

# Università di Trieste

## Laurea in ingegneria elettronica e informatica

Enrico Piccin - Corso di Fondamenti di informatica - Prof. Francesco Fabris

Anno Accademico 2021/2022 - 27 Settembre 2021

### Indice

|          |                                                          |          |
|----------|----------------------------------------------------------|----------|
| <b>1</b> | <b>Analogico e digitale</b>                              | <b>4</b> |
| 1.1      | Analogico . . . . .                                      | 4        |
| 1.2      | Digitale . . . . .                                       | 4        |
| 1.3      | Teorema del campionamento . . . . .                      | 4        |
| 1.4      | Rumore . . . . .                                         | 4        |
| 1.4.1    | Rumore nei segnali digitali . . . . .                    | 5        |
| 1.5      | L'importanza dell'alfabeto binario . . . . .             | 5        |
| 1.6      | Vantaggi sistemi digitali . . . . .                      | 6        |
| 1.7      | Riassunto . . . . .                                      | 8        |
| <b>2</b> | <b>Algebra booleana</b>                                  | <b>9</b> |
| 2.1      | Costanti booleane . . . . .                              | 9        |
| 2.2      | Calcolo funzionale di verità . . . . .                   | 9        |
| 2.3      | Tavole di verità . . . . .                               | 9        |
| 2.4      | Connettivi binari . . . . .                              | 10       |
| 2.5      | Insiemi minimi di connettivi . . . . .                   | 11       |
| 2.6      | Teorema di De Morgan . . . . .                           | 12       |
| 2.7      | Riassunto . . . . .                                      | 14       |
| 2.8      | Insieme minimo . . . . .                                 | 14       |
| 2.9      | Impostazione assiomatica dell'algebra booleana . . . . . | 14       |
| 2.9.1    | Assioma 0 (A0) - Leggi di composizione . . . . .         | 15       |
| 2.9.2    | Assioma 1 (A1) - Esistenza . . . . .                     | 15       |
| 2.9.3    | Assioma 2 (A2) - Chiusura . . . . .                      | 15       |
| 2.9.4    | Assioma 3 (A3) - Elemento neutro . . . . .               | 15       |
| 2.9.5    | Assioma 4 (A4) - Commutatività . . . . .                 | 15       |
| 2.9.6    | Assioma 5 (A5) - Associatività . . . . .                 | 16       |
| 2.9.7    | Assioma 6 (A6) - Distributività . . . . .                | 16       |
| 2.9.8    | Assioma 7 (A7) - Complementarietà . . . . .              | 16       |
| 2.10     | Teoremi principali dell'Algebra Booleana . . . . .       | 17       |
| 2.11     | Logica duale . . . . .                                   | 18       |
| 2.12     | Variabili, funzioni Booleane e porte logiche . . . . .   | 19       |
| 2.12.1   | Funzioni a una variabile (unarie) . . . . .              | 20       |
| 2.12.2   | Funzioni a due variabili . . . . .                       | 21       |
| 2.13     | Riassunto . . . . .                                      | 23       |
| 2.13.1   | Funzioni a tre variabili . . . . .                       | 25       |
| 2.14     | Realizzazione circuitale delle porte logiche . . . . .   | 26       |
| 2.15     | Riassunto . . . . .                                      | 32       |
| 2.16     | Forme canoniche: <i>minterm</i> . . . . .                | 33       |
| 2.17     | Forme canoniche: <i>maxterm</i> . . . . .                | 34       |
| 2.18     | Interpretazione circuitale . . . . .                     | 35       |
| 2.19     | Riassunto . . . . .                                      | 37       |
| 2.20     | Semplificazione delle espressioni Booleane . . . . .     | 39       |

|          |                                                                    |           |
|----------|--------------------------------------------------------------------|-----------|
| 2.20.1   | Mappe di Karnaugh . . . . .                                        | 39        |
| 2.21     | Riassunto . . . . .                                                | 40        |
| 2.21.1   | Mappa di Karnaugh sui termini massimi . . . . .                    | 42        |
| 2.21.2   | Mappe di Karnaugh a 5 o più variabili . . . . .                    | 42        |
| 2.22     | Condizioni non specificate . . . . .                               | 42        |
| 2.23     | Riassunto . . . . .                                                | 44        |
| 2.24     | Metodo tabellare Quine - Mc Cluskey . . . . .                      | 50        |
| <b>3</b> | <b>Circuiti combinatori</b>                                        | <b>53</b> |
| 3.1      | Introduzione . . . . .                                             | 53        |
| 3.1.1    | Itinerari e livelli . . . . .                                      | 53        |
| 3.2      | Riassunto . . . . .                                                | 54        |
| 3.3      | Analisi dei circuiti combinatori . . . . .                         | 57        |
| 3.4      | Sintesi dei circuiti combinatori . . . . .                         | 57        |
| 3.5      | Riassunto . . . . .                                                | 59        |
| 3.6      | Moduli combinatori . . . . .                                       | 60        |
| 3.6.1    | Decodificatori . . . . .                                           | 60        |
| 3.6.2    | Codificatore . . . . .                                             | 61        |
| 3.6.3    | Selettori . . . . .                                                | 61        |
| 3.6.4    | Selettore d'uscita . . . . .                                       | 61        |
| 3.7      | Riassunto . . . . .                                                | 64        |
| 3.8      | Costruzione modulare di una funzione Booleana . . . . .            | 64        |
| 3.8.1    | Diodi . . . . .                                                    | 65        |
| 3.9      | Riassunto . . . . .                                                | 66        |
| 3.10     | Memorie E-PROM . . . . .                                           | 67        |
| 3.11     | Moduli per la realizzazione dell'unità logico-aritmetica . . . . . | 67        |
| 3.11.1   | Il semisommatore e il sommatore completo . . . . .                 | 67        |
| 3.11.2   | Calcolo della differenza mediante sommatore . . . . .              | 68        |
| 3.12     | Riassunto . . . . .                                                | 69        |
| <b>4</b> | <b>Circuiti sequenziali</b>                                        | <b>71</b> |
| 4.1      | Moduli sequenziali asincroni . . . . .                             | 71        |
| 4.2      | Il Flip-Flop Set-Reset - FFSR . . . . .                            | 72        |
| 4.2.1    | Latch di NOR . . . . .                                             | 72        |
| 4.3      | Riassunto . . . . .                                                | 74        |
| 4.4      | Latch di NAND . . . . .                                            | 76        |
| 4.5      | <i>Flip-Flop</i> SR sincrono . . . . .                             | 76        |
| 4.6      | <i>Flip-Flop</i> JK . . . . .                                      | 76        |
| 4.7      | Flip-Flop di tipo T . . . . .                                      | 77        |
| 4.8      | <i>Flip-Flop</i> di tipo D . . . . .                               | 77        |
| 4.9      | Registri e contatori . . . . .                                     | 77        |
| 4.10     | Contatore . . . . .                                                | 77        |
| 4.11     | Riassunto . . . . .                                                | 78        |
| <b>5</b> | <b>Codifica</b>                                                    | <b>80</b> |
| 5.0.1    | Codifica dei caratteri . . . . .                                   | 80        |
| 5.0.2    | Codifica dei numeri . . . . .                                      | 81        |
| 5.0.3    | Conversione da base 2 a base 10 . . . . .                          | 81        |
| 5.0.4    | Conversione da base 10 a base 2 . . . . .                          | 81        |
| 5.1      | Operazioni binarie . . . . .                                       | 81        |
| 5.2      | Riassunto . . . . .                                                | 82        |
| 5.3      | Rappresentazioni in <i>floating-point</i> . . . . .                | 82        |
| 5.3.1    | Segnali analogici . . . . .                                        | 83        |
| 5.3.2    | Immagini . . . . .                                                 | 83        |
| 5.4      | Riassunto . . . . .                                                | 85        |
| 5.5      | Rappresentazione fisica dei bit . . . . .                          | 86        |
| 5.5.1    | Bit nelle memoria RAM . . . . .                                    | 87        |
| 5.5.2    | Seriale e Parallelo . . . . .                                      | 87        |
| 5.6      | Riassunto . . . . .                                                | 88        |

|          |                                                        |            |
|----------|--------------------------------------------------------|------------|
| <b>6</b> | <b>Rivoluzione microelettronica</b>                    | <b>89</b>  |
| 6.1      | Seconda fase: Transistor . . . . .                     | 89         |
| 6.2      | Circuiti integrati . . . . .                           | 89         |
| 6.3      | Concetto di informazione . . . . .                     | 90         |
| 6.4      | Riassunto . . . . .                                    | 91         |
| 6.5      | Ridondanza . . . . .                                   | 91         |
| <b>7</b> | <b>Storia dell'informatica</b>                         | <b>93</b>  |
| 7.1      | Riassunto . . . . .                                    | 94         |
| 7.2      | Nozione di algoritmo . . . . .                         | 94         |
| 7.3      | Modello RAM . . . . .                                  | 95         |
| 7.4      | Computazione . . . . .                                 | 95         |
| 7.4.1    | Prossima istruzione . . . . .                          | 95         |
| 7.4.2    | STOP della computazione . . . . .                      | 95         |
| 7.4.3    | Configurazione iniziale . . . . .                      | 96         |
| 7.4.4    | Configurazione finale . . . . .                        | 96         |
| 7.4.5    | Convergenza . . . . .                                  | 96         |
| 7.4.6    | Calcolo di una funzione tramite un programma . . . . . | 96         |
| 7.4.7    | Funzione calcolabile . . . . .                         | 96         |
| 7.4.8    | Linguaggio delle funzioni . . . . .                    | 96         |
| 7.5      | Riassunto . . . . .                                    | 97         |
| 7.6      | Programmazione imperativa . . . . .                    | 99         |
| 7.6.1    | Istruzioni logiche di controllo . . . . .              | 99         |
| 7.6.2    | Selezione . . . . .                                    | 99         |
| 7.6.3    | Iterazione while - do . . . . .                        | 100        |
| 7.6.4    | Iterazione repeat - until . . . . .                    | 100        |
| 7.7      | Tesi di Church-Turing . . . . .                        | 100        |
| <b>8</b> | <b>Architettura dei calcolatori</b>                    | <b>101</b> |
| 8.1      | Componenti essenziali di un calcolatore . . . . .      | 101        |
| 8.2      | Ciclo <i>fetch-decode-execute</i> . . . . .            | 101        |
| 8.3      | Riassunto . . . . .                                    | 102        |
| 8.4      | Struttura di base del calcolatore . . . . .            | 102        |
| 8.4.1    | Processore . . . . .                                   | 102        |
| 8.5      | Istruzioni in linguaggio ASSEMBLY . . . . .            | 103        |
| 8.6      | Gerarchia di memoria . . . . .                         | 104        |
| 8.6.1    | Registri CPU . . . . .                                 | 104        |
| 8.6.2    | RAM . . . . .                                          | 105        |
| 8.6.3    | Memoria Cache . . . . .                                | 105        |
| 8.6.4    | Memoria secondaria . . . . .                           | 105        |

# 1 Analogico e digitale

## 1.1 Analogico

Un **segnale analogico** è un segnale (tipicamente di natura elettrica) che segue in modo analogo l'andamento di una grandezza di riferimento (una grandezza fisica).

Pertanto varia in modo continuo (con continuità) nel tempo (senza interruzioni) e ha caratteristiche analoghe alla grandezza di riferimento.

Per trasformare un segnale analogico in uno digitale è necessario impiegare un **trasduttore**, come per convertire un segnale di variazione (come la temperatura) in un segnale elettrico.

## 1.2 Digitale

Un **segnale digitale** (o **discreto**) viene espresso attraverso una sequenza di **simboli** di un **alfabeto finito**.

Un modo per rappresentare un segnale digitale composto da 0 e 1 è quello di associare un valore di tensione al valore 1 e uno al valore 0, assicurandosi che siano ben distinti e identificabili.

L'**ordine di un alfabeto** costituisce il numero di simboli che si possono impiegare per rappresentare un segnale digitale.

**Osservazione:** Negli ultimi 30-40 anni si è assistito a un passaggio graduale dai sistemi analogici ai sistemi digitali. È possibile, infatti, codificare un segnale analogico mediante un segnale digitale, attraverso la tecnica del campionamento.

## 1.3 Teorema del campionamento

Per trasformare un segnale analogico in uno digitale è necessario campionare il segnale, ovvero rilevare il valore del segnale analogico ad intervalli di tempo regolari. La frequenza con cui avvengono i campionamenti prende il nome di **frequenza di campionamento**. Tanto più alta sarà la frequenza di campionamento tanto maggiore sarà la precisione di rappresentazione digitale del segnale analogico corrispondente.

Dopo aver ottenuto la successione di valori di campionamento è possibile ricostruire il segnale analogico per mezzo di quello digitale.

Se la **frequenza di campionamento** è stata sufficientemente alta, ovvero

$$f_c > 2 \cdot B$$

**maggiore del doppio della larghezza di banda del segnale analogico originario**, allora è possibile ricostruire il segnale senza ambiguità.

**Osservazione:** Si osservi che la **larghezza di banda** di un segnale è l'**insieme delle frequenze** di cui è composto il segnale. Non solo, ma il segnale analogico che si otterrà attraverso la trasformazione inversa (digitale - analogico), a seguito di un precedente campionamento, sarà indistinguibile dal segnale analogico di partenza se la frequenza adottata corrisponde a quella di Nyquist.

**Osservazione:** Per la rappresentazione di un segnale **ad alta fedeltà** è necessario avere una larghezza di banda maggiore, dai 20Hz ai 22KHz. Mentre per una normale conversazione telefonica è necessaria una larghezza di banda dai 300Hz ai 3400Hz.

L'elettronica che gestisce i segnali digitali è molto più conveniente rispetto a quella impiegata per gestire i segnali analogici.

## 1.4 Rumore

Nei sistemi reali, al segnale analogico si sovrappone sempre un segnale di **rumore** (ovvero tutto ciò che **non è segnale**). Nel caso dei circuiti elettronici chiusi, secondo la legge di **Lenz**, ogni variazione di campo elettromagnetico è rumore per il segnale elettrico del circuito stesso.

Il più comune dei rumori è quello **termico**: l'agitazione delle particelle atomiche dovuta al calore induce del rumore.

I rumori elettromagnetici possono essere risolti ponendo il circuito interessato all'interno di uno schermo chiuso conduttore, ovvero la **gabbia di Faraday**: è quello che accade negli ospedali

all'interno delle stanze di misurazione molto precisa, come la T.A.C..

Il rumore termico, invece, è molto difficile da arginare, in quanto non si ha sempre la disponibilità (fatta eccezione per casi particolari) di lavorare a  $-273,15^{\circ}\text{C}$ , ovvero in corrispondenza dello 0 assoluto.

Tuttavia, quando un segnale analogico viene **inquinato** dal rumore, è impossibile eliminare a questo stadio tale inquinamento; infatti, l'inquinamento da rumore è un fenomeno che presenta un carattere di **irreversibilità** nei segnali analogici.

L'unico modo per poter arginare il problema dell'inquinamento da rumore è quello di aumentare il **rapporto segnale-disturbo**. Pertanto, si dovrà lavorare con segnale più consistenti, anche se questo determina sempre un aumento della complessità (e del costo) della circuiteria a disposizione.

#### 1.4.1 Rumore nei segnali digitali

Se si usano i segnali digitali, il segnale può essere ricostruito in modo perfetto, anche se è stato inquinato dal rumore (a meno che il segnale di rumore non sia troppo forte).

Infatti, nel mondo digitale, l'inquinamento da rumore non è irreversibile. Nel caso di una onda quadra inquinata da rumore, per esempio, calcolando la media delle variazioni del segnale determinate dal rumore, se essa è superiore o inferiore ad un certo valore si può discernere se codificare il segnale come 0, oppure 1.

La percezione del rumore, quindi, nei segnali digitali è diversa rispetto a quella dei segnali analogici. Infatti, a differenza dei **segnali analogici**, in cui **il rumore si sovrapponeva costantemente a tutto il segnale**, alterandolo in maniera irreversibile, **nei sistemi digitali è sempre possibile ricostruire il segnale originario**, con l'aggiunta di una probabilità più o meno elevata di incorrere in un errore di codifica (risolvibile con algoritmi correttori): taluno costituisce un primo grande vantaggio dei sistemi digitali.

### 1.5 L'importanza dell'alfabeto binario

Tutti gli alfabeti con cui operano i sistemi digitali hanno **ordine 2**. Questo per due motivi:

1. Da un punto di vista concettuale, esiste l'**algebra booleana** estremamente efficiente e performante.
2. Da un punto di vista tecnico-pratico, se si dovesse operare con alfabeti con ordine maggiore di 2, aumenterebbe rovinosamente la **temperatura di esercizio** dei sistemi digitali. Infatti, nei circuiti elettronici sono presenti elementi attivi e passivi.

Gli elementi passivi (che seguono le consuete regole dell'elettronica) sono

- **resistenze**
- **induttanze**
- **capacità**

Mentre gli elementi attivi sono i **transistor**, i quali vengono integrati in piccolissime piastrine di silicio. Le loro caratteristiche sono completamente diverse rispetto a quelle degli elementi passivi. La ragione dell'ordine binario degli alfabeti di codifica dei sistemi digitali è da ricercarsi nel funzionamento del transistor.

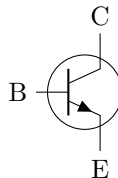


Figura 1: Transistor

Questo componente elettronico attivo è un dispositivo che comanda un flusso di corrente tra **collettore** ed **emettitore** attraverso un elettrodo che prende il nome di **base**. Se la corrente di base è pressoché nulla, allora si ha una **interdizione** (non passa corrente), mentre se la corrente di base è significativamente maggiore di 0, allora si ha **conduzione piena** (o

**saturazione).**

Pertanto, un transistor, in un dato istante, si può trovare solamente in uno di due stati:

- Se la corrente è massima e la tensione è nulla (**saturazione**) si ottiene il valore logico 0.

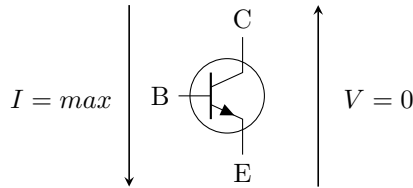


Figura 2: Transistor in saturazione

- Se la corrente è nulla e la tensione è massima (**interdizione**) si ottiene il valore logico 1.

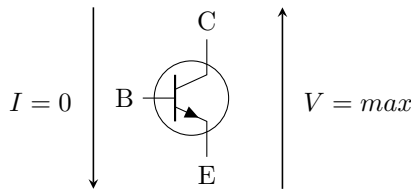


Figura 3: Transistor in interdizione

La temperatura di esercizio viene determinata dalla seguente formula:

$$P = V \cdot I$$

per cui se il transistor è in saturazione, la corrente è massima e la tensione è nulla, quindi la potenza è nulla. Mentre se il transistor è in interdizione, la corrente è nulla e la tensione è massima, quindi la potenza è ancora una volta nulla.

Per cui la potenza dissipata dai transistors è sempre nulla in questi due stati, ma assume un valore non nullo nel passaggio dall'uno all'altro stato.

A queste potenze deve aggiungersi anche la dissipazione di calore delle resistenze, che può essere variabile.

Se si aggiungesse un terzo stato logico, ad esso bisognerebbe attribuire un valore di tensione e uno di corrente, necessariamente diversi da 0, per cui anche la potenza dissipata sarà diversa da 0 e questo aumenterebbe la temperatura di esercizio.

Ciò accadrebbe per ogni stato logico diverso dai due del sistema binario che si aggiungerebbe se si avesse un alfabeto di codifica di ordine  $> 2$ . E la dissipazione, per quanto piccola, deve essere moltiplicata per il numero di transistors che si trovano in questo ipotetico stato che, in un moderno chip, sono **miliardi**: pertanto la dissipazione di calore non sarebbe trascurabile.

## 1.6 Vantaggi sistemi digitali

Come si è potuto appurare, vi sono molti vantaggi nell'adottare sistemi digitali in luogo di quelli analogici:

1. Minor sollecitazione al rumore
2. Maggior affidabilità (arginando la **deriva termica**)
3. Presenza dell'algebra booleana
4. Minori dissipazioni di potenza

I dispositivi analogici, invece, lavorano con la dissipazione di potenza, sempre ed in ogni stato; questo costituisce un problema, in quanto dissipazione di potenza significa scaldare per **effetto Joule**. Se la temperatura si innalza eccessivamente (**deriva termica**) il componente (transistor) si brucia.

Ciò non accade (o accade molto raramente) nella circuiteria digitale, in quanto i dispositivi attivi dei circuiti non dissipano potenza a riposo.

5 Ottobre 2021

## 1.7 Riassunto

Ci sono due diversi sistemi in utilizzo al giorno d'oggi, analogico e digitale, anche se negli ultimi 40 anni si è progressivamente passati dai sistemi analogici a quelli digitali.

È possibile passare da un segnale analogico ad uno digitale attraverso la tecnica del campionamento. Il teorema di campionamento afferma che se la frequenza di campionamento è almeno doppia della larghezza di banda del segnale da campionare, allora il segnale digitale risultante sarà indistinguibile da quello originario.

Il segnale di rumore, inoltre, è sempre presente nei sistemi reali, e si sovrappone in maniera inevitabile al segnale di partenza, inquinandolo; per cercare di far fronte a tale problema bisogna aumentare il rapporto **segnale-rumore**.

Nei sistemi digitali, invece, il segnale di rumore non altera il valore del segnale di partenza, ma può indurre a degli errori di ricostruzione (che possono essere risolti con dei sistemi di correzione degli errori).

I motivi per cui i sistemi digitali sono basati su una logica binaria sono svariati:

1. La presenza e l'efficienza dell'algebra booleana;
2. Migliore affidabilità, in quanto meno soggetti alla deriva termica, in quanto presentano una dissipazione nulla a riposo;
3. Meno soggetti al segnale di rumore e maggiore facilità di ricostruzione del segnale;
4. Nei sistemi elettronici sono presenti dispositivi passivi e attivi, ovvero i transistor, i quali non dissipano potenza a riposo, proprio perché a riposo si possono trovare in due soli stati, in cui la corrente è massima e la tensione è nulla, oppure la corrente è nulla e la tensione è massima.

Se si avesse un terzo stato logico, infatti, il transistor che si trovasse su quello stato dissiperebbe potenza a riposo, per cui ammettendo che anche solo  $\frac{1}{3}$  dei transistor totali si trovi su questo stato, la dissipazione di calore sarebbe considerevole e non trascurabile: pertanto, si necessiterebbe di sistemi di raffreddamento molto costosi per un corretto funzionamento del sistema.

Nei circuiti analogici (come gli amplificatori) i transistor devono essere polarizzati e quindi presentano una corrente di base tale da dissipare potenza anche a riposo e questo determina un aumento della temperatura di esercizio.

Nei transistor, infatti, è presente una corrente di base chiamata **corrente di deriva termica**  $I_{C_{v_0}}$  che tende ad aumentare esponenzialmente all'aumentare della temperatura di esercizio; aumentando la corrente, aumenta la dissipazione e ciò alla **deriva termica**, e quindi alla rottura dei transistor.



## 2 Algebra booleana

Tutta la tecnologia elettronica fa uso dell'algebra booleana che, grazie agli studi di Shannon, è stato dimostrato essere fondamentale nella progettazione digitale.

L'ideazione dell'algebra booleana è da attribuire a **George Boole**, ovvero un logico, mentre essa verrà proficuamente impiegata, come anticipato, da **Claude Elwood Shannon** per la prima volta come strumento per la progettazione logica di circuiti composti da **relé**, ovvero un interruttore comandato da correnti.

Shannon, infatti, si accorse che l'algebra booleana poteva essere utilizzata per la gestione di una rete complessa di interruttori, tale da rendere tale operazione molto più semplificata.

### 2.1 Costanti booleane

Le due costanti booleane sono **vero** e **falso**, corrispondenti ai due soli valori logici contemplati nei sistemi binari (0, ovvero il valore basso di tensione, in generale) e (1, ovvero il valore alto di tensione, in generale).

### 2.2 Calcolo funzionale di verità

Per la risoluzione di un problema logico è necessario tenere conto di una serie di condizioni che devono essere verificate affinché possa essere prodotto un risultato logico (o pratico); in un problema concreto, si potrebbe pensare di disporre di una serie di sensori atti ciascuno a rilevare il valore logico di una certa proposizione.

L'algebra booleana viene usata, invece, per manipolare delle equazioni logiche composte da valori anch'essi di natura logica.

Il modo per rappresentare il valore logico di tali proposizioni è costituito dalle **variabili binarie booleane**; il metodo che permette di manipolare tali variabili e di assegnare ad esse un valore di verità è conosciuto come **calcolo funzionale di verità**.

Per ogni affermazione assertiva quale

$$\text{"La cintura di sicurezza è allacciata"} \rightarrow B_L$$

si attribuisce il valore della proposizione ad una variabile booleana.

Per poter tenere conto della composizione di funzioni di verità si usano dei **connettivi logici**, come l'**and** ( $\cap$ ) che produce un valore positivo se e solo se entrambe le proposizioni elementari sono vere:

$$A \cap B$$

Un altro connettivo logico è l'**or** ( $\cup$ ) che produce un valore positivo se almeno una delle due proposizioni elementari è vera:

$$A \cup B$$

Il modo per ottenere il risultato logico di una funzione composta di verità è rappresentato dalle tabelle di verità.

### 2.3 Tavole di verità

Di seguito la tavola di verità dei principali connettivi:

|     |     | AND        | OR         | XOR          | XNOR         | NOR              | NAND  |
|-----|-----|------------|------------|--------------|--------------|------------------|-------|
| $A$ | $B$ | $A \cap B$ | $A \cup B$ | $A \oplus B$ | $A \equiv B$ | $A \downarrow B$ | $A B$ |
| $F$ | $F$ | $F$        | $F$        | $F$          | $V$          | $V$              | $V$   |
| $F$ | $V$ | $F$        | $V$        | $V$          | $F$          | $F$              | $V$   |
| $V$ | $F$ | $F$        | $V$        | $V$          | $F$          | $F$              | $V$   |
| $V$ | $V$ | $V$        | $V$        | $F$          | $V$          | $F$              | $F$   |

Tabella 1: Tavola di verità dei principali connettivi

I primi quattro connettivi logici sono molto comuni, mentre gli ultimi due sono fondamentali, perché sono in grado di ricostruire ciascuna delle  $2^4 = 16$  possibili combinazioni di  $V$  e  $F$ .

Si tenga presente che ciascun connettivo logico presenta un suo corrispettivo, chiamato **porta logica**. Pertanto, con le porte logiche **NOR** e **NAND** (le cui funzioni booleane associate sono le ultime due della Tabella 1) è possibile costruire qualunque funzione logica.

L'equazione logica si ottiene combinando fra di loro tutte le variabili logiche prese in considerazione attraverso i connettivi logici, pertanto:

$$C \equiv A \cap G \cap [(S_L \cap \overline{B_L}) \cup (S_R \cap \overline{B_R})]$$

Per costruire tutte le possibili combinazioni di  $V$  e  $F$  su 4 posti basta eseguire un semplice calcolo

$$2^4 = 16$$

Infatti, in generale, se si dispone di  $n$  bit il numero delle possibili  $n$ -ple binarie è pari a

$$2^n$$

## 2.4 Connettivi binari

Nella Tabella 1 sono stati individuati sei connettivi logici, AND, OR, XOR e XNOR, NOR e NAND, che sono i più interessanti. Poiché per ciascuna coppia di valori delle variabili ci sono due possibilità, come si è detto vi sono 4 righe, e quindi  $2^4 = 16$  connettivi logici, come illustrato di seguito:

|     |     | 0      | 1   | 2   | 3   | 4        | 5   | 6   | 7   | 8      | 9            | 10       | 11        | 12  | 13  | 14   | 15        |     |
|-----|-----|--------|-----|-----|-----|----------|-----|-----|-----|--------|--------------|----------|-----------|-----|-----|------|-----------|-----|
|     |     | AND    |     |     |     | XOR      |     |     |     | OR     | NOR          | XNOR     |           |     |     | NAND |           |     |
| $A$ | $B$ | $\cap$ |     |     |     | $\oplus$ |     |     |     | $\cup$ | $\downarrow$ | $\equiv$ | $\subset$ |     |     |      | $\supset$ | $ $ |
| $F$ | $F$ | $F$    | $F$ | $F$ | $F$ | $F$      | $F$ | $F$ | $F$ | $V$    | $V$          | $V$      | $V$       | $V$ | $V$ | $V$  | $V$       |     |
| $F$ | $V$ | $F$    | $F$ | $F$ | $F$ | $V$      | $V$ | $V$ | $V$ | $F$    | $F$          | $F$      | $F$       | $V$ | $V$ | $V$  | $V$       |     |
| $V$ | $F$ | $F$    | $F$ | $V$ | $V$ | $F$      | $F$ | $V$ | $V$ | $F$    | $F$          | $V$      | $V$       | $F$ | $F$ | $V$  | $V$       |     |
| $V$ | $V$ | $F$    | $V$ | $F$ | $V$ | $F$      | $V$ | $F$ | $V$ | $F$    | $V$          | $F$      | $V$       | $F$ | $V$ | $F$  | $V$       |     |

Tabella 2: Tavola di verità dei 16 connettivi binari

Due ulteriori connettivi logici sono quelli di implicazione  $A \subset B$  e  $A \supset B$  (ovvero B implica A e A implica B).

Si noti che

$$A \equiv B \leftrightarrow A \supset B \text{ e } A \subset B$$

ovvero

$$(A \equiv B) \equiv (A \supset B) \cap (A \subset B)$$

e si può verificare attraverso le tabelle di verità.

| $A$ | $B$ | $A \supset B$ | $A \subset B$ | $(A \supset B) \cap (A \subset B)$ | $A \equiv B$ |
|-----|-----|---------------|---------------|------------------------------------|--------------|
| $F$ | $F$ | $V$           | $V$           | $V$                                | $V$          |
| $F$ | $V$ | $V$           | $F$           | $F$                                | $F$          |
| $V$ | $F$ | $F$           | $V$           | $F$                                | $F$          |
| $V$ | $V$ | $V$           | $V$           | $V$                                | $V$          |

Tabella 3: Tabella di verità dell'implicazione

## 2.5 Insiemi minimi di connettivi

Come si è visto nella Tabella 3, alcuni connettivi binari possono essere espressi sulla base di altri connettivi; in questo caso la XNOR viene espressa in funzione di  $\supset$ ,  $\subset$  e  $\cap$ .

Si presenta quindi spontanea la domanda di quali siano i connettivi idonei a esprimere tutti gli altri, e quale sia l'insieme minimo tra essi che permette di esprimere tutte le 16 combinazioni della Tabella 2.

Si consideri, ad esempio, l'insieme dei connettivi che contiene solo  $\cup$  e  $\cap$ , ovvero OR e AND. I casi in cui sia A che B sono falsi sono chiaramente situazioni critiche; infatti dalla Tabella 1 si osserva che sia  $A \cup B$  che  $A \cap B$  sono ambedue falsi; quindi, sulla base di questi due soli connettivi, non è possibile esprimere connettivi che assumono valore vero quando A e B sono falsi.

Per poter esprimere anche questi casi, è necessario arricchire l'insieme introducendo anche il connettivo NOT. Ecco allora che l'insieme dei connettivi AND, OR, NOT è un insieme sufficiente per ricostruire tutti gli altri connettivi, come si può verificare dalla tavola di verità in Tabella 4.

|             |                                                     |     |             |                                                     |           |
|-------------|-----------------------------------------------------|-----|-------------|-----------------------------------------------------|-----------|
| <i>FFVV</i> | <i>A</i>                                            |     | <i>FFVV</i> | <i>A</i>                                            |           |
| <i>FVFF</i> | <i>B</i>                                            |     | <i>FVFF</i> | <i>B</i>                                            |           |
| <i>FFFF</i> | $f_0 \equiv A \cap \bar{A}$                         | AND | <i>VFFF</i> | $f_8 \equiv \bar{A} \cup \bar{B}$                   | NOR       |
| <i>FFFV</i> | $f_1 \equiv A \cap B$                               |     | <i>VFFV</i> | $f_9 \equiv (A \cup \bar{B}) \cap (\bar{A} \cup B)$ | XNOR      |
| <i>FFVF</i> | $f_2 \equiv A \cap \bar{B}$                         |     | <i>VFVF</i> | $\equiv (\bar{A} \cap \bar{B}) \cup (A \cap B)$     |           |
| <i>FFVV</i> | $f_3 \equiv A$                                      |     | <i>VFVV</i> | $f_{10} \equiv \bar{B}$                             | $\subset$ |
| <i>FVFF</i> | $f_4 \equiv \bar{A} \cap B$                         | XOR | <i>VVFF</i> | $f_{11} \equiv A \cup \bar{B}$                      |           |
| <i>FVFF</i> | $f_5 \equiv B$                                      |     | <i>VVFF</i> | $f_{12} \equiv \bar{A}$                             | $\supset$ |
| <i>FVVF</i> | $f_6 \equiv (A \cap \bar{B}) \cup (\bar{A} \cap B)$ |     | <i>VVVF</i> | $f_{13} \equiv \bar{A} \cup B$                      |           |
|             | $\equiv (\bar{A} \cup \bar{B}) \cap (A \cup B)$     |     | <i>VVVF</i> | $f_{14} \equiv \bar{A} \cap \bar{B}$                | NAND      |
| <i>FVVV</i> | $f_7 \equiv A \cup B$                               | OR  | <i>VVVV</i> | $f_{15} \equiv A \cup \bar{A}$                      |           |

Tabella 4: I 16 connettivi binari espressi mediante AND, OR, NOT

**Osservazione: Ogni connettivo binario** si può esprimere, pertanto, tramite combinazione di altri connettivi binari.

Infatti, come si è detto, i connettivi **NAND** e **NOR** sono **connettivi universali** e quindi sono in grado di rappresentare tutti gli altri connettivi binari.

Si noti, infatti, come utilizzando **AND**, **OR** e **NOT** si possono costruire tutte e 16 le possibili funzioni logiche.

Quasi tutti i connettivi da  $f_0, \dots, f_{15}$  possono essere costruiti usando solo il connettivo  $\cup$  oppure solo il connettivo  $\cap$  (con l'eventuale negazione). L'unica eccezione è costituita dai connettivi  $f_6$  ed  $f_9$ , che necessitano l'impiego contemporaneo di  $\cup$  e  $\cap$ .

L'espressione risolutiva per la  $f_9$  si può ricavare tenendo conto dell'equivalenza messa in evidenza nella Tabella 3 e del fatto che  $A \supset B \equiv \bar{A} \cup B$  e  $A \subset B \equiv A \cup \bar{B}$ , e dunque la  $f_9 \equiv (A \cup \bar{B}) \cap (\bar{A} \cup B)$ , mentre la  $f_6$  è la sua negazione.

È interessante notare, inoltre, che ci sono due modi per realizzare  $f_6$  e  $f_9$ , combinando assieme  $\cup$  e  $\cap$ , ma a ben guardare un po' tutti i connettivi possono essere espressi in modi alternativi: per

esempio  $f_0 \equiv A \cap \bar{A}$  ma anche  $f_0 \equiv \overline{A \cup \bar{A}}$ , e così via.

Tutto ciò deriva dalla circostanza che l'insieme dei connettivi AND, OR, NOT non costituisce l'insieme minimo di connettivi atti ad ottenere tutti gli altri tramite combinazione dei propri elementi; il connettivo AND può, infatti, essere espresso in funzione del connettivo OR (e viceversa); a tal riguardo vale l'importante **Teorema di De Morgan** (§ 2.6).

## 2.6 Teorema di De Morgan

Si espone di seguito il teorema di **De Morgan**:

| TEOREMA DI DE MORGAN |                                              |
|----------------------|----------------------------------------------|
| Si verifica che      | $\overline{A \cup B} = \bar{A} \cap \bar{B}$ |
| così come            | $\overline{A \cap B} = \bar{A} \cup \bar{B}$ |

Si può dimostrare tale teorema attraverso il **metodo dell'induzione perfetta**, quindi compilando una tabella di verità per verificare l'effettiva veridicità dell'enunciato.

| $A$ | $B$ | $\bar{A}$ | $\bar{B}$ | $\bar{A} \cap \bar{B}$ | $A \cup B$ | $\overline{A \cup B}$ | $A$ | $B$ | $\bar{A}$ | $\bar{B}$ | $\bar{A} \cup \bar{B}$ | $A \cap B$ | $\overline{A \cap B}$ |
|-----|-----|-----------|-----------|------------------------|------------|-----------------------|-----|-----|-----------|-----------|------------------------|------------|-----------------------|
| $F$ | $F$ | $V$       | $V$       | $V$                    | $F$        | $V$                   | $F$ | $F$ | $V$       | $V$       | $V$                    | $F$        | $V$                   |
| $F$ | $V$ | $V$       | $F$       | $F$                    | $V$        | $F$                   | $F$ | $V$ | $V$       | $F$       | $V$                    | $F$        | $V$                   |
| $V$ | $F$ | $F$       | $V$       | $F$                    | $V$        | $F$                   | $V$ | $F$ | $F$       | $V$       | $V$                    | $F$        | $V$                   |
| $V$ | $V$ | $F$       | $F$       | $F$                    | $V$        | $F$                   | $V$ | $V$ | $F$       | $F$       | $F$                    | $V$        | $F$                   |

Tabella 5: Dimostrazione del teorema di De Morgan con tabelle di verità

Il teorema di **De Morgan** si può generalizzare al caso di più di due variabili. Si verifica, infatti, che la negazione di un connettivo di più variabili si ottiene negando ogni variabile e scambiando tra di loro i due operatori  $\cup$  e  $\cap$ .

In termini formali si ha

$$\overline{F}(x_1, x_2, \dots, x_n)[\cup, \cap] = F(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)[\cap, \cup]$$

I connettivi che legano le variabili non negate si esprimono allora come

$$A \cap B \equiv \overline{\bar{A} \cup \bar{B}} \quad \text{e} \quad A \cup B \equiv \overline{\bar{A} \cap \bar{B}}$$

Poiché l'AND può essere espresso in funzione di OR e NOT, quest'ultimo insieme è da solo sufficiente per coprire tutti i connettivi formulati in Tabella 4 che sono riportati nella prima colonna della Tabella 6. Per farlo bisogna sostituire tutti i simboli  $\cap$  con l'espressione  $A \cap B \equiv \overline{\bar{A} \cup \bar{B}}$ ; così facendo si ottiene la prima colonna della Tabella 6.

