Capitolo 2

Informazione e mondo esterno

Abbiamo già ricordato che in molti casi un problema del mondo esterno può essere risolto descrivendolo mediante un'opportuna codifica astratta, sulla quale si applica nel seguito una procedura algoritmica di risoluzione. Se la codifica e la procedura sono realizzate correttamente, il risultato simbolico ottenuto dopo un tempo finito può essere decodificato nei termini di una soluzione al problema primitivo. Anche se i problemi che la macchina risolve sono di natura strettamente logico-matematica, la potenza espressiva della codifica simbolica consente di gestire problemi del mondo reale che possono spaziare fra tipologie molto diverse tra loro. Come esempio si possono confrontare il problema di manipolare alcune immagini digitali, per esempio per effettuare un foto-ritocco, con quello di trovare il percorso ottimale, tenendo conto dei sensi unici, che un furgone deve effettuare per approvvigionare i diversi supermercati di una grande città. O ancora il problema dell'individuazione di una o più parole chiave all'interno delle miliardi di pagine disponibili nella rete di Internet, tramite il cosiddetto *motore di ricerca*, con quello del calcolo della traiettoria ottimale per il lancio di un vettore spaziale. Come ultimo esempio, quello forse più estremo, citiamo la ricostruzione, fatta dalla macchina, di un ambiente virtuale come l'interno di una piramide, che può essere "visitato" comodamente stando seduti su una poltrona, come se la visita fosse fatta in prima persona. Da un punto di vista sistemico la situazione è allora quella descritta nella figura 2.1, dove viene messo in evidenza un ambiente esterno, quello del mondo reale, in cui le grandezze fisiche (o dati d'ingresso), associate implicitamente all'enunciazione e alla risoluzione di un certo problema, devono essere codificate in simboli comprensibili al calcolatore. I dati d'ingresso possono essere costituiti da lettere su un alfabeto discreto, tipicamente i simboli di una tastiera di un calcolatore o dalle cifre necessarie a comporre i numeri che si usano per fare i calcoli, ma anche da immagini o da da suoni o più in generale da segnali che possono derivare da strumenti di misura ecc. Come vedremo fra poco i calcolatori lavorano sulla base di un alfabeto binario, e ciò per tutta una serie di motivi che illustreremo nel seguito. Tutte queste informazioni devono dunque essere trasformate, tramite una *codifica*, in lunghe sequenze di *stringhe binarie*, l'unica tipologia d'informazione che i circuiti elettronici, basati sul sistema della logica Booleana, sono in grado di gestire.

Il calcolatore elabora, secondo l'algoritmo escogitato per risolvere il problema, i dati codificati d'ingresso, fornendo alla fine della procedura un risultato simbolico binario, che va poi decodificato nelle grandezze del mondo esterno che rappresentano la soluzione, in modo da poter essere lette e interpretate da esseri umani.

Dunque, qualunque sia l'algoritmo e comunque funzioni il sistema di elaborazione, il primo passo è quello di studiare il problema della codifica. Per farlo è necessario specificare la tipologia e le modalità di rappresentazione delle grandezze esterne da una parte, e considerare la tipologia di simboli che la macchina è in grado di gestire dall'altra. Cominciamo allora a descrivere le modalità di rappresentazione dei dati d'ingresso.

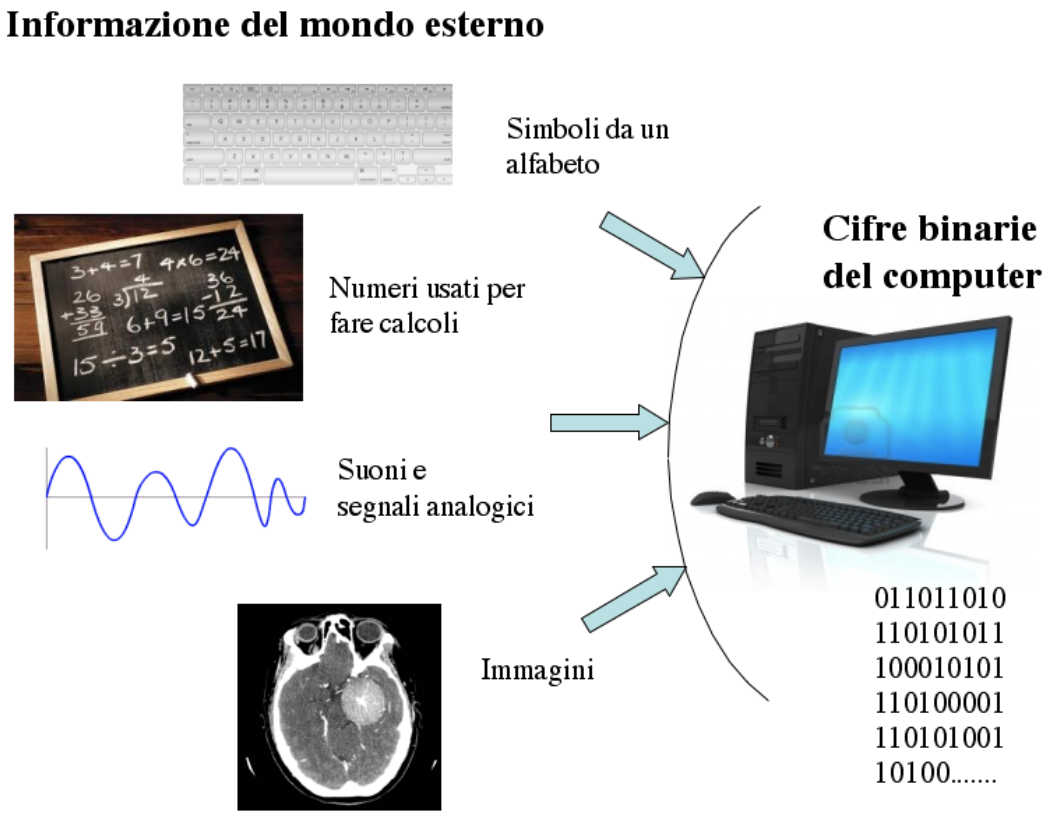


Figura 2.1: Tutte le informazioni del mondo esterno vanno codificate in lunghe stringhe binarie

2.1 Informazione e ridondanza

Quando acquisiamo un dato, la lettera di un alfabeto, un'immagine, un suono, un numero, ecc., stiamo acquisendo *informazione* dal mondo esterno. Se riflettiamo per un momento su quella che è la matrice primitiva dell'informazione, ci rendiamo conto che la sua generazione è in qualche misura associata alla variazione di una qualche grandezza fisica: la lettura di un testo scritto è possibile grazie alla nostra capacità di discriminare la variazione di luminosità tra le lettere nere che compongono il brano e il bianco della pagina; l'ascolto di una frase nel linguaggio parlato è reso possibile dalla differenza di pressione dell'aria generata, tramite le corde vocali, nella bocca del parlante; la percezione di un particolare ritenuto interessante in un immagine, p.es. una macchia chiara in una tomografia, è resa possibile dalla capacità del nostro occhio di valutare differenze di contrasto, e così via.

L'informazione nasce dunque come stato oggettivo per l'osservatore, nel momento in cui questi individua una *differenza* nella grandezza di riferimento, in corrispondenza di due istanti successivi o di due punti dello spazio. Inoltre la quantità d'informazione fornita nell'esperimento è intuitivamente legata alla numerosità degli stati possibili *a priori*. Una volta generata, l'informazione si propaga nell'ambiente esterno alla sorgente attraverso opportuni *canali* di trasmissione, che costituiscono il sostrato fisico attraverso il quale le differenze di cui prima si parlava si propagano al di fuori della sorgente. In generale il canale potrebbe essere un qualunque elemento fisico che consenta la lettura a distanza della configurazione assunta dalla sorgente, ma in pratica, perlomeno nel contesto tecnico-ingegneristico in cui di fatto si finisce con l'operare, esso è quasi sempre una linea di trasmissione (doppino telefonico, cavo coassiale, guida d'onda o fibra ottica), scelta sulla base della tecnologia in auge, e che consenta la trasmissione a distanza dei cosiddetti *segnali* che incorporano l'informazione che la sorgente genera. Si noti che la trasmissione può essere a distanza nello *spazio*, nel qual caso si connettono due punti distanti fisicamente, ma

anche nel *tempo*, quando si effettua una memorizzazione dell'informazione su un opportuno supporto.

L'informazione non ha però una valenza assoluta, poiché dipende in sostanza dalla capacità di discriminazione dell'osservatore a essa interessato; questi ha di fronte a sé un sistema fisico, la *sorgente*, e una (o più) grandezze che lo descrivono, suscettibili di assumere *più stati diversi*. L'informazione rilevata da un secondo osservatore, più acuto del primo e che sia in grado di discriminare tra più stati diversi della stessa grandezza (o tra più grandezze), rimane per il primo osservatore in uno stato di latenza, che si desta solo nel momento in cui anche il primo osservatore dispone della capacità fisico-sensoriale (e della volontà) di rilevare le differenze di cui prima era inconsapevole. Come esempio si può pensare all'esperimento "lancio di un dado", con un osservatore O_1 (che potrebbe essere un giocatore d'azzardo) interessato solamente al numero uscito, e un osservatore O_2 (p.es. uno studente di Fisica) che apprezza anche l'orientamento delle facce, la posizione occupata dal dado nell'istante finale ecc. I due punti di vista sull'informazione associata all'esperimento sono evidentemente molto diversi, e riflettono il diverso tipo di interesse manifestato dai due osservatori nei confronti dell'evento in oggetto. Si noti per altro che lo stato di latenza cui si è accennato precedentemente può derivare anche dall'indisponibilità, da parte dell'osservatore, dei rilevatori sensoriali idonei a cogliere la grandezza fisica associata al funzionamento della sorgente: l'accensione di una luce emessa da una lampada che lavora sull'infrarosso non potrà essere percepità da chi non disponga degli apposti occhiali per la visione notturna.

Le *differenze* di cui si parla hanno una loro strutturazione gerarchica, poiché può *cambiare* il modo in cui si manifesta una differenza; anche questa differenza del secondo livello (differenza di differenze) è rilevabile in modo oggettivo ed è essa stessa fonte d'informazione. Si osservi il segnale rappresentato in figura 2.2(a), che potrebbe essere relativo alla luminosità di una torcia, con la quale si lanciano dei segnali luminosi a distanza, di notte. Il passaggio tra torcia accesa e torcia spenta costituisce una variazione, e quindi un'informazione a livello base dei dati. Tuttavia una variazione nella frequenza delle accensioni/spegnimenti (fig.2.2(b) o (c)), è una variazione del secondo livello, poiché varia la modalità di variazione della luminosità. Riprendendo l'esempio del dado si può invece pensare a una successione di lanci, fatta dallo studente di Fisica, in cui ciascuna faccia abbia una frequenza relativa di (circa) 1/6, seguita da una seconda successione "sospetta", eseguita dal giocatore d'azzardo, in cui il 6 esca con una frequenza stranamente elevata. Se dal punto di vista delle differenze del prim'ordine ci si può limitare a registrare l'accaduto, un'analisi dell'informazione associata alle differenze del second'ordine, cioè al diverso "comportamento" del dado lanciato dal giocatore, porterebbe a un'informazione di secondo livello su una presunta manomissione del dado da parte del giocatore d'azzardo. Questa gerarchia si colloca comunque su un piano

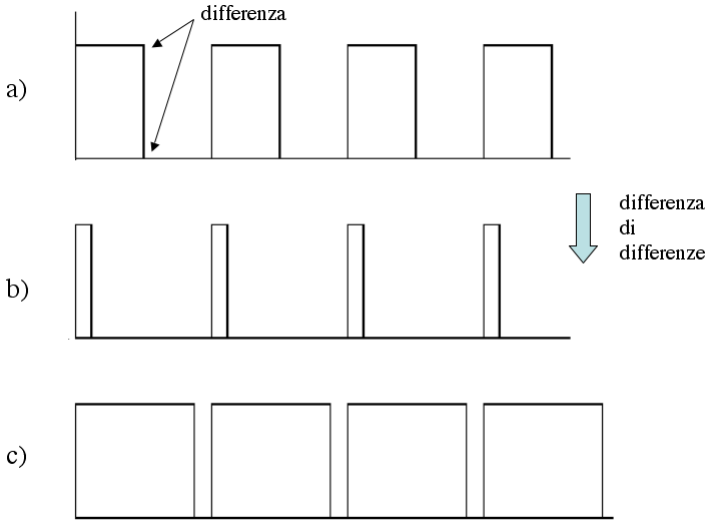


Figura 2.2: L'informazione di primo livello (la variazione del segnale) e quella di secondo livello (la variazione della variazione), che corrisponde in pratica a un cambio nella frequenza del segnale

strettamente *sintattico*, poiché riguarda direttamente i dati. Nell'esempio precedente l'informazione sintattica è

relativa al fatto che il comportamento del dado è cambiato; tuttavia la conseguente congettura sul dado truccato deriva da un'*interpretazione* delle differenze relativamente a un certo contesto logico, l'unico che possa esprimere un *significato* per esse. Ciò induce una gestione delle differenze anche su un piano *semantico*, molto difficile da gestire con un modello matematico formale.

L'informazione ha dunque un carattere relativo, dipende sensibilmente dall'utente che interroga la sorgente, dalla sua capacità di discriminazione, e presuppone una fase *sintattica* o di *rilevamento* della stessa, che si distingue nettamente dalla fase *semantica* relativa alla *comprensione* del significato, processo che consiste essenzialmente nel riportare ad una propria esperienza pregressa i caratteri sintattici emersi dall'informazione disponibile, riconducendo la massa disaggregata di dati verso quel piano di coerenza semantica che porta al *significato*. Come esempio illuminante si rifletta sulla lettura di una frase scritta in una lingua che non si conosce, come *Fartyget kommer att segla på16*, contrapposta alla lettura della stessa frase scritta nella propria lingua, cioè *La nave salperà alle ore 16*. In entrambi i casi avviene il rilevamento, ma la comprensione del significato della frase si ha solamente nel caso in cui l'osservatore conosca la lingua in questione.

Tutto quanto descritto finora ha come immediata conseguenza un'intrinseca difficoltà nel giungere a una quantificazione e a una gestione globale dell'informazione, che tenga cioè conto tanto dei suoi aspetti sintattici quanto di quelli semantici, che possono fra l'altro essere assai poco correlati tra loro.
Il modello di gestione e di trasmissione dell'informazione suggerito da Shannon, nel suo lavoro precedentemente citato, fa riferimento al solo piano sintattico, e ciò costituisce contemporaneamente un pregio e una limitazione. La limitazione è conseguente alla circostanza che, essendo svincolato da connotazioni di tipo semantico, il modello descrive solo in minima parte la ricchezza del processo di comunicazione, così come avviene nel mondo reale. Il pregio risiede nel fatto che tale limitazione, circostanziando la validità del modello, ha consentito di ottenere risultati rilevanti sul piano tecnico-operativo, che hanno dato un impulso straordinario allo sviluppo dell'informatica. Bisogna ricordare che ci sono stati e ci sono tuttora diversi tentativi di inserire la semantica nelle macchine o nei programmi (si ricordi la cosiddetta *Intelligenza Artificiale* o i *Sistemi Esperti*), ma finora i risultati sono stati piuttosto modesti. La semantica presuppone infatti la *conoscenza*, che attraverso l'apprendimento porta all'*esperienza*, dalle cui stratificazioni emerge il profilo semantico; ma questo processo richiede l'acquisizione di una quantità enorme di dati e, soprattutto, di *relazioni* tra i dati, e nessun calcolatore sembra ancora in grado di raggiungere vette così alte di complessità. Si pensi che il cervello umano contiene qualcosa come 10^{11} neuroni, e che ciascun neurone può essere connesso ad altri neuroni attraverso un migliaio di sinapsi, in modo che il numero totale di connessioni sinaptiche è dell'ordine di $10^{14} - 10^{15}$. Nei più sofisticati calcolatori, basati sull'integrazione a larga scala VLSI, il numero di elementi attivi (transistor) e di connessioni, che in una primissima e rozza approssimazione possiamo considerare equivalenti ai neuroni biologici e alle sinapsi, rimane confinato a un ordine di qualche unità per 10^9, e quindi c'è ancora un rapporto di almeno 10000:1 tra i due sistemi, quello biologico e quello artificiale. Ma naturalmente questo discorso vale solo in primissima approssimazione, poiché c'è anche da considerare che il cervello biologico non è un sistema rigidamente binario, e dunque per tener conto di questo fatto potrebbe essere necessario peggiorare ancora il rapporto visto prima.

L'informazione che riceviamo può in qualche caso essere in eccesso rispetto a quella strettamente necessaria per la comprensione del significato; in tal caso alla differenza fra le due si attribuisce il nome di *ridondanza*. Per esempio il completamento della frase "*Il primo mese dell'anno è Ge...*" non offre alcun tipo di difficoltà, poiché la conoscenza del significato in lingua italiana dei termini "primo", "mese" e "anno" porta univocamente a concludere che la parte mancante è "nnaio". Tuttavia, accanto a questa ricostruzione di tipo semantico, è possibile impostare anche una ricostruzione su base sintattica, basata sui semplici dati e senza nessun riferimento al loro significato. In tal caso la ridondanza si manifesta allorquando le lettere dell'alfabeto che si sta usando non hanno una probabilità uniforme di comparire. Si faccia riferimento alla tabella di figura 2.3, nella quale viene riportata la distribuzione delle frequenze relative delle lettere dell'alfabeto italiano nel romanzo *I promessi sposi*, considerati un riferimento per la nostra lingua. Si può notare che non tutte le lettere compaiono con la medesima frequenza, e anzi certe lettere sono piuttosto rare. Se approfondiamo l'analisi andando a calcolare le frequenze relative delle coppie, delle terne ecc. tipiche della lingua italiana, ci renderemo conto che la forbice fra combinazioni di lettere verosimili o anche solo possibili e quelle che non lo sono aumenta. Già l'analisi del secondo ordine ci mostra che alcune combinazioni sono vietate (p.es. la "qr" o la "mc" ecc), e altre sono esclusive, quali ad esempio la "qu", il che significa che una "q", in italiano, è quasi sempre seguita da una "u" (ci sono le eccezioni della doppia "q", come in "soqquadro" e "aqquartieramento").

E	0.12059	L	0.05567	V	0.02305
A	0.11512	S	0.05459	G	0.01713
O	0.09640	C	0.04692	H	0.01352
I	0.09530	D	0.03731	F	0.01052
N	0.07292	U	0.03569	B	0.00973
R	0.06599	P	0.02967	Q	0.00779
T	0.06083	M	0.02365	Z	0.00760

Figura 2.3: Tabella delle frequenze relative delle lettere in Italiano (ricavate da *I promessi sposi*).

Se si volesse ricostruire una frase basandosi su elementi sintattici, sarebbe allora necessario stimare la lettera (o il gruppo di lettere) che hanno la massima probabilità a priori di comparire subito dopo "Ge". Supponiamo che dalla lettura della tabella del terz'ordine emerga che la terna più probabile a priori che inizia con "Ge" sia "Ger". A questo punto dovremmo valutare la terna più probabile che inizia con "er", che supponiamo essere "ero", e così via. Il processo si può bloccare dopo aver raggiunto quella che può essere considerata una lunghezza tipica di una parola italiana. Se il risultato dell'esperimento di ricostruzione sintattica è la parola "Gerosa", è chiaro che la frase "*Il primo mese dell'anno è Gerosa*" è priva di significato in lingua italiana, ma tuttavia la parola ricostruita *sembra* essere una parola italiana, e ciò perché essa ha impressi i caratteri sintattici (statistici) della lingua italiana.

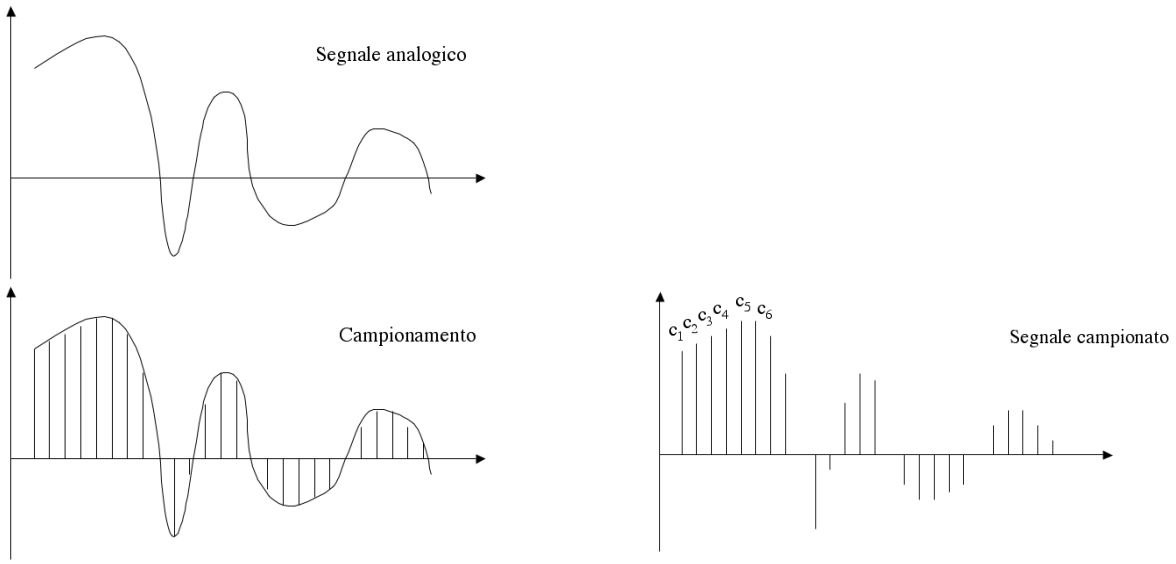
Riassumendo si può affermare che l'informazione non coincide col supporto, poiché può essere trasferita facilmente da un supporto all'altro, non segue le leggi di conservazione tipiche della fisica, poiché la sua distribuzione non la diminuisce, non è una grandezza fisica misurabile nel *Sistema Giorgi*, poiché è adimensionale, dipende dal contesto e dall'osservatore, almeno quando la si consideri per i suoi aspetti semantici. Si noti inoltre che anche l'assenza di informazione può essere informazione (0 e 1 sono tra loro alternatitivi e mutuamente escludentisi), poiché 0 è diverso da 1. L'informazione si sviluppa dunque nell'interazione tra supporto modulato dalle differenze e osservatore.

