Evoluzione storica dei calcolatori

Il primo calcolatore in assoluto fu l'abaco, uno strumento inventato nel 3000 a.C. in grado di eseguire addizioni e sottrazioni. Per un lungo periodo non ci furono miglioramenti negli strumenti in grado di assistere in qualche modo le operazioni matematiche, fino al 1617 in cui Nepero inventò dei bastoncini o delle bacchette (che oggi ereditano il suo nome) in grado di eseguire non solo addizioni e sottrazioni ma anche moltiplicazioni, divisioni e radici.

Tra il 1621 e il 1623 vennero inventati degli strumenti in grado di rendere più meccanico il processo di calcolo progettato da Nepero attraverso dei cilindri ruotanti (macchine calcolatrici di Oughtred e Schickard).
Invenzione seguente, nel 1645 nacque la prima calcolatrice meccanica automatica, la pascalina (di Pascal appunto), in grado di eseguire somme e differenze con massimo 12 cifre e ponendo automaticamente il riporto.

La prima calcolatrice completa fu invece quella progettata da Leibnitz nel 1674. Questa si trattava di un meccanismo talmente complesso per la tecnologia dell'epoca che per realizzarla ci vollero ben 120 anni. Siamo ormai nel 1804 quando Joseph-Marie Jacquard decise di fare il passo decisivo verso la nascita dei computer: inventare un metodo per programmare una macchina.

Inventare non è esattamente il termine corretto, poiché egli si ispirò al funzionamento dei pianoforti per rendere automatica la sua macchina da tessitura. La macchina utilizzava delle schede perforate che permettevano il passaggio, quando il momento era opportuno, del filo attraverso i fori e lo ritiravano su con dei contrappesi.

Nonostante questa macchina non abbia molto a che fare col mondo della matematica, viene considerato uno dei primi predecessori del computer.

Charles Babbage invece è, a pieno titolo, il "padre dei computer". Egli fu un grande inventore: inventò il "pilota" della locomotiva, il dinamometro, l'intermittenza nella luce dei fari, un sistema unificato di tariffazione postale, era appassionato di decifrazione di codici e molto alto ancora.

Egli progettò (ma non realizzò mai per via della difficoltà di reperire i materiali) quella che prese il nome di "macchina differenziale", una macchina in grado di risolvere qualsiasi polinomio.

Fu però considerato il "padre dei computer" per via di una sua macchina in particolare, che andò a sostituire la "macchina differenziale" e migliorare le sue funzionalità (ricordarsi la leggenda): la "macchina analitica", in grado di trovare una soluzione a qualsiasi calcolo, acquisire input da schede, era dotata di una unità di controllo e una aritmetica e POTEVA ESSERE PROGRAMMATA. Furono proprio queste ultime due caratteristiche ad elevare questa macchina al titolo di "predecessore del computer" e ad essere un punto di svolta per lo sviluppo dei calcolatori.

Lady Augusta Ada, contessa di Lovelace, la prima programmatrice della macchina di Babbage e a cui fu anche dedicato il primo linguaggio di programmazione mai inventato, l'Ada appunto, si appassionò molto al lavoro di Babbage che decise di raccontare e descrivere le sue invenzioni al pubblico.

Ben presto, con la scoperta dell'elettricità, queste macchine che prima erano esclusivamente meccaniche inizieranno a sfruttare l'energia elettrica a loro favore.

Hollerith, nel 1884, per via dell'aumento della popolazione sempre crescente e della necessità di censire sempre più persone per ragioni diplomatiche, decise di automatizzare il processo del censimento attraverso una macchina. Questa macchina era formata da un lettore di schede perforate (le quali avrebbero dovuto contenere i dati), una macchina elaboratrice e una macchina ordinatrice. La lettura delle schede funzionava tramite una matrice di fili elettrici che se passavano attraverso il foro di una scheda chiudevano il circuito.

Il risultato fu un censimento che avvenne in 3 mesi anziché 7 anni del precedente censimento. L'evoluzione di questa macchina portò alla nascita della Computer Tabulating Recording Company nel 1913, nel 1918 all'assunzione di Thomas J. Watson come presidente dell'attività e nel 1924 alla nascita di International Business Machines (IBM).

L'IBM, insieme ad un fisico di Harvard, nel 1944 costruì un computer noto come IBM ASCC, un computer ad altissima potenza per gli standard passati, pesante 35 tonnellate, fatto da relay meccanici e con cavi per una lunghezza complessiva di 800 km.

Fu proprio in questo periodo che venne trovato il primo "bug", una falena che si era incastrata tra i contatti elettrici del computer e ne aveva alterato il comportamento. La persona che lo trovò fu l'ammiraglio della Marina Americana Grace Murray Hopper (da cui prenderà il nome il linguaggio di programmazione COBOL).

Alan Turing, appassionato in teoria dei giochi, fu uno dei tanti matematici che vennero radunati per risolvere un problema che avrebbe cambiato le sorti della Seconda Guerra Mondiale: decifrare Enigma, la macchina che utilizzavano i tedeschi per inviare i propri messaggi.

Egli fu il "padre dell'informatica" e creatore della macchina di Turing, o "macchina universale", una macchina teorica in grado di leggere dei dati, memorizzarli su un nastro ed eseguire le operazioni su di essi. Risolve ogni problema "risolvibile", lavora in binario ed è rappresentata attraverso concetti matematici.

La macchina creata seguendo queste tracce, il Colossus, riuscì a decifrare enigma e fu anche il primo prototipo di computer al mondo.

Ad utilizzare il concetto di macchina di Turing seguirono un professore ed il suo studente di dottorato, Atanasoff e Berry, che ebbero l'idea di computer moderno nel 1937. Questo prenderà il nome di ABC (Atanassof and Berry Computer).

Atanassof e Berry diederò l'idea per la creazione dell'ENIAC (che non venne loro riconosciuta), una macchina di 30 tonnellate, 18.000 valvole e con la capacità di calcolo di un normale computer odierno.

Però programmare nell'ENIAC consisteva nel cambiare manualmente le connessioni elettriche e impostarle per eseguire delle determinate operazioni.

Il "padre del computer moderno" invece fu Von Neumann, che creò un modello di computer (ancora tutt'oggi utilizzato) e che superò le limitazioni dell'ENIAC permettendo la memorizzazione delle istruzioni nella sua stessa memoria, rendendo il processo di programmazione meno prono ad errori umani e più rapido.

Mentre i computer di prima generazione utilizzavano le valvole (tubi di vetro contenenti circuiteria a vuoto per proteggere le componenti elettroniche), quelli di seconda generazione passarono ai transistor, degli interruttori on-off di piccolissime dimensioni che non solo risparmiavano spazio, ma erano anche più rapidi e meno costosi.

