Sommario

[John Von Neumann 6](#_Toc92985745)

[Calcolatore elettronico 6](#_Toc92985746)

[Circuiti di commutazione 6](#_Toc92985747)

[Rete di commutazione 6](#_Toc92985748)

[Algebra di Boole 6](#_Toc92985749)

[Operazioni booleane 6](#_Toc92985750)

[Tabella di verità AND 6](#_Toc92985751)

[Simbolo porta logica AND 6](#_Toc92985752)

[Tabella di verità OR 7](#_Toc92985753)

[Simbolo porta logica OR 7](#_Toc92985754)

[Tabella di verità NOT 7](#_Toc92985755)

[Simbolo porta logica NOT 7](#_Toc92985756)

[Proprietà dell’algebra di Boole 7](#_Toc92985757)

[Letterale 8](#_Toc92985758)

[Clausola 8](#_Toc92985759)

[Fattore elementare 8](#_Toc92985760)

[Implicante 8](#_Toc92985761)

[Primo implicante 8](#_Toc92985762)

[Implicante essenziale 8](#_Toc92985763)

[Mintermine 8](#_Toc92985764)

[On-set 8](#_Toc92985765)

[Maxtermine 8](#_Toc92985766)

[Off-set 8](#_Toc92985767)

[Copertura 8](#_Toc92985768)

[Prima forma canonica 8](#_Toc92985769)

[Seconda forma canonica 8](#_Toc92985770)

[Rappresentazione dei numeri 8](#_Toc92985771)

[Base binaria, ottale, esadecimale 8](#_Toc92985772)

[Rappresentazione binaria pura 9](#_Toc92985773)

[Rappresentazione di un numero N in base r 9](#_Toc92985774)

[Rappresentazione di un numero N in base r 9](#_Toc92985775)

[Overflow 9](#_Toc92985776)

[Rappresentazione in segno e modulo 9](#_Toc92985777)

[Rappresentazione in complemento alla base 9](#_Toc92985778)

[Complemento diminuito 9](#_Toc92985779)

[Rappresentazione in eccesso-k 9](#_Toc92985780)

[Virgola fissa 9](#_Toc92985781)

[Underflow 9](#_Toc92985782)

[Virgola mobile 10](#_Toc92985783)

[Standard IEEE 754 10](#_Toc92985784)

[Rete combinatoria 10](#_Toc92985785)

[Progettare una rete combinatoria 10](#_Toc92985786)

[Analisi di un sistema 10](#_Toc92985787)

[Sintesi di un sistema 10](#_Toc92985788)

[Circuito combinatorio 10](#_Toc92985789)

[Rete sequenziale 10](#_Toc92985790)

[Circuito sequenziale 10](#_Toc92985791)

[Automa a stati finiti 11](#_Toc92985792)

[Decoder binario 11](#_Toc92985793)

[Macchina sequenziale 12](#_Toc92985794)

[Sintesi di una macchina sequenziale 12](#_Toc92985795)

[Sintesi di macchine sequenziali sincrone 12](#_Toc92985796)

[Tabella degli stati 12](#_Toc92985797)

[Diagramma degli stati 13](#_Toc92985798)

[Sistemi asincroni 13](#_Toc92985799)

[Sistemi sincroni 13](#_Toc92985800)

[Sistemi digitali 13](#_Toc92985801)

[Latch e Flip Flop 13](#_Toc92985802)

[Bistabili 14](#_Toc92985803)

[Tabella delle transizioni 14](#_Toc92985804)

[Tabella delle eccitazioni 14](#_Toc92985805)

[Latch asincroni 14](#_Toc92985806)

[Latch sincroni 14](#_Toc92985807)

[Latch SR (Set-Reset) 14](#_Toc92985808)

[Latch JK 14](#_Toc92985809)

[Latch D 14](#_Toc92985810)

[Latch trigger o toggle 15](#_Toc92985811)

[Sincronia 15](#_Toc92985812)

[Bistabile JKT 15](#_Toc92985813)

[Bistabile DT 15](#_Toc92985814)

[Bistabili Master-Slave 15](#_Toc92985815)

[Flip-Flop Master-Slave 15](#_Toc92985816)

[Macchine per il trattamento di codici 15](#_Toc92985817)

[Codice 15](#_Toc92985818)

[Informazione 15](#_Toc92985819)

[Tipi 16](#_Toc92985820)

[Cardinalità 16](#_Toc92985821)

[Alfabeto 16](#_Toc92985822)

[Stringa 16](#_Toc92985823)

[Lunghezza di una stringa 16](#_Toc92985824)

[Codifica 16](#_Toc92985825)

[Codice a lunghezza fissa 16](#_Toc92985826)

[Codice incompleto 16](#_Toc92985827)

[Codice ridondante 16](#_Toc92985828)

[Codice a lunghezza variabile 16](#_Toc92985829)

[Rappresentazione decodificata 17](#_Toc92985830)

[Decoder (Decodificatore) 17](#_Toc92985831)

[Bus 17](#_Toc92985832)

[Trasferimento dati da bus a registro 17](#_Toc92985833)

[Trasferimento dati da registro a bus 17](#_Toc92985834)

[AND Tristate 17](#_Toc92985835)

[OR di bus 17](#_Toc92985836)

[Registri a scorrimento 17](#_Toc92985837)

[Memoria 18](#_Toc92985838)

[DMA (Direct Memory Access) 18](#_Toc92985839)

[Memoria cache 18](#_Toc92985840)

[Livelli della cache 18](#_Toc92985841)

[Storage (memoria persistenti) 18](#_Toc92985842)

[Memorie volatili 18](#_Toc92985843)

[SRAM (Static Random Access Memory) 18](#_Toc92985844)

[DRAM (Dynamic Random Access Memory) 18](#_Toc92985845)

[SDRAM (Synchronous Dynamic Random Access Memory) 19](#_Toc92985846)

[FeRAM (Ferroelectic Dynamic Random Access Memory) 19](#_Toc92985847)

[Dischi magnetici 19](#_Toc92985848)

[Hard Disk 19](#_Toc92985849)

[Scheda video 19](#_Toc92985850)

[Compito della scheda video 19](#_Toc92985851)

[Tubo a raggio catodico 19](#_Toc92985852)

[Monitor CRT 20](#_Toc92985853)

[Monitor LCD 20](#_Toc92985854)

[VGA (Video Graphics Array) 20](#_Toc92985855)

[DVI (Digital Visual Interface) 20](#_Toc92985856)

[HDMI (High-Definition Multimedia Interface) 20](#_Toc92985857)

[DP (DisplayPort) 20](#_Toc92985858)

[Colori 20](#_Toc92985859)

[Memoria video 20](#_Toc92985860)

[AGP (Accelerated Graphic Port) 20](#_Toc92985861)

[PCIe (PCI Express) 21](#_Toc92985862)

[SLI e CrossfireX 21](#_Toc92985863)

[GPU (Graphical Processing Unit) 21](#_Toc92985864)

[Device driver 21](#_Toc92985865)

[GPGPU (General Purpose Graphical Processing Unit) 21](#_Toc92985866)

[Input/Output 21](#_Toc92985867)

[Periferiche di input 21](#_Toc92985868)

[Periferiche di output 21](#_Toc92985869)

[Periferiche di input/output 21](#_Toc92985870)

[Modelli di collegamento 21](#_Toc92985871)

[Bus di collegamento 21](#_Toc92985872)

[Tastiera 21](#_Toc92985873)

[Periferiche 22](#_Toc92985874)

[Controller 22](#_Toc92985875)

[Terminale 22](#_Toc92985876)

[Bus 22](#_Toc92985877)

[Processore 22](#_Toc92985878)

[Processore o CPU (Central Processing Unit) – definizione 22](#_Toc92985879)

[Modello di riferimento 23](#_Toc92985880)

[Istruzioni 23](#_Toc92985881)

[Registri 23](#_Toc92985882)

[Stack 24](#_Toc92985883)

[Stack Pointer (SP) 24](#_Toc92985884)

[Processore a voce 24](#_Toc92985885)

[Processore a byte 24](#_Toc92985886)

[Stati del processore 24](#_Toc92985887)

[Ciclo di un processore 24](#_Toc92985888)

[Interruzioni 24](#_Toc92985889)

[Latenza di un’interruzione 24](#_Toc92985890)

[Assembly 25](#_Toc92985891)

[Architettura del processore MC68000 25](#_Toc92985892)

[Istruzioni assembly 25](#_Toc92985893)

[Indirizzamenti 25](#_Toc92985894)

[Istruzione MUL 26](#_Toc92985895)

[Istruzione BTST 26](#_Toc92985896)

[Istruzione BCLR 26](#_Toc92985897)

[Istruzione BSET 26](#_Toc92985898)

[Istruzione BCHG 26](#_Toc92985899)

[Sottoprogramma 26](#_Toc92985900)

[Passaggio dei parametri 26](#_Toc92985901)

## John Von Neumann

*John von Neumann* fu colui che costituì *l’architettura di von Neumann*, un’architettura hardware per computer digitali programmabili a *programma memorizzato*. In quest’architettura abbiamo:

* *Unità di controllo*: legge un dato da una memoria; il risultato sarà un codice che indica quale operazione effettuare successivamente.
* *CPU*
* *Memoria*
* *Unità aritmetico-logica (ALU)*: è un sistema che effettua operazioni di calcolo.

