Appunti di

Architettura degli Elaboratori

Rosario Terranova

v 1.1.4

http://rosarioterranova.blogspot.it/

Sommario

Struttura delle macchine da calcolo e cenni storici 2
Calcolo 2
Struttura di un Computer 2
Cenni storici 3
Unità funzionali delle macchine da calcolo 5
Tipi di calcolatori 5
Sottosistema di I/O 5
Unità di memoria 5
Processore (o CPU, unità di elaborazione centrale) 7
Concetti imperativi di base 7
Prestazioni del calcolatore 9
Rappresentazione dei numeri e aritmetica 10
Rappresentazione binaria dei numeri interi 10
Rappresentazione dei caratteri 11
Rappresentazione delle immagini 12
Rappresentazione dei suoni 12
Artimetica Maya 12
Strutture algebriche e reticoli 14
Logica matematica 14
Livello logico digitale 16
Porte logiche 16
Algebra di boole 16
Circuiti integrati 18
Circuiti per l'aritmetica 20
Accenni sugli esercizi 22
Calcolo binario 22
Codifica dei caratteri 23
Codifica delle immagini 24
Codifica dei suoni 24

Struttura delle macchine da calcolo e cenni storici

Calcolo

Un calcolo è un processo che trasforma uno o più dati in ingresso in uno o più risultati.

Questo termine deriva dal latino **calculus**, che significava "*pietruzza*". Più specificamente indicava le pietruzze utilizzate per aiutarsi nei conti o usate come contatore in un **abaco**. Quest'ultimo era uno strumento usato fin dall'antichità come ausilio nei conteggi e nell'esecuzione delle operazioni aritmetiche.

Computer

Un calcolatore, più conosciuto come **computer**, è una macchina calcolatrice in grado di eseguire automaticamente sequenze di operazioni logico-aritmetiche sui dati in ingresso (*input*) e di restituire i risultati di tali operazioni in uscita (*output*).

Nel corso della storia, l'implementazione tecnologica di questa macchina si è modificata profondamente sia nei meccanismi di funzionamento (meccanici, elettromeccanici ed elettronici), che nelle modalità di rappresentazione dell'informazione (analogica e digitale) che in altre caratteristiche (architettura interna, programmabilità, ecc.).

Al giorno d'oggi, ci si riferisce comunemente al computer come ad un dispositivo elettronico e digitale, programmabile a scopo generico costruito secondo la cosiddetta architettura di **von Neumann** ed il modello teorico-computazionale della cosiddetta **macchina di Turing**.

Struttura di un Computer

Un calcolatore digitale è una macchina che può risolvere problemi eseguendo le istruzioni che le vengono assegnate. Con il termine **programma** si intende una sequenza d'istruzioni che descrive come portare a termine un dato compito. I circuiti elettronici di un qualsiasi computer possono riconoscere ed eseguire direttamente soltanto un insieme limitato d'istruzioni semplici in cui tutti i programmi devono essere convertiti prima di poter essere eseguiti, come:

- sommare due numeri
- controllare se un numero vale zero
- copiare una porzione di dati da una parte all'altra della memoria.

L'insieme di queste istruzioni primitive forma un linguaggio, chiamato **linguaggio macchina,** attraverso il quale è possibile comunicare con il computer.
Dato che quasi tutti i linguaggi macchina sono semplici, elementari e soprattutto lunghi, risulta difficile e tedioso utilizzarli.
Nel corso del tempo, questa semplice osservazione ha portato a **strutturare** i computer come una serie di livelli di astrazione, ciascuno dei quali è costruito sulla base di quello sottostante. In questo modo la complessità degli algoritmi è gestibile più agevolmente e i computer possono essere progettati in modo sistematico e organizzato.

Approccio strutturale

Consideriamo che il programmatore vorrebbe scrivere delle determinate istruzioni per un computer; queste nuove istruzioni formano un linguaggio, che chiamiamo L1, allo stesso modo in cui le istruzioni del linguaggio macchina formano a loro volta un linguaggio, che chiamiamo L0. Il computer quindi potrà eseguire soltanto quelle istruzioni scritte nel proprio linguaggio macchina, ovvero L0.

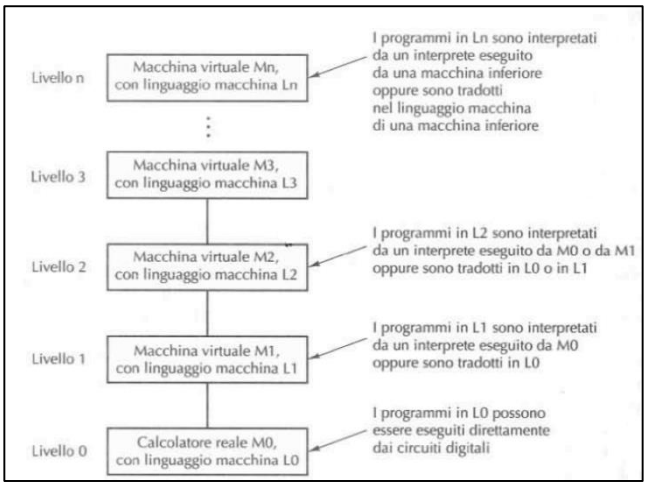
Un metodo per eseguire un programma scritto in L1 consiste nel sostituire ogni sua istruzione con un'equivalente sequenza di istruzioni in L0. Il programma che ne risulta è costituito interamente da istruzioni di L0 e può essere eseguita dal computer al posto del programma L1 originale. Questa tecnica è chiamata **traduzione**.

L'altra tecnica consiste invece nello scrivere un programma in L0 che accetti come dati d'ingresso programmi in L1; tale programma li esegue esaminando un'istruzione alla volta e sostituendola direttamente con l'equivalente sequenza di istruzioni L0. Questa tecnica, è chiamata **interpretazione** e il programma che la esegue è detto **interprete**.

La traduzione e l'interpretazione sono simili. In entrambi i metodi il computer può trattare istruzioni in L1 eseguendo le equivalenti sequenze di istruzioni L0. La differenza è che, nel caso della traduzione, il programma L1 viene, all'inizio, convertito interamente in un programma L0. Il programma L1 può quindi essere gettato via, mentre il programma L0 viene caricato nella memoria del computer per essere eseguito. Durante l'esecuzione il computer ha il controllo del nuovo programma L0.

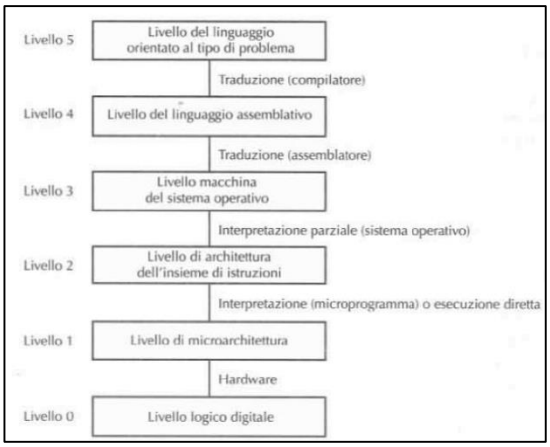
Invece di ragionare in termini di traduzione e interpretazione, è più semplice immaginare l'esistenza di un ipotetico computer o **macchina virtuale**, il cui linguaggio macchina sia proprio L1. Chiamiamo questa macchina virtuale M1 (e chiamiamo M0 La macchina virtuale corrispondente al linguaggio L0). Se ma tale macchina potesse essere costruita in modo economico, non ci sarebbe affatto bisogno del linguaggio L0, né di una macchina capace di eseguire programmi LO0. I programmatori potrebbero semplicemente scrivere i loro programmi in L1, che verrebbero eseguiti direttamente dal computer. Anche se una macchina virtuale con linguaggio macchina L1 è troppo costosa o complicata da essere costruita realmente, si possono tuttavia scrivere programmi dedicati. Questi programmi possono essere interpretati oppure tradotti da un programma scritto in

L0, eseguibile direttamente sui computer esistenti. In altre parole, si possono scrivere programmi per macchine virtuali come se queste esistessero veramente.
Per rendere la traduzione o l'interpretazione utilizzabili in pratica, i linguaggi L0 e L1 non devono essere "troppo" diversi fra loro.
L'approccio più ovvio consiste nell'inventare un nuovo insieme d'istruzioni che sia, rispetto a L1, maggiormente al linguaggio del programmatore piuttosto che a quello delle macchine. Anche questo terzo insieme forma a sua volta un linguaggio, che chiamiamo L2 (e la cui corrispondente macchina virtuale sarà M2). Questi programmi possono essere tradotti in L1 oppure eseguiti da un interprete scritto in L1.



La definizione di una serie di linguaggi, ciascuno dei quali più pratico da utilizzare rispetto al precedente, può continuare indefinitamente finché non se ne ottenga uno sufficientemente adeguato. Ciascun linguaggio utilizza il precedente come base; un computer che usa questa tecnica può quindi essere immaginato come una serie di **strati o livelli** disposti l'uno sopra l'atro.
Il livello, o linguaggio, che si trova più in basso è il più semplice per la macchina, mentre quello più in alto è il più sofisticato.
Un computer con *n* livelli pub essere interpretato come *n* distinte macchine.

Attuali macchine multilivello



La maggior parte dei moderni computer consiste di due o più livelli.

- Il livello 0, che si trova alla base, rappresenta il vero e proprio hardware della macchina, i cui circuiti eseguono i programmi scritti nel linguaggio macchina del livello 1.

Il livello **logico digitale** sono le porte (gate). Queste, pur essendo costruite utilizzando componenti analogici come i transistor, possono essere correttamente modellate come dispositivi digitali. Ciascuna porta è dotata di uno o più input digitali (segnali con valori 1 e 0) e calcola in output una semplice funzione del valori d'ingresso, per esempio AND od OR.
È possibile combinare un piccolo numero di porte per formare una memoria a 1 bit, che può memorizzare i valori 0 e 1. Combinando le memorie a 1 bit in gruppi, per esempio di 16,32 o 64, è possibile formare quelli che vengono chiamati **registri**. Ciascun registro può contenere un numero il cui valore può variare fino a un certo limite.

- Nel livello 1 troviamo il livello di **microarchitettura**. Qui vi è una memoria locale, formata da un gruppo di registri, e un circuito chiamato **ALU** (Atithmetic Logic Unit, unità aritmerico-logica), capace di effettuare semplici operazioni aritmetiche. I registri sono connessi alla ALU per formare un **percorso dati** lungo il quale questi ultimi si spostano
- Il livello 2 è costituito da quello che chiamiamo il **livello ISA** (Instruction Set Architetture Level, livello di architettura dell'insieme d'istruzioni). Ogni produttore di computer pubblica un manuale per ciascuno dei propri modelli che trattano questo livello.
- Il livello 3 è generalmente un livelle ibrido: vi è un insieme di nuove istruzioni, una diversa organizzazione della memoria e la capacità di eseguire programmi in modo concorrente, oltre a varie altre funzionalità.

Tra i livelli 3 e 4 vi è una spaccatura fondamentale. I tre livelli inferiori non sono progettati per essere utilizzati dal programmatore medio, ma concepiti principalmente per eseguire interpreti e traduttori necessari come supporto per i livelli più alti. Questi interpreti e traduttori sono scritti da professionisti chiamati **programmatori di sistema**, specializzati nella progettazione e nell'implementazione di nuove macchine virtuali. Il Livello 4 e quelli superiori sono invece pensati per i programmatori che devono risolvere problemi applicativi.

- Il livello 4 fornisce ai programmatori un modo per scrivere programmi per i livelli 1, 2 e 3 in una forma meno difficoltosa rispetto a quella dei linguaggi delle rispettive macchine virtuali.
- Il livello 5 consiste generalmente di linguaggi, spesso chiamati **linguaggi ad alto livello**, definiti per essere utilizzati dai programmatori di applicazioni. Ne esistono letteralmente a centinaia (C,C++,Java,…). I programmi scritti in questi linguaggi sono generalmente tradotti al livello 3 o ai livello 4 da un traduttore detto **compilatore**; in casi meno frequenti è anche possibile che essi siano **interpretati**. I programmi in Java, per esempio, di solito sono tradotti inizialmente in un linguaggio di tipo ISA chiamato *Java byte code,* che viene successivamente interpretato.

L'insieme dei tipi di dati, delle operazioni e delle funzionalità di ciascun livello è chiamato **architettura**. Questa tratta gli aspetti visibili agli utenti di quel determinato livello. Lo studio di come progettare le parti di un computer visibili ai programmatori è chiamato **architettura degli elaboratori**.

Cenni storici

Il computer è la versione più evoluta di una serie di strumenti di calcolo inventati sin dall'antichità: l'abaco, la macchina di Anticitera, i bastoncini di Nepero, ecc.

Si inizia a parlare di strumenti meccanici di calcolo nel XVII secolo con:

- **Pascalina** (1642) : macchina di *Pascal* che permette di addizionare e sottrarre;
- **Stepped Reckoner** (1672): macchina di *Leibniz* capace anche di moltiplicare e dividere.