Questi transistori venivano integrati con altre componenti in circuiti integrati nei computer di terza generazione.

Oggi si utilizzano i VLSIC (Very Large Scale Integrated Circuit), anche chiamati “microchip”, evoluzione dei circuiti integrati.

Rappresentazione dei numeri naturali

I calcolatori elettronici sono tutte quelle macchine in grado di elaborare delle informazioni trasformandole in altre informazioni.

I calcolatori elettronici devono essere generali (general purpose) ossia devono essere in grado di risolvere problemi di natura differente, in base alle necessità di chi lo programma.

Nell'informatica si intende per informazione tutto ciò che può essere scritto o descritto attraverso una sequenza di simboli in un certo alfabeto prefissato. Il codice è un insieme di parole composte da simboli appartenenti ad un certo alfabeto.

Un alfabeto è un insieme discreto, ossia un insieme contenente un numero finito di simboli. Un insieme può essere discreto per sua natura intrinseca (tipo l'insieme di abitanti sulla Terra), o per quantizzazione della misurazione, ossia per una misurazione effettuata ad intervalli e che è resa quindi finita per questioni di praticità.

In generale, ogni insieme con un numero finito di elementi è definito discreto. I segnali sono a loro volta grandezze fisiche discrete. Nei calcolatori elettronici, la lettura di un certo segnale in input e la scrittura di un certo segnale in output danno come risultato una sequenza di 0 e 1. Questi stati, che possono essere chiamati in tanti modi (LOW e HIGH, 0 e 1, SPENTO E ACCESO), vengono rappresentati dalla misurazione della tensione d'entrata ed uscita.

Una tensione tra -0.5 e 1 volt di uscita equivale ad uno stato di LOW, tra 4 e 5,5 volt ad uno stato di HIGH. Una tensione d'entrata tra -0,5 e 2 volt invece viene tradotta in uno stato di LOW, e una tra 3 e 5,5 volt in uno stato di HIGH. C'è da notare che tra segnale d'entrata e d'uscita c'è uno scarto: questo permette al circuito di funzionare correttamente nonostante eventuali (e direi inevitabili) rumori.

La codifica è una funzione che trasforma un insieme di informazioni in un codice di un determinato alfabeto. La decodifica è l'operazione inversa della funzione di codifica.

I criteri di valutazione di un codice sono:

- l'economicità: sono considerati migliori i codici che utilizzano un alfabeto con numero di simboli più basso;
- semplicità di codifica e decodifica, ossia la facilità con cui si può passare da quel codice ad un altro e viceversa;
- semplicità di elaborazione, ossia la semplicità con cui le operazioni di calcolo vengono eseguite.

Il computer è formato da una CPU (a sua volta formato da un *datapath* che esegue le operazioni aritmetiche ed una unità di controllo che coordina i dati), una FPU (anche a sua volta formata da datapath e unità di controllo) che si occupa esclusivamente delle operazioni a virgola mobile e una MMU che si occupa della gestione della memoria. La MMU, insieme alla cache interna ed esterna e alla memoria RAM, formano la parte del blocco della memoria.

Un sistema numerico viene detto posizionale quando il valore di un simbolo cambia a seconda della sua posizione nella parola, e dove la cifra a destra viene chiamata LSD (Least Significant Digit) e quella a sinistra MSD (Most Significant Digit).

$$\sum_{i=0}^{n-1} c_i * b^i, \qquad c_i \in \Sigma$$

b è $la\ base\ di\ rappresentazione$
$sigma$ è $l'alfabeto\ delle\ cifre\ di\ una\ determinata\ base$
c è $una\ cifra\ in\ una\ determinata\ posizione\ rispetto\ al\ numero$

Due parole uguali possono aver un significato diverso in base alla funzione che viene usata per codificarle. Bisogna pertanto interpretarle correttamente.

Un codice binario è un codice costituito solo da una sequenza di 0 e 1. Uno 0 o un 1 vengono chiamati bit (Binary digIT).

Il codice binario viene utilizzato nella computazione per due motivi (ricollegandoci alla qualità del codice):

- George Boole dimostrò che ogni operazione logica può essere risolta utilizzando un codice binario (vero o falso);
- Claude Shannon, padre della teoria dell'informatica, dice che il sistema ACCESO o SPENTO è perfetto per descrivere il comportamento dei circuiti digitali.

Metodo polinomiale per passare da una base qualsiasi a base dieci e metodo delle divisioni successive (o iterate) per passare da base 10 a base qualsiasi.

Metodo polinomiale:

$$N_a = \sum_{i=0}^{n-1} c_i * a^i, c_i \in \{0, \dots, a-1\}$$

$La\ base\ di\ partenza$ è $a, quella\ di\ arrivo$ è b
C'è $una\ cifra\ in\ una\ determinata\ posizione\ rispetto\ al\ numero\ in\ base\ a$
$Le\ operazioni\ vengono\ eseguite\ in\ base\ b$

Metodo delle divisioni successive:

- Teorema della divisione euclidea:

$$D = d * q + r$$

D è $il\ dividendo$
d è $il\ divisore$
q è $il\ quoziente$
r è $il\ resto$

- Applicazione del metodo:

$$N_a = c_0 + a * c_1 + a^2 * c_2 + \cdots + a^{n-1} * c_{n-1}$$
$$= c_0 + a * (c_1 + a * c_2 + \cdots + a^{n-2} * c_{n-1})$$

$Si\ noti\ la\ somiglianza\ con\ il\ teorema\ euclideo$:
c_0 è $il\ resto, a\ il\ divisore\ e\ quello\ tra\ parentesi$ è $il\ quoziente$.

Si continua così, dividendo per il nuovo quoziente, finché il resto è uguale a zero.
Se si passa da base a per arrivare a base b, le divisioni vengono effettuate in base a e anche la base è rappresentata in base a.

L'intervallo dei numeri Naturali va da $\{0, \dots\ 2^{n-1}\}$
Per rappresentare un numero N in una base b qualsiasi, si prende la parte intera di $\log_b N$ e si aggiunge uno.
Il risultato ottenuto è il numero di bit necessari a rappresentare quel numero.

Operazioni con i numeri naturali

Per rappresentare i numeri naturali in base 2 di solito si utilizza un numero di bit prefissati.
Questo apre alla possibilità della presenza di un *overflow*, ossia di un risultato il cui numero di bit supera il numero di bit prefissato per la rappresentazione, rendendolo non rappresentabile.
Per la somma tra numeri naturali, una volta fissata la dimensione della rappresentazione, è presente un overflow se e solo se il riporto risultante dalla somma di due numeri naturali è 1 sul MSB.