2.2 Informazione analogica e informazione discreta

L'informazione generata dalle varie sorgenti, impressa nei segnali che transitano sui vari canali, può avere tanto natura *continua* che *discreta*. Nel primo caso la grandezza fisica di riferimento cambia con continuità nel tempo, e il valore che assume istante per istante è dato da un numero reale. La precisione con la quale verrà poi letto questo valore in un certo istante, impiegando un opportuno strumento di misura, dipenderà essenzialmente dalla *classe* dello strumento, cioè dalle sue caratteristiche certificate di accuratezza e precisione.
Nel caso discreto, invece, l'informazione viene associata all'emissione di una lettera appartenente a un certo alfabeto finito, che potrebbe essere quello della lingua italiana nel caso di un testo di un romanzo, quello delle cifre da 0 a 9 nel caso si debbano fare dei calcoli, o quello della tastiera del computer (molto più ricco, e che comprende come sottoinsiemi i due alfabeti appena citati) nel caso si debba lavorare con dei programmi di videoscrittura.
La distinzione tra analogico e discreto (o *digitale*) si estende ovviamente anche ai sistemi elettronici che elaborano l'informazione, e si parla pertanto di sistemi *analogici* e di sistemi *digitali* (o *numerici*).

Nei primi decenni del novecento le apparecchiature per la manipolazione dell'informazione nelle sue varie forme (audio e video, principalmente) erano tutte analogiche (radio, televisori, telefoni ecc.); la parola *analogico* è stata coniata per ricordare che i segnali elettrici, che rappresentano le varie grandezze coinvolte nella riproduzione dell'informazione, seguono in modo *analogo* la grandezza fisica primitiva (la pressione acustica nel caso di un segnale audio, l'intensità luminosa nel caso di un segnale video ecc.). La manipolazione analogica dei segnali audio e video è coerente col fatto che questa informazione *nasce* in forma analogica, ed è del tutto ovvio gestirla usando lo stesso ambito.
L'informazione discreta era all'epoca confinata nel solo ambito specialistico e ristretto dei primi calcolatori, la cui logica di funzionamento, per quanto abbiamo anticipato nel paragrafo dedicato agli aspetti storici, era di tipo logico-matematico, e quindi basata su insiemi discreti di simboli.
Con l'andar del tempo si è però assistito a un progressivo trapasso dalle tecniche analogiche a quelle digitali, anche

in quegli ambiti dove l'informazione nasce in forma continua. Ciò è stato reso possibile dal celebre *teorema del campionamento* di Nyquist, risultato centrale nella teoria delle comunicazioni elettriche, che stabilisce la possibilità di *approssimare* un segnale analogico con uno discreto, con un errore asintoticamente nullo. Il campionamento prevede la lettura dei valori (campioni) del segnale analogico, nel suo sviluppo temporale, con una frequenza f_c denominata *frequenza di campionamento* (fig.2.4a). Il valore numerico espresso per ogni campione è già in forma



(a) Campionamento di un segnale analogico

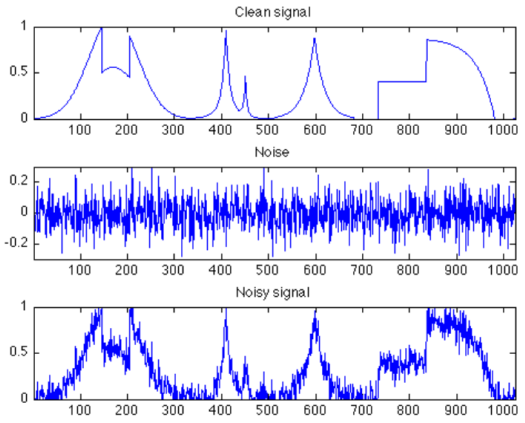
(b) Segnale campionato

Figura 2.4: Campionamento di un segnale analogico secondo il teorema di Nyquist

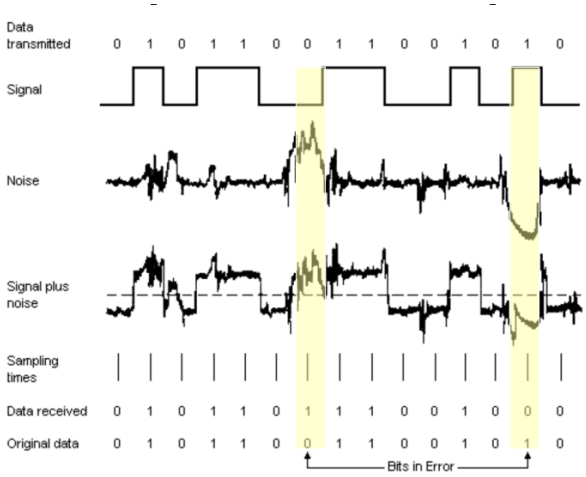
digitale, e se i campioni sono in numero sufficiente per ogni unità temporale (cioè se le frequenza di campionamento è sufficientemente alta), allora il teorema ci garantisce la possibilità di ricostruire in modo esatto il segnale continuo, senza altra perdita d'informazione che quella implicita nell'imprecisione della lettura (che porta al cosiddetto *rumore di campionamento*).
La possibilità di rielaborare i segnali convertiti in forma digitale mediante i processori, ha spinto a ricorrere sempre più frequentemente alle tecniche di *conversione analogico-digitale* (o conversione A/D); tale spinta è stata sostenuta anche dal fatto che, in genere, le apparecchiature digitali sono più affidabili di quelle analogiche, e ciò ha decretato di fatto l'abbandono della tecnologia analogica a favore di quella digitale in tutti i settori più importanti. Naturalmente nei casi in cui l'informazione nasca in forma analogica e debba poi essere riprodotta nella stessa veste, una volta effettuata la conversione A/D e le necessarie elaborazioni è necessario effettuare una successiva ri-*conversione digitale-analogica* (o conversione D/A), per poter recuperare il segnale analogico nella forma riproducibile presso l'utente, p.es. in quella di un segnale audio telefonico.

Un altro importante vantaggio delle apparecchiature digitali rispetto a a quelle analogiche è dato dalla loro resistenza al *rumore*. Ogni segnale elettrico gestito da circuiti elettronici viene irrimediabilmente inquinato da un segnale spurio denominato rumore, dato dalla sovrapposizione di numerosi segnali indotti, per la *legge di Lenz*, all'interno delle maglie chiuse dei circuiti elettronici. L'induzione avviene perché nello spazio nel quale siamo immersi esiste una moltitudine di onde elettromagnetiche dalle frequenze più disparate, sia di tipo artificiale (telefonia mobile, reti wi-fi, servizi radiofonici e televisivi, ...), sia di tipo naturale (radiazione solare, scariche atmosferiche, ...). La variazione del campo elettromagnetico esterno induce per la legge di Lenz una differenza di potenziale nel circuito interessato, che sviluppa una corrente spuria. Non dimentichiamo però che accanto al rumore di natura elettromagnetica c'è un'importante componente dovuta al *rumore termico* di agitazione degli elettroni, dovuto al fatto che i circuiti elettronici si trovano a una temperatura ambiente ben maggiore dello zero assoluto ($-273,16$ °C). Quando il segnale d'informazione viene colpito dal rumore, il segnale che ne deriva è dato dalla somma dei due, istante per istante. L'effetto è ben visibile in figura 2.5a e a questo punto non è più possibile separare i due segnali (a meno di casi particolari in cui stiano su bande diverse di frequenza); la compromissione

diviene irreversibile, e l'unico modo per preservare il contenuto informativo dall'inquinamento del rumore è quello di incrementare il rapporto segnale/rumore, amplificando il segnale d'informazione *prima* che venga attaccato dal rumore.



(a) Nei segnali analogici il rumore si somma al segnale in modo irreversibile

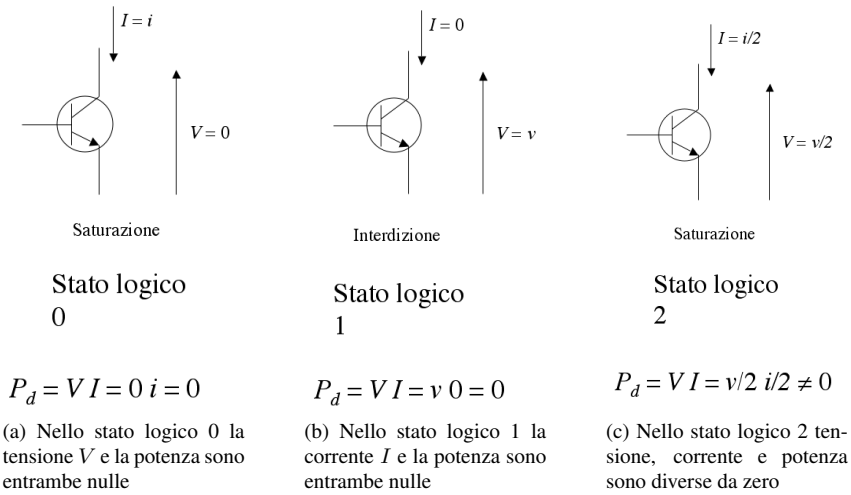


(b) Nei segnali digitali il segnale si può ricostruire in modo esatto, con una probabilità che può essere portata a valori prossimi a 1

Figura 2.5: Effetto del rumore sui segnali analogici e digitali

Viceversa, se un segnale associato a un alfabeto discreto viene compromesso dal rumore, è sempre possibile ricostruire l'unità elementare d'informazione (1 o 0) usando l'espediente di valutare l'area sottesa dal segnale intorno al suo valore medio (vedi figura 2.5b); il segnale discreto d'informazione potrà quindi essere ricostruito in modo esatto.
All'interno dei sistemi digitali si è poi assistito a uno sviluppo straordinario dei sistemi *binari*, motivati principalmente dall'esistenza dell'*algebra di Boole*, che come ricordato nel precedente capitolo ha fornito ai progettisti gli strumenti più idonei per progettare circuiti anche molto complessi. Si noti però che esistono anche motivazioni di carattere meramente tecnico che privilegiano la scelta di una base binaria rispetto altre basi ipotetiche, quali ad esempio quella ternaria, esadecimale o altro.
Come vedremo tra poco la scelta del binario comporta anche una maggiore affidabilità dei circuiti sui quali si basano le apparecchiature digitali.

A livello circuitale l'informazione binaria è associata allo stato di funzionamento dei dispositivi attivi (transistor), che sono in grado di controllare, tramite la base, il flusso di corrente che si sviluppa tra collettore ed emettitore. Contrariamente a quanto avviene per i *relè*, tale controllo è possibile con un dispendio minimo di energia e con continuità tra un valore minimo e un valore massimo. Un transistor correttamente polarizzato può trovarsi in uno stato di conduzione piena, o *saturazione* (fig.2.6a), in quello di *interdizione* (fig.2.6b) o in quello di conduzione parziale (fig.2.6c). Durante la saturazione del circuito collettore-emettitore la corrente I assume il suo valore massimo, mentre la tensione è nulla. Viceversa durante l'interdizione non scorre alcuna corrente, mentre la tensione V ai capi dei due elettrodi assume il suo valore massimo. Nello stato intermedio di conduzione parziale, entrambe le grandezze V e I sono diverse da zero. Se ora si vuole associare un'informazione binaria a un transistor, il modo più logico è quello di scegliere una delle grandezze che caratterizza il suo funzionamento, p.es. la tensione V, e considerare che il transistor sta nello stato logico 1 quando V assume il valore massimo, mentre sta nello stato logico 0 quando la V va a zero. Con questa convenzione lo stato di interdizione corrisponde allo stato 1, mentre quello di saturazione corrisponde allo stato 0. Se andiamo a valutare la potenza P_d dissipata dal transistor nelle due condizioni logiche 1 e 0 (interdizione e saturazione), ricordando che essa è data per la legge di *Joule* dal prodotto tra tensione e corrente, si avrà per la saturazione $P_d = V \cdot I = 0 \cdot I = 0$ e per l'interdizione $P_d = V \cdot I = V \cdot 0 = 0$. Ciò significa che in entrambi gli stati logici 0 e 1 *il transistor non dissipa potenza*. Se



(a) Nello stato logico 0 la tensione V e la potenza sono entrambe nulle

(b) Nello stato logico 1 la corrente I e la potenza sono entrambe nulle

(c) Nello stato logico 2 tensione, corrente e potenza sono diverse da zero

Figura 2.6: Potenza dissipata nei tre stati logici

ora volessimo introdurre un terzo stato logico, p.es. lo stato 2, per esso si dovrebbe individuare uno stato fisico di funzionamento del transistor che fosse massimamente distinguibile dagli stati logici 1 e 0, p.es. quello di una conduzione parziale di figura 2.6c, con tensione $V/2$ e corrente $I/2$. In questo caso, però, la potenza dissipata dal transistor sarebbe $P_d = V/2 \cdot I/2 \neq 0$, cioè maggiore di zero. Questo significherebbe la necessità di provvedere ad uno smaltimento del calore generato dal transistor, e dato che nei moderni circuiti integrati basati su una integrazione a larga scala ci sono, come ricordavamo nello scorso paragrafo, anche 10^9 transistor, si avrebbe un'insostenibile produzione di calore, con corrispondente aumento della temperatura di funzionamento delle giunzioni dei transistor, sulle quali si sviluppa la potenza. La temperatura è la principale nemica dei dispositivi a semiconduttore, poiché una sua deriva verso valori superiori a quelli massimi tollerati dal silicio, che sta intorno ai $180 - 200°$C, porta alla fusione della giunzione cristallina e alla distruzione del dispositivo. Ecco allora che queste motivazioni di carattere tecnologico, coniugate con lo sviluppo della logica Booleana, hanno portato al predominio assoluto dei dispositivi binari rispetto a qualunque altra soluzione. Anche i circuiti analogici, che vedono funzionare i transistor nelle condizioni di conduzione parziale, sono da questo punto di vista estremamente più critici e inaffidabili.
Concludendo questa analisi possiamo dire che il teorema del campionamento ci dà la possibilità di digitalizzare tutta l'informazione che nasce in forma analogica; la logica Booleana ci consente un'accurata progettazione dei circuiti binari; l'assenza di pericolose derive termiche legata alla scelta binaria aumenta in modo notevole la loro affidabilità. L'impiego sinergico di questo tre fattori ha di fatto decretato la sparizione delle tecnologie analogiche a favore di quelle digitali-binarie. Anche il nostro calcolatore, nato per altro per gestire informazione discreta, ha quindi una logica di funzionamento di tipo binario. Nei prossimi paragrafi dovremo allora occuparci di trasformare tutta l'informazione del mondo esterno in una modalità binaria.

2.3 L'alfabeto del calcolatore

Si è più volte sottolineata la natura simbolica degli oggetti che il calcolatore è in grado di gestire. I simboli in questione appartengono a un insieme, detto *alfabeto*, e ogni calcolatore può gestire uno o più alfabeti contemporaneamente. Può per esempio accettare in ingresso i simboli della tastiera immessi da un operatore, lavorare con un alfabeto binario $\mathbf{B} = \{0, 1\}$ a livello circuitale e magari rappresentare le informazioni che stanno sul disco rigido usando un alfabeto esadecimale del tipo $\mathbf{E} = \{0, 1, 2, \ldots, 9, A, B, C, D, E, F\}$. Tutti gli alfabeti visti finora hanno un numero finito di lettere, e si dicono *finiti*; per quanto esistano anche alfabeti infiniti, non avremo mai bisogno di usarli, e dunque tutte le considerazioni che faremo nel seguito riguarderanno solo alfabeti finiti.

L'esistenza delle diverse lettere è funzionale, più che altro, alla necessità di poter *distinguere* tra diversi

oggetti (o stati) che alle lettere vengono associati dal procedimento di codifica simbolica. Per fare un esempio banale potremmo codificare la posizione di una colonnina termometrica, che misuri le temperature di un ambiente condizionato, attribuendo all'intervallo $16 - 25°C$ una codifica nell'alfabeto $\mathbf{D} = \{0, 1, 2, \ldots, 8, 9\}$, in modo tale che 0 indichi 16, 1 indichi $17 \ldots 9$ indichi 25. In questo caso bisogna distinguere tra 10 stati, e possiamo usare le dieci cifre arabe. Il caso più semplice è quello in cui gli stati possibili sono solo due.
Per riprendere l'esempio della misura di una temperatura, potremmo riferirci alla situazione in cui sia sufficiente distinguere tra temperature sopra o sotto zero (lo zero lo poniamo per esempio in quest'ultima classe). In tal caso l'insieme che rappresenta i possibili stati consta di soli due elementi, che possiamo identificare convenzionalmente come $\mathbf{Z} = \{R, B\}$, dove R e B indicano rispettivamente i colori rosso e blu, tradizionalmente associati all'idea di caldo e freddo. Nessuno c'impedice però di usare altri simboli, per esempio $\mathbf{B} = \{0, 1\}$, poichè l'importante non è tanto il simbolo di per sé, che ha natura convenzionale, quanto piuttosto il numero di stati possibili da rappresentare, che in questo caso sono due. Si noti che la distinzione tra due stati diversi è minimale, e dunque l'alfabeto binario è il più piccolo alfabeto che si possa impiegare per distinguere tra situazioni diverse. Gli alfabeti che si usano poi effettivamente dipendono in modo essenziale dal contesto d'uso. Per esempio è ovvio usare l'alfabeto italiano di 21 simboli per scrivere frasi in tale lingua, mentre se si passa alla lingua inglese sarà necessario usare un alfabeto più ricco, contenente 26 simboli (in inglese ci sono le lettere j, k, x, y, w che mancano in italiano).

E' interessante osservare che, per quanto gli alfabeti possano essere apparentemente molto diversi l'uno dall'altro, la loro essenza simbolica ci consente di trasformare un alfabeto in un altro, ricorrendo sempre alla tecnica della codifica. In tal modo un alfabeto quaternario del tipo $Q = \{A, B, C, D\}$ può essere espresso nei termini di un alfabeto binario, per esempio usando coppie di cifre binarie, ottenendo $Q = \{00, 01, 10, 11\}$. Si noti che in tal caso si è di fatto riusciti a esprimere un alfabeto quaternario usandone uno binario. Poiché le due cifre $0, 1$ non bastano per distinguere tra i quattro stati A, B, C, D, si è ricorso al semplice espediente di impiegare più cifre binarie, facendo aumentare le combinazioni con le quali quest'ultime possono essere organizzate. Anche quest'aspetto verrà approfondito nel seguito; per ora ci basti riflettere sulla circostanza, solo apparentemente banale, che *con un alfabeto binario si può rappresentare potenzialmente un qualunque alfabeto (finito), pur di usare blocchi binari di lunghezza adeguata*.
Nella precedente sezione 2.2 abbiamo giustificato ampiamente il fatto che i calcolatori sono costretti a lavorare con un alfabeto binario a livello circuitale. Se dunque le informazioni del mondo esterno di figura 2.1 devono essere codificate in stringhe binarie, bisogna trovare una modalità per effettuare la codifica qualunque sia la tipologia di informazione con la quale abbiamo a che fare, analogica o discreta che sia.

