo prima di tutti trasformarli in binario e poi possiamo agire. Ecco un po' di regole per non commettere errori:

- Allinea sempre a destra per non commettere errori.
- Segui il principio della XOR e ricordati i ***riporti***.
- I riporti possono essere molteplici e per questo andranno avanti fino a quando non trovi 2 zeri, questo perchè la combinazione 01 con un 1 di riporto continuerà a dare 1 di riporto per la successiva.

Sottrazione binaria

Per fare la sottrazione in binario dobbiamo prima spiegare dei **concetti necessari** ai fini di una corretta esecuzione:

- **Primo concetto** ==> i numeri positivi portano alla loro sinistra uno 0 ad indicare la loro positività, mentre quelli negativi portano un 1. Esempio 76 = **0**1001100 mentre -76 = **1**1001100. Infatti i 2 mìnumerisono uguali tranne per la prima cifra a sinistra che differisce per la diversità di segno.
- **Secondo concetto** ==> Con **complemento a uno** definiamo l'operazione necessaria per la trasformazione di un numero da positivo a negativo. Ovvero prendiamo il numero in bianrio e sostituiamo ogni zero con un uno e viceversa. Esempio -76 = -01001100 == 10110011 così otteniamo il numero negativo in binario.
- **Terzo concetto** ==> Con **complemento a due** definiamo l'addizione di 1 al numero binario negativo ottenuto con il commplemento ad 1. *Bada bene che entrambi i complementi sono necessari alla risoluzione di una sottrazione binaria*. Esempio -76 = -01001100 == 10110011 + 1 = 10110100
- **Quarto concetto** ==> È anche il più *difficile da applicare*, ovvero l'**Eccesso 2^m-1** dove possiamo rappresentare i numeri come somma di se stessi con 2^m-1 dove m è il numero di bit utilizzati per rappresentare il valore. Esempio: supponendo di volere -76 dobbiamo pensare che il suo range di esistenza è quella di -128 e +127, quindi 255 che è **2^8**. Quindi abbbiamo bisogno di 8 bit per la rappresentazione. Ora **eleviamo il 2 a m-1** dove m è il numero di bit (8) e otterremo **2^7 = 128**. Con questo 128 andiamo a fare -76 + 128 = 52 che in binario è 00110100ed infatti rispecchia il complemento a 2 ma positivo.

- **NOTA BENE!** Quando poi andrai a risolvere una sottrazione dovrai ricordarti che il numero massimo di bit a tua disposizione sono 8 e che quindi se il riporto arriva fino alla **cifra numero nove dovrai ignorarla**. Ecco un esempio completo di sottrazione binaria.

	(Decimale)	*(Binario)*
Addendo	10	00001010
Addendo	-3	11111101
Somma	7	00000111
Riporti		**1**1111000

Ignorato

Ora con un po' di attenzione la posiamo risolvere da 0. Prima cosa si **trasformano entrambi i numeri** in binario con un massimo di **8 bit**. Il numero 10 è positivo e quindi in binario sarà 00001010, il numero 3 pur essendo negativo lo prendiamo per positivo e lo trasformiamo in bianrio ottenendo 00000011. Ora con il **complemento ad uno** andiamo a sostituire gli 0 con 1 e viceversa, quindi otteniamo 11111100. Adesso con il **complemento a due** andiamo a sommare 1 al prodotto del complemento ad uno ottenendo ***11111101 che è il nostro valore negativo***. Passiamo a fare la somma e alla fine ignoriamo il riporto finale perchè sfora il nostro limite di bit, ovvero 8. Ecco che il numero 00000111 è uguale a 7 in binario.

Moltiplicazione binaria

La moltiplicazione in binario si può ricondurre a una serie di addizioni successive che hanno come moltiplicando e moltiplicatore solo il vero moltiplicando della moltiplicazione stessa. Il concetto può essere un po' dispersivo ma spero di riuscire a spiegarlo nel migliore dei modi. Prendiamo per esempio una moltiplicazione come 12 * 6 e dividiamo tutto in step per vedere come funziona.

- **Step 1** ==> Individuiamo il **moltiplicando e il moltiplicatore** che, in questo caso come moltiplicando è 12 e moltiplicatore è 6.
- **Step 2** ==> Trasformiamo entrambi i numeri in binario ottenendo rispettivamente **12 = 1100** e **6 = 0110**.
- **Step 3** ==> Adesso tutto si concentra sul moltiplicando, ovvero 1100 (12). Controlliamo quanti 1 sono **presenti nel moltiplicatore**. Da qui in poi possiamo avere **due esiti**, ovvero quello dove **l'1 occorre almeno 2 volte** oppure dove è **presente un solo 1**. Ma per adesso prendiamo in considerazione solo la prima possibilità, ovvero quella dove gli 1 sono >= 2.
- **Step 4** ==> Ora dobbiamo procedere con una sola addizione se il numero di 1 presenti è uguale a 2, con 2 addizioni successive se con 3 occorrenze del numero 1 e così via. Per poter svolgere le addizioni vediamo il **posto della prima occorrenza dell'1** (partendo da destra verso sinistra e da 0 a +∞ con i posti) nel moltiplicatore. Se maggiore di 0 (perchè con 1 presente in prima posizione non si cambia niente poichè dovremo inserire 0 volte 0 a destra del moltiplicando) andremo ad aggiungere un numero di 0 uguale al numero della posizione in cui si tova il primo 1 e poi tutti i successivi. **Esempio**: in questo caso il numero 0110 presenta il primo 1 in posizione 1 quindi

aggiungiamo uno zero a 1100 facendolo diventare 11000 e lo andiamo a sommare a 1100 con l'aggiunta della seconda ricorrenza di 1 che è in posizione numero 2. Quindi avremo 11000 + 110000 = 1001000 che in decimale è 72. Con un numero di occorrenze di 1 maggiore a 2 basta continuare l'addizione sucessiva fino all'ultima ricorrenza.

- **Step 4.5** ==> Questo caso invece è per quando si verifia **una sola occorrenza di 1**. Per esempio nella moltiplicazione di 12 * 8 abbiamo 1100 e 0100. In questo caso basta aggiungere tanti 0 quanti precedono l'1 nel moltiplicatore e si otterrà il quoziente. In questo caso il quoziente è 110000 quindi in decimale 96 che è proprio 12 * 8.

Notazione scinetifica

Per la rappresentazione di numeri molto grandi o molto piccoli utilizziamo la notazione scientifca per poterli rappresentare in modo più veloce come per esempio un numero che è seguito da 10 zeri verrà scritto così:

$$n * 10^{10}$$

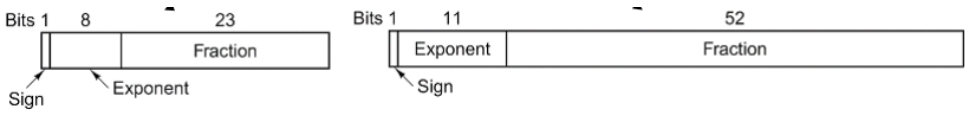
Questo tipo di rappresentazione permette in particolar modo di rappresentare i bit in virgola mobile (floating point) e anche di poterli scrivere in **codifica standard**. Ecco alcuni esempi di codifica standard:

$$3.14 = 0.314 * 10^{1}$$
$$0.00001 = 0.1 * 10^{-4}$$
$$2^{16} =$$

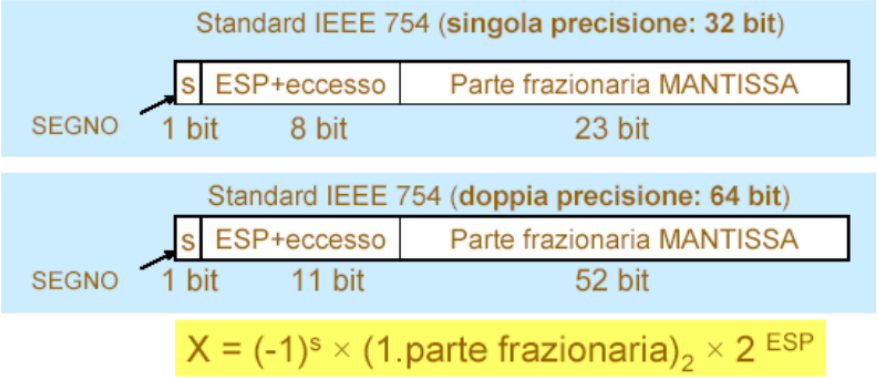
La **mantissa** (parte decimale) deve essere compresa tra 0.1 <= f < 1.
Con i float non è possibile rappresentare tutti i numeri reali perchè la densità di numeri stessi nonè infinita, ecco perchè nummeri periodici non opssono essere rappresentati e quindi vanno arrotondati.

IEEE 754

Con l'avvento dello standard **IEEE 754** negli anni 80' abbiamo un formato standard di rappresentazione dei numeri floating point binari.



Elemento	Singola Precisione	Doppia Precisione
Numero bit nel segno	1	1
Numero bit nell'esponente	8	11
Numero bit nella mantissa	23	52
Numero bit totale	32	64
Rappresentazione dell'esponente	Eccesso 127	Eccesso 1023
Campo dell'esponente	Da −126 a +127	Da −1022 a +1023
Numero più piccolo	$2^{-126} \approx 10^{-38}$	$2^{-1022} \approx 10^{-308}$
Numero più grande	$2^{127} \approx 10^{38}$	$2^{1023} \approx 10^{308}$



$$X = (-1)^s \times (1.\text{parte frazionaria})_2 \times 2^{ESP}$$

ATTENZIONE La formula in giallo permette di determinare il numero originale partendo da un numero standard.