Nei naturali, la sottrazione non è definita per m – s se m è minore di s.

Nei naturali, la moltiplicazione tra due numeri della stessa lunghezza usa una rappresentazione di lunghezza da 2n – 1 a 2n.

Rappresentazione degli interi

I numeri interi differiscono dai numeri naturali per il segno.
Esistono tre modi principali per rappresentare i numeri interi in binario:

- la rappresentazione in *modulo e segno*;
- la rappresentazione in *complemento a uno*;
- la rappresentazione in *complemento a due*.

Il metodo più utilizzato è la rappresentazione in complemento a due, poiché i primi due metodi rendono le operazioni sugli interi più complesse.

Con il terzo metodo invece la somma è immediata e la sottrazione è la somma con il numero opposto.

Per rappresentare un numero in complemento a due, c'è bisogno di n + 1 bit, dove n è il numero di bit che servirebbero per rappresentare il valore assoluto del numero in base 2 e un bit è per il segno.

La seguente dimostrazione dimostra che la somma tra un numero e il suo complemento a due da sempre come risultato zero, pertanto i due numeri possono essere visti come numeri opposti.

$$N = -c_{n-1} * b^{n-1} + \sum_{i=0}^{n-2} c_i * b^i, \qquad c_i \in \{0, \dots, b-1\}$$

$$N' = -\overline{c_{n-1}} * b^{n-1} + \sum_{i=0}^{n-2} \overline{c_i} * b^i + 1, \qquad c_i \in \{0, \dots, b-1\}$$

$$N + N' = -c_{n-1} * b^{n-1} + \sum_{i=0}^{n-2} c_i * b^i - \overline{c_{n-1}} * b^{n-1} + \sum_{i=0}^{n-2} \overline{c_i} * b^i + 1 =$$

$$-(\overline{c_{n-1}} + c_{n-1}) * b^{n-1} + \sum_{i=0}^{n-2} (c_i + \overline{c_i}) * b^i + 1 = -b^{n-1} + \sum_{i=0}^{n-2} b^i + 1 = -b^{n-1} + b^{n-1} + 1 - 1 = 0$$

L'intervallo di rappresentabilità dei numeri interi è *asimmetrico*.

$$[-2^{n-1}, +2^{n-1} - 1]$$

Infatti, il numero 100…0 non ha un simmetrico positivo, infatti provando a calcolarne il complemento otterremo lo stesso numero: per questo motivo, lo consideriamo fuori dal range di rappresentabilità e ogni operazione che produce questo numero produce un errore di *overflow*.

Un errore di overflow si verifica nella somma quando a segni concordi si produce risultato discorde, o più genericamente, quando il risultato di una qualsiasi operazione tra quei due numeri in base 10 non è rappresentabile con n bit in complemento a due, quindi occupa più di n-1 bit.

Rappresentazione dei numeri razionali

In un sistema posizionale qualunque, un numero razionale è rappresentato in questo modo.

$$c_{n-1} \dots c_1 c_0,\ c_{-1} c_{-2} \dots c_{-m}$$
$$che\ in\ forma\ polinomiale\ si\ traduce\ in:$$

$$\sum_{i=0}^{n-1} c_i * b^i + \sum_{i=1}^{m} \frac{c_{-i}}{b^i}$$

Per convertire la parte intera dei numeri razionali si usano gli stessi metodi che si usano per i numeri interi (metodo polinomiale e divisioni iterate). Per la parte frazionaria invece, si usano il metodo polinomiale e il *metodo delle moltiplicazioni iterate*.

La rappresentazione numerica si ferma quando il quoziente raggiunge il valore zero.

Non sempre i numeri rappresentabili in una base usando un numero finito di cifre dopo la virgola sono rappresentabili con un numero finito di cifre dopo la virgola in un'altra base.
I numeri convertiti inoltre potrebbero risultare periodici.
In entrambi questi casi, si prende una approssimazione del numero scartando le cifre binarie che eccedono i bit dedicati alla rappresentazione di quel numero.

I problemi della rappresentazione a virgola fissa ne limitano il campo di utilizzo:

- l'intervallo dei reali rappresentabile è piccolo e con approssimazioni grossolane;
- l'ampiezza e la precisione di rappresentazione non sono adeguate per calcoli scientifici.

Si usa pertanto la rappresentazione in virgola mobile, rappresentata dalla terna <s, m, e> formata da segno, mantissa ed esponente.
Questa rappresentazione si ispira alla notazione scientifica.

Se il numero è negativo, allora il bit di segno vale 1, altrimenti vale 0.
La mantissa è rappresentata attraverso il metodo di rappresentazione dei numeri naturali, quindi senza complementazione.
L'esponente invece è rappresentato in complemento a due.

Per avere uno standard di rappresentazione si utilizza una *forma normalizzata*, in modo tale che la parte intera sia nulla e che la parte frazionaria inizi con una parte frazionaria non nulla.
Il bit della parte intera, essendo sempre nullo in questo tipo di rappresentazione normalizzato, viene tralasciato e considerato sottointeso.

L'unico numero non rappresentabile in virgola mobile normalizzata è il numero *zero*.

L'intervallo di rappresentazione dei numeri in virgola mobile dipende dal numero dei bit della mantissa e dell'esponente.

$$e\ = [-2^{m-1} - 1, \qquad 2^{m-1} - 1]$$
$$s, m = [-2^n - 1, \qquad +2^n - 1]$$
$$range = \ [(-0{,}10 \dots 0) * (2^{m-1} - 1), \qquad (0.11 \dots 1) * (2^{m-1} - 1)]$$

Operazioni in virgola mobile

Anche tra numeri in virgola mobile si possono eseguire operazioni.

Nella somma, prima di effettuare la somma i due numeri devono essere portati allo stesso esponente (può accadere che i due numeri siano di ordini di grandezza così diversi da azzerare completamente il numero), la somma tra i due numeri prende il segno del numero il cui valore assoluto è maggiore e la mantissa è semplicemente la somma (o sottrazione, in base ai segni) delle due mantisse successivamente normalizzata. L'esponente diventa quello dei due addendi se il risultato è normalizzato, altrimenti deve essere modificato opportunamente.

Nel prodotto, non c'è bisogno che i due numeri abbiano lo stesso esponente.

Il segno del risultato è dato dal prodotto dei segni dei due numeri, la mantissa è data dal prodotto delle due mantisse e l'esponente dalla somma degli esponenti. Se il risultato non è normalizzato, allora bisogna modificare opportunamente l'esponente.

Nel prodotto può accadere che si verifichi l'overflow degli esponenti e che il risultato non sia pertanto rappresentabile.

