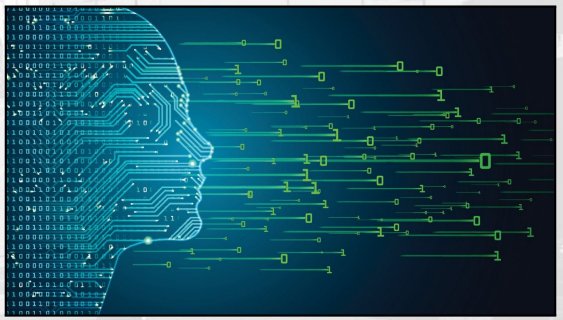
Programmazione Informatica
4. Architettura del calcolatore

Riccardo Zamolo
rzamolo@units.it

Università degli Studi Trieste
Ingegneria Civile e Ambientale



A.A. 2020-21

- Abbiamo visto che l'oggetto dell'informatica è l'elaborazione *automatica* delle *informazioni* attraverso il calcolatore, istruito attraverso un *programma*.
- Si è assunto quindi che il calcolatore è *programmabile*: può risolvere problemi diversi tramite una "semplice" riprogrammazione che non riguarda la modifica dell'*hardware* del calcolatore stesso.
- Un *programma* è perciò una lista ordinata e finita di *istruzioni* che vanno eseguite in sequenza: ogni istruzione può essere pensata come un ordine che viene impartito alla macchina virtuale del linguaggio di programmazione utilizzato (programmazione *imperativa*).
- *Architettura di von Neumann*: modello di computer a *programma memorizzato* costituito da:
  - una memoria indirizzabile, che contiene sia il programma che i dati;
  - un'unità logico-aritmetica, che possa elaborare il contenuto della memoria;
  - un program counter, ossia un registro che indica l'indirizzo di memoria dell'istruzione che deve essere eseguita.

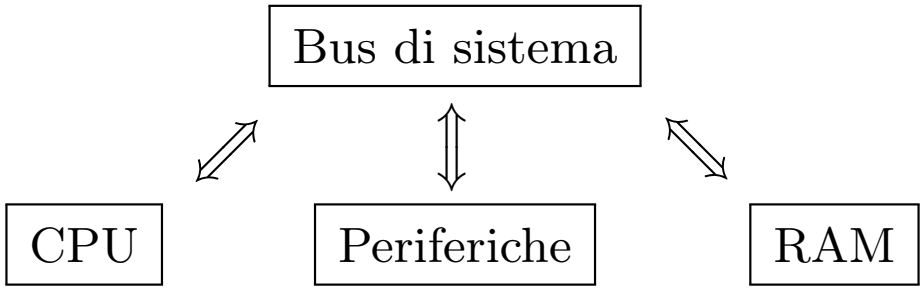
- In un computer a programma memorizzato, un programma viene eseguito secondo il seguente *ciclo* concettuale (*ciclo fetch-decode-execute*):

  PC ← 0
  repeat
    istruzione ← memoria[PC]
    decode(istruzione)
    fetch(operandi)
    execute
    PC ← PC+1
  until istruzione=STOP

- PC è il program counter, un numero intero che individua in memoria l'istruzione da eseguire e cresce di un'unità ad ogni esecuzione del ciclo;
- istruzione ← memoria[PC] indica che il contenuto della posizione di memoria indirizzata da PC viene caricato nel registro "istruzione" (instruction fetch) a disposizione dell'unità logico-aritmetica;
- decode(istruzione) indica che il contenuto del registro "istruzione" viene decodificato, cioè viene interpretato che tipo di istruzione eseguire;
- fetch(operandi) indica che gli operandi necessari all'esecuzione dell'istruzione vengono caricati in appositi registri;
- execute indica che l'istruzione viene effettivamente eseguita.