Con *Singola precisione e Doppia precisione* vogliamo dire le architetture delle CPU, ovvero quella a 32 e 64 bit. Con differenti architetture cambiano i canoni da rispettare per la rappresentazione dei bit.
Nel caso del singola precisione vediamo come il primo bit è sempre quello che definisce il segno del nostro numero a virgola mobile, quindi 1 per negativi e 0 per positivo. Poi abbiamo **l'eccesso** che viene calcolato **normalizzando** il nostro numero trasformato in binario e, sommando l'esponente al creiterio di rappresentazione preso come standard (nel caso del singolaprecisione abbiamo 8 bit a disposizione quindi 2 elevato a 8 che sono 255. Ma in realtà è un range compreso tra -128 e +127. Mentre in quella a doppia precisione abbiamo 11 bit quindi 2047 che sono rispettivamente -1024 e +1023) che in questo caso è 127 otterremo 139 e, trasformandola in binario avremo il nostro eccesso. Ora con i bit rimasti a disposizione possiamo riportare la parte frazionaria e inserire tanti 0 quante sono le cifre mancanti per arrivare a 23 bit. *Sulle dispense è anche presente un esercizio.*
Stessa cosa vale per il prcedimento inverso, ovvero inizio col prendere il bit di segno e l'esponente. Con l'esponente in bianrio faccciamo un passaggio in decimale e poi sottraiamo 127 per ottenere l'esponente vero del nostro numero. Ora riportiamo la mantissa e moltiplichiamo per 2 elevato l'esponente per avere il numero in binario. Ora con la conversione abbiamo il nostro numero decimale.

ASCII

Il codice ASCII è stato un metodo sviluppato per risolvere un problema nato con la diffusione del calcolatore a livello mondiale, ovverlo la trascrizione delle lettere dell'alfabeto e delle caratteristiche dei singoli caratteri di ogni lingua. All'inizio l'ASCII veniva rappresentato con 7 bit e 1 bit di parità. Poi è arrivato l'ASCII esteso con 8 bit (1 byte) per rappresentare il doppio dei caratteri. Ma, con l'evoluzione delle lingue e l'aumento dei caratteri la codifica ASCII non è bastata e quindi siamo arrivati ad usare l'**UNICODE** con ben 2 byte a disposizione per rappresentare i caratteri, ovvero 2 elevato 16 che sono 65.536 caratteri. In ogni caso anche con l'UNICODE standard non avevamo abbastanza bit per rappresentare tutti i caratteri a livello mondiale, per questo vediamo l'**UTF-8** che, a differenza degli altri standard, questo è dinamico e la quantità di bit utilizzati può cambiare da un minimo di 8 a un massimo di 32 (quindi da 1 a 4 byte). Ha anche un controllo sulla concatenazione dei bit dove i 2 bit più a sinistra di un ottetto (con ottetto definisco un gruppo di 8 bit) possono avere valore 10 o 0. Con 0 avremo sapremo che quel byte è la *testa* e che dietro di se ci possono essere altri byte. Invece con 10 sappiamo con certezza che quel byte è seguito da almeno un altro byte.

Le immagini

Partiamo dalla definizione di immagini di tipo **raster** e **vettoriale**.

- Con raster definiamo quell'immagine che è un continuo o meglio un intero. Non ha una rappresentazione reale e quindi tende a perdere di chiarezza se modficata.
- Con vettoriale definiamo quell'immagine che rappresenta forme geometriche che anche se cambiate nello spazio avrenna sempre la stessa forma e valenza.

Nelle immagini digitali il **PIXEL** è quel singolo ***quadratino*** in cui viene scomposa l'immagine. Questi sono essenziali nella rappresentazione delle immagini. Infatti nella fase di rappresentazione troviamo due fasi, ovvero la **discretizzazione** e la **quantizzazione**.

- La fase di discretizzazione prevede i processi di **definizione** e **risoluzione**.
-
    - Il primo definisce il numero di pixel chiamati **DPI (dots per inch)**.
-
    - La risoluzione invece rappresenta la dimensione della nostra *griglia* come per esempio 1920x1080.
- La fase di quantizzazione prevede la rappresentazione di ogni pixel con una sequenza di bit. Per esempio in un'immagine in bianco e nero useremo 2 bit la rappresentazione, quindi i pixel bianchi verranno quantificati con 1 e quelli neri con 0. Se invece l'immagine fosse a colori, in modo analogo assegneremo ad ogni pixel una sequenza di bit per definire il colore rappresentato. Quando andiamo ad utilizzare i colori ci basiamo sui 3 colori generatori **RGB (red, green, blue)** e quindi se assegnassimo 8 bit per ogni colore andremo a rappresentare ogni pixel con 3 byte.
  Anche per le immagini il concetto di ridurre spazio di archiviazione vale e infatti vediamo che molte sono le estensioni delle nostr immagini. La differenza essenziale tra ogni estensione è l'utilizzo dell'immagine stessa e il peso (ovvero la quantità di colori utilizzati o **palette**).
  Le immagini **vettoriali** si affidano ad equazioni matematiche per disegnare le imamgini, per questo è *flessibile* e non si perde qualità quando la si ingrandisce o rimpicciolisce. I formati più comuni sono l'**SVG, JPG e PNG**. Una caratteristica molto importante dei vettoriali è che possono essere trasformati in raster ma non il contrario.

Algebra Booleana

Teoria

Ecco le leggi basiche della matematica **booleana**

Name	AND form	OR form
Identity law	1A = A	0 + A = A
Null law	0A = 0	1 + A = 1
Idempotent law	AA = A	A + A = A
Inverse law	$A\bar{A} = 0$	$A + \bar{A} = 1$
Commutative law	AB = BA	A + B = B + A
Associative law	(AB)C = A(BC)	(A + B) + C = A + (B + C)
Distributive law	A + BC = (A + B)(A + C)	A(B + C) = AB + AC
Absorption law	A(A + B) = A	A + AB = A
De Morgan's law	$\overline{AB} = \bar{A} + \bar{B}$	$\overline{A + B} = \bar{A}\bar{B}$
Absorption law 2	$A + \bar{A}B = A + B$	

Queste sono le leggi fondamentali per l'algebra booleana. Ogni formula può essere spiegata con addizioni e sottrazioni in colonna con la scomposizione della singola formula

La **semplificazione di funzioni booleane** è un processo che permette la riduzione di una funzione booleana in una equivalente così da essere rappresentabile graficamente e avere una migliore rappresentazione tramite le tabelle di verità.

f(x, y, z) = $\bar{x}y\bar{z} + \bar{x}yz + yz =$

$\bar{x}y(z + \bar{z}) + yz =$

$\bar{x}y + yz =$

$y(\bar{x} + z)$

f(x, y) = $x + \bar{y} + \bar{x}y + (x + \bar{y})\bar{x}y =$

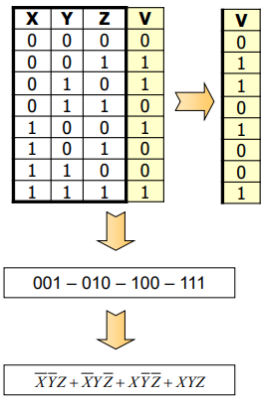
$x + y + \bar{y} + (x + \bar{y})\bar{x}y =$

$x + 1 + (x + \bar{y})\bar{x}y =$

1

Tramite questaimmagine vediamo 2 funzoini che vengono ridotte al minimo. Generalmente si usano le formule matematiche classiche come la raccolta o scomposizione di singole funzioni.

Con tabella di verità definiamo tabelle che permetono di verificare tutte le conbinazioni vere dati in input n valori.



Tramite questa tabella infatti vediamo che dati in input 3 valori avremo alla fine solo 4 possibili combinazioni. Il numero di combinazioni è dettato dal contesto della nostra tabella. Poi possiamo trasformare le combinazioni in espressione negando le variabili con valore 0 e prendendo quelle con valore 1 positive e mettendole insieme con OR (+).

Il calcolatore

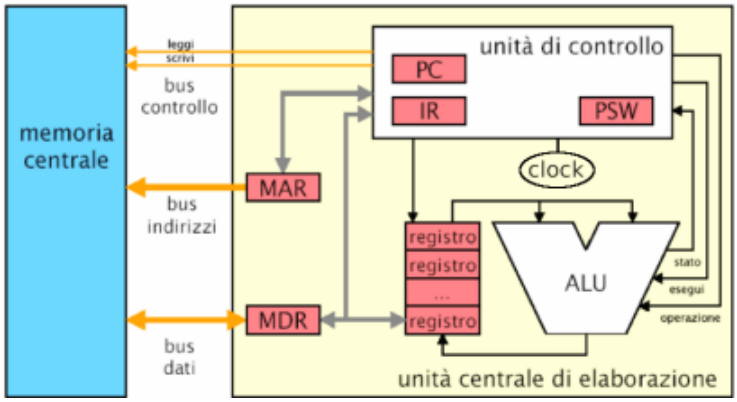
Con calcolatore definiamo una macchina che può elaborare, memorizzare e trasferire dati, tutto questo in modo controllato. Il primo calcolatore venne inventato da Von Neumann che lo rese possibile grazie ai componenti principali, ovvero **la CPU, memoria e bus** (collegamenti fra le varie unità). Con bus in realtà definiamo una specie di strade dove i dati passano. La gestione dei bus e dei dati è fatta dalla CPU (master) che gestisce il flusso di dati alle altre unità (slave). Di bus ne esistono di 3 tipi e sono:

- **Bus dati** dove viaggiano in ingresso e in uscita i dati dalla CPU;
- **Bus indirizzi** che trasmette l'indirizzo della locazione di memoria ed è unidirezionale dalla CPU verso l'esterno.
- **Bus controllo** è un insieme di linee che svolgono diversi compiti raggruppati in un solo bus che invia segnali alla CPU.

La scheda madre

La scheda madre è una componente principale che regge tutto il nostro calcolatore e permette alla CPU e agli altri componenti di interagire tra di loro. Molto importanti sono i connetori interni o meglio **PCI (Peripheral Component Interconnect)**. Di questi ne esistono 2 tipi, il PCI e il PCIe che differiscono in larghezza di tramissione e comunicazione in serie e parallelo. Sulla scheda madre troviamo anche tutti i connettori per le periferiche come gli USB, cavi lan, hdmi...