Nella rivoluzione industriale troviamo invece i **telai Jacquard**; si trattano di normali telai a cui si è aggiunto un macchinario che permette la movimentazione automatica dei singoli fili. Per la loro natura programmabile, anch'essi sono considerati degli antenati dei calcolatori.

Il passaggio da macchina calcolatrice a vero e proprio computer si deve a *Charles Babbage* e la sua **Macchina analitica**; progettata nel 1833 ma mai realizzata, è il primo computer della storia. Si trattava di una colossale macchina a ingranaggi, alimentata a vapore e dotata di input, output, unità di memoria, di unità di calcolo decimale con registro di accumulo dei dati e di un sistema di collegamento tra le varie parti.

Nel corso dei secoli seguenti il computer passerà attraverso vari stadi: il computer analogico, i computer digitali meccanici ed elettromeccanici, ed infine quelli digitali ed elettronici.

I primi calcolatori sono virtuali, con la **macchina di Turing** (un modello astratto che definisce una macchina in grado di eseguire algoritmi, dotata di un nastro infinito su cui può leggere e/o scrivere dei simboli), e la **tesi di Curch-Turing** (se un problema è intuitivamente calcolabile, allora esisterà una macchina di Turing, o un dispositivo equivalente come il computer, in grado di risolverlo, cioè di calcolarlo).

Sarà *von Neumann* a progettare fisicamente l'**architettura di von Neumann**, una tipologia di architettura hardware per computer digitali programmabili a programma memorizzato la quale si contraddistingue per memorizzare i dati del programma e le istruzioni del programma nello stesso spazio di memoria.
L'idea di von Neuman era rappresentare e memorizzare il **programma** di calcolo così come si rappresentano e memorizzano i **dati** su cui opera (idea non del tutto nuova, ma ripresa dalla Macchina di Turing Universale)
Tale architettura sarà utilizzata per costruire **l'ENIAC**, il primo computer elettronico *general purpose* (dispositivi elettronici che non siano dedicati ad un solo possibile utilizzo) della storia e primo computer elettronico Turing completo della storia.

Nel XX secolo l'elaborazione automatica di dati diventa su larga scala, per motivi:

- Civili: censimento, grandi aziende, ecc.
- Militari: codici crittografici, calcoli balistici, ecc.

Dunque, questi motivi, e gli importanti progressi nel campo dell'elettronica - come il transistor e il circuito integrato - e dell'informatica hanno contribuito all'evoluzione del computer nella sua forma attuale passando da dispositivo elettronico presente solo in aziende e centri di ricerca a dispositivo ad uso comune e consumo di massa per gli utenti comuni.

<u>Generazioni tecnologiche</u>
I periodi definiti dalle principali innovazioni tecnologiche sono:

Generazione	Processore	Memoria centrale	Dispositivi I/O	Linguaggi di programmazione
Prima (1945-1955)	valvole termoioniche	linee di ritardo a mercurio, prime memorie a nuclei magnetici	schede e nastri (di carta) perforati, primi tamburi magnetici	Simbolici (assembler)
Seconda (1955-1965)	Transistor BJT	memoria a nuclei magnetici perfezionate	tamburi magnetici perfezionati e i primi dischi magnetici	primi linguaggi di programmazione al alto livello (FORTRAN, COBOL, Lisp) e compilatori
Terza (1965-1975)	circuiti integrati	circuiti integrati	dischi magnetici	Nascono i sistemi operativi, il time-sharing e la programmazione strutturata. Abbiamo i computer mainframe e la memoria virtuale, ed anche i primi minicalcolatori e le prime memorie cache. Nasce l'industria del software.

- Quarta generazione: 1975-oggi

Dal 1975 in poi le innovazioni tecnologiche sono state così numerose che si è smesso di raggrupparle.
Abbiamo il processore e memoria virtuale con circuiti integrati LSI, VLSI, ULSI,ecc., la

Legge di Moore: *le prestazioni dei processori, e il numero di transistor ad esso relativo, raddoppiano ogni 18 mesi*.

L'I/O ha lo standard di connessione a bus. Nasce inoltre:

- programmazione di sistema (Unix), dichiarativa, ad oggetti
- microcalcolatori, calcolatori personali, interfacce grafiche (GUI)
- reti di calcolatori, basi di dati
- Internet, World Wide Web, standard per l'industria del software
- architetture parallele, supercalcolatori, *Grid*
- calcolatori portatili e palmari, interfacce multimediali
- standard per suoni, immagini e filmati digitali
- sistemi *embedded*, informatica pervasiva

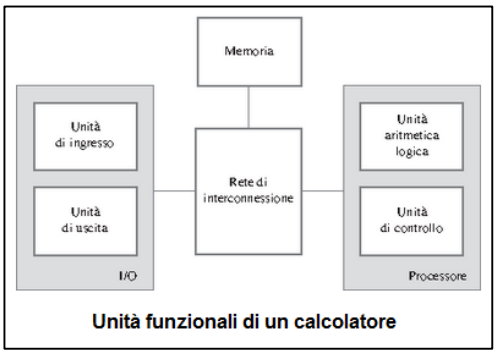
Unità funzionali delle macchine da calcolo

Tipi di calcolatori

Esistono quattro grandi categorie di calcolatori:

- **Embedded:** incorporati in un sistema più grande e programmati per uno scopo specifico e non riprogrammabili dall'utente (Es. sportelli Bancomat, Stampanti, Cellulari, CD, lettori DVD, Console,…)
- **Personali:** di largo uso, ha tre sottotipi principali
  - da tavolo (desktop)
  - portatili (laptop)
  - stazioni di lavoro (workstation)
- **Server** e **Sistemi enterprise:** condivisi da molti utenti in rete
- **Supercalcolatori, Grid:** prestazioni elevate, per simulazioni complesse (NASA, Computer militari, …)

Una tendenza recente è il cloud computing.



Unità funzionali di un calcolatore

Sottosistema di I/O

È una codifica binaria di tutta l'informazione che fluisce attraverso un calcolatore: numeri, caratteri, informazione multimediale

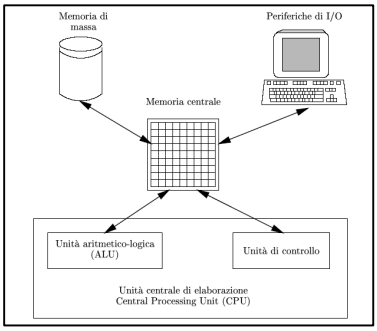
- Dispositivi di ingresso (**input**): tastiera, mouse, joystick, microfono, ecc.
- Dispositivi di uscita (**output**): stampante, schermo, riproduttore audio, ecc.
- Dispositivi di ingresso e uscita: touchscreen

Il complesso dei dispositivi di I/O di un calcolatore è anche detto periferica

Unità di memoria

In ambito informatico la memoria è la parte del computer destinata a conservare informazioni per un certo periodo di tempo. La memorizzazione di informazioni in memoria, e il successivo recupero delle medesime, sono funzioni fondamentali nel funzionamento del computer.

Una memoria può essere considerata astrattamente come una **sequenza finita di celle** in cui ogni cella contiene una **sequenza finita di bit**. Normalmente i bit sono gestiti a gruppi di otto, detti **byte**. Pertanto lo spazio fisico della memoria può essere immaginato come una sequenza di locazioni, ognuna contenente un byte. Ogni posizione è individuata da un preciso indirizzo, normalmente espresso tramite un numero intero positivo.



La macchina di Von Neumann (a destra) è divisa in una parte destinata al calcolo, detta processore, ed in una parte destinata alla memorizzazione.
Le operazioni effettuate sulla memoria di un computer sono fondamentalmente le seguenti:

- *Inizializzazione*: È il trattamento che subisce la memoria prima dell'uso normale;
- *Scrittura*: È l'operazione di memorizzazione delle informazioni.
  Es. assegnare il byte 123 alla cella di indirizzo 1000.
- *Lettura*: È l'operazione di recupero di informazioni memorizzate.

Si accede inoltre alla memoria e ai suoi contenuti ogni qual volta *l'Unità di Controllo* (Control Unit) del Processore richiede dati utili su cui eseguire un'operazione di elaborazione. Tutto ciò avviene grazie alle procedura di indirizzamento gestite dall'**Address Logic,** sottosistema del processore.
Per parlare indifferentemente di lettura o di scrittura, si usa il termine **accesso**. Per esempio, per indicare che una memoria è veloce sia in lettura che in scrittura, si dice che ha un basso tempo di accesso

La memoria informatica si può classificare secondo vari criteri:

- ordine di accesso (memorie ad accesso diretto o memorie ad accesso sequenziale);
- possibilità di scrittura (memorie a lettura-scrittura, memorie scrivibili una sola volta, memorie a sola lettura);
- velocità di lettura;
- velocità di scrittura;
- costo unitario;
- volatilità;
- tecnologia (elettronica, magnetica, ottica, magneto-ottica).

Memoria principale (o primaria)

Collegata alla scheda madre tramite connettori chiamati socket ed alla CPU tramite il BUS di sistema, la memoria primaria contiene dati ed istruzioni in attesa che questi siano prelevati dal microprocessore per essere elaborati. Qualora la memoria primaria venga esaurita molti sistemi di elaborazione moderni sono in grado di implementare il cosiddetto meccanismo della memoria virtuale (swap) come estensione provvisoria della memoria primaria.
Per realizzare una memoria principale vengono normalmente utilizzate tecnologie a semiconduttore (cioè vengono utilizzati circuiti integrati a base di silicio).

Tale memoria è costituita da numerosissime **celle di memoria**, dispositivi bistabili, capaci cioè di assumere due stati stabili alternativi attraverso i quali è possibile memorizzare la quantità minima di informazione (1 bit). Le operazioni che possono essere effettuate su questo componente sono due: lettura (load) e scrittura (store). Poiché nella realtà quotidiana le scelte possibili in vari casi sono molte più di due, un bit non è più sufficiente a rappresentarle e pertanto si è pensato di unire più celle di memoria in **registri di memoria**. Se con un solo bit è possibile ottenere solo due diversi stati, con due celle (2 bit) è possibile rappresentare 2^2 alternative, con l'associazione di 3 celle (3 bit) 2^3 alternative, e così via... In particolare, la dimensione del registro più diffusa è quella da 8 bit. L'unione di 8 bit forma infatti **1 byte** di memoria, il primo multiplo del bit, il quale può rappresentare fino a 256 possibili combinazioni diverse (2^8). In un registro di memoria le operazioni di lettura/scrittura avvengono contemporaneamente su tutte le celle facenti parte del registro. In memorie con registri da 1 byte, quindi, vengono lette 8 celle alla volta. Ogni registro di memoria è denotato da un **indirizzo.**

Bisogna distinguere tra vari tipi di memorie primarie, a seconda della funzione svolta e delle loro caratteristiche peculiari. Di seguito vengono elencate quelle più importanti.

- **RAM**, l'acronimo per "random access memory", ovvero "memoria ad accesso casuale", è la memoria in cui vengono caricati i dati che devono essere utilizzati dal calcolatore per elaborare. La RAM può essere volatile (si cancella spontaneamente ed ha bisogno di essere aggiornata), statica o tamponata (mantiene l'alimentazione anche a macchina spenta).
- **ROM**, l'acronimo per "read only memory", ovvero "memoria in sola lettura (o solamente leggibile)", è una memoria permanente (cioè ha un contenuto fisso che non può essere cancellato ed inoltre non è volatile), presente sulla scheda madre, che contiene le istruzioni che la CPU deve caricare per consentire l'avvio del sistema e le routine di base che prendono il nome di BIOS (Basic I/O System).

tecnologia corrente: *transistori a semiconduttori*

- *memoria veloce, costosa, volatile*

unità elementari da 1 bit, parole di lunghezza fissa

- *oggi quasi sempre una potenza di 2, e.g. 8, 16, 32, 64*

lettura e *scrittura* di una parola a un *indirizzo* specificato

- *modello sequenziale, tuttavia RAM: tempo di accesso costante, indipendente dall'indirizzo*

Memoria cache

È una memoria temporanea, non visibile al software, che memorizza un insieme di dati che possano successivamente essere velocemente recuperati su richiesta.

Quando è necessario l'accesso ad un dato, questo dato viene prima cercato nella cache. Se è presente e valido, viene utilizzata la copia presente. Viceversa, viene recuperato dalla memoria principale, e memorizzato nella cache, nel caso possa servire successivamente.

La **CPU cache** viene utilizzata per accelerare l'accesso alle posizioni di memoria RAM usate più frequentemente. Si tratta di una piccola quantità di memoria veloce installata direttamente sul processore o nelle sue immediate vicinanze.

- ad accesso più veloce, memoria di *prossimità*
- *gerarchia di memoria* centrale: più livelli di cache

Memoria secondaria (o memoria di massa)

I maggiori rappresentanti sono gli hard disk, ma anche supporti rimovibili come dischi floppy, CD, DVD, nastri magnetici, memorie flash di ogni tipo ed altro ancora.