Lo stesso ragionamento può essere fatto per l'OR, che può essere espresso in funzione di AND e NOT; per farlo bisogna sostituire tutti i simboli  $\cup$  con l'espressione  $A \cup B \equiv \overline{\bar{A} \cap \bar{B}}$ ; così facendo si ottiene la seconda colonna della Tabella 6.

|           | AND, OR, NOT                                        | OR, NOT                                             | AND, NOT                                            |
|-----------|-----------------------------------------------------|-----------------------------------------------------|-----------------------------------------------------|
| AND       | $f_0 \equiv A \cap \bar{A}$                         | $\overline{\bar{A} \cup A}$                         | $A \cap \bar{A}$                                    |
|           | $f_1 \equiv A \cap B$                               | $\overline{\bar{A} \cup \bar{B}}$                   | $A \cap \bar{B}$                                    |
|           | $f_2 \equiv A \cap \bar{B}$                         | $\overline{\bar{A} \cup B}$                         | $A \cap \bar{B}$                                    |
|           | $f_3 \equiv A$                                      | $A$                                                 | $A$                                                 |
|           | $f_4 \equiv \bar{A} \cap B$                         | $\overline{A \cup \bar{B}}$                         | $\bar{A} \cap B$                                    |
|           | $f_5 \equiv B$                                      | $B$                                                 | $B$                                                 |
| XOR       | $f_6 \equiv (A \cap \bar{B}) \cup (\bar{A} \cap B)$ | $\overline{(\bar{A} \cup B) \cup (A \cup \bar{B})}$ | $\overline{(A \cap \bar{B}) \cap (\bar{A} \cap B)}$ |
| OR        | $f_7 \equiv A \cup B$                               | $A \cup B$                                          | $\overline{\bar{A} \cap \bar{B}}$                   |
| NOR       | $f_8 \equiv \overline{A \cup B}$                    | $\overline{A \cup B}$                               | $\bar{A} \cap \bar{B}$                              |
| XNOR      | $f_9 \equiv (A \cup \bar{B}) \cap (\bar{A} \cup B)$ | $\overline{(A \cup \bar{B}) \cup (\bar{A} \cup B)}$ | $\overline{(\bar{A} \cap B) \cap (A \cap \bar{B})}$ |
| $\subset$ | $f_{10} \equiv \bar{B}$                             | $\bar{B}$                                           | $\bar{B}$                                           |
|           | $f_{11} \equiv A \cup \bar{B}$                      | $A \cup \bar{B}$                                    | $\overline{\bar{A} \cap B}$                         |
|           | $f_{12} \equiv \bar{A}$                             | $\bar{A}$                                           | $\bar{A}$                                           |
| $\supset$ | $f_{13} \equiv \bar{A} \cup B$                      | $\bar{A} \cup B$                                    | $\overline{A \cap \bar{B}}$                         |
| NAND      | $f_{14} \equiv \overline{A \cap B}$                 | $\overline{A \cap B}$                               | $\overline{A \cap B}$                               |
|           | $f_{15} \equiv A \cup \bar{A}$                      | $A \cup \bar{B}$                                    | $\overline{\bar{A} \cap A}$                         |

Tabella 6: I 16 connettivi binari espressi rispettivamente mediante AND, OR, NOT, mediante OR, NOT e mediante AND, NOT

**Osservazione:** Quindi non solo è possibile esprimere le 16 possibili funzioni logiche attraverso le tre sole funzioni logiche **AND**, **OR** e **NOT**, ma addirittura è possibile farlo solo con **AND** e **NOT** o solo con **OR** e **NOT**, grazie al teorema di **De Morgan**.

## 2.7 Riassunto

L'algebra booleana è atta a gestire le combinazioni dei due valori logici VERO e FALSO, mentre i sensori sono gli strumenti pratici che vengono utilizzati per ottenere la validazione di certe condizioni logiche al fine di estrapolarne il corrispettivo valore logico.

La combinazione di tali condizioni avviene attraverso i connettivi logici:

- AND: Produce il valore VERO se entrambe le condizioni sono vere.
- OR: Produce il valore VERO se almeno una delle due condizioni è vera.
- XOR: che produce il valore VERO se le due condizioni hanno un diverso valore logico.

Naturalmente oltre a queste funzioni vi sono le corrispettive negazioni NAND, NOR e XNOR. Poiché per ciascuna coppia di valori delle variabili è possibile associare solamente due valori, ovvero VERO e FALSO, si possono avere fino a  $2^4 = 16$  possibili combinazioni; da ciò si evince che, oltre ai connettivi logici principali, ve ne sono anche degli altri che possono essere ottenuti attraverso la combinazioni dei connettivi binari noti, per i quali, al fine di verificare la corretta creazione di una nuova funzione di verità è possibile utilizzare il metodo dell'induzione perfetta.

Per poter capire come è possibile creare le 16 funzioni logiche non è possibile procedere attraverso la combinazione delle sole funzioni AND e OR, ma deve essere utilizzato anche il NOT. Non solo, ma attraverso il teorema di **De Morgan** è possibile anche impiegare solamente AND e NOT, oppure OR e NOT per la creazione di tutte le 16 possibili funzioni logiche.

Ancora una volta, la correttezza di quanto affermato da **De Morgan** può essere verificata attraverso il metodo dell'**induzione perfetta**, come si è visto in Tabella 5.

## 2.8 Insieme minimo

Si osservi che l'insieme minimo che consente di rappresentare tutti i connettivi può essere ulteriormente ridotto al solo connettivo NOR, oppure al solo connettivo NAND. Per farlo è sufficiente esprimere AND, OR e NOT in funzione del solo NOR oppure del solo NAND. Per questo motivo i connettivi NOR e NAND sono chiamati **connettivi universali**, in quanto da soli sono in grado di creare tutti i 16 connettivi logici, come si vede di seguito per la porta NOR:

- $\bar{A} \equiv \overline{A \cup A} \equiv A \downarrow A$
- $A \cup B \equiv \overline{\overline{A \cup B}} \equiv \overline{A \downarrow B} \equiv (A \downarrow B) \downarrow (A \downarrow B)$
- $A \cap B \equiv \overline{\overline{A \cap B}} \equiv \overline{A \cup B} \equiv \overline{A \downarrow B} \equiv (A \downarrow A) \downarrow (B \downarrow B)$

E la stessa cosa accade anche per la porta NAND:

- $\bar{A} \equiv \overline{A \cap A} \equiv A|A$
- $A \cup B \equiv \overline{\overline{A \cup B}} \equiv \overline{A \cap B} \equiv \overline{A|B} \equiv (A|A)|(B|B)$
- $A \cap B \equiv \overline{\overline{A \cap B}} \equiv \overline{A|B} \equiv (A|B)|(A|B)$

Tale risultato è straordinario e può avere dei risvolti molti vantaggiosi, specialmente di natura economica, in quanto è possibile impiegare solamente una porta per realizzare un'intera rete logica.

## 2.9 Impostazione assiomatica dell'algebra booleana

Il calcolo booleano costruito in precedenza in forma puramente euristica deve essere rielaborato in forma più rigorosa in modo tale da manipolare complicate espressioni logiche in maniera semiautomatica e più semplificata.

Per la costruzione di uno strumento operativo che abbia una infrastruttura logica solida e affidabile è necessario formulare una teoria, di tipo algebrico, che sia ben costituita dal punto di vista formale, ovvero una teoria assiomatica che si articola in una sequenza di assiomi fondamentali.

In generale, è possibile affermare che gli **assiomi** siano delle **verità primitive, non contestabili**,

**non dimostrabili e date una volta per tutte.**

A partire dalle verità primitive si possono costruire (attraverso la tecnica di inferenza o il principio di induzione) delle nuove verità, ovvero i teoremi, e la combinazione di tali strumenti costituisce una **teoria**.

Di seguito si espone la definizione di **Algebra Booleana**:

### ALGEBRA BOOLEANA

Un'Algebra Booleana è un insieme  $\mathcal{B}$  caratterizzato dagli assiomi (univocamente determinati) che vengono esposti di seguito.

#### 2.9.1 Assioma 0 (A0) - Leggi di composizione

Vi sono due leggi di composizione interna di-arie, denominate rispettivamente AND e OR, e una legge di composizione interna unaria, denominata NOT; tali leggi (od “operatori Booleani”) sono definite come segue:

| Operatore | Nome         | Simboli usati              | che soddisfa                                                    |
|-----------|--------------|----------------------------|-----------------------------------------------------------------|
| AND(x, y) | coniunzione  | $x \wedge y$ o $x \cdot y$ | $x \cdot y = 1$ se $x = 1, y = 1$<br>$x \cdot y = 0$ altrimenti |
| OR(x, y)  | disgiunzione | $x \vee y$ o $x + y$       | $x + y = 0$ se $x = 0, y = 0$<br>$x + y = 1$ altrimenti         |
| NOT(x)    | negazione    | $\neg x$ o $\bar{x}$       | $\bar{x} = 0$ se $x = 1$<br>$\bar{x} = 1$ se $x = 0$            |

Tabella 7: Leggi di composizione interna dell'Algebra Booleana

Inoltre, per gli elementi dell'insieme  $\mathcal{B}$  valgono i seguenti assiomi:

#### 2.9.2 Assioma 1 (A1) - Esistenza

Esistono due elementi  $x, y \in \mathcal{B}$  tali che  $x \neq y$ .

#### 2.9.3 Assioma 2 (A2) - Chiusura

Se  $x, y \in \mathcal{B}$  allora  $x \cdot y \in \mathcal{B}$ ,  $x + y \in \mathcal{B}$ ,  $\bar{x} \in \mathcal{B}$ ,  $\bar{y} \in \mathcal{B}$ .

#### 2.9.4 Assioma 3 (A3) - Elemento neutro

Esiste un elemento neutro per la disgiunzione, indicato con 0, tale che

$$x + 0 = x$$

Esiste un elemento neutro per la congiunzione, indicato con 1, tale che

$$x \cdot 1 = x$$

#### 2.9.5 Assioma 4 (A4) - Commutatività

$\forall x, y \in \mathcal{B}$  valgono le seguenti relazioni:

$$x + y = y + x$$

$$x \cdot y = y \cdot x$$

### 2.9.6 Assioma 5 (A5) - Associatività

$\forall x, y \in \mathcal{B}$  valgono le seguenti relazioni:

$$x + (y + z) = (x + y) + z$$

$$x \cdot (y \cdot z) = (x \cdot y) \cdot z$$

### 2.9.7 Assioma 6 (A6) - Distributività

$\forall x, y \in \mathcal{B}$  valgono le seguenti relazioni:

$$x + (y \cdot z) = (x + y) \cdot (x + z)$$

$$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$$

### 2.9.8 Assioma 7 (A7) - Complementarietà

$\forall x \in \mathcal{B}$  vale:

$$x \cdot \bar{x} = 0$$

$$x + \bar{x} = 1$$

Il più semplice insieme  $\mathcal{B}$  che soddisfa i postulati di cui sopra è rappresentato dalla cosiddetta **Algebra Booleana a due elementi**, o **Algebra Binaria**, basata sugli elementi 0 e 1 e caratterizzata dalle seguenti tre leggi di composizione

|         |   |   |  |     |   |   |  |           |   |   |
|---------|---|---|--|-----|---|---|--|-----------|---|---|
| $\cdot$ | 0 | 1 |  | $+$ | 0 | 1 |  | $A$       | 0 | 1 |
| 0       | 0 | 0 |  | 0   | 0 | 1 |  | $\bar{A}$ | 1 | 0 |
| 1       | 0 | 1 |  | 1   | 1 | 1 |  |           |   |   |
| AND     |   |   |  | OR  |   |   |  | NOT       |   |   |

Tabella 8: Le 3 leggi di composizione fondamentali

L'Algebra Booleana a due elementi e quella impiegata per la progettazione delle reti logiche; i suoi due elementi 1 e 0 corrispondono alle costanti logiche VERO e FALSO (V e F) prima introdotte, e le tre leggi di composizione interna sono rispettivamente AND, OR e NOT e corrispondono ai connettivi precedentemente introdotti nelle tavole di verità. In questo contesto la simbologia che si usa è però diversa; in particolare si osservi che i simboli  $+$  e  $\cdot$  nulla hanno a che vedere con gli stessi simboli usati nel contesto dell'aritmetica.

Un altro importante esempio di Algebra Booleana è quello derivante dalla teoria degli insiemi; si consideri un insieme  $S$  e il corrispondente insieme delle parti  $\mathcal{P}(S)$ , costituito dall'insieme di tutti i sottoinsiemi costruiti con elementi di  $S$ . Allora  $\mathcal{P}(S)$  è un'Algebra Booleana, nella quale sono definite tre leggi di composizione interna date dall'intersezione  $\cap$ , dall'unione  $\cup$  e dalla complementazione tra insiemi  $\bar{A}$ . L'elemento neutro per la disgiunzione, o 0 dell'insieme, è costituito dall'insieme vuoto, mentre l'elemento neutro per la congiunzione, o 1 dell'insieme, è l'insieme  $S$  stesso.

Disponendo degli operatori Booleani e agendo sugli elementi di  $\mathcal{B}$ , si potrebbero ricavare infinite relazioni notevoli, dette **leggi Booleane** o **teoremi**, alcune delle quali sono già state viste precedentemente con le tavole di verità; un esempio potrebbe essere il Teorema di De Morgan (§ 2.6). Una legge Booleana è dunque un'identità tra due termini Booleani, costruiti su un insieme di variabili e sulle costanti 0 e 1 a partire dagli operatori  $\wedge$ ,  $\vee$ ,  $\neg$ . Per verificare una legge è sufficiente procedere con la cosiddetta **induzione perfetta**, che consiste nella sostituzione di tutti i possibili valori per le variabili, verificando che l'uguaglianza sia sempre soddisfatta.

Le leggi Booleane sono ovviamente infinite, ma poiché l'Algebra Booleana è un insieme **finitamente assiomatizzabile**, è possibile descrivere compiutamente l'Algebra Booleana usando solamente un insieme finito di leggi base, che sono gli assiomi precedentemente descritti. Ciò significa che



partendo dagli assiomi si possono poi ricavare tutte le leggi (o teoremi) dell'Algebra Booleana, combinando gli assiomi e i teoremi che ne derivano in tutti i modi possibili.

Gli assiomi non sono soggetti a verifica, nel senso che sono **verità primitive** che costituiscono il punto di partenza di una teoria matematica. Essi devono inoltre essere **non contraddittori** tra loro e possibilmente **indipendenti**. Quest'ultima caratteristica è in qualche modo legata al problema di determinare se, assegnato un insieme di assiomi, esso sia il minimo possibile o se ne esista uno più piccolo.

Per esempio, si può osservare che l'operatore disgiunzione  $\cdot$ , usato come operatore base nel quadro assiomatico appena analizzato, può essere definito in funzione degli altri due operatori  $+$  e  $\neg$ , come già verificato nel teorema di De Morgan (§ 2.6). Ecco allora che si potrebbe rimuovere tale operatore, pervenendo a un quadro assiomatico più sintetico. Tale problema fu affrontato da Edward V. Huntington, che nel 1933 riuscì a formulare un insieme minimo di assiomi tra loro indipendenti, che partono dai soli operatori  $+$  e  $\neg$  e che prevedono, oltre alla commutatività e all'associatività, anche la cosiddetta equazione di Huntington:

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

## 2.10 Teoremi principali dell'Algebra Booleana

A partire dagli assiomi sopra esposti, si potrebbero ricavare tutti i possibili teoremi dell'Algebra Booleana; tra questi ve ne sono alcuni molto semplici e utili nella manipolazione delle formule, di seguito elencati, che sono validi sia in forma base che in forma duale:

| Teorema                      | Base (a)                                                                          | Duale (d)                                                                           |
|------------------------------|-----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| <b>T1 - Idempotenza</b>      | $x + x = x$                                                                       | $x \cdot x = x$                                                                     |
| <b>T2 - Nullifico</b>        | $x + 1 = 1$                                                                       | $x \cdot 0 = 0$                                                                     |
| <b>T3 - Doppia negazione</b> | $\overline{\overline{x}} = x$                                                     |                                                                                     |
| <b>T4 - Assorbimento 1</b>   | $x + x \cdot y = x$                                                               | $x \cdot (x + y) = x$                                                               |
| <b>T5 - Assorbimento 2</b>   | $x + \overline{x} \cdot y = x + y$                                                | $x \cdot (\overline{x} + y) = x \cdot y$                                            |
| <b>T6 - Assorbimento 3</b>   | $x \cdot y + \overline{x} \cdot z + y \cdot z = x \cdot y + \overline{x} \cdot z$ | $(x + y) \cdot (\overline{x} + z) \cdot (y + z) = (x + y) \cdot (\overline{x} + z)$ |
| <b>T7 - De Morgan</b>        | $\overline{x + y} = \overline{x} \cdot \overline{y}$                              | $\overline{x \cdot y} = \overline{x} + \overline{y}$                                |
| <b>T8</b>                    | $x \cdot (x + y + z) = x$                                                         | $x + (x \cdot y \cdot z) = x$                                                       |
| <b>T9</b>                    | $(x + y) \cdot (\overline{x} + y) = y$                                            | $(x \cdot y) + (\overline{x} \cdot y) = y$                                          |
| <b>T10</b>                   | $(x + y) \cdot (\overline{x} + z) = x \cdot z + \overline{x} \cdot y$             | $(x \cdot y) + (\overline{x} \cdot z) = (x + z) \cdot (\overline{x} + y)$           |

Tabella 9: Teoremi principali dell'Algebra Booleana

La dimostrazione di uno qualunque di questi teoremi può essere fatta per induzione perfetta, che è la via più lunga e tediosa, oppure sfruttando gli assiomi e/o i teoremi già dimostrati.

Si considerino alcune dimostrazioni dei teoremi più importanti, giacché i primi (da T1 a T3) sono banali, poiché è sufficiente operare una verifica diretta tramite **induzione perfetta**.

Si analizzi, invece, la dimostrazione di T4, T5 e T6, ponendo tra parentesi, subito dopo il segno di

uguaglianza, l'assioma o il teorema usato. Per la dimostrazione di tali teoremi, è possibile naturalmente ricorrere al metodo dell'**induzione perfetta**, ma quando i casi si complicano è necessario ricorrere a dei teoremi precedentemente dimostrati:

| Teorema             |       | Dimostrazione                                                                                                                                                                                                                                                                                                |
|---------------------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| T4 - Assorbimento 1 | Base  | $x + xy \stackrel{(A3)}{=} x \cdot 1 + xy \stackrel{(A6)}{=} x \cdot (1 + y) \stackrel{(T2)}{=} x$                                                                                                                                                                                                           |
|                     | Duale | $x(x + y) \stackrel{(A6)}{=} xx + xy \stackrel{(T1)}{=} x + xy \stackrel{(T4.b)}{=} x$                                                                                                                                                                                                                       |
| T5 - Assorbimento 2 | Base  | $x + \bar{x}y \stackrel{(T2)}{=} x(1 + y) + \bar{x}y \stackrel{(A6)}{=} x + xy + \bar{x}y \stackrel{(A6)}{=} x + (x + \bar{x})y \stackrel{(A7)}{=} x + y$                                                                                                                                                    |
|                     | Duale | $x(\bar{x} + y) \stackrel{(A6)}{=} x\bar{x} + xy \stackrel{(A7)}{=} xy$                                                                                                                                                                                                                                      |
| T6 - Assorbimento 3 | Base  | $xy + \bar{x}z + yz \stackrel{(A7)}{=} xy + \bar{x}z + yz(x + \bar{x}) \stackrel{(A6)}{=} xy + \bar{x}z + xyz + \bar{x}yz \stackrel{(A6)}{=} xy(z + 1) + \bar{x}z(y + 1) \stackrel{(T2)}{=} xy + \bar{x}z$                                                                                                   |
|                     | Duale | $(x + y)(\bar{x} + z)(y + z) \stackrel{(A7)}{=} (x + y)(\bar{x} + z)(y + z + x\bar{x}) \stackrel{(A6)}{=} (x + y)(\bar{x} + z)(y + z + x)$<br>$(x + y)(\bar{x} + z)(y + z + x)(y + z + \bar{x}) \stackrel{(A4)}{=} (x + y)(x + y + z)(\bar{x} + z)(\bar{x} + z + y) \stackrel{(T4)}{=} (x + y)(\bar{x} + z)$ |

**Osservazione:** Il teorema dell'idempotenza viene utilizzato frequentemente, non tanto nel senso da sinistra verso destra, ma più che altro per clonare una variabile ed ottenerne una identica.

**Dimostrazione del teorema di De Morgan:** La verifica del **T7 - Teorema di De Morgan** basata l'induzione perfetta, e già stata effettuata nella dimostrazione nella Tabella 5, ma si coglie l'occasione per rinnovare l'importanza fondamentale di questo teorema, per le applicazioni e per la semplificazione delle espressioni complesse.

## 2.11 Logica duale

Se si osserva l'elenco dei teoremi di Tabella 9, è possibile notare che le proposizioni della colonna di destra possono essere ottenute dalla colonna centrale semplicemente scambiando “0” con “1” e “+” con “.” e viceversa. Questa è una manifestazione del cosiddetto **Principio di dualità**.

Per comprenderlo si considerino le seguenti tabelle di verità, limitatamente ai valori di AND e OR, e si operi una sostituzione “0” con “1” e “+” con “.”. Si ottiene

| $x$ | $y$ | $x + y$ | $x \cdot y$ | $x$ | $y$ | $x \cdot y$ | $x + y$ |
|-----|-----|---------|-------------|-----|-----|-------------|---------|
| 0   | 0   | 0       | 0           | 1   | 1   | 1           | 1       |
| 0   | 1   | 1       | 0           | 1   | 0   | 0           | 1       |
| 1   | 0   | 1       | 0           | 0   | 1   | 0           | 1       |
| 1   | 1   | 1       | 1           | 0   | 0   | 0           | 0       |

Tabella 10: Principio di dualità

Si può osservare che la tavola di verità a che ne deriva conserva la propria validità, sia pur con una permutazione delle righe, del tutto ininfluente. Il principio di dualità consente allora di trasformare in modo duale le espressioni Booleane, realizzando nuove espressioni che continuano a valere. Si osservi tuttavia che i valori delle espressioni così ottenute sono in generale diversi da quelle di partenza.

Si consideri, a tal proposito, il teorema T10: l'espressione Booleana  $(x + y) \cdot (\bar{x} + z)$  ha come duale la  $x \cdot y + \bar{x} \cdot z$ .

| 1   | 2   | 3   | 4                             | 5                             | 6   | 7   | 8   | 9                             |
|-----|-----|-----|-------------------------------|-------------------------------|-----|-----|-----|-------------------------------|
| $x$ | $y$ | $z$ | $(x + y) \cdot (\bar{x} + z)$ | $x \cdot y + \bar{x} \cdot z$ | $x$ | $y$ | $z$ | $x \cdot y + \bar{x} \cdot z$ |
| 0   | 0   | 0   | 0                             | 0                             | 0   | 0   | 0   | 0                             |
| 0   | 0   | 1   | 0                             | 1                             | 0   | 0   | 1   | 1                             |
| 0   | 1   | 0   | 1                             | 0                             | 0   | 1   | 0   | 0                             |
| 0   | 1   | 1   | 1                             | 1                             | 0   | 1   | 1   | 1                             |
| 1   | 0   | 0   | 0                             | 0                             | 1   | 0   | 0   | 0                             |
| 1   | 0   | 1   | 1                             | 0                             | 1   | 0   | 1   | 0                             |
| 1   | 1   | 0   | 0                             | 1                             | 1   | 1   | 0   | 1                             |
| 1   | 1   | 1   | 1                             | 1                             | 1   | 1   | 1   | 1                             |

Tabella 11: Principio di dualità applicato all'espressione Booleana  $(x + y) \cdot (\bar{x} + z)$ 

Nella Tabella 11 sono riportati, in colonna 4 e 5, i valori di queste due espressioni duali, e come si vede sono diversi. Per verificare la validità del principio di dualità è necessario operare una sostituzione di “0” con “1” e di “+” con “ $\cdot$ ” nelle colonne 1, 2, 3 e nella colonna 4, che riporta i valori dell'espressione; così facendo si ottengono rispettivamente le colonne 6, 7, 8 e 9.

Si può verificare che a ciascuna terna delle colonne duali 6, 7, 8 corrisponde un valore duale di colonna 9 che è esattamente il valore attribuito dall'espressione di colonna 5 alla stessa terna letta nelle colonne primitive 2, 3, 4. Per esempio, a 011 delle colonne 6, 7, 8 corrisponde 1 in colonna 9; questo è lo stesso valore attribuito dalla colonna 5 alla terna 011 delle colonne primitive 1, 2, 3.

**Osservazione:** La logica duale è estremamente importante e vantaggiosa, in quanto se in una espressione è presente il “+” ed ad esso si sostituisce il “ $\cdot$ ”, si ottiene una espressione ancora vera. Tutto ciò trova spiegazione nel fatto che il principio di dualità è un fondamento base della manipolazione booleana, che permette di trasformare proposizioni vere in altre proposizioni vere, pur tenendo presente che le espressioni che si ottengono attraverso lo strumento della dualità sono pur sempre delle espressioni diverse da quelle di partenza.

## 2.12 Variabili, funzioni Booleane e porte logiche

Data una funzione di verità, è possibile associare alla combinazione di  $n$  variabili logiche uno e un solo valore logico. Se dunque  $x_1, x_2, \dots, x_n$  sono  $n$  variabili Booleane (o binarie) che possono assumere l'uno o l'altro dei due valori possibili 0 e 1, si indica con

$$f(x_1, x_2, \dots, x_n)$$

una **generica funzione Booleana** del tipo:

$$f : 2^n \rightarrow 2$$

che a ogni valore della n-upla  $x_1, x_2, \dots, x_n$  associa un valore dell'insieme  $\{0, 1\}$ . La tabella di verità di una simile funzione è rappresentata nella tabella che segue:

| $x_1$ | $x_2$ | ... | $x_{n-1}$ | $x_n$ | $f(x_1, x_2, \dots, x_n)$ |
|-------|-------|-----|-----------|-------|---------------------------|
| 0     | 0     | ... | 0         | 0     | $f(0, 0, \dots, 0, 0)$    |
| 0     | 0     | ... | 0         | 1     | $f(0, 0, \dots, 0, 1)$    |
| ...   | ...   | ... | ...       | ...   | ...                       |
| 1     | 1     | ... | 1         | 0     | $f(1, 1, \dots, 1, 0)$    |
| 1     | 1     | ... | 1         | 1     | $f(1, 1, \dots, 1, 1)$    |

Tabella 12: Tabella di verità di una generica funzione Booleana n-aria

Si osservi che il numero di n-uple binarie dell'insieme  $2^n$  cresce in modo esponenziale. In generale, poiché l'insieme  $2^n$  contiene  $2^n$  n-uple binarie, e a ciascuna di esse si associa un valore della funzione (che può essere 0 o 1), il numero totale di funzioni Booleane n-arie che si possono realizzare risulta essere:

$$2^{2^n}$$

Risulta fondamentale, più specificatamente, studiare in particolare le funzioni Booleane per  $n = 1$  (a una variabile, o **unarie**) e  $n = 2$  (a due variabili, o **di-arie**).

### 2.12.1 Funzioni a una variabile (unarie)

Le funzioni ad una variabile (in cui  $n = 1$ ) che si possono costruire sono in tutto

$$2^{2^1} = 2^2 = 4$$

e sono funzioni del tipo  $y = f(x)$ .

|     | $f_0$ | $f_1$ | $f_2$     | $f_3$ |     | <b>Nulla</b> |     | <b>Unitaria</b> |     | <b>Buffer</b> |     | <b>NOT</b> |
|-----|-------|-------|-----------|-------|-----|--------------|-----|-----------------|-----|---------------|-----|------------|
| $x$ | 0     | $x$   | $\bar{x}$ | 1     | $x$ | $z$          | $x$ | $z$             | $x$ | $z$           | $x$ | $z$        |
| 0   | 0     | 0     | 1         | 1     | 0   | 0            | 0   | 1               | 0   | 0             | 0   | 1          |
| 1   | 0     | 1     | 0         | 1     | 1   | 0            | 1   | 1               | 1   | 1             | 1   | 0          |

Tabella 13: Tutte le possibili  $2^{2^1} = 4$  funzioni Booleane a 1 variabile

1. La prima funzione fornisce sempre 0 in uscita, qualunque sia l'ingresso; è dunque la **funzione nulla**, che si nota con **0**.
2. Un discorso analogo vale per la seconda funzione, che fornisce sempre 1 in uscita, qualunque sia l'ingresso; è dunque la **funzione unitaria**, che si denota con **1**.
3. La terza funzione replica il valore di  $x$  in uscita, cioè  $y = x$  e si tratta dunque della **funzione identità**. Poiché la logica Booleana è connessa, come già anticipato, all'ambito circuitale, per la rappresentazione della funzione identità si usa il termine **Buffer**, e le si attribuisce lo schema circuitale dato da un triangolo:

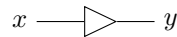


Figura 4: Buffer

Esso corrisponde a un dispositivo che fornisce sull'uscita a destra esattamente ciò che si presenta in ingresso a sinistra.

4. La quarta funzione è la più importante dal punto di vista della logica Booleana, poiché è uno dei connettivi base già analizzati, cioè il NOT. Esso fornisce in uscita la negazione di quanto sta all'ingresso; si può dunque scrivere  $y = \bar{x}$ . Per la sua rappresentazione circuitale si usa il triangolo di prima concatenato con un piccolo cerchio, che in tutta la circuiteria logica ha sempre il significato di una **negazione** o **complementazione**:

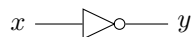


Figura 5: Negazione (o complementazione)

### 2.12.2 Funzioni a due variabili

Le funzioni a due variabili (in cui  $n = 2$ ) che si possono costruire sono

$$2^{2^2} = 2^4 = 16$$

Sono funzioni Booleane di-arie del tipo  $z = f(x, y)$  e sono così importanti da meritare una descrizione anche circuitale.

Di seguito vengono riportate usando il linguaggio dell'Algebra Booleana:

|     |     | $f_0$    | $f_1$   | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$    | $f_7$ | $f_8$        | $f_9$   | $f_{10}$ | $f_{11}$ | $f_{12}$ | $f_{13}$ | $f_{14}$ | $f_{15}$ |
|-----|-----|----------|---------|-------|-------|-------|-------|----------|-------|--------------|---------|----------|----------|----------|----------|----------|----------|
|     |     | AND      |         |       |       | XOR   |       |          |       | OR           | NOR     | XNOR     | NAND     |          |          |          |          |
| $A$ | $B$ | <b>0</b> | $\cdot$ |       |       |       |       | $\oplus$ | $+$   | $\downarrow$ | $\odot$ |          |          |          |          | $ $      | <b>1</b> |
| $F$ | $F$ | $F$      | $F$     | $F$   | $F$   | $F$   | $F$   | $F$      | $F$   | $V$          | $V$     | $V$      | $V$      | $V$      | $V$      | $V$      | $V$      |
| $F$ | $V$ | $F$      | $F$     | $F$   | $F$   | $V$   | $V$   | $V$      | $V$   | $F$          | $F$     | $F$      | $F$      | $V$      | $V$      | $V$      | $V$      |
| $V$ | $F$ | $F$      | $F$     | $V$   | $V$   | $F$   | $F$   | $V$      | $V$   | $F$          | $F$     | $V$      | $V$      | $F$      | $F$      | $V$      | $V$      |
| $V$ | $V$ | $F$      | $V$     | $F$   | $V$   | $F$   | $V$   | $F$      | $V$   | $F$          | $V$     | $F$      | $V$      | $F$      | $V$      | $F$      | $V$      |

Tabella 14: Tutte le possibili  $2^{2^2} = 16$  funzioni Booleane a due variabili

Tra tutte queste funzioni è noto che alcune sono particolarmente significative, precisamente le sei già evidenziate in Tabella 14, che sono rispettivamente AND, OR, XOR, NAND, NOR e XNOR; nella tabella seguente esse sono espresse, assieme alle rimanenti, mediante i simboli  $\cdot, +, \oplus, \downarrow, \odot$  e  $|$  che si usano tradizionalmente nell'ambito dell'Algebra Booleana; si noti che  $+, \cdot$  e  $\odot$  stanno rispettivamente per  $\cup, \cap$  e  $\equiv$ , visti nella precedente Tabella 2 parlando di connettivi.

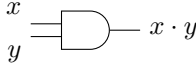
|                         |     |                        |      |
|-------------------------|-----|------------------------|------|
| $f_0 = \mathbf{0}$      | AND | $f_8 = x \downarrow y$ | NOR  |
| $f_1 = x \cdot y$       |     | $f_9 = x \odot y$      | XNOR |
| $f_2 = x \cdot \bar{y}$ |     | $f_{10} = \bar{y}$     |      |
| $f_3 = x$               |     | $f_{11} = x + \bar{y}$ |      |
| $f_4 = \bar{x} \cdot y$ | XOR | $f_{12} = \bar{x}$     |      |
| $f_5 = y$               |     | $f_{13} = \bar{x} + y$ |      |
| $f_6 = x \oplus y$      | OR  | $f_{14} = x   y$       | NAND |
| $f_7 = x + y$           |     | $f_{15} = \mathbf{1}$  |      |

Tabella 15: Le 16 funzione Booleane espresse mediante  $\cdot, +, \oplus, \odot, \downarrow, |$  e complementazione

Come anticipato, data l'estrema importanza, in ambito circuitale, delle funzioni AND, OR, XOR, NAND, NOR e XNOR, dette anche operatori Booleani, vengono analizzate singolarmente di seguito:

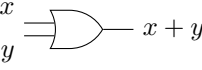
#### • Porta AND

È detta anche **prodotto logico**. La porta AND restituisce 0 in uscita se anche uno solo dei due valori d'ingresso è pari a 0; restituisce 1 solo quando entrambi gli ingressi sono a 1. Il simbolo circuitale è:

| $x$ | $y$ |                                                                                   | $x \cdot y$ |
|-----|-----|-----------------------------------------------------------------------------------|-------------|
| 0   | 0   |  | 0           |
| 0   | 1   |                                                                                   | 0           |
| 1   | 0   |                                                                                   | 0           |
| 1   | 1   |                                                                                   | 1           |

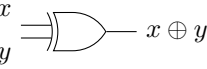
- **Porta OR**

È detta anche **somma logica**. La porta OR restituisce 1 in uscita se almeno uno dei due valori d'ingresso è pari a 1; restituisce 0 solo quando entrambi gli ingressi sono a 0. Il simbolo circuitale è:

| $x$ | $y$ |                                                                                   | $x + y$ |
|-----|-----|-----------------------------------------------------------------------------------|---------|
| 0   | 0   |  | 0       |
| 0   | 1   |                                                                                   | 1       |
| 1   | 0   |                                                                                   | 1       |
| 1   | 1   |                                                                                   | 1       |

- **Porta XOR**

È l'OR esclusivo. La porta XOR restituisce 1 in uscita se o uno o l'altro dei due valori d'ingresso è pari a 1, ma non entrambi; restituisce 0 quando entrambi gli ingressi sono a 0 o a 1. Il simbolo circuitale è

| $x$ | $y$ |                                                                                     | $x \oplus y$ |
|-----|-----|-------------------------------------------------------------------------------------|--------------|
| 0   | 0   |  | 0            |
| 0   | 1   |                                                                                     | 1            |
| 1   | 0   |                                                                                     | 1            |
| 1   | 1   |                                                                                     | 0            |

### 2.13 Riassunto

L'algebra booleana è l'apparato teorico che costituisce le fondamenta della logica binaria. Attraverso il teorema di De Morgan, in particolare, l'insieme minimo dei connettivi in grado di esprimere le 16 espressioni booleane può essere ristretto a AND, NOT oppure a OR, NOT.

Non solo, ma impiegando i connettivi universali NOR e NAND è possibile impiegare un solo connettivo per costruire ogni funzione logica.

Gli assiomi che si pongono alla base dell'algebra booleana sono verità assolute, incontestabili e non dimostrabili e costituiscono le fondamenta della teoria assiomatica quale quella booleana. L'algebra booleana, tuttavia, non è solo quella delle porte logiche, ma anche quella delle operazioni insiemistiche, secondo **Kantor**; esse, infatti, rispettano tutti i dettami dell'algebra booleana.

A partire dagli assiomi fondamentali si possono, poi, formulare dei teoremi che permettono la semplificazione di espressioni logiche, tramite il **principio di inferenza**. Il **principio di dualità**, invece, permette di trasformare una proposizione vera con la congiunzione in un'altra proposizione vera con la disgiunzione. Naturalmente, si ottiene una seconda espressione sempre vera, ma che risulta comunque differente rispetto alla funzione di partenza.

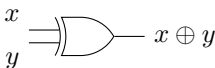
Le funzioni booleane vanno da  $2^n \rightarrow 2$ , ovvero ad ogni n-upla viene associato uno dei 2 possibili valori (VERO o FALSO).

Con  $n = 1$ , si possono avere solamente  $2^2 = 4$  funzioni unarie del tipo  $y = f(x)$ , mentre con  $n = 2$  ci sono  $2^{2^2} = 16$  funzioni booleane di-arie del tipo  $z = f(x, y)$ . Tali funzioni sono le 16 funzioni logiche precedentemente esposte, ma che presentano dei simboli circuitali differenti rispetto ai connettivi analizzati.

Pertanto, dopo aver discusso ampiamente le due porte principali dell'Algebra Booleana (AND e OR), è doveroso rivolgere un'importante attenzione alle porte solo in apparenza secondarie, di seguito esposte:

- **Porta XOR**

È l'OR esclusivo. La porta XOR restituisce 1 in uscita se uno o l'altro dei due valori d'ingresso è pari a 1, ma non entrambi; restituisce 0 quando entrambi gli ingressi sono a 0 o a 1. Il simbolo circuitale è

| $x$ | $y$ |                                                                                     | $x \oplus y$ |
|-----|-----|-------------------------------------------------------------------------------------|--------------|
| 0   | 0   |  | 0            |
| 0   | 1   |                                                                                     | 1            |
| 1   | 0   |                                                                                     | 1            |
| 1   | 1   |                                                                                     | 0            |

Si osservi che una porta XOR è in grado di realizzare la somma binaria modulo 2 e che costituisce un **rilevatore di differenza** tra i due valori d'ingresso.

Inoltre, si osserva che:

$$x \oplus y = x\bar{y} + \bar{x}y$$

che corrisponde a un OR di due AND. Mediante manipolazione algebrica si ottiene anche la seguente espressione:

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

che corrisponde a un AND di due OR. Si possono pertanto realizzare due circuiti del tutto equivalenti, visibili in Figura 6:

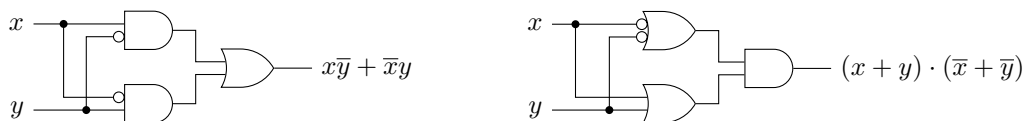
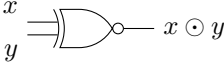


Figura 6: La funzione XOR realizzata mediante porte AND e OR

- **Porta XNOR**

È la negazione della porta XOR. La porta XNOR restituisce 1 in uscita solo quando entrambe le variabili d'ingresso hanno lo stesso valore, o entrambe a 0 oppure entrambe a 1. Il simbolo circuitale si ottiene concatenando la porta XOR con la porta NOT:

| $x$ | $y$ |                                                                                   | $x \odot y$ |
|-----|-----|-----------------------------------------------------------------------------------|-------------|
| 0   | 0   |  | 1           |
| 0   | 1   |                                                                                   | 0           |
| 1   | 0   |                                                                                   | 0           |
| 1   | 1   |                                                                                   | 1           |

Si osservi che una porta XNOR costituisce un **rilevatore di identità** tra i due valori d'ingresso. Per trovare una rappresentazione circuitale mediante porte AND e OR bisogna partire dal fatto che si tratta di uno XOR negato, traendone le conseguenze sempre usando il Teorema di De Morgan (§ 2.6):

$$x \odot y = \overline{x \oplus y} = \overline{xy + x\bar{y}} = \overline{xy} \cdot \overline{x\bar{y}} = (x + \bar{y}) \cdot (\bar{x} + y) = x\bar{x} + xy + \bar{x}\bar{y} + \bar{y}y = xy + \bar{x}\bar{y}$$

Nella prima parte dello svolgimento si ottiene  $(x + \bar{y}) \cdot (\bar{x} + y)$ , e quindi l'AND di due OR, mentre nella seconda parte si ottiene  $xy + \bar{x}\bar{y}$  le quali portano alla realizzazione circuitale seguente

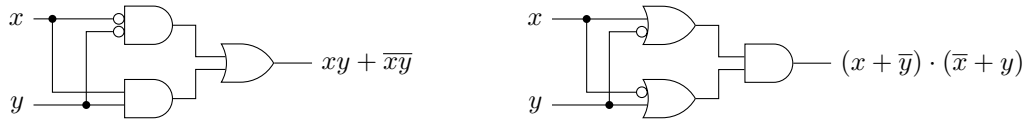
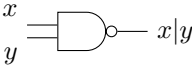


Figura 7: La funzione XNOR realizzata mediante porte AND e OR

Dal punto di vista ingegneristico, non vi è una sostanziale differenza tra le due, dal momento che il numero di operazioni binarie e di negazioni sono esattamente identici.