La CPU



La CPU ha come compito quello di eseguire i comandi dei programmi immagazzinati nella memoria centrale in linguaggio macchina. Ma la CPU non è un unico componente, al suo interno ci sono molte parti, come i registri che sono delle memorie molto veloci, che gli permettono di lavorare. Queste sono:

Nome	Descrizione
Unità di controllo	Legge les istruzioni della memoria centrale e ne determina il tipo
ALU	Esegue le operazioni necessarie all'esecuzione delle istruzioni (Aritmetic Logic Unit)
PC	Programm Coutner che mantiene in memoria l'indirizzo di memoria della prossima stringa da far eseguire

Nome	Descrizione
IR	Instruction Register che immagazzina una copia dell'istruzione che sta venendo elaborata
MAR	Memory Address Register contine l'indirizzo della memoria RAM dal quale si andrà a prendere o scrivere un dato
MDR	Memory Data Register contiene momentaneamente i dati per la CPU, quseto perchè è una memoria molto veloce ad accesso diretto
PSW	Processor Status Word indica l'esito di operazioni aritmetiche

La grandezza dei regisri può variare e questo è determinato dall'**architettura** o meglio **word** della CPU stessa. La word indica la dimensione dei registri che in quelli odierni può essere a 32 o 64 bit.
Quando la CPU esegue dei processi lo fa in modo ciclico, ovvero si basa su quattro operazioni che vengono ripetute fino alla fine dell'esecuzione del programma. Queste operazioni sono:

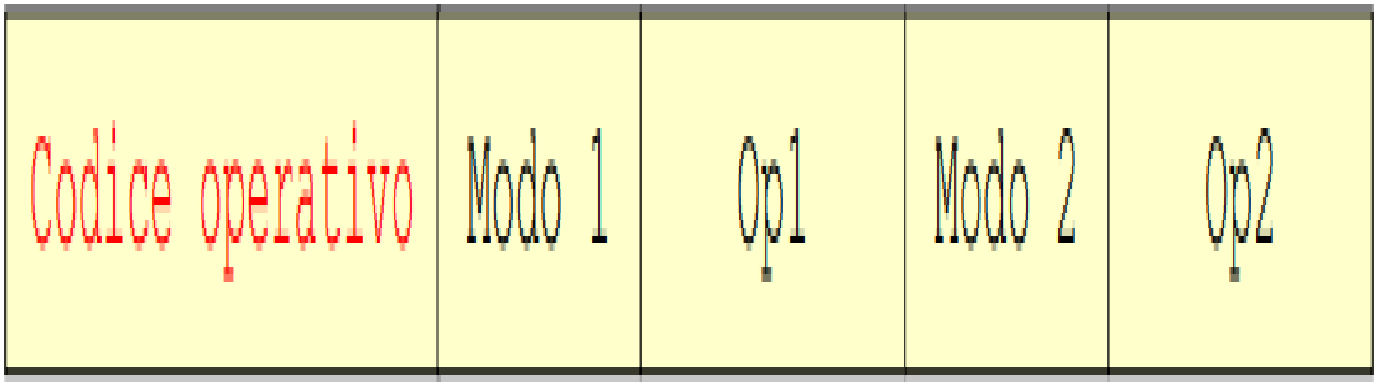
- **Fetch** (Caricamento) dove acquisisce dalla memoria un'istruzione del programma;
- **Decode** (Decodifica) che identifica il tipo di operazione da eseguire;
- **Execute** (Esecuzione)effettua le operazioni corrispondenti all'istruzione.
  Queste operazioni però sono in realtà più articolate e richiedono l'utilizzo di tutti i registri. Il processo di esecuzione è il seguente:
  Alla CPU viene dato il sesgnale di avvio, al Programm Counter viene assegnata la prima istruzione del programma. La CPU imposta nel MAR l'indirizzo contenuto nel PC e viene attivata la modalità di lettura sul bus di controllo. Grazie al MAR i bus vanno a prendere le istruzione dalla RAM. La CPU imposta nell'IR l'indirizzo del MDR e **qui finisce la fase di fetch**. La CPU incrementa il PC in modo che punti alla prossima istruzione da eseguire ed esamina l'IR e determina la prossima operazione da svolgere, **questa è infatti la fase di decode**. Tutte le unità richiamate dalla CPU si attivano ad eseguire il programma fino alla fine, andando a prelevare dati dalle memorie o anche da periferiche. Una volta terminata la fase di execute si torna alla fase 2.

Assembly

La CPU permette di eseguire le applicazioni in memoria centrale. Grazie a un **assembler** i programmatori possono dare istruzioni alla CPU traducendo il loro codice sorgente in linguaggio macchina. L'**assembly 8086** è un tipo di linguaggio che lavoro sui registri e permette di fare operazioni elementari con la CPU.

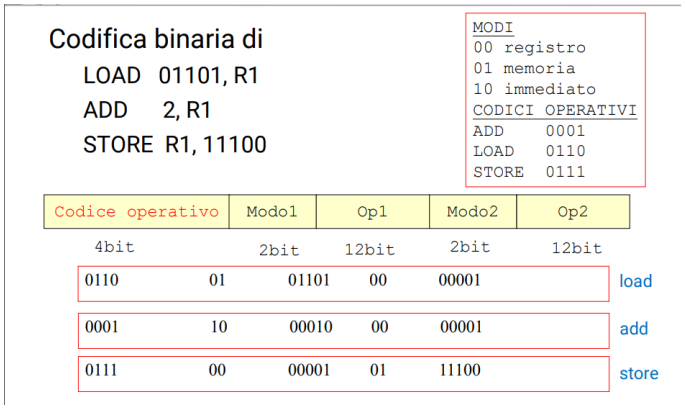
Un esempio di programma assembly può essere l'addizione x = y + 2 ottenendo:
LOAD Y, R1
ADD 2,R1
STORE R1,X
Questo codice permette di caricare il valore della variabile Y nel registro R1, sommare 2 al registro R1 e infine memorizzare il valore di R1 sulla variablie X. Tutto questo però deve essere espresso in linguaggio riconoscibile per la macchina, quindi in binario. Iniziamo traducendo la Y con 01101 e X con

11100, ottenendo così l'inidirzzo di memoria delle 2 variabili. Ora dobbiamo definire un formato semplificato per trascrivere queste informazioni alla macchina.



Questo è l'assegnamento dei bit e la loro versione semplificata. Ora però dobbiamo definire quali sono i pisto per i singoli bit.In Assembly dobbiamo definire che le istruzioni vengono lette in modo differente dalla macchina(naturalmente la lunghezza di queste stringhe varia in base alla *word* del processore, quindi il numero di bit del processore stesso). Per esempio dati dei comandi per fare un'addizione vedremo che il codice finale letto dalla CPU sarà formato da:

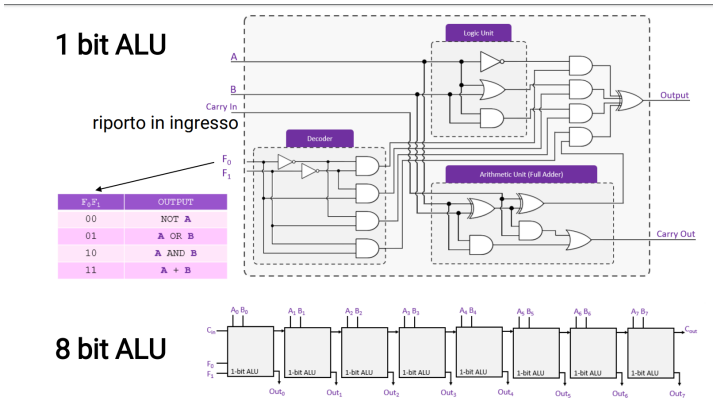
- **Codice Operativo** che codifica i singoli operatori in binario, quindi per esempio l'operatore ADD può essere scritto come 0001;
- **Modo N** indica dove si trova l'operando in base a una tabella, ovvero **00 per registri, 01 per valroi in memoria e 10 se un valore dato in input**;
- **Op** indica la posizione dove andare a prendere l'operazioni che deve andare a fare la CPU.
  ESEMPIO A PAGINA 30 LUCIDI 5



Il processore può fare tutto questo seguendo 3 fasi, ovvero:

- **Fetch** dove legge una nuova istruzione da eseguire dalla memoria centrale;
- **Decode** dove risale alla operazione decodificando i bit;
- **Execute** dove porta a termine l'operazione richiesta.

Questa immagine rappresenta come l'ALU esegue delle operazioni.



Vediamo come possiamo avere in entrata solo 2 valori, A e B, insieme a un **Carry in** ovvero il resto dell'operazione precedente. Il Entrata nel **decoder** abbiamo F0 e F1 che sono una sequenza di bit che permettono di capire quale operazione verrà svolta. Come output avremo il valore finale della nostra operazione e il **Carry out**, ovvero il possibile resto dell'operazione.

Come dovrebbe funzionare l'addizione seguendo tutto il processo interno. Prendendo ADD 2,R1 come esempio avremo che l'istruzione verrà presa dalla memoria (fetch) tramite il MAR e letta dal **PC** come indirizzo di memoria. Ora tramite il bus dati il valore codificato in binario del codice sorgente e, tramite l'**MDR** viene messo nell'**IR**.Ora il tutto viene mandato all'**ALU** e il valore 2 viene messo nello **stack dei registri** insieme al registro R1. Tramite la fase di esecuzione l'**ALU** esegue l'operazione e inserisce il risultato nel registro R1.

Il processore esegue le istruzioni in modo sincorna, ovvero esiste un segnale (**tick**) che scandisce l'inizio di un'operazione ed è riconosciuto da tutti i componenti. La frequenza con cui viene inviato il tick è misurata in *numero di tick al secondo* e si misura in **Hertz**.
La **frequenza di clock** indica il numero di cicli di clock per secondo.