La caratteristica principale della memoria di massa è la "non volatilità", ovvero la possibilità di memorizzare permanentemente i dati (per questo si parla anche di memoria di archiviazione).

La ragione della maggior velocità delle memorie primarie rispetto alle memorie secondarie risiede nel fatto che i tempi medi di accesso a memoria principale sono dell'ordine delle centinaia di nanosecondi, contro i millisecondi delle memorie di archiviazione, che, quindi, necessitano di tempi di accesso maggiori di ben 5 ordini di grandezza.

I dati sono riuniti in entità omogenee dette **file**. Le memorie di massa sono gestite da un componente fondamentale dei sistemi operativi, il **file system**: ogni sistema operativo ne utilizza uno diverso ed i più famosi sono FAT32 ed NTFS di casa Microsoft ed ext2-ext3 dei sistemi Linux. Fondamentalmente l'organizzazione delle memorie di massa è gestita tramite strutture dati collegate che possono essere liste o, molto più frequentemente, B-Alberi, oppure tabelle di indirizzamento.

- *persistente*, basso costo, grande capacità, lenta

Processore (o CPU, unità di elaborazione centrale)

È una tipologia di processore digitale general purpose (dispositivo elettronico non dedicato ad un solo possibile utilizzo) la quale si contraddistingue per sovrintendere tutte le funzionalità del computer digitale basato sull'architettura di von Neumann. In particolare l'unità di elaborazione centrale è una tipologia di processore estremamente diffusa in quanto i moderni computer general purpose normalmente sono digitali e basati sull'architettura di von Neumann.

Il compito della CPU è quello di eseguire le istruzioni di un programma presente in memoria. Durante l'esecuzione del programma la CPU legge o scrive dati in memoria. Il risultato dell'esecuzione dipende dal dato su cui si opera e dallo stato interno in cui la CPU stessa si trova, e può mantenere la traccia delle operazioni passate.
Attualmente la CPU è tipicamente implementata come **microprocessore**.

Contenuto

Una generica CPU contiene:

- un'**unità di controllo** (CU) che legge dalla memoria le istruzioni, se occorre legge anche i dati per l'istruzione letta, esegue l'istruzione e memorizza il risultato se c'è, scrivendolo in memoria o in un registro della CPU. Controlla e organizza l'attività dei dispositivi collegati all'elaboratore: recupera tutte le istruzioni dalla memoria, le decifra e le esegue.
- un'**unità aritmetica e logica** (ALU) che si occupa di eseguire le operazioni logiche e aritmetiche. È l'unità aritmetico logica in cui vengono effettuati i calcoli aritmetici e logici richiesti dalle istruzioni del programma.
- dei **registri**, speciali locazioni di memoria interne alla CPU, molto veloci, a cui è possibile accedere molto più rapidamente che alla memoria: il valore complessivo di tutti i registri della CPU costituisce lo stato in cui essa si trova attualmente. Due registri sempre presenti sono:

Una CPU è un circuito digitale sincrono: vale a dire che il suo stato cambia ogni volta che riceve un impulso da un segnale di sincronismo detto **clock**, che ne determina di conseguenza la velocità operativa, detta **velocità di clock**: quindi il tempo di esecuzione di una istruzione si misura in cicli di clock, cioè in quanti impulsi di clock sono necessari perché la CPU la completi.

Funzionamento (semplificato) del processore

- Unità di ingresso: accettazione delle informazioni in forma di programma e dati attraverso un'unità di input → memorizzazione nella memoria
- L'informazione memorizzata è recuperata con un programma di recupero e processata dalla ALU
- Le informazioni processate lasciano il computer grazie alle unità di output
- Tutte questa attività sono dirette dalla CU.

Concetti imperativi di base

Le istruzioni di una CPU (**instruction set**) sono semplicemente dei numeri, detti **opcode** o codici operativi: in base al loro valore l'unità di controllo intraprende delle azioni predefinite, come per esempio leggere la successiva locazione di memoria per caricare un dato, oppure attivare la ALU per eseguire un calcolo, oppure scrivere il contenuto di un registro in una certa locazione di memoria o in un altro registro, oppure una combinazione di queste.

Per una persona, stendere programmi scrivendo direttamente gli opcode è estremamente noioso e prono all'errore. Per questo motivo si utilizza l'**assembly**, che associa un simbolo mnemonico ad ogni istruzione della CPU e introduce una sintassi che permette di esprimere i vari metodi di indirizzamento in modo più intuitivo.

Istruzioni macchina

Abbiamo detto che le attività del computer sono governate da istruzioni. Per eseguire un'operazione specifica, un programma deve consistere di una lista di istruzioni memorizzate in memoria. Ogni istruzione individuale è recuperata dalle memoria al processore che la processa.

Esistono diversi **tipi** di istruzioni, che coinvolgono parti diverse del calcolatore (ma sempre l'unità di controllo), per esempio (in forma simbolica):

- **trasferimento dalla memoria al processore**:

```
Load R2, LOC
```

L'istruzione legge il contenuto di una locazione di memoria il quale indirizzo è rappresentato simbolicamente dall'etichetta LOC, e carica questo contenuto nel registro R2. Il contenuto originale nella locazione LOC viene preservato, mentre quello nel registro 2 viene sovrascritto.
L'esecuzione di questa istruzione richiede vari passaggi: per primo, l'istruzione è recuperata dalla memoria al processore, poi l'operazione è determinata dalla CU il quale recupera l'operazione e il riferimento alla locazione nel processore; infine il nuovo contenuto e memorizzato nel registro R2.

- **operazione aritmetica**:

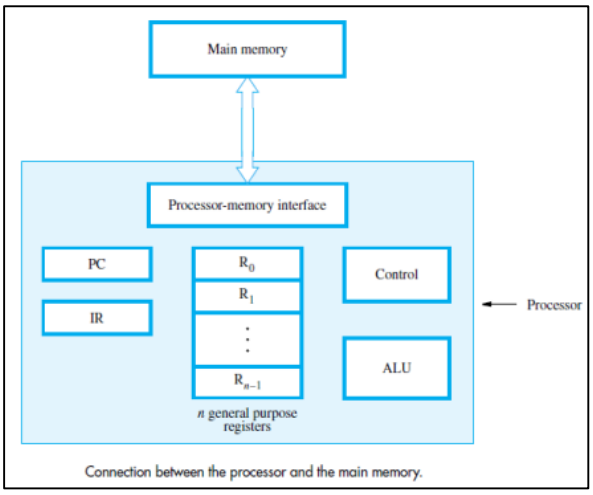
```
Add R4, R3, R2
```

Aggiunge il contenuto della somma dei registri R3 e R4 nel registro R4.

- **trasferimento dal processore alla memoria:**

Store R4, LOC

L'istruzione copia il contenuto del registro R4 nella locazione di memoria LOC. Il contenuto originale di LOC viene sovrascritto, mentre quello di R4 preservato.



Connection between the processor and the main memory.

Registri del processore

La figura a destra mostra come il processore e la memoria sono connessi. Oltre alla ALU e alla CU, il processore contiene dei registri usanti per scopi differenti:

- **Registro di istruzione (IR)**: contiene l'istruzione **corrente** che si sta eseguendo; i suoi output sono disponibili alla CU, la quale genera un segnale di temporizzazione che controlla i vari elementi del processo in esecuzione con l'istruzione;
- **Contatore di programma (PC):** contiene l'indirizzo di memoria dell'istruzione **successiva** che sarà eseguita; si dice che il PC punta alla prossima istruzione che è recuperata dalla memoria;
- **Registri di uso generale (R_0, R_1, ... , R_{n-1})**: chiamato anche registro di processo, servono per una varietà di funzioni che includono per dati, puntatori ecc. Inoltre sono sempre (esplicitamente) accessibili al programmatore.

Esecuzione di istruzioni macchina

L'**interfaccia memoria-processore** è un circuito che gestisce il trasferimento di dati tra la memoria principale ed il processore. Se una parola deve essere letta dalla memoria, l'interfaccia manda l'indirizzo di questa parola alla memoria con un segnale di controllo Read. L'interfaccia aspetta che la parola sia recuperata, e poi la trasferisce in un appropriato registro. Similmente, se la parola deve essere scritta in memoria, l'interfaccia manda sia l'indirizzo che la parola nella memoria con un segnale di controllo Write.

Alcuni tipici passi operativi sono:

- Caricamento in memoria del programma (spesso trasferito dalla memoria secondaria);
- L'esecuzione inizia quando il PC è settato per essere la prima istruzione del programma. I contenuti del c sono trasferiti nella memoria con il segnale Read e caricato nel registro specifico (in questo caso IR).
- Le istruzioni come Load, Store e Add eseguono il programma. Il risultato è mandato al registro di processo.

Ad un certo punto dell'istruzione i contenuti della PC sono incrementati così che il PC punta alla prossima istruzione da eseguire; così il processore è pronto a recuperare una nuova istruzione.

Passi operativi

- Caricamento in memoria del programma
- PC ← indirizzo dell'istruzione iniziale
- Cicli prelievo-esecuzione determinati dal programma

Passi operativi elementari per ciascuna istruzione:

- **prelievo**: lettura IR ← (PC)+
- **decodifica**: quale operazione è da eseguire
- **esecuzione**: per esempio, nel caso di una Load
  - indirizzo: → interfaccia processore-memoria
  - segnale di lettura → interfaccia processore-memoria
  - dato letto: interfaccia processore-memoria → registro

Interruzioni

Oltre al trasferire dati tra la memoria e il processore, i computer accettano dati da dispositivi di input e mandano dati a dispositivi di output. Può capitare che un in un dispositivo I/O avvenga un imprevisto o un guasto; in ogni caso la normale esecuzione di un programma può essere prevaricata se un dispositivo richiede un urgente servizio, sospendendo la sua esecuzione tramite un programma chiamato *interrupt-service routine*. Quando quest'ultimo programma finisce, lo stato per processo interrotto viene recuperato dalla memoria continuando la sua esecuzione.

Processo:

1. *segnalazione* della richiesta;
2. *sospensione* (se è il caso) dell'esecuzione del programma corrente, e *salvataggio* dell'informazione necessaria alla sua successiva ripresa;
3. esecuzione della relativa *routine di servizio dell'interruzione;*
4. al termine di questa, *ripristino* dell'informazione salvata e *ripresa* dell'esecuzione interrotta.

Prestazioni del calcolatore

La più importante misura per calcolare la perfomance di un computer è quanto velocemente può eseguire programmi. La velocità con cui un computer esegue programmi è influenzata da:

- Le sue instruction set;
- Il suo hardware;
- Il suo software, incluso il sistema operativo;
- La tecnologia con cui l'hardware è implementato.

Poiché i programmi usualmente sono scritti in linguaggi di alto livello, la performance è anche influenzata dalla velocità dei compilatori che traducono i programmi in linguaggio macchina.

Tecnologia dell'hardware

La tecnologia della **Very large scale integration (VLSI)** (usata per fabbricare circuiti elettronici per processori con elevata integrazione di transistor all'interno di un singolo chip) è un fattore critico nella velocità di esecuzione delle istruzioni.
La velocità di switchare tra lo stato 0 e 1 dei circuiti logici è largamente determinata dalla dimensione dei transistore che implementano i circuiti (più sono piccoli i circuiti, più sono veloci).
La riduzione dei transistor ha 2 vantaggi:

- Le istruzioni possono essere eseguite più velocemente;
- Più transistor possono essere piazzati in un singolo chip.

Parallelismo

La performance può essere incrementata eseguendo un numero di operazioni in parallelo. Il parallelismo può essere implementato in molti modi differenti:

- **Parallelismo a livello di istruzione**: il modo più facile di eseguire una sequenza di istruzioni in un processore è completare tutti i passi della corrente istruzione prima di iniziare i passi dell'istruzione successiva. Se si sovrappongono le esecuzioni con i passi della successiva istruzione, il tempo di esecuzione totale viene ridotto.
- **Processori multicore**: delle multiple unità di processo possono essere fabbricate in un solo chip. Il termine *core* è utilizzato per questi processori multipli, dalla quare nasce la terminologia *dual-core, quad-core,* e *octo-core*, aventi rispettivamente 2, 4 e 8 core.
- **Multiprocessori**: sistemi di computer contenenti più processori, con la possibilità di contenere più core. Questi sistemi possono eseguire un numero di differenti programmi in parallelo (multitask).
    - Tutti i processori usualmente hanno accesso a tutta la memoria del sistema, e il termine *shared-memory multiprocessor* è spesso usata per indicare questo.
    - I computer normalmente hanno accesso solamente alle loro unità di memoria, ma quando i programmi da eseguire necessitano di condividere dati, loro scambiano messaggi grazie ad una comunicazione di rete. Questa proprietà è distinta dalla memoria condivisa, prendendo il nome di *message-passing multicomputers*.