Secondo l’architettura di von Neumann, le varie unità sono collegate tra loro tramite *bus*.

## Calcolatore elettronico

Un *calcolatore elettronico* è costituito da:

* *CPU (Processore)*: costituisce la capacità elaborativa del calcolatore; può eseguire un set di azioni elementari più o meno complesse.
* *Memoria centrale*
* *Sottosistema di Input/Output (I/O)*

Il *calcolatore elettronico* è un sistema:

* *Numerico*, cioè deve memorizzare numeri secondo una codifica.
* *Automatico*, cioè la sequenza di azioni va avanti da sola
* *A programma registrabile*, la logica di controllo deve essere generica.

Esso viene realizzato mediante circuiti elettronici.

## Circuiti di commutazione

Tutte le informazioni che vengono rappresentate da un dispositivo numerico sono espresse in forma di bit.

I circuiti elettronici preposti alla memorizzazione ed elaborazione di bit sono detti *circuiti di commutazione* o *circuiti logici* (*switching circuits*).

### Rete di commutazione

Un *circuito logico* costituisce una *rete di commutazione*, ed il relativo progetto viene definito *logico*.

## Algebra di Boole

*L’algebra di Boole* è utilizzata per la realizzazione di circuiti logici. In questo tipo di algebra le variabili:

* Sono *binarie* (possono assumere valori *0*, *1*)
* Si indicano con *lettera maiuscola*.

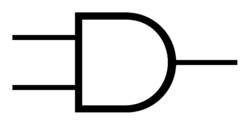
### Operazioni booleane

Le principali operazioni dell’algebra di Boole sono: *AND* (), *OR* () e *NOT* ()

### Tabella di verità AND

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

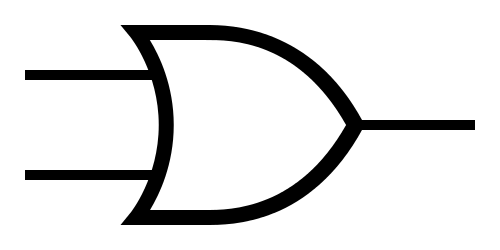
### Simbolo porta logica AND



### Tabella di verità OR

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

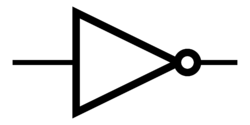
### Simbolo porta logica OR



### Tabella di verità NOT

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

### Simbolo porta logica NOT



### Proprietà dell’algebra di Boole

* *Commutativa*
* *Associativa*: per questa proprietà AND e OR possono essere definiti con più operandi
* *Idempotenza*
* *Assorbimento*
* *Distributiva*
* *Convoluzione*
* *Massimo*
* *Minimo*
* *Teorema di De Morgan*

### Letterale

Un *letterale* è una coppia *(variabile, valore)*. Ogni variabile ha due letterali () () che vengono indicati rispettivamente con e .

### Clausola

Una *clausola* è il prodotto logico di più letterali, negati o meno.

### Fattore elementare

Un *fattore elementare* è la somma logica di più letterali, negati o meno.

### Implicante

Un *implicante* è un prodotto di letterali che se assume valore 1, la funzione assumerà valore 1.

### Primo implicante

Un implicante si definisce *primo* se non esiste nessun altro implicante di dimensioni maggiori (ovvero con minor numero di letterali) che lo contenga interamente.

### Implicante essenziale

Un implicante si definisce *essenziale* se esiste un mintermine coperto dall’implicante che non è coperto da nessun altro implicante della funzione.

### Mintermine

Un *mintermine* consiste nell’avere tutte le variabili che appaiono come letterali.

### On-set

*L’On-set* consiste nell’insieme dei mintermini, ovvero l’insieme delle possibili combinazioni di valori in ingresso tali che l’uscita vale 1.

### Maxtermine

Un *maxtermine* si ha quando la funzione vale 0.

### Off-set

*L’Off-set* consiste nell’insieme dei maxtermini, ovvero l’insieme delle possibili combinazioni in ingresso tali che l’uscita vale 0.

### Copertura

La *copertura* è l’insieme di implicanti che coprono tutti i mintermini (o maxtermini) della funzione

### Prima forma canonica

La *prima forma canonica* di una funzione consiste nella somma di prodotti:

Dove:

* sono i mintermini contenuti nell’On-set di .
* Ogni mintermine è realizzato con una porta *AND* a ingressi.
* Le uscite delle porte che rappresentano i mintermini sono collegate tramite una porta *OR* a ingressi.

### Seconda forma canonica

La *seconda forma canonica* di una funzione è il prodotto di somme:

* sono i maxtermini contenuti nell’Off-set di .
* Ogni maxtermine è realizzato con una porta *OR* a ingressi.
* Le uscite delle porte che rappresentano i maxtermini sono collegate tramite una porta *AND* a ingressi.

## Rappresentazione dei numeri

### Base binaria, ottale, esadecimale

I numeri possono essere rappresentati in:

* *Base 2*, ovvero base *binaria*. I numeri vengono rappresentati con le cifre *0*, *1*.
* *Base 8, ovvero ottale*. I numeri vengono rappresentati con le cifre che vanno da *0* a *7*.
* *Base 16*, ovvero *esadecimale*. I numeri vengono rappresentati con le cifre che vanno da *0* a *9*, poi susseguono:
  + *A* per 10
  + *B* per 11
  + *C* per 12
  + *D* per 13
  + *E* per 14
  + *F* per 15

### Rappresentazione binaria pura

Si definisce *rappresentazione binaria pura* la rappresentazione di un numero che è la stessa per qualsiasi base: .

### Rappresentazione di un numero N in base r

Per rappresentare un numero *N*, con , si fa riferimento alla seguente formula:

### Rappresentazione di un numero N in base r

Per rappresentare un numero *N* frazionario, quindi con , si fa riferimento alla seguente formula:

### Overflow

L’*overflow* si genera quando si tenta di rappresentare un numero non contenuto nel sottoinsieme considerato.

### Rappresentazione in segno e modulo

Per rappresentare un numero binario in *segno e modulo* su *n* bit, si considera:

* *un bit* per il segno (quello più significativo)
* *bit* per il modulo

L’*intervallo* di numeri rappresentabili è .

La *legge di codifica* è .

### Rappresentazione in complemento alla base

La rappresentazione in *complemento alla base* su *n* bit di associa:

* Ai *numeri positivi*, , il numero stesso
* Ai *numeri negativi*, , il numero

### Complemento diminuito

Data la rappresentazione binaria di un numero, il *complemento diminuito* si ottiene effettuando il complemento cifra per cifra. Esempio:

### Rappresentazione in eccesso-k

La rappresentazione in *eccesso-k* si effettua traslando tutti i numeri verso l’alto di una costante .

*Proprietà* di questa rappresentazione sono:

* La somma va corretta aggiungendo/sottraendo la costante .
* Moltiplicazioni e divisioni sono più complesse.

*Vantaggio* di questa rappresentazione è che mantiene la *relazione d’ordine* sulle rappresentazioni:

### Virgola fissa

I numeri in *virgola fissa* si rappresentano separando *parte intera* e *parte frazionaria* di un numero reale. La posizione della virgola è *fissa* e resta *sottintesa*.

Problema della virgola fissa è che c’è errore di *approssimazione* (*underflow*).

### Underflow

L’*underflow* si verifica quando si rappresenta con zero un numero vicino allo zero.

### Virgola mobile

Un numero reale può essere rappresentato dalla coppia , dove:

* viene definito come *mantissa*
* viene definito come *esponente*
* viene definito come *base numerica adottata*

Tale codifica prende il nome di *virgola mobile*.