Sulla base della citata figura possiamo distinguere tra quattro tipologie diverse d'informazione che possiamo accettare sui dispositivi o *periferiche* d'ingresso del computer:

- *Lettere e simboli grafici* vari derivanti dalla tastiera, dalla connessione *Internet*, da memorie esterne quali chiavette USB, ecc. (informazione discreta)
- *Numeri* usati per effettuare calcoli (informazione discreta) quando si impiega software matematico
- *Segnali analogici* di varia natura che costituiscono il segnale d'uscita di un *trasduttore* che trasforma una grandezza fisica che vogliamo rilevare (temperatura, pressione acustica, luminosità, contrasto, ...) in un segnale elettrico (tensione, corrente, resistenza, capacità, induttanza, ...)
- *Immagini* acquisite con una telecamera (informazione analogica)

2.3.1 Lettere e simboli grafici dalla tastiera

Consideriamo ora il primo caso, quello dei simboli della tastiera. È noto che bisogna distinguere tra lettere maiuscole e minuscole (A, a, B, b, ...), segni di interpunzione (, . : ...), numeri (0, 1, 2, ..., 9), ma ci sono anche dei simboli speciali, quale il tasto "invio", il tasto "tabulatore", ecc. Si noti tuttavia che questi simboli potrebbero non derivare direttamente dall'operatore che sta alla tastiera, poiché potrebbero essere i simboli di un generico file di testo che è memorizzato nel nostro disco rigido e che vogliamo rielaborare, oppure che è memorizzato su un

supporto esterno di memoria, o ancora che deriva dalla ricezione di un pacchetto dati dalla rete *Internet* mediante un cavo o mediante *wi-fi*. Per mantenere il nostro discorso a un livello astratto, chiamiamo *sorgente d'informazione* il dispositivo (disco rigido, chiavetta USB, porta Ethernet...) che genera la sequenza di simboli nel nostro alfabeto di riferimento. Chiamiamo allora *codifica di sorgente* la traduzione dall'alfabeto della sorgente, che in generale può essere assimilato all'alfabeto di un linguaggio naturale, all'alfabeto di funzionamento del sistema digitale (calcolatore, canale di trasmissione, memoria,...), che è legato a considerazione di carattere tecnico-operativo, per esempio alla circostanza che la tecnologia elettronica, che incarna il funzionamento dei varî dispositivi, è essenzialmente legata a una logica binaria, e quindi a un alfabeto di due simboli.
Sia allora $\mathcal{A} = \{a_1, a_2, \ldots, a_K\}$ l'alfabeto della sorgente, o alfabeto *primario*, e $\mathcal{B} = \{b_1, b_2, \ldots, b_D\}$ l'alfabeto sul quale si basa il funzionamento del sistema digitale, o alfabeto *secondario*. Di solito $D = 2$ e $K > D$. La traduzione tra le sequenze sui due alfabeti avviene a opera di un dispositivo chiamato *codificatore*; essa viene attuata in modo tale che dalla sequenza secondaria che esce dal *codificatore* si riesca a ricavare in modo univoco la sequenza primaria generata dalla sorgente; tale condizione si esprime affermando che il codice dev'essere *univocamente decodificabile* (u.d.). È ovvio che l'ipotesi di univoca decodificabilità risulta irrinunciabile.
Esigenze di carattere economico richiedono, in qualche caso, che la traduzione sia accompagnata da una efficienza che si può misurare sulla base del rapporto tra lunghezza (media) della sequenza secondaria e lunghezza (media) della sequenza primaria. Questa efficienza non è di per sé irrinunciabile, ma sicuramente fortemente auspicabile. Infatti l'occupazione della memoria o del canale di trasmissione ha un costo che si può assimilare proporzionale alla lunghezza della sequenza secondaria. Il problema generale da risolvere, nell'ambito della codifica di sorgente, è allora quello di individuare una strategia di traduzione u.d. che comporti un'elevata efficienza, cioè un'elevata *compressione dei dati*. In qualche caso, però, è sufficiente effettuare una semplice *traduzione* tra i due alfabeti, senza cioè richiedere una compressione dati. E' questo il caso dei simboli della tastiera, quando vengono immessi codificati sotto forma binaria all'interno del calcolatore.
Passiamo ora alla descrizione del modello vero e proprio [9]. Sia $\mathcal{A}^+$ l'insieme di tutte le sequenze finite (stringhe) costruite con elementi di $\mathcal{A}$ e $*$ la legge di composizione interna per i suoi elementi data dalla *concatenazione* tra gli stessi; così se $\boldsymbol{\alpha_1}, \boldsymbol{\alpha_2} \in \mathcal{A}^+$, anche $\boldsymbol{\alpha_1} * \boldsymbol{\alpha_2} \in \mathcal{A}^+$. Se consideriamo ora l'insieme $\mathcal{B}^+$ associato all'alfabeto $\mathcal{B}$ delle sequenze secondarie si definisce *codice astratto* una qualunque applicazione

$$\varphi : \mathcal{A}^+ \to \mathcal{B}^+$$

Questa definizione è però troppo generale e porta a un codice impraticabile, poiché pone in corrispondenza gli elementi di due insiemi infiniti. Tuttavia essa può essere circostanziata al caso in cui la φ sia un *omomorfismo*; se ciò accade si ha una semplificazione notevole, poiché in tal caso $\varphi(\boldsymbol{\alpha_1} * \boldsymbol{\alpha_2}) = \varphi(\boldsymbol{\alpha_1}) * \varphi(\boldsymbol{\alpha_2})$ (si assume che anche $\mathcal{B}^+$ abbia come legge di composizione interna la $*$). In altre parole la *codifica* di una sequenza di stringhe primarie concatenate si ottiene mediante *concatenazione delle codifiche* (basate sull'alfabeto secondario) delle singole stringhe primarie. Come conseguenza di ciò verificheremo che è sufficiente stabilire una corrispondenza tra gli elementi di due insiemi finiti, quello dei *messaggi*, $\mathcal{M} = \{m_1, m_2, \ldots, m_{_T}\}, m_i \in \mathcal{A}^+$, che contiene stringhe (eventualmente a lunghezza variabile) costruite sulle lettere di $\mathcal{A}$, e l'insieme dei *valori* (o *parole di codice*) che la φ fa assumere a tali messaggi. Ecco allora che accanto all'insieme primario $\mathcal{M} = \{m_i\}_{i=1}^{K}$ considereremo il *dizionario*

$$\mathcal{W} = \{\varphi(m_i)\}_{i=1}^{K} \quad \varphi(m_i) \in \mathcal{B}^+$$

delle *parole di codice* (eventualmente a lunghezza variabile) associate a $\mathcal{M}$. In particolare i messaggi possono ridursi a singole lettere, e in questo caso il dizionario rappresenta la codifica dell'alfabeto primario; la costruzione di $\mathcal{M}$ è allora immediata, poiché $\mathcal{M} = \mathcal{A}$.

Definizione 2.1. *Si definisce codice la terna* $\mathcal{C} = \{\mathcal{M}, \varphi, \mathcal{W}\}$.

Si noti che la costruzione di $\mathcal{M}$ non è in generale banale: bisogna infatti fare in modo che con i suoi elementi si possa comporre una qualunque stringa $\boldsymbol{\alpha} \in \mathcal{A}^+$; ciò equivale a dire che qualunque sia la sequenza $\boldsymbol{\alpha}$ emessa dalla sorgente deve essere possibile effettuare una sua *segmentazione* negli elementi di $\mathcal{M}$.

Definizione 2.2. *Una famiglia* $\mathcal{M} = \{m_1, m_2, \ldots, m_{_T}\}$ *di messaggi si dice esauriente se, qualunque sia la successione* $\boldsymbol{\alpha}$ *semi-infinita a destra costruita con gli elementi di* $\mathcal{A}$*, esiste sempre almeno un prefisso di* $\boldsymbol{\alpha}$ *che appartiene a* $\mathcal{M}$.

L'univoca decodificabilità può essere introdotta imponendo *l'iniettività* della funzione di codifica φ

Definizione 2.3. *Un codice* $\mathcal{C} = \{\mathcal{M}, \varphi, \mathcal{W}\}$ *è univocamente decodificabile se,* $\forall\, \boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathcal{A}^+$, $\boldsymbol{\alpha} \neq \boldsymbol{\beta}$ *implica* $\varphi(\boldsymbol{\alpha}) \neq \varphi(\boldsymbol{\beta})$.

Per quanto attiene i codici di sorgente distingueremo gli stessi sulla base della lunghezza (costante o variabile) dei messaggi e delle parole di codice. Ci sono dunque quattro possibilità:

Codici blocco-blocco (*B-B*). In questo caso i messaggi si riducono a blocchi di lunghezza costante pari a n, cioè $\mathcal{M} = \mathcal{A}^n$ per $n \geqslant 1$; anche la lunghezza delle parole di codice è costante;

Codici blocco-lunghezza variabile (*B-LV*). Le parole di codice $\varphi(m_i)$ sono a lunghezza variabile, mentre i messaggi hanno lunghezza costante pari a n. Molto spesso $n = 1$, e la codifica a lunghezza variabile riguarda le singole lettere di $\mathcal{A}$. Se $n > 1$ allora i messaggi sono tutte le possibili K^n n-ple di $\mathcal{A}^n$.

Codici lunghezza variabile-blocco (*LV-B*). I messaggi sono a lunghezza variabile e bisogna garantire l'esaurienza. Le parole di codice sono invece a lunghezza costante.

Codici lunghezza variabile-lunghezza variabile (*LV-LV*). È il caso più generale: a ciascun messaggio primario di lunghezza variabile si fa corrispondere una parola di codice anch'essa di lunghezza variabile.

Esempio 2.1. Sia $\mathcal{A} = \{a, b, c\}$ e $\mathcal{B} = \{0, 1\}$. Effettuiamo una codifica per ciascuno dei quattro casi precedenti, supponendo che la sequenza emessa sia $\boldsymbol{\alpha} = bbacab$.

B-B Sia $\mathcal{W} = \{\varphi(a), \varphi(b), \varphi(c)\} = \{00, 01, 10\}$. La codifica va fatta sulle singole lettere: $\varphi(bbacab) = 01/01/00/10/00/01$.

B-LV Se $\mathcal{W} = \{\varphi(a), \varphi(b), \varphi(c)\} = \{0, 10, 11\}$, allora $\varphi(bbacab) = 10/10/0/11/0/10$.

LV-B In questo caso bisogna costruire una famiglia di messaggi a lunghezza variabile; poniamo p.es. $\mathcal{M} = \{aa, ab, ac, b, c\}$. La funzione di codifica associa ai messaggi parole di codice a lunghezza costante, p.es. $\mathcal{W} = \{\varphi(aa), \varphi(ab), \varphi(ac), \varphi(b), \varphi(c)\} = \{000, 001, 010, 011, 100\}$. La segmentazione della sequenza primaria è allora $b/b/ac/ab$ e la successione delle parole di codice emesse dal codificatore è $011/011/010/001$.

LV-LV Manteniamo la stessa famiglia di messaggi vista prima $\mathcal{M} = \{aa, ab, ac, b, c\}$. La funzione di codifica associa a ciascun messaggio parole di codice a lunghezza variabile, p.es. $\mathcal{W} = \{\varphi(aa), \varphi(ab), \varphi(ac), \varphi(b), \varphi(c)\} = \{000, 001, 01, 10, 11\}$. Con la stessa segmentazione in messaggi del caso precedente si ottiene la seguente successione secondaria: è $10/10/01/001$. ◯

Dall'esempio si noti come, nel caso in cui le parole di codice siano a lunghezza costante, l'omomorfismo della funzione di codifica sposti la richiesta di univoca decodificabilità dalle sequenze alle singole parole di codice; in tal caso è infatti sufficiente fare in modo che esse siano tutte distinte.

Osservazione 2.1. Un discorso analogo vale anche per la *segmentazione* delle sequenze primarie nel caso di codici con messaggi a lunghezza variabile; la principale differenza operativa consiste nel fatto che in questo caso una segmentazione non univoca (mancanza di univoca *codificabilità*) non è pregiudizievole, poiché si può scegliere una delle due (o più) segmentazioni secondo opportuni criteri di convenienza.

Osservazione 2.2. Mentre la condizione u.d. è irrinunciabile per il dizionario di una codifica *B-LV*, la sua esaurienza non è rilevante, poiché la decodifica va fatta su sequenze costruite con elementi dello stesso dizionario. Viceversa l'esaurienza è strettamente necessaria per la famiglia di messaggi in una codifica *LV-B*, poiché dalla sorgente può uscire una sequenza qualunque. Per contro, come riferito nell'osservazione precedente, si può operare anche in assenza di univoca codificabilità.

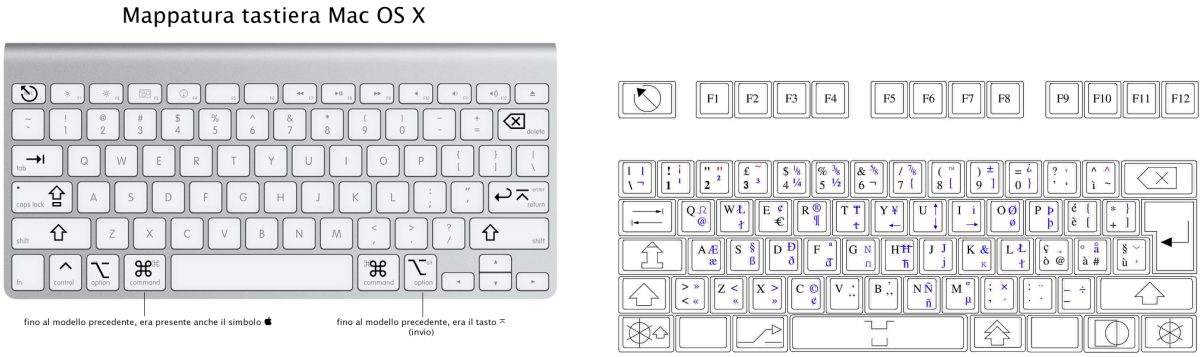
Nel caso dell'alfabeto di una tastiera, la prima necessità è quella di eseguire una semplice traduzione in binario, in modo che le lettere siano disponibili in forma binaria ai circuiti interni del calcolatore; in questo caso siamo chiaramente nell'ambito dii una codifica B-B per lettere singole. Si potrebbe dimostrare che la codifica B-B non consente di effettuare una compressione dati, e si tratta dunque di una semplice traduzione tra due alfabeti. Per comprimere dati bisogna ricorrere a una codifica a lunghezza variabile, sulla quale si basano le varie procedure usate per questo scopo (*zip, pkzip, compress, compact*, ...); non tratteremo però questa parte, perché esula dagli obiettivi del corso.
Supponiamo dunque che K sia il numero dei simboli della tastiera e che si effettui una codifica a *lunghezza costante*, in modo che ciascuna stringa abbia una lunghezza n. Il problema è allora quello di calcolare il valore di n. Quante sono le possibili n-ple, cioè le stringhe binarie di lunghezza n?
Per capirlo prendiamo $n = 1$. Se abbiamo una sola cifra binaria essa può essere 0 o 1, e dunque 2 possibilità. Se prendiamo due cifre binarie consecutive ci sono 4 possibilità, 00, 01, 10, 11; ciò accade perché per ciascuna delle 2 possibilità per la prima cifra ci sono 2 possibilità per la seconda, e quindi $2 \cdot 2 = 2^2 = 4$. Per $n = 3$ abbiamo, con un ragionamento analogo, $2 \cdot 2 \cdot 2 = 2^3 = 8$. Nel caso generico n ci sono allora $2 \cdot 2 \cdot \ldots \cdot 2$ moltiplicato per n volte, cioè 2^n. Se vogliamo che sia garantita l'ipotesi di univoca decodificabilità dobbiamo avere almeno una n-pla per ciascun simbolo della tastiera, e dunque deve valere la relazione

$$2^n \geqslant K$$

Applicando il logaritmo in base 2 e risolvendo rispetto a n, che deve essere un numero intero, si ottiene

$$n \geqslant \log_2 K \qquad \text{cioè} \qquad n = \lceil \log_2 K \rceil$$

dove il simbolo $\lceil x \rceil$ significa la *parte intera superiore* di x, cioè il più piccolo intero maggiore o uguale a x. Per esempio $\lceil 3,2 \rceil = 4$, $\lceil 5,99 \rceil = 6$, $\lceil 6 \rceil = 6$, $\lceil 3,001 \rceil = 4$. Analogamente per il simbolo $\lfloor x \rfloor$, che rappresenta la *parte intera inferiore* di x, cioè il più grande intero minore o uguale a x.
Se ora prendiamo la tastiera di figura 2.7a e contiamo quanti sono i simboli di base troviamo $26 \cdot 2$ lettere, $10 \cdot 2$ simboli sui tasti numerici, 22 segni di interpunzione e altri di vario genere e 17 altri tasti speciali, per un totale di 111 simboli. Poiché $\lceil \log_2 111 \rceil = 7$, con 7 bit si possono costruire $2^7 = 128 > 111$ possibili n-ple binarie



(a) Tipica tastiera di un computer portatile

(b) Possibili funzioni che possono essere associate ai tasti di una tastiera

Figura 2.7: Tastiere e funzioni di una tastiera

e nel 1961 un ingegnere della *IBM* propose una tabella di codifica, rappresentata in figura 2.8, che divenne successivamente lo standard ASCII (*American Standard Code for Information Interchange*). Sette bit non sono però sufficienti per le lingue più comuni, poiché ci sono anche le lettere accentate, le dieresi, le cediglie ecc. La figura 2.7b ci illustra per esempio tutte le funzioni che si possono ricavare dalla tastiera di un moderno computer portatile. Si può notare che per ogni tasto ci sono quasi sempre 3 o 4 funzioni diverse, e questo porta ben oltre 128 il numero delle possibili combinazioni. Tant'è che subito dopo l'introduzione dell'ASCII, venne proposta un'estensione di un bit, per portare il codice a $8\,\text{bit} = 1\,\text{byte}$. Ciò consente di raddoppiare i simboli, passando da 128 a 256 possibili combinazioni e di inserire anche le lettere con accenti speciali delle varie lingue europee. Si pervenne così a un

Bits b_4 b_3 b_2 b_1	Row ↓ / Column → 0	1	2	3	4	5	6	7
	b_7 b_6 b_5 = 0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	10	LF	SUB	*	:	J	Z	j	z
1 0 1 1	11	VT	ESC	+	;	K	[	k	{
1 1 0 0	12	FF	FC	,	<	L	\	l	|
1 1 0 1	13	CR	GS	-	=	M	]	m	}
1 1 1 0	14	SO	RS	.	>	N	^	n	~
1 1 1 1	15	SI	US	/	?	O	_	o	DEL

Figura 2.8: Tabella del codice ASCII. La 7-pla binaria che corrisponde a ciascun simbolo si legge come $b_7b_6b_5b_4b_3b_2b_1$. P.es. la codifica di 9 è 0111001, mentre quella di m è 1101101

insieme di standard, denominati ISO 8859-n, nei quali le prime 128 codifiche coincidono con quelle dell'ASCII standard, mentre l'estensione degli ulteriori 128 bit dipende dal valore di n (compreso tra 1 e 16). Per esempio ISO 8859-1 contiene le lettere speciali delle principali lingue europee nord-occidentali ($\tilde{n}$, ü, å, è, é, ß ecc.), mentre ISO 8859-5 contiene simboli del cirillico, ISO 8859-8 l'ebraico e così via (fig.2.9). Tuttavia questi standard non erano seguiti da tutti i costruttori, per cui poteva accadere che passando da una piattaforma *hardware* a un'altra ci fossero dei problemi con la decodifica. Attualmente la codifica è basata sullo standard perfezionato dall'*Unicode Consortium*, un consorzio di aziende interessate all'unificazione delle codifiche a livello internazionale, che possa comprendere tutte le lingue parlate nel mondo. Partito nella versione a 16 bit, che consente 65536 possibili combinazioni, viene ora proposto nelle versioni a 32 bit con sottocodifiche denominate *Unicode Transformation Format* del tipo UTF-8, UTF-16 e UTF-32. Le prime due codifiche sono a lunghezza variabile, nel senso che UTF-8 usa da 1 a 4 byte per rappresentare i vari simboli, mentre UTF-16 ne usa 2 o 4. Esse sono più efficienti, in termini di spazio occupato, della versione UTF-32, che necessita di 32 bit per tutti i simboli.
Con questo tipo di codifica, le cui tabelle di corrispondenza non sono ancora state completate, si riescono a rappresentare i caratteri di tutte le lingue vive e di molte lingue morte, ma anche i caratteri di scritti del patrimonio storico dell'umanità, nelle diverse lingue e negli svariati sistemi di segni utilizzati nel passato.
La codifica ASCII appena analizzata e le sue generalizzazioni più recenti, quali l'Unicode, consentono di codificare in binario qualunque simbolo venga immesso dalla tastiera o provenga dalla rete *Internet* attraverso il cavo di connessione, oppure attraverso un collegamento *wi-fi*.