Esistono due correnti di pensiero per la modalità di esecuzione delle istruzioni della CPU.

- **CISC**(Complex Instruction Set Computer) sostiene che il set di istruzioni di un compilatore debba contenere quante più istruzioni possibili;
- **RISC**(Reduced Instruction Set Computer) sostiene che ogni istruzione debba essere eseguita in un solo cilco sebbene saranno necessarie più istruzioni RISC.
  Con il tempo si è arrivati ad avere la necessità di migilorare le prestazioni delle CPU, per questo siamo arrivati ad avere il **parallelismo** e il **pipelining**.
  Per parallelismo abbiamo quello a livello di **processore** (fisico) dove più CPU cooperano per la soluzione del problema e quello di **istruzione** (virtuale) dove più istruzioni vengono eseguite contemporaneamente all'interno della stessa CPU.
  Il **pipelining** invece divide ogni istruzione in più fasi e viene gestita da un hardware dedicato. Questo permette di eseguire più istruzioni allo stesso tempo. Bisogna fare attenzione però alla dipendenza delle singole istruzioni, ovvero se un'istruzione dipende da quella precedente, questo perchè in questo caso non potrà essere eseguita perchè in attesa di input.

$$T_{esecuzione} = T_{clock} \cdot \left(\sum_{i=1}^{n} N_i \cdot CPI_i \right)$$

- T_{clock} -> periodo di clock della macchina
  dipende dalla tecnologia e dalla organizzazione del sistema
- CPIi -> numero di clock per istruzione di tipo i
  il numero di clock occorrenti affinché venga eseguita l'istruzione di tipo *i*. Se il set di istruzioni contiene istruzioni di tipo semplice si ha un CPI basso, se invece le istruzioni eseguono operazioni più complesse si ha un CPI elevato
- Ni -> numero di istruzioni di tipo i (somme , sottrazioni , salti, ecc.)
  dipende dal set di istruzioni e dalla qualità dei compilatori (ossia la capacità dei compilatori di ottimizzare il programma). Se il set di istruzioni contiene istruzioni di tipo semplice si ha un N_i alto; caso opposto per istruzioni di tipo più complesso che permettono di abbassare N_i. Infatti, con un set di istruzioni di tipo semplice ne occorrono di più per svolgere le stesse funzioni.

La logica dei calcolatori RISC è quella di avere un set di istruzioni molto semplici che mantengono basso il CPI a scapito del valore di N_i, mentre quella dei calcolatori CISC è esattamente opposta.

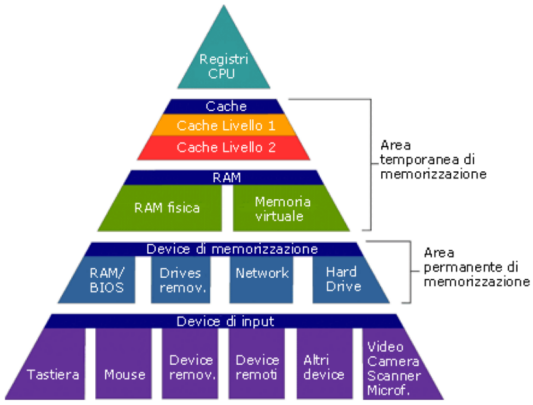
Piccolo schema sul calcolo del periodo di clock

La memoria

Introduzione

La memoria in un qualsiasi tipo di device elettronico è il posto in cui vengono memorizzate le informazioni. Di memorie però ne esistono di diversi tipi e con caratteristiche differenti che le contraddistinguono, alcune delle caratteristiche fondamentali sono:

- **Velocità**
- **Dimensione**
- **Permanenza dei dati**, ovvero che mantengono il contenuto anche senza alimentazione
  Queste caratteristiche però non sono conciliabili tra i diversi tipi di memorie disponibili sul mercato, per questo dobbiamo distinguere le memorie in due famiglie, ovvero le **Memorie centrali** e le **Memorie di massa**.
  Le prime, ovvero le centrali (RAM) , sono memorie molto veloci che salvano i programmi in esecuzione con i relativi dati, ma non hanno permanenza. Quelle di massa (HDD) invece sono generalmente più lente di quelle centrali, ma con una capienza massima molto più grande. Inoltre sono **persistenti**, ovvero che anche senza alimentazioni mantengono i dati al loro interno.



Le memorie si possono distinguere anche per le tecnologie come:

- **memorie elettroniche**, ovvero con velocità elevate ma necessitano di alimentazione (memoria centrale);

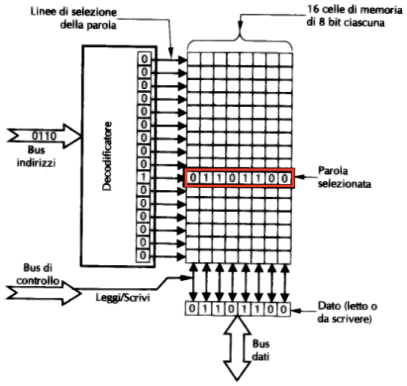
- **Memorie magnetiche**, ovvero permanenti e più lente (memoria di massa);
- **Memorie ottiche**, ovvero delle memorie riscrivibili come i CD e DVD

La memorie è strutturata in una successione ordinata di elementi binari, ogni unità è detta cella ed ha un indirizzo ben specifico per essere riconosciuta. La dimensione massima della memorie indirizzabile dipende dal numero di bit utilizzati per codificare gli indirizzi.

L'accesso alle celle di memoria può avvenire in diversi modi, ovvero:

- **Acesso sequenziale**, l'accesso a un dato comporta la lettura di tutti quelli che lo precedono;
- **Accesso casuale** il tempo di accesso non dipende dalla posizione del dato ed è costante per tutte le celle;
- **Accesso misto** dove la posizione della cella non è determinabile nello specifico e per questo si accede in prossimità della cella e si controllano tutte le celle in modo sequenziale (dischi);
- **Accesso associativo** è un metodo ad accesso casuale nel quale le celle vengono selezionate in base al loro contenuto anzichè utilizzando un indirizzo (memoria cache)

A livello fisico l'accesso alla memoria centrale avviene tramite il *MAR* (Memory Address Register) e si prelevano o scrivono dati tramite l'*MDR* (Memory Data Register) in base al segnale definito dal bus di controllo (il segnale può essere lettura o scrittura). In un computer odierno lo spazio di indirizzamento è di almeno 32 bit (dimensione del MAR) e quindi possiamo indirizzare 2^32 bit quindi 4 GB.



Tipologie di memorie

Nome	Descrizione
RAM	**Random Access Memory**, memoria ad accesso casuale di tipo volatile (si resetta ad ogni avvio del PC)
ROM	**Read Only Memory**, memoria non volatile scritta al momento della produzione e che non può essere cancellata. Utilizzata per contenere operazioni del BIOS
Memorie Flash	Memorie elettroniche non volatili ma riscrivibili. Una pendrive è un tipo di memoria flash
SIMM	**Single Inlne Memory Module**, un tipo di RAM basata su condensatori che immagazzinano un bit per condensatore. Se il condensatore perde la carica l'informazione è perduta. Ha ampiezza di bus di 32 bit e sono usate per i bus dati

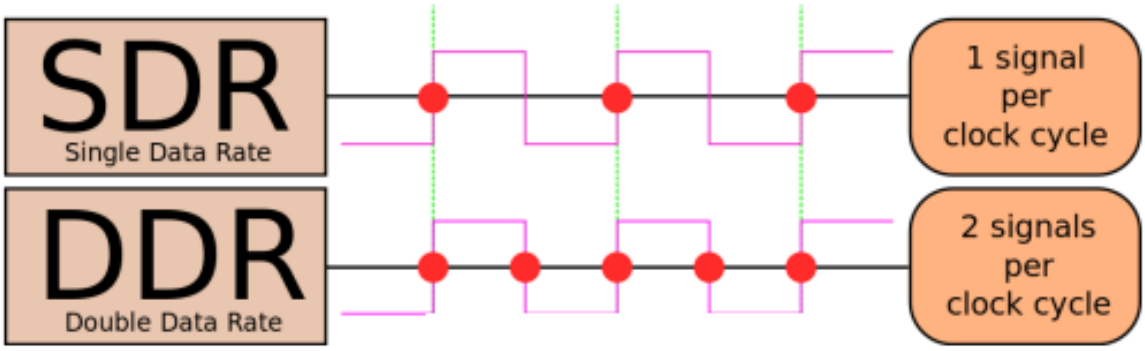
Nome	Descrizione
DIMM	**Dual Inline Memory Module** sono RAM con i connettori su entrambi i lati della scheda e un'ampiezza del bus a 64 bit
SO-DIMM	**Small Outline Dual In-Line Memory Module** è un formato di RAM diffuso nei computer a circuito integrato come portatili, router e stampanti

Tutte queste memorie hanno dei parametri che le rendono differenti per il loro scopo, abbiamo infatti:

- **Tempo di accesso** ovvero il tempo necessario a rendere disponibile un valore in una cella di memoria;
- **Ciclo di memoria** valutato sommando al tempo di accesso la durate del *periodo refrattario*, ovvero il tempo in cui la memoria non è utilizzabile;
- **Velocità di trasferimento** ovvero la quantità di dati trasferiti nell'unità di tempo, si misura in bit/sec.

Sul mercato odierno abbiamo due tipi principali di RAM, ovvero le **SDR** e le **DDR**.

- Con **DDR** (Double Data Rate)indichiamo le RAM più performanti al giorno d'oggi, questo eprchè possono trasferire un byte sil sul fronte di salita che su quello di discesa del clock. Al giorno d'oggi contiamo 5 generazioni di DDR, la più recente è la DDR5.
- Con **SDR** (Single Data Rate) invece, possiamo trasferire dati solo sul fronte di salita del clock.



Controllo degli errori

Oggi abbiamo RAM molto affidabili, ma nonostante ciò i sistemi effettuano un controllo all'avvio usando la tecnica del **bit di parità**, ovvero un controllo dove si aggiunge un bit alla fine di una sequenza. Queto bit può avere valore 1 se è presente un numero di 1 dispari, altrimenti varrà 0.