Per ciascun numero esistono infinite coppie che lo rappresentano. Tra queste viene definito un *unico rappresentante*: si fissa la virgola subito dopo la prima cifra significativa.

In questa rappresentazione esistono due tipi di errori:

* *Assoluto*: grande su numeri grandi, piccolo su numeri piccoli
* *Relativo*: costante su tutto l’asse di rappresentazione.

Ci sono poi due tipo di rappresentazione:

* *rappresentazione esplicita*
* *rappresentazione implicita*

### Standard IEEE 754

Lo *standard IEEE 754* in *singola precisione* (*32 bit*) prescrive le rappresentazioni:

* *Mantissa*: 1 bit per segno, 23 bit per modulo
* *Esponente*: 8 bit, con un eccesso 127

Tale standard può essere usato anche a *doppia precisione*, quindi:

* IEEE 754 a *singola precisione* corrisponde ai numeri *float* (*32 bit*)
* IEEE 754 a *doppia precisione* corrisponde ai numeri *double* (*64 bit*)

## Rete combinatoria

Una *rete combinatoria* è un circuito logico formato da ed , ciascuno dei quali può assumere valori oppure .

Da un punto di vista logico, ogni uscita può essere definita come *funzione booleana* degli ingressi.

Ad ogni *istante* il valore delle uscite dipende dai valori in ingresso in quel *determinato istante*.

### Progettare una rete combinatoria

Per *progettare* una rete combinatoria:

1. Definizione *univoca* del problema.
2. *Analisi* del problema con *individuazione* di variabili in ingresso e funzioni di uscita.
3. Scrittura delle *tabelle di verità* di ogni funzione.
4. *Sintesi* e *semplificazione* delle funzioni con le *mappe di Karnaugh*.
5. Disegno dello *schema logico* della rete.

### Analisi di un sistema

L’*analisi* di un sistema consiste nell’individuare il *comportamento* del sistema, data la *rappresentazione schematica*.

Individua le relazioni *causa/effetto* tra i segnali *ingresso/uscita*.

### Sintesi di un sistema

La *sintesi* di un sistema consiste nell’individuare la struttura, data la *specifica funzionale*.

Individua *componenti elementari* e *interconnessioni* necessarie per realizzare la *specifica funzionale*.

### Circuito combinatorio

In un *circuito combinatorio* il valore delle *uscite* in un *determinato istante* dipende unicamente dal valore degli *ingressi* in quello *stesso istante*.

## Rete sequenziale

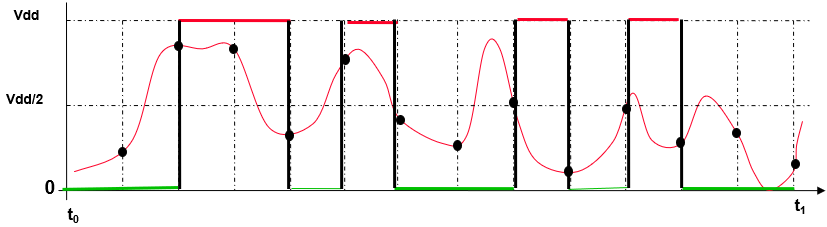
### Circuito sequenziale

In un *circuito sequenziale* il valore delle *uscite* in un *determinato istante* è funzione del valore degli *ingressi* in quel *determinato istante* e negli *istanti precedenti*. Da tale definizione susseguono

* ***Concetto di tempo***

Dato un intervallo di tempo vi sono *infiniti istanti*.

Dato un intervallo di tempo vi è un numero finito di *istanti discreti*.

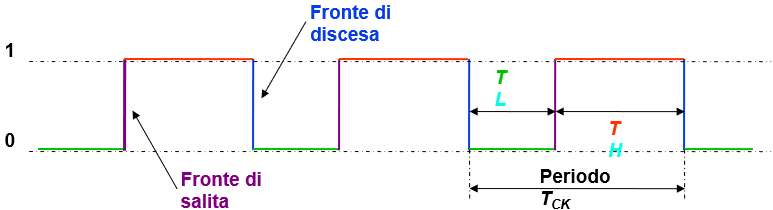


Il valore di viene *campionato* in istanti di tempo determinati.

Il segnale è rappresentato da *campioni*:

* + Assume valore se
  + Assume valore se .

Gli *istanti discreti* vengono scanditi attraverso un *segnale periodico* (*binario*), definito *clock*.



Durante un ciclo di tempo *TCK* il segnale assume:

* + Valore per un tempo *TH*
  + Valore per un tempo *TL*

Il *duty-cycle* è il rapporto

Il *fronte di salita* è il passaggio da valore a

Il *fronte di discesa* è il passaggio da valore a

* ***Concetto di stato***

Lo *stato* di un circuito sequenziale è determinato da un insieme di *variabili di stato*, e da una *funzione di transizione* che, in funzione dello stato presente e degli ingressi forniti, determina lo *stato prossimo*.

Lo stato di un circuito ad un dato istante dipende:

* + Dagli *ingressi* all’istante
  + Dallo *stato* al tempo

Per memorizzare lo stato di un circuito si utilizzano elementi di memoria detti bistabili, comunemente conosciuti come *Flip-Flop*.

### Automa a stati finiti

Una rete sequenziale con numero di *stati finiti* prende il nome di *automa a stati finiti*.

Una *macchina sequenziale* implementa un automa a stati finiti.

### Decoder binario

Un *decoder binario* ha tante uscite quante sono le combinazioni delle variabili in ingresso.

Esso attiva solamente l’uscita che corrisponde alla combinazione in ingresso.

È una *rete combinatoria* formata da e . Per ogni combinazione in ingresso, solo *un’uscita* assume , mentre le altre assumono .

### Macchina sequenziale

Una *macchina sequenziale* è definita dalla quintupla (), dove:

* : insieme finito dei simboli in ingresso
* : insieme finito dei simboli in uscita
* : insieme finito e non vuoto degli stati
* : funzione stato prossimo
* : funzione di uscita

La funzione stato prossimo associa ad ogni *stato presente* e per ogni *simbolo in ingresso*, uno *stato futuro*.

La funzione d’uscita genera un *simbolo d’uscita*:

* Nella *macchina di Mealy*, l’uscita dipende sia dallo stato che dall’ingresso ()
* Nella *macchina di Moore*, l’uscita dipende solo dallo stato ()

### Sintesi di una macchina sequenziale

Per *sintetizzare* una macchina sequenziale bisogna:

* Identificare e
* Sintetizzare la rete combinatoria che la realizza

Gli *elementi di memoria* sono costituiti da *Flip-Flop*. Quelli più usati sono quelli di tipo *D*.

dipende dal tipo di bistabile utilizzato.

non dipende dal tipo di bistabile utilizzato.

### Sintesi di macchine sequenziali sincrone

Per sintetizzare reti sequenziali *sincrone* bisogna:

1. Realizzare il diagramma degli stati.
2. Realizzare la tabella degli stati.
3. Minimizzare il numero degli stati.
4. Codificare gli stati interni

Per codificare gli stati interni si possono usare due configurazioni:

* + *Binario naturale*: in questa codifica il numero di bit è *minimo* e la configurazione di bit corrisponde al valore binario dello stato ().
  + *One-hot*: in questa codifica il *numero di bit* è uguale al *numero degli stati*. In ogni configurazione *un solo bit* assume valore , il resto ().

1. Realizzare la tabella delle transizioni.
2. Scegliere gli elementi di memoria

Dalla tabella delle transizioni è possibile notare la relazione tra la funzione di uscita e la funzione di stato prossimo. Come possibili scelte abbiamo:

* + Bistabile *SR*
  + Bistabile *JK*
  + Bistabile *D*
  + Bistabile *T*

1. Realizzare la tabella delle eccitazioni.
2. Sintetizzare la rete combinatoria che realizza la funzione stato prossimo e la funzione uscita.

### Tabella degli stati

Una macchina a stati finiti può essere descritta mediante la *tabella degli stati*, dove:

* Gli *indici di colonna* sono i *simboli in ingresso*: .
* Gli *indici di riga* sono i *simboli dello stato presente*: .

Gli *elementi della tabella* sono:

* *Coppia*  dove:
  + è uguale alla *funzione d’uscita*
  + è uguale alla *funzione stato prossimo* .

Ciò vale nella macchina di Mealy.