- **Porta NAND**

È la negazione della porta AND, rappresentata dal seguente simbolo circuitale:

| $x$ | $y$ |                                                                                     | $x y$ |
|-----|-----|-------------------------------------------------------------------------------------|-------|
| 0   | 0   |  | 1     |
| 0   | 1   |                                                                                     | 1     |
| 1   | 0   |                                                                                     | 1     |
| 1   | 1   |                                                                                     | 0     |

Com'è noto, la porta NAND è un **operatore universale**, nel senso che da sola consente di costruire una qualunque tra le 16 possibili funzioni. Per declinare le relazioni succitate nei termini del linguaggio dell'Algebra Booleana è sufficiente usare gli assiomi e i teoremi visti in precedenza:

$$\begin{aligned}\bar{x} &= \overline{x \cdot x} = x|x \\ x + y &= \overline{\bar{x} \cdot \bar{y}} = \bar{x}|\bar{y} \\ x \cdot y &= \overline{\bar{x} \cdot \bar{y}} = \overline{x|y}\end{aligned}$$

Da cui derivano le seguenti rappresentazioni circuitali:



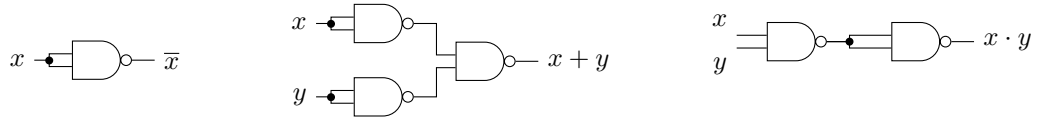
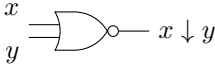


Tabella 16: Funzioni NOT, OR e AND realizzazate mediante la porta universale NAND

### • Porta NOR

È la negazione della porta OR, rappresentata dal seguente simbolo circuitale:

| $x$ | $y$ |                                                                                   | $x \downarrow y$ |
|-----|-----|-----------------------------------------------------------------------------------|------------------|
| 0   | 0   |  | 1                |
| 0   | 1   |                                                                                   | 0                |
| 1   | 0   |                                                                                   | 0                |
| 1   | 1   |                                                                                   | 0                |

Com'è noto, anche la porta OR è un **operatore universale**, nel senso che da sola consente di costruire una qualunque tra le 16 possibili funzioni. Per declinare le relazioni succitate nei termini del linguaggio dell'Algebra Booleana è sufficiente usare gli assiomi e i teoremi visti in precedenza:

$$\begin{aligned}\bar{x} &= \overline{x + x} = x \downarrow x \\ x + y &= \overline{\overline{x + y}} = \overline{x \downarrow y} \\ x \cdot y &= \overline{\overline{x} + \overline{y}} = \overline{x \downarrow \overline{y}}\end{aligned}$$

Da cui derivano le seguenti rappresentazioni circuitali:

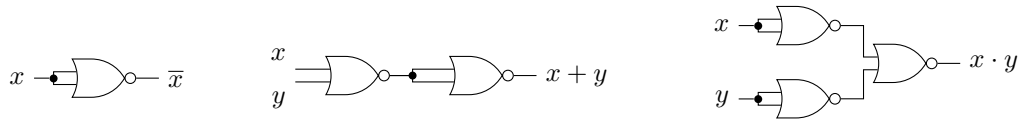


Tabella 17: Funzioni NOT, OR e AND realizzazate mediante la porta universale NOR

E questo è un risultato molto importante dal punto di vista tecnico industriale, dal momento che potrebbero comportare significativi vantaggi dal punto di vista economico-produttivo.

#### 2.13.1 Funzioni a tre variabili

Finora sono state trattate solamente porte associate a funzioni a due variabili. La generalizzazione a  $n$  variabili ha pieno senso solo per le porte AND e OR; si tratterà per esteso il caso con  $n = 3$ , poiché gli altri sono la banale generalizzazione di questo. Le tavole di verità delle funzioni AND e OR a 3 variabili sono riportate nella successiva Tabella 18:

| $x$ | $y$ | $z$ | AND | $x$ | $y$ | $z$ | OR |
|-----|-----|-----|-----|-----|-----|-----|----|
| 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0  |
| 0   | 0   | 1   | 1   | 0   | 0   | 1   | 1  |
| 0   | 1   | 0   | 0   | 0   | 1   | 0   | 1  |
| 1   | 0   | 0   | 1   | 1   | 0   | 0   | 1  |
| 1   | 0   | 1   | 0   | 1   | 0   | 1   | 1  |
| 1   | 1   | 0   | 0   | 1   | 1   | 0   | 1  |
| 1   | 1   | 1   | 0   | 1   | 1   | 1   | 1  |
| 1   | 1   | 1   | 1   | 1   | 1   | 1   | 1  |

Tabella 18: Funzioni AND e OR a 3 variabili

Naturalmente, la funzione AND vale 1 solo quando tutte le variabili in ingresso hanno valore 1; la funzione OR vale 0 solo quando tutte le variabili in ingresso hanno valore 0.

## 2.14 Realizzazione circuitale delle porte logiche

Le funzioni logiche di base, a 1 e 2 variabili, possono essere facilmente realizzate impiegando i transistor. Tuttavia, in fase di realizzazione è necessario tenere conto di alcuni fondamentali obiettivi:

1. Minimizzazione della potenza dissipata. Infatti, la deriva termica dei transistor rappresenta un esito deleterio per il dispositivo, in quanto verso i 200°C il cristallino di silicio si fonde e il transistor si brucia.
2. Minimizzazione dei tempi di risposta. Infatti, la variazione di tensione dei transistor determina la generazione di un impulso che deve transitare per il circuito e deve essere validato dalle porte logiche, anche in cascata. Non è ammissibile che vi siano tempi di risposta differenti in quanto la validazione delle variabili logiche ne potrebbe risentire.
3. Minimizzazione dei costi
4. Aumento progressivo della densità di integrazione sui relativi circuiti integrati.

Nel tempo si sono succedute numerose **famiglie logiche**, che a seconda del periodo in cui sono state introdotte e dello sviluppo tecnologico corrente hanno privilegiato l'uso di uno o dell'altro dei vari dispositivi a semiconduttore per la realizzazione delle funzioni di base, quali diodi, transistor, CMOS, ecc...

Anche se in pratica sono molte di più, le principali famiglie logiche sono le seguenti (in ordine cronologico):

- Resistor-Transistor Logic (RTL) è una classe di porte logiche costruite usando resistenze nella rete d'ingresso e transistori bipolari a giunzione (BJT) come dispositivi di commutazione. La RTL è stata la prima classe di circuito logico digitale basata sull'impiego dei transistor. Le prime porte RTL furono costruite con elementi discreti, ma nel 1961 divenne la prima famiglia logica a esser realizzata su un circuito integrato monolitico. Circuiti integrati RTL sono stati utilizzati nel computer Apollo Guidance, il cui progetto risale al 1961 con un primo volo fatto nel 1966.
- Diode-Transistor Logic (DTL) è la classe di porte logiche che precede la grande famiglia TTL. Si chiama così perché la funzione logica viene eseguita da una rete di diodi, mentre la funzione di inversione-amplificazione viene eseguita da un transistor.
- Transistor-Transistor Logic (TTL) è senza dubbio la classe più nota e famosa, poiché ha avuto un larghissimo impiego. Fa uso di transistori bipolari a giunzione (BJT) e resistenze. Si chiama logica transistor-transistor perché il transistor svolge sia la funzione logica che la funzione di inversione-amplificazione. I circuiti integrati basati sulla famiglia TTL sono stati

ampiamente usati in applicazioni quali computer, controlli industriali, apparecchiature di prova, strumentazione elettronica, elettronica di consumo, sintetizzatori e molto altro. Dopo la loro introduzione come circuito integrato a opera di Sylvania nel 1963, i circuiti integrati TTL sono stati prodotti da diverse aziende di semiconduttori. La serie 7400 (chiamato anche 74xx) della Texas Instruments è diventata particolarmente popolare. I produttori di porte basati sulla tecnologia TTL hanno offerto una vasta gamma di porte logiche, flip-flop, contatori, multiplexer e altri circuiti.

- Complementary Metal-Oxide-Semiconductor (CMOS) è la tecnologia più recente per la costruzione di porte logiche inserite in circuiti integrati. La tecnologia CMOS viene usata in microprocessori, microcontrollori, nella RAM statica e in molti altri circuiti logici digitali. È la tecnologia più raffinata, che consente un abbattimento dei consumi e dei tempi di risposta, congiuntamente alla possibilità di realizzare un'altissima densità di integrazione. I transistor CMOS sono chiamati transistor MOS-FET a effetto di campo (dall'inglese FET - Field Effect Transistor), richiedono molta meno potenza e non dissipano potenza in ingresso, sono molto più piccoli e garantiscono un minore tempo di risposta e sono diversi dai transistor a giunzione (VJT).

Si consideri la realizzazione tramite transistor delle principali porte logiche.

- **Porta NOT**

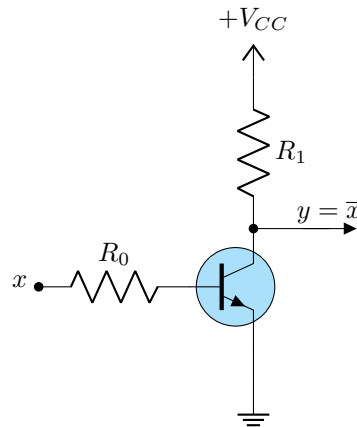


Figura 8: Porta NOT realizzata tramite transistor

Nella porta NOT bisogna realizzare un'inversione del valore logico. Questa funzione si realizza mediante un singolo transistor, nel quale la variabile d'ingresso va ad alimentare la base, mentre quella di uscita si ricava sul collettore del transistor. Se si associa la costante 1 a un livello alto di tensione, come la tensione di alimentazione  $V_{CC}$  di Figura 8, e la costante 0 a un livello basso, cioè la tensione di massa pari a 0V, il funzionamento dell'invertitore è il seguente: quando la variabile  $x$  assume valore logico 1, e cioè viene portata a tensione  $V_{CC}$ , il transistor si polarizza direttamente ed entra in piena conduzione. Tale condizione corrisponde alla saturazione del transistor; in questa configurazione la tensione  $V_{CE}$  crolla idealmente a 0V, portando la variabile  $y$  in uscita nello stato logico 0.

Quando, viceversa, la variabile  $x$  assume valore logico 0, e cioè viene portata a tensione 0V (connessione a massa), il transistor blocca la conduzione e la  $I_C$  diventa (praticamente) nulla. Tale condizione corrisponde all'interdizione del transistor; in questa configurazione, pertanto, la tensione  $V_{CE}$  diventa pari alla  $V_{CC}$ , quindi viene portato all'1 logico il valore dell'uscita. Infatti, con corrente in ingresso, il transistor va in saturazione, per cui la tensione è nulla; se, invece, la corrente in ingresso è nulla, il transistor va in interdizione, per cui la tensione è massima.

La resistenza in ingresso deve essere ovviamente commisurata per non bruciare il circuito, così come è fondamentale il ruolo della **resistenza di carico** che permette di separare la tensione di alimentazione e avere una corrente variabile.

La tensione di saturazione  $V_{SAT}$  non è mai nulla, nella realtà, ma se la corrente in ingresso

è significativamente maggiore di 0, allora in uscita si avrà il valore logico 0, a causa di un **cortocircuito** che si viene a determinare.

Se, invece, la corrente in ingresso è nulla, allora il transistor va in **interdizione**, per cui la corrente in uscita viene **interdetta**, pertanto la tensione in uscita sarà pari a quella di alimentazione. Pertanto, l'uscita corrisponde al valore 1 logico.

In questo modo si sta usando il transistor in modo digitale, come se fosse un vero interruttore, in quanto si opera ai suoi valori estremi, 0 e 1; un transistor che opera, invece, in modo analogico, viene usato come amplificatore, e può assumere anche valori intermedi.

Se si utilizzasse un transistor CMOS, per esempio, il funzionamento sarebbe esattamente analogico, solo che viene comandato in tensione, con significativi vantaggi in termini di **dissipazione di potenza**. Inoltre, i CMOS possono essere anche realizzati in modo molto più microscopico.

### • Porta AND

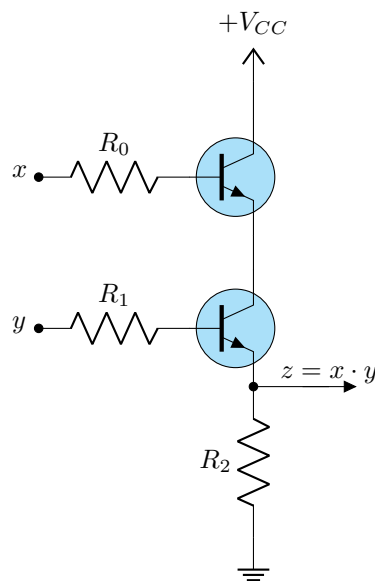


Figura 9: Porta AND realizzata tramite transistor

Nella porta AND bisogna fare in modo che se anche uno solo degli ingressi è a 0, l'uscita resti a 0. Il circuito di Figura 9 realizza questa condizione. Poichè l'uscita a 1 significa livello alto di tensione, per realizzarla bisogna che entrambi i due transistor associati ai due ingressi, che sono posti in serie, siano in piena conduzione. Per realizzare ciò è necessario portare le variabili  $x$  e  $y$  d'ingresso a  $V_{CC}$ .

Se anche uno solo dei due ingressi rimane a livello basso, uno dei due transistor va in interdizione e la tensione di uscita resta a livello basso, il che corrisponde a 0 logico.

Se in ingresso  $x$  e  $y$  sono a valore logico 1, allora entrambi i transistor vanno in saturazione, pertanto diventano dei cortocircuiti, e lasciano passare la tensione di alimentazione, quindi l'uscita avrà valore 1.

Se almeno uno dei due transistor va in interdizione, allora si forma un circuito aperto e quindi l'uscita ha valore 0.

- Porta OR

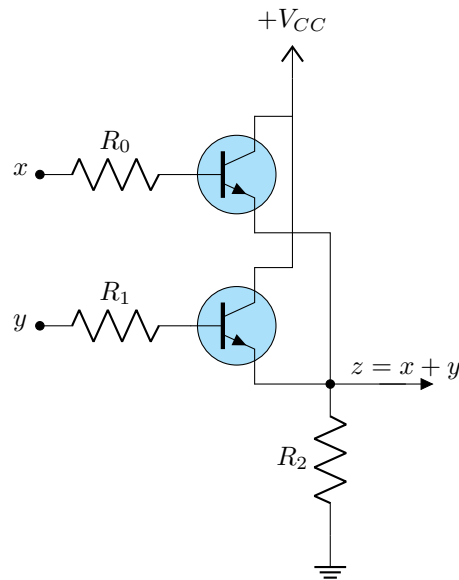


Figura 10: Porta OR realizzata tramite transistor

Nella porta OR bisogna fare in modo che se anche uno solo degli ingressi è a 1, l'uscita resti a 1. Il circuito di Figura 10 realizza questa condizione. Poiché l'uscita a 1 significa livello alto di tensione, per realizzarla basta che anche uno solo dei due transistor associati ai due ingressi, che sono posti in parallelo, sia in piena conduzione. Per realizzare ciò è sufficiente portare o l'ingresso  $x$  o l'ingresso  $y$  al valore  $V_{CC}$ . Solo se entrambi gli ingressi rimangono a livello basso i transistor sono in interdizione e la tensione di uscita resta a livello basso, il che corrisponde a 0 logico.

Effettivamente, essendo i due transistor posti in **parallelo**, a differenza della porta AND, è sufficiente che almeno uno dei due vada in saturazione perché l'uscita sia a 1.

Inoltre, solamente quando entrambi i transistor sono interdetti, ovvero sono a valore 0, allora anche in uscita sarà valore 1.

- **Porta NAND**

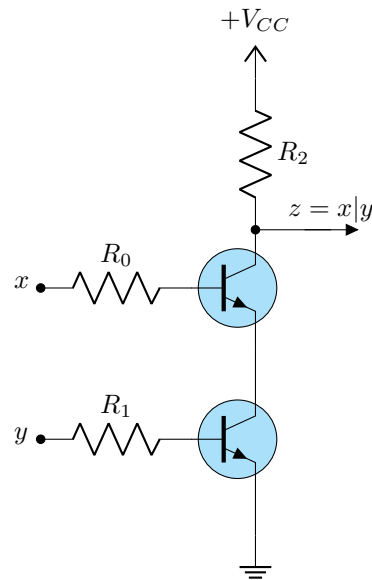


Figura 11: Porta NAND realizzata tramite transistor

La porta NAND viene realizzata a partire dal circuito della porta AND, spostando semplicemente la resistenza  $R_2$  di carico dall'emettitore del secondo transistor al collettore del primo. In questo modo, quando entrambi i transistor sono in piena conduzione grazie al fatto che entrambi gli ingressi sono a 1, l'uscita va a 0; e questa rimane l'unica condizione per la quale si ha uscita bassa. Infatti se anche uno solo degli ingressi va a 0, uno dei due transistor va in interdizione e l'uscita resta a 1.

- **Porta NOR**

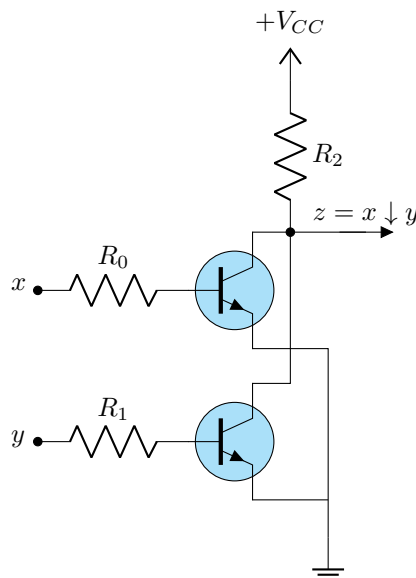


Figura 12: Porta NOR realizzata tramite transistor

Anche in questo caso si parte dalla porta negata, che è la OR, spostando semplicemente la resistenza  $R_2$  di carico dall'emettitore del secondo transistor al collettore del primo. In

questo modo, quando entrambi i transistor sono in interdizione grazie al fatto che entrambi gli ingressi sono a 0, l'uscita va a 1; e questa rimane l'unica condizione per la quale si ha uscita alta. Infatti se anche uno solo degli ingressi va a 1, uno dei due transistor va in saturazione e l'uscita collassa a 0.

12 Ottobre 2021

## 2.15 Riassunto

Una funzione a 3 variabili è una funzione che va dall'insieme  $2^3$  a 2, pertanto il numero di funzioni che si possono costruire con  $n$  variabili cresce molto velocemente, in quanto è dato da:

$$2^{2^n}$$

Considerando le funzioni più semplici, ad una sola variabile, si distinguono 4 sole possibili funzioni, quella **nulla**, quella **identica**, la funzione **buffer** che trasmette a distanza il valore della variabile  $x$ , mentre la funzione **complementare** trasmette a distanza il valore complementare della variabile  $x$ .

Le funzioni a due variabili sono  $2^{2^2} = 16$  e sono le più comuni e importanti, tali da giustificare una dignità circuitale. In particolare si considerano le porte AND e OR, a cui si aggiungono:

- XOR, la quale può essere espressa come:

$$x \oplus y = (A \cap \bar{B}) \cup (\bar{A} \cap B)$$

che, in termini di logica binaria si esprime anche come

$$x \oplus y = x\bar{y} + \bar{x}y$$

ma anche

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

che, dal punto di vista logico, circuitale ed economico sono perfettamente equivalenti.

- XNOR, ovvero la negazione della porta XOR, la quale può essere espressa come:

$$x \odot y = \overline{x \oplus y} = \overline{x\bar{y} + \bar{x}y} = \overline{x\bar{y}} \cdot \overline{\bar{x}y} = (x + \bar{y}) \cdot (\bar{x} + y)$$

ma anche

$$x \odot y = \overline{(x + y) \cdot (\bar{x} + \bar{y})} = \overline{x + y} + \overline{\bar{x} + \bar{y}} = \bar{x}\bar{y} + xy$$

- NAND, ovvero la negazione della porta AND.
- NOR, ovvero la negazione della porta OR.

Queste ultime due funzioni logiche sono definite connettivi universali, in quanto da soli sono in grado di costruire tutte e 16 le possibili funzioni logiche a 2 variabili.

**Osservazione:** Il principio di funzionamento della porta NOT prevede l'utilizzo di un solo transistor, il quale, quando viene polarizzato direttamente con una corrente (opportunamente commisurata alle caratteristiche tecniche del transistor stesso) va in **saturazione** e la tensione  $V_{CE}$ , ovvero la differenza di potenziale tra collettore ed emettitore, in uscita collassa al valore 0.

Se il transistor, invece, non viene polarizzato, allora esso va in **interdizione** e la tensione sul collettore si trova allo stesso valore della tensione di alimentazione ( $V_{CC}$ ), per cui la tensione di uscita ha valore 1.

**Osservazione:** Il principio di funzionamento della porta AND si prevede l'utilizzo di due transistor a giunzione posti in serie, o anche di tipo MOS, o MOS FET, con un principio di funzionamento analogo. Solamente quando entrambi i transistor sono in piena conduzione, ossia in saturazione, allora la tensione in uscita assume il valore 1 logico; in tutti gli altri casi, quando anche un solo transistor viene interdetto, a causa di una tensione di alimentazione pressoché nulla, la tensione in uscita assume il valore 0 logico.

**Osservazione:** Il principio di funzionamento della porta OR prevede l'utilizzo di due transistor posti in parallelo, pertanto solamente quando ambedue i transistor sono in interdizione allora la tensione in uscita avrà valore basso.

Altrimenti, se almeno uno dei due è in **saturazione**, allora la tensione in uscita sarà a valore alto.



## 2.16 Forme canoniche: *minterm*

L'obiettivo di questa sezione si traduce nel formulare una struttura logica tale da poter semplificare nei minimi termini una funzione logica, riducendo il numero di connettivi (e porte) logiche necessari per la realizzazione di una rete logica, con significativi vantaggi in termini di latenza e tempi di risposta.

Le modalità per raggiungere tali fini sono 3: la prima di tipo squisitamente algebrico e dal risultato dubbio (o incerto), in quanto vi possono essere diverse linee di semplificazione che producono espressioni booleane diverse, ma perfettamente equivalenti dal punto di vista algebrico. Dopodiché vi è la **mappa di Karnaugh** e infine il **metodo tabellare di Quine-McCluskey**.

Ricapitolando, l'obiettivo di questa sezione è quello di poter manipolare e gestire qualunque tipo di funzione logica, con qualunque numero di variabili.

Si consideri, per semplicità, la seguente funzione

$$f = (x + z) \cdot (\bar{x} + z)$$

di cui si espone, per chiarezza, la tavola di verità corrispondente nella Tabella 19:

| $x$ | $y$ | $z$ | $f = (x + z) \cdot (\bar{x} + z)$ |
|-----|-----|-----|-----------------------------------|
| 0   | 0   | 0   | 0                                 |
| 0   | 0   | 1   | 0                                 |
| 0   | 1   | 0   | 1                                 |
| 0   | 1   | 1   | 1                                 |
| 1   | 0   | 0   | 0                                 |
| 1   | 0   | 1   | 1                                 |
| 1   | 1   | 0   | 0                                 |
| 1   | 1   | 1   | 1                                 |

Tabella 19: Esempio di funzione booleana

Com'è evidente, tale funzione assumerà il valore in uscita 1 solamente in corrispondenza di specifiche terne di valori delle variabili in ingresso, ovvero 010, 011, 101 e 111 che prendono il nome di **termini minimi**, o **minterm**: ciò consente di costruire la funzione considerata in principio a partire dalla somma logica di funzioni elementari, cioè quelle funzioni che valgono 1 esattamente in corrispondenza di una e una sola configurazione d'ingresso, precisamente quella che entra nella somma logica.

Per poter comporre in una funzione tali **minterm** bisogna sommarli logicamente, in modo tale che la funzione risultante produca il valore logico 1 solamente in una di tali 4 possibili casi (per la funzione oggetto di studio, ovviamente). La funzione di partenza, allora, si può esprimere come somma di prodotti delle variabili d'ingresso, nella forma

$$f = \bar{x}y\bar{z} + \bar{x}yz + x\bar{y}z + xyz$$

ovvero si è costruita la **somma dei termini minimi**, ovvero una **somma logica dei minterm**, ovvero una funzione che vale sempre 0, tranne che per le configurazioni per le quali vale 1. La **somma di minterm** è in grado di rappresentare una qualunque funzione logica. Infatti, la tecnica esposta può essere generalizzata a qualunque funzione  $y = f(x_1, x_2, \dots, x_n)$  di  $n$  variabili, tenuto conto che i termini minimi  $m_i$  sono in tutto tanti quante sono le possibili configurazioni d'ingresso, cioè  $2^n$ . Se si codificano le  $n$ -ple d'ingresso associate a ciascun termine minimo con il corrispondente intero rappresentato in notazione posizionale in base 2, è possibile indicare i termini minimi che compongono la sommatoria usando gli interi compresi tra 0 e  $2^n - 1$ .

Pertanto la generica funzione si rappresenta come segue

$$f(x_1, x_2, \dots, x_n) = \sum_{i=0}^{2^n-1} \mu_i m_i = \sum_{i:\mu_i=1} m_i$$

dove  $\mu_i$  è il valore assunto dalla funzione in corrispondenza del termine minimo  $m_i$  e  $0 \leq i \leq 2^n - 1$ . Nel caso analizzato, si ha  $m_2 = \bar{x}y\bar{z}$ ,  $m_3 = \bar{x}yz$ ,  $m_5 = x\bar{y}z$ ,  $m_7 = xyz$ ,  $\mu_2 = \mu_3 = \mu_5 = \mu_7 = 1$ ,  $\mu_0 = \mu_1 = \mu_4 = \mu_6 = 0$ .

La sommatoria così ottenuta rappresenta la funzione nei termini della cosiddetta **prima forma canonica** (o **somma di prodotti**). Se nella tavola di verità della funzione si pone in evidenza la codifica dei termini minimi si ottiene:

|       |                   | $x$ | $y$ | $z$ | $f$ |         |
|-------|-------------------|-----|-----|-----|-----|---------|
| $m_0$ | $\bar{x}y\bar{z}$ | 0   | 0   | 0   | 0   | $\mu_0$ |
| $m_1$ | $\bar{x}yz$       | 0   | 0   | 1   | 0   | $\mu_1$ |
| $m_2$ | $\bar{x}y\bar{z}$ | 0   | 1   | 0   | 1   | $\mu_2$ |
| $m_3$ | $\bar{x}yz$       | 0   | 1   | 1   | 1   | $\mu_3$ |
| $m_4$ | $x\bar{y}\bar{z}$ | 1   | 0   | 0   | 0   | $\mu_4$ |
| $m_5$ | $x\bar{y}z$       | 1   | 0   | 1   | 1   | $\mu_5$ |
| $m_6$ | $xy\bar{z}$       | 1   | 1   | 0   | 0   | $\mu_6$ |
| $m_7$ | $xyz$             | 1   | 1   | 1   | 1   | $\mu_7$ |

Tabella 20: Codifica dei termini minimi

Per cui si ottiene

$$f(x, y, z) = \sum_{i \in \{2,3,5,7\}} m_i = m_2 + m_3 + m_5 + m_7 = \bar{x}y\bar{z} + \bar{x}yz + x\bar{y}z + xyz$$

poiché 2, 3, 5 e 7 sono le codifiche in base 2 di 010, 011, 101 e 111.

Dal punto di vista circuitale, la somma di *minterm* si rappresenta come segue:

## 2.17 Forme canoniche: *maxterm*

Un discorso del tutto analogo si può formulare procedendo per dualità, e realizzando un **prodotto di somme**. Ciò richiede di analizzare i termini per i quali la funzione vale 0; dalla figura 4.20a si osserva che ciò accade in corrispondenza delle terne 000, 001, 100 e 110, cioè 0, 1, 4 e 6 con la codifica impiegata in precedenza.

L'idea è quella di esprimere il valore della funzione come prodotto di somme che sono sempre a 1, tranne che per una singola configurazione che le manda a 0; una funzione che vale sempre 1, tranne che per una singola configurazione per la quale vale 0 si chiama **termine massimo** (o **maxterm**), e viene indicata con  $M_i$ . In figura 4.22 viene rappresentato  $M_1$ ; il modo più semplice per esprimere un termine massimo è quello di ricorrere alla somma logica di variabili dirette e negate, le prime in corrispondenza delle variabili che in ingresso valgono 0, le seconde in corrispondenza delle variabili che in ingresso valgono 1. Nel caso di figura 4.22 si tratta della funzione  $x + y + \bar{z}$

### Tabella di verità

Tenendo conto di tutto ciò si perviene alla *seconda forma canonica* (o prodotti di somme), nella quale la generica funzione si rappresenta come

$$f(x_1, x_2, \dots, x_n) = \prod_{i=0}^{2^n-1} (\mu_i + M_i) = \prod_{i: \mu_i=0} M_i$$

Dove  $0 \leq i \leq 2^n - 1$ ,  $\mu_i$  è il valore della funzione in corrispondenza del termine massimo  $M_i$ , il quale si codifica secondo la procedura prima descritta: nel caso analizzato si ha  $M_0 = x + y + z$ ,  $M_1 = x + y + \bar{z}$ ,  $M_4 = \bar{x} + y + z$ ,  $M_6 = \bar{x} + \bar{y} + z$  e così via. Dal punto di vista circuitale, il prodotto dei *maxterm* si rappresenta come segue:

**Osservazione:** Si verifichi che:

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

Infatti:

$$\bar{x}y\bar{z} + \bar{x}yz + x\bar{y}z + xyz \stackrel{(A6)}{=} \bar{x}y(\bar{z} + z) + xz(\bar{y} + y) \stackrel{(A7)}{=} \bar{x}y \cdot 1 + xz \cdot 1 \stackrel{(A3)}{=} \bar{x}y + xz$$

Se, invece, si considera il prodotto di somme si ottiene:

$$(x + y + z) \cdot (x + y + \bar{z}) \cdot (\bar{x} + y + z) \cdot (\bar{x} + \bar{y} + z) \stackrel{(T9)}{=} (x + y) \cdot (\bar{x} + z) \stackrel{(T10)}{=} \bar{x}z + xz$$

In questo caso si è ottenuta la medesima espressione, anche dal punto di vista formale, ma potrebbe accadere che, con funzioni più complesse e, con un maggior numero di variabili si ottenga un risultato logicamente equivalente, ma formalmente differente. A tal proposito, sempre sfruttando i teoremi precedentemente esplicitati, si dovrà dimostrare che sono equivalenti anche formalmente. A seguito della semplificazione la complessità del circuito associato alla funzione si riduce drasticamente, come si può notare dalla figura 4.25; servono in tutto 3 porte a 2 ingressi al posto di 11 porte a 2 ingressi.

La semplificazione certamente apporta vantaggi enormi alla rete logica ottenuta, in quanto si riduce la dissipazione (che comunque non sarà mai nulla) e anche una minore probabilità di guasto. Naturalmente, maggiore è il numero di porte, maggiori saranno le transizioni a cui la rete logica sarà soggetta. Ad ogni transizione corrisponde una commutazione di stato del transistor, e quindi per una frazione di secondo si avrà sia tensione che corrente e quindi si rilevano dei picchi di dissipazione deleteri per le componenti circuitali.

Le formule (4.17) e (4.19), che portano entrambe alla forma semplificata  $\bar{x}y + xz$  per la funzione appena analizzata, aprono il problema della ricerca della forma algebrica minima, che consenta, cioè, di realizzare la funzione usando il minimo numero possibile di porte logiche.

Tanto per inquadrare la questione, si tenga conto che la (4.17) richiederebbe quattro porte AND con tre ingressi e una porta OR a quattro ingressi, più tutti i NOT che servono alla complementazione delle varie variabili; nel caso della (4.19) servono invece quattro porte OR con tre ingressi e una porta AND a quattro ingressi; se invece usiamo la funzione semplificata bastano due AND e un OR a due ingressi.

Affronteremo nel seguito il problema della minimazione delle espressioni. Si osservi inoltre che è sempre possibile passare dal prodotto di somme (4.21) alla somma di prodotti (4.20) e viceversa, sfruttando il Teorema di De Morgan a partire dalla doppia negazione T3

$$\begin{aligned} \overline{(x + y + z) \cdot (x + y + \bar{z}) \cdot (\bar{x} + y + z) \cdot (\bar{x} + \bar{y} + z)} &\stackrel{(T7)}{=} \overline{(x + y + z)} + \overline{(x + y + \bar{z})} + \overline{(\bar{x} + y + z)} + \overline{(\bar{x} + \bar{y} + z)} \\ &\stackrel{(T7)}{=} \bar{x}y\bar{z} + \bar{x}yz + x\bar{y}z + xyz \end{aligned}$$

La semplificazione si può a questo punto concludere nel modo seguente

$$\bar{x}y\bar{z} + \bar{x}yz + x\bar{y}z + xyz \stackrel{(A6)}{=} \bar{x}y(\bar{z} + z) + xz(\bar{y} + y) \stackrel{(A7)}{=} \bar{x}y + xz \stackrel{(T7)}{=} \bar{x}y \cdot \bar{x}z \stackrel{(T7)}{=} (x + y) \cdot (\bar{x} + z) \stackrel{(T10)}{=} \bar{x}y + xz$$

## 2.18 Interpretazione circuitale

Come già accennato in precedenza, la prima applicazione circuitale dell'Algebra Booleana si deve a Shannon, che la usò nella progettazione di circuiti complessi per la commutazione a contatti. Questo tipo di applicazione è ancora largamente usata, anche se i contatti dei rel 'e sono oggi sostituiti da dispositivi a stato solido (SCR, TRIAC). Tuttavia l'importanza dell'Algebra Booleana è legata soprattutto all'impiego nell'ambito dei circuiti logici, il cui peso è oggi preponderante.

Nello stesso periodo di Shannon, Bush stava progettando un calcolatore analogico che viene utilizzata per la risoluzione di equazioni differenziali mediante la progettazione di circuiti di resistenze retti proprio dall'equazione differenziale di interesse. L'evoluzione del circuito, poi, permetterà di rilevare tutte le soluzioni dell'equazione analizzata. Nell'interpretazione di Shannon le costanti logiche 0 e 1 indicano rispettivamente un circuito aperto o uno chiuso (mediante, per esempio, un interruttore), mentre le variabili indicano il contatto di un interruttore o di un relè.

Con i simboli  $x$  e  $\bar{x}$  si indicano due contatti azionati contemporaneamente, ma sempre tali che quando uno è aperto, l'altro è chiuso e viceversa.

Se si pongono degli interruttori in parallelo, si ottiene un circuito che simula una porta logica OR, mentre se gli interruttori vengono posti in serie si ottiene un circuito perfettamente equivalente ad una porta logica AND.

La stessa cosa accade con un circuito deviatore che permette di ottenere la negazione di una variabile.

In questo modo è possibile esprimere una funzione logica complicata che sia attraverso la realizzazione di un circuito equivalente, e viceversa.

13 Ottobre 2021

## 2.19 Riassunto

È stata precedentemente introdotta una duplice modalità per manipolare le espressioni logiche, ovvero la **prodotto dei termini massimi** (o *maxterm*, che valgono sempre 1, tranne che per una sola combinazione per cui valgono 0) e **somma dei termini minimi** (o *minterm*, che valgono sempre 0 tranne che per una sola combinazione per la quale valgono 1).

Naturalmente, le espressioni sono **mutuamente escludenti**, per cui è chiaro che in ambedue i casi è ovvio che in ingresso si presenti solamente uno solo dei *minterm* (o *maxterm*) individuati.

Il discernimento tra una o l'altra configurazione è assolutamente ininfluenza: certo è che, se la funzione logica da semplificare assume maggiori 1 rispetto allo 0, allora sarà più conveniente procedere alla trasformazione della funzione in **prodotti di somme**, o di *maxterm*.

Naturalmente, però, se, come si è detto, i due procedimenti conducono al medesimo risultato, pertanto sarà sempre possibile ricondurre una somma di *minterm* ad un prodotto di *maxterm*. È chiaro che è possibile che vi siano più forme minime equivalenti, che differiscono solamente dal punto di vista formale.

Un procedimento atto a passare dal prodotto di somme alla somma di prodotti e viceversa, sfruttando il Teorema di De Morgan 2.6 a partire dalla doppia negazione.

**Osservazione:** L'interpretazione circuitale dell'algebra booleana si deve a Claude Shannon, il quale osservò, molto semplicemente che:

- Interruttori posti in serie simulano il funzionamento di una porta AND
- Interruttori posti in parallelo simulano il funzionamento di una porta OR
- Un deviatore permette di simulare il funzionamento di una porta NOT

**Esempio:** Tale osservazione può avere dei risvolti particolarmente interessanti dal punto di vista pratico.

Si consideri, come esempio concreto, il problema seguente: in un appartamento c'è un corridoio; quando si entra si vuole accendere la luce dall'interruttore  $x$  prossimo all'ingresso, che si supponga sia a destra nel disegno di figura 4.31). Uscendo si vuole spegnere la luce dall'interruttore  $y$  prossimo all'uscita sulla sinistra.

Dal momento che si hanno due variabili binarie  $x$  e  $y$ , è possibile costruire una tavola di verità che permetta di rappresentare tutti i possibili stati della luce (accesa o spenta) in funzione del valore delle variabili in ingresso.

Pertanto:

| $x$ | $y$ | $L$ |
|-----|-----|-----|
| 0   | 0   | 0   |
| 0   | 1   | 1   |
| 1   | 0   | 1   |
| 1   | 1   | 0   |

Tabella 21: Circuito di comando di una luce da due punti e relativa tavola di verità

Si proceda, ora, alla traduzione di tale funzione logica in forma canonica. Pertanto, tale funzione espressa come **somma di *minterm***:

$$\overline{x}y + x\overline{y}$$

che, dal punto di vista circuitale, corrisponde ad un circuito in parallelo con due deviatori, dimodoché sia soddisfatta la funzione logica di partenza.

Mentre come **prodotto di *maxterm***:

$$(\overline{x} + \overline{y}) \cdot (x + y)$$

che, dal punto di vista circuitale, corrisponde ad un circuito in serie, in cui sono sempre presenti due deviatori. Tuttavia, tale soluzione non è conveniente, in quanto si necessita di un cavo in più per la connessione tra i due deviatori.

**Osservazione:** Naturalmente, in questo caso si palesa la necessità di prevedere l'adeguamento degli interruttori in funzione della tipologia di carico da azionare.