2.3.2 La rappresentazione dei numeri

Quando scriviamo un indirizzo su una lettera, p.es. *via Roma 51*, usiamo le cifre 5 e 1 come semplici simboli grafici. Se vogliamo invece eseguire la somma $51 + 34 = 85$, l'uso dei simboli numerici 5 e 1 sottintende un impiego diverso delle cifre, legato alle regole dell'aritmetica dei *numeri naturali*, indicati in matematica col simbolo $\mathbb{N}$; essi sono $0, 1, 2, 3, 4, ...$, cioè infiniti. Anche se non possiamo sperare di rappresentare *tutti* i numeri naturali, vista la loro infinitezza, sarebbe comunque logico pensare a una codifica diversa per un numero naturale

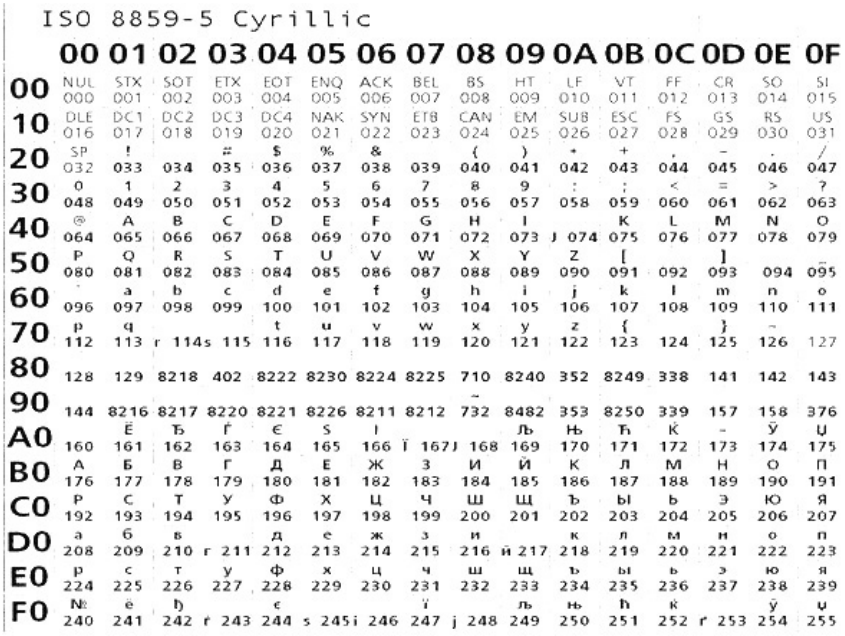


Figura 2.9: Estensione ISO 8859-5 del codice ASCII, che codifica le lettere dell'alfabeto cirillico

quando questo debba essere impiegato per i calcoli. Poiché all'interno di un computer si lavora in binario, riprendendo l'esempio della somma di prima l'ideale sarebbe che la codifica (binaria) del 51 *sommata* alla codifica del 34 fornisse la codifica del numero 85. Se vogliamo tentare questo approccio dobbiamo fare qualche passo indietro e pensare alla notazione che usiamo fin da piccoli per i numeri decimali, che ha il pregio di consentirci di fare agevolmente le quattro operazioni.

La rappresentazione dei numeri interi: la notazione posizionale

Quando scriviamo il numero 357 lo leggiamo come *trecentocinquantasette*; è uno schema mentale automatico, che ci sembra normale e che abbiamo acquisito nella scuola elementare. Si tratta della cosiddetta *notazione posizionale*, introdotta in Cina due secoli prima di Cristo e perfezionata in India nel 500. In Europa verrà usata solo a partire dal Rinascimento, e infatti la numerazione romana, di cui conserviamo ancora qualche traccia, è una notazione non-posizionale con la quale è estremamente complesso far di conto.
Notazione posizionale significa che il *valore* di ciascuna cifra $0..9$ dipende dalla *posizione* nella quale si trova all'interno del numero. Di conseguenza 357 significa 3 centinaia, 5 decine e 7 unità, che scritto in modalità matematica usando le potenze di 10 porta a

$$357 = \mathbf{3} \cdot 10^2 + \mathbf{5} \cdot 10^1 + \mathbf{7} \cdot 10^0$$

La notazione sottintende che stiamo usando la base 10, ma naturalmente si potrebbe usare anche una qualunque altra base b per rappresentare un numero intero come

$$\boldsymbol{a_n a_{n-1} ... a_0} = \boxed{\boldsymbol{a_n}} \cdot b^n + \boxed{\boldsymbol{a_{n-1}}} \cdot b^{n-1} + ... + \boxed{\boldsymbol{a_0}} \cdot b^0 \tag{2.1}$$

con la convenzione che ogni a_i deve essere compreso tra 0 e $b-1$ (tra 0 e 9 nella base $b = 10$). Il vantaggio della notazione posizionale è quello relativo alle regole elementari per effettuare le quattro operazioni, quali la somma e la moltiplicazione in colonna.

La conversione da base 2 a base 10 e viceversa Se vogliamo usare un alfabeto binario per rappresentare i numeri, possiamo pensare di usare una base $b = 2$ nella rappresentazione posizionale 2.1. In questo modo ciascuna cifra a_i sarà compresa tra 0 e 1, e dunque binaria. Con questa convenzione il numero binario 101001 corrisponde a

$$\mathbf{101001} = \mathbf{1} \cdot 2^5 + \mathbf{0} \cdot 2^4 + \mathbf{1} \cdot 2^3 + \mathbf{0} \cdot 2^2 + \mathbf{0} \cdot 2^1 + \mathbf{1} \cdot 2^0$$

Per capire di che numero decimale si tratta è sufficiente sviluppare i calcoli degli esponenti e sommare i termini, ottenendo $32 + 8 + 1 = 41$, che corrisponde a una *conversione* da base 2 a base 10.
Se vogliamo invece fare la conversione opposta, da base 10 a base 2, bisogna rielaborare la formula 2.1, espressa in base 2, che per comodità riportiamo sotto

$$\boldsymbol{a_n a_{n-1} \ldots a_0} = \boldsymbol{a_n} \cdot 2^n + \boldsymbol{a_{n-1}} \cdot 2^{n-1} + \ldots + \boldsymbol{a_2} \cdot 2^2 + \boldsymbol{a_1} \cdot 2^1 + \boldsymbol{a_0} \cdot 2^0 \tag{2.2}$$

In questo caso partiamo dal numero decimale, per esempio $N = 357$, e dobbiamo individuare gli elementi binari $\boldsymbol{a_n a_{n-1} \ldots a_0}$ in modo che espressi nella notazione 2.2 rappresentino 357 in decimale. In pratica i coefficienti $\boldsymbol{a_n a_{n-1} \ldots a_0}$ sono le nostre incognite. Per individuarle possiamo fare in questo modo; mettiamo in evidenza un 2 dal blocco della notazione

$$\begin{aligned} N &= (\boldsymbol{a_n} \cdot 2^{n-1} + \boldsymbol{a_{n-1}} \cdot 2^{n-2} + \ldots + \boldsymbol{a_2} \cdot 2^1 + \boldsymbol{a_1}) \cdot 2 + \boldsymbol{a_0} \\ &= \boldsymbol{q_1} \cdot 2 + \boxed{\boldsymbol{a_0}} \end{aligned}$$

Si nota che l'incognita $\boldsymbol{a_0}$ risulta essere il resto della divisione di N per 2. Se continuiamo in questo modo possiamo poi scrivere $\boldsymbol{a_1}$ come resto della divisione di $\boldsymbol{q_1}$ per 2, $\boldsymbol{a_2}$ come resto della divisione di $\boldsymbol{q_2}$ per 2 e così via:

$$\begin{aligned} \boldsymbol{q_1} &= (\boldsymbol{a_n} \cdot 2^{n-2} + \boldsymbol{a_{n-1}} \cdot 2^{n-3} + \ldots + \boldsymbol{a_2}) \cdot 2 + \boldsymbol{a_1} \\ &= \boldsymbol{q_2} \cdot 2 + \boxed{\boldsymbol{a_1}} \\ \\ \boldsymbol{q_2} &= (\boldsymbol{a_n} \cdot 2^{n-3} + \boldsymbol{a_{n-1}} \cdot 2^{n-4} + \ldots + \boldsymbol{a_3}) \cdot 2 + \boldsymbol{a_2} \\ &= \boldsymbol{q_3} \cdot 2 + \boxed{\boldsymbol{a_2}} \\ &\vdots \\ \boldsymbol{q_n} &= 0 \cdot 2 + \boxed{\boldsymbol{a_n}} \end{aligned}$$

Questo procedimento risolve il nostro problema della conversione da base 10 a base 2. Proviamo col 537

$$\begin{aligned} 537 &= 268 \cdot 2 + \boxed{1} && \Uparrow \\ 268 &= 134 \cdot 2 + \boxed{0} && \\ 134 &= 67 \cdot 2 + \boxed{0} && \\ 67 &= 33 \cdot 2 + \boxed{1} && \Uparrow \\ 33 &= 16 \cdot 2 + \boxed{1} && \\ 16 &= 8 \cdot 2 + \boxed{0} && \\ 8 &= 4 \cdot 2 + \boxed{0} && \text{lettura dal bit più significativo} \\ 4 &= 2 \cdot 2 + \boxed{0} && \text{a quello meno significativo} \\ 2 &= 1 \cdot 2 + \boxed{0} && \\ 1 &= 0 \cdot 2 + \boxed{1} && \Uparrow \end{aligned}$$

Poiché il primo bit che si ottiene dal procedimento è $\boldsymbol{a_0}$, cioè quello meno significativo, la lettura della stringa binaria deve essere fatta dal basso verso l'alto, il che porta al numero 1000011001.
Se ora vogliamo verificare che la conversione sia stata fatta correttamente, è sufficiente fare la riconversione da base 2 a base 10

$$\begin{aligned} \mathbf{1000011001} &= \mathbf{1} \cdot 2^9 + \mathbf{1} \cdot 2^4 + \mathbf{1} \cdot 2^3 + \mathbf{1} \\ &= 512 + 16 + 8 + 1 = 537 \end{aligned}$$

La notazione posizionale in base 2 con n bit ci consente di rappresentare tutti i 2^n numeri interi (positivi) compresi tra 0 e $2^n - 1$. Se disponiamo invece di una generica base b, l'intervallo di rappresentazione va da 0 a $b^n - 1$.
Se prendiamo per esempio $n = 4$ nel caso binario, otteniamo la rappresentazione dei numeri da 0 a $2^4 - 1 = 15$ evidenziata in figura 2.10a. Grazie all'impiego della notazione posizionale possiamo ora fare somme e moltiplica-

0111	7	1111	15
0110	6	1110	14
0101	5	1101	13
0100	4	1100	12
0011	3	1011	11
0010	2	1010	10
0001	1	1001	9
0000	0	1000	8

(a) Rappresentazione posizionale in base 2 dei numeri da 0 a $2^4 - 1$

Somma	Prodotto
$0 + 0 = 0$	$0 \cdot 0 = 0$
$0 + 1 = 1$	$0 \cdot 1 = 0$
$1 + 0 = 1$	$1 \cdot 0 = 0$
$1 + 1 = 0$	$1 \cdot 1 = 1$
con riporto	

(b) Somma e prodotto in base 2

Figura 2.10: Rappresentazione binaria e operazioni relative

zioni tra numeri binari usando le stesse regole usate per i numeri decimali. La figura 2.10b ci mostra le operazioni di base; si noti che $1+1 = 10$ e dunque, in una somma in colonna, bisogna effettuare il riporto di 1 esattamente come in decimale, quando $1 + 9 = 10$, scrivo 0 e riporto 1. Nella figura 2.11a sono invece riportate una somma e una moltiplicazione eseguite in colonna, secondo le regole tradizionali. Con l'espediente della notazione posizionale si realizza l'obiettivo iniziale di una codifica della somma (del prodotto) che corrisponde alla somma (prodotto) delle codifiche. Quando si deve gestire l'informazione che deriva da una codifica in binario, p.es. quella associata

$$\begin{array}{cccc} & {\scriptstyle 1} & & \\ 1010 & + & 10 & \\ \underline{0011} & = & \underline{3} & \\ 1101 & & 13 & \end{array} \qquad \begin{array}{cccc} & & & \\ 0100 & * & 4 & \\ \underline{0011} & = & \underline{3} & \\ 0100 & & & \\ \underline{0100} & & & \\ 01100 & & 12 & \end{array}$$

(a) Somme e prodotti in base 2 si eseguono secondo le regole consuete

$$\begin{array}{cccc} & {\scriptstyle 1} & & \\ & 1 & 0 & 1 & 0 & 10 & + \\ & 1 & 1 & 0 & 0 & \underline{12} & = \\ {\scriptstyle 1} & 0 & 1 & 1 & 0 & 6 & \end{array}$$

(b) Somma che porta a un errore di *overflow*

Figura 2.11: Esempi di operazioni in base 2

a un simbolo ASCII o alla codifica in notazione posizionale di un numero, bisogna assegnare a priori una certa risorsa di memoria all'oggetto che stiamo manipolando. Per esempio se usiamo un codice ASCII esteso ISO 8859 assegneremo sempre 8 bit, cioè 1 byte a ciascun simbolo, mentre se usiamo un UTF-32 assegneremo 32 bit, cioè 4 byte. Nel caso dei numeri, qualunque sia il valore n scelto per rappresentare tutti gli interi tra 0 e $2^n - 1$, si deve però fare i conti con la possibilità di fare delle operazioni il cui risultato esca dalla capacità di rappresentazione scelta. Se per esempio destiniamo 4 bit ai numeri interi che vanno da 0 a 15 e sommiamo 10 con 12, il risultato 22 è maggiore del massimo numero rappresentabile; ciò porta a un errore di *overflow*, ben visibile in figura 2.11b.

legato al fatto che l'1 di riporto non trova spazio per essere collocato, poiché sarebbe necessario disporre di una quinta cella di memoria, che invece non esiste. Si osservi che per risolvere il problema non è sufficiente aumentare n, poiché per qualunque valore abbia ci saranno sempre dei numeri interi la cui somma eccede la capacità di rappresentazione. Nei computer moderni in presenza di un errore di *overflow* avviene una commutazione automatica della notazione, che passa da quella intera a quella detta *a virgola mobile* (o *floating point*), che sarà analizzata in un prossimo paragrafo.

La rappresentazione mediante complemento a 2 La notazione posizionale in base 2 ci consente di rappresentare tutti i numeri naturali tra 0 e $2^n - 1$. Resta ancora da risolvere il problema della rappresentazione dei numeri interi negativi, degli interi che escono dall'intervallo $0 \dots 2^n - 1$ e di quelli non interi (positivi o negativi che siano).
Cominciamo con gli interi negativi, che assieme ai positivi e allo 0 costituiscono l'insieme $\mathbb{Z}$. Di primo acchito potremmo ingenuamente pensare di caratterizzare il segno + o − di un numero anteponendo semplicemente 1 o 0 al numero in questione. In tal modo si otterrebbe la codifica di figura 2.12a, che consentirebbe di codificare con 4 bit tutti gli interi da -7 a $+7$. In questo modo si avrebbe il problema di una doppia rappresentazione dello zero, che non è mai opportuna poiché costringe a un doppio controllo quando si debba fare una verifica del tipo $x = 0$, per una certa variabile x. Ma il motivo che impedisce l'uso di una tale codifica è che la codifica della somma (algebrica) tra due numeri *non* corrisponde alla somma delle codifiche. Come esempio possiamo vedere la somma tra -3 e $+4$ di figura 2.12b, che produce -7 invece di $+1$. Per risolvere il problema si ricorre allora alla

0111	+7	1111	−7
0110	+6	1110	−6
0101	+5	1101	−5
0100	+4	1100	−4
0011	+3	1011	−3
0010	+2	1010	−2
0001	+1	1001	−1
0000	+0	1000	−0

(a) Rappresentazione ingenua (sbagliata) dei numeri negativi

1	0	1	1	−3	+
0	1	0	0	+4	=
1	1	1	1	−7	

(b) Errore nella somma dovuto a una rappresentazione inadeguata dei numeri negativi

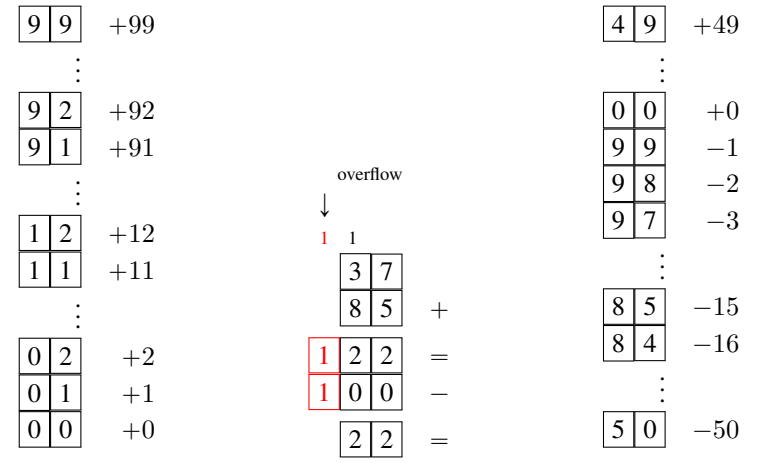
Figura 2.12: Rappresentazione errata dei numeri negativi

rappresentazione mediante *complemento a due*, che oggi è usata nella totalità dei processori, anche se in passato si era diffusa, in un primo momento, una rappresentazione simile chiamata *complemento a uno*.
Introdurremo la notazione *complemento a due* partendo da un esempio con la base decimale, con la quale abbiamo più confidenza (si tratterebbe in tal caso di un *complemento a dieci*).
Si supponga di avere a disposizione due celle decimali di memoria; ciascuna cella può contenere un numero qualunque tra 0 e 9, e dunque con esse possiamo rappresentare tutti gli interi tra 0 e $10^2 - 1 = 99$ (fig.2.13a). Supponiamo di voler fare la somma tra 37 e -15; servirebbe allora una rappresentazione coerente di -15, dove per "coerente" intendiamo che la somma algebrica delle codifiche porti alla codifica della somma. Se dobbiamo fare $37 - 15$ si può operare aggiungendo e togliendo 100 come segue

$$\begin{aligned} 37 - 15 &= 37 + (100 - 15) - 100 \\ &= 37 + 85 - 100 \\ &= 122 - 100 = 22 \end{aligned} \tag{2.3}$$

Il risultato non cambia, cioè 22, ma questo approccio ci fa capire che per ottenere il risultato corretto bisogna togliere 100 alla somma tra 37 e il complemento a 100 di 15, che è 85 (complemento a 100 significa in questo caso

"ciò che manca a 15 per arrivare a 100"). Se eseguiamo l'operazione sottoponendola ai vincoli di memoria imposti, possiamo osservare (fig.2.13b) che sommando 37 a 85 su una memoria con due sole celle si perde il riporto di 1 per *overflow*, e questa perdita corrisponde a una sottrazione automatica di 100 dal risultato finale, che porta a un risultato corretto. Poiché la somma di 37 e 85 è pari a 22 (in *overflow*), possiamo lecitamente *interpretare il numero* 85 *come una rappresentazione di* -15. In tal modo la rappresentazione di $-x$ diventa quella di $10^2 - x$, e quindi -1 è codificato dal 99, -2 dal 98 ecc. Il più grande numero positivo che si può codificare è $+49$, mentre il più grande numero negativo è -50 (fig.2.13c). La notazione basata sul complemento (a dieci, in questo caso) gode del



(a) Rappresentazione degli interi da 00 a 99

(b) L'errore di *overflow* consente di sottrarre 100 in modo automatico

(c) Rappresentazione mediante complemento a 10

Figura 2.13: Rappresentazione decimale tradizionale e a complemento a 10 con due celle di memoria

grosso vantaggio che le somme e le sottrazioni si eseguono sempre come somma (algebrica) delle codifiche, con un unico tipo di circuiteria. Si ha inoltre un'unica rappresentazione dello zero. La possibilità di rappresentare i numeri negativi porta a un bilanciamento attorno allo zero della capacità di rappresentazione, che passa da $0 \ldots 99$ a $-50 \ldots +49$.

Se ora passiamo al caso binario cambia la base, ma non il ragionamento. Supponiamo di avere 4 bit a disposizione per la codifica. Invece che rappresentare gli interi tra 0 e 15 vogliamo centrare attorno allo 0 l'intervallo di rappresentazione, in modo da disporre anche di una codifica per i numeri negativi. Con un ragionamento simile a quello fatto prima possiamo notare che, p.es., la somma algebrica tra 5 e -2 può essere scritta aggiungendo e togliendo 2^4 (in generale b^n con b base e n numero di celle di memoria)

$$\begin{aligned} 5 - 2 &= 5 + (16 - 2) - 16 \\ &= 5 + 14 - 16 \\ &= 19 - 16 = 3 \end{aligned} \tag{2.4}$$

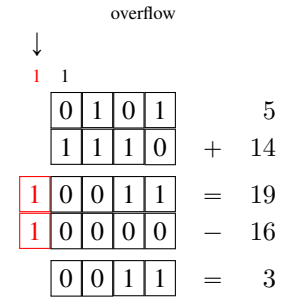
In questo modo la codifica di -2 diventa quella del complemento a $2^4 = 16$, cioè 14; quest'ultimo sommato con 5 porta a 19, al quale viene tolto 16 in modo automatico dall'overflow su 4 bit (fig.2.14b). La codifica degli interi tra 0 e 15 di figura 2.14a viene allora trasformata nella codifica tra -8 e $+7$ di figura 2.14c. Se disponiamo di n bit, l'intervallo di rappresentazione passa da $[0, 2^n - 1]$ per la codifica dei soli numeri positivi a

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

per la codifica dei numeri interi positivi e negativi, il che equivale a centrare l'intervallo di rappresentazione sullo 0 (si veda figura 2.15).