Avendo in mente il precedente ciclo fetch-decode-execute, un computer con architettura di von Neumann prevede più in generale i seguenti elementi essenziali:

- CPU (Central Processing Unit): esegue le istruzioni secondo il ciclo fetch-decode-execute;
- RAM (Random Access Memory) o memoria centrale: memoria indirizzabile che contiene sia il programma che i dati;
- Periferiche di Input/Output (I/O): permettono lo scambio di informazioni tra il computer e l'esterno;
- Bus di sistema: permette lo scambio di dati tra i precedenti elementi:



La CPU (Central Processing Unit) è l'insieme dei seguenti elementi:

- Registri: memorie di piccola dimensione ma molto veloci, servono a contenere i dati necessari alle operazioni richieste dall'istruzione da eseguire. Sono accessibile direttamente dalla CPU. Il numero di bit di una CPU (32/64 bit) indica la dimensione in bit dei registri.
- Unità di controllo (CU, Control Unit): controlla e coordina l'attività di tutte le componenti della CPU. In particolare, è responsabile del prelievo delle istruzioni da eseguire e dei relativi operandi (fetch), della decodifica delle istruzioni (decode) e dell'invio dei segnali di controllo agli altri elementi necessari all'esecuzione delle istruzioni (execute).
- Unità logico-aritmetica (ALU, Arithmetic Logic Unit): esegue le operazioni aritmetiche (somma/prodotto di due registri, ecc.) e logiche (controlla se registro A $>$ registro B, ecc.).
- Unità di calcolo in virgola mobile (FPU, Floating Point Unit): unità aritmetica specializzata nell'esecuzioni di calcoli in virgola mobile. È anche chiamato coprocessore matematico.
- Clock: scandisce il ritmo temporale delle operazioni elementari della CPU, permettendo il sincronismo delle operazioni tra i vari elementi. La frequenza di clock è quella tipicamente riportata nei dati di una CPU, in GHz.

I registri principali di una CPU sono i seguenti:

- PC (Program Counter): registro contatore che individua l'indirizzo in RAM dell'istruzione da eseguire;
- Registro istruzione (IR, Instruction Register): contiene l'istruzione, in forma codificata, che è attualmente in esecuzione dalla CPU;

- Registro indirizzi di memoria (AR, Address Register): contiene l'indirizzo in RAM interessato dalla lettura o dalla scrittura di dati;
- Registro dati di memoria (DR, Data Register): contiene i dati da leggere o da scrivere nella RAM, indirizzati dall'AR;
- Registro di controllo (CR, Control Register): specifica se i dati indirizzati in RAM dall'AR sono in lettura o scrittura;

- Registri accumulatori o di lavoro (R1, R2, ...): contengono operandi e risultati intermedi delle operazioni eseguite;
- Registro interruzioni (INTR, Interrupt Register): contiende dati relativi alla richiesta di operazioni dalle periferiche di I/O;
- Registro di stato (SR, Status Register): contiene informazioni sul risultato dell'ultima operazione eseguita dalla ALU (overflow, riporto, segno, ecc.).

L'unità di controllo (CU, Control Unit) controlla e coordina l'attività di tutta la CPU. In particolare si occupa delle seguenti operazioni:

- *fetch*: l'istruzione da eseguire viene reperita nella RAM all'indirizzo contenuto nel registro contatore PC (Program Counter) e caricata nel registro istruzione IR (Instruction Register). Questa operazione viene eseguita sfruttando i registri di memoria AR (Address Register), DR (Data Register) e CR (Control Register);
- *decode*: l'istruzione appena caricata nel IR viene interpretata/decodificata, poichè il contenuto del IR è ovviamente codificato, cioè è una sequenza di bit. Gli operandi necessari all'operazione vengono caricati dalla RAM ai registri di lavoro, sempre per mezzo dei registri di memoria, se non sono già presenti;
- *execute*: l'istruzione viene mandata in esecuzione, sovraintendendo a tutti gli scambi di dati tra registri, ALU, FPU e RAM. Il risultato viene quindi scritto in un registro di lavoro oppure in memoria attraverso gli opportuni registri.