Infatti, quando le lamine dei conduttori vengono poste a contatto, per quanto possa essere irrilevante, si ha una resistenza di contatto che comporta la dissipazione di potenza. Ma non solo, quando le lamine vengono avvicinate per la chiusura del contatto, si ha una **carica distruttiva** che comporta la generazione di una scarica elettrica che, con successivi e multipli utilizzi, determina un danneggiamento dell'interruttore. Per questo si necessita l'applicazione di un interruttore che esercita una maggiore pressione fra le lamine, ma questo comporta un maggior costo.

In questo secondo caso, si necessita la presenza di un *transistor* e di un relè, in modo tale che la **circuiteria di potenza** (ovvero il grosso interruttore e la lampada) sia separata dalla **circuiteria di controllo logico**, che viene realizzata con delle porte logiche.

Questo, chiaramente, comporta un maggior costo, ma con due grandi vantaggi. Il primo rappresentato dal fatto che si ha una netta separazione tra i due circuiti, mentre il secondo è rappresentato dalla sicurezza, in quanto si ha una barriera tra il circuito ad elevata tensione e quello che opera ai livelli di tensione propri delle porte logiche. Pertanto, nel malaugurato caso in cui vi sia una perdita dell'interruttore, l'utente è separato e protetto da eventuali scariche di alta tensione.

**Esempio:** Si consideri, ora, il caso in cui al posto di avere due soli interruttori per comandare un carico.

In questo secondo caso, si avranno tre variabili binarie  $x, y$  e  $z$  per le quali si costruisce la seguente tabella di verità:

| $x$ | $y$ | $L$ |   |
|-----|-----|-----|---|
| 0   | 0   | 0   | 0 |
| 0   | 0   | 1   | 1 |
| 0   | 1   | 0   | 1 |
| 0   | 1   | 1   | 0 |
| 1   | 0   | 0   | 1 |
| 1   | 0   | 1   | 0 |
| 1   | 1   | 0   | 0 |
| 1   | 1   | 1   | 1 |

Tabella 22: Circuito di comando di una luce da tre punti e relativa tavola di verità

La sintesi del circuito mediante termini minimi fornisce

$$\overline{x}yz + \overline{x}y\overline{z} + x\overline{y}\overline{z} + xyz$$

Prima di procedere alla realizzazione circuitale, è però opportuno semplificare il più possibile l'espressione, ottenendo

$$\overline{x}(\overline{y}z + y\overline{z}) + x(\overline{y}\overline{z} + yz)$$

Il circuito che si ricava è rappresentato nella figura seguente:

Si noti che i circuiti ottenuti presentano un numero di invertitori maggiore del numero di interruttori richiesti dal problema.

Si necessita, allora, l'impiego di un dispositivo noto con il nome di **invertitore**, ovvero un **doppio deviatore** che si può trovare solamente in due possibili stati  $=$ , oppure  $\times$ .

Si provi, ora, a realizzare la rete logica mediante i termini massimi, per cui si ottiene

$$(x + y + z) \cdot (x + y + \overline{z}) \cdot (\overline{x} + y + \overline{z}) \cdot (\overline{x} + \overline{y} + z) = [x + (y + z) \cdot (\overline{y} + \overline{z})] \cdot [(\overline{x} + (y + \overline{z})) \cdot (\overline{y} + z)]$$

che con la semplificazione porta al circuito della figura seguente:

Anche in questo caso la soluzione basata sui termini massimi è meno vantaggiosa, poiché richiede 5 deviatori invece di 4, ma tale circuito è del tutto analogo a quello precedente, ma differente dal punto di vista topologico.

Al di là della straordinaria importanza applicativa del metodo formale basato sull'Algebra Booleana, importante riflettere, soprattutto alla luce di quest'ultimo esempio, che molto spesso in ambito ingegneristico la teoria offre diverse soluzioni tra loro equivalenti; sta poi al progettista scegliere la migliore, tenuto conto della razionalità e semplicità delle soluzioni prospettate, dei costi, del numero di componenti impiegati ecc.

## 2.20 Semplificazione delle espressioni Booleane

Si è già visto, dagli esempi che precedono, che le forme canoniche non esauriscono le espressioni analitiche di una funzione; anzi, come per qualsiasi relazione algebrica, anche quelle logiche possono essere trasformate in un certo numero di espressioni formalmente diverse, ma equivalenti dal punto di vista matematico.

Ad esempio:

$$f = \overline{xy}z + \overline{x}yz + \overline{x}yz + xyz = \overline{x}z(\overline{y} + y) + yz(\overline{x} + x) = \overline{x}z + yz$$

Si diranno equivalenti due funzioni che abbiano la stessa tavola di verità, forma semplificata di una funzione ogni sua espressione non canonica, forma minima quella in cui ogni variabile, diretta o negata che sia, compare il minor numero di volte.

Le tecniche di semplificazione sono due:

1. Mappe di Karnaugh
2. Metodo tabellare di Quine - Mc Cluskey

### 2.20.1 Mappe di Karnaugh

Il metodo proposto da Karnaugh è un metodo grafico di semplificazione che permette di ottenere molto semplicemente la forma minima di una funzione espressa come somma di prodotti (*minterm*), facendo ricorso a particolari mappe di rappresentazione. Quale limitazione si ha che, sebbene il metodo sia concettualmente applicabile a funzioni di qualsiasi numero di variabili, esso diviene difficoltoso già per 5, 6 variabili.

Le mappe di Karnaugh, che possono essere considerate un ulteriore metodo di rappresentazione di una funzione logica, consistono in matrici di  $m$  righe e  $k$  colonne, in cui  $m = 2^i$  e  $k = 2^j$  per qualche  $i, j$ , col vincolo che

$$2^i \cdot 2^j = 2^n$$

ovvero pari al numero di elementi della matrice, in cui  $n$  è il numero delle variabili. Di conseguenza, le mappe di Karnaugh presentano 4 celle nel caso di funzioni di due variabili, 8 per quelle di tre variabili, 16 per quelle di quattro e così via.

Ogni elemento della matrice rappresenta un termine minimo, che entra in un'espressione *minterm* di somma di prodotti.

Infatti, il concetto alla base delle mappe di Karnaugh prevede di disporre i termini minimi nello spazio in modo tale che i termini minimi da semplificare si trovino vicini fra di loro.

In particolare, le mappe devono essere costruite in modo tale che nel passaggio da una cella all'altra ci sia sempre e solo un'unica variabile che viene a modificarsi. Ciò implica che la tabella è chiusa in se stessa tanto in senso orizzontale, quanto in senso verticale.

Per questo motivo, infatti, nella disposizione dei numeri binari sulle righe e sulle colonne si utilizza la sequenza 00, 01, 11, 10 dimodoché nel passaggio da una colonna riga all'altra vi sia il cambio di una sola variabile.

18 Ottobre 2021

## 2.21 Riassunto

Le forme canoniche sono di due tipi: *minterm* e *maxterm*. In particolare considerando la funzione che vale sempre 1 tranne per un solo caso in cui vale 0 prende il nome di *maxterm*, mentre la funzione che vale sempre 0 e vale 1 solamente per un solo caso, ovvero quello oggetto di studio prende il nome di *minterm*. Talune forme sono esattamente equivalenti per l'espressione di una funzione logica: **prodotto di *maxterm*** oppure **somma di *minterm***.

Dal punto di vista pratico, come si è detto, sono perfettamente equivalenti: certo è che, se una funzione presenta più 0 che 1, sarà più conveniente ricorrere al **prodotto di *maxterm*** anziché una **somma di *minterm***.

Attraverso l'applicazione dei teoremi fondanti dell'algebra booleana, così come attraverso i teoremi di De Morgan, è possibile passare da una forma all'altra in modo molto equivalente, giungendo ad una semplificazione decisamente vantaggiosa, sia in termini circuitali che in termini logici.

L'interpretazione circuitale dell'Algebra Booleana si deve a Shannon, che la scelse come tema del suo master. La chiave centrale di tale interpretazione sono gli interruttori che rappresentano le variabili booleane e che possono essere poste in serie per simulare il funzionamento di una porta AND; se sono poste in parallelo simulano il funzionamento di una porta OR.

La commutazione di ciascun interruttore rappresentante una variabile determina una particolare configurazione del circuito da cui è possibile estrarre la corrispondente funzione logica.

L'utilizzo dell'Algebra Booleana e delle tavole di verità è fondamentale nella progettazione di un circuito e per la determinazione della soluzione più conveniente e pratica dal punto di vista circuitale.

Dal punto di vista tecnico, inoltre, si è osservato che il carico che è necessario comandare comporterebbe la necessità di dimensionare tutti gli interruttori esposti all'elevata tensione del carico, con un elevato costo per ciascun interruttore. Attraverso l'utilizzo di un relè e di un transistor, si ha una separazione netta tra la parte del circuito ad elevata tensione e quella di controllo, per cui gli interruttori con cui interagisce l'utente lavorano alla tensione delle porte logiche, ovvero pochi volt, eliminando il rischio delle perdite di elevate tensioni.

Inoltre, gli interruttori che operano ad elevata tensione sono molto costosi e pesanti e presentano una molla di richiamo che sarebbe stato difficile azionare, anche dal punto di vista meccanico. Implementando, invece, interruttori che operano sui 5V si agevola anche l'azione di accensione e spegnimento da parte dell'utente.

L'accessione di un punto luce da più di 2 interruttori, invece richiederebbe l'uso di un dispositivo denominato **invertitore**, che può essere installato in quantità molteplice per assolvere il compito richiesto. Le semplificazioni booleane effettuate ricorrendo ai teoremi dell'Algebra Booleana o ai teoremi di De Morgan sono molto complesse e dal basso tasso di successo.

Le mappe di Karnaugh, invece, sono delle mappe **bidimensionali** (che, comunque, ne limita le applicazioni) che presentano un numero di righe e di colonne che deve essere sempre  $2^i \times 2^j = 2^n$ , ove  $n$  è il numero delle variabili di cui si sta analizzando l'equazione logica. Inoltre, le  $n$ -uple di variabili devono essere disposte in modo tale che tra due  $n$ -uple successive vi sia la variazione di una sola variabile.

Si presti particolare attenzione che la mappa deve essere considerata **chiusa** sia in senso orizzontale che in senso verticale, pertanto la combinazione 00,01,11,10 rispetta la regola precedentemente esposta, in quanto la dupla 10 è considerata adiacente a 00, ma comunque tra le due si ha il cambiamento di una sola variabile.

Nella Figura 13 sono riprodotte le strutture delle mappe per 2, 3 e 4 variabili.



|     |   |                 |  |                 |   |
|-----|---|-----------------|--|-----------------|---|
|     |   | $x$             |  | 0               | 1 |
| $y$ | 0 | $\overline{xy}$ |  | $x\overline{y}$ |   |
|     | 1 | $\overline{x}y$ |  | $xy$            |   |

|     |   |                  |                             |                  |                  |
|-----|---|------------------|-----------------------------|------------------|------------------|
|     |   | $xy$             |                             |                  |                  |
| $z$ | 0 | $\overline{xyz}$ | $\overline{x}y\overline{z}$ | $xy\overline{z}$ | $x\overline{y}z$ |
|     | 1 | $\overline{x}yz$ | $\overline{xy}z$            | $xyz$            | $x\overline{y}z$ |

|      |    |      |  |  |  |
|------|----|------|--|--|--|
|      |    | $xy$ |  |  |  |
| $zw$ | 00 |      |  |  |  |
|      | 01 |      |  |  |  |
|      | 11 |      |  |  |  |
|      | 10 |      |  |  |  |

Figura 13: Mappe di Karnaugh a 16, 8 e 2

Grazie al fatto che le mappe sono costruite in modo tale che due celle contigue differiscono solamente per una sola variabile è possibile costruire il **prodotto di maxterm** e la **somma di minterm** osservando dove la funzione vale 0 o 1 rispettivamente. Costruendo i **sottocubi** come multipli di 2 è possibile costruire il **determinante** semplicemente osservando sulle righe e sulle colonne interessate le variabili che rimangono costanti.

In particolare:

|     |   |      |   |   |   |
|-----|---|------|---|---|---|
|     |   | $xy$ |   |   |   |
| $z$ | 0 | 1    | 0 | 0 | 0 |
|     | 1 | 1    | 0 | 0 | 1 |

Figura 14: Mappa di Karnaugh a 8

Pertanto costruendo i due implicanti corrispondenti a due *minterm* si ha:

$$\overline{xy} + z\overline{y}$$

**Osservazione:** Quando si costruiscono i **sottocubi** bisogna fare attenzione a non includere dei termini già considerati, per cui si costruirebbero degli **implicanti non primi**.

In particolare, vige la regola dei **sottocubi massimali** e della **copertura minima**.

**Osservazione:** Naturalmente, attraverso la costruzione degli implicanti è possibile ottenere, in conclusione, delle semplificazioni corrette, ma formalmente diverse, ma deve essere sempre possibile passare dall'una all'altra.

**Esempio:** Si consideri la seguente mappa di Karnaugh:

|      |    |      |   |   |   |
|------|----|------|---|---|---|
|      |    | $xy$ |   |   |   |
| $zw$ | 00 | 1    | 0 | 0 | 0 |
|      | 01 | 0    | 0 | 0 | 0 |
|      | 11 | 1    | 1 | 0 | 0 |
|      | 10 | 0    | 1 | 0 | 0 |

Figura 15: Mappa di Karnaugh a 16

**Esempio:** Si consideri la seguente mappa di Karnaugh:

|      |    | $xy$ |    |    |    |
|------|----|------|----|----|----|
|      |    | 00   | 01 | 11 | 10 |
| $zw$ | 00 | 1    | 0  | 0  | 1  |
|      | 01 | 1    | 1  | 0  | 1  |
|      | 11 | 1    | 1  | 0  | 1  |
|      | 10 | 1    | 0  | 0  | 1  |

Figura 16: Mappa di Karnaugh a 16

### 2.21.1 Mappa di Karnaugh sui termini massimi

Le mappe di Karnaugh possono essere usate anche per costruire la funzione semplificata secondo la II forma canonica basata sul prodotto di somme. In tal caso i sottocubi e la relativa copertura minima vanno costruiti sugli 0.

Si consideri la seguente mappa di Karnaugh:

|     |   | $xy$ |    |    |    |
|-----|---|------|----|----|----|
|     |   | 00   | 01 | 11 | 10 |
| $z$ | 0 | 1    | 0  | 0  | 1  |
|     | 1 | 1    | 1  | 0  | 1  |

Figura 17: Mappa di Karnaugh a 8

### 2.21.2 Mappe di Karnaugh a 5 o più variabili

Per la realizzazione delle mappe di Karnaugh con più di 4 variabili è impossibile costruire un'unica mappa, ma è sufficiente, nel caso di 5 variabili, sdoppiare la mappa in due sottomappe da 4:

| $zw$ |    | $xy$ |    |    |    |
|------|----|------|----|----|----|
|      |    | 00   | 01 | 11 | 10 |
| 00   | 00 | 0    | 0  | 0  | 1  |
| 01   | 01 | 0    | 0  | 0  | 1  |
| 11   | 11 | 0    | 0  | 0  | 1  |
| 10   | 10 | 0    | 0  | 0  | 1  |

| $zw$ |    | $xy$ |    |    |    |
|------|----|------|----|----|----|
|      |    | 00   | 01 | 11 | 10 |
| 00   | 00 | 0    | 1  | 0  | 1  |
| 01   | 01 | 0    | 0  | 0  | 1  |
| 11   | 11 | 1    | 1  | 1  | 1  |
| 10   | 10 | 0    | 0  | 0  | 1  |

Figura 18: Mappa di Karnaugh a 32

Se si avessero 6 variabili, si necessiterebbe la costruzione di 4 mappe di Karnaugh a 16 e così via.

## 2.22 Condizioni non specificate

Nella sintesi di una funzione logica di  $n$  variabili si può presentare il caso in cui per  $k$  configurazioni delle variabili di ingresso la funzione vale 1, per  $m$  configurazioni vale 0, ma  $k+m = 2^n$ . Le restanti

$2^n - (m + k)$  configurazioni vengono dette **condizioni non specificate** (o anche d.c.c., acronimo di *don't care condition*).

In pratica questa situazione si verifica ogni qualvolta in un circuito certe configurazioni di ingresso sono fisicamente impossibili, o rendono privo di significato il valore dell'uscita. Da un punto di vista strettamente analitico ciò accade quando una funzione  $F$  è funzione delle variabili  $\phi_1, \phi_2, \phi_3, \dots, \phi_m$ , ognuna delle quali è a sua volta funzione delle variabili  $x_1, x_2, \dots, x_n$ , cioè

*...continua...*

Si nota che le terne di possibili valori  $\phi_1, \phi_2, \phi_3$  sono 000, 001, 010, 100, 101, mentre le configurazioni rimanenti 011, 110, 111 non compaiono. Ne consegue che la tavola di verità della  $F$  conterrà condizioni non specificate in corrispondenza di queste configurazioni d'ingresso.

Sulla tavola di verità le condizioni non specificate vengono indicate con un trattino, nella forma canonica raccogliendo in parentesi i termini minimi corrispondenti, sulle mappe di Karnaugh contrassegnando con il simbolo  $\wedge$  la casella corrispondente a ciascuna condizione non specificata. Le condizioni non specificate possono venir sfruttate nelle semplificazioni, in modo da pervenire ad espressioni minime più semplici. Se si opera con le mappe di Karnaugh, le semplificazioni vanno ancora fatte in modo da coprire tutte le caselle contrassegnate con un 1, ma se serve si possono aggregare anche le caselle associate a condizioni non specificate, che possiamo contrassegnare con  $\wedge$ ; in pratica si tratta di utilizzare le condizioni non specificate per allargare al massimo i sottocubi di copertura della funzione, assegnando a ciascuna di loro il valore 1 o 0 a seconda che torni o meno utile per ottenere sottocubi più ampi. Nell'esempio di sopra si ottiene

11 Ottobre 2021

### 2.23 Riassunto

La semplificazione delle espressioni booleane può avvenire in tre modi egualmente validi:

1. Teoremi dell'Algebra Booleana
2. Mappa di Karnaugh
3. Metodo tabellare Quine-McCluskey

Le mappe di Karnaugh sono delle **tabelle piane** che presentano un numero di celle pari a  $2^n$ , ove  $n$  è il numero di variabili oggetto di studio. Le righe e le colonne devono essere formate nel seguente modo

$$2^i \cdot 2^j = 2^n$$

La realizzazione prevede di disporre le variabili sulle righe e sulle colonne in modo tale che tra una cella all'altra, tanto in senso orizzontale quanto in quello verticale, vi sia una sola variazione di una variabile, nel senso della sua complementarietà.

L'individuazione dei blocchi di *minterm* deve avvenire rispettando due importanti condizioni: **sottocubi massimali** e della **copertura minima**.

Per determinare, poi, a quale **implicante** corrisponde al sottocubo evidenziato è sufficiente rilevare quelle variabili che nel sottocubo rimangono costanti sia in orizzontale che in verticale, nel senso della loro complementarietà.

Si ricordi, ancora, che le mappe di Karnaugh devono essere considerate chiuse in senso circolare (**adiacenza per circolarità**), per cui è necessario disporre le variabili come 00, 01, 11, 10 per cui tra 00 – 01, 01 – 11, 11 – 10 e 10 – 00 vi è la variazione di una sola variabile.

Naturalmente, con questa modalità è possibile anche individuare **sottocubi non primi**, ovvero sottocubi che includono minterm già inglobati in altri sottocubi. Certo è che la somma di minterm risultante sarà comunque correttamente espressa, solamente sarà presente una ridondanza superflua, che può essere semplificata tramite i teoremi dell'Algebra Booleana. Naturalmente, però, è sufficiente evitare di considerare sottocubi non primi per ottenere, alla fine, la formula minima di espressione della funzione booleana.

Naturalmente, in forza del **principio di dualità**, così com'è possibile ottenere una somma di *minterm* tramite le mappe di Karnaugh, sarà anche possibile ottenere un prodotto di *maxterm*, che può rivelarsi più conveniente qualora nella tabella siano maggiormente presenti gli 0 piuttosto che gli 1.

Naturalmente, è possibile procedere all'applicazione delle mappe di Karnaugh anche a 5, 6 variabili, con un sensibile aumento della complicazione della procedura di semplificazione. In questi casi, sarà maggiormente conveniente operare tramite il metodo tabella di Quine-McCluskey (anche più facilmente programmabile).

È possibile, inoltre, che la funzione analizzata non possa presentare determinate combinazioni di variabili in ingresso, le quali sarebbero prive di significato. Allora, nella procedura di semplificazione tramite mappa di Karnaugh, laddove figurano le combinazioni di variabili in ingresso si pone una  $X$  che può tramutarsi in 1 o 0 al fine di **massimizzare i sottocubi** e garantire una **copertura minima**.

**Esercizio:** Si costruisca una funzione  $f$  a  $n = 3$  variabili:

| $x$ | $y$ | $z$ | $f(x, y, z)$ |
|-----|-----|-----|--------------|
| 0   | 0   | 0   | 1            |
| 0   | 0   | 1   | 1            |
| 0   | 1   | 0   | 0            |
| 0   | 1   | 1   | 1            |
| 1   | 0   | 0   | 1            |
| 1   | 0   | 1   | 0            |
| 1   | 1   | 0   | 1            |
| 1   | 1   | 1   | 1            |

Tabella 23: Funzione a  $n = 3$  variabili

Considerando la **somma di *minterm*** si ottiene

$$f = \overline{x}y\overline{z} + \overline{x}yz + \overline{x}yz + x\overline{y}\overline{z} + x\overline{y}z + xyz$$

Che si può semplificare come segue, sfruttando i **teoremi di assorbimento** e dell'**idempotenza**:

$$f = \overline{x}y \cdot (\overline{z} + z) + yz \cdot (\overline{x} + x) + \overline{y}z \cdot (\overline{x} + x) + xy \cdot (\overline{z} + z)$$

ottenendo

$$f = \overline{x}y + yz + \overline{y}z + xy = \overline{y} \cdot (\overline{x} + \overline{z}) + y \cdot (x + z)$$

Considerando il **prodotto di *maxterm*** si ottiene

$$f = (\overline{x} + y + \overline{z}) \cdot (x + \overline{y} + z)$$

ovvero

$$f = \overline{x}x + \overline{x}y + \overline{x}z + xy + \overline{y}y + xz + \overline{x}z + y\overline{z} + z\overline{z}$$

da cui si ottiene

$$f = xy + x\overline{z} + \overline{x}y + \overline{y}z + \overline{x}z + yz$$

Ma per il **teorema dell'assorbimento 3**,  $x\overline{y} + \overline{x}z + \overline{y}z = x\overline{y} + \overline{x}z$  così come  $xy + \overline{x}z + yz = xy + \overline{x}z$ , pertanto si ottiene:

$$f = x\overline{z} + \overline{x}y + xy + \overline{x}z$$

Si realizzi, allora, la mappa di Karnaugh corrispondente, scegliendo i *minterm*

|     |   |      |    |    |    |
|-----|---|------|----|----|----|
|     |   | $xy$ |    |    |    |
|     |   | 00   | 01 | 11 | 10 |
| $z$ | 0 | 1    | 0  | 1  | 1  |
|     | 1 | 1    | 1  | 1  | 0  |

Da cui si ottiene

$$f = \overline{y}z + \overline{x}z + xy$$

la quale è differente dalla espressione ottenuta all'inizio. Per cui, costruendo un'ulteriore mappa di Karnaugh si individuano tutti i termini presenti nell'espressione di partenza, ovvero

$$f = \overline{x}y + yz + \overline{y}z + xy$$

|     |   | $xy$ |    |    |    |
|-----|---|------|----|----|----|
|     |   | 00   | 01 | 11 | 10 |
| $z$ | 0 | 1    | 0  | 1  | 1  |
|     | 1 | 1    | 1  | 1  | 0  |

Pertanto, come si nota, sono stati scelti dei sottocubi *non primi*, pertanto vi sono 4 termini al posto di 3, come formula minima.

Pertanto si deve cercare di eliminare i due sottocubi non primi  $xy$  e  $\bar{x}z$  che deve divenire  $yz$ , ovvero

$$xy + \bar{x}z = yz$$

Si consideri una nuova funzione a  $n = 3$  variabili

| $x$ | $y$ | $z$ | $f(x, y, z)$ |
|-----|-----|-----|--------------|
| 0   | 0   | 0   | 1            |
| 0   | 0   | 1   | 0            |
| 0   | 1   | 0   | 1            |
| 0   | 1   | 1   | 0            |
| 1   | 0   | 0   | 1            |
| 1   | 0   | 1   | 0            |
| 1   | 1   | 0   | 0            |
| 1   | 1   | 1   | 0            |

Tabella 24: Funzione a  $n = 3$  variabili

Considerando la **somma di minterm** si ottiene

$$f = \bar{x}y\bar{z} + \bar{x}y\bar{z} + x\bar{y}\bar{x}$$

che per l'idempotenza di suppo scrivere come:

$$f = \bar{x}y\bar{z} + \bar{x}y\bar{z} + x\bar{y}\bar{x} + \bar{x}y\bar{z}$$

Per cui, per il teorema dell'assorbimento si ha

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

Considerando il **prodotto di maxterm** si ottiene

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

Che sulla mappa di Karnaugh figura come segue:

|     |   | $xy$ |    |    |    |
|-----|---|------|----|----|----|
|     |   | 00   | 01 | 11 | 10 |
| $z$ | 0 | 1    | 1  | 0  | 1  |
|     | 1 | 0    | 0  | 0  | 0  |

**Esercizio:** Si consideri la seguente funzione logica:

Si consideri una nuova funzione a  $n = 3$  variabili

| $x$ | $y$ | $z$ | $f(x, y, z)$ |
|-----|-----|-----|--------------|
| 0   | 0   | 0   | 1            |
| 0   | 0   | 1   | 1            |
| 0   | 1   | 0   | 0            |
| 0   | 1   | 1   | 1            |
| 1   | 0   | 0   | 0            |
| 1   | 0   | 1   | 0            |
| 1   | 1   | 0   | 1            |
| 1   | 1   | 1   | 0            |

Tabella 25: Funzione a  $n = 3$  variabili

Considerando la **somma di *minterm*** si ottiene

$$f = \overline{x}y\overline{z} + \overline{x}yz + \overline{x}yz + xy\overline{z}$$

Procedendo alla semplificazione si ottiene

$$f = \overline{x}y(\overline{z} + z) + \overline{x}yz + xy\overline{z}$$

e procedendo per **idempotenza**, si ottiene

$$f = \overline{x}y(\overline{z} + z) + \overline{x}yz + \overline{x}yz + xy\overline{z} = \overline{x}y(\overline{z} + z) + \overline{x}z(\overline{y} + y) + xy\overline{z}$$

che equivale a

$$f = \overline{x}y + \overline{x}z + xy\overline{z}$$

Considerando il **prodotto di *maxterm*** si ottiene:

$$f = (x + \overline{y} + z) \cdot (\overline{x} + y + z) \cdot (\overline{x} + y + \overline{z}) \cdot (\overline{x} + \overline{y} + \overline{z})$$

che per il teorema **T9** si ottiene

$$f = (x + \overline{y} + z) \cdot (\overline{x} + y + z) \cdot (\overline{x} + \overline{z})$$

Procedendo per **idempotenza** si procede alla duplicazione di  $\overline{x} + y + \overline{z}$  per cui si ottiene:

$$f = (x + \overline{y} + z) \cdot (\overline{x} + y + z) \cdot (\overline{x} + y + \overline{z}) \cdot (\overline{x} + \overline{z})$$

da cui, sempre per il teorema **T9**, si ottiene

$$f = (x + \overline{y} + z) \cdot (\overline{x} + y) \cdot (\overline{x} + \overline{z})$$

Ancora una volta si è ottenuto un risultato di **prodotto di *maxterm*** differente dalla **somma di *minterm***, per cui, sviluppando i prodotti si ha:

$$f = (x + \overline{y} + z) \cdot (\overline{x} + y) \cdot (\overline{x} + \overline{z}) = (x + \overline{y} + z) \cdot (\overline{x} + \overline{x}\overline{z} + \overline{x}y + y\overline{z})$$

che per il teorema dell'assorbimento 3 si ottiene:

$$f = (x + \overline{y} + z) \cdot (\overline{x} + y\overline{z})$$

che, sviluppando il prodotto è uguale a

$$f = (x\overline{x}...continua...$$

Di fatto, come si osserva, si è ottenuto il medesimo risultato di partenza.

La realizzazione della mappa di Karnaugh si ottiene

|     |   | $xy$ |    |    |    |
|-----|---|------|----|----|----|
|     |   | 00   | 01 | 11 | 10 |
| $z$ | 0 | 1    | 0  | 1  | 0  |
|     | 1 | 1    | 1  | 0  | 0  |

Che ancora una volta produce il risultato cercato:

$$\overline{xy} + \overline{x}z + xy\overline{z}$$

Si consideri una nuova funzione a  $n = 3$  variabili

| $x$ | $y$ | $z$ | $f(x, y, z)$ |
|-----|-----|-----|--------------|
| 0   | 0   | 0   | 1            |
| 0   | 0   | 1   | 0            |
| 0   | 1   | 0   | 1            |
| 0   | 1   | 1   | 1            |
| 1   | 0   | 0   | 0            |
| 1   | 0   | 1   | 1            |
| 1   | 1   | 0   | 1            |
| 1   | 1   | 1   | 0            |

Tabella 26: Funzione a  $n = 3$  variabili

Considerando la **somma di *minterm*** si ottiene

$$f = \overline{x}\overline{y}\overline{z} + \overline{x}y\overline{z} + \overline{x}yz + x\overline{y}\overline{z} + xy\overline{z}$$

Procedendo alla semplificazione si ottiene

$$f = \overline{x}(\overline{z} + yz) + x\overline{y} + x\overline{y}z + xy\overline{z} = \overline{x}(\overline{z} + y) + x\overline{y} + x\overline{y}z + xy\overline{z}$$

ovvero

$$f = \overline{x}\overline{z} + \overline{x}y + x\overline{y} + x\overline{y}z + xy\overline{z}$$

Ancora una volta è possibile raccogliere il primo e l'ultimo addendo

$$f = \overline{z} \cdot (\overline{x} + xy) + \overline{x}y + x\overline{y} + x\overline{y}z$$

ovvero

$$f = \overline{x}\overline{z} + \overline{z}y + \overline{x}y + x\overline{y} + x\overline{y}z$$

da cui

$$f = \overline{x}\overline{z} + \overline{x}y + y\overline{z} + x\overline{y}z$$

Considerando, ora, il **prodotto di *maxterm*** si ottiene

$$f = (\overline{x} + \overline{y} + z) \cdot (x + \overline{y} + \overline{z}) \cdot (x + y + z)$$



Non potendo semplificare, si procede allo sviluppo dei prodotti:

$$f = (x\bar{x} + xy + xz + \bar{x}y + y + yz + \bar{x}z + y\bar{z} + z\bar{z}) \cdot (\bar{x} + \bar{y} + \bar{z})$$

da cui si ottiene

$$f = (y \cdot (x + \bar{x} + 1 + z + \bar{z}) + xz + \bar{x}z) \cdot (\bar{x} + \bar{y} + \bar{z})$$

ovvero

$$f = (y + xz + \bar{x}z) \cdot (\bar{x} + \bar{y} + \bar{z})$$

Ancora una volta si sviluppano i prodotti:

$$f = \bar{x}y + y\bar{y} + y\bar{z} + x\bar{x}z + x\bar{y}z + xz\bar{z} + \bar{x}z + \bar{x}y\bar{z} + \bar{x}\bar{z}$$

da cui

$$f = \bar{x}y + y\bar{z} + x\bar{y}z + \bar{x}z + \bar{x}y\bar{z} + \bar{x}\bar{z}$$

che, per il teorema dell'assorbimento si semplifica come:

$$f = \bar{x}z + \bar{x}y + y\bar{z} + x\bar{y}z$$

e procedendo per **idempotenza**, si ottiene

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

che equivale a

$$f = \bar{x}z + \bar{x}y + y\bar{z} + x\bar{y}z$$

La realizzazione della mappa di Karnaugh si ottiene

|     |   |      |    |    |    |
|-----|---|------|----|----|----|
|     |   | $xy$ |    |    |    |
|     |   | 00   | 01 | 11 | 10 |
| $z$ | 0 | 1    | 1  | 1  | 0  |
|     | 1 | 0    | 1  | 0  | 1  |

Da cui si può osservare:

$$f = \bar{x}z + \bar{x}y + y\bar{z} + x\bar{y}z$$

20 Ottobre 2021

## 2.24 Metodo tabellare Quine - Mc Cluskey

Il metodo di Quine - Mc Cluskey è un procedimento tabellare che consente di ottenere la forma minima come somma di prodotti per qualsiasi funzione logica. Esso si basa sulla relazione:

$$fx + f\bar{x} = f$$

già usata in precedenza per le mappe di Karnaugh; solo che ora essa viene applicata in modo sistematico a tutti i termini minimi della funzione.

Per capirne la logica si consideri la seguente funzione logica:

|   |                             | $x$ | $y$ | $z$ | $f$ |  | Livello |   |                             | $x$ | $y$ | $z$ | $f$ |  |
|---|-----------------------------|-----|-----|-----|-----|--|---------|---|-----------------------------|-----|-----|-----|-----|--|
| 0 | $\overline{xyz}$            | 0   | 0   | 0   | 0   |  | 0       | 0 | $\overline{xyz}$            | 0   | 0   | 0   | 0   |  |
| 1 | $\overline{xy}z$            | 0   | 0   | 1   | 0   |  |         | 1 | $\overline{xy}z$            | 0   | 0   | 1   | 0   |  |
| 2 | $\overline{x}y\overline{z}$ | 0   | 1   | 0   | 1 * |  | 1       | 2 | $\overline{x}y\overline{z}$ | 0   | 1   | 0   | 1 * |  |
| 3 | $\overline{x}yz$            | 0   | 1   | 1   | 1 * |  |         | 4 | $x\overline{y}\overline{z}$ | 1   | 0   | 0   | 0   |  |
| 4 | $x\overline{y}\overline{z}$ | 1   | 0   | 0   | 0   |  |         | 3 | $\overline{x}yz$            | 0   | 1   | 1   | 1 * |  |
| 5 | $x\overline{y}z$            | 1   | 0   | 1   | 1 * |  | 2       | 5 | $x\overline{y}z$            | 1   | 0   | 1   | 1 * |  |
| 6 | $xy\overline{z}$            | 1   | 1   | 0   | 0   |  |         | 6 | $xy\overline{z}$            | 1   | 1   | 0   | 0   |  |
| 7 | $xyz$                       | 1   | 1   | 1   | 1 * |  | 3       | 7 | $xyz$                       | 1   | 1   | 1   | 1 * |  |

| Livello |   |                             | $x$ | $y$ | $z$ | $f$ |  |
|---------|---|-----------------------------|-----|-----|-----|-----|--|
| 1       | 2 | $\overline{xy}\overline{z}$ | 0   | 1   | 0   | 1 * |  |
| 2       | 3 | $\overline{x}yz$            | 0   | 1   | 1   | 1 * |  |
|         | 5 | $x\overline{y}\overline{z}$ | 1   | 0   | 1   | 1 * |  |
| 3       | 7 | $xyz$                       | 1   | 1   | 1   | 1 * |  |

Tabella 27: Trasformazione della tavola di verità della funzione logica  $f$  per ottenere la tabella di Quine-Mc Cluskey

La cui semplificazione per via algebrica porta a:

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

È ben evidente che la struttura della semplificazione  $\bar{x}y\bar{z} + \bar{x}yz = \bar{x}y(\bar{z} + z)$ , che riguarda i termini minimi contrassegnati con \* in tabella, e del tipo  $f\bar{z} + fz = f$  e riguarda due termini minimi che differiscono per la sola variabile  $z$ , che si trova diretta in uno dei due termini e negata nell'altro. La stessa cosa si può dire per i due termini minimi contrassegnati con \*.

Si procede, quindi, all'ordinamento delle righe della tabella in modo da mettere su livelli adiacenti le n-uple che differiscono per una sola variabile.

Una possibilità è quella di procedere per peso crescente delle n-uple binarie associate ai termini minimi (dove per peso di un'n-upla è da intendersi il numero di 1 in essa presenti) in modo che a ciascun peso corrisponda un livello; così facendo si ottiene la seconda tabella riportata.

Se ora si eliminano le righe che corrispondono a termini minimi per i quali la funzione vale 0 (che non rientrano nella somma di prodotti) si ottiene la terza tabella di figura, che corrisponde alla tabella di Quine-Mc Cluskey.

A questo punto si può procedere con le semplificazioni, che generano una seconda tabella. Poiché  $\bar{x}y\bar{z}$  si semplifica con  $\bar{x}yz$  per ottenere  $\bar{x}y$ , su questa tabella vengono contrassegnate con una lineetta la variabile  $z$  che è stata oggetto di semplificazione; vengono, inoltre, contrassegnati a lato i termini minimi che sono coinvolti nella semplificazione, cioè 2 - 3; viceversa la coppia 2 - 5 non porta ad alcuna semplificazione. Esaurito il confronto tra i livelli 1 e 2, è possibile ora controllare le semplificazioni tra i livelli 2 e 3; in questo caso si trovano semplificazioni per entrambe le coppie, cioè 3 - 7 e 5 - 7.

La tabella che si ottiene è la seguente:

|       | $x$ | $y$ | $z$ |     |
|-------|-----|-----|-----|-----|
| 2 – 3 | 0   | 1   | –   | $A$ |
| 3 – 7 | –   | 1   | 1   | $B$ |
| 5 – 7 | 1   | –   | 1   | $C$ |

Tabella 28: Tabella di semplificazione

Poiché non è possibile fare altre semplificazioni ci si ferma, contrassegnando con delle lettere progressive gli implicant che non si possono più semplificare.

L'espressione finale deriva dalla somma degli implicant coinvolti,  $f = A + B + C = \bar{x}y + yz + xz$ . Si osservi tuttavia che l'espressione che otteniamo non è in forma minima, poiché come noto il termine  $yz$  si può eliminare per il terzo teorema dell'assorbimento.

Infatti, ciò si può facilmente osservare anche attraverso la realizzazione della mappa di Karnaugh della funzione considerata

| $xy \backslash z$ | 00 | 01 | 11 | 10 |
|-------------------|----|----|----|----|
| 0                 | 0  | 1  | 0  | 0  |
| 1                 | 0  | 1  | 1  | 1  |

Per individuare il minimo numero di implicant che implicano tutti i termini minimi è necessario costruire il **reticolo del metodo**, disponendo sulle righe gli implicant, sulle colonne i termini minimi, e ponendo un segno (p.es. un pallino) in corrispondenza dei termini minimi che formano ciascun impicante.

Infatti si osserva che:

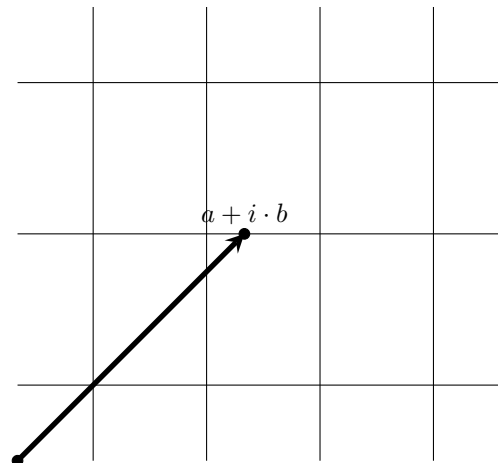


Figura 19: Modulo di un numero complesso

La mappa di Karnaugh corrispondente è

| $z \backslash xy$ |   | 00 | 01 | 11 | 10 |
|-------------------|---|----|----|----|----|
|                   |   | 0  | 1  | 0  | 0  |
| 0                 | 1 | 1  | 1  | 0  | 0  |
| 1                 | 1 | 1  | 1  | 1  | 0  |

... continua ... Tali realizzazioni si comprendono non appena si valuta attentamente la mappa di Karnaugh associata alla funzione a  $n = 4$  variabili:

| $zw \backslash xy$ |   | 00 | 01 | 11 | 10 |
|--------------------|---|----|----|----|----|
|                    |   | 0  | 1  | 1  | 0  |
| 00                 | 0 | 0  | 1  | 1  | 0  |
| 01                 | 1 | 1  | 0  | 1  | 1  |
| 11                 | 1 | 1  | 1  | 1  | 1  |
| 10                 | 0 | 0  | 1  | 1  | 1  |

Figura 20: Mappa di Karnaugh a 16

## 3 Circuiti combinatori

### 3.1 Introduzione

Nella realtà pratica, l'Algebra Booleana non sempre viene impiegata per raggiungere la **forma minima** di una funzione logica, in quanto se così si procedesse non si potrebbe sfruttare il **fattore di scala**.