1	1	1	1	+15
1	1	1	0	+14
1	1	0	1	+13
1	1	0	0	+12
1	0	1	1	+11
1	0	1	0	+10
1	0	0	1	+9
1	0	0	0	+8
0	1	1	1	+7
0	1	1	0	+6
0	1	0	1	+5
0	1	0	0	+4
0	0	1	1	+3
0	0	1	0	+2
0	0	0	1	+1
0	0	0	0	+0

(a) Rappresentazione degli interi da 0 a 15



(b) L'errore di *overflow* consente di sottrarre 16 in modo automatico

0	1	1	1	+7
0	1	1	0	+6
0	1	0	1	+5
0	1	0	0	+4
0	0	1	1	+3
0	0	1	0	+2
0	0	0	1	+1
0	0	0	0	+0
1	1	1	1	−1
1	1	1	0	−2
1	1	0	1	−3
1	1	0	0	−4
1	0	1	1	−5
1	0	1	0	−6
1	0	0	1	−7
1	0	0	0	−8

(c) Rappresentazione mediante complemento a 2 con 4 bit

Figura 2.14: Rappresentazione binaria tradizionale e a complemento a 2 con 4 celle di memoria

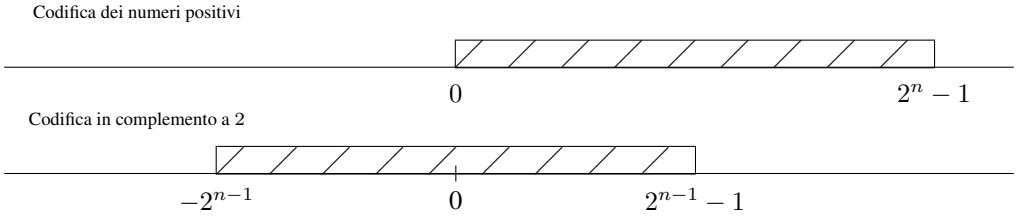


Figura 2.15: Intervallo di rappresentazione degli interi nella notazione mediante complemento a 2 usando n celle binarie

La regola che deriva è la seguente

$$\text{codifico } \pm x \qquad \text{come} \qquad 2^n \pm x \qquad \text{letto con } n \text{ bit in } \textit{overflow}$$

e quindi -3 ha la codifica di $2^4 - 3 = 13$, cioè 1101, mentre $+3$ ha la codifica di $2^4 + 3 = 19$ (10011), che letto con 4 bit in overflow corrisponde a 0011. In pratica la codifica di un numero negativo $-x$ si ottiene complementando la rappresentazione binaria di $+x$ e poi sommando 1. Ciò deriva dalla circostanza che per ogni x, se sommiamo la sua rappresentazione in binario con quella del proprio complemento $\overline{x}$, ottenuta scrivendo 0 al posto di 1 e viceversa, si ottiene

$$x + \overline{x} = 2^n - 1 \qquad \text{cioè} \qquad \overline{x} + 1 = 2^n - x \tag{2.5}$$

Vale la pena osservare la genialità della rappresentazione mediante complemento a 2, che sfrutta un difetto del sistema (l'*overflow*) per realizzare una efficace codifica dei numeri negativi.

Nei computer attuali il numero di bit che si usano per i numeri interi è 32 o 64; questa scelta consente di rappresentare i seguenti intervalli:

$$\begin{array}{llllll}
32 \text{ bit} & \Rightarrow & [0, 2^{32}-1] & \Rightarrow & [0, & 4.294.967.295] \quad \text{positivi} \\
32 \text{ bit} & \Rightarrow & [-2^{31}, +2^{31}-1] & \Rightarrow & [-2.147.483.648, & 2.147.483.647] \quad \text{complemento a 2} \\
64 \text{ bit} & \Rightarrow & [0, 2^{64}-1] & \Rightarrow & [0, & 1,84467E+19] \quad \text{positivi} \\
64 \text{ bit} & \Rightarrow & [-2^{63}, +2^{63}-1] & \Rightarrow & [-9,22337E+18, & +9,22337E+18] \quad \text{complemento a 2}
\end{array}$$

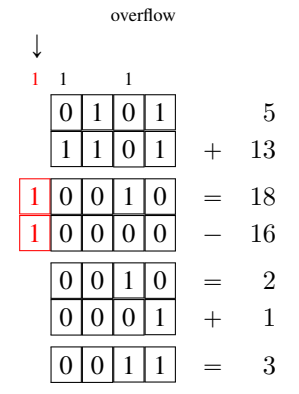
Si può notare che 32 bit non sono sufficienti in pratica, poiché si arriva a poco oltre i due miliardi nella notazione in complemento a 2, che è un numero intero che può capitare di dover trattare.

La rappresentazione mediante complemento a 1 Per completezza, facciamo ora un breve cenno alla notazione mediante complemento a 1; anche se è oramai divenuta obsoleta, tale notazione fu usata in macchine che hanno segnato la storia recente dell'informatica (CDC 6600, Univac 1100/2200). La notazione mediante complemento a 1 è una variante di quella a complemento a 2.
Riprendiamo le equazioni (2.4) e (2.5) e riscriviamole aggiungendo e togliendo b^n-1, con $b = 10$ e $b = 2$ rispettivamente. Si ottiene

$$\begin{array}{rcl}
37-15 &=& 37+(99-15)-99 \\
&=& 37+84-99 \\
&=& 121-99 \\
&=& 121-(100-1) \\
&=& 121-100+1 \\
&=& 21+1=22
\end{array}
\qquad
\begin{array}{rcl}
5-2 &=& 5+(15-2)-15 \\
&=& 5+13-15 \\
&=& 18-15 \\
&=& 18-(16-1) \\
&=& 18-16+1 \\
&=& 2+1=3
\end{array}
\tag{2.6}$$

1	1	1	1	+15
1	1	1	0	+14
1	1	0	1	+13
1	1	0	0	+12
1	0	1	1	+11
1	0	1	0	+10
1	0	0	1	+9
1	0	0	0	+8
0	1	1	1	+7
0	1	1	0	+6
0	1	0	1	+5
0	1	0	0	+4
0	0	1	1	+3
0	0	1	0	+2
0	0	0	1	+1
0	0	0	0	+0

(a) Rappresentazione degli interi da 0 a 15



(b) L'errore di *overflow* consente di sottrarre 16 in modo automatico; poi bisogna sommare 1 per bilanciare l'equazione

0	1	1	1	+7
0	1	1	0	+6
0	1	0	1	+5
0	1	0	0	+4
0	0	1	1	+3
0	0	1	0	+2
0	0	0	1	+1
0	0	0	0	+0
1	1	1	1	−0
1	1	1	0	−1
1	1	0	1	−2
1	1	0	0	−3
1	0	1	1	−4
1	0	1	0	−5
1	0	0	1	−6
1	0	0	0	−7

(c) Rappresentazione mediante complemento a 1 con 4 bit

Figura 2.16: Rappresentazione binaria tradizionale e a complemento a 1 con 4 celle di memoria

Si noti che ora la rappresentazione di $-x$ è quella di $(2^n–1)-x$; in binario ciò corrisponde a fare la negazione (il complemento a 1) di ogni cifra binaria di x; in questo modo la rappresentazione di -2 è il complemento a 1

di 0010, cioè 1101; la tabella di figura 2.16c riporta la codifica di tutti gli interi compresi tra -7 e $+7$. Nel fare la differenza riportata nell'equazione (2.6) ora dobbiamo sottrarre $2^4 - 1 = 15$; il valore 2^4 viene sottratto automaticamente dall'overflow, mentre il valore 1 deve essere sommato alla fine, per bilanciare l'equazione. Si osservi tuttavia che, se il valore della differenza è negativo, allora non è necessario sommare 1:

$$\begin{aligned} 2-5 &= 2+(15-5)-15 \\ &= 2+10-15 \\ &= 12-15 \\ &= -3 \end{aligned} \qquad \begin{array}{cccc} & & 1 & \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ \hline 1 & 1 & 0 & 0 \end{array} \quad \begin{array}{c} \\ 2 \\ + \quad 10 \\ = \quad 12 \end{array} \tag{2.7}$$

Ciò accade perché in tal caso la stringa binaria che si ottiene (1100), e che codifica il numero 12, rappresenta già la notazione in complemento a 1 del risultato (-3).
Questo tipo di codifica soffre di due difetti principali: il primo è la circostanza che la differenza va eseguita in modo diverso a seconda che il risultato sia positivo o negativo; il secondo, non meno grave, è dato dalla doppia rappresentazione dello 0, associato tanto alla stringa 0000 che alla 1111. Questo è il motivo principale del suo disuso.

La notazione a virgola mobile o *floating point*

Risolto il problema della rappresentazione dei numeri interi, bisogna ora affrontare quelli dei numeri non interi. Come noto essi sono i *numeri razionali* $\mathbb{Q}$, che si possono rappresentare come frazione a/b con a e b interi, e la loro estensione dei *numeri reali* $\mathbb{R}$. Questi ultimi contengono $\mathbb{Q}$ e $\mathbb{Z}$ come sottinsieme, ma contengono anche i numeri *irrazionali algebrici* (come la radice quadrata di 2, che non è esprimibile nella forma a/b) e i *numeri irrazionali trascendenti* (come π ed e, che non sono soluzione di alcuna equazione polinomiale a coefficienti razionali). Poiché i numeri reali hanno in genere un'espansione decimale infinita, per descrivere un singolo reale all'interno di un calcolatore sarebbe necessaria una quantità infinita di memoria. I reali non sono dunque gestibili a livello informatico, se non che mediante un *troncamento* (o un *arrotondamento*) delle cifre che ci consenta di trattarli in modo approssimato. Per esempio se dobbiamo lavorare col numero *Pi greco*

$$\pi = 3,14159265358979323846264338327950288419716939937510582097494 4\ldots$$

nel quale i tre punti esprimono il fatto che ci sono altre infinite cifre, per poterlo gestire è necessario effettuare un troncamento con un certo numero di cifre. Se scegliamo p.es. 10 cifre dopo la virgola, la rappresentazione approssimata di *Pi greco* diventa

$$\pi = 3,1415926535$$

che contenendo un numero finito di cifre decimali potrà essere gestita da una memoria di un computer. È ovvio che maggiore è il numero di cifre che usiamo e migliore sarà la precisione dei nostri calcoli. A questo punto il numero $3,1415926535$ può facilmente essere espresso nella forma a/b di un numero razionale, poiché possiamo scriverlo come

$$3,1415926535 = 31.415.926.535/10.000.000.000 = 31415926535 \cdot 10^{-10} \tag{2.8}$$

dove nell'ultima uguaglianza abbiamo usato la *notazione esponenziale*, detta anche *notazione scientifica* quando la base è 10. Se riusciamo a rappresentare in modo efficace i numeri razionali, potremo dunque lavorare anche con le approssimazioni dei reali. Come si può notare dalla rappresentazione (2.8), per identificare in modo univoco il numero $3,1415926535$ basta disporre dei due numeri interi della notazione esponenziale, che sono rispettivamente 31415926535 e -10, e concordare la base dell'esponente (in questo caso 10). Con questa convenzione si può scrivere un generico numero N come

$$N = \pm\, m \cdot 10^{\pm e}$$

Al numero m si dà il nome di *mantissa*, mentre e è chiamato *esponente*.
La notazione esponenziale è estremamente flessibile, poiché al prezzo di una piccola perdita di precisione, data dal

fatto che la mantissa ha un numero prefissato di cifre che potrebbe essere inferiore al numero di cifre significative che si vogliono rappresentare, si possono gestire numeri molto grandi o molto piccoli, positivi o negativi che siano. Per esempio

$$\begin{aligned} +12\,300 &= +1,23 \cdot 10^{+4} = +123 \cdot 10^{+2} \\ -4\,500\,000\,000 &= -4,5 \cdot 10^{+9} = -45 \cdot 10^{+8} \\ -0,000\,67 &= -6,7 \cdot 10^{-4} = -67 \cdot 10^{-5} \\ +0,000\,000\,089 &= +8,9 \cdot 10^{-8} = +89 \cdot 10^{-9} \end{aligned}$$

Poiché sappiamo già come rappresentare un intero, possiamo codificare un numero razionale usando m, e e la rispettiva coppia di segni. Dovremo evidentemente decidere quanti bit attribuire alla mantissa, quanti all'esponente e poi serviranno due bit per il segno. In questo modo la rappresentazione diventa

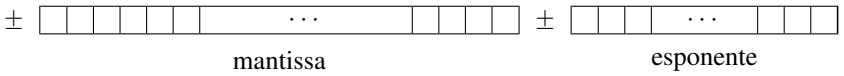


Figura 2.17: Notazione *floating point*

e viene chiamata notazione *a virgola mobile* o *floating point* (con la convenzione che la base sia 2).
Sempre ragionando in base 10, che ci è più familiare, e supponendo di attribuire p.es. 5 celle di memoria alla mantissa e 2 all'esponente, si voglia rappresentare il numero $23,89729$. Per far emergere 5 cifre per la mantissa bisogna spostare la virgola di 3 posizioni a destra e poi moltiplicare per 10^{-3}, ottenendo $23897,29 \cdot 10^{-3}$; ma nella rappresentazione le due cifre 29 dopo la virgola devono essere sacrificate, perdendo in precisione (fig.2.18). Poiché

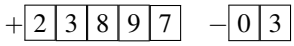


Figura 2.18: Rappresentazione in *floating point* del numero 23,89729 nella quale si perdono le ultime due cifre significative

il 29 è andato perso, è ovvio che al suo posto avrebbe potuto esserci qualunque altra coppia di interi compresi tra 00 e 99. Ecco allora che tutti i numeri che stanno a sinistra della figura 2.19 hanno la stessa rappresentazione di destra.

Con la stessa notazione basata su 5 celle per la mantissa e 2 per l'esponente, si possono rappresentare numeri

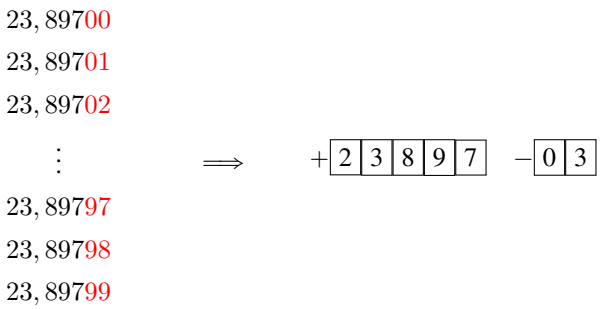


Figura 2.19: Tutti i numeri di sinistra hanno la stessa rappresentazione *floating point* di destra

con una dinamica enorme, da $+99999 \cdot 10^{+99}$ a $+00001 \cdot 10^{-99}$ per i numeri positivi, e da $-00001 \cdot 10^{-99}$ a $-99999 \cdot 10^{+99}$ per quelli negativi, a scapito della piccola perdita di precisione dopo la quinta cifra significativa. Si osservi inoltre che per l'aritmetica dei numeri basati sulla notazione a virgola mobile non valgono le leggi associative e distributive della somma e moltiplicazione; per esempio $x+(y+z) \neq (x+y)+z$, e dunque cambiando *l'ordine* con il quale vengono fatte le operazioni può cambiare il risultato. Sempre riprendendo l'esempio di prima

si ha

$$1 \cdot 10^{-4} + (23897 \cdot 10^{-3} - 23897 \cdot 10^{-3}) = 1 \cdot 10^{-4} \quad \text{ma} \quad (1 \cdot 10^{-4} + 23897 \cdot 10^{-3}) - 23897 \cdot 10^{-3} = 0$$

In generale possiamo dire che la notazione a virgola mobile offre una straordinaria fessibilità, legata alla possibilità di rappresentare numeri molto grandi e molto piccoli, a scapito di una piccola perdita di precisione.
Per tornare al caso binario possiamo dire che ci sono sostanzialmente due standard che hanno preso piede nell'architettura dei calcolatori denominati rispettivamente IEEE 754 a 32 e 64 bit. In entrambi la struttura di base è costituita dal bit del segno del numero, seguito dall'esponente espresso in una notazione *traslata* (che rappresenta tanto esponenti positivi che negativi), seguito infine dalla dalla mantissa. Cominciamo con la IEEE 754-32, denominata anche *single precision*, di cui vediamo la struttura in figura 2.20. Come si può notare c'è un segno,

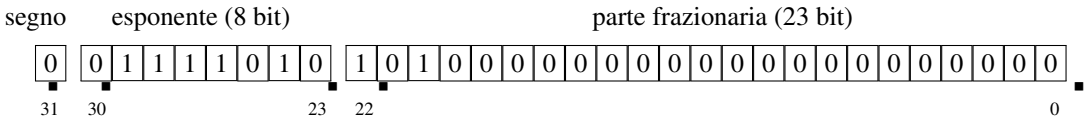


Figura 2.20: Rappresentazione *floating point* IEEE 754-32 o *single precision*

seguito da 8 bit per l'esponente e dalla mantissa che in questo caso rappresenta 23 bit della *parte frazionaria* del numero $1, b_{-1}b_{-2}\ldots b_{-23}$ (con i b_{-i} espressi in base 2). Il valore 1 prima della virgola viene sottinteso, e quindi non si riporta nella notazione risparmiando un bit. Tuttavia esso viene eliminato quando nell'esponente ci sono solamente zeri.
Per l'esponente si usa una notazione speciale, denominata *a eccesso* 127, nella quale il numero letto nella notazione posizionale in binario rappresenta il numero da codificare + 127, e dunque per dedurre il numero codificato dobbiamo sottrarre 127 al numero letto. La tabella di figura 2.21 ci fa vedere la conversione.

Valore binario	Interpretazione a eccesso 127	Lettura in notazione posizionale
11111111	+128	255
11111110	+127	254
11111101	+126	253
⋮		
10000001	+2	129
10000000	+1	128
01111111	0	127
01111110	−1	126
⋮		
00000010	−125	2
00000001	−126	1
00000000	−127	0

Figura 2.21: Rappresentazione mediante *eccesso a* 127

Tenuto conto di ciò il numero che deriva dalla notazione IEEE 754-32 *single precision* è nella forma

$$(-1)^{\text{segno}}\,(1, b_{-1}b_{-2}\ldots b_{-23}) \cdot 2^{e-127} = (-1)^{\text{segno}} \left(1 + \sum_{i=1}^{23} b_{23-i}\, 2^{-i}\right) \cdot 2^{e-127}$$

Vediamo allora, come esempio, il numero che corrisponde alla codifica di figura 2.20, cioè

0 01111010 10100000000000000000000

Il segno vale 0. L'esponente 01111010 espresso in notazione posizionale vale $2^6 + 2^5 + 2^4 + 2^3 + 2^2 = 122 = e$, che sottraendo 127 porta a -5. Per la parte frazionaria si ha invece $1 + 2^{-1} + 2^{-3} = 1,625$. Si ottiene allora

$$(-1)^{\text{segno}} \left(1 + \sum_{i=1}^{23} b_{23-i}\, 2^{-i}\right) \cdot 2^{e-127} = (-1)^0\,(1,625) \cdot 2^{-5} = 5,0781$$

L'altra notazione *floating point* di tipo standard, la IEEE 754 a 64 bit detta *double precision*, è del tutto simile a quella a 32 bit, con l'unica differenza che riserva 11 bit per l'esponente e 52 bit per la parte frazionaria (fig.2.22). Per la lettura dell'esponente si fa sempre riferimento alla notazione a eccesso, solo che in questo caso l'eccesso

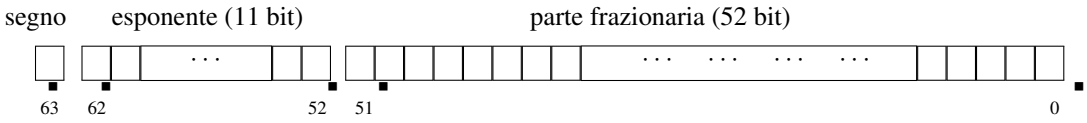


Figura 2.22: Rappresentazione *floating point* IEEE 754-64 o *double precision*

vale 1023 poiché l'esponente ha 11 bit invece che 8. Con gli stessi simboli usati per il caso precedente si ottiene allora

$$(-1)^{\text{segno}}\,(1, b_{-1}b_{-2}\ldots b_{-52}) \cdot 2^{e-1023} = (-1)^{\text{segno}} \left(1 + \sum_{i=1}^{52} b_{52-i}\, 2^{-i}\right) \cdot 2^{e-1023}$$

Sulla base di quanto visto finora possiamo usare i 64 bit (o i 32 bit) per la rappresentazione dei numeri o nella notazione a complemento a 2 per codificare i numeri interi positivi e negativi, oppure i numeri non interi usando la notazione a virgola mobile. Nel caso in cui, lavorando con gli interi, si dovesse superare la soglia di rappresentazione si avrà una commutazione automatica alla notazione *floating point*, perdendo eventualmente un po' di precisione.

La rappresentazione fisica del *bit*

Prima di passare alla descrizione delle modalità che ci consentono di codificare i segnali analogici e le immagini, facciamo una breve digressione sul problema della rappresentazione fisica del *bit*. Come già spiegato precedentemente tutte le informazioni del mondo esterno devono essere codificate in stringhe binarie, cioè stringhe su un alfabeto di due elementi. Ogni singola cifra binaria, che può essere 0 o 1, viene chiamata *bit*, parola che deriva dalla contrazione di *Binary digIT*.
Il *bit* è la più piccola quantità d'informazione possibile, cioè quella che ci consente di distinguere tra due due stati alternativi ed equiprobabili.
Quando per esempio lanciamo una moneta (non truccata), per la quale la probabilità a priori di ottenere testa o croce è pari a $1/2$, l'informazione che ne ricaviamo è di 1 *bit*.