Memoria cache

La **memoria cache** è un tipo di memoria molto veloce che troviamo vicino alla CPU, questo perchè permette di salvare le istruzioni da svolgere in modo che il processo non diventi più lento andandole a cercare nella memoria centrale, questo perchè sarebbe troppo lenta. In generale la cache richiede alla memoria centrale e, insieme all'istruzione necessaria richiede anche le celle successive, così che non dovrà attendere per la prossima richiesta e le invia alla CPU. Questo ci fa capire che la cache è un chip speciale, molto e veloce e quindi costota. Attualmente abbiamo tre livelli di cache:

- **L1 o cache di primo livello** è un tipo di cache presente fisicamente nel processore;

- **L2 o cache di secondo livello** si può trovare alloggiato sia nella scheda madre che nel core;
- **L3 o cache di terzo livello** che è condivisa tra i core dell'elaboratore.
  Qundi alla fine avremo L1 e L2 situate in ogni core della CPU e L3 condivisa da tutta la CPU.

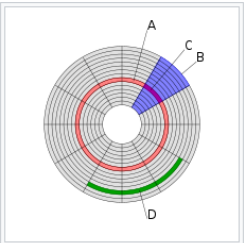
Le ROM

Con **ROM** definiamo delle memorie di sola lettura (Read Only Memory) che contengono istruzioni necessarie al funzionamento della macchina e che non possono essere riscritte da un utente. Vengo scritte e installate al momento della saldatura sulla scheda madre. Di queste ROM ne abbiamo di diverso tipo:

- **PROM** ovvero Programmable Read Only Memory, sono delle ROM programmabili una sola volta;
- **EPROM** ovvero Erasable Programmable Read Only Memory che possono essere riprogrammate con apparecchiature speciali;
- **EEPROM** Electrically Erasable Programmable Read Only Memory sono delle memorie flash che possono essere sempre riscritte

Memoria di massa

- **I Nastri Magnetici** permetto solo accesso sequenziale ai dati, quindi non è possibile decidere quale celle prendere. Inoltre queste memorie sono usate per scopi particolari, come archivi storici;
- **I Floppy Disk** sono stati i prmi supporti magnetici con accesso diretto ai dati, la loro grandezza massima è di 1,44 MB e al giorno d'oggi non sono più utilizzati.
- **I Dischi Magnetici** o meglio Hard disk sono dei dispositivi elettro-meccanico per conservare informazioni sotto forma magnetica tramite la lettura e scrittura fatta da una **testina rotante**. Questa testina contine un induttore ed è sospesa sopra i dischi. L'hard disk è strutturato in **cilindri**, **tracce** e **settori**. Le tracce sono dei cerchi concentrici sul disco, i settori sono una suddivisione delle tracce mentre i cilindri sono dei gruppi di tracce.



Struttura della superficie di un piatto:
A) Traccia
B) Settore geometrico
C) Settore di una traccia (o anche traccia di un settore)
D) Cluster, insieme di settori di tracce contigui

  Ogni traccia è una sequenza di bit e ogni settore ha una grandezza fissa. **La scrittura** avviene quando la corrente negativa o positiva attraversa la testina così da magnetizzare il settore del disco. Una volta magnetizzato in base alla polarizzazione della corrente le particelle si allineano verso destra o sinistra. **La lettura** invece aviene quando la testina passa sopra un'area magnetizzata e viene indotta una corrente positiva o negativa nella testina. Questo permette di rileggere i bit memorizzati sul settore. La capacità di memorizzazione dipende principalmente dalla densità di registrazione:
-

- **Densità lineare** ovvero la difficoltà di individurare le variazionidel campo magnetico sulla superifice del disco;

- **Densità di area** ovvero la capacità di bit/in^2 (bit su pollice quadrato) del disco. I più recenti arrivano fino a 1 Tbit/in^2 ovvero un Terabit su indice cubo.
  Le prestazioni degli hard disk dipendono da:

- **Seek time** ovvero il tempo di spostamento delle testine sul cilindro desiderato;

- **Latency time** ovvero il tempo necessario al settore interessati di passare sotto la testina. Questo fattore è collegato alla velocità di rotazione del disco stesso. Oggi abbiamo dischi che vanno dai 5400 a 15000 RPM (round per minute).

- **SSD (Solid state disk)** sono dei dischi di nuova generazione che hanno superato sotto ogni aspetto i dischi magnetici perchè basati su componenti elettrnoche (memorie flash NAND). I chip di memoria di tipo NAND sono composti da celle che includono un **transistor a due gate**. Il secondo gate viene utilizzato per conservare il valore della cella ed è conosciuto come **gate flottante**. Esistono diversi tipi di memorie SSD in base a quanti valori possono essere contenuti nel gate, ecco una lista:

- **SLC (Single level cell)** può tenere traccia di due valori;

- **MLC (Multi level cell)** può tenere traccia di quattro valori;

- **TLC (Triple level cell)** può tenere traccia di otto valori;

- **QLC (Quad level cell)** può tenere traccia di sedici valori;
  Questi valori vengono mantenuti in base alla tensione applicata sul transistor. Maggiori sono le celle più saranno i livelli di divisione della tensione. Nonostante gli SSD sono molto più performanti non vengono ancora largamente utilizzati come gli HDD, questo per via di alcuni fattori cruciali che li dividono, ovvero il prezzo (gli hard disk costano molto meno e sono più capienti), il recupero di dati in caso si dovesse rompere non è possibile con un SSD e soprattutto gli SSD si deteriorano con frequenti operazioni di scrittura.

SSD e HDD a confronto

SSD	Parameters	HDD
0.1 ms	Access Times	5.5~8.0 ms
6000 io/s	I/O Performance	400 io/s
20 ms	I/O Request Time	400 ~ 500 ms
0.5%	Lifespan (Failure Rate)	2 ~ 5%
2 & 5 Watts	Energy Savings	6 & 15 Watts
1%	CPU Power Consumption	7%
6 Hours	Backup Rates	20~ 24 Hours

- Con **Memoria flash** definiamo un tipo di memoria a stato solido non volatile. Inventate negli anni 80 dalla Toshiba sono diventate sempre più popolari e capienti, fino ad arrivare alle pennette USB e scehde SD.

DIschi ottici

I **CD (Compact Disc)** sono stati inventati negli anni 80 come supporto per la musica. Le informazioni sono codificate per mezzo di fori chiamati **Pit** di 0.8 micron alternati a delle zone piane chiamate **Land** lungo un'unica spirale interna al disco. Il passaggio pit-land codifica un 1, mentree l'assenza di passaggio codifica 0. La lettura avviene tramite un laser che viene riflesso su pit e land. Durante la rotazione del CD la velocità varia perchè avendo una densità di informazionin costante ma con diametro diverso il laser deve girare in modo più o meno veloce in base alla zona. La lettura avviene con il segnale in **radio frequenza** che, tramite circuiti, viene filtrato e trasformato in **un'onda quadra** in cui i fronti di salita e discesa rappresentano i bit a 0 e 1. Da qui si passa al segnale **NRZ** dal quale vengono prelevati i sincronismi e i bit buoni che vengono fatti proseguire al fine di ottenere un'informazioni.

CD-ROM

I CD-ROM sono uno speciale tipo di CD di sola lettura che utilizzano una tecnologia per evitare la perdita di bit durante la scrittura. Hanno una capienza di 650-700 MB e sono formati da frame e settori. Ogni frame è formato da 42 simboli dove ogni simbolo è formato da 14 bit. 98 frame foramtno un settore.

CD-R

I **Compact Disc-Recordable** sono nati a metà degli anni 90 e svolgono la stessa funzione dei CD-ROM ma sono registrabili dall'utente. La scrittura avviene tramite la *bruciatura* del pit-land tramite un raggio laser.

CD-RW

I **CD Rewritable** sono dischi riscrivibili molto simili ai CD-R ma hanno la possibilità di essere cancellati, ovvero di tornare allo stato precedente alla scrittura laser.

DVD

I **DVD (Digital Video Disk)** nascono alla fine degli anni 90 per la memorizzazione dei video digitali. La differenza principale con i CD sono: la dimensione di 4,7 GB, distanza ridotta tra i pit e land di 0.4 micron e la velocità di lettura che è di 1,4 MB. Inoltre è formato da un doppio strato di policarbonato che riflette il laser di lettura da uno dei due lati.
I DVD hanno degli standard ben distinti e sono:

- **DVD-R** è lo standard più diffusi grazie alla sua compatibilità con la maggior parte dei lettori DVD da tavolo. Inoltre possono essere di 4,7 GB o 9,4 GB.
- **DVD+R** è uno standard meno popolare ma che è anche riscirvibile come un DVD-RW
- **DVD-RAM** è lo standard con la maggior qualità e numero di riscirzioni ma ha una bassissima compatibilità con i lettori DVD.
  La letteru di un qualsiasi CD/DVD avviene tramite il lettore, oggetto che tramite un raggio laser passa attraverso lo strato di policarbonato, riflette lo strato di alluminio e colpisce un componente ottico che essendo sensibile ai cambiamenti di luce riesce a determinare la presenza di pit e land.