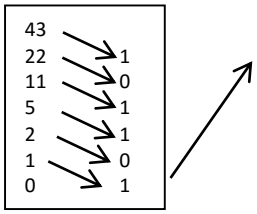
Rappresentazione dei numeri e aritmetica

Il modo più naturale per rappresentare un numero in un computer è utilizzando una stringa di bit, chiamato numero binario.

Rappresentazione binaria dei numeri interi

La rappresentazione **posizionale in base 2** dei numeri **naturali** avviene nel seguente metodo:
Si divide ripetutamente il numero per 2, se il risultato ha resto si scrive 1 e si tronca, se non lo ha si mette 0, fino a quando il numero diventa 0. I risultati dei resti vanno poi letti al contrario

Es. 43



101101 Dunque $43_{(10)} = 101101_{(2)}$

Conversione da binario a decimale

Si prende la prima cifra del numero decimale e si moltiplica per 2 elevato alla grandezza massima dell'indice numero decimale e si somma alla seconda a cui applichiamo questo processo, e così via.

Es. 101101 $1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 32 + 8 + 4 + 1 = 45$

Li stessi metodi possono essere applicati anche a basi diverse da 2

Addizione di interi

$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ 1 \\ \hline 1\,0 \end{array}$$

La somma di 1+1 è il vettore a 2bit 10; questo avviene perché in binario noi non usiamo il 2, quindi la somma di 1+1 diventa 0, ma dobbiamo riportare un bit a sinistra.
Diremo che la *somma* è 0, e il *carry-out* è 1. Il carry-out di un di bit diventa il *carry-in* del prossimo bit a sinistra.

$$carry - in \leftarrow carry - out$$

Scalamento (Shift)

Lo shift a sinistra di un bit (inserendo uno 0) equivale a moltiplicare per due.

Es. $0011 = 3_{10}$ $0011 \leftarrow 0$ $0110 = 6_{10}$

Lo shift a destra di un bit (inserendo uno 0) equivale a dividere per 2 (con troncamento).

Es. $0100 = 4_{10}$ $0 \rightarrow 0100$ $0010 = 2_{10}$

Rappresentazione dei numeri positivi e negativi

Abbiamo bisogno di rappresentare anche i numeri positivi e negativi. Per tale rappresentazione solo utilizzati 4 sistemi:

- **Binario puro**: basta inserire il segno prima della stringa binaria calcolata come l'esempio precedente.
  Es. $6 \rightarrow 110$ $-6 \rightarrow -110$
- **Modulo e segno**: la codifica è identica al binario, con la differenza che il primo bit indica il segno (0 negativo, 1 positivo)
  Es. $6 \rightarrow 0110$ (modulo e segno) $-6 \rightarrow 1110$ (modulo e segno)
- **Complemento a 1**: la codifica è identica al modulo e segno, con la differenza che i numeri negativi vengono complementati a 1.
  Es. $6 \rightarrow 0110$ (modulo e segno) $-6 \rightarrow 1110$ (modulo e segno) $\rightarrow 1001$ (complemento a 1)
- **Complemento a 2**: la codifica è identica al complemento a 1, con la differenza che ai numeri negativi del complemento a 1 viene sommata una unità
  Es. $6 \rightarrow 0110$ (modulo e segno) $-6 \rightarrow 1001$ (complemento a 1) $\rightarrow 1010$ (complemento a 2)

La rappresentazione della stringa è comunque data dal numero di bit che ha noi interessa rappresentare. Se ad esempio vogliamo rappresentare i bit dei numeri da 1 a 63 bisogna utilizzare 7 bit ($2^6 = 64$, meno 1 del segno).

Es. Rappresentare il binario puro, modulo e segno e complemento 1 e 2, dei seguenti numeri su 7 bit.

Decimale	Binario	Modulo e Segno	Complemento a 1	Complemento a 2
15	1111	0001111	0001111	00001111
-15	-111	100111	1110000	1110001
-91	-10011	1010011	1101100	1101101

Somma algebrica in complemento a due

Come quella binaria naturale, ma trascurando il riporto in uscita

Normale		Complemento a 2	
0010 + 0011 = 0101	(+2) + (+3) = (+5)	0100 + 1010 = 1110	(+4) + (−6) = (−2)
1011 + 1110 = 1001	(−5) + (−2) = (−7)	0111 + 1101 = 0100	(+7) + (−3) = (+4)

La **sottrazione** è invece l'addizione con il complemento a due del sottraendo

Sottrazione		Complemento a 2	
1101 − 1001	(−3) − (−7)	1101 + 0111 = 0100	(+4)
0010 − 0100	(+2) − (+4)	0010 + 1100 = 1110	(−2)

Estensione e riduzione del segno

Spesso usiamo rappresentare un valore in un certo numero base di bit. Anche queste operazioni risultano semplici nella rappresentazione in complemento a due:

- **estensione del segno**: allungare una stringa binaria in modo che rappresenti lo stesso numero
  - si duplica a sinistra il bit di segno tante volte quanto occorre
- **riduzione del segno**: accorciare una stringa binaria in modo che rappresenti lo stesso numero
  - si rimuove il bit più a sinistra tante volte quanto occorre, purché il bit successivo abbia ugual valore

Trabocco (overflow)

Con i complementi a 2, i valori di n bit possono essere rappresentati nel raggio di $[-2^{n-1}, 2^{n-1}-1]$.

Es. 4 bit possono rappresentare da -8 a +7

Se il risultato di un'operazione aritmetica travalica questi limiti, non è valido: si ha un evento di **trabocco (*overflow*).**

Quando sommiamo un numero senza segno, un carry-out di 1 dalla più significate posizione del bit indica che si è scatenati un overflow.

addizione, due semplici regole:

- addendi *discordi*: non può aversi trabocco
- addendi *concordi*: si può avere il trabocco quando il valore della stringa di bit risultata dalla somma è già presente nel raggio di numeri.

Rappresentazione dei numeri razionali

virgola binaria: collocata fra due posizioni della stringa binaria (o anche oltre)

virgola fissa: la sua collocazione determina estensione e precisione della rappresentazione, per esempio con *n* bit:

- subito a destra della stringa: rappresentazione di 2^n interi
- subito a destra del bit di segno: rappresentazione di razionali in $[-1, 1-2^{-(n-1)}]$, con precisione $2^{-(n-1)}$

virgola mobile, per conciliare estensione e precisione: $n = m\ b^e$

- ***base*** *b* prefissata, 2 nella rappresentazione binaria
- ***mantissa*** *m* ed ***esponente*** *e*: interi con segno, in due parti di lunghezza prefissata della stringa binaria

Rappresentazione dei caratteri

3210	000	001	010	011	100	101	110	111
0000	NUL	DLE	SPACE	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	/	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Il più comune schema di codifica per la rappresentazione di caratteri è l'**ASCI** (American Standard Code for Information Interchange).
I caratteri alfanumerici, gli operatori, i simboli e i caratteri di controllo sono rappresentati da sequenze binarie a 7 bit, mostrati nella tabella a sinistra.
È comunque conveniente usare 8bit (un byte) per rappresentare e immagazzinare il carattere; il codice del carattere occuperà ugualmente 7 bit.

Per rappresentare la tabella dei codici ASCII è necessario fare riferimento alla tabella a sinistra di corrispondenza di bit.
Nelle colonne troviamo la prima parte del codice binario, mentre nelle righe la seconda parte.

Es. Sulla base di questa tabella è possibile intuire come funziona un documento:

Marco = M=01001101 (1° byte), A=01000001 (2° byte), …

Un carattere = 1 byte

Estensioni del codice ASCII per l'**internazionalizzazione**:

- famiglia di codici **ISO 8859-*x* :** a 8 bit, estensioni nazionali
- **UCS** (ISO/IEC 10646: *Universal Character Set*): base per
- **Unicode, UTF-8** : a lunghezza variabile (multipla di 8)

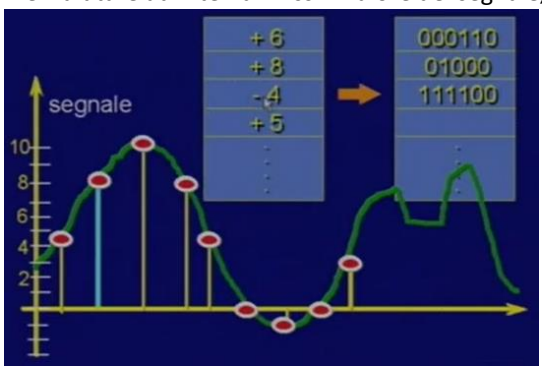
Rappresentazione delle immagini

Un monitor costituito da una sequenza di pallini (pixel) che possono accendersi e spegnersi come i bit 0 e 1 per rappresentare l'immagine. Ad esempio uno schermo 800x600 è composto da 800 righe di pixel e 600 colonne di pixel. Il singolo elemento (punto del disegno) è detto **PIXEL** (picture element), quanti più sono i pixel, tanto maggiore è la precisione e la qualità dell'immagine

Rappresentazione dei suoni

Il trasferimento di un suono in segnale elettrico avviene secondo il seguente schema:
onde acustiche ⇒ (trasduttore, es. microfono) ⇒ onde elettriche ⇒ (trasduttore, es. altoparlante) ⇒ onde acustiche.

Il meccanismo che ci permette di trasformare un onda elettrica in una sequenza di bit vene detta **campionamento**. Esso consiste nel valutare ad intervalli fissi il valore del segnale, e di codificarlo come sequenza numerica.



Es. Supponiamo di avere un segnale elettrico (andamento della tensione o corrente). Il campionamento prende i valori +5, +8, -4, +8, +6,… i quali numeri interi relativi vengono convertiti in binario.

Data questa sequenza di bit ricaviamo dei valori che messi in sequenza costituiscono il segnale acustico che possiamo ricostruire.

Il dispositivo che permette di realizzare questa trasformazione è **l'ADC** (convertitore analogico digitale) che riceve i dati dal trasduttore e li converte in bit.

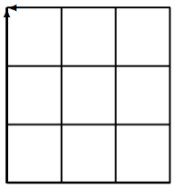
Registrazione: Onda sonora ⇒ trasduttore ⇒ ADC ⇒ $(campionamento)$ 010001

Trasmissione: 010001 ⇒ DAC(convertitore digitale analogico) ⇒ trasduttore

Questo avviene anche quando noi parliamo al microfono.

Artimetica Maya

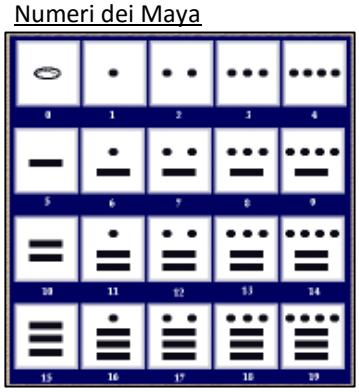
La matematica Maya permette di realizzare tutte le operazioni aritmetiche senza fare ricorso all'astrazione o alla memoria: non comporta la necessità di studiare, né la comprensione di regole, né la memorizzazione della tavola pitagorica, eliminando così una tradizionale barriera psicologica all'apprendimento divertente e ricreativo della materia.
Si tratta piuttosto dell' insieme di alcune semplici operazioni manuali che danno luogo a una grande varietà di strategie molto libere che conducono al risultato di un'operazione data. Assomiglia più a un gioco che ad una materia scolastica.
Le sue proprietà visuali e tattili sembrano poter trovare l'impiego più felicemente utile nell'apprendimento della matematica in condizione di inabilità sensoriale. La sua concretezza permette di visualizzare ogni passo di un'operazione aritmetica, facilitando il successivo passaggio all'astrazione, anche nel caso di ritardo mentale.



L'abaco Maya: algoritmi manipolativi di calcolo

- rappresentazione vigesimale dei numeri (posizionale in base 20)
- forma additiva e ostensiva delle cifre

Numeri dei Maya



La costruzione dei numeri presso gli antichi Maya avveniva con un procedimento additivo, usando soltanto due oggetti fondamentali (oltre il simbolo indicante lo zero):

- cerchio, avente valore zero
- il punto, avente il valore di unità
- il trattino, che aveva il valore numerico di 5.

Essi vengono chiamati **caracol**, **frijol** (fagiolo) e **palito** (legnetto, stecchino), in quanto i fagioli e i bastoncini di legno erano effettivamente usati per comporre i numeri (probabilmente su un una scacchiera, tablero, corrispondente al nostro abaco).
Si trattava di una numerazione di tipo posizionale in base 20; ciò era possibile in quanto, fin dai primi tempi di cui si abbia testimonianza, i Maya avevano elaborato il concetto dello zero.