* Simbolo *stato prossimo* uguale alla funzione *stato prossimo* .

Ciò vale nella macchina di Moore.

### Diagramma degli stati

Il *diagramma degli stati* è un *grafo*  dove:

* corrisponde *all’insieme di nodi*, dove:
  + Ogni nodo rappresenta uno *stato*
  + Ogni nodo è associato con un *simbolo d’uscita*. Ciò vale solo per la macchina di Moore.
* corrisponde *all’insieme degli archi*, che rappresentano una *transizione di stato*.
* corrisponde *all’insieme di*:
  + *Ingressi* e *uscite*, per la macchina di Mealy.
  + *Ingressi*, per la macchina di Moore.

### Sistemi asincroni

Nei *sistemi asincroni* i segnali di uscita possono cambiare ogni qualvolta che cambiano uno o più ingressi.

### Sistemi sincroni

Nei *sistemi* *sincroni* *l’istante* in cui l’uscita può cambiare dipende da un *segnale di clock*.

Il *clock* forma *un’onda impulsiva periodica a frequenza costante*.

Un *impulso* è la *transizione* di una variabile logica da a o viceversa.

La *frequenza* (o *tempo di ciclo*) consiste *nell’intervallo di tempo tra due impulsi consecutivi*:

frequenza


## Sistemi digitali

I sistemi digitali si possono suddividere in due classi:

* *Sistemi combinatori*: il *valore delle uscite* al generico istante dipende solo dal *valore degli ingressi* applicati nello *stesso istante*.
* *Sistemi sequenziali*: il *valore delle uscite* al tempo dipende non solo dal *valore* *degli ingressi* applicati nello *stesso istante*, ma anche dalle *configurazioni di ingresso precedenti*.

Il termine *sequenziale* fa riferimento agli eventi in sequenza che hanno portato il sistema nella sua condizione attuale.

La *memoria* che il sistema conserva dagli elementi passati costituisce lo *stato* del sistema.

## Latch e Flip Flop

I circuiti capaci di memorizzare un singolo bit sono:

* *Latch*, ovvero un circuito capace di mantenere qualsiasi stato di uscita da un segnale momentaneamente in ingresso fino a quando non viene ripristinato da un nuovo segnale.
* *Flip Flop*, ovvero un circuito che ha due stati stabili e può essere utilizzato per memorizzare le informazioni sullo stato: è un multivibratore bistabile. È consentita una sola operazione per *ciclo di clock*: lo stato può cambiare solamente in corrispondenza di un *fronte del clock* (*edge-triggered*).

Ogni circuito di questo tipo:

* È *bistabile*: memorizza 0 oppure 1 in base all’ingresso che mantiene (*stabile*) in assenza di input;
* Ha *due output* (*Q* e ), che sono uno il complemento dell’altro.

### Bistabili

Un *bistabile* è un elemento di memoria sequenziale capace di mantenersi stabilmente fra due stati. Essi sono in grado di *memorizzare* l’informazione binaria relativamente ad un singolo evento, cioè ricordare se all’istante precedente l’ingresso rispettivo fosse oppure .

Le *differenze* principali tra bistabili sono:

* *Numero di ingressi*
* *Modo in cui gli ingressi ne determinano lo stato*

Possono essere *classificati* in:

* *Asincroni*: solo la *variazione* di un segnale *ad uno degli ingressi* dati può determinare l’*evoluzione* del *bistabile* imponendone il *cambiamento di stato*. Hanno solo *ingressi dati*.

A questa classe appartengono i Latch.

* *Sincroni*: le *variazioni* degli ingressi dati vengono *campionate* dal *segnale* iningresso del *sincronismo*, e solo quando tale segnale assume un particolare valore, il bistabile può *evolvere*.

A questa classe appartengono i Flip-Flop.

### Tabella delle transizioni

La *tabella delle transizioni* consente di descrivere il *comportamento* di un bistabile.

### Tabella delle eccitazioni

La *tabella delle eccitazioni* consente di conoscere la *configurazione* degli ingressi da applicare affinché possa avvenire una determinata *transizione* di stato.

Tale tabella può essere ricavata dalla tabella delle transizioni.

### Latch asincroni

Nei *latch asincroni* le *transizioni* di stato dipendono direttamente dalle *variabili in ingresso*.

### Latch sincroni

Nei *latch sincroni* le *transizioni* di stato dipendono da *segnali esterni di tipo impulsivi*, ovvero *segnali di clock*.

### Latch SR (Set-Reset)

Il *latch SR* è dotato di:

* *Due ingressi*, che corrispondono a S *set* e R *reset*. S setta l’uscita *Q* allo stato stabile 1 e lo memorizza. R resetta il circuito e porta *Q* a 0 (stato stabile) e lo memorizza.
* *Due uscite*, che corrispondono a *Q* e .

Tale latch viene realizzato con due porte NAND oppure due porte NOR.

### Latch JK

Il *latch JK* rappresenta una soluzione parziale al problema della configurazione del latch SR. Per bypassare questa configurazione, i valori di e si portano in AND con gli input del latch; dato una delle due AND avrà necessariamente l’output a 0, di conseguenza non si verificherà mai la combinazione .

### Latch D

Il *latch D*, derivato del latch SR, presenta un solo segnale d’ingresso, *D*, dove:

* .
* (*D* va inserito prima in un invertitore).

Avendo un solo ingresso sono possibili solo due casi:

Come output, il latch D ripete il segnale in ingresso.

### Latch trigger o toggle

Il *latch trigger* ha un solo ingresso *T*. Quando avviene il cambiamento di stato, altrimenti il latch resta nello stato corrente.

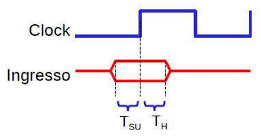
### Sincronia

Il *segnale di sincronismo* che, dato una determinata combinazione, consente al bistabile di *cambiare stato* prende il nome di *segnale di clock*.

I bistabili che hanno anche un segnale di sincronismo vengono definiti *bistabili sincroni*. Un esempio è il latch SR con l’aggiunta di *sincronismo*.

In un segnale di clock vi sono:

* Tempo di *set-up* (): minimo intervallo che precede il fronte di clock durante il quale il segnale deve essere mantenuto stabile.
* Tempo di *hold* (): minimo intervallo che segue il fronte di clock durante il quale il segnale deve essere mantenuto stabile.



### Bistabile JKT

Il *bistabile JKT* risolve la condizione presente nel latch SR nel quale gli ingressi hanno entrambi valore :

* Con *sincronismo non attivo*, il bistabile *mantiene il proprio stato*
* Con *sincronismo attivo*:
  + Se gli ingressi () sono *diversi* o *uguali* *a ,* il bistabile si comporta come il latch SR ().
  + Se gli ingressi () sono *uguali a* , lo stato del bistabile *commuta*.

### Bistabile DT

Il *bistabile DT* si ottiene dal bistabile , ponendo e .

Nel bistabile :

* Con *sincronismo* *attivo*, all’*uscita* del bistabile si trasferisce il *valore di*  presente in ingresso.
* Con *sincronismo non attivo*, il *valore dell’uscita non cambia*.

### Bistabili Master-Slave

I *bistabili Master-Slave* sono implementati per garantire un buon funzionamento di una rete sequenziale. Esempio è il *registro a scorrimento* basato su bistabili DT.

### Flip-Flop Master-Slave

Nel *Flip-Flop Master-Slave* si collegano due latch in serie controllate da un apposito *segnale di clock*. Il *latch Master* riceve gli ingressi, mentre il *latch Slave* produce le uscite.

## Macchine per il trattamento di codici

### Codice

Un *codice* è un modo per mantenere l’informazione.

### Informazione

L’*informazione* è costituita da:

* *Attributo*: ciò che rappresenta l’informazione
* *Tipo*: come l’informazione viene rappresentata
* *Valore*: l’elemento assunto dall’informazione

Da ciò, possiamo dire che l’informazione è definita come l’*attributo che assume il valore di un certo tipo*. Tale informazione viene definita *completa*.

### Tipi

Esistono due *tipi*:

* *Semplici*: i dati sono *entità atomiche* (esempio: carattere)
* *Strutturati*: i dati sono composti da più tipi semplici (esempio: stringa)

### Cardinalità

La *cardinalità* dell’informazione indica il *numero di valori* che può assumere un attributo.

### Alfabeto

Un *alfabeto* è un insieme finito *non vuoto* di simboli.