Altri codici importanti

ASCII è un acronimo per *American Standard Code for Information Interchange*.

Creato dall'IBM nel 1961, diventa standard ISO (*International Organization for Standardization*) nel 1968. Codifica con 7 bit tutte le lettere maiuscole, minuscole dell'alfabeto inglese, le cifre decimali, i simboli di interpunzione e vari caratteri speciali:

- i primi tre bit indicano la categoria del tipo di carattere che deve essere rappresentato (cifre decimali, lettere minuscole, …);
- i restanti quattro bit ordinano i caratteri all'interno della categoria secondo il loro ordine naturale.

Molto presto però l'ASCII si rivelò insufficiente per la quantità di caratteri che bisognava gestire, poiché 7 bit consentivano la gestione di solamente 128 caratteri.

Ci furono varie estensioni dell'ASCII a 8 bit, ma ogni compagnia aveva il proprio standard. Pertanto l'ISO decide di creare uno standard che sarebbe poi potuto essere usato univocamente da tutti.
In realtà gli standard creati dall'ISO erano numerosi, uno per ogni parte del mondo.

La limitazione dello Standard ISO era che codici uguali venivano utilizzati per rappresentare diversi caratteri e pertanto non era un tipo di rappresentazione autenticamente globale; molto presto perciò nacque l'Unicode, a 16 bit, che comprendeva simboli di tutte le lingue (vive e morte), simboli matematici, chimici ecc …

Nonostante non sia uno standard De Jure, è uno standard De Facto e viene oggi comunemente utilizzato. Gestito e costantemente aggiornato dall'Unicode Consortium, oggi ormai utilizza 21 bit, ma lascia molte sequenze inutilizzate.

Indipendentemente da quello che rappresenta, il rumore elettrico può alterare una sequenza di bit e renderla irriconoscibile.
Poiché se tutte le combinazioni di simboli in una parola sono valide allora diventa impossibile individuare se una sequenza in codice è corretta o no, solitamente una maggiore ridondanza di codici disponibili determina una maggiore protezione dagli errori.

Esistono tanti modi per individuare errori nelle sequenze, ma il più semplice è senza dubbio il controllo del bit di parità. Una codifica che conta un numero pari di 1 si chiama parità pari, una che conta un numero dispari di 1 si chiama parità dispari.

Aggiungere un bit di parità permette di rivelare un errore ma non permette di correggerlo, perché è impossibile capire qual è il bit cambiato. Si nota che se avviene un numero pari di errori allora l'errore può non essere rilevato!

Nel codice con parità longitudinale e trasversale il messaggio disposto in una matrice, e per ogni riga e colonna della matrice viene posto un bit di parità.

Il codice con parità longitudinale e trasversale mette un numero di bit di parità pari alla somma della larghezza e della lunghezza della matrice, che nella migliore delle ipotesi (il caso di una matrice quadrata) equivale a $2\sqrt{n}\ bit$, (dove *n* è la lunghezza del messaggio), fino a un massimo di *n+1* bit.

Il codice con parità longitudinale e trasversale può correggere un errore, rilevarne due ma non correggerli e potrebbe anche non rilevare tre errori.

Il codice di Hamming corregge 1 errore e ne rileva due, ma con un numero inferiore di bit di controllo. Infatti, utilizza sempre $log_2 n\ bit$ invece che $2\sqrt{n}$ nella migliore delle ipotesi del codice con parità longitudinale e trasversale.

I bit di controllo si mettono nelle posizioni che sono potenze di due, e nel caso il numero di bit del messaggio *n* non sia potenza di 2, il numero di bit di controllo è pari alla parte intera del logaritmo +1.

L’algebra di Boole

Come già dimostrato in passato, il codice binario è utile per la rappresentazione del comportamento di un circuito elettrico, o più in generale, per descrivere gli stati dei circuiti digitali.

I motivi per cui questo tipo di rappresentazione è molto utilizzato derivano dalle osservazioni di due persone in particolare:

- prima di tutto lo dobbiamo a Boole che, grazie alla sua invenzione, l'algebra di Boole, ha dimostrato come la logica possa essere rappresentata utilizzando soltanto un codice binario e tre operazioni;
- per secondo, lo dobbiamo alla teoria della comunicazione di Claude Shannon che è riuscito a dimostrare come l'utilizzo di segnali di acceso/spento siano tutto ciò che serve per rappresentare lo stato di un circuito digitale.

Ai tempi di Claude Shannon. i circuiti utilizzavano dei relè i quali quando aperti non permettevano il passaggio di corrente; al contrario quando questi relè erano chiusi permettevano il passaggio di corrente.

Claude non fece altro che formalizzare le operazioni che potevano essere effettuate su uno più circuiti attraverso un alfabeto binario composto da 0 e 1, dove zero significa spento/falso/aperto e uno significa acceso/vero/chiuso. Queste operazioni sono tre.

Prima, la commutazione di stato che consiste nell'apertura di un circuito se esso è chiuso o alla chiusura di un circuito se esso è aperto: considerando che gli stati che un circuito può assumere sono 2, la commutazione di stato ci fa ottenere lo stato del circuito opposto a quello corrente. Questa operazione prende anche il nome di operazione NOT ed è rappresentata con una sbarra sopra l’espressione booleana.

Seconda, la composizione in serie, che consiste nel porre 2 relè uno dopo l'altro sullo stesso circuito, appunto in serie. Pertanto, il circuito consente il passaggio di corrente e solo se entrambi i relè sono chiusi. Questa operazione viene anche chiamata operazione di AND ed è rappresentata con il simbolo della moltiplicazione.

Terza la composizione in parallelo, che consiste nel porre 2 relè collegati appunto in parallelo: in questo modo, basta che è solo uno dei 2 relè sia chiuso, per permettere il passaggio di corrente. Questa operazione viene invece chiamata operazione di OR ed è rappresentata con il simbolo dell'addizione.