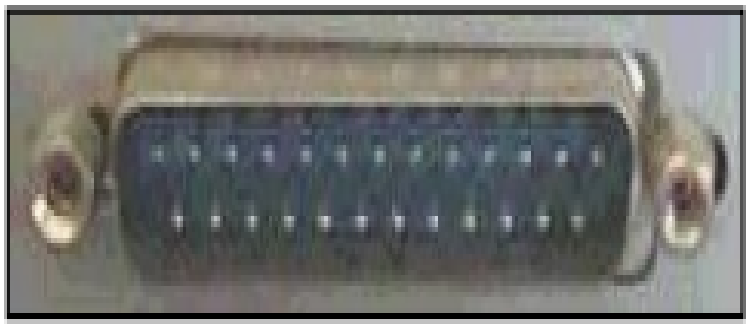
Periferiche di I/O

La parte principale di un calcolatore è quella di poter interagire con l'esterno, mandando e ottenendo dati. Questo è possibile grazie alle **periferiche**, ovvero strumenti che grazie a delle **interfacce** *parlano* con la macchina. Il collegamento con le periferiche avviene mediante **porte** di I/O presenti sulla scheda madre.
La modalità trasmissiva dei dati tra interfaccia e periferica può essere **seriale o parallela**. Come dice il nome nella prima la trasmissione dei bit avviene in successione, mentre con la parallelo i bit vengono inviati in gruppi. La più efficiente rimane però la seriale.
Quando una periferica si interfaccia con la macchina il trasporto dei bit non è diretto, ma getstito dai registri di stato che si trovano tra la periferica e la CPU. Questo perchè la priorità data dalla CPU è molto bassa. A questo problema di sincronizzazione subentrano delle modalità di trasmissione dei dati, ovvero:

- A controllo di progrmma (polling);
- A interruzione (interrupt);
- Con accesso diretto alla memoria (DMA);

Interfaccia seriale RS-232C

Questo tipo di interfaccia si basa sul protocollo seriale di tipo asincrono dove i bit sono trasmessi uno alla volta su di un solo filo e, soprattutto, senza l'aggiunta di un segnale di clock, cioè di un segnale comune che permette di sincronizzare la trasmissione con ala ricezione. Naturalmente dal punto di vista locale il clock essite da entrambi calcolatore e perifierica per poter codificare e decodificare i bit. Sul mercato sono disponibili due connettori rispoettivamente a 9 e 25 pin.



In realtà alla comunicazione basterebbero solo 3 pin, infatti tutti gli altri sono per la fase di **handshake** tra PC e perifierica. Questo handshake è una specie di pre-protocollo che permette alle 2 parti di connettersi.
Però con l'arrivo delel porte seriali è stato necessario fare un chip che permettesse ai bus dati, dove i bit vengono mandati in parallelo, di viaggiare in modo seriale. Per questo è nato l'**UART** (Universal Asynchronous Receiver Trasmitter)

Interfaccia parallela

L'interfaccia parallela fornisce la possibilità di trasmettere 8 bit contemporaneamente. Utilizza un connettore DB25 femmina ed è nata per la trasmissione dei dati alle stampanti. Inoltre ha bisogno di un'applicazione per la trasmissione dei dati

I 25 fori sono suddivisi nel modo seguente:

- **12** sono usati per inviare dati in **uscita**
- **5** sono usati per accogliere dati in **ingresso**
- **8** sono usati come **massa**

Seriale Vs Parallela

Le principali differenze tra le interfacce seriali e parallele sono:

- **Trasmissione** dove nella parallela avviene 8 bit alla volta mentre nella seriale gli 8 bit sono mandati in fila;
- **Velocità di trasmissione** dove nelle porte paralelle è più alta e va dai 50 Kb al secondo fino ai 2 MB.
- **Lunghezza del cavo** dove nelle porte parallele il cavo di connessione non può superare i 5 mentri, mentre nella seriale la lunghezza è maggiore e senza perdita.
- **La programmazione delal porta** dove la porta seriale è gestita dall'hardware, mentre quella parallela necessita di un software apposito che getisce i registri interni.

Firewire

Tipologia di porta seriale inventata negli anni 80 per il trasporto dei dati. È il predecessore della USB e ha untraffic omassmo di 400 Mbits al secondo

USB

L'**USB** (Universal Serial Bus) è un'interfaccia di comunicazione che permette l'interconnessione tra periferiche e, allo stesso tempo, le carica. Un dispositivo più avere al massimo 127 dispositivi USB collegati contemporaneamente e questo ci fa capire quanto sia sviluppata questa tecnologia. L'USB ha subito delle evoluzioni dalla sua apparizione negli anni 90, infatti vediamo come oggi abbiamo le **USB1.1** e **USB2.0** . La prima è un tipo di USB a bassa priorità utilizzata per la gestione di periferiche che non richiedono molta banda come mouse, stampanti o scanner e hanno la stessa capacità trasmissiva di una classica USB. Le 2.0 invece sono 40 volte più veloci di una classica USB (quindi 480Mbit/s) e sono molto utilizzate per la connessione con dispositivi di lettura e scrittura come driver ottici.
Negli ultimi anni gli USB si sono evoluti fino al tipo 3.0 il quale ha portato grossi cambiamenti a questo standard. Infatti ha una trasmissione di 5Gbit/s nel caso della 3.0 e, nelle successive 3.1 e 3.2 raddoppia di volta in volta. Tutti gli USB sono retrocompatibili tranne il 3.1 e 3.2 i quali utilizzano lo standard **USB tipo C**. La sigla USB 3.0 è spesso seguita da **SS** che vuol dire Super Speed.

Bluetooth

Il **Bluetooth** è uno standard creato per la comunicazione tra dispositivi senza fili e tramite le onde radio a corto raggio. Questa tecnologia continua a essere molto sviluppata perchè è semplice ed economica. Di solito lavorano con una banda radio a 2.4Ghz e un transfer rate pari a 1Mbit

Altre periferiche

- La **testiera** è una periferica che utilizza lo standard **QWERTY** e che permette la scrittura delle parole tramite una matrice circuitale di righe e colonne. Ogni volta che un tasto viene premuto due segnali (riga e colonna) vengono inviati al microprocessore che, grazie a una mappa dei caratteri sulla ROM, codifica ogni carattere
- La **scheda video** è un componente che traduce la rappresentazione dell'immagine prodotta dal processore in un formato visualizzabile dal monitor. Negli ultimi anni le schede video hanno assunto un ruolo importante per lo sviluppo 3D e quindi sono diventate così potenti da poter sostituire i processori nella lavorazione di queste immagini.

I sistemi operativi

Sistema operativo = programma intermedio tra persona e macchina. Il sistema operativo viene lanciato e rimane in memoria per tutta l'esecuzione del pc. Un SO permette di gestire gli elementi fisici di un calcolatore e di virtualizzarli, inoltre è una piattaforma per far eseguire gli applicativi e permette di utilizzare le API
Il sistema operativo crea un sistema virtuale che può più o meno rispettare il calcolatore su cui si trova. Basta pensare agli emulatori che permettono di **emulare o virtualizzare** un sistema operativo diverso dal nostro come quello Androi per esempio. La virtualizzazione permette di utilizzare le caratteristiche della macchina sottostante e di farci eseguire sopra le applicazioni. Molte suddivisioni sono solo logiche, perchè avvengo dal sistema operativo.
In parole povere il SO permette di virtualizzare tutte le risorse del pc e garantisce un ambiente facile all'utente finale. Un SO si dice ideale quando è bello, sicuro, veloce e responsivo alla rechieste dell'utente.
I livelli di un sistema operativo sono: **Utente** e l'**Hardware**.
Per *Utente* definiamo qulla persona o sviluppatore che usa le applicazione implementate sopra. Lo sviluppatore utilizza delle librerie che chiamano direttamente funzioni messe a disposizione dal SO.
L'hardware è gestito dal sistema operativo e gestisce le risorse destinate alle applicazione.
Ora possiamo dire che il sistema operativo può essere visto come quel livello software che permette a programmi e applicazione di lavorare su macchine virtualizzate. Inoltre fornisce le *API (Application Programming interface)* , ovvero un set di procedure che funzionano indipendentemente dall'hardware della macchina che permettono interfacciamento tra utente e sistema operativo.

Storia

Il sistema operativo e il calcolatore non nascono con lo scopo di dover intergire con l'utente, ma con quello di alleggerire il carico di operaizione da far fare all'utente. vediamo come le prime generazioni di computer non permettevano all'utente di interagire, ma solo di assegnare dei **jobs**, ovvero dei programmi in batch. Con l'arrivo dell'**MS-DOS** da parte di Bill Gates e del **System 1.0** di Apple iniziamo a vedere che il target dei sistemi operativi si sposta dalle sue radice, permettendo anche all'utente di controllare i processi grazie ai sistemi di **Multitasking e time sharing** sviluppati in seguito. Questo ha permesso ai SO di poter avere dei servizi essenziali come:

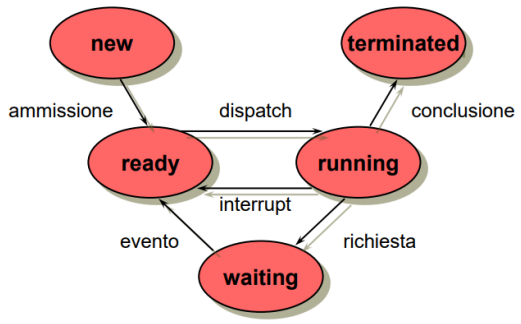
Servizi	Descrizione
Interfaccia grafica con l'utente	può essere di tipo **GUI**, quindi grafica oppure **CLI**, ovvero a riga di comando
Esecuzione dei programmi	gestione della finestra delle attività, permettendo di caricare, eseguire e terminare un programma
Comunicazione tra i processi in esecuzione	Il SO permette la comunicazione tra processo mediante memoria condivisaa oppure dei messaggi bidirezionali tra i processi
Gestione della memoria	Il SO permette la gestione degli spazi di indirizzi virtuali

Servizi	Descrizione
Gestione delle periferiche di I/O	Ha pieno controllo delle periferiche di I/O che si possono connettere alla macchina
Gestione del *File System*	Permette all'utente di gestire i file sulla macchina senza che si occupi realmente di andarli a sistemare in una parte del disco fisico
Rilevamento degli errori	Effettua un controllo periodico per far si che il sistema funzioni al massimo delle sue potenzialità senza problemi

- Il concetto del **Multitasking** si fonda sul concetto di mantenere diversi processi in memoria contemporaneamente e, quando uno ha bisogno di una risorsa, viene messo in attesa e il suo successivo viene eseguito. Questo avviene soprattutto grazie alle architetture **multi processore**.
- Con **time-sharing** definiamo una tecnologia utilizzata per le architetture a un solo processore che, grazie a un appostio hardware chiamato **timer**, interrompe i processi in esecuzione. Queste interruzioni avvengono così velocemente che all'essere umano sembra come se i processi lavorino in parallelo, invece loro vengono continuamente interrotti e ripresi secondo un *quanto di tempo*.