### Stringa

Dato un alfabeto, una *stringa* è definita come una *sequenza di caratteri* appartenenti all’alfabeto considerato.

### Lunghezza di una stringa

Data una stringa, il *numero di caratteri* che la compongono è definita *lunghezza* di una stringa. Si indica con .

Una stringa di *lunghezza*  viene definita *vuota* o *nulla*.

### Codifica

Si definisce *codifica* quella funzione che permette di rappresentare una stringa descritta attraverso l’*alfabeto origine* in una stringa descritta mediante l’*alfabeto di destinazione*.

Esempio: supponendo di avere un alfabeto origine e un alfabeto di tipo , la codifica di *mediante* è una codifica detta *codice*, che *trasforma* ogni elemento in una stringa di *lunghezza* di elementi detta *parola codice*.

### Codice a lunghezza fissa

Se si pone la lunghezza per tutti gli elementi di , si ottiene una *codifica a lunghezza fissa*. In tal caso a ciascun elemento corrisponde una delle disposizioni con simboli dei simboli di tipo sugli posti della stringa e quindi:

Da cui:

### Codice incompleto

Se ma non è potenza di , il codice è definito *incompleto* e la fornisce il *numero di parole codice* non associate ad alcun elemento dell’alfabeto origine.

### Codice ridondante

Il *codice ridondante* si ottiene per adoperando più caratteri dell’alfabeto in codice di quanti *strettamente necessari*.

### Codice a lunghezza variabile

Nel *codice a lunghezza variabile* la lunghezza del codice è funzione di ciascun elemento :

*Proprietà* di questa tipologia di codice è:

* La/e parola/e più corta/e non si ritrova/ritrovano come sequenza iniziale delle parole più lunghe: è la lunghezza stessa che deve essere riconosciuta nel contesto della parola codice.

*Vantaggi* di questo codice sono:

* *Risparmio* di *spazio* nella memorizzazione
* *Risparmio* di *tempo* nella trasmissione

### Rappresentazione decodificata

La *rappresentazione decodificata* consiste in codifica mediante una tabella in cui la *lunghezza del codice* è pari alla *cardinalità dell’insieme da codificare*:

### Decoder (Decodificatore)

Un *decoder* (o *decodificatore*) è una macchina che in *ingresso* riceve una *parola codice* e in *uscita* presenta la sua *rappresentazione decodificata*.

## Bus

Un bus è un canale di comunicazione che permette a periferiche e componenti di un sistema elettronico di interfacciarsi tra loro scambiandosi informazioni o dati di sistema attraverso la trasmissione e la ricezione di segnali.

I dati possono essere trasferiti con due soluzioni:

* *Bus dedicato*: esiste una linea per ciascuno dei trasferimenti possibili; è una soluzione efficiente, ma costosa.
* *Bus comune*: esiste un’unica via in cui sono raccolti più trasferimenti potenziali. Viene scelto, di volta in volta, il bus attivo e i trasferimenti avvengono in due tempi distinti; è una soluzione diffusa ed economica, ma meno efficiente rispetto al bus dedicato in quanto non permette trasferimenti simultanei.

Esistono tuttavia soluzioni intermedie:

* *Trasferimenti paralleli (n)*
* *Trasferimenti seriali (1)*
* *Trasferimenti misti (seriali su byte)*

### Trasferimento dati da bus a registro

Il *trasferimento dati da bus a registro* consiste nel caricare un registro; tale caricamento avviene attraverso un segnale di *stobe*: il dato viene caricato nel registro quando è attivo il segnale di caricamento, tipicamente sincrono con il segnale di clock.

### Trasferimento dati da registro a bus

Nel trasferimento dati da registro a bus il dato viene immesso nel bus, prelevandolo da un registro.

### AND Tristate

L’*AND Tristate (3-state)* è la tecnica più diffusa per collegare più registri sorgenti verso un bus comune. La tabella dell’AND 3-state è:

|  |  |  |
| --- | --- | --- |
| a | x | y |
| 0 | 1 | z |
| 1 | 1 | 1 |
| 0 | 0 | z |
| 1 | 0 | 0 |
| 0 | z | z |
| 1 | z | z |

È possibile realizzare un AND 3-state di parola tra bus e un segnale di abilitazione *a*. Così, il bus *y* riproduce *x* se *a* è attivo, altrimenti *y* assume valore *neutro*.

Si afferma che “*y* è sostenuta da *buffer 3-state*”: .

### OR di bus

Per collegare più registri in uscita si collegano in una *OR di bus*. Tale OR si ottiene con un collegamento delle sorgenti, ciascuna sostenuta da buffer 3-state. Si deve verificare che nelle *n* linee che confluiscono nel bus, al più una alla volta si attiva: si ha un *segnale attivo* e i rimanenti neutri.

### Registri a scorrimento

I *registri a scorrimento* sono costituiti da una catena di celle di memoria a *1 bit* interconnesse tra loro.

Ad ogni impulso di *clock* essi consentono lo *scorrimento* dei bit da una cella a quella adiacente; lo scorrimento avviene in un’unica direzione *predeterminata*, o in una direzione *variabile* comandata da *registri bidirezionali*.

## Memoria

La *memoria* è un elemento di un computer o di un suo sottosistema deputato alla persistenza dei dati e/o delle istruzioni dei programmi. Esistono due tipi di memorie:

* *Volatili*, come registri, memoria cache, memoria principale.
* *Persistenti*, come dischi magnetici, memoria a stato solido (SSD). Questo tipo di memoria si divide a sua volta in:
  + *online*, dove i dati sono sempre accessibili
  + *offline*, dove i dati sono accessibili solo se il supporto è montato, come nastri e dischi ottici.

La memoria è caratterizzata da:

* *Dimensione*, indicata dal numero di celle. È di centinaia di MB oppure GB.
* *Tempo di accesso*, ovvero il ritardo che intercorre dopo l’invio del comando di lettura. È in ordine di nanosecondi.

La memoria è, quindi, una componente fondamentale di un calcolatore elettronico. È un dispositivo capace di mantenere lo *stato dei dati*, memorizzati in forma di parole di *bit*.

*L’accesso alla memoria* avviene attraverso la *selezione* (bus-indirizzi), la *lettura/scrittura* (bus-dati) ed il *controllo* (bus-controlli).

Essa è gestita dalla CPU.

Se il bus è condiviso da più unità e queste non possono accedere direttamente in memoria, è necessario un *DMA*.

### DMA (Direct Memory Access)

Il *DMA (Direct Memory Access)* è quel meccanismo che permette ai sottosistemi di un computer di accedere direttamente alla memoria interna per lo scambio di dati, senza coinvolgere la CPU.

### Memoria cache

La *memoria cache* è una memoria prossimale, che memorizza dati e programmi.

È interposta tra RAM e CPU ed ha il fine di ridurre gli intervalli di attesa della CPU. Essa è una memoria veloce.

Consiste in una copia delle istruzioni principali presenti in RAM, in modo da non dover costantemente accede al disco fisso per prelevare istruzioni di programma.

### Livelli della cache

La *memoria cache* si diversifica in livelli:

* *Primo livello* (32-64 KB): garantisce un flusso continuo all’interno del processore;
* *Secondo livello* (28 KB-2 MB): limita il degrado delle prestazioni dovute alla lentezza della memoria centrale. La cache di secondo livello viene definita anche SRAM (Static RAM);
* *Terzo livello* (8-32 MB): condivisa da più core della stessa CPU.

### Storage (memoria persistenti)

Le memorie persistenti permettono di, dopo aver spento il computer, continuare a memorizzare dati.

### Memorie volatili

A differenza dello storage, le memorie volatili (esempio: RAM) non permettono la memorizzazione di dati in assenza di corrente elettrica.

### SRAM (Static Random Access Memory)

La SRAM è costituita da Flip-Flop di tipo D. Le celle sono disposte a matrice e l’acceso avviene specificando le colonne. Vengono usate per la cache.

### DRAM (Dynamic Random Access Memory)

Nella DRAM lo stato è mantenuto da un transistor e da un condensatore che si carica periodicamente. La DRAM è una memoria asincrona usata per la memoria centrale.

### SDRAM (Synchronous Dynamic Random Access Memory)

Nelle SDRAM l’accesso è regolato da un clock. Vengono usate per CPU e GPU.

### FeRAM (Ferroelectic Dynamic Random Access Memory)

Le FeRAM mantengono lo stato anche senza alimentazione.