L’alfabeto dell'algebra di Boole pertanto è formato da 5 elementi, di cui 2 rappresentano gli stati dei circuiti, e 3 rappresentano le operazioni che è possibile fare sui suddetti stati. L'algebra di Boole gode di proprietà che abbiamo già visto nelle classiche operazioni di somma e moltiplicazione:

- la proprietà commutativa, secondo la quale scambiando gli operandi il risultato non cambia.
  Es. $x + y = x + y,\ x * y = y * x;$
- la proprietà associativa, secondo la quale l'ordine in cui faccio la somma o ogni prodotto di 3 o più elementi non è importante.
  Es. $x + (y + z) = (x + y) + z,\ x * (y * z) = (x * y) * z;$
- la proprietà distributiva, secondo la quale nel prodotto posso distribuire una somma come prodotto del primo addendo moltiplicato al primo fattore del prodotto più il secondo addendo moltiplicato al primo fattore del prodotto.
  Es. $x * (y + z) = (x * y) + (x * z);$
  A differenza delle operazioni di somma e moltiplicazione regolari, la proprietà distributiva vale anche in relazione alla somma sul prodotto.
  Es. $x + (y * z) = (x + y) * (x + z);$

- l'elemento neutro, ossia quell'elemento che non cambia il risultato dell'operazione. Così come nelle operazioni normali, nella somma l'elemento neutro è 0, nel prodotto è 1.
  Es. $x + 0 = x,\ x * 1 = 1$;
- il complemento, che definisce che, un numero booleano qualsiasi sommato al suo complemento fornisce come risultato 1. Un numero booleano qualsiasi invece, per il suo complemento, fornisce come risultato 0.
  Es. $x + \bar{x} = 1,\ x * \bar{x} = 0$;

Da questi assiomi, si possono derivare numerose leggi. Tra queste, osserveremo le più importanti.

La legge di involuzione specifica che un numero booleano qualsiasi equivale a sé stesso complementato per 2 volte.

$$x = \bar{\bar{x}}$$

La legge di idempotenza specifica che un numero booleano qualsiasi moltiplicato per sé stesso fornisce come risultato lo stesso numero booleano. La stessa legge vale anche per la somma.

$$x * x = x, \qquad x + x = x$$

Il principio di dualità specifica che, nell'algebra di Boole, leggi e dimostrazioni valgono anche se di esse viene calcolato la duale. Per calcolare la duale bisogna:

- scambiare gli 0 con 1 e viceversa;
- scambiare i * con i + e viceversa.

Questo principio è valido poiché tutti gli assiomi godono del principio di dualità.

Anche una delle due identità della legge di idempotenza, descritta in precedenza, può essere dimostrata o ricavata, a partire dalla dimostrazione o dalla legge dell'altra identità, effettuando la duale di quest'ultima.

La legge dell'elemento annullatore specifica che un numero booleano qualsiasi moltiplicato per 0 fornisce come risultato sempre 0, e, grazie al principio dualità, sappiamo per certo che anche la sua duale è vera: quindi, un numero booleano qualsiasi aggiunto ad 1 fornisce come risultato sempre 1.

$$x * 0 = 0 \xrightarrow{per\ dualità} x + 1 = 1$$

La legge dell'assorbimento specifica che sommando ad un numero booleano qualsiasi il prodotto di quello stesso numero con un altro numero booleano qualsiasi, il risultato dipende esclusivamente dal primo numero booleano.

$$x + (x * y) = x \xrightarrow{per\ dualità} x * (x + y) = x$$

La legge di De Morgan dice che la somma di due numeri booleani qualsiasi complementata da come risultato il prodotto dei due numeri complementati. Per il principio di dualità, anche la sua duale è valida.

$$\overline{x + y} = \bar{x} * \bar{y} \xrightarrow{per\ dualità} \overline{x * y} = \bar{x} + \bar{y}$$

Con le tavole di verità (di Shannon, ma applicabili anche nell'algebra di Boole) è possibile ottenere tutte le possibili combinazioni di valori e dimostrare per esaurimento che ognuna di queste leggi è vera.

Espressioni e operatori booleani

Un'*espressione booleana* è una sequenza di parentesi, operatori, costanti e variabili booleane, definita induttivamente così:

$$Sia\ V\ un\ insieme\ numerabile\ di\ variabili; allora:$$

$$-\ Passo\ base:$$

$$0, 1 \in EB;$$
$$se\ x \in V, allora\ x \in EB.$$

$$-\ Passo\ induttivo:$$

$$se\ E \in EB, allora\ x \in EB;$$
$$se\ E_1, E_2 \in EB, allora\ E_1 + E_2, E_1 * E_2 \in\ EB.$$

Un'*espressione duale* si ottiene scambiando 0 e 1, + e *.
Un'*espressione complementare* si ottiene negando l'espressione di partenza.

Due espressioni booleani si dicono *equivalenti* se a fronte dello stesso assegnamento di valori booleani alle loro variabili assumono lo stesso valore. La dimostrazione che due espressioni booleane sono equivalenti si ottiene o per induzione perfetta o con dimostrazioni formali.

Quindi, una equazione booleana identifica una *funzione booleana*, una legge che in base ai valori delle variabili restituisce un valore booleano in maniera univoca.

Una funzione booleana può essere rappresentata attraverso una tavola di verità che associa agli elementi del dominio quelli del codominio.
Date n variabili, la tavola della verità sarà composta di 2^n combinazioni possibili, ognuna delle quali sarà associata al valore di verità della funzione in corrispondenza della detta combinazione.

Le porte elementari sono l'AND, l'OR e il NOT.
Le porte logiche sono la NAND, la NOR, la XOR e la XNOR.

Le porte AND, OR, XOR e XNOR sono associative.
Le porte NAND e NOR non sono associative.

Le porte NAND e NOR si dicono universali perché da esse si possono ricavare tutte le altre porte.

Nonché usando AND, OR e NOT I circuiti necessiterebbero di meno porte, l'uso delle NAND o NOR come uniche porte permette la loro produzione in massa nei così detti circuiti integrati e quindi porta ad un significativo risparmio dei prezzi.

Da una SOP è molto facile costruire un ALL-NAND equivalente e da una POS è molto facile costruire una ALL-NOR.

Forme canoniche e forme normali

Per ogni espressione booleana esiste un'unica funzione booleana associata, però una funzione booleana è associata a più espressioni booleane.

Le *forme canoniche* servono a fare in modo che ogni funzione booleana abbia un'unica espressione booleana associata (in forma canonica).

Una *forma canonica disgiuntiva (o SOP)* è una somma di *mintermini* distinti tra loro.

Un mintermine m è un *implicante* di f se applicando il mintermine alla funzione booleana si ottiene 1.
Per ogni mintermine m, esiste un'unica n-pla di bit che la fa valere 1: ogni mintermine perciò deve contenere tutti i termini (negati o affermati) della colonna sinistra della tavola di verità.
In una forma normale gli addendi non sono in generale mintermini.

Una *forma canonica congiuntiva (o POS)* è un prodotto di *maxtermini* distinti tra loro.
Per ogni maxtermine M, esiste un'unica n-pla di bit che la fa valere 0: ogni maxtermine perciò deve contenere tutti i termini (negati o affermati) della colonna sinistra della tavola di verità.
In una forma normale i fattori non sono in generale maxtermini.

Le forme canoniche sono uniche a meno dell'ordine dei termini, quindi a meno della proprietà commutativa.