Operazioni

- semplice conversione decimale
- moltiplicazione "senza tabellina"
  - nonché divisione, radice quadrata, ...
- due semplici **regole di equivalenza** di gruppi di oggetti sull'abaco:
  - 5 unità ↔ 1 cinquina (nella stessa posizione)
  - 4 cinquine ↔ 1 unità in posizione adiacente più significativa
- algoritmi **manipolativi**:
  - **addizione**: mettere assieme gli oggetti di uguale significatività e applicare le regole di equivalenza per la riduzione in cifre
    (+) funziona egualmente per la somma di più di due addendi!
    (+) non prescrive ordine di esecuzione per significatività
    → parallelismo
  - **sottrazione**: estrarre una copia del sottraendo dal minuendo, usando se serve le regole di equivalenza per formare la copia
- regole e algoritmi del tutto simili per la rappresentazione decimale: basta rimpiazzare '4' con '2' nella seconda regola di equivalenza
- Conversione fra base 20 e base 10:
  - si può effettuare secondo un algoritmo generale di conversione di base
  - oppure si può usare l'algoritmo R, escogitato da Bruna Radelli:
    - di tipo manipolativo sull'abaco, con le note regole di equivalenza
    - spostamento di oggetti lungo le diagonali secondarie
    - raddoppio/dimezzamento a ogni passo lungo tali diagonali
    - applicazione delle regole della base di partenza o di quella di arrivo determinata dalla corrispondente direzione di adiacenza fra caselle
- si può moltiplicare senza la tavola pitagorica, con gli operandi lungo due lati ben scelti dell'abaco, come nel modo "alla musulmana": *e.g.* 1025 x 521
  - significatività delle caselle triangolari costante lungo diagonali secondarie
  - semplici regole per costruire il prodotto di due cifre, sommando i prodotti delle coppie di oggetti
  - esecuzione parallela dei prodotti delle coppie di cifre ...
  - quindi delle somme parziali (lungo le diagonali secondarie) ...
  - e delle riduzioni in cifre del risultato finale

Strutture algebriche e reticoli

Logica matematica

La logica è una scienza autonoma che studia i metodi e i principi più usati nel distinguere i ragionamenti corretti da quelli scorretti. Per **Aristotele** la logica è la scienza che riguarda gli enunciati che possono essere veri o falsi. In una parola la logica si occupa della relazione di "conseguenza logica", cioè analizza come nel ragionamento si può preservare la verità (se un ragionamento ha premesse vere ed è corretto porterà a conclusioni vere).
Da Aristotele e gli stoici a **Leibniz** si sono usati in logica simboli per proposizioni e termini.

I precursori della formalizzazione logica furono appunto Aristotele e Leibniz con le seguenti teorie:

- **Teoria del sillogismo**: un tipo di ragionamento dimostrativo il quale partendo dai tre tipi di termine "maggiore" (che funge da predicato nella conclusione), "medio" e "minore" (che nella conclusione funge da soggetto) classificati in base al rapporto contenente - contenuto, giunge ad una conclusione collegando i suddetti termini attraverso brevi enunciati (premesse).
  Es.
  - (premessa maggiore) Tutti gli uomini sono mortali
  - (premessa minore) Tutti i greci sono uomini
  - (conclusione) Dunque tutti i greci sono mortali
- **Mathetis Universalis**: saggio di Leibiniz dove da la deduzione di tutte le leggi del sillogismo a partire da alcuni principi costruiti sulla base di alcune definizioni

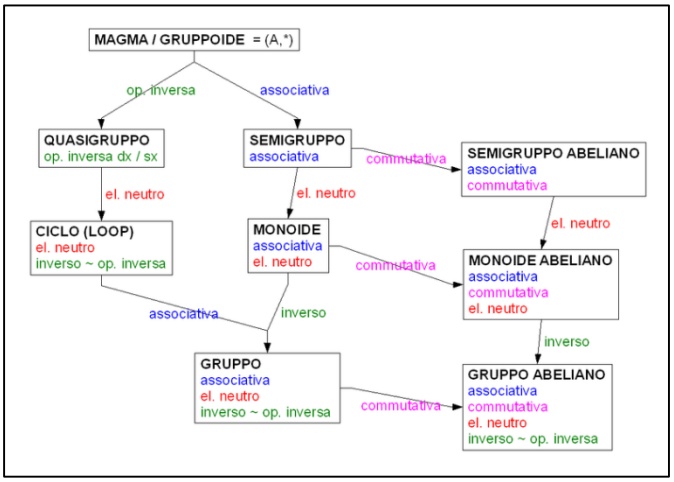
Leibniz anticipa di un paio di secoli l'intuizione di George Boole (0 e 1), il quale le raggrupperà insieme ai temi dell'analisi matematica della logica nelle **leggi di un'algebra del pensiero**.

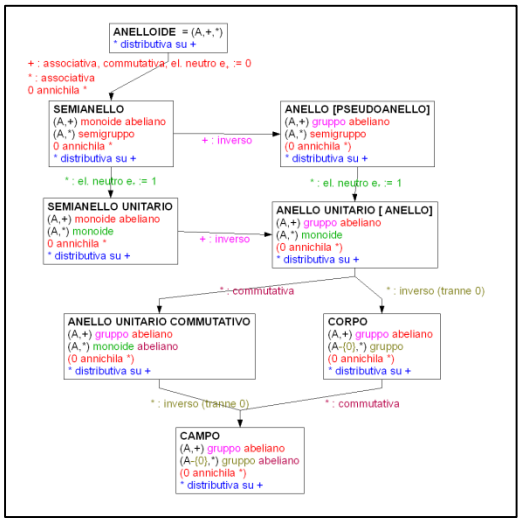
La logica matematica è il settore della matematica che studia i sistemi formali dal punto di vista del modo di codificare i concetti intuitivi della dimostrazione e di computazione come parte dei fondamenti della matematica.

Algebra e strutture algebriche

L'**algebra** è un insieme di più operazioni su di una struttura matematica.
Es. L'aritmetica dei naturali, degli interi, dei reali, ...

Le **equazioni** giocano un ruolo fondamentale nell'algebra tradizionale; qui le incognite sono le **variabili.**
Sono non meno fondamentali nell'**algebra astratta** (XX secolo, campo della matematica che si occupa dello studio delle strutture algebriche come gruppi, anelli e campi), dove però le incognite sono i **simboli di operazione.**





Una struttura algebrica è un insieme S, chiamato insieme **sostegno** (della struttura), munito di una o più operazioni che possono essere nullarie, unarie, binarie, ... e che sono caratterizzate dall'avere proprietà quali commutatività e associatività.

In linea generale un insieme sostegno può essere munito di diversi sistemi di operazioni e per individuare una struttura algebrica senza incorrere in possibili ambiguità, vanno specificate tutte le sue operazioni. Per esempio per specificare la struttura ordinaria di un gruppo additivo sull'insieme Z dei numeri interi, si può ricorrere alla notazione $(Z, +, 0, -)$, dove + è la somma, 0 lo zero come operazione nullaria, e – indica l'operazione unaria che a un intero associa il suo opposto.

Specie di strutture algebriche

- **semigruppo**: (A; ·), con · un'operazione binaria associativa; il semigruppo è detto commutativo se tale è l'operazione binaria

- **monoide**: (A; e, ·), un semigruppo (A; ·) con e costante neutra rispetto a ·; il monoide è detto commutativo se lo è il suo ridotto semigruppo
- **gruppo** [commutativo]: (A; e, ·, —), un monoide [commutativo] (A; e, ·) con un'operazione unaria — che dà l'inverso rispetto a · ed e: $x \cdot —x = —x \cdot x = e$
- **semianello** [commutativo]: (A; 0, 1, +, ·), un monoide commutativo (A; 0, +), detto additivo, e un monoide [commutativo] (A; 1, ·), detto moltiplicativo, tali da soddisfare:
  - distributività: $x \cdot (y+z) = (x \cdot y)+(x \cdot z)$, $(x+y) \cdot z = (x \cdot z)+(y \cdot z)$
  - cancellazione: $0 \cdot x = x \cdot 0 = 0$
- **anello** [commutativo]: (A; 0, 1, +, ·,—), un semianello [commutativo] (A; 0, 1, +, ·), con il monoide additivo esteso a un gruppo commutativo (A; 0, +, —)

Deduzione equazionale

Classi di algebre in senso stretto quali quelle viste in precedenza sono caratterizzate da assiomi di forma molto semplice:

- **equazioni** di termini, costituiti da simboli di operazione e variabili: $\mathbf{t_1 = t_2}$ (con implicita quantificazione **universale**)

Quando una classe di algebre è caratterizzata da un insieme di equazioni, che ne costituisce la **base assiomatica**, è possibile dedurre da tale base tutte le equazioni valide in ogni algebra della classe mediante le regole del **calcolo equazionale** di Garrett Birkhoff:

- **riflessività:** $\mathbf{t = t}$
- **simmetria:** *se* $\mathbf{t_1 = t_2}$ *allora* $\mathbf{t_2 = t_1}$
- **transitività:** *se* $\mathbf{t_1 = t_2, t_2 = t_3}$ *allora* $\mathbf{t_1 = t_3}$
- **sostituzione:** *se* $\mathbf{t_1 = t_2}$ *allora* $\mathbf{\tau(t_1) = \tau(t_2)}$
- **rimpiazzamento:** *se* $\mathbf{t_1 = t_2}$ *allora* $\mathbf{t[u \leftarrow t_1] = t[u \leftarrow t_2]}$
  - *il termine* ***t[u←t']*** *è ottenuto dal termine* ***t****rimpiazzando il suo sottotermine al posto* ***u****con il termine* ***t'***

Reticolo

Un reticolo è un insieme parzialmente ordinato in cui ogni coppia di elementi ha sia un **estremo inferiore (inf)** che un **estremo superiore (sup)**. I reticoli possono anche essere caratterizzati come strutture algebriche che soddisfano determinate identità. Un reticolo è un'algebra $(A; \vee, \wedge)$ tale che le sue due ridotte $(A; \vee)$, $(A; \wedge)$ sono semireticoli, e inoltre valgono gli assiomi di **assorbimento**:

$$x \vee (x \wedge y) = x$$
$$x \wedge (x \vee y) = x$$

Un reticolo è completo se ogni insieme di suoi elementi ha minimo maggiorante e massimo minorante.

Strutture algebriche e Algebra di Boole

L'algebra di Boole è un ramo dell'algebra astratta che comprende tutte le algebre che operano con i soli valori di verità 0 o 1, detti variabili booleane o logiche. La struttura algebrica studiata dall'algebra booleana è finalizzata all'elaborazione di espressioni nell'ambito del calcolo proposizionale.
Essendo un reticolo dotato di particolari proprietà, l'algebra booleana risulta **criptomorfa**, cioè associata biunivocamente e in modo da risultare logicamente equivalente, ad un insieme parzialmente ordinato reticolato. Ogni algebra booleana risulta criptomorfa ad un particolare tipo di anello, chiamato **anello booleano**.

Tale algebra permette di definire gli **operatori logici** AND, OR e NOT, la cui combinazione permette di sviluppare qualsiasi funzione logica e consente di trattare in termini esclusivamente algebrici le operazioni insiemistiche dell'intersezione, dell'unione e della complementazione, oltre a questioni riguardanti singoli bit 0 e 1, sequenze binarie, matrici binarie e diverse altre funzioni binarie.

Livello logico digitale

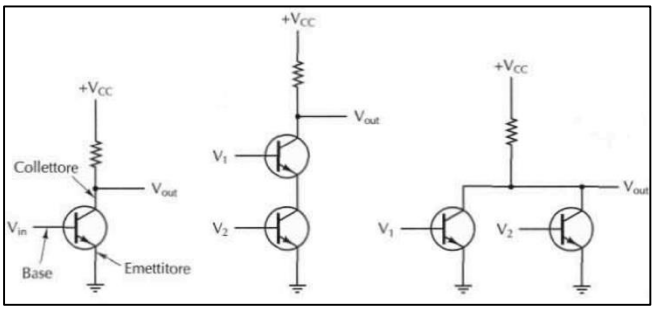
Si trova alla base della gerarchia della macchina a multilivelli, ed è il vero e proprio hardware del calcolatore.

Porte logiche

Un circuito digitale è un circuito in cui sono presenti solo due valori logici. Generalmente un valore (per esempio il numero binario 0) è rappresentato da un segnale compreso tra 0 e 1 volt, mentre l'altro valore (per esempio il numero binario 1) è rappresentato da un segnale compreso tra 2 e 5 volt.
La base hardware di tutti i calcolatori digitali è costituita da alcuni piccoli dispositivi elettronici, chiamati **porte logiche (gate)** ciascuna delle quali calcola una diversa funzione di questi segnali.

Composizione delle porte logiche

Tutta la moderna logica **digitale** si fonda sul fatto che un transistor può essere costruito in modo da funzionare come un velocissimo interruttore binario. I transistor hanno tre connessioni verso il mondo esterno: il **collettore,** la **base** e **l'emettitore.**
Quando la tensione in ingresso scende sotto un **valore critico**, chiamato V_{in}, il transistor viene disabilitato; la conseguenza è che l'output del circuito, V_{out}, assume un valore vicino a V_{α}. Questo tipo di tensione vale generalmente +5 volt. Quando, al contrario, V_{in} supera il valore critico, il transistor si attiva e si comporta come un **conduttore ideale**, facendo scaricare V_{out} a terra.