L'unità logico-aritmetica (ALU) si occupa di:

- effettuare operazioni aritmetiche e logiche sugli operandi A e B assegnati, entrambi numeri interi (binari). Il risultato viene messo a disposizione della CU per essere indirizzato opportunamente (verso un dato registro o un dato indirizzo in RAM). Il tipo di operazione logico-aritmetica da eseguire è specificato da un *codice operativo*.
  Le operazioni aritmetiche possono essere:
  - somma di A e B con o senza riporto. Nel primo caso il riporto è preso dal registro di stato (SR, Status Register);
  - differenza tra A e B con o senza riporto, sempre preso dal SR;
  - complemento a 2 (cambio del segno) di uno dei due operandi;
  - incremento o decremento di un'unità di uno dei due operandi.

  Le operazioni logiche possono essere:
  - operazioni Booleane (AND, OR, XOR) effettuate bit a bit sugli operandi;
  - complemento a 1 (inversione di tutti i bit) di uno dei due operandi;
  - shift a destra o a sinistra dei bit di uno dei due operandi, con o senza circolarità.
- riportare nel registro di stato (SR, Status Register) l'esito, codificato, delle operazioni eseguite. Dal contenuto dello SR si può quindi capire se l'ultima operazione eseguita dalla ALU ha dato luogo ad overflow, riporto, risultato di segno positivo o negativo, ecc.

La *memoria cache* è un particolare tipo di memoria caratterizzata da un'elevata velocità di accesso, dalle 10 alle 100 volte maggiore di quella della RAM, ma dimensione minore. Il suo impiego è giustificato dalle seguenti osservazioni:

- il trasferimento di dati tra RAM e CPU è "lento" poichè essi sono due elementi fisicamente separati e connessi attraverso un terzo elemento, ossia il bus di sistema;
- molto spesso è richiesto l'accesso multiplo a dati ed istruzioni di frequente utilizzo che non possono risiedere nei registri di lavoro per motivi di spazio.

La memoria cache contiene quindi copia di dati ed istruzioni di frequente utilizzo, presenti nella RAM, ed alle quali la CPU può accedere velocemente senza dover accedere alla RAM. La memoria cache può essere di vari livelli a seconda delle dimensioni e del tempo di accesso:

- cache di primo livello o L1: è interna alla CPU, è la più veloce ed è di dimensione ridotta (8-64 kB);
- cache di secondo livello o L2: è solitamente esterna alla CPU, è più lenta di L1 ma ha dimensione maggiore (256 kB-8MB).

Quando un dato o un'istruzione vengono richiesti dalla CPU, questi vengono ricercati prima nella memoria cache (nell'ordine prima L1, poi L2): se sono presenti vengono copiati nei registri (*cache hit*), altrimenti vengono copiati dalla RAM alla cache (*cache miss*).

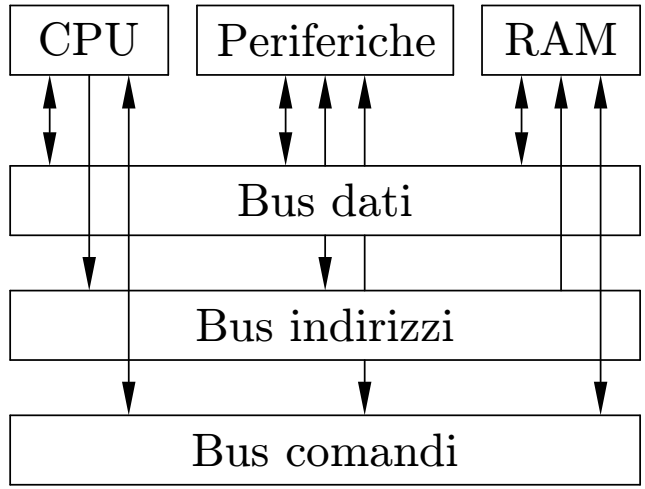
- La memoria centrale contiene le istruzioni, cioè il programma, ed i dati necessari all'elaborazione.
- È *volatile*: i dati contenuti vanno persi in assenza di alimentazione.
- È di tipo RAM (Random Access Memory): il tempo necessario ad accedere ad un indirizzo di memoria non dipende dalla posizione fisica occupata da quell'indirizzo.
- È concettualmente strutturata come una sequenza di celle, dette *parole* (o *word*) di dimensione prefissata, sempre multipla del byte: 8, 16, 32 o 64 bit (1, 2, 3 o 4 byte). Ogni cella w_i è univocamente identificata dal suo indirizzo, rappresentato da un numero intero i:

Celle di memoria:	w_0	w_1	.	.	w_i	.	.	w_N
Indirizzo:	0	1	·	·	i	·	·	N

- L'indirizzo i per l'accesso alla RAM, in lettura o scrittura, è contenuto nel registro indirizzi di memoria (AR), ovviamente codificato in base binaria.
- Se k è il numero di bit del AR, possono essere indirizzate 2^k celle di memoria w_i $(0 \leq i \leq N = 2^k - 1)$.

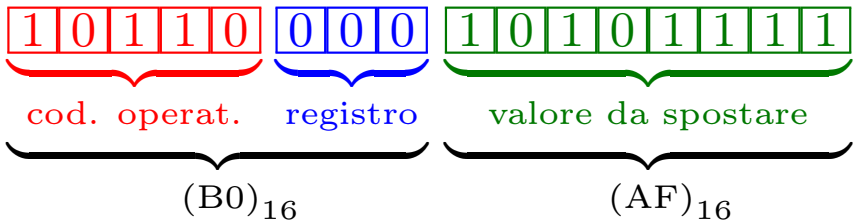
- È l'elemento che collega CPU, RAM e periferiche e permette lo scambio di dati tra questi.
- Collega due unità funzionali alla volta: una trasmette e l'altra riceve (modalità *master-slave*).
- Il trasferimento di dati viene sotto il controllo della CPU.
- Può essere *seriale* o *parallelo*: nel primo caso i bit vengono trasmessi in maniera sequenziale su un'unica linea; nel secondo caso vi sono più linee che trasmettono i bit contemporaneamente aumentando la velocità di trasmissione, ma ovviamente necessitano di opportuna sincronizzazione.

- *Bus dati*: utilizzato dalla CPU per trasmettere dati dalla RAM al registro dati o alle periferiche, e viceversa;
- *Bus indirizzi*: utilizzato dalla CPU per trasmettere il contenuto del registro indirizzi alla RAM o alle periferiche di per selezionare il dispositivo da usare;
- *Bus comandi*: utilizzato dalla CPU per inviare alla RAM o alle periferiche i segnali di comando/sincronizzazione tra i vari elementi.



- La CPU reperisce in memoria le istruzioni (fetch), le decodifica (decode) e le esegue (execute).
- L'insieme delle istruzioni che una CPU può eseguire costituisce il *linguaggio macchina* ed è costituito da un numero limitato di istruzioni; è il livello più basso al quale si può programmare un computer.
- Un'istruzione in linguaggio macchina è costituita da:
  - *codice operativo*: sequenza di bit che identifica in maniera univoca una specifica operazione. Le operazioni possono essere:
    - logico-aritmetiche, ossia tutte le operazioni effettuate dalla ALU, sui numeri interi, o dalla FPU, sui numeri in virgola mobile;
    - di trasferimento dati tra RAM, registri e periferiche;
    - di controllo, per spostare l'esecuzione in un altro punto del programma sotto determinate condizioni.
  - *operandi*: gli operandi dell'operazione da eseguire, identificata dal codice operativo, espressi in termini binari diretti o di indirizzi in RAM, nei registri o delle periferiche. Il numero degli operandi è tipicamente compreso tra 1 (es. salto dell'esecuzione ad un dato indirizzo in RAM), 2 (es. somma di due registri) e 3 (es. comparazione di due registri con scrittura del risultato in un terzo registro).
- Un programma in linguaggio macchina è quindi una sequenza ordinata di codici operativi ed operandi, espressi in termini binari.