Minimizzazione di espressioni booleane

Il problema di ricavare una espressione minima nasce dalla necessità di voler ridurre il numero di porte logiche necessarie a realizzare una rete combinatoria.
Questo permette di diminuire non soltanto il costo dei circuiti, ma anche il tempo di attraversamento.
Per minimizzare le espressioni booleane in modo più rapido si utilizzano le mappe di Karnaugh (per un numero di variabili booleane fino a 4).

Un implicante corrisponde ad un rettangolo formato da 2^k "1", cioè 2^k mintermini che variano tra loro per una sola variabile booleana.
Un implicante si dice *primo* se non esiste un implicante di dimensioni maggiori.
Un implicante si dice *essenziale* se contiene almeno un "1" non contenuto in nessun altro implicante.

Le MdK permettono di ottenere le forme normali congiuntive e disgiuntive minime usando AND, OR e NOT.
Le MdK potrebbero essere ulteriormente minimizzabili con l'utilizzo di porte composte come XOR, NAND, XNOR ecc...

Reti combinatorie

Una *rete combinatoria* è un circuito in grado di calcolare in automatico una funzione booleana.
Un *circuito elettronico* è un sistema costituito da blocchi elementari (porte) interconnesse tra loro in maniera aciclica, ossia in modo tale che l'output di ogni blocco non possa essere anche l'input dello stesso. Ingressi che non sono uscite di altre porte sono gli *input*, le uscite che non sono ingressi di altre porte sono *output*.

Uno *schema circuitale* è un collegamento di porte tramite delle linee:

- linee di ingresso, ognuna delle quali è etichettata con una delle n variabili di entrata;
- linee di uscita, ognuna etichettata con una delle m variabili di uscita;
- linee interne, ciascuna delle quali collega l'uscita di ogni porta all'entrata della successiva.

È importante l'assenza di collegamenti ciclici, che differenzierà le reti combinatorie da quelle sequenziali.

Per induzione, la rete combinatoria può essere in questo modo definita:

- PASSO BASE: una rete combinatoria ad una uscita può essere una linea che collega un input x ad un'uscita z senza nessuna porta in mezzo; una linea che collega un input x ad un'uscita z con una porta ad una entrata (NOT) in mezzo; una linea che collega due input x e y ad un'uscita z usando delle porte a due ingressi;
- PASSO INDUTTIVO: una rete combinatoria è la giustapposizione (ossia, la combinazione) di due o più reti combinatorie in cui gli ingressi uguali vengono uniti.

Per ogni equazione booleana esiste un'unica RC in grado di rappresentarla utilizzando solo porte NOT, AND e OR. Se si usa anche porte NAND, NOR, XOR e XNOR si perde l'unicità.
Per ogni equazione booleana pertanto esiste un'unica RC. Per ogni RC esiste un'unica funzione booleana. Per ogni funzione booleana, esiste un'unica tavola di verità. Per ogni tavola di verità esistono infinite equazioni booleane.

Circuiti notevoli

Un sommatore parallelo fa la somma aritmetica tra due stringhe di bit. Un sommatore utilizza due tipi di celle elementari: un half adder e tanti full adder.

Con il sommatore uniforme invece utilizzo tutti full adder.
Per trovare l'opposto di una stringa di bit, bisogna effettuare il NOT bit a bit e poi aggiungere 1.
Per la sottrazione si fa la somma con l'opposto.

Un comparatore aritmetico ha 2 uscite: una che indica che il primo numero è maggiore del secondo (se settata a 1), e una che indica che il primo numero è uguale al secondo (se settata a 1).
Se entrambe sono zero, allora il primo numero è minore del secondo.

Altri circuiti integrati

Tra gli altri circuiti integrati c'è il codificatore.

Il codificatore 4 a 2 ha quattro linee di ingresso (x3, x2, x1 e x0) e due linee di uscite (y1, y0).
y1 = x3 + x2
y0 = x3 + x1

In un codificatore $2^n - a - n$ si usa una matrice di OR in cui la linea verticale associata ad x_i contiene la codifica binaria di i, vedendo il pallino come 1 e l'assenza di pallino come 0.

In un codificatore generalizzato si usa una matrice di OR in cui la linea verticale associata al valore che corrisponde al numero da codificare è la codifica del numero da codificare.

Si sotto-intende che in un codificatore possa essere attivo solo un ingresso alla volta.

Una ROM (Read Only Memory) è un dispositivo con n ingressi (detti linee di indirizzamento) e m uscite (dette linee dati). Il motivo per cui prende questo nome è perché, essendo composta da una matrice di AND e OR già predisposta, può essere vista come una memoria e una volta saldati i diodi non è modificabile.

Spesso la matrice di AND si sostituisce con un decodificatore, in modo tale da dover solo riempire le righe della matrice di OR, chiamata matrice della ROM.

Un PLA (Programmable Logic Array) è una rete combinatoria integrata con n ingressi, m uscite e tre parti:

- una parte in cui gli ingressi vengono negati;
- una matrice di AND;
- una matrice di OR.

Il PLA consente però di implementare espressioni booleane in forma FND, in particolare equazioni booleane in FND minime (è pertanto più efficiente ed economico di una ROM).

Questo avviene perché anche la matrice di AND è personalizzabile in base alle decisioni dell'utente.
Infatti, ogni PLA possiede una serie di K porte AND e m porte OR (una per ogni uscita) i cui ingressi non sono collegati a niente e devono perciò essere creati autonomamente.
E' perciò più laborioso costruire la PLA della ROM.

Un multiplexer in senso stretto possiede n linee dati e n linee di controllo in ingresso, e una linea di uscita.
Un multiplexer (MUX) è una rete combinatoria con n ingressi, una uscita e $log_2 n$ segnali di controllo.
Un multiplexer è un multiplexer in senso stretto le quali entrate delle linee di controllo sono le uscite di un decodificatore di $log_2 n$ ingressi.

In un multiplexer in senso stretto, ogni entrata del segnale di controllo è messa in AND con la linea che gestisce: tutti gli AND sono successivamente messi in OR e collegati ad un'unica uscita.
Si sotto-intende che nel multiplexer in senso stretto può essere attiva solo una linea di controllo alla volta.

Un de-multiplexer in senso stretto possiede una linea dati e n linee di controllo in ingresso, e n linee di uscita.
Un de-multiplexer (DEMUX) è una rete combinatoria con un ingresso, n uscite e $log_2 n$ segnali di controllo.
Un de-multiplexer è un de-multiplexer in senso stretto le quali entrate delle linee di controllo sono le uscite di un decodificatore di $log_2 n$ ingressi.