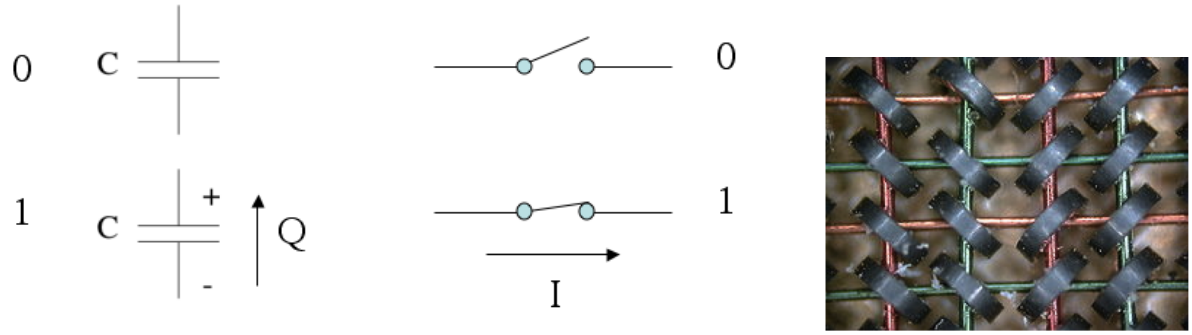
Per quanto riguarda i multipli del *bit* c'è da dire che essi non sono praticamente usati, poiché in informatica si tende a usare il *byte* come unità d'informazione. Ciò deriva dal fatto che il *byte* è solitamente la più piccola quantità di memoria *indirizzabile* all'interno di un computer, quella associata a una lettera nel codice ASCII. Purtroppo, però, questo non è l'unica specificità relativa ai multipli che si usano in informatica, poiché esiste una più seria ambiguità legata al valore dei prefissi *kilo, mega, tera*, ecc., introdotti nel *Sistema Internazionale* come multipli in base 10 e usati in Informatica intendendo multipli in base 2. Di conseguenza quando si dice $kilo$ si pensa a un fattore 10^3 nel *Sistema Internazionale*, ma a un fattore $2^{10} = 1024$ in ambito informatico. La necessità di usare la base 2 deriva dal fatto che con n bit si indirizzano 2^n celle di memoria, e quindi le quantità di memoria è sempre espressa come potenza di 2. Poiché $1024 \approx 1000$, si usa impropriamente il multiplo *kilo* per indicare 2^{10}, mentre i prefissi *Mega* ($2^{20} = 1.048.576$) e *Giga* ($2^{30} = 1.073.741.824$) vengono usati come approssimazione di milione e miliardo. Gli organismi internazionali di standardizzazione hanno tentato di imporre una denominazione non ambigua (si veda la colonna di sinistra della tabella di figura 2.23), basata sull'introduzione della parola *binary* nella struttura del nome (*kilo binary byte* = *kibibyte* e così via), ma essa non è però entrata nell'uso comune. Si

noti che la differenza percentuale tra valore effettivo e multiplo in base 10 del Sistema Internazionale porta a degli errori che sono percentualmente crescenti con l'ordine di grandezza (si veda l'ultima colonna della tabella 2.23). Si è detto che l'esistenza del teorema di campionamento ha decretato il trapasso dalla tecnologia analogica a quella

Prefissi binari				Prefissi SI			
Fattore	Simbolo	Nome		Simbolo	Nome	Fattore	Errore %
2^{10}	KiB	*kibibyte*	$\approx$	kB	*kilobyte*	10^3	$+2,4$
2^{20}	MiB	*mebibyte*	$\approx$	MB	*megabyte*	10^6	$+4,9$
2^{30}	GiB	*gibibyte*	$\approx$	GB	*gigabyte*	10^9	$+7,4$
2^{40}	TiB	*tebibyte*	$\approx$	TB	*terabyte*	10^{12}	$+10,0$
2^{50}	PiB	*pebibyte*	$\approx$	PB	*petabyte*	10^{15}	$+12,6$
2^{60}	EiB	*exbibyte*	$\approx$	EB	*exabyte*	10^{18}	$+15,3$
2^{70}	ZiB	*zibibyte*	$\approx$	ZB	*zettabyte*	10^{21}	$+18,1$
2^{80}	YiB	*yobibyte*	$\approx$	YB	*yottabyte*	10^{24}	$+20,9$

Figura 2.23: Multipli del byte usando prefissi binari (praticamente inutilizzati) e prefissi del *Sistema Internazionale*, che esprimono però un valore approssimato

digitale, e che l'esistenza della logica Booleana, assieme ad alcune altre questioni di carattere tecnico, ha fatto prevalere i circuiti binari rispetto a qualunque altra tipologia discreta (ternario, esadecimale, ecc; cfr. sez. 2.2). Ora resta da capire come prendono corpo i *bit* all'interno dei circuiti elettronici che costituiscono il processore e le memorie dei *computer*.
In generale un *bit* può esser associato ai livelli di una grandezza fisica di riferimento, che per i dispositivi moderni può essere tipicamente una tensione, una corrente, una carica elettrica o la presenza di una polarizzazione magnetica. In figura 2.24 possiamo vedere alcuni esempi; nel condensatore di figura 2.24a il valore logico 1 è associato alla presenza di una carica elettrica Q tra le armature dello stesso condensatore, il valore logico 0 alla sua assenza, mentre nell'interruttore di fig. 3.1 i due stati logici 0 e 1 sono rappresentati dall'interruttore aperto (non c'è flusso di corrente) e dall'interruttore chiuso (scorre una corrente I). Un discorso simile vale per la polarizzazione che si ha quando un nucleo toroidale di *ferrite* viene polarizzato N-S (Nord-Sud) o S-N, a seconda del verso della corrente che scorre nel filo che attraversa il nucleo. La figura 2.24c) mostra in particolare la tecnica usata nei calcolatori degli anni '50 -'60 per assemblare i nuclei di ferrite in modo da costituire blocchi estesi di memoria; nella memoria della figura sono presenti 16 nuclei associati a 16 bit.



(a) Condensatore nei due stati logici possibili, carico e scarico

(b) Interruttore nei due stati logici possibili, acceso e spento

(c) Nuclei di ferrite di una memoria in uso negli anni '50-'60

Figura 2.24: Rappresentazione fisica del *bit* usando diverse grandezze elettriche e magnetiche

Attualmente si tende a usare un transistor come supporto fisico del *bit*, soprattutto nella sua variante MOS-FET (*Metal Oxide Semiconductor- Field Effect Transistor* o transistor *a effetto di campo*). Se osserviamo la figura 2.6,

notiamo che si può prendere in considerazione la tensione v tra collettore ed emettitore di un transistor e associare il valore logico 1 a un valore elevato di v (*logica positiva*) e un valore logico 0 a un valore basso della stessa tensione. Nei circuiti integrati delle famiglie logiche TTL (*Transistor-Transistor-Logic*) la tensione v di uscita associata allo stato 1 deve essere maggiore di $2,6$ V, mentre la tensione dello stato 0 deve essere minore di $0,4$ V (fig. 2.25). Un secondo metodo per realizzare fisicamente un *bit*, usato soprattutto per le memorie e che deriva dal

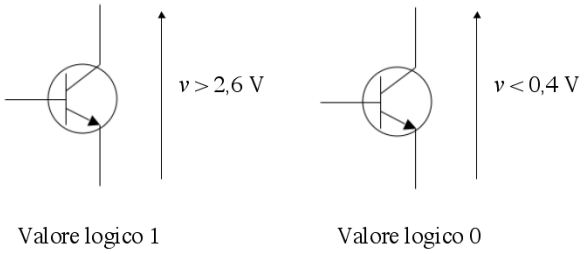
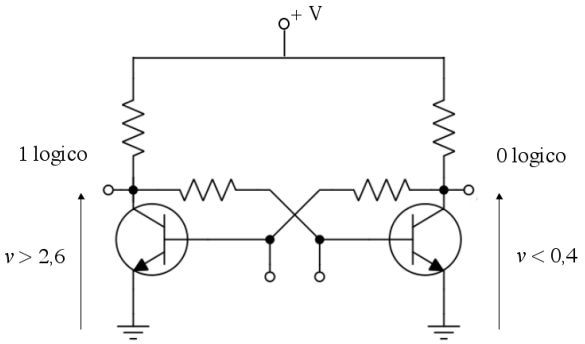
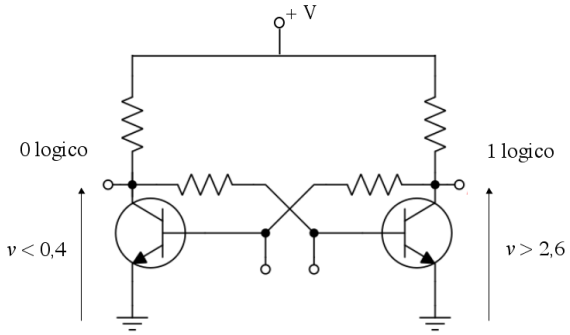


Figura 2.25: Valore logico 1 e 0 per la famiglia di circuiti integrati TTL

circuito di fig. 2.25, è quello di connettere l'uscita di un transistor in interdizione con l'ingresso di un transistor in conduzione, realizzando il cosiddetto *Flip-Flop* (si veda figura 2.26) Il circuito che si ottiene è *bistabile*, nel senso



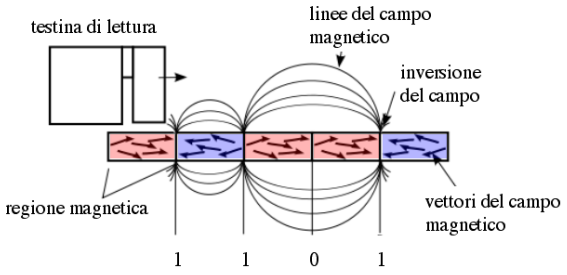
(a) Il transistor di sinistra del *Flip-Flop* è interdetto; ciò porta in conduzione il transistor di destra, che fornisce uscita logica 0



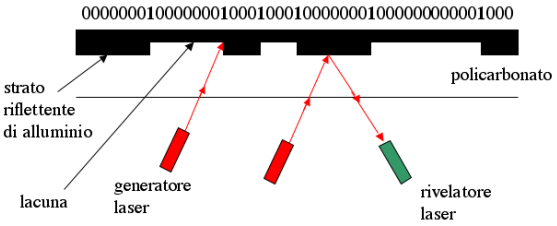
(b) Il transistor di sinistra del *Flip-Flop* è in conduzione; ciò porta in interdizione il transistor di destra, che fornisce uscita logica 1

Figura 2.26: Circuito bistabile del *Flip-Flop*

che esso può stare indifferentemente e stabilmente in uno o nell'altro dei due stati rappresentati in figura 2.26a e 2.26b. 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 ($v > 2,6$ V); ciò polarizza la base del transistor di destra, che entra in piena conduzione, facendo collassare a un valore basso ($v < 0,4$ V) la sua tensione di uscita (fig. 2.26a), che corrisponde a uno 0 logico. I ruoli dei due transistor si scambiano quanto la tensione di collettore del transistor di destra viene portata (con un impulso) a un valore ($v > 2,6$ V) (fig. 2.26b).
I circuiti a *Flip-Flop* possono essere realizzati anche con i MOS-FET prima citati, che danno migliori caratteristiche per quanto riguarda le potenze dissipate per il funzionamento, e gli stessi MOS-FET vengono impiegati, in una versione speciale, anche come elemento per le memorie *flash*, che sono memorie elettroniche di tipo permanente.
Altri sostrati fisici usati per memorizzare i *bit* sono quelli basati sul sottile strato di magnetizzazione che ricopre la superficie dei dischi rigidi (*Hard Disk Drive*, vedi figura 2.27a), che viene suddiviso in piccole aree magnetizzate in senso S-N (area rosa) o N-S (area azzurra). La transizione tra due aree diverse viene associata a un 1 mentre il mantenimento della stessa area corrisponde a uno 0.
Un altro tipo di tecnologia usata per memorizzare i *bit* è quella laser dei dischi ottici (fig.2.27b). Su un disco di policarbonato viene steso uno strato sottile di alluminio, che se lasciato integro è in grado di riflettere con alta efficienza un raggio laser; la superficie di alluminio viene invece forata in corrispondenza delle zone dove la luce laser



(a) Bit associati al sostrato ferromagnetico di un disco rigido



(b) Bit associati alla variazione di riflettenza di una superficie colpita da un raggio laser

Figura 2.27: Alcuni metodi per memorizzare bit fisici su *Hard Disk* e *Compact Disk*

non dev'essere riflessa. Anche in questa tecnologia si ha un 1 in corrispondenza di una variazione, che riguarda in questo caso la riflettenza della superficie.
Le tecniche di rappresentazione fisica dei bit che abbiamo analizzato finora riguardano la *memorizzazione* degli stessi su appositi supporti, che coincidono quasi sempre con dei dispositivi a semiconduttore. Quando si deve trasferire un pacchetto di bit all'interno della scheda madre del computer, oppure tra *computer* connessi in rete, si pone il problema di come organizzare tale trasmissione. Esistono a tal riguardo due modalità diverse, chiamate rispettivamente *trasmissione seriale* e *trasmissione parallela*, illustrate in figura 2.28 con riferimento alla codifica del numero 7. Nella trasmissione seriale i singoli bit della rappresentazione posizionale del numero vengono

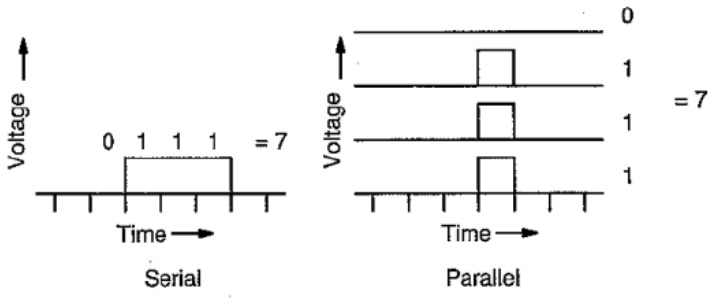
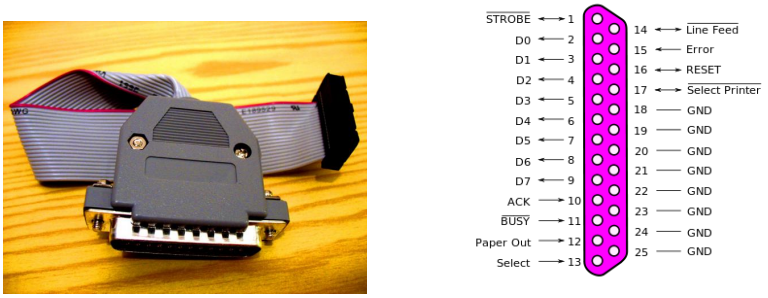


Figura 2.28: Trasmissione seriale e trasmissione parallela dei bit che codificano il numero 7

dislocati lungo l'asse temporale, nel senso che sono spediti uno dopo l'altro su una singola linea. Al contrario la trasmissione parallela prevede un numero di connessioni pari al numero di bit da trasmettere; i bit vengono a questo punto trasmessi contemporaneamente. È ovvio che, a parità di frequenza di sincronizzazione del sistema (chiamata in gergo *frequenza di clock*), il sistema parallelo a n bit offre una velocità di trasferimento n volte superiore al sistema seriale. In cambio esso è più costoso, poiché il cavo di interconnessione deve contenere $n + 1$ conduttori al posto dei 2 sufficienti per il caso seriale. I dispositivi fisici di connessione (parallela) vengono solitamente chiamati *bus*, e sono costituiti da un cavo con molti conduttori in parallelo. Nella figura 2.29a si può osservare un cavo parallelo con il relativo connettore (maschio), mentre nella figura e 2.29b viene rappresentata la struttura dei collegamenti standard per una *porta parallela*, cioè un connettore (femmina) che di solito si trova connesso con lo *chassis* del computer.

2.3.3 La codifica dei segnali analogici

Nella sezione 2.2 abbiamo sottolineato il fatto che l'informazione può nascere tanto in forma discreta (p.es. quella associata all'alfabeto di una tastiera) quanto in forma analogica (p.es. il suono captato da un microfono



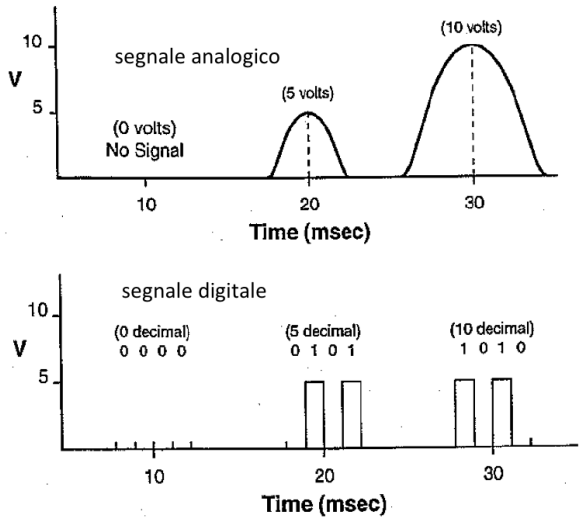
(a) Connettore (maschio) e relativo cavo parallelo

(b) Connessione circuitale di una porta parallela (femmina) associata a un *bus* di 8 bit (piedini 2-7)

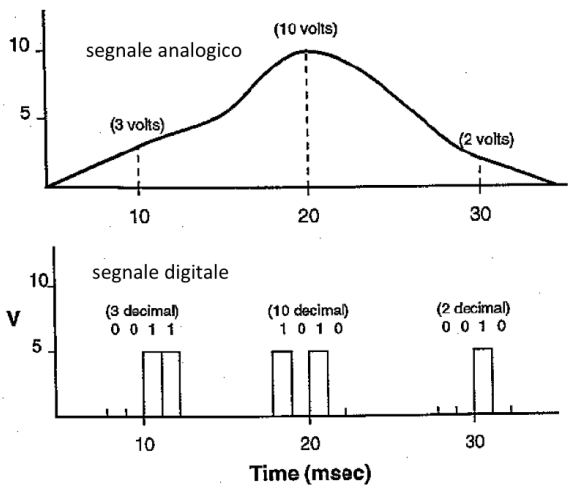
Figura 2.29: Connessione parallela

in una sessione di *Skype*). Se l'informazione è già in forma discreta abbiamo l'unico problema di codificarla in binario, p.es. usando una tabella simile a quella del codice ASCII. Se invece essa assume la forma di un segnale analogico, bisogna porsi il problema di capire se la codifica in binario sia un'operazione effettivamente realizzabile. L'informazione associata a un segnale analogico è infatti potenzialmente infinita, poiché è necessario specificare, istante per istante su una scala continua dei tempi, il valore della grandezza analizzata. Poiché tale valore è un numero reale, anche per identificare un solo punto della curva sarebbe in linea di principio necessario avere a disposizione una quantità infinita di memoria. Ne consegue che dovremo ricorrere a una rappresentazione approssimata del valore basata sulla notazione *floating point*.