- Programmare direttamente in linguaggio macchina, cioè in termini di sequenze binarie, non è ovviamente agevole.
- Nel linguaggio *assembly* i codici operativi e gli operandi vengono sostituiti da nomi simbolici facilmente riconoscibili e memorizzabili dall'uomo, mentre altre caratteristiche di sintassi di questo linguaggio permettono ulteriori semplificazioni nella scrittura o nella lettura di un programma da parte dell'uomo. Per esempio, l'istruzione codificata dal codice operativo binario 10110 viene interpretata dalle CPU x86 come lo spostamento nel registro identificato dai seguenti 3 bit degli 8 bit successivi. La sequenza binaria



  rappresenta quindi l'istruzione di spostamento del valore $(10101111)_2 = (\text{AF})_{16} = 175$ nel registro identificato dalla sequenza binaria 000, registro chiamato AL. In linguaggio assembly questa istruzione diventa:

```
MOV AL, AFh
```

- Vi è una corrispondenza stretta tra linguaggio assembly e linguaggio macchina, cioè l'insieme di istruzioni che una CPU con una determinata architettura può eseguire. Sono perciò detti linguaggi a *basso livello*.

- Nonostante programmare in assembly risulta più agevole che scrivere in linguaggio macchina, usare il linguaggio assembly risulta comunque, in generale, troppo dispendioso e scomodo per i seguenti motivi:
  - richiede di conoscere a fondo l'architettura della CPU (istruzioni disponibili, registri, indirizzamenti, ecc.)
  - essendo le istruzioni fortemente legate alla specifica architettura di CPU, ogni programma va scritto per uno specifico tipo di CPU e non può essere portato su architetture diverse;
  - è un linguaggio ad un livello troppo basso per poter risolvere efficacemente generici problemi pratici.
- In alcuni ambiti molto specifici il linguaggio assembly è stato usato fino a pochi anni fa per ottenere il massimo delle prestazioni, su parti specifiche di un programma, per computer con architetture particolarmente "difficili" (es. vecchie console per videogiochi).
- Per colmare il gap tra i linguaggi a basso livello ed esigenze di programmazione pratica, sono nati i liguaggi di programmazione ad *alto livello*.
- Nei linguaggi di programmazione ad alto livello ci si astrae dai dettagli costruttivi della CPU e del computer (es. accessi ai registri, alla RAM, ecc.), in favore di un linguaggio più naturale per l'uomo. Questo passaggio prevede l'introduzioni di nuovi elementi astratti, non più legati all'architettura del computer, come *variabili*, *oggetti*, espressioni complesse, *funzioni*, *cicli*, ecc.

- Siccome la CPU può eseguire ovviamente solo istruzioni scritte in linguaggio macchina, i programmi scritti con un linguaggio ad alto livello devono essere "tradotti" in qualche modo in linguaggio macchina. Questa operazione può essere fatta principalmente in due modi:
  - *compilazione:* il programma origine, scritto in linguaggio ad alto livello, viene sottoposto ad una complessa serie di operazioni (processamento, analisi, ottimizzazione, ecc.) che portano alla generazione di un programma eseguibile, scritto perciò in linguaggio macchina, che può essere eseguito in un secondo momento. Ogni architettura di CPU richiede perciò il suo *compilatore*, che è a sua volta un programma.
  - *interpretazione:* ogni dichiarazione del programma origine viene interpretata e direttamente eseguita dall'*interprete* ogni volta che questa viene incontrata durante l'esecuzione. Non viene generato nessun eseguibile: l'esecuzione è possibile solo in presenza del programma interprete.
- Il linguaggio assembly viene tradotto in linguaggio macchina da un programma detto *assembler*, che non è nè un compilatore nè un interprete.
- Un particolare linguaggio di alto livello non sottintende tassativamente uno dei due precedenti tipi di traduzione in linguaggio macchina, in quanto i linguaggi ad alto livello nascono proprio per astrarsi dal linguaggio macchina. Tuttavia si parla di *linguaggi compilati* o di *linguaggi interpretati* intendendone l'implementazione tradizionale. Alcuni esempi:
  - linguaggi compilati: FORTRAN, C, C++;
  - linguaggi interpretati: Python, MATLAB; in questo caso i programmi vengono chiamati *script*.