Infatti, quando si deve realizzare una rete logica, molto spesso è più conveniente impiegare dei **moduli precostituiti**, anche ridondanti, piuttosto che richiedere la realizzazione di un modulo *custom*, costruito appositamente per lo scopo.

In questi termini si parla di **circuiti combinatori**. Si chiamano combinatori quei circuiti dotati di uno o più ingressi e uno o più uscite, il cui funzionamento è descritto da una funzione logica; in questi circuiti gli ingressi e le uscite possono assumere solo uno di due valori binari previsti (0 e 1); inoltre in ogni istante l'uscita è funzione deterministica unicamente degli ingressi. La figura seguente illustra la struttura generale di un circuito combinatorio con  $n$  ingressi e  $m$  uscite, il cui funzionamento è descritto dalla funzione  $F : 2^n \rightarrow 2^m$ , che è una funzione Booleana da  $2^n$  a  $2^m$  realizzata mediante  $m$  funzioni Booleane  $f_i : 2^n \rightarrow 2$ :

$$y_1 = f_1(x_1, x_2, \dots, x_n)$$

$$y_2 = f_2(x_1, x_2, \dots, x_n)$$

...

$$y_m = f_m(x_1, x_2, \dots, x_n)$$

#### 3.1.1 Itinerari e livelli

Ogni rete logica è formata da un certo numero di porte (AND, OR, ecc.) tra loro variamente interconnesse, da un certo numero di ingressi, contraddistinti in figura seguente con i simboli  $A_i$ , e da un certo numero di uscite, contraddistinte nella medesima figura con i simboli  $B_k$ .

Si dice itinerario tra due elementi  $X$  e  $Y$  qualsiasi

25 Ottobre 2021

### 3.2 Riassunto

Il metodo di semplificazione di Quine Mc Cluskey si basa su una semplice proprietà:

$$f \cdot x + f \cdot \bar{x} = f$$

Una volta che è nota la tabella di verità della funzione logica oggetto di interesse si rilevano tutte le combinazioni delle variabili in ingresso che producono il valore logico 1 e si classificano tali combinazioni sulla base del peso, (ovvero “quanti 1 sono presenti in ciascuna combinazione delle variabili in ingresso”).

Classificate le combinazioni in ingresso si confrontano tutte le combinazioni di ciascun peso con quelle di tutti gli altri pesi al fine di verificare se vi sono delle coppie di variabili che si conservano e che, quindi, si possono semplificare, in modo tale da raggiungere la forma minima, maggiormente semplificata.

Quando non si può più procedere ad un’ulteriore semplificazione si è, di fatto, ottenuto l’insieme di tutti gli **implicanti** che non significa che costituiscono la forma minima, ma possono essere ancora ridondanti.

Infatti, gli implicanti ottenuti tramite la tabella di Quine Mc Cluskey devono essere analizzati ulteriormente tramite il cosiddetto **reticolo**, al fine di determinare la **copertura essenziale e minima**, e mai sovrabbondante.

Infatti, realizzando successivamente la mappa di Karnaugh ci si rende conto, effettivamente, della presenza di sottocubi non primi che vengono comunque ottenuti anche attraverso il metodo di Quine Mc Cluskey e che può essere arginato attraverso il reticolo, così come nella mappa di Karnaugh si osservava che alcuni *minterm* erano già stati “coperti” da altri sottocubi precedentemente individuati.

**Osservazione:** Si presti particolare attenzione che, nella fase di confronto e di semplificazione, ogni qualvolta si semplificano dei *minterm*, questi vengono “spuntati”, per tenere conto che sono stati considerati.

Se, infatti, un *minterm* non viene considerato in fase di semplificazione allora bisognerà tenerne conto, in quanto diventerà un **implicante** nella fase finale di semplificazione (per quanto possa essere modesto, in quanto implicherà solamente se stesso).

**Esercizio:** Si consideri la seguente funzione logica:

Si consideri una nuova funzione a  $n = 3$  variabili

| $x$ | $y$ | $z$ | $f(x, y, z)$ |
|-----|-----|-----|--------------|
| 0   | 0   | 0   | 1            |
| 0   | 0   | 1   | 1            |
| 0   | 1   | 0   | 0            |
| 0   | 1   | 1   | 1            |
| 1   | 0   | 0   | 1            |
| 1   | 0   | 1   | 0            |
| 1   | 1   | 0   | 1            |
| 1   | 1   | 1   | 1            |

Tabella 29: Funzione a  $n = 3$  variabili

La realizzazione della mappa di Karnaugh si ottiene

| $xy$ |   | 00 | 01 | 11 | 10 |
|------|---|----|----|----|----|
| $z$  | 0 | 1  | 0  | 1  | 1  |
|      | 1 | 1  | 1  | 1  | 0  |

Oppure, equivalentemente

| $xy$ |   | 00 | 01 | 11 | 10 |
|------|---|----|----|----|----|
| $z$  | 0 | 1  | 0  | 1  | 1  |
|      | 1 | 1  | 1  | 1  | 0  |

Procedendo, invece, tramite il metodo tabella di Quine Mc Clusky si ottiene

| Livello |   |                  | $x$ | $y$ | $z$ | $f$ |
|---------|---|------------------|-----|-----|-----|-----|
| 0       | 0 | $\overline{xyz}$ | 0   | 0   | 0   | 1   |
| 1       | 1 | $\overline{xy}z$ | 0   | 0   | 1   | 1   |
|         | 4 | $x\overline{y}z$ | 1   | 0   | 0   | 1   |
| 2       | 3 | $\overline{x}yz$ | 0   | 1   | 1   | 1   |
|         | 6 | $xy\overline{z}$ | 1   | 1   | 0   | 1   |
| 3       | 7 | $xyz$            | 1   | 1   | 1   | 1   |

Tabella 30: Tabella di Quine Mc Cuskley

Da cui si ottiene

|       | $x$ | $y$ | $z$ |     |
|-------|-----|-----|-----|-----|
| 0 – 1 | 0   | 0   | –   | $A$ |
| 0 – 4 | –   | 0   | 0   | $B$ |
| 1 – 3 | 0   | –   | 1   | $C$ |
| 4 – 6 | 1   | –   | 0   | $D$ |
| 3 – 7 | –   | 1   | 1   | $E$ |
| 6 – 7 | 1   | 1   | –   | $F$ |

Tabella 31: Tabella di semplificazione

La realizzazione del reticolo conduce al seguente risultato

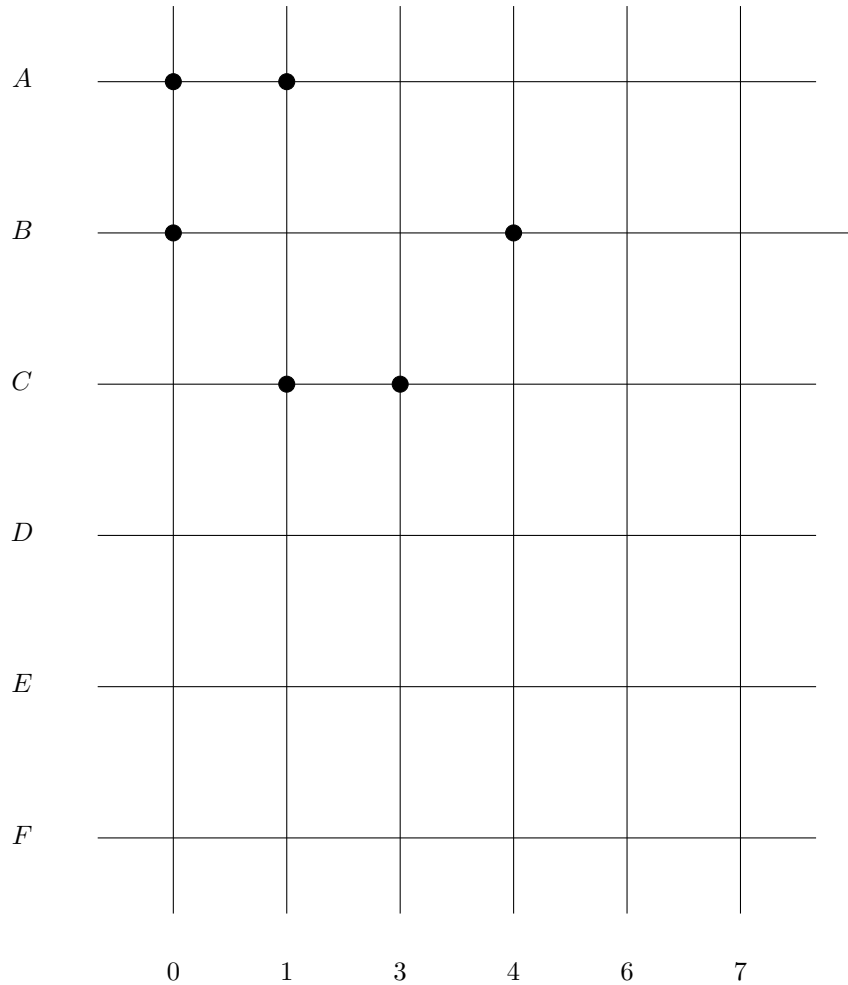


Figura 21: Modulo di un numero complesso

Da cui, tramite la creazione de reticolo, si possono ottenere le due possibili funzioni

$$f_1 = x\bar{z} + \bar{x}y + yz$$

$$f_2 = xy + \bar{y}\bar{z} + \bar{x}z$$

ovvero

$$0 - 1 + 4 - 6 + 3 - 7$$

e

$$0 - 4 + 1 - 3 + 6 - 7$$

I circuiti combinatori devono essere studiati attraverso due procedure

- L'**analisi logica**, al fine di determinarne il funzionamento e i componenti;
- La **sintesi logica**, al fine di ottimizzarne il funzionamento o semplicemente modificarlo.

Ogni rete logica e formata da un certo numero di porte (AND, OR, ecc.) tra loro variamente interconnesse, da un certo numero di ingressi, contraddistinti con i simboli  $A_i$ , e da un certo numero di uscite, contraddistinte con i simboli  $B_k$ .

Si dice **itinerario** tra due elementi  $X$  e  $Y$  qualsiasi il percorso che colleghi  $X$  con  $Y$ .

Si dice invece **livello di un elemento  $X$  rispetto all'uscita  $B_j$**  e a un determinato itinerario  $I$ , il numero di elementi,  $X$  compreso, disposti lungo l'itinerario a partire dall'uscita  $B_j$ .

Si dice **livello di una variabile rispetto all'uscita  $B_j$  e all'itinerario  $I$**  il numero di elementi



compresi tra il rispettivo ingresso e l'uscita  $B_j$  lungo l'itinerario  $I$ .

**Osservazione:** La presenza di più livelli all'interno di un circuito è molto critica, in quanto più livelli introducono più latenze che potrebbero determinare una scorretta analisi degli ingressi logici, in quanto le porte logiche presenti in un circuito introducono un tempo di propagazione che non necessariamente è trascurabile.

Tali ritardi prendo il nome di **alee**, ovvero dei ritardi che non sono ammissibili, in quanto le variabili di uscita devono essere processati senza eterogeneità di ritardi in ingresso.

### 3.3 Analisi dei circuiti combinatori

L'analisi di un circuito combinatorio tende a ottenere una rappresentazione della funzione d'uscita  $y$  o nella sua forma analitica, oppure sotto forma di tavola di verità. Poiché la rappresentazione circuitale è simbolica, l'analisi non è legata a considerazioni di logica positiva o negativa.

Per effettuare l'analisi, nel caso di circuiti AND-OR-NOT, è sufficiente partire dagli elementi su cui entrano le variabili e procedere verso il terminale di uscita secondo tutti i possibili itinerari, usando la funzione di uscita di ciascun elemento come variabile di ingresso dell'elemento successivo.

L'analisi dei circuiti basati su porte NAND e NOR è notevolmente più complessa, a causa della non associatività degli operatori.

### 3.4 Sintesi dei circuiti combinatori

Eseguire la sintesi di un circuito combinatorio consiste, come già accennato, nel progettare un circuito a  $n$  ingressi che soddisfi una determinata funzione di uscita  $y$  di progetto. La funzione  $y$  che il circuito deve soddisfare può essere assegnata in diverse forme. precisamente:

- con la **descrizione a parole** del funzionamento del circuito. È questa la forma di assegnazione più comune, ma anche la più imprecisa. È necessario porre un'estrema attenzione alla corretta interpretazione di eventuali condizioni implicite e all'esistenza di vincoli di qualsiasi natura. Dalla descrizione verbale è necessario passare poi alla tavola di verità, assegnando il valore della funzione per ognuna delle  $2^n$  configurazioni degli ingressi;
- con una vera e propria **tavola di verità**, che è in definitiva l'effettivo punto di partenza della sintesi cui tutti gli altri tipi di assegnazione devono essere ricondotti;
- con un'**espressione analitica**, che è il modo più conciso, anche se non univoco, di descrivere il funzionamento di un circuito;
- con uno **schema logico**, che è una procedura generalmente usata quando un determinato circuito logico debba essere riprogettato con componenti diversi. In tal caso, con le regole dell'analisi si ricava un'espressione analitica della funzione  $y$ .

Qualunque sia il metodo di assegnazione, la sintesi procede partendo dalla tavola di verità o da un'espressione analitica; applicando i metodi di semplificazione delle funzioni logiche si perviene alla forma più conveniente per gli scopi che ci si propone.

Si noti che non sempre la forma più conveniente corrisponde alla forma minima della funzione. Ad esempio non è sempre opportuno realizzare circuitualmente la forma minima algebrica, in quanto vi possono essere dei vincoli sul numero massimo di livelli. Infatti, se è ben vero quanto esposto precedentemente, e cioè che nei circuiti combinatori l'uscita è, istante per istante, funzione unicamente degli ingressi, non significa che la variazione degli ingressi sia avvertita immediatamente in uscita; tale affermazione sta piuttosto a significare che ogni configurazione di ingresso dà luogo a una determinata uscita e che eventuali transitori di commutazione possono ritardare, ma non modificare questa uscita.

Poiché il tempo di commutazione  $\Delta$  di qualsiasi porta logica, per quanto piccolo, non è mai nullo, il tempo di risposta di un circuito a  $n$  livelli, al variare della configurazione d'ingresso, è  $n \cdot \Delta$ .

In definitiva, il ritardo totale tra ingresso e uscita è proporzionale al numero di livelli e potendo la forma minima di una funzione contenere un numero di livelli molto elevato, la sua diretta realizzazione circuitale potrebbe dar luogo a ritardi intollerabili.

La forma in cui si ha il minimo ritardo è quella a due livelli, che d'altra parte è quella che si ottiene con i metodi di semplificazione che sono stati esposti in precedenza.

Inoltre si osservi che, molto spesso, anche dal punto di vista economico, è più conveniente introdurre nel circuito delle componenti ridondanti piuttosto che personalizzate al fine di ottimizzare anche i costi di produzione della propria rete logica. La convenienza di eventuali fattorizzazioni va valutata caso per caso. Si può dunque concludere che la sintesi di un circuito combinatorio procede attraverso i seguenti passi:

1. Descrizione del funzionamento del circuito
2. Determinazione della tavola di verità
3. Sintesi della funzione Booleana
4. Semplificazione della funzione logica relativa
5. Determinazione della forma minima più conveniente
6. Disegno del circuito

Si osservi che il passo 5 non può essere attuato secondo un procedimento sistematico, e l'effettiva forma minima più conveniente andrà valutata di volta in volta, eventualmente facendo uso di tecniche basate sul concetto di decomponibilità.

**Esempio:** Si voglia realizzare un circuito a tre ingressi e quattro uscite; sugli ingressi si può presentare un numero binario compreso tra 0 e 5. All'uscita di tale circuito si deve ottenere il prodotto per 3 del numero in ingresso.

Il massimo numero di uscita rappresentabile con 4 bit è 15 e sarà necessario sintetizzare quattro funzioni logiche.

| $3 \cdot \mathbf{x} = \mathbf{y}$ | $x$   |       |       | $y$   |       |       |       |
|-----------------------------------|-------|-------|-------|-------|-------|-------|-------|
|                                   | $x_2$ | $x_1$ | $x_0$ | $y_3$ | $y_2$ | $y_1$ | $y_0$ |

Tabella 32: Caption

Se il circuito viene impiegato rispettando le specifiche di progetto, allora il circuito funzionerà come previsto. Altrimenti il circuito risponde erroneamente, rispondendo in modo irregolare e scorretto.

**Esempio:** Si voglia sintetizzare un circuito con la stessa funzione Booleana di quello illustrato in figura 5.9a, ma possibilmente più economico.

Si ottiene

$$y_1 = \overline{x_1} + \overline{x_3} + x_4 + \dots$$

26 Ottobre 2021

### 3.5 Riassunto

I circuiti combinatori sono dei dispositivi che permettono di realizzare delle funzioni booleane della forma

$$F : 2^n \rightarrow n$$

Le modalità per realizzare tali funzioni sono di due tipi:

- **Logica positiva:** Al valore 1 logico viene associato il valore di tensione alto, a 0 il valore di tensione basso.
- **Logica negativa:** Al valore 1 logico viene associato il valore di tensione basso, a 0 il valore di tensione alto.

I circuiti che vengono realizzati possono presentare più livelli di itinerario che può comportare una latenza, ovvero un ritardo di propagazione al variare delle variabili in ingresso.

Pertanto, è possibile che ad una porta giunga il valore della variabile  $y_1$  istantaneamente, mentre il valore della variabile  $y_2$  giunge con un ritardo di  $2\Delta$ , ove  $\Delta$  è il tempo di assestamento di una porta logica, variabile a seconda della tecnologica impiegata.

Se tali variazioni sono infinitesime, possono essere trascurati, ma se non sono trascurabili determinano una instabilità della rete, a causa di continue ed irregolari oscillazioni di valori delle medesime variabili.

Tale fenomeno prende il nome di **Alea** ed è cruciale nella creazione delle porte logiche.

Lo studio e la manipolazione dei circuiti logici prevede due operazioni

- Lo **studio della rete logica:** ovvero determinare ed estrapolare la funzione logica a partire dalla rete logica realizzata.
- La **sintesi della rete logica:** Semplificare un circuito o riprogettarlo con delle nuove porte, oppure con un diverso numero di livelli a seguito della constatazione della presenza di ritardi e conseguente instabilità del circuito.

Eseguire la sintesi di un circuito combinatorio consiste, come già accennato, nel progettare un circuito a  $n$  ingressi che soddisfi una determinata funzione di uscita  $y$  di progetto. La funzione  $y$  che il circuito deve soddisfare può essere assegnata in diverse forme.

Più precisamente:

- con la **descrizione a parole** del funzionamento del circuito. È questa la forma di assegnazione più comune, ma anche la più imprecisa. È necessario porre un'estrema attenzione alla corretta interpretazione di eventuali condizioni implicite e all'esistenza di vincoli di qualsiasi natura. Dalla descrizione verbale è necessario passare poi alla tavola di verità, assegnando il valore della funzione per ognuna delle  $2^n$  configurazioni degli ingressi;
- con una vera e propria **tavola di verità**, che è in definitiva l'effettivo punto di partenza della sintesi cui tutti gli altri tipi di assegnazione devono essere ricondotti;
- con un'**espressione analitica**, che è il modo più conciso, anche se non univoco, di descrivere il funzionamento di un circuito;
- con uno **schema logico**, che è una procedura generalmente usata quando un determinato circuito logico debba essere riprogettato con componenti diversi. In tal caso, con le regole dell'analisi si ricava un'espressione analitica della funzione  $y$ .

Indipendentemente dalla modalità di assegnazione della funzione logica, si deve sempre giungere ad una tabella di verità, da cui si può ricavare la mappa di Karnaugh.

Dopodiché si dovrà verificare caso per caso se sia effettivamente più conveniente creare un circuito attraverso la semplificazione massimale, in quanto molto spesso il costo di un circuito "customizzato" è più elevato rispetto a moduli standard e precostituiti.

**Esempio:** Si voglia realizzare un circuito a tre ingressi e quattro uscite; sugli ingressi si può presentare un numero binario compreso tra 0 e 5. All'uscita di tale circuito si deve ottenere il

prodotto per 3 del numero in ingresso.

Il massimo numero di uscita rappresentabile con 4 bit è 15 e sarà necessario sintetizzare quattro funzioni logiche.

|                                   | $x$   |       |       | $y$   |       |       |       |
|-----------------------------------|-------|-------|-------|-------|-------|-------|-------|
| $3 \cdot \mathbf{x} = \mathbf{y}$ | $x_2$ | $x_1$ | $x_0$ | $y_3$ | $y_2$ | $y_1$ | $y_0$ |

Tabella 33: Caption

In cui, naturalmente, le combinazioni in ingresso che non possono essere considerate dalla rete logica costituita, in quanto produrrebbero in uscita un valore non rappresentabile con soli 4 bit, costituiranno delle **condizioni non specificate**, ovvero delle *don't care conditions* e che verranno impiegate proficuamente nella creazione dei sottocubi nella mappa di Karnaugh.

**Esempio:** Si voglia costruire un circuito che esegue la moltiplicazione tra due numeri espressi in notazione posizionale in base 2.

Siano  $\mathbf{x} = x_1 \cdot x_2$  i bit del moltiplicando e  $\mathbf{y} = y_1 \cdot y_2$  i bit del moltiplicatore; ciascun fattore varia dunque nell'intervallo  $[0, 3]$  mentre il prodotto è compreso nell'intervallo  $[0, 9]$ .

Il progetto richiede la costruzione della tavola di verità per tutte le possibili  $2^2 \cdot 2^2 = 16$  combinazioni, tenendo conto che per rappresentare il valore del prodotto serviranno  $\lceil \log_2(9) \rceil = 4$  bit. Se, per esempio, si ha  $\mathbf{x} = 10$ , ovvero 2 in decimale e  $\mathbf{y} = 11$ , ovvero 3 in decimale, a questa configurazione d'ingresso deve corrispondere  $\mathbf{z} = 0110$ , ovvero 6 in decimale, considerato che  $2 \cdot 3 = 6$ .

Operando in questo modo si ottiene la tavola di verità seguente nella quale ogni colonna  $i$ -esima rappresenta i valori assunti dalla funzione  $z$  in corrispondenza delle varie quaterne d'ingresso.

Si devono, allora, valutare quattro funzioni Booleane del tipo  $2^2 \rightarrow 2$ . Poiché ci sono pochi 1 e molti 0, conviene usare la I forma canonica, basata sui termini minimi.

### 3.6 Moduli combinatori

L'algebra Booleana permette di analizzare e progettare qualunque tipo di rete combinatoria, di qualunque complessità. In precedenza abbiamo mostrato anche le tecniche per semplificare le reti ottenute direttamente dalle espressioni delle forme canoniche, basate sulla somma di prodotti o sui prodotti di somme. Sebbene in linea di principio quei metodi possono essere utilizzati per reti combinatorie di qualunque estensione, nella pratica si preferisce affrontare il progetto di una rete complessa secondo una tecnica di scomposizione della rete in blocchi funzionali, detti moduli combinatori. In questo modo si rinuncia alla soluzione teoricamente ottima, a favore di una maggiore comprensibilità, gestibilità e soprattutto modularità del progetto. Ciò porta anche a un effettivo risparmio economico, poiché i moduli combinatori, se prodotti in larga scala, hanno prezzi via via sempre più bassi; dunque non sempre un numero minore di porte implica un risparmio economico. Già a partire dagli anni '70 vennero prodotti moduli integrati, che svolgendo precise funzionalità di carattere combinatorio trovarono impiego nello sviluppo di progetti complessi. Nella parte che segue vengono esaminati alcuni moduli combinatori di uso corrente.

#### 3.6.1 Decodificatori

I **decodificatori** convertono un numero espresso in notazione posizionale in base 2 in un numero intero in base 10. Un decodificatore accetta in ingresso  $n$  bit e presenta in uscita  $m = 2^n$  linee, numerate da 0 a  $2^n - 1$ , in modo tale che va a 1 la sola linea  $y_j$  che corrisponde all'intero  $j$  codificato in notazione posizionale dagli  $n$  bit d'ingresso. Se dunque si indicano con  $y_0, y_1, \dots, y_m$  le uscite del decodificatore, la generica uscita  $y_j$  ottenuta come AND delle  $n$  variabili che compongono il *minterm*  $j$ .

Nelle figure 5.14a e 5.14b vengono riportati, rispettivamente, la tavola di verità e il circuito di un decodificatore a 2 bit; analogamente, nelle figure 5.15a e 5.15b si presentano la tavola di verità e il circuito di un decodificatore a 3 bit.

In figura 5.14c viene invece rappresentato il simbolo circuitale per un generico decodificatore a  $n$  bit.

**Osservazione:** Si noti che questo processo non deve creare confusione. Infatti, essendo i calcolatori binari, si sarebbe portati a rappresentare ogni valore, in ingresso come in uscita, sfruttando la logica binaria. Ma allora, se si hanno  $n$  bit in ingresso, che possono produrre  $2^n$  combinazioni di valori, che corrispondono a  $2^n$  numeri decimali. Allora, sarà sufficiente attivare solamente l'uscita corrispondente e si è ottenuto il valore da decodificare.

### 3.6.2 Codificatore

I codificatori convertono un numero intero in base 10 in un numero espresso in notazione posizionale in base 2.

Un codificatore svolge dunque la funzione inversa di un decodificatore, nel senso che esso prevede  $m = 2^n$  ingressi e  $n$  uscite. Le uniche configurazioni ammesse per gli ingressi sono quelle in cui c'è esattamente un solo 1 in corrispondenza della linea  $x_j$ ; dunque  $x_j = 1$  e  $x_i = 0$  per ogni  $i \neq j$ .

Indicando con  $x_0, x_1, \dots, x_{m-1}$  gli ingressi, la corrispondente configurazione di uscita sulle variabili  $y_{n-1}, y_{n-2}, \dots, y_0$  è tale che esse codificano  $j$  notazione posizionale in base 2.

Nelle figure 5.16a e 5.16b vengono riportate, rispettivamente, la tavola di verità e le due mappe di Karnaugh associate alle variabili  $y_1$  e  $y_0$ . Si noti che nella tabella di verità sono state riportate le sole configurazioni di ingresso definite; le altre danno luogo a **condizioni non specificate**, che consentono un amplissimo margine di libertà nella realizzazione effettiva del circuito. Ciò è evidente dalle mappe di Karnaugh di figura 5.16b, che consentono almeno 2 soluzioni; la prima implica l'uso di due porte OR e la seconda due porte AND e due NOT. Nella figura 5.17a e 5.17b vengono riprodotti, rispettivamente, il circuito del codificatore a 2 bit associato alla soluzione con le porte OR e il simbolo circuitale di un generico codificatore a  $n$  bit. Analogamente, nelle figure 5.18a e 5.18b si presentano la tavola di verità e il circuito di un codificatore a 3 bit.

### 3.6.3 Selettori

Un **selettore d'ingresso (o Multiplexer o Mux)** è un modulo che permette di selezionare uno tra  $2^n$  ingressi e presentarlo sull'unica uscita. La selezione si effettua attraverso  $n$  linee di comando. Nella figura 5.19a viene fornita la tavola di verità del Multiplexer a 2 vie (2 : 1 Mux), che si ricava tenendo conto che per  $s_0 = 0$  viene riportato in uscita il contenuto di  $x_0$ , mentre per  $s_0 = 1$  si porta in uscita il contenuto di  $x_1$ ; la semplificazione tramite minterm di figura 5.19b porta alla soluzione circuitale di figura 5.19c.

Nelle figure 5.20a, 5.20b sono invece rappresentati i Multiplexer a 4 e 8 vie, mentre la figura 5.20c illustra il simbolo circuitale di un generico Multiplexer a  $n$  vie.

La mappa di Karnaugh corrispondente sarà

|     |   |      |    |    |    |
|-----|---|------|----|----|----|
|     |   | $xy$ |    |    |    |
|     |   | 00   | 01 | 11 | 10 |
| $z$ | 0 | 0    | 0  | 1  | 1  |
|     | 1 | 0    | 1  | 1  | 0  |

Che permette di individuare i due determinanti, ovvero

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

ove  $s$  è il selettore preso in esame, che si comporta come previsto:

- Se  $s = 0 \rightarrow y = x_0$
- Se  $s = 1 \rightarrow y = x_1$

### 3.6.4 Selettore d'uscita

Un selettore d'uscita (o Demultiplexer o Demux) è al contrario un dispositivo che permette di dirottare l'unico ingresso su una delle possibili  $2^n$  uscite.

La figura 5.21 mostra un Demultiplexer a 2 vie (1 : 2 Demux), uno a 4 vie (1 : 4 Demux) e uno a

8 vie (1 : 8 Demux), oltre al simbolo grafico di un generico selettore d'uscita a  $2^n$  vie.

Si noti che nel caso di selettore di uscita l'ingresso viene presentato sulla via selezionata, mentre tutti gli altri ingressi sono a 0.

In entrambi i selettori (d'ingresso e d'uscita) l'n-upla binaria della linea *Sel* seleziona quale, tra le  $2^n$  linee, è interessata alla connessione con l'uscita (l'ingresso).

27 Ottobre 2021

Si consideri un **display 7 segmenti** e si realizzi un sistema per rappresentare dei numeri attraverso proprio 7 diodi emettitori di luce, contrassegnati dalle lettere da **a** a **g**,  
Si ottiene

| $x$ | $y$ | $z$ | $w$ | a | b | c | d | e | f | g |
|-----|-----|-----|-----|---|---|---|---|---|---|---|
| 0   | 0   | 0   | 0   | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0   | 0   | 0   | 1   | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0   | 0   | 1   | 0   | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0   | 0   | 1   | 1   | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0   | 1   | 0   | 0   | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0   | 1   | 0   | 1   |   |   |   |   |   |   |   |
| 0   | 1   | 1   | 0   |   |   |   |   |   |   |   |
| 0   | 1   | 1   | 1   |   |   |   |   |   |   |   |
| 1   | 0   | 0   | 0   |   |   |   |   |   |   |   |
| 1   | 0   | 0   | 1   |   |   |   |   |   |   |   |
| 1   | 0   | 1   | 0   |   |   |   |   |   |   |   |
| 1   | 0   | 1   | 1   |   |   |   |   |   |   |   |
| 1   | 1   | 0   | 0   |   |   |   |   |   |   |   |
| 1   | 1   | 0   | 1   |   |   |   |   |   |   |   |
| 1   | 1   | 1   | 0   | – | – | – | – | – | – | – |
| 1   | 1   | 1   | 1   | – | – | – | – | – | – | – |

Tabella 34: Tabella di verità display 7 segmenti

Si realizzino, ora, le mappe di Karnaugh corrispondenti per ciascuna delle 7 funzioni di uscita, al fine di semplificare la funzione logica associata.

Si parta dalla funzione logica  $a$ , ovvero:

| $xy$    | 00 | 01 | 11 | 10 |
|---------|----|----|----|----|
| $zw$ 00 | 1  | 0  | X  | 1  |
| 01      | 0  | 1  | X  | 1  |
| 11      | 1  | 1  | X  | X  |
| 10      | 1  | 1  | X  | X  |

Figura 22: Mappa di Karnaugh a 16

Tale sistema poteva essere effettivamente realizzato anche sfruttando i **codificatori**. Infatti, com'è noto, un codificatore presenta in ingresso una sola linea a 1 corrispondente al numero naturale compreso tra  $[0, 9]$  che si vuole rappresentare. Il codificatore provvederà a rappresentare il numero in ingresso in base decimale nella base binaria. E le 4 uscite del codificatore rappresenteranno i 4 ingressi della rete logica che si è proprio realizzata in questo esercizio che, a sua volta, piloterà il display 7 segmenti.

Ciò agevolerà l'operazione di input che non sarà più composta da 4 linee, ma da una sola, quella del codificatore, che a sua volta provvederà a convertire il numero richiesto in binario.

### 3.7 Riassunto

I moduli combinatori sono stati introdotti per razionalizzare la realizzazione dei circuiti logici. I più importanti circuiti combinatori sono i seguenti:

- **Decodificatori**, che presentano  $n$  bit in ingresso e  $2^n$  bit in uscita, permettendo la rappresentazione di numeri in base 10 di numeri espressi in forma binaria. Naturalmente verrà attivata solamente 1 delle  $2^n$  possibili uscite del decodificatore per indicare il valore decimale corrispondente.
- **Codificatori**, che presentano  $2^n$  bit in ingresso e  $n$  bit in uscita, in cui solamente una linea in ingresso verrà portata ad 1 per indicare il valore da codificare, mentre saranno  $n$  le uscite che produrranno il valore decimale codificato in binario.  
Naturalmente, in questo caso, se vi sono  $2^n$  ingressi si potranno rappresentare solamente  $n$  valori binari, pertanto  $2^n - n$  ingressi saranno inutilizzati e costituiranno delle condizioni non specificate.
- **Multiplexer**, che presenta  $2^n$  ingressi e  $n$  selettori per selezionare, ovvero dirottare, uno dei  $2^n$  ingressi in un'unica uscita.
- **Demultiplexer**, che presentano un solo ingresso,  $2^n$  uscite e  $n$  selettori.

### 3.8 Costruzione modulare di una funzione Booleana

Un impiego particolarmente interessante dei selettori d'ingresso, o multiplexer: la costruzione di qualunque funzione Booleana di  $n$  variabili attraverso un selettore d'ingresso a  $2^n$  vie.

Una qualunque funzione di  $n$  variabili, com'è noto, può essere ricondotta a una somma di *minterm*. Allora, se si considera una funzione Booleana  $f : n \rightarrow 2^n$  con  $n = 3$  e  $2^n = 8$  uscite, sarà sufficiente porre in ingresso al multiplexer a 8 vie le 8 costanti dei valori della funzione, mentre gli  $n = 3$  selettori comanderanno quale degli 8 ingressi dovrà essere dirottato in uscita.

Tuttavia, si capisce bene che un multiplexer a  $2^n$  vie presenta  $2^n$  porte AND e 1 porta OR di uscita, che sono sicuramente maggiori delle porte che si impiegherebbe per realizzare la medesima funzione logica riducendola in forma minima.

Ma allo stesso tempo si osservano i significativi vantaggi:

1. Anche se il circuito è molto complesso, garantisce una straordinaria versatilità
2. Un multiplexer è un dispositivo molto comune e prodotto **in larga scala**, quindi il suo costo sarà inferiore rispetto a quello di un circuito personalizzato e costruito appositamente per il proprio scopo.  
Si consideri, inoltre, che se non si volesse realizzare un circuito con esclusivamente le porte ottenute tramite semplificazione si dovrebbe ripiegare su circuiti integrati con un numero di porte AND e OR che, in ogni caso, sono ormai standardizzati. Pertanto, nella realizzazione del circuito con questa modalità, si avrebbe un risparmio solamente apparente.

**Osservazione:** C'è tuttavia un metodo più economico per costruire, in modo modulare, una qualunque funzione Booleana di  $n$  variabili. Facendo sempre riferimento alla I forma canonica, la sua realizzazione richiede un numero di porte AND a  $n$  ingressi pari al numero di minterm (cioè  $2^n$ ) e una porta OR con un numero di ingressi pari al numero di porte AND (cioè sempre  $2^n$ ). Si tratta allora di "prolungare" le uscite delle porte AND agli ingressi della porta OR per i soli minterm che entrano nella funzione.

La figura 5.24a mostra un circuito di questo genere, chiamato matrice di contatti, nel quale si evidenzia la presenza di un contatto per i soli minterm che servono per realizzare la funzione  $z = x_1 x_2 + x_1 \bar{x}_2$ ; nella pratica circuitale la configurazione diventa quella di figura 5.24b. Se ora vogliamo usare questa tecnica per costruire la funzione (5.4) otteniamo il circuito di figura 5.25a. La sua struttura è estremamente vantaggiosa se viene realizzata direttamente dal costruttore di integrati, che privilegia un'alta uniformità nella struttura circuitale. Il confronto con la soluzione



a selettori di figura 5.23 e chiarificatore: in quel caso, per realizzare una qualunque ‘ funzione di tre variabili occorre un integrato che abbia almeno  $3 \cdot 8 \cdot 1 = 12$  piedini (a parte l’alimentazione e la massa). Con la soluzione ora illustrata il numero di piedini sarebbe pari a soli  $3 \cdot 1 = 4$ . La struttura di figura 5.25a fa uso di diodi per realizzare le connessioni della matrice; inoltre essa può essere estesa in modo tale ‘ da fornire non una, ma più funzioni di uscita, passando p.es. a una rete con ‘  $n$  ingressi e  $k$  uscite. Una simile rete fornisce, per ogni configurazione degli  $n$  ingressi, una configurazione sulle  $k$  uscite e viene a costituire quella che si chiama una memoria ROM, acronimo di Read Only Memory (memoria di sola lettura); la memoria in questione ha  $M = 2^n$  celle da  $k$  bit ciascuna, che vengono indirizzate dalle  $n$  linee di indirizzamento. Tenendo conto della realizzazione circuitale effettiva illustrata in figura 5.24b, riportiamo in figura 5.25b la struttura di una memoria ROM con 23 celle di memoria da 4 bit ciascuna. Nella parte destra della figura 5.25b viene riportato l’indirizzo di ciascuna cella e, tra parentesi, il suo contenuto. Per capirne il funzionamento supponiamo che si voglia leggere il contenuto della memoria all’indirizzo 3; cioè significa che deve essere  $x_2x_1x_0 = 011$ . In tal caso ci sarà una sola porta NAND che ha un’uscita bassa, la porta ‘ della riga 3, mentre tutte le altre porte NAND avranno uscita alta. I catodi dei diodi relativi alle due colonne  $z_3$  e  $z_0$  vanno allora a massa, inducendo una polarizzazione diretta per i corrispondenti diodi; sui rispettivi anodi si ha allora una tensione virtualmente nulla (in pratica ci sarà solo la ‘ VAK di saturazione, pari 0, 30, 6 V ). Le linee  $z_3$  e  $z_0$  sono allora basse, e a seguito della complementazione finale diventano alte. Gli altri diodi sulle colonne  $z_3$  e  $z_0$  risultano invece interdetti, le linee rimangono allo stato alto (‘VCC ) e quindi c’è 0 in uscita, a seguito della complementazione finale. Dunque  $z_3 = 1$  e  $z_0 = 1$ , mentre le altre due linee  $z_2$  e  $z_1$  si trovano a zero, e dunque  $z = 1001$ .

### 3.8.1 Diodi

L’idea alla base dell’ottimizzazione appena analizzata è costituita dai diodi. I diodi, in teoria, sono di due tipologie:

1. **Diodo a vuoto**, basata sulle ormai inutilizzate valvole termoioniche
2. **Diodo a giunzione**, che può essere a giunzione *PNP* oppure *NPN* a seconda del drogaggio a cui è sottoposto.