### Dischi magnetici

Le azioni possibili sui dischi magnetici sono:

* *Formattazione:* consiste nella suddivisione della superficie di un disco in tracce e settori.
* *Blocco:* consiste in una minima unità indirizzabile in un disco magnetico e il suo indirizzo è dato da una coppia di numeri che rappresentano il numero della traccia ed il numero del settore.
* *Accesso diretto:* consiste nel posizionare la testina su un qualunque blocco senza passare sui blocchi precedenti.

Per effettuare la lettura/scrittura è necessario spostare la testina in modo da posizionarla all’indirizzo desiderato; in questo caso la testina è ferma mentre il disco si muove.

Si considera anche il tempo di accesso:

* Tempo di posizionamento (*seek time*): tempo per spostare la testina in senso radiale fino a raggiungere la traccia interessata.
* Tempo di latenza: tempo che bisogna attendere perché il settore desiderato passi sotto la testina
* Tempo di trasmissione: tempo perché il blocco da leggere/scrivere transiti sotto la testina.

### Hard Disk

Gli hard disk sono utilizzati come supporto di memoria secondaria fissa all’interno di un elaboratore; il tempo di accesso è nell’ordine di millisecondi ().

Caratteristiche degli hard disk sono:

* *Non volatilità*: i dati persistono anche senza alimentazione elettrica.
* *Grande capacità*: le memorie secondarie hanno capacità maggiore (anche di diversi ordini di grandezza) rispetto a quelle centrali
* *Bassi costi*: il costo per bit di una memoria secondaria è minore (di diversi ordini di grandezza) rispetto alla memoria centrale
* *Minore velocità di accesso*: I tempi di accesso a una memoria secondaria sono maggiori (di qualche ordine di grandezza) rispetto alla memoria principale

## Scheda video

Una scheda video ha la funzione di inviare al monitor le informazioni da visualizzare; è dotata di RAM definita SGRAM; ha una propria capacità elaborativa. È dotata anche di convertitore da digitale ad analogico (DAC Digital to Analogue Converter).

### Compito della scheda video

Il compito della scheda video, quindi, è tradurre la rappresentazione dell’immagine prodotta dal processore in un formato adatto al monitor.

Ogni volta che il video deve essere ridisegnato la scheda video legge la rappresentazione digitale dell’immagine dalla memoria e invia al monitor il segnale che permette di rappresentare il colore di ogni singolo pixel.

### Tubo a raggio catodico

Il *catodo* è un piccolo elemento metallico riscaldato all'incandescenza che emette elettroni per effetto termoionico.

All'interno del *tubo catodico* gli elettroni vengono diretti in un fascio per mezzo di un'elevata differenza di potenziale elettrico tra catodo e anodo.

Con l'aiuto di altri campi elettromagnetici opportunamente disposti il fascio è focalizzato opportunamente.

Il *raggio* (detto anche *pennello elettronico*) viene deflesso dall'azione di:

* *Campi magnetici*, Forza di Lorentz (deflessione magnetica).
* *Campi elettrici* (deflessione elettrostatica).

Colpisce un punto qualunque sulla superficie interna dello schermo, l'anodo.

### Monitor CRT

I monitor CRT hanno input analogico e il ruolo della scheda video è quello di convertire la rappresentazione digitale dell’immagine da visualizzare in segnali analogici. La traduzione del segnale è realizzata da un componente denominato RAMDAC (Random Access Memory Digital-Analog Converter).

### Monitor LCD

I monitor LCD hanno input digitali e i ruoli della scheda video sono:

* Adattare l’immagine adeguando colori e risoluzione
* Mettere a disposizione una memoria locale da utilizzare come buffer
* Supportare l’accelerazione grafica

Il *funzionamento* dei monitor LCD: utilizzando un campo elettrico per modificare l’orientamento delle molecole si variano le proprietà ottiche dei cristalli e quindi l’angolo di polarizzazione della luce che li attraversa.

### VGA (Video Graphics Array)

Il VGA è lo standard analogico progettato per monitor a tubo catodico e utilizzato anche da monitor LCD. Tuttavia, ha problemi di rumore elettrico, distorsione dell’immagine e alcuni errori nella valutazione dei pixel

### DVI (Digital Visual Interface)

Il DVI è stato introdotto nei monitor LCD. Risolve i problemi del VGA e trasporta anche segnali analogici VGA.

### HDMI (High-Definition Multimedia Interface)

L’HDMI è uno standard pubblicato nel 2003 ed ha come obiettivo la sostituzione degli standard precedenti; può trasportare anche dati audio.

### DP (DisplayPort)

Il DP è stato pubblicato nel 2006 ed è uno standard utilizzato per collegare una sorgente video digitale ad un monitor; può trasportare dati audio ed altri tipi di dati. (Esempio: ethernet).

### Colori

La profondità del colore dei pixel indica il numero di bit utilizzati per codificare il colore di ogni singolo pixel.

Le configurazioni tipiche sono:

|  |  |  |  |
| --- | --- | --- | --- |
| *Profondità* | *Numero colori* | *Numero byte* | *Nome* |
| 4 bit | 16 | 0.5 | Standard VGA |
| 8 bit | 256 | 1.0 | 256 Color Mode |
| 16 bit | 65.536 | 2.0 | High Color |
| 24 bit | 16.777.216 | 3.0 | True color |

Da precisare: in modalità true color ogni colore viene ottenuto con 256 tonalità diverse dei 3 colori primari adattivi (RGB Red, Green, Blue).

### Memoria video

La *memoria video* è la memoria residente sulla scheda grafica, che aumenta continuamente le proprie dimensioni.

### AGP (Accelerated Graphic Port)

Il bus AGP (Accelerated Graphic Port) ha la funzione di ampliare la banda di trasmissione del bus, oppure di inserire bus o porte dedicate. Connette la scheda video direttamente alla memoria centrale della macchina, e tale soluzione consente di ridurre il traffico sul bus e di mettere a disposizione del coprocessore grafico una grande quantità di memoria

### PCIe (PCI Express)

Il bus PCIe (PCI Express) è l’evoluzione del bus PCI e ha sostituito il bus AGP come interfaccia in quanto offre una larghezza di banda maggiore e maggiore potenza erogata.

### SLI e CrossfireX

Le tecnologie dei bus SLI (NVIDIA) e CrossfireX (AMD) consentono di collegare e utilizzare più schede grafiche contemporaneamente.

### GPU (Graphical Processing Unit)

La GPU (Graphic Processing Unit) è il processore dedicato delle schede video e ha lo scopo di limitare l’uso della CPU.

Alcune librerie apposite sono OpenGL e Direct3D.

### Device driver

Il device driver adatta il software eseguito sulla CPU a utilizzare le primitive grafiche messe a disposizione dalla GPU.

### GPGPU (General Purpose Graphical Processing Unit)

Le GPGPU (General Purpose Graphical Processing Unit) permettono l’uso della GPU per applicazioni di tipo generico, non necessariamente grafico.

## Input/Output

Il sottosistema di input/output permette al processore di interagire con periferiche di input e di output.

### Periferiche di input

Come periferiche di input abbiamo: mouse, tastiera, ecc…

### Periferiche di output

Come periferiche di output abbiamo: schermo, stampante plotter, ecc…

### Periferiche di input/output

Le periferiche di input/output permettono di trasmettere e ricevere dati. Esempi sono: memorie di massa, schede di rete, porte seriali, ecc…

### Modelli di collegamento

Esistono due modelli possibili di collegamento:

* A bus unico, dove su un unico bus sono collegati memoria e sistema I/O (*Memory Mapped I/O*)
* A bus distinti, dove esistono due bus fisici diversi: uno per la memoria e un per il sistema I/O (*Isolated I/O*)

### Bus di collegamento

Il bus di collegamento tra CPU, memoria e I/O si suddivide in:

* Bus-dati, utilizzato per il trasferimento dati
* Bus-indirizzi, utilizzato per il trasferimento indirizzi
* Bus-controlli, utilizzato per il controllo delle attività di sistema

### Tastiera

La tastiera è il principale strumento di input di un computer. Presenta i seguenti tasti:

* Alfanumerici
* Speciali
* Frecce direzionali
* Funzione

Video e tastiera non sono direttamente collegati tra loro, bensì spetta al processore di riprodurre sul video le informazioni fornite tramite tastiera.

La tastiera è gestita da:

* Encoder: trasforma una combinazione riga-colonna in una codice di n-bit.
* Shift register: trasforma gli n-bit in una sequenza di n-bit da gestire in modo seriale.
* Interfaccia seriale: invia il codice a n-bit al bus di sistema in maniera opportuna.