Latch e Flip Flop

Finora abbiamo considerato solo circuiti *aciclici*, questo perché abbiamo assunto implicitamente che le porte siano *ideali*, cioè il loro tempo di attraversamento sia nullo.
Dire che un circuito è aciclico significa collegare all'entrata di un circuito la sua uscita, cosa che in un circuito con tempo di attraversamento nullo non ha senso.

In realtà, le porte hanno un tempo di attraversamento, solitamente rappresentato con un blocco indicante il ritardo TAU.
Questo introduce il tempo nei circuiti, che prenderanno il nome di *reti sequenziali*.
Le variabili pertanto non avranno un unico assegnamento, bensì questo assegnamento cambia nel tempo.
Viene pertanto costruito quello che si chiama diagramma temporale per rappresentare la variazione delle variabili nel tempo.
Adesso ha quindi senso scrivere variabili booleane che dipendono da sé stesse.
Le reti sequenziali pertanto, a differenza delle reti combinatorie, calcolano funzioni booleane che variano nel tempo in base a variazioni dei valori di input e valori di uscita della rete ad istanti precedenti (circuiti con memoria).

Circuiti di memoria elementare sono i *Latch* (SR, D, JK, T).

Le reti sequenziali possono operare in modo sincrono o asincrono:

- nei sistemi *asincroni* i circuiti logici cambiano ogni volta che uno o più ingressi cambiano;
- nei sistemi *sincroni* i circuiti logici cambiano dipendentemente alla frequenza di un segnale chiamato segnale di "clock".

Le reti asincrone hanno il vantaggio di reagire istantaneamente alle variazioni degli input, ma questo rende molto più difficile progettare un sistema complesso in cui le operazioni devono essere eseguite in un ordine preciso.
Oggi i computer utilizzano solo reti sincrone.

I Gated Latch sono una variante dei Latch che utilizzano un clock come segnale di "cadenza". La differenza tra un Latch qualsiasi e un Gated Latch è che il Gated Latch ha come input l'AND tra il segnale clock e il segnale di input del Latch di base.

I Gated Latch però funzionano bene solo se il clock è ad uno per pochissimo tempo: se il clock rimane a 1 per troppo tempo, si ripropone lo stesso problema del Clock normale, e le variabili cambiano asincronamente al suo interno. In alternativa si utilizzano i Flip-Flop master-slave, ottenuti mettendo in cascata due gated latch, il primo collegato al clock e il secondo complementato.

Automi a stati finiti

I Flip Flop sono le reti sequenziali più semplici, ma già esibiscono le caratteristiche fondamentali di tali reti:

- memorizzazione di valori booleani (stato);
- modifica dei valori memorizzati in base ai segnali di ingresso.

Mentre le tavole di verità erano un formalismo per rappresentare reti combinatorie, si utilizzano altri metodi per rendere più intuitiva la rappresentazione di reti sequenziali.

Il primo metodo è quello di un *Sistema di Transizioni Etichettate*, una quadrupla (Q, sigma, q0 e delta minuscolo), dove:

- Q è l'insieme degli stati;
- Σ è l'alfabeto (di input);
- q_0 è lo stato iniziale;
- $\delta: Q\ x\ \Sigma\ \longrightarrow Q$ è la funzione di transizione.

  Essendo δ una funzione, data la stessa coppia di valori, lo stato di destinazione è sempre lo stesso.

Se al Sistema di Transizioni Etichettate aggiungo F (l'insieme degli stati finali) e impongo il vincolo che tutti gli insiemi considerati siano finiti, ottengo un automa a stati finiti, una quintupla.
Graficamente si rappresentano con un cerchio doppio.
Gli stati finali servono per indicare quali sequenze sono accettate dall'automa (e sono tutte quelle che portano l'automa dallo stato iniziale ad uno stato finale).

L'automa con output è una sestupla (con insiemi finiti), estensione del Sistema di Transizioni Etichettate che ha due elementi aggiuntivi: delta maiuscolo, l'insieme degli output, e lambda, una funzione di output.
Negli automi di Moore, la funzione di output dipende esclusivamente dallo stato; negli automi di Mealy, la funzione di output dipende invece dallo stato e dall'input.

Equivalenza dei modelli di Mealy e Moore

Definiamo due automi equivalenti in base al loro comportamento, astraendo quindi da come sono definiti. Si utilizza l'approccio "a scatola nera", cioè l'unico modo per studiare gli automi è facendo esperimenti:

- due automi NON sono equivalenti se riesco a trovare una sequenza di input che nei due automi genera output diversi; altrimenti sono equivalenti.

Due atomi dello stesso tipo si dicono *equivalenti* se per ogni possibile sequenza di input generano entrambi la stessa sequenza di output.
Due automi di tipo diverso sono equivalenti se, per ogni possibile sequenza di input, generano sequenze di output che differiscono esclusivamente per il primo carattere nell'output dell'automa di Moore.

Il problema è che, per dimostrare un'equivalenza devo considerare tutte le sequenze di input possibili, che sono infinite anche considerando l'alfabeto di input più piccolo possibile.
Un modo per dimostrare una proprietà P su tutte queste stringhe è usando il principio di induzione:

- PASSO BASE: dimostra la proprietà per la stringa vuota;
- PASSO INDUTTIVO: assumendo vera la proprietà P(n), si dimostra vera anche per P(n+1);

Teorema: Sia $M_1 = (Q, \Sigma, \Delta, q_0, \delta, \lambda)$ un automa di Moore; allora esiste un automa di Mealy ad esso equivalente.

Sia $M_2 = (Q, \Sigma, \Delta, q_0, \delta', \lambda'), dove\ \lambda'(q,a) = \lambda\left(\delta(q,a)\right)$
Sia σ una sequenza di input: dimostriamo per induzione sulla lunghezza di σ (cioè sul numero di caratteri presenti) che $M_1(\sigma) = c_0 M_2(\sigma)$, dove $c_0 = \lambda(q_0)$.

- PASSO BASE: $M_1(\varepsilon) = c_0\ e\ M_2(\varepsilon) = \varepsilon$
  $c_0\varepsilon = c_0$
- PASSO INDUTTIVO: Se σ è lunga n+1 caratteri, allora $\sigma = \sigma' a, per\ qualche\ a \in \Sigma; quindi, \sigma'$ è $lunga\ n.$
  $Per\ ipotesi\ induttiva$: $M_1(\sigma') = c_0 M_2(\sigma')$

Minimizzazione di automi

Due automi si dicono equivalenti se, per ogni possibile sequenza di input, generano entrambi la stessa sequenza di output.

L'automa minimo equivalente a M è quell'automa equivalente a M ma con il minimo numero possibile di stati. Si può dimostrare che questo automa è unico, a meno di ridenominazioni di stati.