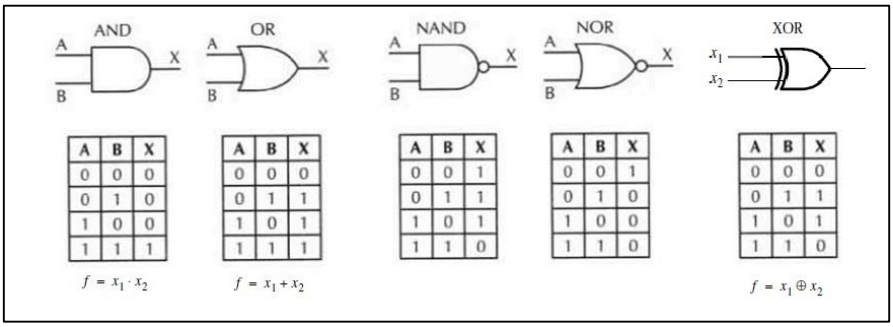
Da notare che quando V_{in} è basso, V_{out} è alto e viceversa. Il primo circuito nella figura a destra è quindi un invertitore che converte un valore logico 0 in un valore logico 1 e viceversa. La **resistenza** (la linea a zig zag) è necessaria per limitare la quantità di corrente nel transistor ed evitare che esso si fonda.

Il secondo circuito mostra invece due transistor collegati in serie; se V_1 e V_2 sono alte, allora entrambi i transistor saranno in conduzione e V_{out} sarà portato al valore basso, mentre, se entrambi gli ingressi sono bassi, allora i transistor corrispondenti saranno disattivati e l'uscita sarà alta. In altre parole V_{out} sarà alto se e soltanto se sia V_1 che V_2 sono alte.

Nel terzo circuito i due transistor sono collegati in parallelo, invece che in serie. In questa configurazione se uno dei due ingressi è alto, allora il transistor corrispondente sarà attivato e l'uscita sarà scaricata a terra, mentre, se entrambi gli ingressi sona bassi, l'uscita rimarrà alta.

Questi tre circuiti formano le tre porte logiche più semplici, chiamate rispettivamente NOT, NAND e NOR. Utilizziamo la convenzione di interpretare "alto" (Vcc volt) come un valore logico 1, e "basso" (messa a terra, simbolo ⏚) come un valore logico 0.

Tipi di porte logiche



Nella figura, A e B rappresentano gli ingressi, X l'uscita e ogni riga specifica il valore in uscita data una particolare combinazione dei valori in ingresso. Alcuni simboli sono gli inversi di altri, come NAND (not and) e NOR (not or). NOT è chiamato **invertitore**. Il pallino sta ad indicare l'inverso del risultato della porta, infatti lo troviamo nel NAND e nel NOR, oltre al NOT.

Algebra di boole

Per poter descrivere i circuiti costruiti combinando le porte logiche occorre definire un nuovo tipo di algebra in cui variabili e funzioni possono assumere soltanto i valori 0 e 1. Questa struttura viene chiamata algebra di Boole, in quanto è stata sviluppata dal matematico inglese George Boole (1815-1864). Per essere precisi, ci riferiamo in realtà a un particolare tipo di algebra booleana, chiamata **algebra booleana minimale**.

Funzioni booleane e tabelle di verità

Una funzione booleana ha una o più variabili di input e genera un valore di output che dipende soltanto dai valori di queste variabili. Una semplice funzione *f* può essere definita assumendo che *f(A)* valga 1 se A vale 0, e *f(A)* valga 0 se A vale 1; essa corrisponde alla funzione NOT (l'invertitore).

Una funzione booleana di *n* variabili può essere completamente descritta mediante una tabella di 2^n righe; dato che esistono 2^n possibili combinazioni dei valori di input, ciascuna riga può quindi indicare il valore della funzione per una data combinazione degli input. Una tabella di questo tipo è chiamata **tabella di verità** (come quella vista per i tipi di porte logiche). Se si decide di elencare sempre le righe di una tabella di verità in ordine numerico, cioè seguendo la sequenza 00,01, 10 e 11 nel caso di due variabili, la funzione può essere completamente descritta dal numero binario a 2^n bit che si ottiene leggendo in verticale la colonna del risultato. Adottando questa convenzione, NAND è 1110, NOR è 1000, AND è 0001 e OR è 0111. Con due variabili esistono ovviamente soltanto 16 diverse funzioni booleane, corrispondenti alle 16 combinazioni possibili della stringa a 4 bit che rappresenta i risultati.

Notazione alternativa

A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

La tabella di verità a sinistra mostra una funzione booleana a tre variabili: $M = f(A, B, C)$. Essa corrisponde alla funzione logica di maggioranza che vale 0 se la maggioranza dei suoi valori di input vale 0, mentre vale 1 se la maggioranza degli input vale 1.
Per indicare invece le funzioni booleane con più variabili si preferisce adottare una notazione alternativa: ogni funzione booleana può essere descritta specificando quali combinazioni delle variabili di input producono come risultato 1. Nel caso della figura a sinistra ci sono 4 combinazioni di variabili di input che rendono M pari a 1. Le variabili con una linea sopra per convenzione sono quelle invertite. Utilizzeremo inoltre la notazione implicita della moltiplicazione (*****) per denotare la funzione booleana AND (prodotto logico) e il simbolo della somma (**+**) per indicate la funzione booleana OR (somma logica).

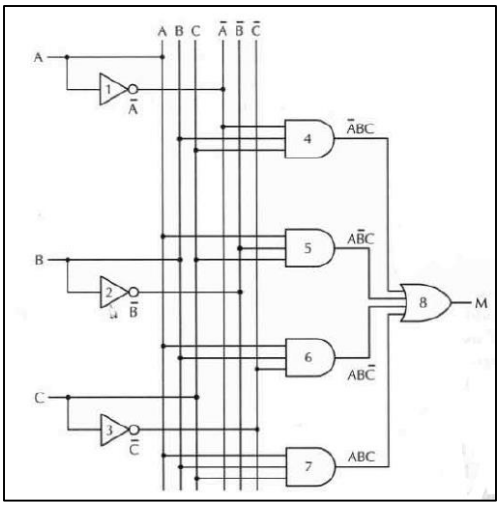
Es. $A\overline{B}C$ (con A=1, B=0 e C=1) assume valore 1, poiché $1 * 1 * 1 = 1$
$\overline{AB} + \overline{BC}$ (con A=1 e B=0, B=1 e C=0) assume valore 1, poiché $\overline{1 * 0} + \overline{1 * 0} = \overline{0} + \overline{0} = 1 + 1 = 1$

Le quattro righe della figura che producono in output il bit 1 sono: $\overline{A}BC, A\overline{B}C, AB\overline{C}\ e\ ABC$. Dunque dato che M è vera (vale 1) se una qualsiasi di queste quattro combinazioni è vera, è possibile riscriverla secondo la seguente notazione:

$$M = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

Una funzione di n variabili può quindi essere descritta mediante una somma di un massimo di 2^n termini, ciascuno dei quali è costituito dal prodotto logico di n variabili.

Implementazione circuitale di una funzione booleana

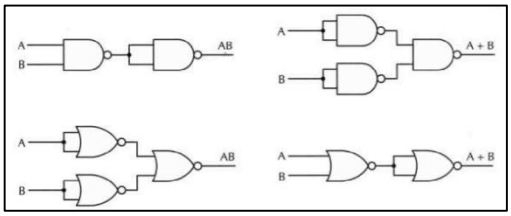


Una funzione booleana può essere implementata mediante un circuito elettronico che utilizza segnali per rappresentare le variabili di input e di output oltre ad alcune porte logiche come AND, OR e NOT.

Nel circuito a sinistra (che è costruito sulla base della tabella di verità sopra) gli input A, B e C sono rappresentati sul lato sinistro, mentre l'output della funzione è mostrato a destra. Dato che è necessario disporre degli inversi delle variabili da input, essi sono generati intercettando gli input e facendoli passare attraverso gli invertitoti indicati con i numeri 1, 2 e 3. Le sei linee verticali rappresentano il valore delle variabili, tre delle quali connesse alle variabili di input e le restanti tre ai loro complementi.
Il circuito contiene quattro porte AND, una per ciascun temine che compone l'equazione di M (cioè una per ogni riga della tabella di verità in cui il bit della colonna risultato vale 1). Ciascuna porta AND calcola una riga della tabella di verità, com'è indicato nella figura. Il risultato finale viene poi Calcolato effettuando l'OR fra tutti i prodotti.

Per implementare un circuito si attuano, dunque, i seguenti passi:

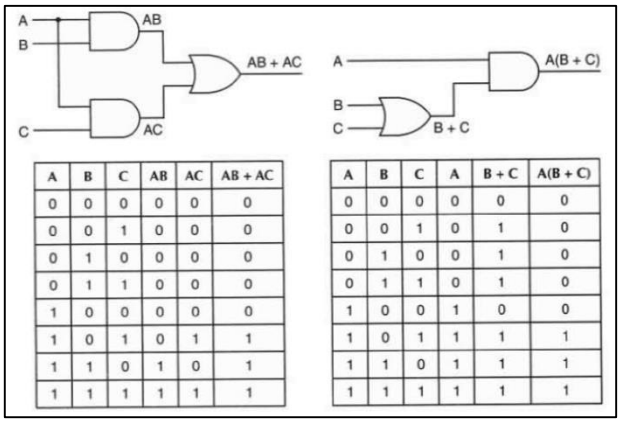
1. Si scrive la tabella di verità della funzione
2. Ci si munisce di invertitori per generare la negazione di ciascun input
3. Si utilizza una porta AND per ciascun termine il cui valore nella colonna risultato vale 1
4. Si collegano le porte AND agli input appropriati
5. Si connettono tutti gli output delle porte and nella porta OR



Fortunatamente esiste un modo diretto per convertire i circuiti generati dall'algoritmo precedente in uno che utilizzi unicamente la porta NAND, oppure la NOR. L'unica cosa che serve per effettuare una simile conversione è un modo per implementare le porte NOT, AND e OR utilizzando un solo tipo di porta logica. La figura a destra mostra l'implementazione di queste tre porte utilizzando soltanto porte NAND (riga in alto), oppure soltanto porte NOR (riga in basso).
Sia le porte NAND sia le porte NOR costituiscono un insieme di connettivi **funzionalmente completo**, nel senso che una qualsiasi funzione booleana può essere calcolata usando soltanto uno di questi due tipi di porte. Nessun'altra porta logica gode di questa proprietà; questo è un ulteriore motivo per il quale NAND e NOR sono spesso utilizzate come elementi base nella costruzione dei circuiti.

Equivalenza di circuiti

Spesso i progettisti cercano di ridurre il numero di porte logiche utilizzate nei loro prodotti per limitare il costo dei componenti, le dimensioni delle schede con i circuiti stampati, il consumo energetico e altri fattori. Per ridurre la complessità di un circuito un progettista deve trovare un altro circuito che calcoli la stessa funzione di quello originario, ma che sia composto da un numero inferiore di porte.



A	B	C	AB	AC	AB + AC
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

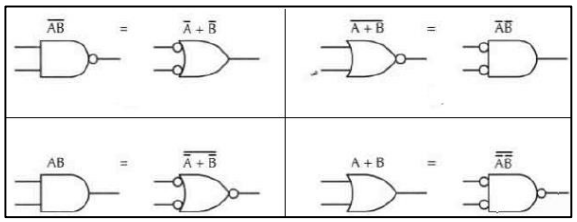
A	B	C	A	B + C	A(B + C)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Molte delie regole dell'algebra ordinaria valgono anche per l'algebra booleana; in particolare usando la proprietà distributiva, è possibile Fattorizzare AB +AC nella forma A(B+C).

Dato che due funzioni sono equivalenti se e solo se hanno lo stesso valore di output per tutti i possibili input, è facile verificare dalle tabelle di verità della figura in alto che A(B+C) e AB +AC sono effettivamente equivalenti. Pur essendo equivalenti il circuito della figura a destra è chiaramente migliore di quello della figura a sinistra, in quanto è composto da un numero minore di porte logiche. Per poter dunque cercare di semplificare un circuito, bisogna conoscere le seguenti formule:

Nome	Forma AND	Forma OR
Elemento neutro	1A=A	0+A=A
Assorbimento	0A=0	1+A=1
Idempotenza	AA=A	A+A=A
Complementazione	A$\overline{A}$=0	A+$\overline{A}$=A
Proprietà commutativa	AB=BA	A+B=B+A
Proprietà associativa	(AB)C=A(BC)	(A+B)+C=A+(B+C)
Proprietà distributiva	A+BC=(A+B)(A+C)	A(B+C)=AB+AC
Legge di assorbimento	A(A+B)=A	A+AB=A
Legge di De Morgan	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{AB}$

La legge di De Morgan si distingue sia perché può essere estesa a tre variabili ($\overline{ABC} = \overline{A} + \overline{B} + \overline{C}$), sia perché ha una notazione alternativa.