Le tastiere attuali hanno come standard il PS2 o l’USB; la trasmissione può avvenire anche in modalità wireless.

## Periferiche

Una periferica è un dispositivo utilizzato dall’utente per l’interazione tra utente e sistema.

Le periferiche hanno un limitata autonomia, sono controllate dal processore e operano in modo asincrono.

Le periferiche odierne non seguono questa definizione in quanto sono realizzate con sufficiente intelligenza artificiale da permettere loro di essere autonome.

### Controller

Per sincronizzare le periferiche con il processore vengono inseriti i controller e sono in grado di gestire le comunicazioni tra dispositivi I/O e tutte le altre componenti della macchina, processore compreso; inoltre sono in grado di indicare quando un processo termina la sua esecuzione attraverso un segnale di interrupt.

### Terminale

Un terminale è il più comune strumento di interazione uomo/macchina; è costituito da tastiera e video

### Bus

Un bus è un canale di comunicazione condiviso da più unità. È caratterizzato da:

* Larghezza, numero di bit del bus dati
* Velocità, frequenza di lavoro del bus e il numero di byte per ciclo di clock
* Transfer rate, MB/sec trasmessi.

I *vantaggi* di usare i bus sono:

* Possono essere aggiunti facilmente nuovi dispositivi
* Le periferiche possono essere spostate da un computer ad un altro purché utilizzino lo stesso tipo di bus
* Costi contenuti

Gli *svantaggi*, invece, sono:

* La banda del bus limita le velocità di trasferimento
* La velocità massima è condizionata da:
  + Lunghezza del bus
  + Numero di dispositivi collegati sul bus

Tra i bus più diffusi abbiamo:

* USB (Universal Serial Bus)
* EISA (Extended ISA)
* AGP (Accelerated Graphic Port)
* IDE (Integrated Device Electronics)
* ISA (Industry Standard Architecture)
* VESA (Video Electronics Standards Association)
* PCI (Peripheral Component Interconnect)
* SCSI (Small Computer Standard Interface)
* PCMCIA (Personal Computer Memory Card International Association).

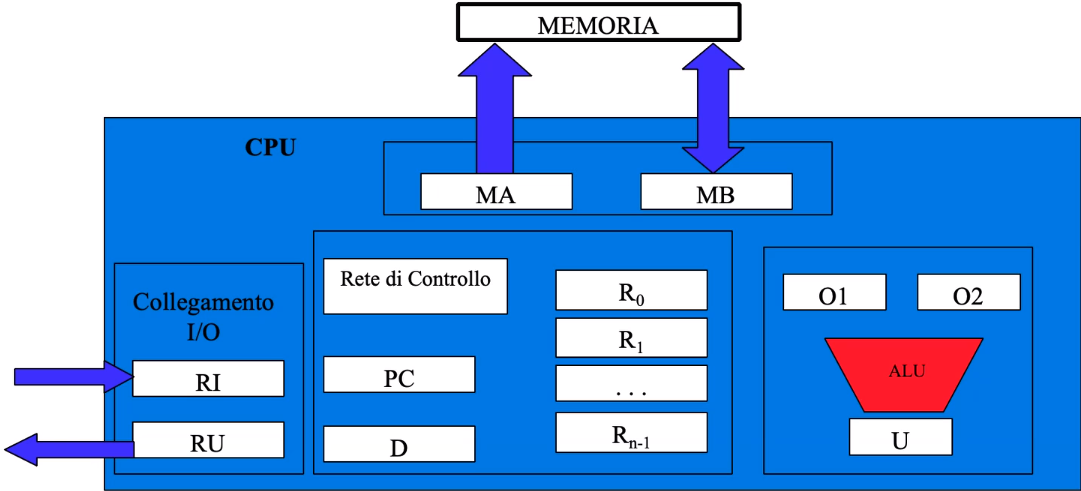
## Processore

Il processore è in grado di eseguire solo un set di istruzioni più o meno complesse. Tali istruzioni sono rappresentate attraverso il linguaggio macchina, che consiste in stringhe di bit che codificano in binario le istruzioni che un processore è in grado di eseguire.

Il processore è formato da:

* *UC*, unità di controllo: costituito da:
  + *Program Counter (PC):* indica la prossima istruzione.
  + *Instruction Register:* registro di decodifica (*D)*.
  + *Registri di macchina (Ri).*
* *ALU*, unità aritmetico-logica, che fondamentalmente esegue operazioni di addizione, moltiplicazione, confronti e salti.
* *Registri*, quelli che costituiscono il collegamento alla memoria sono:
  + *Memory Address (MA)*:registro degli indirizzi di memoria.
  + *Memory Buffer (MB):* registro di transito della memoria.
* *Buffer I/O:* sezione di collegamento con I/O.

### Modello di riferimento



### Istruzioni

Le istruzioni indicano:

* Trasferimento di informazioni.
* Operazioni aritmetico-logiche (addizione, moltiplicazione, divisione, sottrazione, confronti, salti).

*Da ricordare*: dati e istruzioni di un programma sono memorizzati nella memoria centrale.

### Registri

I registri sono costituiti da *n* Flip-Flop di tipo *D*.

Possono essere:

* *Registri interni:* non visibili al programmatore e non necessari al funzionamento del processore.
* *Registri di macchina:* definiscono il *modello di programmazione*, classificabili in:
  + *Speciali:* utilizzati per specifiche funzioni non accessibili direttamente e sono impiegati dal sistema operativo per la gestione del sistema, e sono:
    - Registri che sovraintendonoil *memory boundaries.*
    - Registri di ausilio all’*indirizzamento*.
    - Registri delle *interruzioni*.
  + *Generici:* indicati con R1, R2, R3, ecc… Essi sono organizzati in banchi e svolgono due o più funzioni e sono:
    - *Accumulatore.*
    - *Indirizzamento.*
    - *Pagina 0:*  corrisponde ai primi 256 byte di memoria e sono tutti utilizzati per le *funzioni speciali*.

Tali registri possono essere organizzati come:

* + *Memoria indirizzabile.*
  + *Registri autonomi:* ogni registro ha un nome proprio.

La *dimensione* dei registri dipende dal tipo di informazione che devono contenere:

* La dimensione di MA e PC (Program Counter) dipende dalla dimensione degli indirizzi.
* La dimensione dei registri dell’ALU dipende dalla dimensione dei dati da elaborare.

### Stack

Lo *Stack* corrisponde ad una struttura di dati astratta che segue lo scheduling *LIFO* (Last In First Out). Le operazioni sullo stack prevedono il controllo di:

* *Stack overflow*, in fase di push.
* *Stack empty,* in fase di pop.

### Stack Pointer (SP)

Lo *Stack Pointer (SP)* punta all’ultima locazione occupata.

### Processore a voce

In un *processore a voce* tutte le parole-macchina sono della stessa lunghezza; tale processore è dimensionato sull’informazione più lunga e ha problemi con i dati di natura intrinsecamente variabile.

### Processore a byte

In un *processore a byte* la lunghezza delle parole-macchina è variabile; viene utilizzato per il trattamento stringhe.

### Stati del processore

Il processore ha due stati:

* *Supervisore:* quando esegue il sistema operativo può eseguire tutte le istruzioni dell’Instruction Set.
* *Utente:* quando esegue un programma utente le istruzioni privilegiate non sono eseguibili.

### Ciclo di un processore

Il ciclo di un processore si divide in fasi: *bootstrap, fetch, operand assembly, execute*. In realtà se il ciclo fosse realmente soltanto questo, avremmo diversi *svantaggi*, come ad esempio:

* non ci sarebbe modo di rimuovere forzatamente un'applicazione che entra per errore in un *ciclo infinito*;
* il sistema operativo avrebbe un controllo limitato sul sistema;
* l'interazione col mondo esterno attraverso le unità di I/O sarebbe poco efficiente.

Il ciclo di un processore, quindi, è così definito:

*inizializzazione, instruction fetch, decoding, operand assembly, execute, interrupt check, instruction fetch*.