- Calcolo della successione di Fibonacci $F_n = F_{n-1} + F_{n-2}$:

Assembly

```
; A small program that calculates and prints terms of the Fibonacci series

      maxTerms    equ 15000     ; number of terms of the series to
calculate

; Number digits to use.

       digits          equ (maxTerms*209+1673)/1000

; this is just the number of digits for the term counter
      cntDigits   equ 6   ; number of digits for counter
       org     100h            ; this is a DOS com file

main:
; initializes the two numbers and the counter.  Note that this assumes
; that the counter and num1 and num2 areas are contiguous!

      mov    ax,'00'             ; initialize to all ASCII zeroes
      mov    di,counter          ; including the counter
      mov    cx,digits+cntDigits/2     ; two bytes at a time
      cld                ; initialize from low to high memory
      rep    stosw        ; write the data
      inc    ax           ; make sure ASCII zero is in al
      mov    [num1 + digits - 1],al ; last digit is one
      mov    [num2 + digits - 1],al ;
      mov    [counter + cntDigits - 1],al

      jmp    .bottom             ; done with initialization, so begin

.top
      ; add num1 to num2
      mov    di,num1+digits-1
      mov    si,num2+digits-1
      mov    cx,digits    ;
      call   AddNumbers   ; num2 += num1
      mov    bp,num2             ;
      call   PrintLine    ;
      dec    dword [term] ; decrement loop counter
      jz     .done        ;

      ; add num2 to num1
      mov    di,num2+digits-1
      mov    si,num1+digits-1
      mov    cx,digits    ;
      call   AddNumbers   ; num1 += num2
.bottom
      mov    bp,num1             ;
      call   PrintLine    ;
      dec    dword [term] ; decrement loop counter
      jnz    .top         ;
.done
      call   CRLF         ; finish off with CRLF
      mov    ax,4c00h     ; terminate
      int    21h          ;

AddNumbers:
      std                 ; go from LSB to MSB
      clc                 ;
      pushf               ; save carry flag
.top
      mov    ax,0f0fh     ; convert from ASCII BCD to BCD
      and    al,[si]             ; get next digit of number2 in al
      and    ah,[di]             ; get next digit of number1 in ah
      popf                ; recall carry flag
      adc    al,ah        ; add these digits
      aaa                 ; convert to BCD
      pushf               ;
      add    al,'0'       ; convert back to ASCII BCD digit
      stosb               ; save it and increment both counters
      dec    si           ;
      loop   .top         ; keep going until we've got them all
      popf                ; recall carry flag
      ret                 ;

IncrementCount:
      mov    cx,cntDigits ;
      mov    di,counter+cntDigits-1
      std                 ; go from LSB to MSB
      stc                 ; this is our increment
      pushf               ; save carry flag
.top
      mov    ax,000fh     ; convert from ASCII BCD to BCD
      and    al,[di]             ; get next digit of counter in al
      popf                ; recall carry flag
      adc    al,ah        ; add these digits
      aaa                 ; convert to BCD
      pushf               ;
      add    al,'0'       ; convert back to ASCII BCD digit
      stosb               ; save and increment counter
      loop   .top         ;
      popf                ; recall carry flag
      ret                 ;
```

- Calcolo della successione di Fibonacci $F_n = F_{n-1} + F_{n-2}$:

C

```c
#define Nmax 15000
int n , F[Nmax] ;
F[0] = 0 ;
F[1] = 1 ;
for ( n = 2 ; n < Nmax ; n++ ) {
     F[n] = F[n-1] + F[n-2] ;
}
```

MATLAB

```
F = [ 0 ; 1 ] ;
for n = 3 : 15000
     F(n) = F(n-1) + F(n-2) ;
end
```

- In assembly questo semplicissimo calcolo diventa già proibitivo.
- In C si è dovuto dichiarare il tipo (`int`) e la lunghezza (`Nmax`) dell'*array* `F`, che devono essere noti al momento della compilazione.
- In MATLAB non è stato (apparentemente) necessario dichiarare nè il tipo nè la lunghezza dell'array `F`. Si occuperà poi MATLAB, al momento dell'esecuzione (interpretazione) dello script, di allocare (dispendiosamente...) ad ogni iterazione del ciclo *for* lo spazio in memoria necessario a `F`.
- In generale i linguaggi interpretati sono più "permissivi" e permettono una più rapida ed agevole scrittura dei programmi, che risultano oltretutto più compatti e leggibili. Lo svantaggio da pagare è la velocità d'esecuzione, minore di quella dei linguaggi compilati.