I processi

La prima distinzione importante è quella tra processo e programma. Il programma è la parte generica (parte passiva), mentre il processo è l'istanza che permette al programma di essere eseguito. Tutti i processi concorrono per l'uso delle risorse in esecuzione, quindi CPU, files e memorie.
NOTA BENE che ci sono molte differenze tra algoritmo, programma e processo. Con **algoritmo** definisco la sequenza di passi che consentono di risolvere un problema; con **programma** descrivi un algoritmo tramite un linguaggio che permette l'esecuzione insieme al processore; con **processo** invece, definisco una serie di eventi prodotti dall'esecuzione di un programma.
Ogni processo ha un ciclo di vita rappresentato da questa immagine



Quindi un processo nasce e si trova nello stato di **New** e si prepara ad essere eseguito nello stato di **Ready**. Una volta che la CPU prende in carico il processo abbiamo diversi risultati per il processo. La prima è quello che il processo finisce e quindi diventa **Terminated**, quindi morendo e deallocando tutte le risorse precedentemente impegnate; le altre invece implicano la necessità di dover aspettare una risora dal SO o addirittura l'esecuzione di un processo. In questo caso il processo torna nello stato di **Ready** se sta aspettando delel risorse che sono impiegate in altri processi. Nello stato di **Waiting** invece ci arriva se è in attesa di un input da parte dell'utente. Inoltre, il SO è responsabile dei

meccanismi di **sincornizzazione** dei processi, **comunicazione** tra di loro e controlla che il **deadlock** (morte per attesa)sia evitato.

Ad oggi abbiamo una tecnologia che ci permette di eseguire contemporaneamente più processi alla volta, questo è dato dalle architetture **multi-processore** che da qualche hanno hanno invaso il modno dell'elettronica. Prima infatti i calcolatori potevano eseguire un numero molto limitato di processi contemporaneamente, per esempio l'MS-DOS poteva far eseguire un solo processo per volta.

Il processo, una volta creato, può essere diviso in sezioni, ovvero

- **Codice** che contine il codice del programma;
- **Dati** dove vengono immagazzinate le variabili statiche;
- **Stack** usato per allocare dinamicamente le variabili locali usate nelle funzioni e per passare i parametri;
- **Heap** è un tipo di memoria allocata dinamicamente durante l'esecuzione del programma.

Inoltre, ogni processo è rappresentato nel SO come un blocco chiamato **PCB (Process control block)** che contiene:

- **Stato del processo**
- **Contatore del programma** che indica l'indirizzo dell'istruzione successiva
- **Registri della CPU**
- **Informazioni sullo scheduling** definisce la priorità del processo

I proccessi possono essere di due tipologie:

- **I/O Bound** che sono specializzati nell'esecuzione delle richieste da parte delle periferiche
- **CPU Bound** che invece lavorano solo con processi che richiedono molto tempo di computazione come il rendering 3D o calcoli complessi.

Quando un processo viene fermato e poi ripreso, abbiamo un cambio di contesto (**context switching**) che produce un ritardo che deve essere minimizzato

Parte importante dei processi è la **comunicazione**. Questa può avvenire in diversi modi, il più comune è quella della parentela tra processi, ovvero dove un processo padre crea un figlio con il quale può decidere se comunicare o meno. Possiamo anche avere processi **concorreti** (che concorrono alla stessa risorsa) **cooperanti**. Questi possono avere **memoria condivisa** dalla quale attingere alle risorse, oppure possono avere un **buffer** che a sua volta può essere limitato o illimitato. Il buffer è un vettore dove possono essere messe o prese le risorse. I processi possono anche inviare **messaggi** tra di loro. Questi messaggi sono chiamati IPC (**InterProcess Communication**) che permettono la sincronizzazione senza usare aree di memoria condivise. Lo scambio di questi messaggi avviene tramite funzioni di **send e receive**

Kernel

Con **Kernel**, dall'inglese nocciolo, definiamo lo strato più interno del calcolatore che divide la parte fisica con quella virtuale e che ne permette la comunicazione e convivenza. Per convenzione diciamo che il kernel è una parte del sistema operativo che fornisce alle applicazioni l'accesso sicuro all'hardware. Affinchè un kernel sia detto **monolitico** (ben integrato ed efficiente), ha bisogno che ogni componente del calcolatore sia ben integrata. Questo perechè ogni componente ha un relativo modulo che viene aggiunto al kernel ospitante.

Thread

I nuovi sistemi operativi, più che sui processi , si basano sui thread o **flussoi di controllo**. un thread è l'unità di base nell'uso della CPU. I thread sono delle sotto sezioni che il processo produce a cui il processore alloca il tempo di esecuzione. Ogni processo può creare più thread che comunicano tra di loro grazie alla memoria condivisa. I thread hanno molti benefici come il tempo di risposta molto bassa, risorse condivise e non richiede sforzi da parte del compilatore per la creazione e distruzione

Lo scheduling

Lo **scheduler** della CPU è il meccanismo sul quale si basano i sistemi multitasking dove più processi vengono caricati in memoria. In linea di massima lo scheduler decide quali processi vengono presi da una pila **pseudo immaginaria** chiamata **pila ready**, ovvero l'insieme dei processi pronti ad essere eseguiti. Questa pila ready (ready queue) è una struttura dati in cui lo scheduler gestisce i processi in base a un algoritmo. Esistono diversi tipi di algoritmo, questo perchè ogni algoritmo ha dei parametri diversi. I parametri pincipali che un algoritmo deve massimizzare sono: massimizzare l'**utilizzo della CPU**, massimizzare il **throughput** (numero di processi completati nell'unità di tempo), minimizzare il **tempo di turnaround** (tempo di vita del processo), minimizzare il **tempo di attesa** e **tempo di risposta**. Prima di entrare nel vivo degli algoritmi è bene definire la differenza tra **preemtive** e **non preemptive**. Si dice che un algoritmo è preemptive quando un processo nuovo che ha delle caratteristiche con priorità maggiore di quello attualmente in sviluppo sovrasta il processo in sviluppo. Invece con non preemptive definiamo un processo che una volta iniziato il suo svolgimento non può essere spodestato.

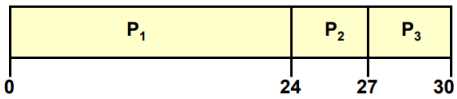
FIFO

L'algoritmo **FIFO (First in first out)** o FCFS (First come first serverd) è l'algoritmo più semplice che lavora come una pila, quindi il primo processo che si mette in fila sarà il primo ad essere sviluppato. È un algoritmo molto semplice ma che ha tante criticità, soprattutto se in testa ci sono dei processi molto che richiedono molto tempo. Se un processo risulta troppo lungo i processi successivi potrebbero morire

ESEMPIO:

Processo	Burst Time
P_1	24
P_2	3
P_3	3

Supponiamo che l'ordine di arrivo sia P_1 , P_2 , P_3



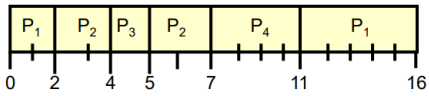
Il tempo di attesa è $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
Il tempo di attesa medio è $(0 + 24 + 27)/3 = 17$

SJF

L'algoritmo **SJF (Shortest job first)** è un algoritmo molto selettivo che privilegia i processi brevi. Il suo difetto principale sta nella versione preemtive dove un processo molto lunghe morirà sicuramente perchè non verrà mai eseguito fino alla fine (starvation).

Esempio:

Processo	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

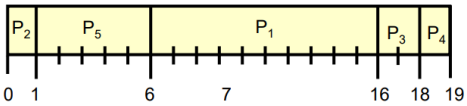


Average waiting time = $(9 + 1 + 0 + 2)/4 = 3$

Però, se aggiungessimo una priorità all'algoritmo SJF vedremo che il problema della starvation si super, arrivando ad avere un tempo di attesa medio non troppo elevato.

Esempio (processi arrivati tutti al tempo 0):

Processo	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2



Average waiting time = $(6 + 0 + 16 + 18 + 1) = 8,2$

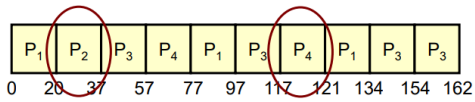
Anche qui però sorge un problema, ovvero che nel caso avessimo un SJF con priorità preemptive i processi con priorità molto bassa non verranno mai eseguiti a prescindere, rendendo inutile tutto il concetto della lunghezza del processo. A questo si può ovviare con un meccaniscmo di **aging** (invecchiamento), andando ad aumentare progressivamente la priorità dei processi *vecchi*.

Round Robin

Algoritmo pensato per i sistemi basati sul time sharing. Prende come esempio l'FCFS ma con l'aggiunta di un **timer** (quanto di tempo) che limita l'esecuzione di ogni singolo processo. Nonostante presenti molti **context switching** l'agoritmo si presta meglio dei precedenti sotto ogni punto di vista.