Un automa è un modello astratto di una rete sequenziale. Quindi, conviene minimizzare un automa, poiché un numero minore di stati implica una riduzione del numero di componenti.

Il primo passo è la rimozione degli stati irraggiungibili, poi si passa all'unione degli stati indistinguibili. In un automa di Moore, due stati si dicono distinguibili se:

- gli output associati ad essi sono diversi (distinguibilità immediata);
- hanno una transizione uscente etichettata con lo stesso carattere che porta a stati distinguibili (distinguibilità propagata).

In un automa di Mealy, due stati si dicono distinguibili se:

- restituisce output diversi (distinguibilità immediata);
- porta a stati distinguibili (distinguibilità propagata).

Registri: caricamento / scaricamento, shifter e contatori

Un registro è un insieme di FF ordinati in cui memorizzare una parola da n bit.

Le problematiche relative ai registri sono:

- caricamento e scaricamento dell'informazione;
- le funzionalità che fornisce il registro;
- interconnessione tra registri.

In base alle modalità di scrittura e lettura dei dati possiamo trovare 4 categorie di registri:

- Parallel Input Parallel Output (PIPO)
- Serial Input Serial Output (SISO)
- Serial Input Parallel Output (SIPO)
- Parallel Input Serial Output (PISO)

Il registro PIPO possiede in input n linee di dati. I flip flop sono separati tra loro, ma sono collegati tutti ad un unico clock, messo in AND con un segnale chiamato di LOAD che specifica quando effettuare la lettura dei dati. Il valore memorizzato viene automaticamente restituito in output dopo tau.

Il registro SISO possiede in input una sola linea dati. L'uscita di un flip flop è collegata all'entrata del successivo, e tutti i flip flop sono collegati ad un unico clock, messo in and con un segnale di SHIFT, che determina quando deve essere effettuata la lettura dell'input. C'è un unico output, ossia il bit restituito dall'ultimo flip flop.

Il registro SIPO possiede in input una sola linea dati. L'uscita di un flip flop è collegata all'entrata del successivo, e tutti i flip flop sono collegati ad un unico clock, messo in and con un segnale di SHIFT, che determina quando deve essere effettuata la lettura dell'input. Ci sono n output, in parallelo.

Il registro PISO possiede in input n linee dati. L'uscita di un flip flop è collegata all'entrata del successivo e tutti i flip flop sono collegati ad un unico clock, messo in and con lo XOR tra il segnale di SHIFT e il segnale di LOAD. Il segnale di LOAD effettua la lettura in parallelo degli n bit che vengono memorizzati nei flip flop.

Indipendentemente dal tipo di caricamento / scaricamento, un registro può fornire delle funzionalità. Ci sono 2 tipi fondamentali di operazioni che un registro può fare:

- Shift
- Count

I registri di shifting permettono di effettuare moltiplicazioni e divisioni per due in modo molto rapido ed efficiente. Si perde però il bit più o meno significativo in base alla direzione dello shift, e il bit perso viene rimpiazzato con zero.

Analizziamo i differenti tipi di shifter:

- Right shifter (shift a destra), effettua la divisione e l'uscita di ogni flip flop è collegata all'entrata del successivo, escluso il primo che ha sempre 0 come input;
- Left shifter (shift a sinistra), effettua la moltiplicazione e collega l'entrata di ogni flip flop all'uscita del successivo, escluso l'ultimo che ha sempre 0 come input;
- Shifter bidirezionale, che attraverso un segnale di controllo right permette di effettuare lo shift o a sinistra o a destra, in base alla decisione che si prende;
- Shifter con mantenimento dei bit, semplicemente esegue lo shifting ma, al posto di aggiungere uno zero, aggiunge sempre l'uscita del primo flip flop (right shifter) o ultimo flip flop (left shifter). È utile per le divisioni o moltiplicazioni per due con complemento a due. Collega l'entrata del primo flip flop

alla sua uscita (right shifter) o l'entrata dell'ultimo flip flop alla sua uscita (left shifter), o entrambe (shifter bidirezionale).
- Registro circolare, inserisce i bit perduti durante lo shifting all'estremità opposta, creando un registro chiuso circolarmente su sé stesso. Basta collegare l'uscita dell'ultimo flip flop all'entrata del primo (right shifter o antiorario) o l'uscita del primo all'entrata dell'ultimo (left shifter o orario), o entrambi (shifter bidirezionale).

I registri contatori sono registri usati per contare il numero di occorrrenze di un determinato evento, sempre modulo un certo numero naturale.

Se formato da n FF, può contare fino a 2^n. Può contare colpi di clock o le occorrenze di alcuni valori (impulsi). Si distinguono in contatori:

- sincroni, se tutti i FF hanno lo stesso clock;
- asincroni, se l'uscita di un FF fa da clock per quello successivo o più in generale, hanno clock differenti.

Possono contare in ordine crescente o decrescente (o entrambi). Possono essere settabili. Solitamente si utilizzano flip flop di tipo T o JK.

Partiamo dai contatori sincroni:

- Contatore modulo 2^n crescente, conta da 0 fino a $2^n - 1$. L'entrata di ogni flip flop è collegata all' AND delle uscite dei flip flop precedenti. Tutti i flip flop sono collegati allo stesso clock.
- Contatore modulo 2^n decrescente, conta da $2^n - 1$ a 0. L'entrata di ogni flip flop è collegata all'AND delle uscite negate dei flip flop precedenti. Tutti i flip flop sono collegati allo stesso clock.
- Contatore modulo 2^n bidirezionale. In base ad un segnale di controllo chiamato Up_Down, si decide la direzione verso cui conta il contatore.
- Contatore modulo k generico, o si crea un automa e quindi si sintetizza il circuito da zero, cosa che può richiedere parecchio tempo, o si decide di utilizzare i contatori sincroni preselezionabili, in grado di settare o cancellare i bit memorizzati nei flip flop in ogni momento. I contatori sincroni preselezionabili sono realizzati con FF asincroni. Per effettuare il setting (PRESET) o la cancellazione (CLEAR) dei bit si utilizza una linea di controllo Parallel_Load.

I contatori asincroni sono uguali a quelli sincroni, l'unica differenza è che le entrate di ogni flip flop sono sempre impostate ad 1 e il clock di ogni flip flop è collegato all'uscita affermata (per i contatori ascendenti) o negata (per i contatori discendenti) del flip flop precedente.

Nei contatori asincroni si verifica un fenomeno chiamato ripple (mormorio), che per via di mancanza di sincronia tra i componenti crea per pochissimo tempo dei valori intermedi non corretti (false counts). Nei contatori asincroni poi, il ritardo si propaga e si accumula.