Nella figura a sinistra il riquadro in alto a sinistra mostra la forma AND in cui la negazione è indicata, sia per gli input sia per gli output, mediante i cerchietti di inversione. Una porta logica OR con gli input invertiti è quindi equivalente ad una porta NAND. Dal riquadro in alto a destra che rappresenta la forma duale della legge di De Morgan, dovrebbe essere chiaro che una porta NOR può essere disegnata come una porta AND con gli input invertiti. Negando entrambe le forme della legge di De Morgan si ottengono le figure rimanenti, che mostrano due rappresentazioni equivalenti delle porte logiche AND e OR.

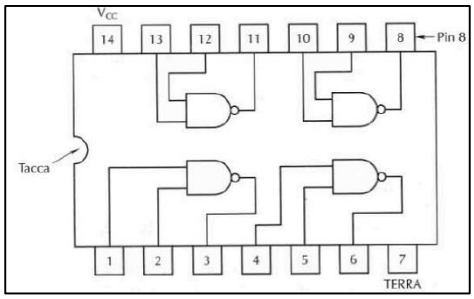
Circuiti integrati

Le porte logiche sono prodotte in unità chiamate **circuiti integrati**, alle quali spesso ci si riferisce con i termini **IC** o **chip.** Un IC è un quadrato di silicio (di circa 5 mm x 5 mm) sul quale sono state posizionate alcune porte. Di solito i piccoli IC sono montati in un contenitore rettangolare di plastica o ceramica. Sul lato più lungo vi sono due file parallele di contatti (***pin***) inseribili in una presa (***socket***) oppure saldabili su schede di circuiti stampati. Ciascun pin è collegato a un input o a un output di una porta logica oppure all'alimentazione o alla messa a terra. Con il termine tecnico **DIP** (*Dual Inline Package*) ci si riferisce al contenitore che all'esterno presenta due file di pin e al suo interno alcuni IC; tutti però lo chiamano chip, rendendo in questo modo più sfocata la distinzione tra il pezzo di silicio e il suo contenitore.

I chip possono essere classificati in base al numero di porte che contengono:

- Circuiti SSI (Small Scale Integrated): da 1 a 10 porte
- Circuiti MSI (Medium Scale Integrated): da 10 a 100 porte
- Circuiti LSI (Large Scale Integrated): da 100 a 100.000 porte
- Circuiti VLSI (Very Large Scale Integrated): più di 100.000 porte

Queste classi hanno proprietà diverse e vengono utilizzate in modi differenti.

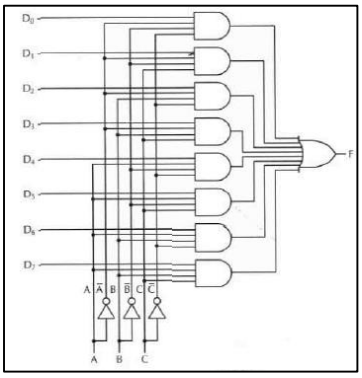


La figura a sinistra rappresenta lo schema di un comune chip SSI con quattro porte NAND. Dato che ciascuna di queste porte ha due input e un output sono necessari 12 pin per le quattro porte. Inoltre il chip necessita della tensione (V_{cc}) e della terra (GND), che sono condivise da tutte le porte.
Negli anni '70 i calcolatori erano costruiti con un gran numero di chip di questo tipo, mentre al giorno d'oggi sono stampate all'interno di un unico chip un intera CPU e una significativa quantità d memoria (cache) .

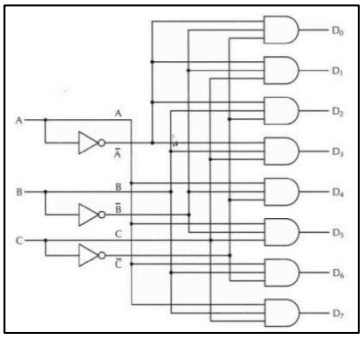
Dato che ogni circuito può essere costruito mediante porte NAND, si potrebbe pensare che un produttore porrebbe realizzare un chip estremamente generalizzato contenente 5 milioni di porte NAND. Sfortunatamente però un chip di questo tipo richiederebbe 15.000.000 pin; considerando che la distanza standard tra i pin è di 0,1 pollici, il chip supererebbe i 19 km! È chiaro che l'unico modo per trarre vantaggio dallo stato attuale della tecnologia consiste nel progettare circuiti con un elevato rapporto porte/pin.

Reti combinatorie

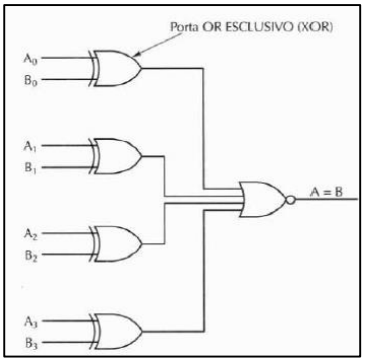
Molte applicazioni della logica digitale richiedono un circuito, chiamato **rete combinatoria**, con più input e più output, in cui gli output sono unicamente determinati dagli input. Non tutti i circuiti hanno questa proprietà; un circuito contenente elementi di memoria può per esempio generare valori che dipendono non solo dalle variabili d'ingresso, ma anche dai valori memorizzati.
Le reti combinatorie utilizzate più frequentemente sono:



- **Multiplexer**: circuito con 2^n dati di input, un valore di output e *n* input di controllo.
La figura a destra ne mostra un esempio con 8 input. Le tre linee di controllo, A, B e C, codificano un numero a 3 bit che specifica quale delle otto linee di input deve essere instradata versa la porta OR e quindi verso l'output. Indipendentemente dal valore definito dalle linee di controllo, sette delle porte AND genereranno sempre il valore 0, mentre quella rimanente produrrà in output 0 oppure 1, a seconda del valore della linea d'ingresso selezionata. Aggiungendo anche la tensione e la terra, può essere confezionato in un contenitore dorato di 14 pin.
L'inverso del multiplexer è il **demultiplexer**, che ridirige il suo segnale di input verso une dei 2^n output in base ai valori delle linee di controllo; se il valore binario definito dalle linee di controllo è k, viene selezionato l'output k.

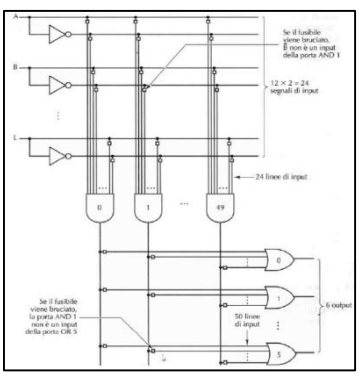


- **Decodificatore:** accetta come input un numero a *n* bit e lo utilizza per impostate a 1 una sola delle 2^n linee di output. La figura a destra mostra il caso di n=3. Per capire in quali situazioni può essere utile questo circuito immaginiamo una piccola memoria di otto chip, da 1 MB ciascuno. Gli indirizzi del chip 0 variano da 0 a 1 MB, quelli del chip 1 da 1 MB a 2 MB, e così via. Quando si fornisce alla memoria un indirizzo, si utilizzano i suoi 3 bit più significativi per selezionare uno degli otto chip. In riferimento ai circuito a destra, questi 3 btc corrispondono ai tre input, A, B e C; a seconda del loro valore solo una delle linee di output, assume il valore 1, mentre le altre rimangono a 0. Ciascuna porta AND ha tre input, il primo dei quali è A o $\overline{A}$,il secondo il secondo B o $\overline{B}$ e il terzo C o $\overline{C}$, e viene abilitata da una diversa combinazione dei valori di input: D_0, da $\overline{A}\overline{B}\overline{C}$, D_1 da $\overline{A}\overline{B}C$,e così via.



- **Comparatore**: permette di confrontare due stringhe di bit. Il comparatore a destra accetta due input, A e B, ciascuno lungo 4 bit, e genera 1 se sono uguali, mentre 0 se sono diversi. Il circuito è basato sulla porta logica **XOR** (EXCLUSIVE OR), che produce in output un valore 0 se i suoi input sono uguali, e un valore 1 se sono diversi. Se due stringhe in ingresso sono uguali, tutte e quattro le porte XOR devono generare come risultato 0. Questi quattro segnali possono poi essere connessi a una stessa porta logica OR in modo da produrre un valore 0 quando gli input sono uguali e un valore 1 nel caso contrario. Nel nostro esempio abbiamo utilizzato una porta NOR nell'ultimo stadio del circuito in modo da invertire il risultato del test: 1 significa uguale, mentre 0 significa diverso.

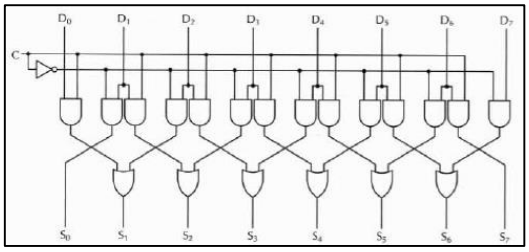
- **Array logici programmabili**: è un chip molto generale che permette di calcolare somme di prodotti. Questo chip ha 12 ingressi e al suo interno questi valori vengono invertiti; quindi il numero totale di segnali di input diventa 24. Il cuore del circuito è costituito da una schiera di 50 porte AND che possono avere come input un qualsiasi sottoinsieme dei 24 segnali di input. Una matrice di 24 x 50 bit fornita dall'utente determina le connessioni desiderate tra i segnali di input e le 50 porte AND. Ogni linea di input connessa alle 50 porte AND contiene un fusibile. Al momento della fabbricazion del chip i 1200 fusibili sono intatti; successivamente l'utente può programmare la matrice bruciando i fusibili con l'applicazione di alta tensione.
L'uscita del circuito consiste in 6 porte OR, che possono avere fino a 50 input, corrispondenti agli output delle porte AND. Anche in questo caso l'utente deve fornire una matrice (50 x 6) per specificare quali connessioni instaurare. Il chip ha 20 pin in tutto, 12 per gli input, 6 per gli output, e 2 per la tensione e la terra.



Circuiti per l'aritmetica

Le reti combinatorie sono soprattutto impiegate per eseguire calcoli aritmetici. Esse utilizzano un semplice registro a scorrimento (*shifter*) a 8 bit.

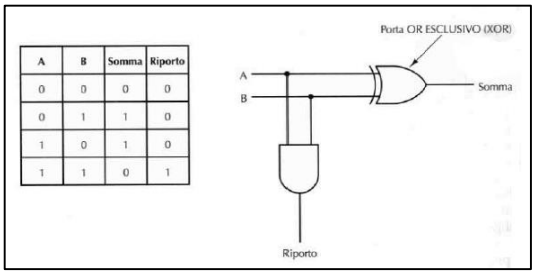
Registri a scorrimento



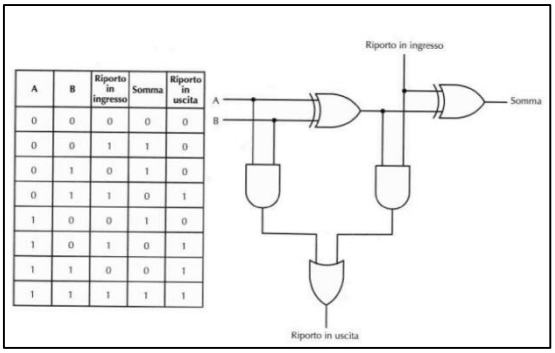
È un circuito aritmetico MSI avente otto input e otto output. Gli input sono collegati alle line $D_0, \dots, D_7$, mentre l'output risulta disponibile sulle linee $S_0, \dots, S_7$. La linea di controllo, C, ha valore 0 se lo spostamento deve avvenire verso sinistra, e 1 in caso contrario. Nel caso di uno spostamento a sinistra si inserisce uno 0 nel bit 7, e nel caso di uno shift a destra si inserisce un 1 nel bit 0.

Per comprendere il funzionamento del circuito si noti che per tutti i bit c'è una coppia di porte AND, fatta eccezione per le porte che si trovano alle estremità. Quando C= 1 la porta che si trova a destra in ciascuna coppia viene abilitata, lasciando passare il bit corrispondente verso l'output. Dato che la porta AND è collegata all'input della porta OR alla sua destra, si ottiene uno scorrimento verso destra. Quando C = 0 è la porta AND di sinistra in ciascuna coppia a essere abilitata, producendo uno spostamento verso sinistra.