I diodi sono dei dispositivi che permettono di essere attraversati dalla corrente in un solo verso. Quando il diodo viene attraversato dalla corrente con **polarizzazione diretta** si ha un cortocircuito, quindi il diodo va in **interdizione**.

Se il diodo viene attraversato dalla corrente con **polarizzazione inversa** si ha un circuito aperto, quindi il diodo va in **saturazione**.

In figura 5.26 viene dato lo schema a blocchi di una memoria ROM; di fatto si tratta di un decodificatore seguito da una **matrice di contatti**. Il numero binario corrispondente alla combinazione degli  $n$  ingressi rappresenta l’indirizzo della corrispondente cella.

Si noti che fissare i contatti della matrice equivale a programmare il comportamento della rete. Nelle ROM propriamente dette, la matrice non ha inizialmente alcun punto di contatto tra righe e colonne. È il costruttore che esegue le connessioni in base alla matrice di bit desiderata dal committente. Per le memorie ROM, che sono appunto di sola lettura, non è possibile variare il contenuto della memoria dopo la programmazione fatta in fabbrica.

Le ROM risultano economicamente convenienti per volumi molto grandi. Nel caso debba essere l’utente a programmare la ROM si ricorre alle cosiddette PROM(Programmable Read Only Memory); la matrice ha inizialmente tutti i punti di contatto tra righe e colonne, attraverso un diodo e un fusibile.

8 Novembre 2021

### 3.9 Riassunto

Consuetamente, la costruzione di una funzione logica ha sempre previsto una minimizzazione massimale del numero di componenti, attraverso le mappe di Karnaugh e il metodo tabella di Quine-McCusky.

Tuttavia, molto spesso, non risulta essere conveniente operare in tal senso, ma è spesso molto più agevole impiegare dei circuiti precostituiti, ovvero dei moduli già pronti, che permette anche di abbattere i costi.

Pertanto, molto spesso si ricorre all'impiego di **multiplexer**, che permette di avere  $2^n$  ingressi e  $n$  selettori e produce in uscita un solo livello logico.

**Osservazione:** I selettori molto spesso prendono il nome di **tip-switch** (denominazione da verificare).

L'impiego di un multiplexer per la realizzazione di una funzione logica è molto pratico e agevole, e inoltre permette di avere una grande programmabilità e versatilità, cosa che non accade quando si definisce una semplificazione massima e si costruisce un circuito ad hoc per una specifica funzione logica.

Si osservi, inoltre, che un circuito di tale tipologia può essere ulteriormente ottimizzata semplicemente evitando di considerare gli ingressi a 0 e prolungando solamente gli ingressi che vengono portati a 1, andando così facendo a costituire una **matrice di contatti**.

Mappando, quindi, a livello circuitale, la funzione di ingresso si perde la programmabilità e la versatilità del circuito, procedendo, però, alla programmazione *in fabbrica* del circuito specificatamente per l'impiego che se ne richiede.

Per eseguire tali programmazioni è sufficiente intervenire su delle apposite tracce già presenti sul circuito e bruciare solamente i fusibili per i quali non si vuole il passaggio della corrente.

**Osservazione:** Quando si realizza logicamente una interconnessione fra le uscite delle porte AND e gli ingressi della porta OR si commette un errore concettuale, in quanto così facendo, se le uscite di due AND in serie sono a 0 e 1 si viene a creare una differenza di potenziale elevata e viene a crearsi un cortocircuito.

Nella programmazione in sola lettura di un circuito con multiplexer è fondamentale l'impiego dei **diodi**.

Quando un diodo viene polarizzato direttamente si comporta come se fosse un cortocircuito, e lascia passare la corrente. Un 1 dalla parte dell'anodo lo polarizza direttamente e permette di avere un 1 anche nel catodo.

Quando un diodo viene polarizzato inversamente si comporta come un circuito aperto, bloccando il passaggio della corrente. Uno 0 dalla parte dell'anodo lo polarizza inversamente e permette di bloccare il valore logico posto in corrispondenza del catodo.

Tale funzionamento è cruciale per bloccare la propagazione di uno o più 1 logici all'interno della rete logica oggetto di studio. L'utilizzo dei diodi, infatti, permette la propagazione solamente degli 1 delle porte interessate, ma non interferiscono con il funzionamento degli zeri prodotti da altre porte AND che, di fatto, rimangono isolate, impedendo ogni forma di cortocircuito e garantendo il funzionamento della rete logica stessa.

**Osservazione:** Un fusibile, dal punto di vista pratico, è costituito da un sottile filo di rame. Essi sono programmati appositamente per essere bruciati ad uno specifico livello di corrente, dimodoché l'effetto Joule sia tale da fondere il rame.

Ciò permette di programmare in fabbrica (o dall'utente) mandando in cortocircuiti i fusibili corrispondenti alla connessione. Naturalmente, questo non permette una riprogrammazione (al limite si può riprogrammare attraverso la fusione di ulteriori fusibili).

Naturalmente, tale operazione ha un vantaggio molto elevato. Se da un lato non permette una riprogrammazione (che non costituisce una procedura consueta), dall'altro permette di eliminare i  $2^n$  ingressi del circuito con multiplexer, con un notevole risparmio di *pin*.

Molto probabilmente, il costo di questa seconda opzione è inferiore rispetto a quello di un circuito ad hoc (per il quale, molto probabilmente, si dovevano anche fronteggiare degli sprechi).

Ciò, di fatto, va a costituire una prima forma di memoria, ovvero una ROM, ovvero una Read-Only Memory, in cui sono presenti  $2^n$  indirizzi di memoria e in grado di contenere  $2^n$  valori numerici, codificati mediante un certo numero di bit.

**Funzionamento:** Mandando in ingresso, su  $n$  bit, l'indirizzo di memoria da interrogare, solamente una porta AND produce in uscita il valore 0. Laddove vi sono dei diodi, questi vengono polarizzati inversamente e quindi si comportano come un circuito aperto, quindi non lasciano passare la corrente e il potenziale della linea va a 0. Tale valore logico viene poi complementato e, alla fine, viene prodotto il valore logico 1.

Alla fine, avendo inserito i diodi correttamente, avendo interrogato la cella che presenta l'indirizzo specificato in ingresso, verrà prodotto in uscita il valore contenuto nella cella interessata codificato in opportuni bit.

### 3.10 Memorie E-PROM

A partire dagli anni '70 sono state introdotte delle memorie E-PROM, ovvero *Erasable Programmable ROM*, che possono essere riprogrammate attraverso l'uso di raggi ultravioletti propagati attraverso un cristallo di quarzo presente sulla memoria stessa.

### 3.11 Moduli per la realizzazione dell'unità logico-aritmetica

Uno degli elementi centrali nell'architettura di un calcolatore è l'*Unità Logico Aritmetica*, meglio nota con l'acronimo *ALU* - che sta per *Arithmetic Logic Unit*; essa lavora a stretto contatto con i registri di memoria, guidata dall'azione dell'*Unità di Controllo*.

Nella ALU si realizzano tipicamente operazioni su numeri interi quali somma, sottrazione, incremento, decremento, scorrimento di bit, ma si attuano anche operazioni logiche sui dati, quali AND, OR, XOR o complementazione; le ALU più recenti contengono anche moduli per eseguire direttamente prodotti e moltiplicazioni.

#### 3.11.1 Il semisommatore e il sommatore completo

Il nucleo di partenza per costruire una ALU è il modulo *semisommatore* (o *Half Adder*), che realizza la somma di due bit con riporto. Di seguito viene riprodotta la tabella aritmetica della somma bit per bit con riporto, che corrisponde alla tavola di verità di due funzioni Booleane, una che realizza la somma  $S_i$  e una che realizza il riporto  $R_i$ , per ogni coppia  $A_i$ ,  $B_i$  dei bit d'ingresso. Si riconosce immediatamente che la  $S_i$  corrisponde a uno XOR, mentre la  $R_i$  è un AND; di conseguenza il modulo ha la realizzazione circuitale di figura 5.29b, mentre la figura 5.29c rappresenta il suo simbolo circuitale.

Se ora vogliamo effettuare la somma completa tra due numeri interi  $\mathbf{A} = [A_{n-1} A_{n-2} \dots A_1 A_0]$  e  $\mathbf{B} = [B_{n-1} B_{n-2} \dots B_1 B_0]$ , espressi in notazione posizionale con  $n$  bit, è necessario costruire una rete che accetti in ingresso  $2n$  bit e generi la somma binaria dei due,  $\mathbf{S} = [S_{n-1} S_{n-2} \dots S_1 S_0]$ , secondo il classico procedimento di somma con riporto evidenziato in figura 5.30a. Partendo da destra si inizia a sommare  $A_0$  con  $B_0$ ; detto  $R_0$  il riporto, questi dovrà essere sommato con la somma di  $A_1$  e  $B_1$ , che genera  $R_1$ ; questo va sommato con la somma di  $A_2$  e  $B_2$  e così via. È evidente che il riporto  $R_{-1}$  della colonna  $A_0 || B_0$  vale 0 e che se l'ultimo riporto,  $R_{n-1}$  dovesse valere 1, siamo di fronte a una situazione di overflow, che richiede il passaggio alla notazione a virgola mobile. Per realizzare un tale circuito bisogna modificare il semisommatore, in modo da includere il contributo  $R_{i-1}$  del riporto del passo precedente. In questo modo si ottiene il sommatore completo (o full adder), la cui tavola di verità è riportata in figura 5.31a. La somma di  $A_i$ ,  $B_i$  e  $R_{i-1}$  vale 1 solo quando c'è un numero dispari di 1 nella somma, ed è quindi lo XOR dei tre bit;  $R_i$  vale 1 quando  $A_i$  e  $B_i$  sono entrambi a 1 (qualunque sia il valore di  $R_{i-1}$ ), oppure quando  $R_{i-1}$  e  $A_i$  o  $R_{i-1}$  e  $B_i$  sono entrambi a 1. Per ricavare in modo formale l'espressione risolutiva scriviamo i minterm di entrambe le funzioni che si ricavano dalla tavola di verità di figura 5.31a e procediamo con

### 3.11.2 Calcolo della differenza mediante sommatore

Com'è noto, i numeri negativi, in binario, vengono rappresentati con la notazione mediante complemento a 2. Ciò significa che la differenza  $A - B$  si realizza come  $A + (-B)$ , ove  $-B$  si ottiene complementando  $B$  e sommando 1. Si ha, dunque

$$A - B = A + (-B) = A + \overline{B} + 1$$

**Osservazione:** Si osservi che la rappresentazione in binario dei numeri negativi sfrutta un difetto di ogni rappresentazione con una memoria limitata, ovvero l'**overflow**. In generale, sarebbe necessario che fosse presente un circuito che appositamente rilevi la presenza di tale overflow e provveda alla modifica della rappresentazione del risultato.

Anche con una base consueta quale quella decimale è possibile operare sfruttando l'overflow, infatti

$$\begin{array}{r} 3 \quad 7 \quad - \\ 1 \quad 5 \quad = \\ 2 \quad 2 \end{array}$$

che si può tradurre anche come

$$\begin{array}{r} 3 \quad 7 \quad + \\ 8 \quad 5 \quad = \\ 1 \quad 2 \quad 2 \quad = \\ 2 \quad 2 \end{array}$$

Ove 85 non è nient'altro che  $100 - 15$  che permette di sfruttare l'overflow per ottenere una corretta rappresentazione del risultato cercato.

La stessa cosa si può ripetere anche per il sistema binario, ovvero, se si deve realizzare  $5 - 2$  con  $n = 4$  bit, si rappresenta 2 in complemento a 2, ovvero  $2^4 - 2 = 14 = (1110)_2$ , da cui

$$\begin{array}{r} 0 \quad 1 \quad 0 \quad 1 \quad + \\ 1 \quad 1 \quad 1 \quad 0 \quad = \\ 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad = \\ 0 \quad 0 \quad 1 \quad 1 \end{array}$$

9 Novembre 2021

### 3.12 Riassunto

Impiegando un multiplexer è possibile costruire qualunque funzione booleana. Inoltre, tale dispositivo permette una grandissima versatilità, in quanto cambiando i valori di ingresso è possibile costruire tutte le funzioni booleane a  $n$  variabili, ovvero  $2^{2^n}$ .

Inoltre, i multiplexer vengono realizzati su larga scala e quindi, alla fine, il costo di una rete logica di questo tipo è inferiore rispetto ad una rete logica costituita ad hoc.

Naturalmente, poi, si osserva che gli ingressi a 0 non è conveniente che siano considerati, in quanto sicuramente porteranno a 0 l'uscita. Quindi gli unici piedini di ingresso che devono essere prolungati sono gli 1, i quali vengono direttamente costruiti internamente al circuito.

Un'altra significativa ottimizzazione prevederebbe l'eliminazione di una porta OR con più ingressi, ma impiegando una sola porta OR con un solo ingresso, dato dall'unione di tutte le uscite delle porte AND. Ma ciò è logicamente corretto, ma fisicamente erroneo, in quanto se due uscite delle AND si trovano a potenziale diverso si viene a creare un cortocircuito, che può essere risolto con la presenza di un diodo per ciascuna uscita delle porte AND, che permette di propagare la corrente in un solo senso.

Tale approccio consente una programmazione in fabbrica della rete circuitale, la quale presenta una matrice di contatti costituita da una rete di tracce che devono essere bruciate a seconda delle proprie esigenze.

Ciò permette di eliminare i  $2^n$  piedini di ingresso, in quanto la rete non viene più programmata dall'esterno, ma solamente dall'interno. Si è di fatto costituita una memoria ROM, la quale può essere interrogata semplicemente fornendo in ingresso l'indirizzo della cella di memoria di cui si vuole sapere il contenuto.

L'Unità Logico Aritmetica è il cuore del funzionamento di un calcolatore, il quale consente di effettuare qualsiasi operazione aritmetica, permettendo anche di controllare la presenza di **overflow**, ovvero il fenomeno che si verifica quando un'operazione produce un valore che eccede la capacità di rappresentazione della struttura aritmetica, il valore risultante è errato.

Naturalmente è noto che la somma tra due valori binari si può realizzare con uno XOR, mentre il riporto si può realizzare con un semplice AND. Questo dispositivo costituisce un semisommatore o *half-adder*, che posto in serie con un altro semisommatore permette di realizzare il sommatore completo o *full-adder*.

Naturalmente, in questo caso, si rinuncia ad una minimizzazione del circuito, ma ad una ottimizzazione del circuito con i dispositivi già noti.

Naturalmente, il sommatore consente di eseguire delle somme, ma per eseguire delle differenze è fondamentale comprendere la modalità di rappresentazione binaria dei numeri negativi in complemento a 2, la quale consente di centrare in 0 l'intervallo  $[0, 2^n - 1]$  costituito da tutti i valori rappresentabili con  $n$  bit, in modo tale da rappresentare anche i valori negativi.

Tale rappresentazione risulta essere geniale, in quanto consente di sfruttare un difetto della rappresentazione binaria, ovvero l'**overflow** per rappresentare anche i numeri negativi, ed eseguire, quindi, le differenze.

Ciò consente di rappresentare correttamente tutti i valori da  $[-2^{n-1}, 2^{n-1} - 1]$ ; se si eccede tale intervallo, naturalmente, si ottiene un errore.

Per rappresentare un numero binario in complemento a 2 sarà necessario complementare il valore binario e poi sommarci 1. Questo, in quanto, l'operazione di complemento propriamente detta sarebbe

$$-x = 2^n - x$$

ma osservando che

$$x + \bar{x} = 2^n - 1 \rightarrow \bar{x} + 1 = 2^n - x$$

per cui il numero negativo  $-x$  si ottiene complementando  $x$  e aggiungendovi 1.

Le tre porte di sinistra servono invece per dare un segnale di allarme quando si è in presenza di overflow; infatti dalla tavola si può osservare che tale condizione si realizza quando si verificano le seguenti condizioni:

- quando si sommano due numeri negativi (cioè con 1 nella prima posizione) e il risultato è, invece, positivo (con 0 nella prima posizione);
- quando si sommano due numeri positivi (con 0 nella prima posizione) e il risultato è, invece, negativo (con 1 nella prima posizione).

La prima delle due condizioni si ha con  $A_{n-1} = 1, B_{n-1} = 1, S_{n-1} = 0$ , e quando si realizza fornisce un'uscita 1 nella porta AND di sinistra; la seconda condizione si ha con  $A_{n-1} = 0, B_{n-1} = 0, S_{n-1} = 1$ , e fornisce un'uscita 1 nella porta AND di destra. La porta OR raccoglie dunque l'unione logica dei due eventi.

Il segnale di allarme viene usato per commutare nella rappresentazione a virgola mobile. In figura 5.34a viene illustrato il simbolo schematico della ALU descritta dalla rete 5.32. Con una piccola modifica della circuiteria è possibile fare in modo da realizzare, oltre alla somma e alla differenza tra  $A$  e  $B$ , anche l'AND e l'OR tra i due ed eventuali altre operazioni logiche; lo schema di figura 5.34b mostra una ALU completa, nella quale i comandi  $C_A, C_B, R_{-1}$  e tutti gli altri per attivare le varie operazioni logiche sono rappresentati con la notazione  $Com_1, Com_2, \dots, Com_n$ .

## 4 Circuiti sequenziali

I circuiti considerati fino a questo punto sono i circuiti combinatori, nei quali in ogni istante la configurazione di una generica variabile di uscita  $y_i$  dipende unicamente dal valore assunto dalle variabili d'ingresso  $x_1, x_2, \dots, x_n$ , secondo la funzione Booleana  $y_i = f_i(x_1, x_2, \dots, x_n)$ , con  $1 \leq i \leq m$ .

Il modello generale di un circuito combinatorio è illustrato di seguito, in cui si suppone che le variabili in ingresso vengono immediatamente propagate in uscita opportunamente processate. Si noti, infatti, che in esso non compare la variabile temporale, a sottolineare il fatto che i valori presi in considerazione per le  $n$  variabili di ingresso si riferiscono allo stesso istante e il comportamento della rete, se escludiamo i fenomeni transitori, è univocamente dedotto dalla tavola di verità che stabilisce il legame tra ciascuna  $y_i$  e le variabili d'ingresso  $x_1, x_2, \dots, x_n$ .




Figura 23: Schema generale di un circuito di commutazione con  $n$  ingressi e  $m$  uscite

Nel caso in cui in un sistema il valore delle uscite dipenda anche dalla storia passata della circuiteria che lo costituisce, ovvero dallo stato della rete, si parla di sistema o circuito sequenziale. Il sistema telefonico è un tipico esempio di sistema sequenziale: se infatti si è in procinto di comporre l'ultima cifra di un numero telefonico, il comportamento del sistema dipenderà anche dalle cifre precedentemente selezionate; questa cifra è l'ingresso attuale del sistema e l'uscita sarà il segnale che effettuerà il collegamento. Ovviamente l'ingresso attuale non è il solo fattore che determina il collegamento, poiché anche le cifre composte precedentemente sono ugualmente importanti. Anche un calcolatore elettronico è un esempio di circuito sequenziale, anzi, è l'esempio per eccellenza; di solito in esso vengono usati in maniera sequenziale diversi sottoinsiemi che possono essere di volta in volta sequenziali o combinatori.

Da un punto di vista formale un circuito sequenziale è un **automa a stati finiti**  $M$ , cioè un sistema dinamico **discreto** (nella scansione del tempo e nella descrizione del suo stato) e **stazionario** (il sistema si comporta alla stessa maniera indipendentemente dall'istante di tempo in cui agisce). Esso è caratterizzato da:

- un insieme finito  $\mathbb{Q} = \{q_1, q_2, \dots, q_S\}$  di stati interni;
- un insieme finito  $\mathbb{A} = \{a_1, a_2, \dots, a_K\}$  di valori che possono essere assunti dalle variabili d'ingresso  $\mathbf{x} = \{x_1, x_2, \dots, x_n\}$ ;
- un insieme finito  $\mathbb{B} = \{b_1, b_2, \dots, b_D\}$  di valori che possono essere assunti dalle variabili di uscita  $\mathbf{y} = \{y_1, y_2, \dots, y_m\}$ ;
- un insieme di regole, detto *mappa di transizione*  $\tau$ , che specifica lo stato  $q^*$  raggiunto dalla macchina a partire dallo stato  $q$  per effetto dell'ingresso  $\mathbf{x}$ ;
- un insieme di regole, detto *mappa delle uscite*  $\mathbb{U}$ , che specifica il valore  $\mathbf{y}^*$  assunto dalle variabili di uscita  $\{y_1, y_2, \dots, y_m\}$  per effetto dell'ingresso  $\mathbf{x}$  applicato allo stato  $q$ .

La macchina sequenziale è pertanto definita dai cinque insiemi citati:

$$M = (\mathbb{Q}, \mathbb{A}, \mathbb{B}, \tau, \mathbb{U})$$

Tanto per fare un breve esempio si consideri la seguente tabella

### 4.1 Moduli sequenziali asincroni

Nei sistemi sequenziali introdotti in precedenza, l'informazione sulla storia passata del circuito, ovvero il valore acquisito dalla variabile di stato  $Q$ , deve essere memorizzato su un qualche supporto. Si possono avere diversi tipi di dispositivi di memorizzazione, ma uno dei più usati è il cosiddetto flip-flop, ovvero un circuito **bi-stabile**.

In figura seguente viene illustrato il flip-flop già introdotto nella figura 2.26, nel quale l'uscita di

un transistor in interdizione viene connessa con l'ingresso di un transistor in conduzione. Sono transistor ad emettitore comune, in quanto l'emettitore viene posto a massa.

Il circuito che si ottiene è bistabile, nel senso che esso può stare indifferentemente e stabilmente in uno o nell'altro dei due stati rappresentati in figura 6.3a e 6.3b. Il funzionamento è basato sul fatto che, quando il transistor di sinistra è interdetto (non passa corrente nel circuito di collettore), allora la sua tensione di collettore è alta ( $v = 2,6V$ ); ciò polarizza la base del transistor di destra, che entra in ' piena conduzione, facendo collassare a un valore basso ( $v \uparrow 0,4V$ ) la sua tensione di uscita (figura 6.3a), che corrisponde a uno 0 logico. I ruoli dei due transistor si scambiano quando la tensione di collettore del transistor di destra viene portata (con un impulso) a un valore ( $v \approx 2,6V$ ) (figura 6.3b). Dal funzionamento del circuito risulta evidente che esso costituisce un elemento di memoria di 1 bit; infatti si può decidere che l'uscita del collettore ' di uno dei due transistor rappresenti la variabile logica il cui valore vogliamo memorizzare; scegliendo p.es. il transistor di destra, se vogliamo memorizzare 1 dobbiamo mandare lo stesso transistor in interdizione; per lo 0 dobbiamo mandarlo in conduzione (in una logica positiva).

**Osservazione:** Tale circuito, anche se perfettamente funzionante, in realtà non viene impiegato, in quanto non è determinabile a priori quale configurazione il sistema raggiungerà quando viene data tensione al circuito, a causa di microscopiche differenze tra le resistenze di carico e dei transistor stessi.

Naturalmente il comportamento di ciascun transistor dipende anche dalle uscite dei due transistor, in quanto in uno dei due stati ciascun transistor si trova in uno stato di **minimo locale energetico** che, però, può essere condizionato dalla variazione di tensione dei collettori dei due transistor stessi.

## 4.2 Il Flip-Flop Set-Reset - FFSR

Anche se perfettamente funzionante, questa realizzazione non viene usata nella pratica per memorizzare bit, poiché si preferisce sempre ricorrere alle porte logiche, che costituiscono le unità elementari di qualunque circuito logico. Il vantaggio di tale approccio consiste nel fatto che, stando all'interno di una certa famiglia logica, tutti i segnali di comando e i livelli di tensione sono uniformati per l'intero circuito; possiamo così aggiungere singole unità funzionali senza preoccuparci di uniformare i livelli di tensione, poiché sono già standardizzati all'interno della famiglia. Mostriamo allora le due realizzazioni principali del cosiddetto Flip-Flop Set-Reset (FFSR), basate rispettivamente sulle porte NOR e sulle porte NAND.

### 4.2.1 Latch di NOR

La figura 6.4 illustra un flip-flop realizzato con due porte NOR; in gergo viene anche chiamato **Latch di NOR**. La prima cosa che balza all'occhio è il fatto che entrambe le uscite  $X$  e  $Y$  sono riportate all'ingresso; questo lascia presagire che il valore assunto dalle variabili di uscita dipenda anche dalle uscite stesse. Se si impostano le equazioni del sistema otteniamo che conferma la nostra previsione. Nonostante l'equazione (6.1) sia impeccabile, non ci rende conto chiaramente del comportamento di questa semplice rete. La cosa migliore da fare è allora quella di fissare i valori di  $S$ ,  $R$ ,  $X$  e  $Y$  in tutti i modi possibili e vedere quali quaterne sono compatibili con i vincoli imposti dalle equazioni 6.1.

Nella figura 6.5a vengono riportate tutte le possibili combinazioni per  $R$ ,  $S$ ,  $X$  e  $Y$ ; quelle in rosso non soddisfano l'equazione 6.1, perché  $X \neq \bar{R} \cdot (S + X)$  oppure  $Y \neq \bar{S} + \bar{X}$ ; queste configurazioni non sono stabili.

Nella successiva figura 6.5b si riportano invece i soli stati stabili, per semplicità di lettura. Si osservi che, a parte il caso in cui  $R = S = 1$ , che escludiamo per i motivi che vedremo nel seguito, in tutte le altre combinazioni lecite si ha sempre  $X = \bar{Y}$ , cioè  $X$  e  $Y$  sono l'uno complementare dell'altro.

Partiamo ora dalla condizione  $R = 0$ ; dalla tabella 6.5b osserviamo che ci sono due stati stabili possibili, uno con  $X = 0$ ,  $Y = 1$  e l'altro con  $X = 1$ ,  $Y = 0$ ; supponiamo di essere nel secondo, cioè  $X = 1$ ,  $Y = 0$ , così come evidenziato in figura 6.6a. Supponiamo ora di portare l'ingresso  $R$  da 0 a 1 nell'istante  $t_1$ ; quando ciò avviene, l'uscita della porta 1 commuta a  $X = 0$  con un certo ritardo, legato ai tempi di commutazione dei transistor della porta. Il nuovo segnale  $X = 0$  alimenta l'ingresso della porta 2, facendo commutare  $Y$  a 1 con un ritardo pari a  $2\tau$ . Se ora riportiamo  $R$



a 0 (si veda figura 6.6b), X e Y rimangono nella stessa configurazione acquisita, cioè  $X = 0$  e  $Y = 1$ , poiché essa è stabile rispetto a  $R = S = 0$ , a norma della tabella 6.5b. Riportando ora R nuovamente a 1, non cambia comunque nulla, perché con  $X = 0$  e  $Y = 1$ , R e S possono stare stabilmente in ciascuno dei due stati  $R = S = 0$  oppure  $R = 1, S = 0$ . Quello che è successo è che inviando un impulso sull'ingresso  $R$ , che viene chiamato impulso di Reset, l'ingresso X va (o permane) a 0.

10 Novembre 2021

### 4.3 Riassunto

La costruzione di un semisommatore è molto semplice, in quanto prevede di costruire la tabella di verità della somma di due valori binari, che si traduce semplicemente in una porta XOR e del riporto della somma che è, appunto, una porta AND.

Il sommatore completo, invece, è dato da due half-adder in serie. Ciò giustifica la presenza di tre input, anziché due, in quanto è necessario tenere conto anche del riporto della somma precedente. Tuttavia, tale approccio, pur essendo significativamente valido, non tiene conto dell'eventualità di un overflow, così come non prevede la presenza del calcolo di differenze.

Per risolvere tale problema, è necessario comprendere la rappresentazione a complemento a 2 dei numeri binari negativi, ovvero

$$-x = \bar{x} + 1$$

Attraverso tale approccio si è in grado di eseguire la differenza, prevedendo la presenza, nel sommatore, di un pin che permette la complementazione di  $B$  e l'azzeramento di  $A$ , anche se si sarebbe potuto procedere anche in modo opposto, ma comunque perfettamente analogo.

Il controllo dell'overflow, a cui si faceva riferimento in precedenza, è sufficiente prevedere due casistiche, ovvero

- la somma di due numeri negativi (con 1 come primo bit) che produce un valore positivo (con 0 come primo bit).
- la somma di due numeri positivi (con 0 come primo bit) che produce un valore negativo (con 1 come primo bit).

eventualità che si possono rilevare grazie a due porte AND, le cui uscite costituiranno gli ingressi di una porta OR che porta il segnale.

Di fatto, in questo modo, si è costruita una unità logico aritmetica, che viene rappresentata specificatamente con il simbolo seguente

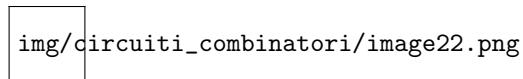
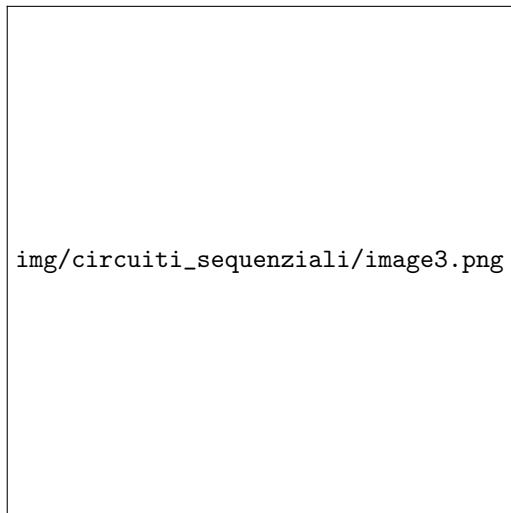


Figura 24: ALU

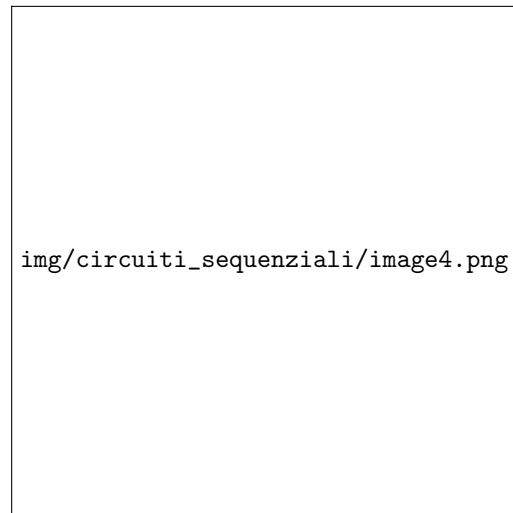
Tuttavia, i circuiti combinatori non sono gli unici dispositivi che vengono considerati. Essi, infatti, producono un risultato in uscita che dipende esclusivamente dalla configurazione degli ingressi, mentre nulla influenza lo stato interno del circuito.

I circuiti sequenziali, invece, vengono influenzati dallo stato interno del sistema, che può essere concepito come un automa a stati finiti, per cui l'ossatura strutturale dei circuiti sequenziali è data dai circuiti sequenziali, in quanto sono fondamentali anche gli ingressi, ma prevedendo anche una componente di retroazione che ne influenza lo stato.

Il circuito sequenziale fondamentale è il *flip-flop*, che può essere rappresentato dalla configurazione dei transistor seguenti, perfettamente bistabile, ovvero in grado di trovarsi equivalentemente e stabilmente su ciascuna delle 2 configurazioni possibili.



Il transistor di sinistra del *flip-flop* è interdetto; ciò  
(a) porta in conduzione il transistor di destra, che fornisce uscita logica 0



Il transistor di sinistra del *flip-flop* è in conduzione; (b) ciò porta in interdizione il transistor di destra, che fornisce uscita logica 1

2

Tale circuito, per quanto costituisca un vero e proprio dispositivo di memoria, non viene utilizzato, in quanto quando viene fornita tensione non è noto su quale stato si andrebbe a posizionare. Inoltre, per cambiare stato, bisognerebbe forza la base di un transistor per comandare il collettore dell'altro transistor, il quale, tuttavia, è collegata al collettore del transistor considerato in principio. Per questo tale dispositivo costituisce unicamente un oscillatore *RC*.

### Latch di NOR

Il Latch di NOR è un classico dispositivo a retroazione, in cui i valori delle uscite non possono essere determinate algebricamente, in quanto dipendono anche dalle uscite.

Una volta ottenute le equazioni delle uscite, che sono influenzate da loro stesse, bisogna costruire una tabella con  $2^4$  entrate e verificare per quali combinazioni di ingressi e uscite del Latch di NOR risultano essere compatibili con i vincoli imposti dal circuito stesso.

Pertanto, la tabella  $2^4$  si traduce in una tabella a 5 entrate, anche se l'ultima non viene prevista nel funzionamento del dispositivo, in quanto è l'unica che non prevede che le uscite siano  $X = X$  e  $Y = \bar{X}$ . Inoltre, fornendo il valore 1 logico su entrambi gli ingressi, il circuito ottenuto non è noto su quale configurazione si andrà a posizionare, a causa di piccole variazioni di costruzione dei dispositivi.

Studiando i diversi stati si ottiene il seguente schema, ottenuto analizzando la variazione di  $R$  da 0 a 1 e da 1 a 0.

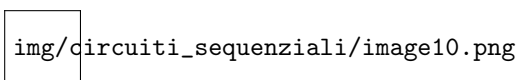


Figura 25: Funzionamento di *flip-flop*

Tabella 35: Reset del *flip-flop*

**Osservazione:** Tuttavia, per comprendere il funzionamento del sistema, è necessario comprendere anche il suo comportamento tendendo presente anche lo stato precedente del circuito, come illustrato nella seguente tabella:

Di seguito il funzionamento: Con il vincolo che uno dei due input  $R$  o  $S$  deve essere uguale a 0, ovvero  $R \cdot S$ , fornendo un impulso di *reset*  $X$  va a 0, mentre fornendo un impulso di *set*  $X$  va a 1. Attraverso la mappa di Karnaugh si ottiene la funzione

$$X = S + x \cdot \bar{R}$$

Si è detto che non si tollerano due ingressi contemporaneamente a 1, in quanto

- ambedue le uscite sarebbero a 0, violando la condizione base di funzionamento di un *flip-flop*, secondo la quale le due uscite devono essere sempre complementari.
- Se ambedue gli ingressi tornassero a 0 al medesimo istante, lo stato in cui il *flip-flop* si porterebbe non sarebbe prevedibile e al limite potrebbe realizzarsi una condizione di oscillazione.

Con il vincolo  $R \cdot S = 0$ , il *flip-flop* *RS* diventa un dispositivo di memorizzazione affidabile.

#### 4.4 Latch di NAND

Analogamente a quanto visto per il **Latch di NOR**, si può procedere per costruire un **Latch di NAND** che sarà il **corrispettivo duale** di quanto visto in precedenza.

Infatti, la configurazione  $R$  e  $S$  uguale a 0 non è ammissibile per il sistema, mentre tutti gli altri stati prevedono un comportamento esattamente identico. Pertanto

Funzionamento: Con il vincolo che  $R + S = 1$ , con un impulso 0 su  $S$  porta l'uscita  $X$  a 0, mentre un impulso a 0 su  $R$  riporta l'uscita  $X$  a 1.

**Osservazione:** Naturalmente tali dispositivi funzionano in modo perfettamente identico, solamente che i segnali di controllo dovranno essere a 1 per il NOR, mentre a 0 in NAND. Entrambi, però, funzionano in modo **asincrono**: è noto che, nei calcolatori, vi è un clock interno che controlla il funzionamento del sistema, per cui costituisce un sistema sincrono, più facilmente controllabile, ove le variazioni del sistema possono avvenire solamente in specifici istanti di clock, cosa che non accade con i *flip-flop*, ove le variazioni degli ingressi possono avvenire in qualsiasi istante. Ciò costituirebbe un problema, in quanto le variazioni degli ingressi potrebbero verificarsi mentre il segnale non è ancora stato propagato all'interno del circuito stesso.

#### 4.5 Flip-Flop SR sincrono

Si distinguono, in generale, due tipi di segnali

- *Impulso*: segnale che normalmente si mantiene ad un livello, usualmente 0, e va all'altro livello solamente per intervalli di tempo estremamente brevi;
- *A livelli*: segnale che può rimanere sia a 0 che a 1 per periodi di tempo indefiniti e comunque molto lunghi se paragonati alla durata di un impulso.

Si ottiene, di fatto, il seguente *flip-flop SR sincrono*, ovvero un dispositivo che funziona come *flip-flop SR* solamente in corrispondenza degli istanti di clock.

#### 4.6 Flip-Flop JK

Per arginare l'impossibilità di gestire gli ingressi di  $R$  e  $S$  entrambi a 1 è stato ideato il *Flip-Flop JK*, che prevede che

- Quando ( $K = J = 1$ ), se  $x = 0$ , allora  $X = 1$ .
- Quando  $K = J = 1$ , se  $x = 1$ , allora  $X = 0$

Ovvero l'uscita deve essere complementata a seconda dello stato precedente. Naturalmente, in questo caso,  $S = J$ , mentre  $R = K$ . Si ottiene, quindi

$$X = \bar{x} \cdot X + x \cdot \bar{K}$$

Si ottiene, pertanto, che

- Quando  $x = 1$ , si abilita la sola porta AND del *reset*  $K$
- Quando  $x = 0$ , si abilita la sola porta AND del *set*  $J$
- Se entrambi  $J$  e  $K$  sono a 1,  $x = 1$  forza un *reset*, mentre  $x = 0$  forza un *set*.

### 4.7 Flip-Flop di tipo T

Nel *flip-flop* T, ove  $T$  sta per *Toggle*, mentre il *clock* è il solo segnale d'ingresso. Invece, tale dispositivo, funziona in modo tale che, quando  $T = 1$ , allora l'uscita cambia stato ad ogni istante di clock, mentre se  $T = 0$  ciò non accade. Un possibile impiego di tale dispositivo permette di dimezzare la frequenza di clock del sistema stessa.

### 4.8 Flip-Flop di tipo D

Nel *flip-flop* D, ove  $D$  sta per *delay*, l'uscita dopo un impulso di *clock* è uguale al valore presente all'ingresso  $D$  all'istante di clock. Infatti si ottiene che

$$X = \bar{x}D + x\bar{D}$$

### 4.9 Registri e contatori

I flip-flop costituiscono la circuiteria di base per la memorizzazione di singoli bit in formato elettronico. A partire da essi si possono costruire unità per la memorizzazione di blocchi di  $m$  bit denominati registri; la figura mostra un esempio in tal senso, nel quale si costruisce un registro di  $m$  celle di memoria a partire da  $m$  flip-flop di tipo D.

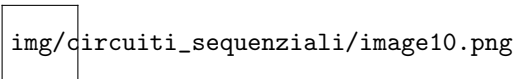


Figura 26: Registro di memoria da  $m$ -bit

Gli stessi  $m$  flip-flop di tipo D possono essere organizzati per realizzare i registri a scorrimento, che sono di fondamentale importanza per i flussi informativi interni ai calcolatori. Il circuito è rappresentato in figura e il suo funzionamento è intuitivo: a ogni istante di tempo il contenuto del registro  $j$ -esimo si sposta nel registro  $(j - 1)$ -esimo e si rende disponibile per l'uscita, che rende possibile, sotto opportune condizioni e attraverso una retroazione, la generazione di sequenze pseudocasuali con un equo bilanciamento di zeri e di uni secondo la legge debole dei grandi numeri, ma vi sarà anche una distribuzione quanto più eterogenea degli zeri e degli uni relativamente alla lunghezza della sequenza considerata.