Nella gestione dei segnali analogici ci sono essenzialmente due situazioni ricorrenti. Nella prima un segnale di tipo impulsivo viene usato per rappresentare un singolo valore numerico, che potrebbe derivare per esempio dalla lettura di una grandezza fisica; è questo il caso di figura 2.30a, nella quale l'impulso in uscita da un tubo fotomoltiplicatore (rilevatore di luce ad altissima sensibilità) viene valutato nella sua ampiezza in corrispondenza del picco di tensione. In questo caso il segnale analogico funge solo da "supporto" all'informazione relativa all'ampiezza. Nel secondo esempio di figura 2.30b si vede invece un segnale analogico in uscita da una video camera



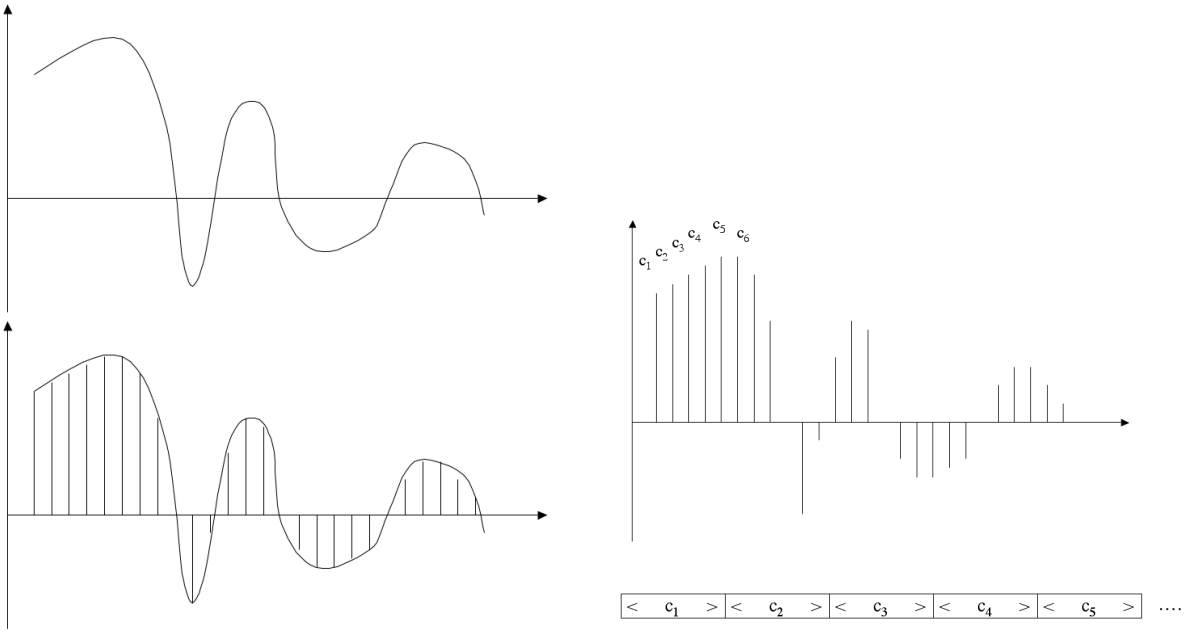
(a) Segnale analogico in uscita da un tubo fotomoltiplicatore connesso con uno scintillatore



(b) Segnale analogico in uscita da una video camera in un sistema a fluoroscopia

Figura 2.30: Esempi di segnali analogici generati da apparecchiature elettromedicali con le corrispondenti codifiche binarie dei valori misurati a intervalli di tempo costanti

in un sistema a fluoroscopia, che varia in modo continuo nel tempo, seguendo le variazioni di una grandezza fisica di riferimento. In entrambi i casi siamo interessati allo sviluppo temporale dell'ampiezza del segnale, che viene valutato a intervalli di tempo regolare (10 ms nell'esempio), solo che nel secondo caso il segnale varia con continuità tra i successivi valori oggetto della misura. Nella parte bassa della figura 2.31 si illustra la rappresentazione delle stesse grandezze numeriche (i valori $0,5,10$ e $3,5,2$ V) espressa però in binario mediante una notazione posizionale.
I segnali analogici con i quali si ha a che fare sono solitamente del secondo tipo, poiché rappresentano delle variazioni continue nel tempo di una grandezza fisica che vogliamo riprodurre ed eventualmente elaborare su un supporto digitale. L'idea di base per trasformare un segnale analogico in una sequenza di *bit* è allora quella di ricorrere all'impostazione di figura 2.30b, nella quale ad intervalli di tempo regolari si effettua una *lettura* del valore assunto dalla grandezza, prelevando un *campione* della stessa (fig. 2.31a). La lettura del valore del campione viene



(a) Campionamento di un segnale analogico

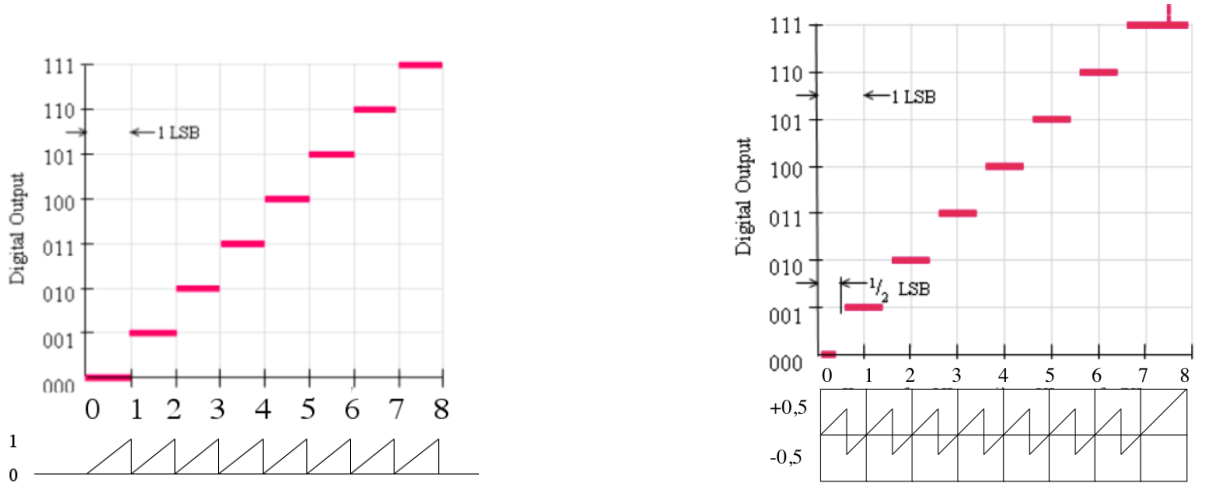
(b) Sequenza dei campioni rappresentati in notazione *floating point*

Figura 2.31: Campionamento di un segnale analogico

effettuata con uno strumento di misura che sarà caratterizzato da una certa accuratezza, che dipenderà dalla *classe* dello strumento. Se con esso si realizzano delle letture con n cifre (binarie) significative, allora ogni campione potrà essere descritto da un numero in notazione *floating point* con una mantissa di n bit. Ecco allora che il segnale analogico di figura 2.31a viene rappresentato dai campioni di figura 2.31b, che costituiscono una successione binaria in *floating point*. A questo punto è realizzata la *conversione analogico-digitale* (A/D) del segnale.
Il punto critico del ragionamento è che l'operazione di campionamento non consente, in generale, la ricostruzione del segnale analogico originario, poiché ovviamente va persa tutta l'informazione sull'andamento del segnale tra i campioni. Tuttavia, sotto opportune condizioni, tale riconversione diventa possibile. Ciascun segnale analogico è infatti caratterizzato dalla propria *banda*, che rappresenta la massima estensione dello *spettro* di frequenze presenti nel segnale. Se facciamo per esempio riferimento alla voce umana sappiamo che essa si estende tra (circa) 65 e 1500 Hz. Se la *banda* del segnale è limitata, allora si può dimostrare che esiste la possibilità di ricostruire in modo esatto il segnale di partenza a patto di effettuare un campionamento con una frequenza f_c sufficientemente elevata, pari *al doppio* della banda base (*frequenza di Nyquist*). Ciò costituisce il contenuto del celebre *teorema del campionamento di Shannon-Nyquist*, il più importante teorema della teoria dei segnali; esso specifica che, detta B la banda del segnale, la condizione per poter effettuare un campionamento senza perdita d'informazione è che valga la seguente disuguaglianza

$$f_c \geqslant 2 \cdot B$$

Per esempio, nel caso del campionamento di un segnale ad alta fedeltà caratterizzato da una banda che si spinge fino a 22 kHz, la frequenza di campionamento è di $44,1$ kHz; ciò rappresenta uno standard per i CD audio musicali. Il numero di *bit* necessari per effettuare la codifica è legato alla precisione nella lettura dei campioni. Se usiamo a scopo illustrativo i tre *bit* della figura 2.32a, che codificano tutti gli interi tra 0 e 7, possiamo notare che i valori



(a) Conversione analogico digitale e relativo errore di quantizzazione

(b) Conversione analogico digitale a *mezzo bit* e relativo errore di quantizzazione

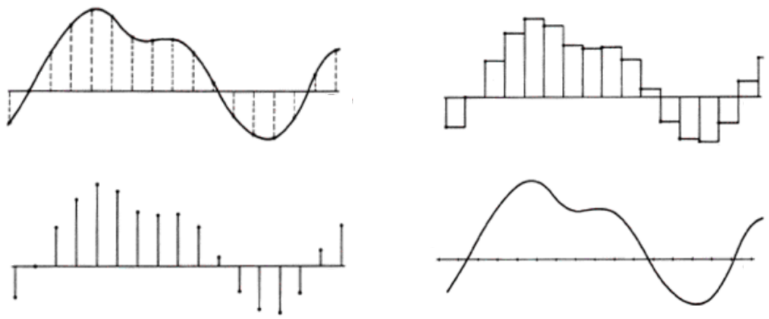
Figura 2.32: Conversione analogico digitale a *bit* pieno e a mezzo *bit*

della lettura compresi tra x e $x+1$ vengono codificati con l'intero $\lfloor x \rfloor$. Ciò determina un *errore di quantizzazione*, dato dalla differenza tra $x - \lfloor x \rfloor$, che oscilla tra 0 e 1; il relativo diagramma a dente di sega è rappresentato sotto la figura. Per limitare l'errore di quantizzazione si può ricorrere all'espediente di bilanciare la codifica, centrandola intorno al valore x (codifica a $1/2$ *bit*). In questo modo tutti i valori compresi tra $x - 1/2$ e $x + 1/2$ saranno codificati come x, e ciò limita l'errore di quantizzazione a $\pm 1/2$ (si veda la figura 2.32b).
Poiché l'errore di quantizzazione modifica di fatto il valore del segnale codificato, la successiva riconversione porta alla sovrapposizione tra il segnale iniziale e un *rumore di quantizzazione*. Per diminuirlo il più possibile bisogna aumentare il numero di livelli della discretizzazione, incrementando il numero di bit associati a ogni campione. La tabella di figura 2.33 mostra il valore percentuale dell'errore in funzione del numero di *bit* usati per

Numero di *bit*	Numero di valori	Valore massimo dell'errore di quantizzazione %
1	2	25
2	4	12,5
4	16	3,125
8	256	0,195
12	4096	$12,20 \cdot 10^{-3}$
16	65536	$7,63 \cdot 10^{-4}$

Figura 2.33: Valore percentuale dell'errore di quantizzazione in funzione del numero di bit e dei livelli usati

la conversione. Dal punto di vista operativo i campioni non rimangono segnali impulsivi isolati (fig.2.34a), ma il loro valore viene mantenuto fino al campione successivo, così come appare in figura 2.34b. L'applicazione di un successivo *filtro passa basso*, che tagliando le alte frequenze tende a smussare le variazioni brusche, restituisce il segnale di partenza, completando la cosiddetta ri-*conversione digitale-analogica* (D/A) del segnale. Quando si ha a disposizione la versione codificata in binario del segnale analogico, è possibile fare su di essa tutte le elaborazioni



(a) Campioni ideali ricavati dal segnale analogico

(b) Rappresentazione del segnale effettivo e successiva riconversione D/A mediante impiego di un filtro passa basso

Figura 2.34: Fasi della conversione A/D e della successiva riconversione D/A

che fossero eventualmente necessarie. La figura 2.35 ci mostra il processo completo di conversione e riconversione di un segnale analogico.

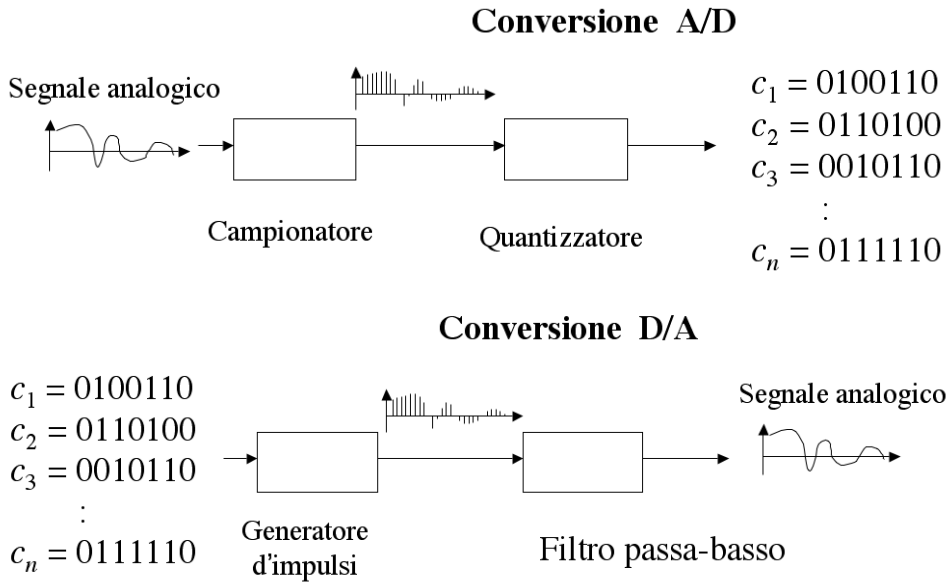


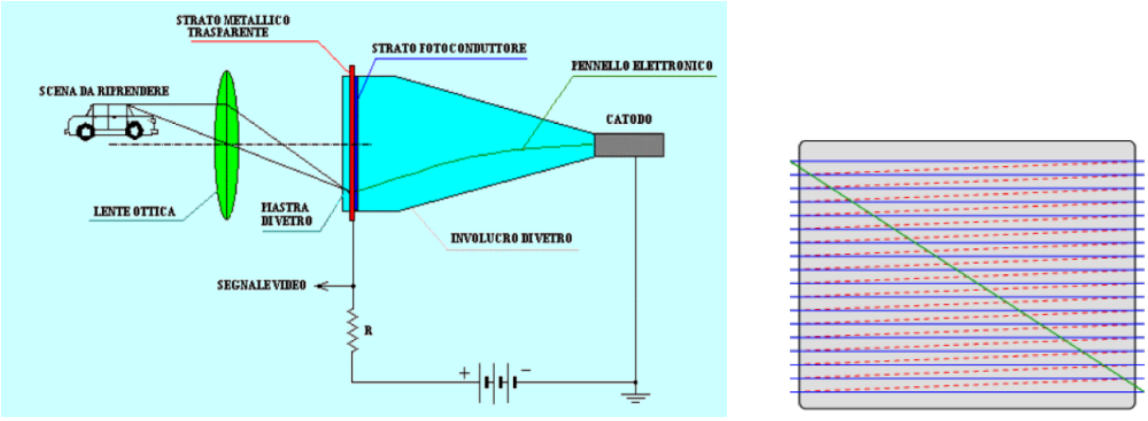
Figura 2.35: Conversione e riconversione di un segnale analogico

2.3.4 La codifica delle immagini

Le immagini sono l'ultima tipologia di informazione che dobbiamo trattare. Prima di addentrarci nel problema della loro trasformazione in *bit*, vale forse la pena raccontare l'evoluzione tecnologica nella loro rappresentazione. Come per altre classi di segnali (telefonici, video, radio,...) si è assistito a un trapasso tra tecnologie analogiche e tecnologie digitali, che in questo caso ha però cambiato anche la struttura fisica e la logica di funzionamento degli schermi usati per la rappresentazione delle immagini.

Elaborazione analogica delle immagini

Un'immagine, così come percepita dal nostri organi di senso, è un "segnale" tipicamente analogico; anche lasciando perdere il problema della rappresentazione tridimensionale degli oggetti, che il nostro cervello riesce a organizzare sfruttando la visione binoculare, e limitandoci a una visione piatta dell'immagine, si ha a che fare con un segnale bidimensionale, che può variare in intensità e cromatismo in modo continuo e lungo tutte le direzioni. Questo aspetto venne sfruttato fin dall'inizio nelle prime telecamere, che usavano un *tubo a raggi catodici* (o CRT, acrostico di *Cathode Ray Tube*) per fare una scansione continua dell'immagine da riprodurre. In questa telecamera un fascio o *pennello elettronico* generato da un catodo (chiamato anche *cannone elettronico*) viene attirato da uno schermo ad alto potenziale (anodo). Sulla parte interna dello schermo, realizzato in vetro, è depositato uno strato sottile di elementi e composti chimici fotoconduttivi (p.es. il *selenio*). L'immagine che si staglia sullo schermo, modulata in intensità luminosa, induce una modulazione analoga anche sullo strato fotoconduttivo, e il pennello elettronico che colpisce lo strato genera una corrente che dipende dallo stato di conduzione (in pratica dalla resistenza) del recettore fotoconduttivo (fig.2.36a). Se al pennello viene impresso una scansione sistematica lungo

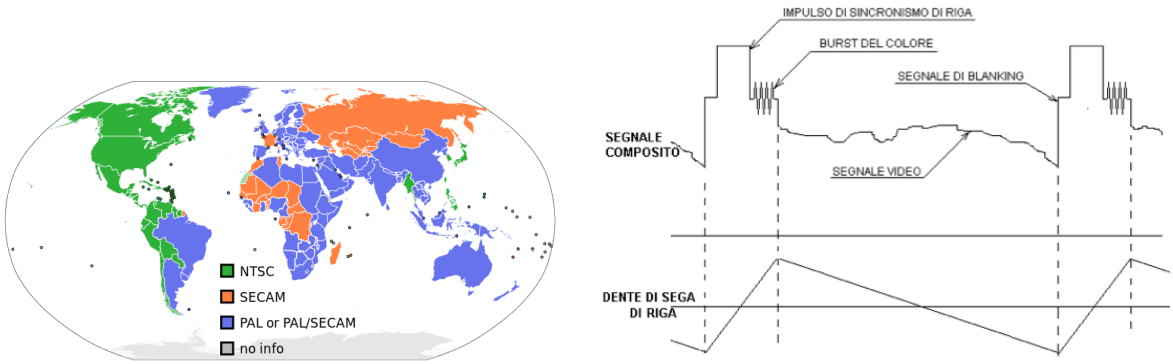


(a) Principio di funzionamento di una telecamera analogica

(b) Scansione di un quadro completo da parte del pennello elettronico

Figura 2.36: Acquisizione di un'immagine con una telecamera analogica

n linee orizzontali appaiate (righe blu della figura 2.36b), si compie una scansione completa dell'immagine (o *quadro*). Quando il pennello elettronico ha esaurito una riga e deve essere riposizionato all'inizio della successiva, allora è necessario spegnerlo (righe rosse tratteggiate). Alla fine del primo quadro il pennello si trova in basso a destra dello schermo, ed è necessario riportarlo nella posizione in alto a sinistra. Nel fare questo bisogna oscurare nuovamente il pennello (riga verde).
Per poter avere un'immagine stabile si deve provvedere a costruire molti quadri al secondo, tipicamente 50 o 60, a seconda dello standard. Il numero di linee da usare per ogni quadro, il numero di quadri per ogni secondo e le modalità di codifica dei colori sono oggetto di tre standard specifici, adottati inizialmente per le trasmissioni televisive; essi sono: il sistema *PAL*, sviluppato in Germania dalla Telefunken e usato principalmente in Europa (a parte la Francia) e in gran parte del mondo; il sistema *SECAM*, sviluppato in Francia e usato anche nella ex Unione Sovietica e in alcune nazioni dell'Africa; il sistema *NTSC*, sviluppato negli USA e impiegato anche in Giappone (fig.2.37a). Mentre i due sistemi europei *PAL* e *SECAM* adottano 625 righe e una frequenza di quadro di 50Hz, lo standard americano *NTSC* usa 525 righe e 60 Hz.
Il segnale di ogni riga, modulato in intensità luninosa, è dunque un segnale analogico simile a quello rappresentato in figura 2.37b, nel quale sono evidenziate anche le informazioni necessarie per attuare il sincronismo tra le righe. Se invece pensiamo allo sviluppo verticale dell'immagine, appare che la stessa viene discretizzata dalle righe, poiché esse sono in numero finito.
Il segnale video generato dalla telecamera modula in ampiezza una portante (onda elettromagnetica ad elevata frequenza) e viene trasmesso mediante antenne agli apparati riceventi. Una volta ricevuta la portante, a seguito della sua demodulazione si riottiene il segnale video originale, che va a comandare un dispositivo di visualizzazione

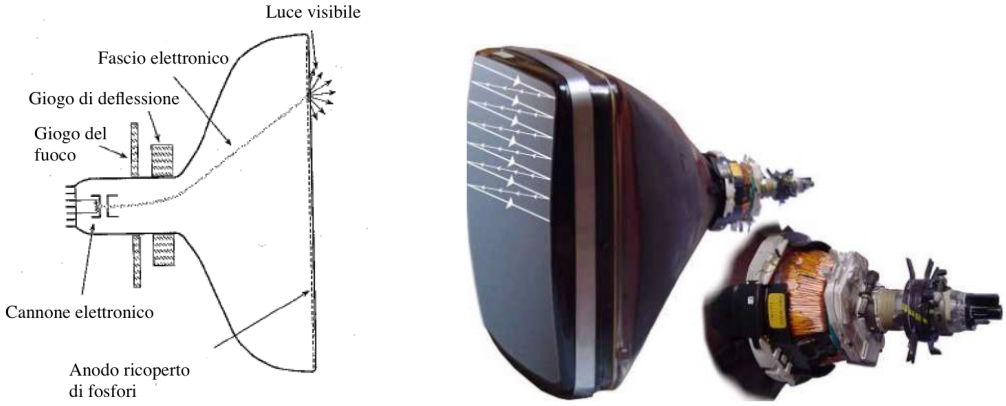


(a) Diffusione degli standard televisivi PAL, SECAM e NTSC

(b) Segnale video

Figura 2.37: Segnale video e relativi standard televisivi

che si basa ancora sulla tecnologia CRT. La ricostruzione dell'immagine avviene con la stessa tecnica usata per la sua scansione, cioè quella delle righe interlacciate che formano un quadro e con la ripetizione dei quadri secondo lo standard dei 50 o 60 Hz. Anche in questo caso c'è un pennello elettronico e un anodo (fig.2.38a); il pennello descrive la scansione delle righe interlacciate e dei quadri usando la stessa matrice della figura 2.36b. La scansione

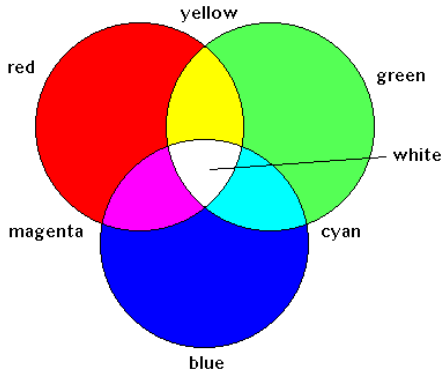


(a) Principio di funzionamento di un tubo CRT

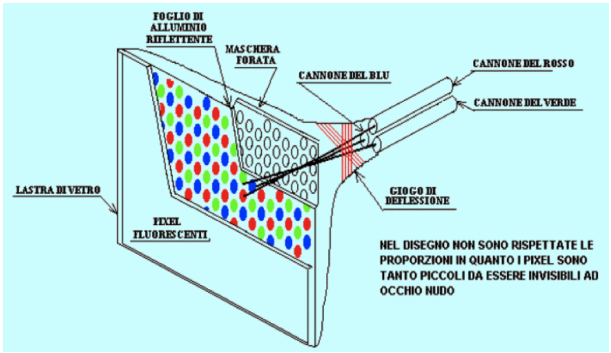
(b) Giogo di deflessione e di fuoco di un tubo CRT