- Esempio (quanto = 20 ms):

Processo	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24



Code multiple

Questa soluzione permette di dividere la ready queue in tante code con priorità diverse, differenziando i processi utili alla macchina e quelli generati dall'utente. Ogni coda ha il suo algoritmo e quindi un comportamento diverso

Concorrenza

Tutti i processi possono essere eseguiti in due modi principali, ovvero **concorrente o parallelo**. Il primo, come dice il nome, ha i processi che competono per utilizzare la risorsa, ed è anche il tipo di esecuzione più comune tra i processi, dato che molto spesso più processi hanno bisogno della stessa risorsa. Con parallelo invece definiamo il tipo di sviluppo dei processi su più unità funzionali o meglio su processori **multicore**.
In ogni caso, abbiamo una sezione critica nella gestione della risorsa ai processi, ovvero quella di garantire l'integrità dei dati condivisi da più processi. Per questo abbiamo bisogno di un meccanismo di **sincronizzazione** che permetta l'interazione tra di loro. Questa è stato pensato subito dopo aver visto il modello **Race condition**, ovvero una specie di gara dove il primo processo o thread ad arrivare alla risorsa la poteva usare. Possiamo immaginare la gravità dei problemi generati da questo modello, come starvation e dead-lock. Per questo dobbiamo analizzare il codice e limitare le porzioni di codice che condividono memoria ai processi e gestire le entrate ed uscite. Questa zone viene chiamata **sezione critica**. L'accesso all'area critica però deve avere delle caratteristiche ben definite ai fini dell'efficienza, come la **mutua esclusione** e la **liveness** dei processi. Con mutua esclusione o mutex definiamo la capacità di una risorsa di essere usata da un solo processo o thread alla volta; con liveness invece dobbiamo garantire la risorsa anche ai processi in attesa senza farli morire. Inoltre, le sezioni critiche non hanno priorità, quindi non privilegiano thread o processi e, non possiamo definire quanto tempo è necessario a un processo per uscire dalla sezione critica.
Un nuovo metodo per entrare nella sezione critica è quello del **lock** (lucchetto) che , come dice il nome, è un tipo di variabile (flag) presente all'inizio della sezione critica. Può assumere valore 0 o 1, infatti a valore 0 il processo entra e imposta il lock a 1, mettendo in attesa il processo successivo. Questo processo continua a fare la chiamata al lucchetto per vedere se il suo valore è ancora 1 o se è diventato 0 fino a quanto la risorsa non è libera.
Il metodo più efficace però è il **semaforo**, ovvero una variabile intera alla quale si può accedere solo tramite 2 funzioni, la **wait** e la **signal**. La prima controlla che il semaforo S sia maggiore uguale a 0, in caso positivo lo decrementa;la signal invece incrementa di uno il semaforo S. Questo rende i semafori dei meccanismi di segnalazione tra processi per la condivisione de una risorsa, mentre un mutex

rappresenta una protezione della risorsa condivisa.
Infine abbiamo i **monitor** (o controllori), ovvero una struttura dati che permette la mutua esclusione e la sincronizzazione su strutture dati condivise da più thread. Questi monitor garantiscono che un solo thread per volta venga eseguito e non distinguono per priorità avendo una coda di tipo FIFO

Problematiche

La concorrenza ha però dei problemi che andrebbero superati con delle accortezze a livello di codice. Il primo problema è il **Busy waiting**, ovvero l'attesa attiva dove un thread o processo continua a mandare richieste in attesa che la condizione di accesso si verifichi. Questo comporta un enorme spreco di CPU mentre potrebbe essere sospeso e inserito nella waiting list.
La **starvation**, o morte di fame, si verifica quando un thread è in attesa di una risorsa che non gli sarà mai data perchè ha privilegi minori ai thread in arrivo.
Il **deadlock** si ha invece quando 2 o più thread aspettano indefinitamente un evento che può avvenire solo se uno di loro rilasci una risorsa. Quindi il problema nasce dal fatto che ogni thread ha una risorsa che serve anche ad altri thread, così da far morire tutte le richieste.

Gestione della memoria

Per far eseguire un qualsiasi programma sulla nostra macchina è necessario un legame (**binding**) tra gli **indirizzi logici** e **fisici** della memoria. Con **spazio logico** definiamo l'insieme degli indirizzi logici usati dal programma, mentre con **spazio fisico** definiamo l'insieme degli indirizzi fisici usati dal programma.
Adesso andiamo a vedere nel dettagli questi indirizzi. In generale tutti i programmi eseguiti vivono nella memoria centrale.

- L'**indirizzo fisico** è quello spazione nella memoria centrale che individua in modo univoco una parola (elemento) in esso contenuta. Questo spazio è un vettore lineare che parte da 0 e arrivo fino all'ultimo spazio indirizzabile della memoria.
- L'**indirizzo logico** è invece un indirizzo astratto esitente solo all'interno di un processo e ne rappresenta lo spazio indirizzabile. In generale questo indirizzo viene assegnato dalla CPU quando sviluppa un processo.

Per far avvenire una corrispondenza tra indirizzo logico e fisico abbiamo tre modalità ben distinte, ovvero la **compilazione, caricamento e esecuzione**. Questa rilocazione si dice **statica** quando il **loader** (parte del SO che carica i programmi) trasforma l'indirizzo logico in fisico. Si dice **dinamica** invece quando l'indirizzo fisico viene calcolato in fase di esecuzione del programma. Questa fase di rilocazione è svolta da un dispositivo chiamato **Memory-management Unit (MMU)** tramite un registro che identifica i processi con un codice inserito nel PCB.

Tutti gli esempi elencati sopra sono basati su architettura a singolo processore, ma se iniziamo a vedere delel architetture odierne, e quindi moltiprogrammate, vediamo che ci sono delel differenze sostanziali nell'assegnazione della memoria fisica. Il gestore della memoria è un componente che si occupa di allocare la memoria, ma soprattutto di proteggere lo spazio di indirizzamento dei processi deallocati. La deallocazione di processi porta delel criticità, come quella della gestione della memoria e

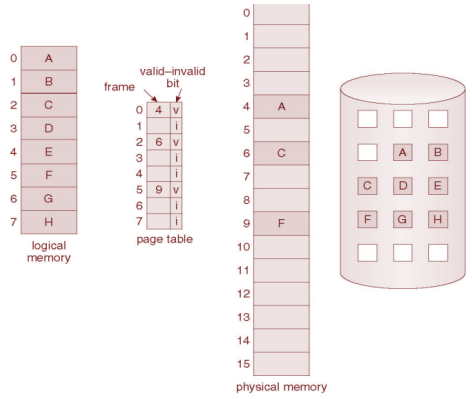
il recupero della memoria deallocata. Questo perchè la RAM è un vettore e, una volta rimosso un processo avremo degli indici vuoi tra due processi.

L'allocazione è gestita da due politiche principali, la **contigua** e la **non contigua**. La contigua permette l'allocazione di un proesso in un'area di memoria continua, senza frammentare il processo. Questo tipo di allocazione ha 4 possibili tipologie, ovvero:

- **Monitor monoprocesso**: metodo utilizzato nell'MS-DOS che divideva la memoria in due aree, una dedicata al sistema operativo e l'altra per le applicazioni.
- **Partizione statica**: suddivisione della memoria in partizioni di dimensioni fisse. Ogni partizione monitorata da una **TDP**, ovvero una tabella delel parttizioni che controlla lo stato. Un problema potrebbe essere quello di non avere una partizione libera per allocare il processo e quindi si potrebbe finire in una dead lock. Per questo esiste il sistema di **swapping** tra i processi, ovvero dove un processo viene forzato a lasciare il posto per uno più grande. Nel momento del cambiamento il processo viene salvato su disco, quindi diventa molto più lenta dei tempi medi di un accesso alla memoria centrale. Il vero problema è la **partizione interna**, dove andiamo a perdere bit importanti se allochiamo un processo più piccolo della partizione ospitante.
- **Allocazione dinamica**: la suddivisione della memoria avviene dinamicamente, con numero e grandezza di partizione variabile. Con questo tipo di partizione ho la necessità di avere allo stesso modo una tabella degli stati e una lista delle aree libere. Nonostante questo metodo risolve molti problemi, rimana la **partizione esterna**, dove tra un processo e l'altro abbiamo degli spazi (più o meno piccoli) did memoria non allocata. Per risolvere questo problema si agisce con la **compattazione**, ovvero dove tutte le aree di memoria vengono unite in una sola, ma per fare questo abbiamo bisogno di fermare tutti i processi in esecuzione.
- **Segmentazione**: questo metodo divide i programmi in più parti, allocandolo in modo contiguo. Ogni segmente porta con se un descrittore che viene registrato nella **TDS**, ovvero la tabella dei segmenti.

L'allocazione non contigua invece, si basa sulla **paginazione**, ovvero la divisione della memoria fisica in blocchi di dimensione fissa dette pagine. Queste pagine sono gestite da una tabella delle pagine, ma anche loro hanno dei difetti come la frammentezione della pagina, dove non tutta la memoria della pagina è utilizzata.

Tutte queste tecniche sono basate sulla RAM, ma se invece avessimo una memoria **virtuale** in grado di aumentare lo spazio a nostra disposizione? Questo è fattibile perchè a prescindere dalla tecnica utilizzata il codice del programma rimane presente nella memoria did massa e quindi possiamo giostrare il nostro processo tra locazione logica e fisica. Questa memoria è implementata con una tecnica chiamata **paginazione su richiesta**, combinando tecniche di paginazione e swapping. Per rendere effettive queste memorie abbiamo bisogno di una memoria di appoggio molto veloce e di un componente chiamato **pager** che permette difare funzioni come il carico (**swap-in**) e lo scarico (**swap-out**) delle pagine. Questo perchè le pagine vengono prima preparate e poi inviate alla memoria centrale.



Parliamo di **Page fault** quando, data una tabelle delle pagine, non abbiamo nessun processo pre-allocato in una specifica pagine e quindi bisogna richiederlo alla memoria virtuale. La tabella presenta bit di verità per la presenza delle pagine già caricate in memoria virtuale. Il page fault bisogna evitarlo, perchè rallenta di molto il tempo di accesso, dovendo prima passare da una memoria intermediara. Possiamo utilizzare degli algoritmo di swapping per modificare velocemente le pagine presenti nella tabelle delle pagine, come il FIFO oppure un algoritmo di predizione basato sulla successione dei processi chiamato **LRU** (Last recently used)

DA FARE SLIDE 5 PARTE 2