Viene illustrato di seguito:

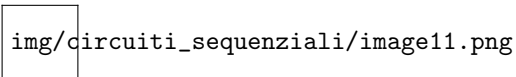


Figura 27: Registro a scorrimento

### 4.10 Contatore

Con  $m$  *flip-flop* di tipo T è possibile realizzare anche un contatore, che scandisce, una dopo l'altra, tutte le  $2^m$  configurazioni da 00...0 a 111...1. Per capirne il funzionamento facciamo riferimento al contatore a 2 bit di figura 6.18, partendo dalla configurazione 00. Il flip-flop di sinistra ha sempre  $J = K = 1$ , e in corrispondenza del primo

15 Novembre 2021

### 4.11 Riassunto

I circuiti sequenziali si distinguono dai circuiti combinatori in quanto è presente una retroazione tra ingresso e uscita, rappresentabile tramite il concetto di automa a stati finiti.

Con un circuito realizzato attraverso due transistori viene definito bistabile, ma non garantisce una affidabilità. Il suo principio, tuttavia, viene trasferito alle porte logiche, realizzando i latch di NOR, per il quale, esprimendo le uscite in funzione degli ingressi, si ottiene un cortocircuito logico. Realizzando le quaterne degli ingressi e delle uscite e ponendole in una tabella si rilevano 11 stati non stabili e non conformi ai vincoli circuitali. Solamente 5 sono gli stati stabili, anche se la configurazione con gli ingressi a 1, non da un punto di vista logico, ma dal punto di vista concettuale.

In questo caso, si hanno due ingressi  $R$  e  $S$ : dando un impulso di  $R$  si porta la variabile di uscita  $X$  a 0, con un impulso  $S$  l'uscita  $X$  va a 1.

Analizzando anche, nella tabella di verità, lo stato precedente  $x$  si ottiene la funzione specifica, con il vincolo che  $R \cdot S = 0$ .

Pertanto, fintantoché viene alimentato, tale circuito mantiene la sua configurazione; è un circuito bistabile in cui il cambio di variabile è più semplificato rispetto al circuito con due transistori.

Naturalmente, è possibile anche realizzare un *flip-flop* perfettamente equivalente, realizzato in logica duale, per cui il vincolo, in questo caso è che

$$R + S = 1$$

Taluni dispositivi, però, sono **asincroni**, ovvero in qualsiasi momento è possibile interagire con il sistema per cambiarne lo stato. Tuttavia, i dispositivi digitali sono sincroni, ovvero cambiano stato esclusivamente in corrispondenza di un preciso istante di clock. Inoltre, è doveroso tenere conto anche dei tempi di ritardo di propagazione del segnale nei circuiti, latenze determinate dalle capacità dei condensatori; anche quando ve ne sono, però, sono sempre da tenere presente le cosiddette **capacità parassite**, ovvero le capacità che si vengono a creare quando delle tracce di rame a differente potenziale sono poste in prossimità, ma non si toccano. Tali capacità parassite sono influenti a basse frequenze, ma quando le frequenze aumentano si avrà un'impedenza prossima allo 0, creando un cortocircuito, in quanto

$$Z = \frac{1}{I \cdot \omega \cdot C}$$

ove per  $Z$  è da intendere l'impedenza. Ciò potrebbe determinare anche dei fenomeni oscillatori. Ciò per spiegare che quando si cambiano i valori di ingresso in una porta si otterrà una variazione del valore di uscita dopo un certo  $\delta$ ; e se vi sono delle porte concatenate si dovranno sommare tutti i  $\delta$  delle porte.

Per tale ragione non si possono impiegare dei *flip-flop* asincroni, in quanto si avrebbe un circuito estremamente instabile.

Per risolvere tale problema sarà sufficiente introdurre due porte AND agli ingressi  $R$  e  $S$  che prendano in ingresso l'impulso di clock.

Un altro significativo *flip-flop* è il *FFJK*, il quale permette di prendere in considerazione anche gli ingressi a 0 e a 1, per cui si prevede in ogni caso la commutazione dell'uscita ( $x = 1$  si forza un *reset*,  $x = 0$  si forza un *set*).

Collegando fra di loro i due ingressi si ottiene il *flip-flop* T, per cui se  $T = 1$  si ha una continua commutazione dell'uscita, ad ogni istante di clock. Ciò permette di dimezzare la frequenza del clock, e così a cascata.

Il *flip-flop* di tipo D, invece, permette di propagare in uscita il valore di  $D$  in ingresso.

**Registri:** I *flip-flop* costituiscono la circuiteria di base per la memorizzazione di un singolo bit in formato elettronico e vengono realizzati attraverso i *flip-flop* di tipo D.

Vi sono anche i registri a scorrimento, costruiti concatenando  $n$  *flip-flop* di tipo D, dimodoché dopo  $n$  istanti si avrà in uscita il primo bit inserito.

Naturalmente l'uscita del registro può andare ad alimentare uno o più registri in ingresso, costituendo i registri a scorrimento retroazionati, che se opportunamente configurati, si possono ottenere

tutte le  $2^n - 1$  configurazioni possibili con  $n$  bit, meno la configurazione nulla, per motivazioni legate al comportamento algebrico del sistema.

Utilizzando i *flip-flop* di tipo  $T$  si riesce ad ottenere anche un contatore. Se si hanno  $n$  *flip-flop*, si riesce a contare da 0 fino a  $2^n - 1$ , con una continua ripetizione del ciclo. Quando si hanno 3 *flip-flop*, si dovrà usare una porta AND che prende in ingresso l'entrata  $X_0$  e l'uscita  $X_1$  e la cui uscita costituirà l'ingresso del terzo *flip-flop*.

## 5 Codifica

La codifica dei segnali che servono per alimentare un computer è fondamentale per la comprensione del funzionamento dei calcolatori. Ciò è la prima base per capire la connessione tra alto livello e basso livello di funzionamento di un computer.

Il calcolatore, infatti, è costituito da supporti di natura diversa che operano sempre in logica binaria. Questo, per quello che si è visto, deriva da una esigenza da un punto di vista pratico (per ragioni di dissipazione del sistema) e anche funzionale (in quanto la logica binaria è retta dall'algebra booleana).

Inoltre, è molto difficile che i dispositivi digitali si guastino, in quanto operano in una condizione ideale di dissipazione: quando si verifica un guasto, infatti, di solito è di natura elettrica riguardante l'alimentazione.

La problematica di gestione della rappresentazione di tutte le possibili informazioni in ingresso in logica binaria non è da sottovalutare. Infatti, le possibili informazioni in ingresso che devono essere processate da un elaboratore sono molteplici e svariate (caratteri alfanumerici, suoni, immagini, calcoli), e ciascuna deve essere opportunamente codificata attraverso una stringa di *bit*.

### 5.0.1 Codifica dei caratteri

Per poter codificare un carattere si potrebbe pensare di associare in modo univoco una stringa binaria a ogni lettera, dimodoché se vi sono  $k$  lettere, si dovranno impiegare  $2^{\log_2(k)}$  n-uple. Questo, naturalmente, nell'ipotesi in cui tutte le lettere siano codificate con lo stesso numero di bit; ma naturalmente si può anche prevedere di comprimere i messaggi rappresentando le lettere più frequenti con meno bit (come nel caso dell'alfabeto Morse, oppure come nei sistemi di compressione di file). In generale, vi sono quattro tipi di codifica delle stringhe:

- Da blocco a blocco
- Da blocco a stringa di lunghezza variabile
- Da stringa di lunghezza variabile a blocco
- Da stringa di lunghezza variabile a stringa di lunghezza variabile

Se si hanno circa  $K = 111$  lettere (come in una tastiera), allora si dovranno usare

$$n \geq \log_2(k)$$

ovvero il più piccolo naturale maggiore di  $\log_2(k)$ , rappresentato come segue

$$n = \lceil \log_2(k) \rceil$$

che, nel caso analizzato, si traduce in

$$n = \lceil \log_2(111) \rceil = 7$$

Da questa evidenza è nata la **tabella ASCII (American Standard Code for Information Interchanges)** che, negli anni '60, era sufficiente per rappresentare tutti i simboli che si impiegavano nelle tastiere dei primi computer che venivano introdotte.

Naturalmente, però, la diffusione dei calcolatori nel mondo occidentale ha palesato la necessità di introdurre un nuovo standard

$$\text{ISO } 8859-n, 1 \leq n \leq 16$$

ove  $n$  rappresenta il numero di bit impiegati per la codifica dei caratteri alfanumerici. Si pervenne, ben presto, alla codifica dei caratteri attraverso lo standard *UTF (Unicode Transformation Format)*, ben presto soppiantato dall'Unicode a 16 bit e poi anche da quello dell'Unicode a 32 bit.

In realtà, vi sono anche delle sottocodifiche

- UTF-8, a lunghezza variabile 1 – 4 byte
- UTF-16, a lunghezza variabile 2 – 4 byte
- UTF-32, a lunghezza costante di 4 byte



### 5.0.2 Codifica dei numeri

Naturalmente i caratteri numerici possono essere impiegati sia in combinazione con i caratteri alfabetici, ma anche per eseguire dei calcoli. Naturalmente, si capisce immediatamente che i valori numerici, essendo infiniti, non potranno essere tutti rappresentati all'interno dei calcolatori, per cui se ne darà una rappresentazione approssimata.

È noto che un valore numerico si rappresenta nella notazione posizionale come segue

$$(523)_{10} = 5 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$$

Più uin generale, si ha che

$$a_n \ a_{n-1} \ \dots \ a_1 \ a_0 = a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + \dots + a_1 \cdot b^1 + a_0 \cdot b^0$$

### 5.0.3 Conversione da base 2 a base 10

Per la conversione da binario a decimale è sufficiente sommare tutte le potenze di 2 corrispondenti agli 1 logici per ottenere il valore numerico decimale cercato.

### 5.0.4 Conversione da base 10 a base 2

Per la conversione da decimale a binario è sufficiente dividere il valore numerico decimale per 2 fino a che non si ottiene un resto  $\leq 1$  e considerare i resti di ciascuna divisione in ordine e per ottenere il valore binario sarà sufficiente leggere tali resti dall'ultimo al primo.

## 5.1 Operazioni binarie

Le operazioni tra valori binari (ma con qualsiasi base, in realtà), palesano un evidente problema, quello dell'**overflow**. Tale inconveniente permette di rappresentare i numeri binari negativi in complemento a 2.

Se si operasse ponendo il bit più significativo a 1 per rappresentare i numeri negativi e 0 per rappresentare i positivi, si incorrerebbe in un errore evidente nelle operazioni di calcolo.

15 Novembre 2021

## 5.2 Riassunto

: La codifica del mondo esterno all'interno del computer prevede una mappatura specifica per ciascuna delle specifiche informazioni da elaborare.

In particolare, le modalità di rappresentazione dei caratteri alfabetici può avvenire in 4 modalità, impiegando stringhe di lunghezza variabile o blocchi di lunghezza fissa. Naturalmente, per codificare  $k$  caratteri con  $n$  bit si devono impiegare

$$n = \lceil \log_2(k) \rceil$$

bit. A partire dalla tabella **ASCII** a 7 bit, si è poi passati ad una codifica a 8 bit per poi introdurre il sistema **UNICODE**, ancora oggi impiegata.

Per quanto riguarda la rappresentazione dei caratteri numerici, si effettua attraverso la rappresentazione posizionale.

La conversione da base 2 a base 10 si ottiene in maniera immediata; mentre la conversione da base 10 a base 2 prevede semplicemente di eseguire reiteratamente la divisione per 2 del quoziente e tenere conto del resto.

La rappresentazione dei numeri binari negativi può avvenire esclusivamente attraverso il complemento a 2. Il procedimento adottato nel caso binario è il medesimo per qualsiasi base. Infatti, considerando la somma  $37 - 15$  si può interpretare come  $37 + (100 - 15) - 100$ , ovvero  $122 - 100 = 22$  che è proprio il risultato cercato. Infatti, nell'ipotetica condizioni in cui si avessero avuto due sole posizioni per rappresentare i valori numerici, il valore 100 sarebbe stato automaticamente sottratto grazie al fenomeno dell'overflow.

In generale, per ottenere un numero in complemento a 10, avendo il vincolo di  $n$  celle di memoria si deve eseguire

$$\text{comp}_{10}(x) = 10^n - x$$

In questo modo si riesce a centrare l'intervallo  $[0 : 10^n - 1]$  in zero. La stessa cosa vale esattamente anche nel caso binario, in cui

$$-x = \bar{x} + 1$$

questo in quanto è noto che  $-x = 2^n - 1$ , ma anche  $\bar{x} + x = 2^n - 1$ , pertanto si ha che  $-x =$

Un'altra rappresentazione possibile, ma oramai desueta, oltre al complemento a 2 si ha anche il complemento a 1, che prevede semplicemente di complementare la rappresentazione dei numeri positivi. Tuttavia, in questo caso, si ha una doppia rappresentazione del valore 0 e bisogna anche cambiare le operazioni di calcolo.

Tuttavia, anche se si è stati in grado di rappresentare i numeri negativi, non è possibile rappresentare tutti i numeri reali, in quanto non si hanno infiniti bit. Per arginare il problema si deve ricorrere ad un ulteriore sistema di rappresentazione, ovvero quello a **virgola mobile**.

## 5.3 Rappresentazioni in *floating-point*

La rappresentazione a virgola mobile prevede di considerare una mantissa  $m$  e un esponente  $e$  che, nel caso della base decimale si ha

$$N = \pm m \cdot 10^{\pm e}$$

da cui si ottiene che, la rappresentazione in virgola mobile dei numeri reali è

$$[\pm] [m] [\pm] [e]$$

Da cui si conviene che, assegnando 1 bit per il segno e i restanti alla mantissa  $m$  e all'esponente  $e$  si riesce ad avere un'ampia gamma e dinamica di rappresentazione dei numeri reali sulla retta reale, pur non riuscendo a saturare interamente la densità dei numeri reali.

Infatti, la rappresentazione esposta introduce un'approssimazione, in quanto la precisione di rappresentazione dipende strettamente dal numero di cifre significative, ovvero dal numero di bit sia della mantissa che dell'esponente.

Ufficialmente, la rappresentazione in virgola mobile è stata standardizzata dalla **IEEE 754**, la quale prevede una precisa rappresentazione del valore numerico, distinto in base alla precisione, ovvero

- *Single-Precision*, con **IEEE 754-32** con 23 bit per la mantissa e 8 bit per l'esponente (rappresentato mediante eccesso a 127);
- *Double-Precision*, con **IEEE 754-64** con 52 bit per la mantissa e 11 bit per l'esponente (rappresentato mediante eccesso a 1023).

### 5.3.1 Segnali analogici

Come è possibile codificare un segnale analogico il quale ha infiniti valori e varia con continuità nel tempo. Per poter eseguire tale rappresentazione si ricorre al **campionamento** del segnale, la cui frequenza, naturalmente, deve essere la più alta possibile. Naturalmente, la lettura del segnale avviene in maniera discreta e finita, in quanto non si dispone di strumenti atti ad una lettura con precisione infinita, ma anche se vi fosse, non si disporrebbe degli strumenti atti a rappresentare tale lettura (se in *floating-point*, per quanto appena visto, si avranno blocchetti di 32 o 64 bit). Per poter ricostruire la forma d'onda di partenza con sufficiente precisione si deve avere una frequenza di campionamento sufficientemente elevata. La frequenza di campionamento, indicata con  $f_c$ , è legata strettamente alla banda (ovvero l'estensione in frequenza del segnale che si sta considerando, così per un apparato ad alta fedeltà si ha una banda che va da  $20Hz$  a  $20kHz$ ) del segnale e, secondo il **teorema di Nyquist** si ha che se

$$f_c > 2B = f_N$$

si può ricostruire il segnale senza ambiguità, ove  $f_N$  è la **frequenza di Nyquist**.

**Osservazione:** Naturalmente, bisogna tenere presente anche gli errori di quantizzazione, ovvero l'errore di rappresentazione. Infatti, il valore del campione viene decodificato usando  $n$  bit; ciò comporta un errore che diminuisce all'aumentare del numero di bit impiegati per la rappresentazione del valore campionato.

Il diagramma dell'errore è un dente di sega con valore assoluto di 1, ma se viene centrato nella metà del diagramma stesso si riesce a diminuire il valore assoluto a 0.5.

**Osservazione:** Da un punto di vista pratico, si ha che una **Conversione A/D** avviene dando in pasto un segnale analogico al **campionatore** il quale, successivamente, produce in uscita dei campioni che vengono dati in pasto al **quantizzatore** che produce in uscita i valori opportunamente rappresentati in binario.

Tali valori, opportunamente quantizzati, per essere convertiti in analogico, vengono dati in pasto a un **generatore d'impulsi** che deve essere poi seguito da un **filtro passa basso** che lascia passare solamente le frequenze più basse. Questo, infatti, perché un impulso è una variazione con una frequenza praticamente infinita del segnale considerato (infatti, è come se si avesse una sinusoide con una elevata frequenza) per cui un filtro basso taglia i contributi alle elevate frequenze e si riesce a creare le connessioni per ricostruire la forma originale del segnale.

È noto che i filtri passa alto e passa basso vengono realizzati con delle **celle R-C** che si comportano in modo opposto alle elevate e basse frequenze.

Naturalmente, tale processo palesa degli errori (errori di campionamento, di quantizzazione e del filtro passa basso).

Naturalmente i circuiti digitali non possono processare impulsi a frequenza arbitrariamente elevata, per cui gli impulsi campionati vengono interpretati sotto forma di "blocchetti" i quali mantengono lo stesso livello del segnale per il tempo di campionamento.

### 5.3.2 Immagini

Per la digitalizzazione delle immagini con sostrato e modalità di tipo digitale è stata la conseguente evoluzione della rappresentazione delle immagini con sostrato e modalità di tipo analogico.

Un tubo termoionico è una sorta di tubo con dentro un'ampolla di vetro con catodo e anodo. Il

filamento del catodo, costituito da ossidi, generalmente, quando viene alimentato con una certa tensione si scalda. Gli ossidi di Bario e Stronzio vengono sollecitati dall'agitazione termica, per cui si viene creare una nuvola elettronica attorno al catodo. Se l'anodo viene alimentato con una corrente positiva, allora gli elettroni vengono attirati dall'anodo e il flusso di elettroni viene ripristinata dalla batteria. Esattamente come per un diodo, si ha il flusso di elettroni solamente in senso.

Inoltre, se fra anodo e catodo si pone una griglia alimentato da una certa tensione negativa si viene a creare una barriera di potenziale che, a seconda della tensione di alimentazione può essere più o meno elevata che limita il flusso di elettroni. Si è costruito, quindi, un **triodo**.

Ebbene, le videocamere analogiche operano con un tubo termoionico, nel quale, però, vengono inserite anche delle bobine di deflessione per convogliare il fascio elettronico in un unico punto, costruendo, di fatto, un pennello elettronico.

17 Novembre 2021

## 5.4 Riassunto

La rappresentazione dei numeri interi in binario si basa sulla tecnica del complemento a 2. I numeri reali, invece, non possono essere interamente rappresentati attraverso un calcolatore, per quanta memoria si possa impiegare. Allora si procede ad effettuare un troncamento del numero, il quale viene rappresentato con due componenti, la *mantissa* e l'*esponente*, ciascuno con opportuno segno. La rappresentazione a virgola mobile consente di avere un'ampia dinamica di rappresentazione, sia dei numeri grandi che di quelli piccoli.

È implicito, tuttavia, che la rappresentazione in *floating-point* porta con sé un'approssimazione, tale per cui non vale più la proprietà associativa dell'addizione. Dal punto di vista della normalizzazione, si ha che lo standard *IEEE754* a 32 bit (singola precisione) e 64 bit (doppia precisione). Per quanto concerne la rappresentazione dei segnali analogici, ovvero segnali che variano con continuità nel tempo, si procede ad eseguire il campionamento, ovvero al prelievo ad intervalli regolari di campioni del segnale che, naturalmente, dovranno pur sempre essere rappresentati in modo discreto. Nyquist afferma che se la frequenza di campionamento è almeno doppia della larghezza di banda allora si riesce a ricostruire il segnale di partenza senza ambiguità. Naturalmente in tal senso si dovrà tenere presente anche dell'errore di quantizzazione, il quale viene progressivamente attenuato all'aumentare del numero di bit di rappresentazione.

Per la rigenerazione del segnale di partenza si impiega un generatore di impulsi e di un filtro passa basso; questo permetterebbe, dal punto di vista ideale, di ricostruire esattamente il segnale di partenza. Tuttavia, bisogna tenere presente sia dell'errore di campionamento, sia dell'errore di quantizzazione, sia dell'errore del generatore d'impulsi. Inoltre, la gestione degli impulsi a frequenza arbitrariamente elevata è difficoltosa, si procede a mantenere il livello del segnale quantizzato per tutto l'intervallo di tempo del campionamento.

Per quanto riguarda le immagini riprodotte dal punto di vista analogico si ha un tubo a raggi catodici di emettitori di elettroni che vengono proiettati in un tubo termoionico, in cui il catodo viene alimentato con una tensione che produce una nuvola elettronica che vengono attirati verso l'anodo e vengono convogliati verso un punto attraverso delle tecniche di deflessione magnetica, attraverso, quindi, delle bobine.

Sullo strato esterno della telecamera analogica vengono poste delle sostanze fotoconduttive, ovvero modificano la loro luminosità a seconda della loro conducibilità. Ovvero la modulazione del pennello elettronico viene modulata dall'intensità di luminosità dell'immagine che si sta riprendendo.

Taluna è una telecamera analogica, la quale rappresenta un dispositivo trasmettitore che, attraverso tecniche di modulazione di una portante (frequenza molto elevata), trasmette ad un dispositivo ricevente analogo, sempre operante a tubo catodico, in cui l'intensità del fascio viene modulata dall'intensità luminosa dell'immagine da riprodurre.

La tecnologia dei tubi **CRT** sono molto costosi, ingombranti e delicati (in quanto è composto di vetro), per cui la tecnologia è molto sofisticata, per cui non si riusciva ad abbattere i costi. Inoltre, essendo vuoti all'interno, esiste la possibilità che implodano e, inoltre, data l'elevata intensità del fascio elettronico, c'era la concreta possibilità di generare dei raggi X, molto pericolosi.

**Osservazione:** La combinazione dei tre colori base produce tutto lo spettro elettromagnetico del visibile. Per cui, per produrre il colore nell'immagine si dovranno avere tre fasci elettronici prodotti da 3 catodi; se ciascuno dei tre fasci va a colpire dei fosfori particolari che reagiscono alla sollecitazione dei tre fasci, per cui si ha l'effetto del colore sulla terna di fosfori che viene percepito dall'occhio umano.

Pertanto si hanno tre fasci (uno riservato al rosso, uno al blu e uno al verde) che vanno a colpire una cella (una terna di fosfori) che viene opportunamente sollecitata per produrre il colore desiderata. Ad oggi, invece, si ricorre alla tecnologia dei cristalli liquidi. Infatti, se con il sistema analogico si aveva una continua rappresentazione dell'immagine nel tempo, dal punto di vista digitale si procede alla discretizzazione dell'immagine attraverso la tecnica della maggioranza (si dispone una griglia di pixel e se un pixel è più colorato che bianco, allora viene colorato).

Naturalmente è ovvio che tanto più è fitta la griglia di pixel (ovvero tanto più è elevata la risoluzione dell'immagine, tanto maggiore sarà la qualità finale dell'immagine discretizzata). Per la rappresentazione della scala di grigi si associa a ciascuna colorazione una opportuna codifica, men-

tre per la riproduzione dell'immagine si procede a riprodurre una differente gradazione luminosa. Tanto maggiore sarà la profondità di colore (ovvero i livelli di grigio) maggiore sarà la colorazione e la qualità risultante dell'immagine riprodotta. La stessa cosa vale anche per i colori.

Per la creazione dei colori non si usa più un pennello elettronico che colpisce un fosforo, ma vengono realizzati da dei dispositivi a semiconduttori, come i LED: in prima approssimazione, infatti, ciascuno dei tre colori primari viene rappresentato da un diodo LED emettitore di luce. Infatti, ciascun colore viene costruito dalla combinazione lineare dei tre colori primari, con una qualità di rappresentazione che dipende dal numero di bit associati a ciascun colore primario.

Le diverse tecnologie di rappresentazione che si sono susseguite hanno previsto, in principio 8 bit per pixel, associandone 3 al rosso e verde, mentre 2 al blu in quanto l'occhio umano è meno sensibile a questo colore e più sensibile rispetto al verde.

Il canale  $\alpha$  permette anche di modificare la rappresentazione attraverso la trasparenza delle immagini.

**Osservazione:** Si capisce facilmente, però, le immagini pesano molto a seconda del numero di bit scelti. Infatti, data una risoluzione di  $1280 \times 1024$  con  $32\text{bit/pixel}$  si ha un peso di

$$1280 \cdot 1024 \cdot 32 = 5MB$$

**Osservazione:** Il numero di immagini che si possono creare con una griglia  $m \cdot n$  con solo bianco e nero si ottiene

$$2^{m \cdot n}$$

**Osservazione:** Il **bifenile** è un **cristallo nematico** con una struttura elicoidale che se viene sollecitato da una luce polarizzata (ovvero una luce in cui è stato isolato una delle due sue componenti, elettrica e magnetica) produce un colore.

Infatti, illuminando dal basso con una luce polarizzata un filtro polarizzatore e ponendo sopra un ulteriore filtro polarizzatore posto ortogonalmente al filtro sottostante e la luce può uscire accompagnata: questo accade in condizione di riposo. Se al filtro viene sollecitato da un campo elettrico, allora la luce non viene più accompagnata in uscita, ma viene sbarrata dal filtro polarizzatore e ciò permette di creare l'immagine. Tuttavia, si necessita di una illuminazione esterna per riuscire a vedere l'immagine: sono i cosiddetti cristalli liquidi passivi.

La tecnologia TFT, ovvero Thin Film Transistor, si ha una illuminazione interna, sottostante, prodotta da un dispositivo a semiconduttore. Il controllo di ogni singolo pixel avviene su ogni riga e su ogni colonna e un segnale a 20V viene fatto passare per ciascuna riga. Taluno è il segnale di attivazione che permette di attivare ciascun subpixel a seconda dello stato del gate di ciascuno. Ogni singolo subpixel, infatti, contiene delle capacità che permettono di mantenere la corrente fino a quando non si costruisce una immagine completa. Tale tempistica è necessaria per fare in modo che al turno successivo la tensione da 20V vada ad alimentare nuovamente i condensatori e così via.

Tale tecnologia, tuttavia, continua ad essere molto costosa, in quanto gli schermi per i quali non risulta funzionante anche un solo pixel vengono scartati.

## 5.5 Rappresentazione fisica dei bit

I simboli binari hanno una loro rappresentazione fisica. Infatti, i due stati 1/0 devono essere equiprobabili ed interpretati come presenza/assenza di una certa grandezza fisica (come una tensione e una corrente).

Da un punto di vista meccanico, un bit è rappresentato attraverso un interruttore (acceso/spento). In maniera più sofisticata, un bit può essere rappresentato anche attraverso un relè, che può essere comandato a distanza. Infatti, un relè è una bobina avvolta ad un nucleo ferromagnetico che quando viene sollecitata, va ad azionare o a chiudere un interruttore fisico.

Ad oggi i relè vengono utilizzati quando si necessita di azionare un grosso carico senza impiegare un interruttore standard. Infatti, quando si deve azionare un grosso carico con una molla dalla resistività più piccola possibile per abbassare quanto più la dissipazione di corrente. Pertanto l'interruttore normale viene impiegato solamente per lasciar passare la corrente e azionare il relè.

Dal punto di vista magnetico, il bit può essere rappresentato attraverso l'anello di ferrite, molto fragile ma capace di memorizzare un bit attraverso un mutamento dell'orientamento del campo

magnetico. Si tratta, comunque, di una tecnologia poco efficiente perché si hanno pochi bit per  $cm^2$ . Gli Hard-Disk vengono realizzati in polycarbonato su cui viene posto del materiale ferromagnetico che viene polarizzato attraverso una testina; pertanto la memorizzazione dei bit viene associata non alla tipologia di polarizzazione (nord/sud), ma al mutamento della polarizzazione dallo stato precedente a quello successivo.

Il bit elettrico viene rappresentato da un condensatore che, molto intuitivamente, rappresento

- Il bit logico 1 se il condensatore è carico.
- Il bit logico 0 se il condensatore è scarico.

Infatti, i condensatori presentano una resistenza di dissipazione, causa del fatto che il condensatore si scarica dopo un certo tempo (la resistenza dovrebbe essere infinita per avere un tempo di scaricamento infinito). Naturalmente, quindi, per mantenere la memorizzazione del bit sul condensatore si ha che questo deve essere sempre alimentato per mantenere il suo stato.

Ancora un altro modo per rappresentare un bit elettronicamente è il transistor; infatti, quando il transistor è in interdizione rappresenta lo stato logico 1, lo stato logico 0 se in saturazione. Questo dispositivo, come il condensatore e il *flip-flop*, devono essere sempre realizzati.

È stato anche possibile realizzare dei CD di memorizzazione sui quali viene posto una lamina di alluminio che, all'atto di masterizzazione, vengono bruciate con un laser alcune tracce dimodochè queste diventino poco riflettente. Quando il CD viene letto il bit 1 solamente sui fronti in cui la riflessione aumenta e diminuisce, lettura eseguita attraverso una tecnologia molto complessa.

Tale tecnologia si è poi evoluta passando da *CD* a *DVD* a *HD DVD* e *Blue Ray*.

#### 5.5.1 Bit nelle memoria RAM

Il bit nelle RAM statiche (SRAM) vengono realizzati con una tecnologia è molto complessa, in cui sono contenuti 6 transistor che vanno a creare un *flip-flop* e che garantisce tempi di risposti ridotti. Mentre la memoria RAM dinamica memorizza i propri bit attraverso un condensatore e i tempi di risposta sono molto ridotti

#### 5.5.2 Seriale e Parallelo

La trasmissione dei bit può avvenire un modo seriale, impiegando una sola linea, ma in maniera molto lenta, mentre in modo parallelo si usano più linee di trasmissione, ma con tempi di trasmissione molto ridotti.

22 Novembre 2021

## 5.6 Riassunto

La codifica delle immagini, ad oggi, è totalmente digitale, mentre in passato si eseguivano delle scansioni esclusivamente analogiche. Per la realizzazione dei colori ci si basa sulla terna dei colori primari **RGB**, per cui, nei tubi catodici il colore viene riprodotto andando a sollecitare attraverso dei fasci elettronici ciascuna delle celle contenente un colore primario che, viste da lontano, riproducevano correttamente l'immagine colorata.

Dal punto di vista digitale, l'immagine viene scansionata attraverso una rete  $m \times n$  di pixel, ciascuno dei quali può essere colorato (prima attraverso una scala di grigio, poi attraverso una opportuna codifica dei diversi colori dell'iride). Naturalmente, tanto più sono i bit per la descrizione di un pixel, tanto più raffinata sarà la riproduzione dell'immagine. Si capisce immediatamente, naturalmente, che le immagini sono molto pesanti in termini di memoria, tanto più pesanti a seconda del tipo di standard impiegati.

Gli schermi, ad oggi, sono realizzate attraverso la tecnologia a cristalli liquidi sfruttando delle particolari proprietà di alcuni composti organici, quali il **bifenile**, ottenendo una matrice a cristalli liquidi passiva, oppure attiva.

La rappresentazione meccanica di un bit può essere ricondotta ad un interruttore, mentre dal punto di vista elettromeccanico si possono impiegare i relè. I bit, negli anni '60, venivano memorizzati in degli anelli di ferrite, le quali potevano essere polarizzate in modo diretto o inverso, ma con un rapporto di memorizzazione e dimensione assolutamente ridicola.

Gli Hard-Disk, invece, sono ancora oggi utilizzati e funzionando attraverso una variazione di intensità ferromagnetica. Le capacità, inoltre, possono rappresentare un bit, associando lo stato carico come stato alto, mentre se il condensatore è scarico, si ha lo 0 (questa tipologia di memoria è naturalmente volatile, in quanto il circuito deve essere sempre alimentata).

Il transistor, in modo perfettamente analogo, consente di memorizzare i bit associando un valore alla condizione di saturazione e interdizione. Anche i *flip-flop* possono memorizzare dei bit, dando un colpo di set si porta l'uscita a 1, mentre si ottiene lo stato logico 0 dando un colpo di reset.

I dispositivi CD sono stati introdotti relativamente di recente, e attraverso una testina laser è possibile leggere il disco, riuscendo a distinguere i bit 1 e 0 attraverso la differenza tra zone bruciate e zone luminose del disco di policarbonato su cui sono stati eseguite delle tracce attraverso un fascio laser.

Le memorie RAM possono essere statiche (costose e molto efficienti) o dinamiche (poco costose, ma molto lente, specialmente nei tempi di risposta). I bit, su una linea, possono essere trasmessi in serie, impiegando una sola linea, ma uno alla volta, oppure in parallelo, impiegando più linee di trasmissione, ma i bit arrivano a destinazione tutti insieme.

Per la trasmissione dei segnali su una linea, i segnali analogici possono essere trasmessi codificando un valore con l'ampiezza del segnale, trasmesso ad intervalli regolari, oppure attraverso un segnale continuo, mentre attraverso un segnale digitale si codificano attraverso i comuni bit.



## 6 Rivoluzione microelettronica

La tecnologia odierna, così rivoluzionaria, è stata ottenuta attraverso un'evoluzione lunga e progressiva di tecnologie precedenti.

La prima fase della rivoluzione microelettronica riguarda l'introduzione di un primo componente attivo (che viene usata per amplificare il segnale, a differenza dei componenti passivi, come le resistenze), ovvero il **diodo di Fleming** (1904): un tubo termoionico è costituito da un anodo e un catodo, il quale, quando viene alimentato, produce una nuvola elettronica che viene attirata dall'anodo quando questo viene alimentato e assume potenziale positivo. Naturalmente, tale circuito, chiuso da una batteria, è un dispositivo unidirezionale, in quanto gli elettroni transitano dal catodo all'anodo, il quale può essere impiegato per **raddrizzare la corrente** alternata, facendola divenire corrente continua.

Il **diodo di Fleming**, tuttavia, non è ancora un dispositivo attivo, ma lo sarebbe diventato due anni dopo, nel (1906), con l'introduzione del **triolo di de Forest**, il quale era un dispositivo che aggiungeva al diodo di Fleming una griglia (una spirale, concretamente) che si trova tra il catodo e l'anodo: in questo modo, gli elettroni che transitano dal catodo all'anodo devono prima passare attraverso la griglia, la quale può essere polarizzata opportunamente per amplificare o attenuare un segnale. Il triolo può anche operare in una condizione di saturazione o interdizione.

In principio, un triolo, ovvero un tubo termoionico, veniva associato ad un singolo bit, ma il dispositivo aveva una dimensione di circa 7 – 8 centimetri.

Successivamente, negli anni '80, i dispositivi venivano realizzati non più in vetro, ma attraverso la bachelite (il primo isolante realizzato in maniera sintetica). Inoltre, bisogna ricordare che i tubi non avevano vita eterna, per cui nel tempo perdevano le loro proprietà e i filamenti interni si consumavano, per cui dovevano essere sostituiti.

Nel tempo, però, si è cercato di miniaturizzare tali dispositivi, ma anche tale processo venne limitato a causa delle **scariche distruttive**, ovvero delle scariche che si vengono a determinare quando due elettrodi vengono posti troppo vicini fra di loro. Di fatto, se non fossero stati introdotti i transistor non si avrebbe avuto la possibilità di evolvere tecnologico.

### 6.1 Seconda fase: Transistor

L'introduzione dei **transistor** avvenne nel (1948), costituito dapprima attraverso un cristallo di **germanio**, che si trova sullo stesso gruppo del silicio nella tavola periodica degli elementi di Mendeleev. Naturalmente le dimensioni erano molto ridotte, ma ancora considerevoli a causa della struttura circostante il cristallo di germanio necessaria per la trasmissione del segnale. In ogni caso, però, la dissipazione e la tensione sono molto ridotte e la durata è eterna, dell'ordine di 100.000 ore.

A partire dagli anni '70 i transistor vengono realizzati con dei contenitori (*case*) di metallo e con un cristallo di silicio.

### 6.2 Circuiti integrati

Con un cristallo di silicio è possibile realizzare un transistor, mentre introducendo dei cristalli di dimensioni più elevate è possibile creare più transistor in un solo chip e interconnessi fra di loro danno vita ad un circuito integrato, le cui dimensioni, dalla fine degli anni '70 e l'inizio degli anni '80 sono stati realizzati con dimensioni sempre più piccole.

**Osservazione:** La crescita dell'integrazione, ovvero della concentrazione di transistor in un singolo chip, è di tipo esponenziale, con un raddoppio di circa 18 e 24 mesi, nota come **legge di Moore**. Ad oggi, invece, la curva incomincia ad appiattirsi, in quanto si è arrivati ad un punto in cui le tracce del chip sono dell'ordine degli atomi. E l'appiattimento della curva è un'eventualità assolutamente inevitabile, che caratterizza tutti i processi in natura.

La legge di Moore ha influenzato significativamente il mercato tecnologico, permettendo di abbattere i costi e di migliorare in maniera importante la qualità del prodotto, ottimizzando persino il consumo di potenza.

Ciò ha delle ripercussioni anche dal punto di vista software. Infatti, *il software è come un gas*, in quanto segue le 3 leggi della termodinamica

- Il software diventa sempre più complesso all'aumentare delle prestazioni del supporto tecnologico

- Il software limita la propria complessità secondo la legge di Moore, a causa della limitazione della tecnologia hardware a disposizione
- Quando la tecnologia hardware non è più in grado di stare al passo con l'esigenza software dell'utente, vengono introdotti dei nuovi dispositivi hardware, sempre più prestazionali.

### 6.3 Concetto di informazione

È noto che i **bit** costituiscono la **minima quantità di informazione associata a due stati equiprobabili**.

L'**informatica** riguarda la **gestione e l'elaborazione automatica dell'informazione**. Ma è importante chiedersi che cosa sia l'**informazione**: l'informazione nasce a seguito di una variazione (una differenza) di una certa grandezza fisica, la quale si può sviluppare tanto in modo analogico, quanto in modo digitale. Tale differenza si propaga attraverso dei canali di comunicazione attraverso i quali è possibile prelevare un'informazione dalla relativa sorgente.

La variazione può essere trasmessa sia oltrepassando la **barriera spaziale**, oppure attraverso una **barriera temporale**. Banalmente, un supporto di trasmissione dell'informazione è la **memoria**. È fondamentale, inoltre, comprendere anche il concetto di **stratificazione gerarchica delle differenze**, in quanto le differenze possono essere rilevate anche su delle differenze (si pensi a dei dati che sono frutto di rilevazioni di differenze e poi agli attributi dei dati, che sono nuovamente delle differenze, ma rilevate su delle nuove differenze).

Inoltre, quando una sorgente di informazione genera un flusso informativo, vi sono diversi canali di informazione che vengono sollecitati, i quali portano con sé informazioni di natura diversa, che possono essere rilevati dagli utenti, i quali sono in grado di recepire tali informazioni, ma hanno anche la possibilità di scartare certe informazioni.