- Le periferiche di Input/Output (I/O) permettono lo scambio di informazioni tra il computer e l'esterno.
- Le più comuni sono banalmente la tastiera, il mouse, la scheda video e la scheda audio, la stampante, ecc.
- Poichè la velocità delle periferiche è solitamente minore di quella della CPU, si utilizzano dei *buffer* di memoria disposti nella periferica per mantenere temporaneamente i dati da trasferire, permettendo alla CPU di eseguire nel frattempo altri processi.
- Un'importante categoria di periferiche è quella dei dispositivi di memorizzazione secondaria o di massa. Sono in grado di memorizzare in maniera permanente grandi o grandissime quantità di dati. Esempi più comuni:
  - Hard Disk Drive (HDD);
  - Solid-State Drive (SSD);
  - dispositivi di archiviazione ottica (CD, DVD, Blu-ray);
  - memory card e chiavi USB.
- Altre periferiche: porte di Input/Output (porta *parallela*, porta *seriale*, USB, porte a radiofrequenza come Bluetooth/Wi-Fi, ecc.).

- Disco rigido o disco fisso (Hard Disk Drive, HDD):
  - è costituito da uno o più *dischi* rotanti rivestiti di materiale magnetico;
  - le informazioni in forma binaria vengono memorizzate sulla superficie del disco nella forma di diverse direzioni di magnetizzazione di opportune aree;
  - la lettura/scrittura su ciascun disco avviene ad opera di *testine* che si muovono parallelamente alla superficie del disco a piccolissima distanza da esso, ma senza mai toccarlo;
  - le testine sono in grado di acquisire le variazioni di campo magnetico di aree contigue del disco, in lettura, o magnetizzare una determinata area del disco, in scrittura, mediante opportuna circuiteria elettronica;
  - la disposizione dei dati sui dischi è organizzata geometricamente secondo *tracce* (cerchi concentrici), *settori* (settori circolari). Viene detto *cilindro* l'insieme delle tracce dei vari dischi equidistanti dal centro, mentre il *blocco* è l'intersezione di una traccia e di un settore.
  - è dotato di una propria elettronica digitale che sovraintende alla movimentazione del disco, alla movimentazione delle testine ed alla gestione dei segnali elettrici delle testine;
- Memoria a stato solido (Solid-State Drive, SSD):
  - si basa sulla tecnologia di memoria *flash*: le informazioni vengono memorizzate permanentemente in celle a semiconduttore;
  - non ha parti in movimento: è più affidabile, più silenziosa e più veloce degli HDD.

- È un'unità dedicata espressamente all'elaborazione grafica: è un computer vero e proprio, dotato perciò di una propria CPU (GPU, Graphics Processing Unit), una propria memoria RAM ed un proprio bus di sistema.
- L'output della scheda video è costituito da uno o più connettori di varie tipologie (VGA, Video Graphics Array; DVI, Digital Visual Interface; HDMI, High-Definition Multimedia Interface, ecc.).
- La CPU della scheda grafica, chiamata GPU (Graphics Processing Unit) è specificatamente progettata per eseguire velocemente un gran numero di operazioni elementari di tipo grafico/geometrico, in quanto l'output della scheda grafica è una sequenza cadenzata nel tempo di immagini con un gran numero di pixel.
- La GPU è quindi tipicamente costituita da un gran numero di piccole CPU indipendenti (sull'ordine del centinaio o migliaio), ognuna con una sua piccola memoria privata. La potenza di calcolo complessiva di una GPU può essere perciò molto elevata.
- *GPGPU (General-Purpose GPU) computing*: si sfruttano le elevate potenze di calcolo grafico delle GPU per effettuare calcoli di tipo generico, tipicamente nell'ambito scientifico (simulazioni di fenomeni fisici, ecc.).