### Interruzioni

Al termine di ciascun ciclo il processore verifica l'occorrenza di eventuali *segnali di interruzione*. In presenza di tali eventi si determina il passaggio allo stato *supervisore*, altrimenti in assenza di tali eventi l’elaborazione ritorna alla fase *fetch*. Delle interruzioni se ne occupa *l'ISR* (*Interrupt Service Routine*), che gestisce questi eventi attivando l'esecuzione della corrispondente funzione di servizio. Possibili cause delle interruzioni possono essere:

* *Interruzioni periodiche*, per computare il tempo speso da un'applicazione;
* *Interruzione di I/O tra periferiche*;
* *Interruzioni per errori software*;
* *Interruzioni per guasti*;
* *Interruzioni programmate*.

Al momento di una *Interrupt Request* (*IRQ*) il sistema deve:

* *Salvare il contesto* (*hardware*);
* Identificare la *causa* di interruzione;
* *Saltare all'ISR*;
* *Salvare il contesto* (*software*);
* *Servire l'interruzione*.

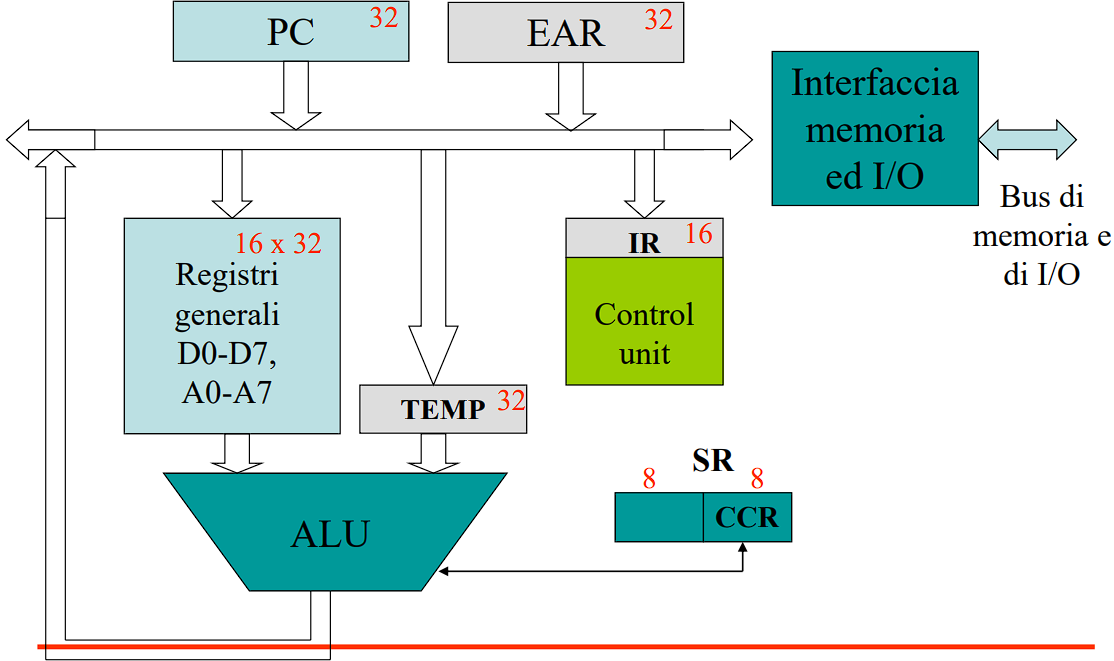
Se è previsto, successivamente, bisogna *ripristinare* il contesto sia software che hardware, ed infine riprendere l'esecuzione del programma sospeso eseguendo l'istruzione successiva.

### Latenza di un’interruzione

La *latenza di un’interruzione* è il ritardo che si viene a creare tra *l'istante in cui il segnale di interruzione diviene alto* e *l'istante in cui inizia l'esecuzione della routine di interrupt*. Per gestire l'annidamento delle interruzioni, si stabilisce un *livello di priorità*. Durante l'esecuzione di una ISR, vengono accettate le richieste di *interrupt* provenienti solo da dispositivi a *priorità più alta* rispetto a quella corrente.

# Assembly

### Architettura del processore MC68000



### Istruzioni assembly

* *JMP*: con questa istruzione si effettuano salti incondizionati, ovvero salta indipendentemente dal verificarsi o meno di una determinata condizione.
* *CLR*: con questa istruzione si inizializza una variabile/un registro.
* *BNE*: con questa istruzione si effettuano salti condizionati, ovvero per poter saltare bisogna che si verifichi una determinata condizione.
* *MOVE*: con questa istruzione si effettua l’operazione di assegnazione. Esempio: move G1, G2 equivale a .
* *MOVEQ (quickly)*: questa istruzione effettua la stessa operazione della move, solo che l’operando immediato è su 8 bit ed il registro di destinazione è un registro dati.
* *LEA*: questa istruzione consente di caricare un indirizzo in un registro indirizzi.
* *ADD*: questa istruzione consente di effettuare l’operazione di somma.
* *MUL*: questa istruzione consente di effettuare l’operazione di moltiplicazione.
* *DC*: questa istruzione consente di assegnare un valore ad una variabile.
* *DS*: queta istruzione consente di definire la dimensione di una variabile.

### Indirizzamenti

In assembly, gli indirizzamenti diretti possibili sono:

* Con registro dati (Esempio: D3)
* Immediato (Esempio: #1)
* Con registro indirizzi (Esempio: A1)
* Assoluto (Esempio: $16)

Mentre, gli indirizzamenti indiretti possibili sono:

* Con registro indirizzi (Esempio: A5)
* Con registro indirizzi e post-incremento (Esempio: (A5)+)
* Con registro indirizzi e pre-decremento: prima si esegue il decremento, poi si esegue l’istruzione successiva. (Esempio: -(A5))
* Con registro indirizzi e spiazzamento di 16 bit: si può indicare a quale elemento ci si vuole posizionare in un array. (Esempio: $f54(A2))
* Con registro indirizzi, indice e spiazzamento: ci si posiziona in una parte precisa della memoria e con l’indice ci si muove all’interno dell’array. (Esempio: $30(A5,A6))
* Relativo a PC (Program Counter) (Esempio: $68°9(PC))
* Relativo a PC (Program Counter) con indice e spiazzamento (Esempio: $2E(PC,A4))

### Istruzione MUL

L’istruzione MUL consente di eseguire la moltiplicazione.

### Istruzione BTST

L’istruzione BTST posizione il flag Z in funzione del valore del bit indicato nella chiamata dell’istruzione, dell’operando indicato

Esempio:

move #5, D0 00000101

btst #0,D0 Z = 0

btst #1,D0 Z = 1

### Istruzione BCLR

L’istruzione BCLR azzera il bit su cui si esegue il test.

### Istruzione BSET

L’istruzione BSET pone a 1 il bit su cui si esegue il test.

### Istruzione BCHG

L’istruzione BCHG complementa il bit su cui si esegue il test.

### Sottoprogramma

Un sottoprogramma è una porzione di codice che viene eseguita mediante una chiamata. Tale chiamata viene fatta utilizzando l’indirizzo della prima istruzione del sottoprogramma, oppure indicando il nome del sottoprogramma.

La chiamata al sottoprogramma avviene mediante la keyword JSR e la sintassi è:

JSR <indirizzo sottoprogramma>

La JSR è simile ad un salto non condizionato: salta all’indirizzo indicato come operando, ma differisce perché stabilisce il punto di ritorno della funzione. Infatti, la JSR salva in cima allo stack il valore del PC (Program Counter).

Con questa operazione, lo Stack Pointer (SP) viene decrementato di per contenere l’indirizzo salvato.

Mentre, per indicare il ritorno di un sottoprogramma viene usata la keyword RTS.

### Passaggio dei parametri

Per poter lavorare sui dati un sottoprogramma necessita che tali dati vengano passati come parametri; il passaggio dei parametri può avvenire in diverse modalità:

* *Mediante i registri*

Tale modalità di passaggio è veloce e non serve accedere in memoria per avere i parametri; tuttavia, bisogna tener conto dei registri utilizzati e dei registri che cambia il sottoprogramma.

Per poter salvare un insieme di registri si utilizza questa istruzione:

movem A0/D2-D4/D6,Address \*salvataggio dati

movem Address,A0/D2-D4/D6 \*recupero dati

In particolare, se si vuole usare lo stack si può effettuare questa chiamata:

movem A0/D2-D4/D6,-(SP) \*decrementa lo Stack Pointer – salvataggio dati

movem (SP)+,A0/D2-D4/D6 \*incrementa lo Stack Pointer – recupero dati

Nella “movem” è necessario che la lista sia ordinata allo stesso sia nel salvataggio e sia nel recupero dati.

* Mediante aree dati in memoria;
* Mediante lo stack di sistema.