L'informazione, pertanto, si può rilevare su diversi piani, ovvero

- Un'informazione che si può considerare
- Un'informazione che si vuole considerare
- Un'informazione latente, ovvero disponibile, ma non immediatamente fruibile

L'informazione, poi, è alla base del cosiddetto **processo di comunicazione**, il quale si dirama in più passi

1. **Rilevamento - Livello sintattico**, ovvero una certa grandezza fisica viene rilevata attraverso uno strumento.
2. **Comprensione - Livello semantico**, la quale richiede che vi sia uno strumento cognitivo superiore, quale il cervello umano, che sia in grado di processare l'informazione attraverso i propri processi cognitivi, mediante l'esperienza e la capacità acquisite.
3. **Impiego - Livello pragmatico**, ovvero reagire all'informazione o fruire dell'informazione per compiere un'azione conseguente o sviluppare un pensiero successivo connesso all'informazione rilevata e compresa.

23 Novembre 2021

## 6.4 Riassunto

La rivoluzione microelettronica si articola in 3 fasi. La prima riguarda la costruzione dei tubi termoionici, ovvero dei dispositivi pesanti, ingombranti, fragili, poco duraturi e con una potenza dissipata assai inferiore.

La seconda fase riguarda i transistor, dispositivi assai più efficienti, in tutti i sensi, rispetto ai tubi termoionici, per poi giungere alla terza fase, riguardante i circuiti integrati, che rappresentano la concentrazione massima possibile dei transistor in un circuito integrato, che segue una legge nota come legge di Moore, di carattere esponenziale.

Tale legge è stata valida fino ad oggi, quando si è arrivati ad una miniaturizzazione tale da scontrarsi con dei limiti quantistici. Infatti, è molto interessante capire come il software si comporti come un vero e proprio gas, cercando sempre di richiedere maggiori prestazioni alle macchine che lo supportano.

Il bit dell'informazione è l'unità minima di informazione in grado di contenere i dettagli di due stati egualmente probabili.

Un sistema basato sulle interazioni informazionali è di natura molto più complessa rispetto ad un qualsiasi sistema fisico o biologico. L'informazione, inoltre, nasce quando si assiste ad una differenza di una certa grandezza fisica: senza differenze non si ha informazione, la quale può essere trasmessa attraverso lo spazio o il tempo.

Tuttavia, è anche fondamentale rilevare che oltre alle differenze di primo livello, meramente superficiali, possono essere rilevate anche differenze di più livelli, ovvero delle differenze sulle differenze. Inoltre le informazioni possono essere destinate a più utenti che avranno la possibilità di valutarle oppure no. Inoltre il processo di comunicazione si diparte in tre rami

1. Rilevamento, a livello sintattico
2. Comprensione, a livello semantico, la quale può avvenire solamente con l'esperienza
3. Impiego, a livello pragmatico, che si pone a livello dell'utente e riguarda l'utilizzo dell'informazione per i propri fini

Nella tecnologia di elaborazione dell'informazione, anche mediante i calcolatori, ci si è sempre fermati al livello sintattico, al fine di garantire una comunicazione intelligibile.

Solamente di recente si è cercato di sfondare la barriera semantica, anche se è molto complesso, cercando di rielaborare in tal senso le informazioni, in quanto per fare ciò si necessita di un cervello, di una infrastruttura logica che è capace di simulare la volontà e la direzione del pensiero umano. Grazie all'enorme patrimonio esperienziale e informazionale contenuto nel Web, ovvero alla stratificazione sintattica di frasi e stringhe strutturate logicamente e aventi senso compiuto, è possibile iniziare ad avere una prima risibile forma di interazione con un sistema esperto, nel senso prettamente pratico del tempo.

## 6.5 Ridondanza

La **ridondanza** è un eccesso d'informazione rispetto a quella strettamente necessaria per ricostruire il tutto, e risulta fondamentale per fare previsioni sui dati mancanti con una probabilità di successo superiore alla distribuzione uniforme.

L'informazione trasmessa, infatti, è costituito da un'informazione essenziale, non compressibile ed essenziale, e da tutto ciò che non è essenziale, appunto, la **ridondanza**.

Anche la ridondanza, come l'informazione, può essere di tipo **sintattico** e **semantico**, la prima riguardante i dati, mentre il secondo riguardante il significato.

Per impiegare la ridondanza semantica al fine di completare un'informazione essenziale è fondamentale valutare la frequenza relativa delle lettere dell'alfabeto della lingua impiegata per la scrittura dell'informazione.

Pertanto, per ricostruire l'integrità dell'informazione essenziale, oltre alle frequenze relative delle singole lettere, sarà necessario anche impiegare la frequenza relativa delle coppie di lettere, in modo

tale da valutare di volta in volta le lettere che seguono quelle precedenti.

Riuscendo a procedere in questo modo si riesce a costruire sintatticamente l'informazione essenziale e, anche se l'informazione non ha senso compiuto, ovvero è scorretta semanticamente, ma assomiglia ad un'informazione della lingua di trasmissione, in quanto costruita proprio sulla base delle frequenze relative delle lettere delle parole della lingua di riferimento.

Quindi l'informazione non coincide con il supporto (infatti è possibile cambiare infinite volte il supporto di trasmissione di trasmissione, ma l'informazione rimane inalterata, al più cambierà la persistenza dell'informazione stessa).

Inoltre, l'informazione non segue le leggi di conservazione della fisica, in quanto distribuendola essa non diminuisce. Pertanto non è una grandezza fisica e dipende dal contesto, in quanto la stessa informazione sintattica può assumere dei significati differenti, dal punto di vista semantico, a seconda del contesto, ovvero a seconda dell'esperienza dell'osservatore che interpreta l'informazione sintattica fornita.

È fondamentale anche capire che l'assenza di informazione è essa stessa un'informazione.

## 7 Storia dell'informatica

L'informatica deriva dal francese *Information Automatique*. L'informatica non è riducibile al calcolatore, in quanto è nata quando il computer ancora non c'era, nel 1936.

Di fatto, l'informatica non dipende dalla tecnologia di supporto e non è vincolata all'esistenza stessa di un calcolatore.

Nel 1936, infatti, sono sfociate due desideri millenari dell'essere umano

- Il sogno millenario di una macchina per eseguire i calcoli in modo automatico
- Il progetto Hilbertiano (dal tedesco David Hilbert, un celeberrimo matematico dei primi anni del '900) di *meccanizzazione della matematica* che consentisse di ottenere tutti i teoremi a partire degli assiomi e dalle regole di inferenza note (ovvero le regole con le quali si passa da affermazioni vere ad altre affermazioni vere, come la dimostrazione per assurdo).

In questo anno si assiste alla costruzione del primo calcolatore da parte di *Conrad Zuse*, un calcolatore meccanico rudimentale, ma straordinario dal punto di vista concettuale e la pubblicazione dei 3 articoli relativi al primo modello di computazione da parte di *Alan Turing*.

*Kurt Godel* ha dimostrato che la matematica ha dei limiti intrinseci, ovvero che tutto ciò che è vero non è dimostrabile, realizzando anche un secondo modello oltre a quello di *Alan Turing*, avviando una rivoluzione matematica paragonabile a quella di Einstein per la fisica.

*Alan Turing* pubblicò, nel 1936 tre articoli che erano una dissertazione meramente matematica e logica, e nulla aveva a che fare con il concetto di informatica, che al tempo nemmeno esisteva.

*Charles Babbage* fu il primo a concepire il concetto di computer moderno, anche se nel primo '800 non riuscì a realizzare un calcolatore vero e proprio, in quanto la tecnologia del tempo non era sufficientemente sofisticata per gli scopi di Babbage.

Nel 1643 venne costruita la Pascalina che riusciva, attraverso processi meccanici, riusciva ad eseguire le somme e le sottrazioni.

Nel 1674 venne concepita una nuova macchina, le *ruote di Leibniz*, che permetteva di eseguire moltiplicazioni e divisioni, con un hardware completamente diverso rispetto alla Pascalina.

Successivamente venne realizzata la *macchina analitica*, che permetteva di riprogrammare la medesima macchina per eseguire più operazioni diverse, a differenza delle macchine cablate precedenti.

*Ada Byron*, figlia di Lord Byron, fu la prima programmatrice della storia, in quanto fu la prima a comprendere le potenzialità dell'invenzione di *Babbage*.

Verso la metà dell'800 *George Boole* concepì l'algebra booleana, senza impiegarla concretamente. Cento anni dopo *Shannon* riuscì ad impiegare proficuamente l'algebra booleana per progettare e controllare le reti di interruttori nei circuiti elettronici.

Verso l'inizio del '900 erano già largamente diffuse delle macchine meccaniche capaci di eseguire dei calcoli, in uso nei grandi magazzini.

Nel 1936 venne realizzato il primo computer meccanico, la *Z1*, realizzato da *Zuse*, che poi venne distrutta nei bombardamenti, poi realizzando la *Z2* e infine la *Z3*.

Pertanto, si possono individuare tre livelli di astrazione del concetto di calcolatore

- Più macchine per assolvere a diversi scopi
- Una stessa macchina, ma con più software
- Una stessa macchina e un solo software per riuscire a risolvere problemi diversi, secondo il calcolatore universale e gli *interprete* di *Alan Turing*.

*John von Neumann* concepì la moderna architettura dei calcolatori moderni, con una memoria e un'unità di calcolo che permette di processare programmi memorizzati in memoria.

Tutti i modelli che si sono susseguiti nella storia sono tutti in grado di processare le medesime funzioni di calcolo, per cui tutti sono fra di loro equivalenti, ma espressione di diversi meccanismi per sfruttare la potenza di calcolo degli elaboratori.

Inoltre, è fondamentale osservare che fino agli anni '60 i modelli di computazione proposti non si basavano sulla tecnologia dei calcolatori, ma erano modelli astratti: questo non sorprende, in quanto quando vennero concepiti i computer non esistevano ancora.

Alla base del processo computazionale vi è, però, il concetto di **algoritmo**, ovvero in elenco di istruzioni che devono essere attuate in modo diretto, non possono essere subroutine complesse.

Il numero di istruzioni che descrivono la procedura deve essere finito.

Il numero di passi per ottenere i risultati deve essere finito.

24 Novembre 2021

## 7.1 Riassunto

Le varie generazioni di computer si sono susseguite su due binari, quello degli ingegneri, che volevano creare una macchina per elaborare i calcoli in maniera automatica, e dei matematici, i quali volevano meccanizzare la matematica, e con essa ottenere tutti i teoremi a partire dagli assiomi e le regole di inferenza.

Nel primo caso vennero realizzate delle prime macchine rudimentali, come la pascalina, per poi sfociare nella straordinaria invenzione della Z1.

Nel 1936 contemporaneamente si realizza il primo computer e vengono pubblicati i primi contributi teorici relativi alla struttura di un modello di computazione, ma solamente dal punto di vista astratto, come da parte di Turing.

Tuttavia, il problema che si ravvisò negli anni fu che i diversi modelli di computazione erano completamente distaccati e slegati dalle caratteristiche tecniche, architetturali e strutturali dei calcolatori veri e propri, proprio perché al tempo non esistevano.

Ciò che però si capì, è che tutti i modelli, per quanto indipendenti fra loro e logicamente distinti, puntano allo stesso insieme di funzioni computabili, ovvero l'insieme di problemi di risolubili.

Ciò, però, è relativamente limitante, in quanto le funzioni computabili sono molto poche, in quanto hanno un ordine naturale, per cui il rapporto tra le funzioni computabili e quelle che teoricamente si potrebbero concepire è pari al rapporto tra i numeri naturali e quelli reali.

Nel 1936 venne concepito il modello **RAM** (anche chiamato **Unlimited** ...) che permette di creare una connessione logica tra il linguaggio macchina e quello di alto livello: alla base del linguaggio RAM si pone la procedura effettiva, chiamata algoritmo, il quale deve presentare delle caratteristiche imprescindibile

1. Ogni istruzione elementare si deve attuare in modo “diretto”
2. Il **numero di istruzioni** che descrivono la procedura deve essere **finito**
3. Il **numero di passi** per ottenere i risultati deve essere **finito**: l'algoritmo, infatti, deve produrre una risposta in un tempo finito, anche se è possibile che i programmi concreti possano incorrere in delle computazioni infinite: in questo si ha la distinzione tra **programma** e **algoritmo**.

## 7.2 Nozione di algoritmo

Un algoritmo presenta le seguenti caratteristiche

1. Esso è composto da istruzioni  $I_1, \dots, I_s$  di lunghezza finita
2. Deve essere presente un **agente di calcolo** (ovvero una circuiteria a cui viene demandato il compito di eseguire i calcoli)
3. Deve essere presente, e a disposizione della memoria.
4. La circuiteria  $\mathcal{A}$  interagisce con  $\mathcal{P}$  in **modalità discreta** e non certamente in modalità analogica.
5. La circuiteria  $\mathcal{A}$  interagisce con  $\mathcal{P}$  in **modalità deterministica**, ovvero la macchina si comporterà sempre allo stesso modo partendo dallo stesso stato iniziale e producendo il medesimo stato d'uscita.

Inoltre è opportuno osservare che il calcolatore non presenta una limitazione sulla dimensione dei dati d'ingresso, così come non è necessario fissare la dimensione dell'insieme  $\mathcal{I} = \{I_1, I_2, \dots, I_k\}$  di istruzioni. Naturalmente non ha senso imporre una limitazione sulla dimensione della memoria (infatti nel modello RAM si richiede che la memoria sia infinita, ma questo è impossibile).

Ciò che è necessario limitare è la capacità di computazione di  $\mathcal{A}$ , in quanto le istruzioni richiedono di eseguire delle operazioni elementari.

Non ha senso imporre un limite sulla lunghezza della computazione, ma ha senso affermare che

la lunghezza della computazione sia finita. Tuttavia, nel modello computazionale, sono ammesse computazioni con un numero infinito di passi (computazioni non terminanti): sono i cosiddetti loop innescati dai banchi, ovvero i *bug*, i quali non sono mai eliminabili completamente (in quanto le possibilità di incorrere in un *bug* sono infinite, e vi sono dei teoremi che permettono di dimostrare che non è possibile costruire un programma di test che possa individuare tutti i *bug*).

Si potrebbe dimostrare che se si imponesse un limite al numero di passi non si sarebbe in grado di costruire dei programmi in grado di esprimere nella sua interezza le capacità computazionali dei computer.

Pertanto, accettando la possibilità di avere delle **computazioni non terminanti** si riesce ad ottenere un programma reale. Dal punto di vista teorico si ha un algoritmo; aggiungendo a tale concetto la possibilità che ci possano essere delle computazioni non terminanti, ossia dei *bug*, si ottengono dagli algoritmi i programmi.

### 7.3 Modello RAM

Il modello RAM presenta un nastro di memoria, costituita da delle celle nelle quali è possibile inserire un qualsiasi numero reale, illimitato superiormente, ovverosia avente la potenzialità dell'infinito. Inoltre è presente, naturalmente, anche il programma  $\mathcal{P}$  che è costituita da una serie di istruzioni, le quali si possono riassumere in 4 tipologie, le prime 3 di natura aritmetica, l'ultima di natura logica (responsabile della capacità di computazione del calcolatore)

1. **Azzeramento**  $Z(n)$ , ovvero  $r_n := 0$ ;
2. **Incremento**  $S(n)$ , ovvero  $r_n := r_n + 1$ ;
3. **Assegnazione**  $T(m, n)$ , ovvero  $r_m = r_n$ ;
4. **Salto condizionato**  $C(m, n, q)$ , per cui se  $r_n = r_m$  si va all'istruzione  $I_q$ .

### 7.4 Computazione

Per eseguire la computazione RAM bisogna fornire un *nastro*, caricato con una *configurazione iniziale* costituita da una sequenza  $a_1, a_2, a_3, \dots, a_n$  di numeri reali; se  $\mathcal{P} = I_1, I_2, \dots, I_s$  è il programma associato alla macchina, la computazione inizia eseguendo l'istruzione  $I_1$ . Dopo averla eseguita, la macchina RAM dovrà eseguire la *prossima istruzione*  $I_{NEXT}$  finché si raggiunge uno STOP (se mai si raggiunge).

#### 7.4.1 Prossima istruzione

Naturalmente l'istruzione successiva  $I_{NEXT}$  sarà quella numericamente successiva  $I_{NEXT} = I_{k+1}$  se l'istruzione precedente  $I_k$  è di tipo aritmetico, altrimenti sarà l'istruzione specificata dall'istruzione logica eseguita, ovvero  $I_q$ , naturalmente se la condizione indicata viene soddisfatta.

#### 7.4.2 STOP della computazione

La computazione si ferma per due possibili motivi

- È stata eseguita  $I_k = I_s$  aritmetica, oppure  $I_k = I_s = C(m, n, q)$ , con  $r_m \neq r_n$ , ovvero è stata eseguita l'ultima istruzione del programma.
- È stata eseguita  $I_k = C(m, n, q)$ , con  $r_m = r_n$  e  $q > s$ , ovvero è stata ottenuta la soluzione richiesta e si esegue una uscita anticipata dal programma e, quindi, si esegue un'istruzione successiva a quella finale, non presente nel programma.

### 7.4.3 Configurazione iniziale

È la configurazione dalla quale si parte per effettuare la computazione. Se  $a_1, a_2, \dots, a_n$  sono i dati di ingresso, per convenzione essi vengono posti all'inizio del nastro (nelle prime  $n$  celle), lasciando a 0 tutte le altre (infinite) celle di memoria.

### 7.4.4 Configurazione finale

È la configurazione che si ottiene alla fine della computazione. Per convenzione il valore  $b$  calcolato dalla computazione è il contenuto della prima cella. Ciò accade ovviamente nel solo caso in cui la computazione termini.

### 7.4.5 Convergenza

Indicando con la notazione  $\mathcal{P}(a_1, a_2, \dots, a_n)$  la computazione del programma  $\mathcal{P}$  a partire dalla configurazione iniziale  $a_1, a_2, \dots, a_n$ , se tale computazione termina si afferma che c'è stata **convergenza**, e si scrive  $\mathcal{P}(a_1, a_2, \dots, a_n) \downarrow$ . Se  $b$  è il contenuto della prima cella alla fine della computazione si dice che la computazione è andata a convergenza su  $b$  e si scrive  $\mathcal{P}(a_1, a_2, \dots, a_n) \downarrow b$ . Se la computazione non termina si dice che si è avuta una **divergenza** e si scrive  $\mathcal{P}(a_1, a_2, \dots, a_n) \uparrow$ .

### 7.4.6 Calcolo di una funzione tramite un programma

Si afferma che il programma  $\mathcal{P}$  RAM-calcola (in quanto il modello adottato è il modello RAM, altrimenti si sarebbe detto Turing-calcola, Charles-calcola) la funzione  $f$  se,  $\forall a_1, a_2, \dots, a_n, b$

$$\mathcal{P}(a_1, a_2, \dots, a_n) \downarrow b \quad \leftrightarrow \quad \begin{cases} a_1, a_2, \dots, a_n \in \text{Dom}(f) \\ b = f(a_1, a_2, \dots, a_n) \end{cases}$$

### 7.4.7 Funzione calcolabile

Una funzione  $f$  si dice **calcolabile** se esiste un programma  $\mathcal{P}$  che la RAM-calcola.

**Osservazione:** Pertanto, il numero delle funzioni calcolabili si traduce effettivamente nel numero di problemi risolubili, in quanto risolvere un problema si traduce nel trovare una funzione che se calcolata permette di ottenere la soluzione richiesta del problema.

### 7.4.8 Linguaggio delle funzioni

Per esprimere il funzionamento e l'attività di un calcolatore si ricorre al linguaggio delle funzioni.

**Esempio:** Si scriva un programma che calcola la funzione  $f(x) = 0, \forall x$ , impiegando le 3 funzioni aritmetiche note e l'unica funzione logica disponibile.

Si consideri, allora, un nastro un cui nella prima cella è presente il valore  $x$ .



26 Novembre 2021

## 7.5 Riassunto

Il percorso storico dei primi modelli di computazione si è diramato in due direzioni: da un lato per la meccanizzazione della matematica, dall'altro per la meccanizzazione dei calcoli in modo automatico.

Ciò portò alla realizzazione del primo calcolatore da parte di *Conrad Zuse* nel 1936 e, sempre nello stesso anno, si diffusero ben 4 differenti modelli di computazione, formulati da parti di logici e matematici: essi si fondano su concetti e processi logici completamente diversi, ma è interessante osservare come tutti puntino al medesimo insieme di funzioni calcolabili, e quindi di problemi risolubili.

L'ultimo modello di computazione **modello RAM** venne introdotto in modo tale da ritagliare le modalità computazionali oggetto di interesse sulla struttura fisica e strutturale di un calcolatore. Ciò permette di capire la limitatezza del calcolo computazionale, il quale non può risolvere tutti i problemi pensabili: infatti, l'insieme delle funzioni computabili ha la cardinalità dei numeri naturali, quindi è **numerabile**, mentre l'insieme dei problemi da risolvere ha l'ordine dei numeri reali, quindi non è numerabile, e lo scarto è infinito.

Alla base del calcolo computazionale vi è la cosiddetta **procedura effettiva** o **algoritmo**, il quale deve presentare delle caratteristiche imprescindibili

- Ogni istruzione deve essere direttamente attuabile
- Il numero delle istruzioni deve essere finito
- Il numero dei passi deve essere finito

Il modello RAM è un modello molto semplice, il quale prevede di disporre di una **memoria infinita** (per questo è un modello astratto, senza una controparte concreta); vi sono delle celle di memoria che contengono dei numeri naturali (di qualunque grandezza, per cui ogni cella può contenere un insieme di informazioni non limitabili superiormente) che vengono impiegate nella computazione, basata su quattro possibili istruzioni:

1.  $Z(n)$ : azzeramento della cella  $n$ -esima
2.  $S(n)$ : incremento della cella  $n$ -esima
3.  $T(m, n)$ : trasferimento del contenuto della cella  $n$ -esima nella cella  $m$ -esima
4.  $C(m, n, q)$ : salto condizionato, per cui se il contenuto della cella  $n$ -esima è uguale al contenuto della cella  $m$ -esima, si esegue l'istruzione  $q$ -esima.

Per eseguire la computazione RAM bisogna fornire un *nastro*, caricato con una *configurazione iniziale* costituita da una sequenza  $a_1, a_2, a_3, \dots, a_n$  di numeri reali; se  $\mathcal{P} = I_1, I_2, \dots, I_s$  è il programma associato alla macchina, la computazione inizia eseguendo l'istruzione  $I_1$ . Dopo averla eseguita, la macchina RAM dovrà eseguire la *prossima istruzione*  $I_{NEXT}$  finché si raggiunge uno STOP (se mai si raggiunge).

Naturalmente l'istruzione successiva  $I_{NEXT}$  sarà quella numericamente successiva  $I_{NEXT} = I_{k+1}$  se l'istruzione precedente  $I_k$  è di tipo aritmetico, altrimenti sarà l'istruzione specificata dall'istruzione logica eseguita, ovvero  $I_q$ , naturalmente se la condizione indicata viene soddisfatta.

La computazione si ferma per due possibili motivi

- È stata eseguita  $I_k = I_s$  aritmetica, oppure  $I_k = I_s = C(m, n, q)$ , con  $r_m \neq r_n$ , ovvero è stata eseguita l'ultima istruzione del programma.
- È stata eseguita  $I_k = C(m, n, q)$ , con  $r_m = r_n$  e  $q > s$ , ovvero è stata ottenuta la soluzione richiesta e si esegue una uscita anticipata dal programma e, quindi, si esegue un'istruzione successiva a quella finale, non presente nel programma.

La configurazione iniziale è la configurazione dalla quale si parte per effettuare la computazione. Se  $a_1, a_2, \dots, a_n$  sono i dati di ingresso, per convenzione essi vengono posti all'inizio del nastro (nelle prime  $n$  celle), lasciando a 0 tutte le altre (infinite) celle di memoria.

La configurazione finale è la configurazione che si ottiene alla fine della computazione. Per convenzione il valore  $b$  calcolato dalla computazione è il contenuto della prima cella. Ciò accade ovviamente nel solo caso in cui la computazione termini.

Indicando con la notazione  $\mathcal{P}(a_1, a_2, \dots, a_n)$  la computazione del programma  $\mathcal{P}$  a partire dalla configurazione iniziale  $a_1, a_2, \dots, a_n$ , se tale computazione termina si afferma che c'è stata **convergenza**, e si scrive  $\mathcal{P}(a_1, a_2, \dots, a_n) \downarrow$ . Se  $b$  è il contenuto della prima cella alla fine della computazione si dice che la computazione è andata a convergenza su  $b$  e si scrive  $\mathcal{P}(a_1, a_2, \dots, a_n) \downarrow b$ . Se la computazione non termina si dice che si è avuta una **divergenza** e si scrive  $\mathcal{P}(a_1, a_2, \dots, a_n) \uparrow$ .

Si afferma che il programma  $\mathcal{P}$  RAM-calcola (in quanto il modello adottato è il modello RAM, altrimenti si sarebbe detto Turing-calcola, Charles-calcola) la funzione  $f$  se,  $\forall a_1, a_2, \dots, a_n, b$

$$\mathcal{P}(a_1, a_2, \dots, a_n) \downarrow b \quad \leftrightarrow \quad \begin{cases} a_1, a_2, \dots, a_n \in \text{Dom}(f) \\ b = f(a_1, a_2, \dots, a_n) \end{cases}$$

Una funzione  $f$  si dice **calcolabile** se esiste un programma  $\mathcal{P}$  che la RAM-calcola.

Una computazione potrebbe anche essere non terminante, come nel caso di una funzione con ciclo infinito: allora si dirà in quel caso che la  $n$ -upla  $a_1, a_2, \dots, a_n$  non appartiene al dominio della funzione, ovvero la funzione non è definita per l' $n$ -upla  $a_1, a_2, \dots, a_n$ .

**Osservazione:** Se all'interno di un programma vengono inserite delle istruzioni superflue che non alterano il funzionamento di un programma, si ottiene un programma strutturalmente e logicamente diverso, ma che produce il medesimo risultato, calcolando la stessa funzione.

**Esempio:** Per sommare fra di loro due numeri  $x$  e  $y$  senza saperne il valore, si dovrà controllare in prima analisi se il contenuto della seconda cella è uguale a quello della terza, ovvero  $y = 0$ .

Se così non è si procede ad incrementare la prima e la terza cella e si esegue un salto incondizionato alla prima istruzione di controllo: se la seconda e la terza cella sono uguali allora si termina il programma, altrimenti si continua il ciclo:

1.  $C(2, 3, 99)$
2.  $S(1)$
3.  $S(3)$
4.  $C(1, 1, 1)$

**Esercizio:** Per eseguire la sottrazione tra due numeri si proceda a capire come si esegue la cosiddetta **somma naturale**, ovvero

$$f(x) = x1 = \begin{cases} x - 1 & \text{se } x > 0 \\ 0 & \text{se } x = 0 \end{cases}$$

Per cui si procede come segue

1.  $C(1, 4, 99)$
2.  $S(3)$
3.  $C(1, 3, 7)$
4.  $S(2)$
5.  $S(3)$
6.  $C(1, 1, 3)$
7.  $T(2.1)$

**Esercizio:** Se si vuole eseguire la differenza tra due valori numerici sarà sufficiente eseguire il programma seguente

1.  $C(1, 2, 5)$
2.  $S(2)$
3.  $S(3)$
4.  $C(1, 1, 1)$
5.  $T(3, 1)$

Nel caso in cui  $x < y$  la differenza non può essere effettuata, allora si deve fare in modo che il programma cicli in maniera indefinita.

## 7.6 Programmazione imperativa

Ogni istruzione corrisponde a un “comando” che viene impartito alla macchina, e che prevede l'esecuzione di un certo lavoro.

Si si fa riferimento al paradigma più usato di **programmazione imperativa**, denominato *programmazione strutturata*, è possibile affermare che un programma è solitamente costruito nel seguente modo:

- **una parte dichiarativa**, in cui si dichiarano tutte le variabili del programma e il loro tipo (come, ad esempio, variabile intera, variabile carattere, etc.)
- **una parte che descrive l'algoritmo** risolutiva utilizzato, basato sulle istruzioni del linguaggio che, a loro volta, si suddividono in
  - istruzioni di lettura e scrittura (scrittura a video, scrittura su disco, lettura da tastiera, ...)
  - istruzioni di assegnamento (del valore a una variabile)
  - istruzioni logiche di controllo del programma

### 7.6.1 Istruzioni logiche di controllo

Esistono essenzialmente tre tipi di strutture logiche di controllo del programma

1. Sequenza
2. Selezione
3. Iterazione

### 7.6.2 Selezione

Per eseguire un **if  $C$  then  $I_{SI}$  else  $I_{NO}$**  è sufficiente eseguire due salti condizionati, come mostrato di seguito

1.  $I_{prec}$
2.  $C(m, n, 5)$
3.  $I_{NO}$
4.  $C(1, 1, 6)$
5.  $I_{SI}$
6.  $I_{succ}$

### 7.6.3 Iterazione while - do

Per eseguire un **while**  $C$  **do**  $I$  si procede come segue

1.  $I_{prec}$
2.  $C(m, n, 4)$
3.  $C(1, 1, 6)$
4.  $I$
5.  $C(1, 1, 2)$
6.  $I_{succ}$

### 7.6.4 Iterazione repeat - until

Per eseguire un **repeat**  $I$  **until**  $C$  si procede come segue

1.  $I_{prec}$
2.  $I$
3.  $C(m, n, 5)$
4.  $C(1, 1, 2)$
5.  $I_{succ}$

## 7.7 Tesi di Church-Turing

Tutto ciò che può essere fatto col calcolatore tradizionale può essere realizzato anche col “calcolatore” astratto del modello RAM (o qualunque altro modello di calcolo equivalente) e viceversa. O meglio, si può fare di più con il modello RAM che con un modello concreto, in quanto si ha una quantità di memoria infinita nel primo, mentre è finita nel secondo.

Per questa ragione, si ha che il modello del calcolatore è **Turing-completo**, in quanto rispetta il modello computazionale di Turing.

## 8 Architettura dei calcolatori

È nota la struttura fisica di un sommatore, in grado di eseguire somme e sottrazioni e capace anche di individuare l'overflow.

Dopodiché sono stati progettati i differenti *flip-flop* che si pongono alla base dei registri normali, sequenziali e dei contatori.

### 8.1 Componenti essenziali di un calcolatore

Le componenti essenziali di un calcolatore sono 3, così come stabilito da Von-Neuman:

1. una *memoria indirizzabile*, che possa contenere il programma e i dati
2. un'*unità logico-aritmetica*, che possa lavorare sui dati della memoria, ovvero l'agente di calcolo
3. un *program counter*, cioè un registro che indica l'indirizzo di memoria dell'istruzione che deve essere eseguita

### 8.2 Ciclo *fetch-decode-execute*

Le istruzioni di un calcolatore vengono eseguite secondo un ciclo che prevede di eseguire tre istruzioni, secondo un ciclo nella forma **repeat** *I* **until** *C*, ovvero

1.  $PC \leftarrow 0$ ;
2. **repeat**
  - (a)  $\text{istruzione} \leftarrow \text{memoria}[PC]$
  - (b)  $\text{decode}(\text{istruzione})$
  - (c)  $\text{fetch}(\text{operandi})$
  - (d)  $\text{execute}$
  - (e)  $PC \leftarrow PC + 1$
3. **until** istruzione = STOP

30 Novembre 2021

### 8.3 Riassunto

L'architettura dei calcolatori è basata sull'infrastruttura logica fornita da Jhon Von Neumann. In particolare, all'interno del calcolatore dovrà essere inserito un programma che dovrà essere compilato dal calcolatore e tradotto in linguaggio macchina per poi essere eseguito.

Il funzionamento del computer si può riassumere con il ciclo *fetch-decode-execute* di seguito presentato

1.  $PC \leftarrow 0$ ;
2. **repeat**
  - (a)  $istruzione \leftarrow memoria[PC]$
  - (b)  $decode(istruzione)$
  - (c)  $fetch(operandi)$
  - (d)  $execute$
  - (e)  $PC \leftarrow PC + 1$
3. **until**  $istruzione = STOP$

Tale ciclo si arresta quando si raggiunge lo STOP della computazione. L'architettura dei calcolatori che consente di svolgere tali operazioni è così formata

1. L'ALU, ovvero l'unità logico aritmetica in cui è presente il sommatore già studiato.
2. I registri, che sono le unità di memoria più vicine all'ALU
3. La memoria RAM, ovvero la memoria volatile sulla quale viene caricato il programma da eseguire.
4. Il Program Counter, il quale è un registro di memoria che indica l'indirizzo della cella di memoria che contiene l'istruzione che deve essere eseguita.
5. Il RI, ovvero il registro che contiene l'istruzione da eseguire.
6. L'ACC, ovvero l'accumulatore dei risultati delle operazioni.

### 8.4 Struttura di base del calcolatore

Alla base dell'infrastruttura del calcolatore si pongono i seguenti quattro elementi

- Processore
- Memoria primaria o principale (RAM)
- Memoria secondaria o di massa (HDD)
- Periferiche di I/O

#### 8.4.1 Processore

All'interno della CPU sono presenti due componenti fondamentali UC (Unità di Controllo) e ALU (Arithmetic Logic Unit), alla quale viene associato un Processore Matematico, impiegato per svolgere le operazioni a virgola mobile. Inoltre sono presenti i seguenti registri

- PC - Program Counter
- RS - Registro di Stato (per gestire, per esempio, gli overflow)

- RI - Registro delle Istruzioni
- Registri Generali
- Registri di gestione RAM, per la gestione della lettura/scrittura
  - RIM - Registro Indirizzi di Memoria
  - RDM - Registro Dati di Memoria
  - RC - Registro di Controllo

**Osservazione:** Se il registro RIM è a 32 bit, significa che possono essere indirizzati fino a  $2^{32} = 4GB$  celle di memoria. Il RC, ovvero il registro di controllo della memoria RAM indica se deve essere eseguita un'operazione di lettura, oppure di scrittura. all'interno del registro dati di memoria (RDM) viene inserito il dato che deve essere letto o scritto da/nell'indirizzo di memoria specificato nel RIM.

Si capisce, però, che il punto in cui devono essere eseguite le operazioni (ALU) è fisicamente distante dal punto in cui si acquisiscono, o inseriscono le informazioni (ovvero la RAM): ciò comporta un rallentamento e una limitazione della capacità di calcolo, per non parlare del bassissimo efficientamento energetico che contraddistingue tale sistema.

## 8.5 Istruzioni in linguaggio ASSEMBLY

Le istruzioni di basso livello in linguaggio ASSEMBLY sono di quattro tipi

- Istruzioni di I/O
  - **READ INP 1322:** Legge da tastiera e mette il dato nella cella 1322 della memoria RAM
  - **WRITE OUT 1902:** Scrive a video il contenuto della cella 1902 della memoria RAM
- Istruzioni logico/aritmetiche
  - **ADD R1 R2**
  - **MUL R1 R2**
- Istruzioni di accesso alla memoria
  - **LOAD 1672 R2**
  - **STORE R1 1559**
- Istruzioni di salto
  - **BRLT**
  - **JUMP**

**Osservazione:** Le operazioni che devono essere eseguite per effettuare un'operazione:

1. L'indirizzo di memoria specificato nel PC deve essere copiato nel registro RIM
2. Nel registro RC viene specificata l'operazione da eseguire: lettura
3. Una volta letto il contenuto della cella di memoria specificata, l'istruzione viene inserita all'interno del RDM
4. Il contenuto del registro RDM viene trasferito nel registro RI in modo tale che possa essere eseguito dal processore
5. Nel caso di una lettura, bisogna ripetere le operazioni di lettura dalla memoria RAM e inserire il contenuto letto nel primo registro disponibile
6. Una volta eseguita l'operazione viene incrementato il PC

7. Si ripetono le operazioni precedenti con iterazioni

Tali istruzioni possono essere riassunte in 5 macroistruzioni

1. IF - Instruction Fetch
2. ID - Instruction Decode
3. EX - Execution
4. MEM - Memorization
5. WB - Write BACK

Ad oggi queste operazioni non vengono più eseguire in serie, ma in parallelo (*pipeline*), con il significativo vantaggio che ad ogni ciclo di clock viene eseguita una operazione.

Le istruzioni che vengono specificate in linguaggio ASSEMBLY sono, per quanto di basso livello, comunque orientate all'utente e, per poter essere eseguite dalla macchina devono essere tradotte in linguaggio macchina. Il processo di traduzione avviene in due passaggi

1. Dopo aver scritto il programma in linguaggio ad alto livello, questo viene **compilato** in linguaggio ASSEMBLY
2. Il programma scritto in **ASSEMBLY** viene assemblato dall'**assemblatore**, il quale converte tale programma in linguaggio macchina (ISA), ovvero come sequenza di 0 e 1.

In realtà, però, sono presenti più di due livelli di traduzione dei programmi, come riportato di seguito

- Livello -1 - Elettronico, a livello circuitale: transistor, mosfet
- Livello 0 - Logico, di porte logiche
- Livello 1 - Microarchitettura, come ALU e registri
- Livello 2 - ISA (*Instruction Set Architecture*)
- Livello 3 - Sistema Operativo
- Livello 4 - Assemblatore
- Livello 5 - Linguaggi applicativi

In questo modo la progettazione del livello  $i$ -esimo deve valutare solamente cosa c'è al livello  $i - 1$  e  $i + 1$ .

## 8.6 Gerarchia di memoria

A mano a mano che ci si allontana dalla unità logico aritmetica si incontrano memorie più capienti, meno costose, e più lente, come HD e SSD, mentre risalendo la gerarchia si incontrano la RAM, la memoria CACHE e i registri.

### 8.6.1 Registri CPU

I registri della CPU sono contraddistinti da

- Altissima velocità di accesso
- Bassissima capacità
- Altissima velocità di trasferimento dati
- Integrati nell'infrastruttura



### 8.6.2 RAM

Le memorie RAM (Random Access Memory), pur sempre volatili, sono di due tipi

- DRAM, poco costosa, poco efficiente e lenta
- SRAM, costosa, efficiente e molto veloce

Le RAM hanno un accesso a tempo costante, sono relativamente veloci e poco capienti, se compa-  
rate alle memorie secondarie.

### 8.6.3 Memoria Cache

La memoria Cache è in stretto contatto con il processore (è realizzata nel chip) e permette di evitare l'accesso alla RAM. Il tempo di accesso è molto ridotto, è basata sulla tecnologia SRAM e ha una dimensione assai inferiore alla RAM.

Anche all'interno dei moderni processori le memorie cache vengono gerarchizzate, in modo tale da ridurre il più possibile il ritardo di lettura/scrittura e gestire le interazioni fra i diversi core.

### 8.6.4 Memoria secondaria

Le memorie secondarie sono due tipologie

- Hard Disk
  - con alta capacità di memorizzazione
  - a basso costo
  - ad alto tempo di accesso
  - con memoria permanente (non volatile)
  - basata sulla tecnologia elettromeccanica (sul disco di polycarbonato viene letto e scritto grazie ad una testina meccanica): per questo la tecnologia è molto arretrata, a causa dell'inerzia tra i dispositivi meccanici e a causa della presenza dei motori ad alto consumo.
  - è possibile anche creare dei banchi di HD in modo tale da aumentare la capacità
- Solid State Disk (SSD), ovvero la prima memoria elettronica **non volatile**, che permette di memorizzare le informazioni imprigionando gli elettroni in una zona di potenziale dalla quale è difficile scappare. Gli SSD
  - hanno alta capacità di memorizzazione
  - hanno alto costo, seppur in diminuzione, ma di sicuro maggiore agli HD
  - hanno elevata velocità di lettura e basso tempo di risposta e di accesso

Naturalmente, a queste memorie di massa interne, si possono sommare delle memorie di massa esterne che, pur essendo molto capienti, hanno tempi di accesso, lettura e scrittura molto elevati.