Sommatori



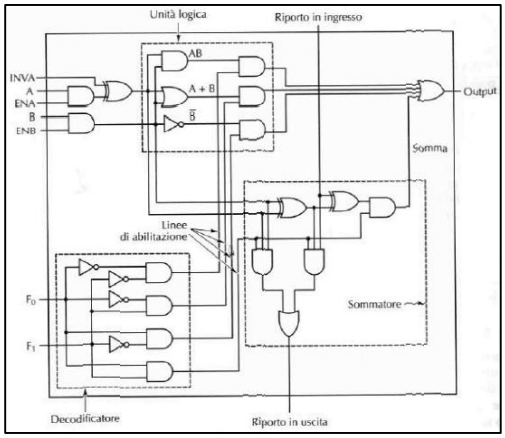
Un circuito che effettui l'addizione è una parte di essenziale importanza per qualsiasi CPU. La figura a sinistra rappresenta un circuito, chiamato **half adder** (semisommatore, poiché non tiene conto del riporto in ingresso di una addizione come 1+1 che in binario fa 0 con riporto 1), capace di calcolare il bit della somma e il bit del riporto. Questo circuito è in grado di sommare correttamente i bit meno significativi di due stringhe binarie, ma non è in grado di eseguire correttamente la somma degli altri bit, dato che non riesce a gestire il riporto che arriva dalle posizioni precedenti. Per far ciò è necessario un **sommatore** come quello mostrato nella figura a destra, in cui notiamo che un sommatore è costruito da due semisommatori.



La linea di output Somma vale 1 se una o tre delle linee A, B e Riporto in ingresso (*Carry-in*) in valgono 1. Il Riporto in uscita (*Carry-out*) vale1 se sia A che B valgono 1 (input di sinistra della porta OR) oppure se uno dei due vale 1 e anche Carry-in vale 1.

Per generare un sommatore di due parole a 16 bit occorre replicare il circuito per 16 volte. Per un dato bit il riporto in uscita viene utilizzato come riporto in entrata per il bit alla sua sinistra. Il riporto in entrata del bit all'estremità destra è collegato al valore 0. Questo tipo di sommatore è chiamato **sommatore a propagazione di riporto**, dato che nel caso peggiore, sommando 1 a 111…111 (binario), la somma non può essere completata finchè il riporto non si sia propagato lungo tutta la parola, dal bit all'estremità destra fino a quello all'estremità sinistra.

Unità aritmetico logiche



La maggior parte dei calcolatori contiene un unico circuito in grado di effettuare le operazioni AND, OR e somma di due parole.

La figura a sinistra è un esempio di tale circuito, chiamato **unità aritmetico logica** o **ALU**; esso può calcolare una qualsiasi delle quattro funzioni, A AND B, A OR B, $\overline{B}$ oppure A+B (somma aritmetica), in base al valore binario (00, 01, 10 oppure 11) definito dalle linee F_0, e F_1, preposte alla selezione della funzione aritmetica.

Il settore in basso a sinistra della nostra ALU contiene un decodificatore a 2 bit che permette di generare, in base ai segnali di controllo F_0 e F_1, i segnali di attivazione delle quattro operazioni. In base ai valori di F_0 e F_1, viene selezionata una sola delle quattro linee di attivazione, permettendo all'output della funzione selezionata di passare attraverso la porta logica OR che genera l'output finale.

Il settore in alto a sinistra contiene le porte logiche necessarie per calcolare A AND B, A OR B e $\overline{B}$. Dato che solo uno degli output del decodificatore varrà 1, soltanto una delle quattro porte AND che forniscono i valori d'ingresso alla porta OR verrà attivata; l'output delle altre sarà invece 0, indipendentemente dai valori di A e B. Oltre a poter usare A e B come input per le operazioni logiche o aritmetiche, è anche possibile forzare uno dei due valori a 0 negando rispettivamente ENA oppure ENB. È inoltre possibile ottenere il valore di A abilitando il segnale INVA.

Il settore in basso a destra della ALU contiene un sommatore per calcolare la somma di A e B, e gestire allo stesso tempo i riporti.

Circuiti come questi sono attualmente disponibili e vengono chiamati **bit slices** ("suddivisioni di un bit"), che permettono ai progettisti di calcolatori di costruire ALU di larghezza arbitraria.

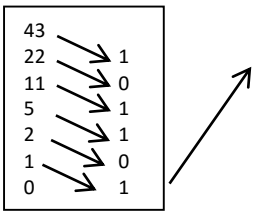
Accenni sugli esercizi

Primo vero computer: calcolatore programmabile di Babbage

Calcolo binario

Conversione da decimale a binario: si divide ripetutamente il numero per 2, se il risultato ha resto si scrive 1 e si tronca, se non lo ha si mette 0, fino a quando il numero diventa 0. I risultati dei resti vanno poi letti al contrario

Es. 43 101101 Dunque $43_{(10)} = 101101_{(2)}$



Conversione da binario a decimale: si prende la prima cifra del numero decimale e si moltiplica per 2 elevato alla grandezza massima dell'indice numero decimale e si somma alla seconda a cui applichiamo questo processo, e così via.

Es. 101101 $1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 32 + 8 + 4 + 1 = 45$

Li stessi metodi possono essere applicati anche a basi diverse da 2

Aritmetica binaria: 0+0=0 0+1=1 1+0=1 1+1=0 (riporto 1, CARRY)
0-0=0 1-1=0 1-0=1 0-1=1 (prestito 1, BORROW)

Es.

$$\begin{array}{r} 0011 \\ +\ 0010 \\ \hline 0101 \end{array}$$

il quale in base 10 vuol dire 3+2=5

$$\begin{array}{r} 0101 \\ -\ 0010 \\ \hline 0011 \end{array}$$

Scalamento (Shift): lo shift a sinistra di un bit (inserendo uno 0) equivale a moltiplicare per due.

Es. $0011 = 3_{10}$ $0011 \leftarrow 0$ $0110 = 6_{10}$

Lo shift a destra di un bit (inserendo uno 0) equivale a dividere per 2 (con troncamento).

Es. $0100 = 4_{10}$ $0 \rightarrow 0100$ $0010 = 2_{10}$

Rappresentazione dei numeri interi relativi: un numero relativo è composto da un segno ($\pm$) e da un modulo (n). Nel calcolo binario

$+ \rightarrow 0$
$- \rightarrow 1$

Es. $9 \rightarrow 01001$ $-9 \rightarrow 101001$ $+17 \rightarrow 010001$

Complemento a uno: si prende il numero con il modulo a segno negativo e si fa il complementare di tutti i bit del numero

Es. $+6 \rightarrow 01100$(modulo e segno) $-6 \rightarrow 1110$ (modulo e segno) $\rightarrow 1001$ (complemento a 1)

Complemento a due: si calcola il complemento a 1 del numero e poi viene sommata una unità
Es. $-6 \rightarrow 1001$ (complemento a 1) $\rightarrow 1010$ (complemento a 2), rappresenta numeri interi relativi

Somma algebrica in complemento a due: l'addizione è come quella binaria naturale, ma trascurando il riporto in uscita

Normale Complemento a 2

0010	(+2)		0100	(+4)	
+ 0011	(+3)		+ 1010	(−6)	
0101	(+5)		1110	(−2)	
1011	(−5)		0111	(+7)	
+ 1110	(−2)		+ 1101	(−3)	
1001	(−7)		0100	(+4)	

La sottrazione è invece l'addizione con il complemento a due del sottraendo

Sottrazione			Complemento a 2	
1101	(−3)	⇒	1101	
− 1001	(−7)		+ 0111	
			0100	(+4)
0010	(+2)	⇒	0010	
− 0100	(+4)		+ 1100	
			1110	(−2)

Trabocco (overflow): con i complementi a 2, i valori di n bit possono essere rappresentati nel raggio di $[-2^{n-1}, 2^{n-1}-1]$.

Es. 4 bit possono rappresentare da -8 a +7

Se il risultato di un'operazione aritmetica travalica questi limiti, non è valido: si ha un evento di **trabocco (*overflow*).** Quando sommiamo un numero senza segno, un carry-out di 1 dalla più significate posizione del bit indica che si è scatenati un overflow. Si hanno due semplici regole:

- addendi *discordi*: non può aversi trabocco
- addendi *concordi*: si può avere il trabocco quando il valore della stringa di bit risultata dalla somma è già presente nel raggio di numeri.

Rappresentazione dei numeri con virgola: i numeri con la virgola sono divisi in numeri interi (quelli prima della virgola), e in numeri decimali (dopo la virgola). Gli indici dei numeri dopo la virgola vanno da -1 a ∞

Es. 10.101 ⇒ $2^1\ 2^0\ 2^{-1}\ 2^{-2}\ 2^{-3}$ ⇒ $1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$

$2 + 2^{-1} + 2^{-3} = 2 + 0{,}5 + 0{,}125 = 2{,}625$

Rappresentazione in virgola mobile: la notazione scientifica ci dice che $217{,}32 \Rightarrow 0{,}21732 \cdot 10^{+3}$ *o* $21732 \cdot 10^{-2}$
La rappresentazione di tali numeri nell'elaboratore è data da:

$N = \pm M \cdot 2^E$ dove N=numeri, M=mantissa, E=esponente

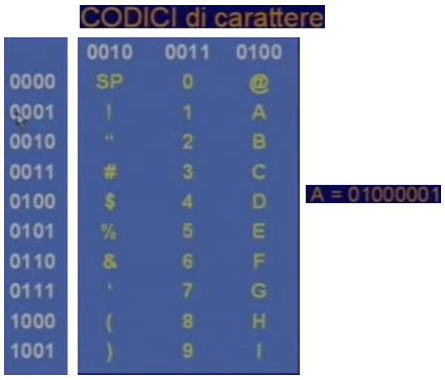
Gli elaboratori odierni impiegano 4 byte (32bit) o 8 byte (64bit), la codifica è articolata in:

Segno | Esponente (compl a 2) | Mantissa (bit rimanenti)

Intervallo di rappresentazione: 32bit ⇒ $\pm 10^{\pm 38}$ 64bit ⇒ $\pm 10^{\pm 308}$

Codifica dei caratteri

Codice unitario riconosciuto dai PC = **ASCII** (American Standard Code for Information Interchange). Si estende su 8 bit ⇒ 1 byte per carattere; 256 codifiche che corrispondono a: codici di controllo (es. "a capo"), codici dei caratteri alfanumerici (0,1,…,A,B,a,b,…,!,:), codici di simboli particolari (non standard, gli ultimi 128).



Per rappresentare la tabella dei codici ASCII è necessario fare riferimento alla tabella a sinistra di corrispondenza di bit.

Nelle colonne troviamo la prima parte del codice binario, mentre nelle righe la seconda parte.

Sulla base di questa tabella è possibile intuire come funziona un documento:

Marco = M=01001101 (1° byte), A=01000001 (2° byte), …

Estensioni del codice ASCII per **l'internazionalizzazione**:

- famiglia di codici **ISO 8859-*x*** : a 8 bit, estensioni nazionali

- **UCS** (ISO/IEC 10646: *Universal Character Set*): base per
- **Unicode, UTF-8** : a lunghezza variabile (multipla di 8)

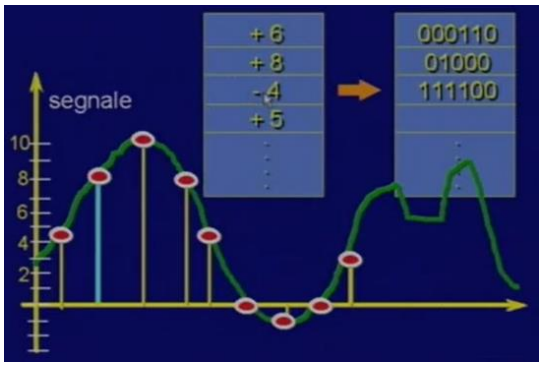
Codifica delle immagini

Monitor: costituito da una sequenza di pallini (pixel) che possono accendersi e spegnersi come i bit 0 e 1 per rappresentare l'immagine. Ad esempio uno schermo 800x600 è composto da 800 righe di pixel e 600 colonne di pixel. Il singolo elemento (punto del disegno) è detto **PIXEL** (picture element), quanti più sono i pixel, tanto maggiore è la precisione e la qualità dell'immagine.

Codifica dei suoni

Come viene trasferito un suono in segnale elettrico: onde acustiche ⇒ (trasduttore, es. microfono) ⇒ onde elettriche ⇒ (trasduttore, es. altoparlante) ⇒ onde acustiche.

Il meccanismo che ci permette di trasformare un onda elettrica in una sequenza di bit vene detta **campionamento**. Esso consiste nel valutare ad intervalli fissi il valore del segnale, e di codificarlo come sequenza numerica.



Es. Supponiamo di avere un segnale elettrico (andamento della tensione o corrente). Il campionamento prende i valori +5, +8, -4, +8, +6,… i quali numeri interi relativi vengono convertiti in binario.

Data questa sequenza di bit ricaviamo dei valori che messi in sequenza costituiscono il segnale acustico che possiamo ricostruire.

Il dispositivo che permette di realizzare questa trasformazione è **l'ADC** (convertitore analogico digitale) che riceve i dati dal trasduttore e li converte in bit.

Registrazione: Onda sonora ⇒ trasduttore ⇒ ADC ⇒ $(campionamento)$ 010001
Trasmissione: 010001 ⇒ DAC(convertitore digitale analogico) ⇒ trasduttore

Questo avviene anche quando noi parliamo al microfono.