Figura 2.38: Tubo a raggi catodici (CRT) e relativi gioghi di deflessione

della riga da parte del pennello e la successiva concatenazione tra le righe viene attuata da delle bobine (*gioghi di deflessione*, fig.2.38b) che realizzano un campo magnetico in grado di deflettere in modo opportuno il fascio elettronico prima che questa raggiunga lo schermo; l'unica differenza rispetto al tubo della telecamera è che ora il pennello deve generare una luminosità proporzionale a quella dell'immagine. Ciò si ottiene colpendo col pennello un sottile strato di fosfori depositati sulla parte interna dello schermo. L'energia del fascio di elettroni del pennello fa compiere un salto quantico agli elettroni delle orbite più esterne degli atomi dei fosfori, i quali decadendo a livelli quantici di minore energia generano una radiazione luminosa che viene percepita oltre il vetro del tubo CRT (fig.2.38a). Si noti che in ogni istante, a rigore, è acceso un solo punto dell'immagine, ma la visibilità complessiva della stessa e la sua apparente stabilità è dovuta solamente a un effetto ottico legato alla persistenza della retina e a quella dei fosfori che sono usati per la rappresentazione dell'immagine. Tutto quanto detto finora va bene per una trasmissione a livelli di grigio (o televisione bianco e nero), nella quale il segnale video descrive solo la luminosità dell'immagine, ma non distingue i colori. La creazione dei colori avviene con l'espediente di intro-



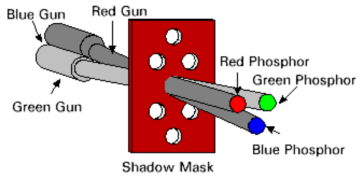
(a) La combinazione dei tre colori base, rosso, verde e blu, in proporzioni opportune consente di creare qualunque colore



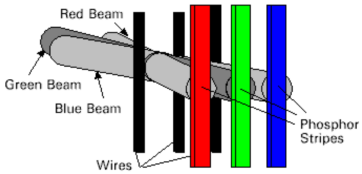
(b) Il tubo CRT a colori è caratterizzato dall'impiego di tre segnali video, uno per ciascun colore

Figura 2.39: Creazione dei colori e principio di funzionamento del tubo CRT a colori

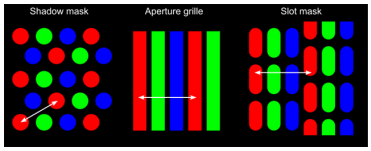
durre tre segnali video contemporanei, associati ai colori base RGB (rosso, verde, blu) che si possono combinare in proporzioni opportune per generare un qualunque colore dello spettro del visibile (fig.2.39a). Di conseguenza il tubo CRT deve avere tre cannoni e tre pennelli elettronici, uno per ciascun colore (fig.2.39b). Per poter creare l'effetto colore che deriva dal mescolamento delle tre componenti, si sfrutta la tendenza dell'occhio a mescolare i colori base quando questi sono percepiti a una distanza opportuna. In altre parole i colori generati dai fosfori dello schermo sono sempre e solo i tre colori base, che vengono percepiti come un unico colore combinazione dei tre se le dimensioni dei punti luminosi sono trascurabili rispetto alla distanza alla quale si osserva l'immagine sullo schermo. I fosfori sullo schermo vengono allora depositati su una griglia che può assumere diverse forme a seconda dello standard usato, come si vede in figura 2.40. Lo standard classico è quello denominato *Shadow mask*



(a) Maschera standard per la deposizione dei fosfori (*Shadow mask*)



(b) Maschera dello standard *Trinitron* denominata *Aperture grille*



(c) Differenza nella visualizzazione dei tre standard

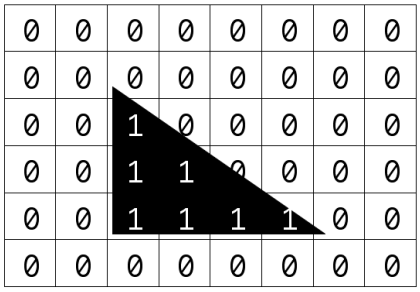
Figura 2.40: Standard in uso per la costruzione del colore a partire dalla posizione dei fosfori

(fig.2.40a), introdotto dalla *RCA* nel 1953 e basato su una maschera forata in cui i tre puntini luminosi RGB sono disposti sui vertici di un triangolo. Lo svantaggio principale di tale soluzione è che circa il 25% dell'energia del fascio va dispersa sulla maschera. Il secondo standard, realizzato dalla Sony nel 1966, si basa su delle linee RGB verticali affiancate (fig.2.40b), perfezionate successivamente dallo standard *slot mask* introdotto successivamente dalla *NEC* (fig.2.40c).
La tecnica della manipolazione analogica delle immagini e la tecnologia degli schermi CRT sono oramai superate.

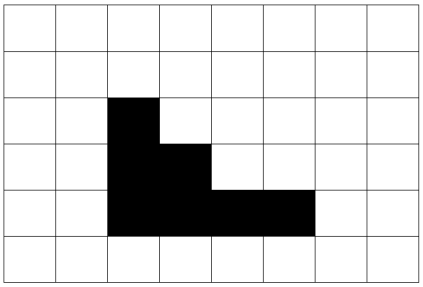
Elaborazione digitale delle immagini

Un approccio completamente diverso alla rappresentazione e alla elaborazione delle immagini emerge a partire dai primi anni '70, grazie allo sviluppo dei microprocessori. Poiché si deve pervenire a una digitalizzazione dell'immagine, l'idea è quella di suddividere la stessa in una griglia di n colonne e m righe, in modo da rappre-

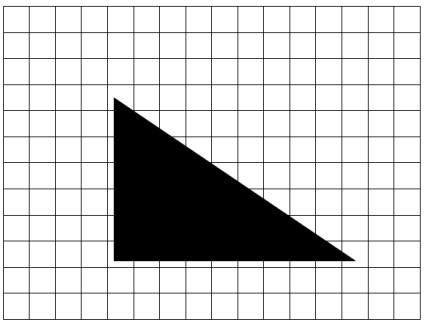
sentarla mediante $n \cdot m$ elementi d'immagine chiamati *pixel* (che deriva da *picture elements*). Una volta costruita la griglia si può attribuire, nel caso di un'immagine a bianco e nero, il valore 1 a un pixel che copre una parte nera dell'immagine e 0 a un pixel che copre una parte bianca. Nel caso in cui un pixel copra contemporaneamente una parte bianca e una nera si può decidere sulla base della maggioranza della superficie. In figura 2.41a si vede un'immagine di 8 colonne e 6 righe, cioè con 48 pixel di *risoluzione*, che rappresenta un semplice triangolo in bianco e nero. Si noti che una volta attribuito il valore 0 o 1 al *pixel* sulla base della predominanza del bianco o del nero, la decodifica porta a una figura (fig. 2.41b) che risulta frastagliata a causa della bassa risoluzione dell'immagine. Aumentando la finezza della grana, e quindi il numero di righe e di colonne (che ora sono $16 \cdot 12$), la qualità



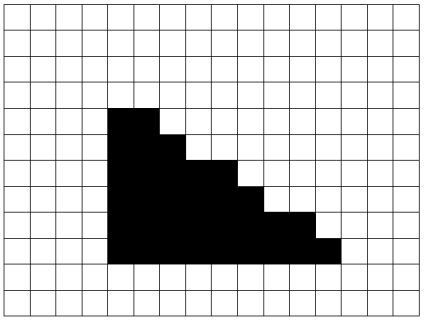
(a) Codifica di un'immagine a bianco e nero con bassa risoluzione



(b) La relativa decodifica genera un'immagine frastagliata



(c) Codifica della stessa immagine con una risoluzione più elevata

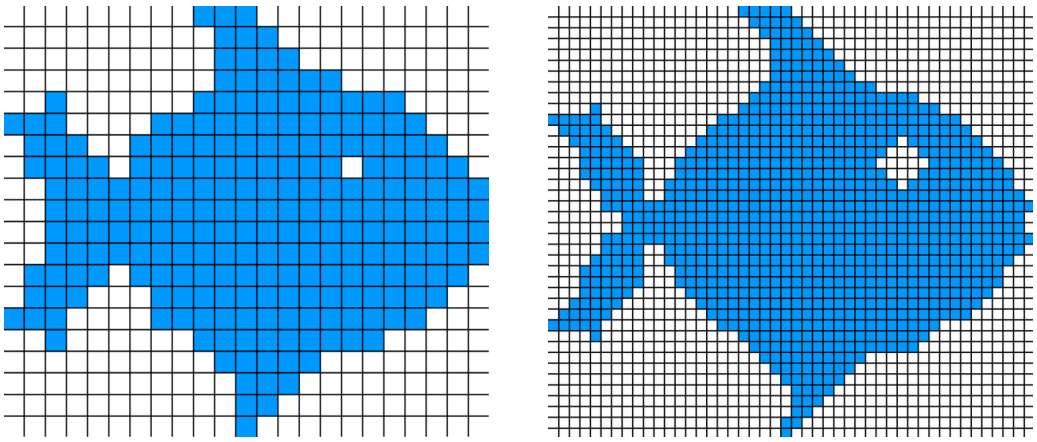


(d) Aumentando la risoluzione la qualità dell'immagine migliora

Figura 2.41: Gestione digitale delle immagini mediante matrice di *pixel*

dell'immagine migliora di conseguenza (fig. 2.41c e 2.41d). La contropartita nel miglioramento della qualità è data dall'aumento considerevole della quantità di memoria associata all'immagine.
La figura 2.42 mostra la variazione della qualità dell'immagine su un esempio un po' più articolato, basato su una risoluzione $23 \cdot 20 = 460$ *pixel* per la figura 2.42a e $46 \cdot 40 = 1840$ *pixel* per la figura 2.42b. Se ipotizziamo una figura quadrata di dimensione $n \cdot n$, il numero totale di *pixel* cresce dunque con il *quadrato* della dimensione del lato.
Le immagini viste finora sono caratterizzate da due soli livelli di colore, bianco e nero; pur restando nell'ambito delle immagini *monocromatiche* possiamo migliorare la loro granularità e la percezione delle sfumature ammettendo una scala di grigi che vada dal bianco brillante al nero. Per distinguere tra due livelli (bianco e nero) basta un singolo *bit*; se vogliamo invece distinguere tra M livelli servono $\lceil \log_2 M \rceil$ *bit*. Se poniamo $M = 2^b$ serviranno b bit. La figura 2.43 ci mostra un esempio con $b = 4$ e $2^4 = 16$ livelli di grigio, assieme alle rispettive codifiche espresse in numero decimale, esadecimale (cioè in base 16) e binario. L'introduzione di 2^b livelli di grigio aumenta l'occupazione di memoria dell'immagine, com'era lecito aspettarsi, visto che ora a ogni *pixel* sono associati b *bit* di memoria; i *bit* totali associati all'immagine sono allora $m \cdot n \cdot b$ e con essi è possibile realizzare $2^{m \cdot n \cdot b}$ possibili immagini diverse. I valori più comuni di b usati in passato per le scale di grigio erano 8, che corrispondono a 1 byte e che consentono 256 livelli di grigio, e 16, che consentono 65536 livelli di grigio.

Dal punto di vista tecnico possiamo fare le seguenti due osservazioni; la prima è che il *pixel* risulta essere la più piccola zona *indirizzabile* dell'immagine, dove per indirizzabile s'intende il fatto che il processore ha su di essa un controllo totale, potendo variare l'intensità di grigio (o il colore) del singolo *pixel* in modo indipedente dai *pixel* adiacenti. La seconda osservazione è che nel caso monocromatico la variazione della tonalità di grigio è legata essenzialmente alla variazione di luminosità del *pixel*. Quando si vuole introdurre il colore nelle immagini digitali bisogna ricorrere alla tecnica della ricostruzione RGB dello stesso basata sui tre colori base, rosso, verde e blu, di cui abbiamo già discusso precedentemente con riferimento ai monitor CRT (vedi fig. 2.39a). In questo caso il *pixel* è suddiviso in tre *subpixel* colorati, uno per ogni componente, ciascuno dei quali viene comandato dalla corrispondente scala di livelli di emissione monocromatica. Poiché i *subpixel* sono molto piccoli, l'effetto è quello di un colore dato dalla mescolanza dei tre colori base quando la distanza da cui si osserva il *pixel* è molto maggiore della sua dimensione. La figura 2.44a ci mostra la struttura dei *subpixel* su un moderno schermo piatto.
Nei vecchi monitor CRT, basati sulle schede grafiche denominate VGA, la *profondità di colore* era di 8 *bit* per ciascun pixel (8 *bit* color), di cui 3 per il rosso e il verde (8 livelli) e 2 per il blu (4 livelli); la scelta di usare solo due bit per il blu è legata al fatto che l'occhio umano ha una minore sensibilità su questa lunghezza d'onda. Nelle codifiche successive si è passati prima a 16 *bit* per pixel (*High color*) e successivamente a $24, 30, 32, 36, 48, 64$ *bit* per pixel. Nel caso del sistema *High color* vengono assegnati 5 *bit* per rosso e blu e 6 *bit* per il verde, cui corrisponde una migliore sensibilità del sistema visivo umano. Il sistema *Truecolor* è invece caratterizzato da 8 *bit* e 256 livelli possibili per ciascun canale RGB. La figura 2.44b ci consente di vedere come vengono creati i colori *Truecolor* calibrando le diverse componenti RGB nell'intervallo $[0 - 255]$. Le altre combinazioni possibili sono riportate nella tabella di figura 2.45. La presenza di un certo numero di *bit* per il cosiddetto *Canale* α consente di mescolare i colori dell'immagine da riprodurre con quelli dello sfondo, in modo da dare l'effetto di una *trasparenza* più o meno marcata. La figura 2.46 ci fa vedere la scomposizione di un'immagine nelle componenti RGB, mentre la 2.47 ci mostra come migliori la qualità dell'immagine all'aumentare dei livelli di colore.
Il progressivo aumento della risoluzione e della dimensione degli schermi ha portato a un significativo miglioramento della qualità delle immagini, che sono passate da una risoluzione 640×200 delle prime schede video CGA, introdotte da IBM nel 1981, ai 3840×2400 *pixel* dello standard WQUXGA dei più grandi monitor ad oggi mai costruiti. La tabella di figura 2.48 ci mostra gli standard principali che si sono succeduti nel corso degli anni. Si osservi che la seconda parte della tabella, da HXGA in poi, fa riferimento a standard che non sono stati ancora impiegati in monitor commerciali, ma che sono in già in uso su sensori di alcune fotocamere. L'ultima riga mette in evidenza le caratteristiche del sensore commerciale a massima risoluzione oggi esistente, da $8176 \times 6132 = 50,14$ Mega *pixel*, che equipaggia la fotocamera *Hasselblad H4D*, ma che non fa riferimento ad alcun standard.
Questo aumento impressionante nella qualità dell'immagine, oramai molto vicina alla granularità ottenibile dallo sviluppo dell vecchie pellicole, ha tuttavia un costo rilevante in termini di memoria occupata. Nell'ultima colonna

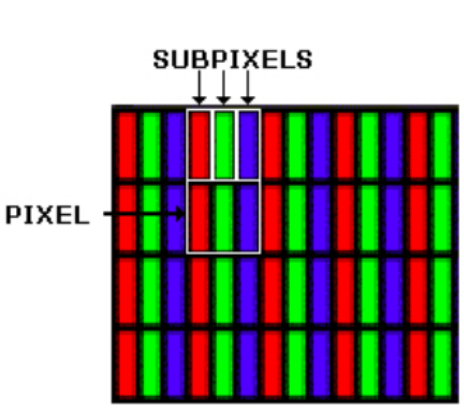


(a) $23 \cdot 20 = 460$ *pixel* (b) $46 \cdot 40 = 1840$ *pixel*

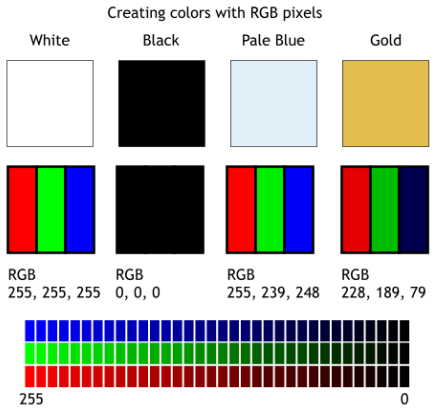
Figura 2.42: Miglioramento della qualità dell'immagine all'aumento della risoluzione

Livello di grigio																
Dec	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hex	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
Bin	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Figura 2.43: Scala dei grigi ricavata da 4 *bit* e 2^4 livelli



(a) Struttura dei *subpixel* che realizzano l'effetto cromatico del colore mediante la mescolanza delle tre componenti RGB



(b) Composizione dei colori con lo schema *Truecolor*; per ogni canale RGB si hanno a disposizione i livelli dell'intervallo $[0 - 255]$

Figura 2.44: Costruzione dei colori mediante *subpixel* RGB

Bit per *pixel*	R	G	B	Canale α	Numero di livelli	Nome
8	3	3	2	-	256	8 *bit* color
16	5	6	5	-	65536	High color
24	8	8	8	-	16777216	True color
32	8	8	8	8	16777216	True color
30	10	10	10	-	$1,073 \cdot 10^9$	Deep color
36	12	12	12	-	$68,7 \cdot 10^9$	Deep color
48	16	16	16	-	$281,5 \cdot 10^{12}$	Deep color
64	16	16	16	16	$281,5 \cdot 10^{12}$	Deep color

Figura 2.45: Principali codifiche RGB

della tabella di figura 2.48 troviamo il peso in *MegaByte* delle singole immagini associate ai vari standard. Possiamo allora notare che un'immagine XGA ha 768 kB, che corrisponde a poco più della metà della capacità dei vecchi *floppy-disk* da $1,44$ MB. Per un'immagine 3840×2400 in formato WQUXGA servono oltre 35 MB, mentre l'immagine fissata dalla *Hasselblad H4D* necessita di oltre 191 MB. È vero che molto spesso si può ricorrere a una compressione del file corrispondente, che però funziona bene solo con immagini piuttosto uniformi. Se l'immagine è molto complessa e con sfondi irregolari la compressione porta a risultati modesti, a meno che non venga effettuata una compressione *psicovisuale*, che comporta però una certa perdita d'informazione (l'immagine decompressa non è più identica all'originale). Se inoltre siamo di fronte a immagini nell'ambito medico non è neanche lecito effettuare una compressione di questo genere, poichè una perdita d'informazione visuale sarebbe eticamente inaccettabile.

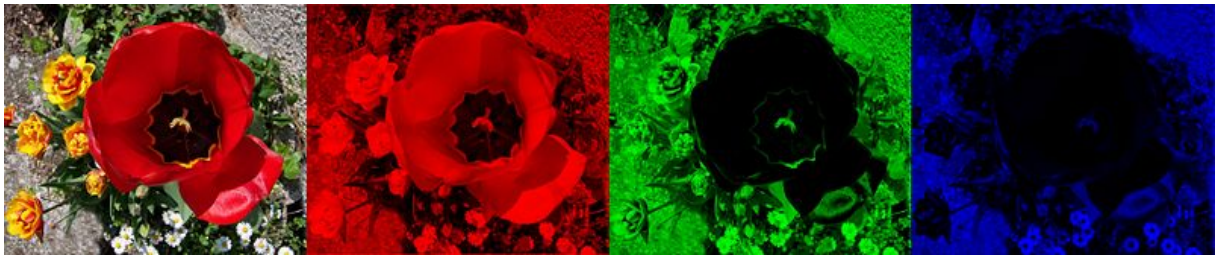
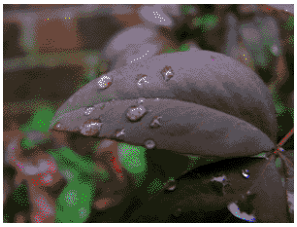


Figura 2.46: Scomposizione di un'immagine nelle componenti RGB

(a) Immagine monocromatica con 2 livelli di colore, bianco e nero (1-*bit* image)

(b) Immagine con 4 livelli di colore (2-*bit* image)



(c) Immagine con 16 livelli di colore (4-*bit* image)

(d) Immagine con 256 livelli di colore (8-*bit* image)

Figura 2.47: Miglioramento della qualità dell'immagine all'aumentare dei livelli di colore

Sigla	Risoluzione	Nome	Aspect Ratio	Bit	M*Pixel*	Memoria
CGA	640×200	Color Graphics Adapter	16:5	1	0,13	15,63 kB
EGA	640×350	Enhanced Graphics Adapter	64:35	4	0,22	109,38 kB
VGA	640×480	Video Graphics Array	4:3	4	0,31	150 kB
SVGA	800×600	Super VGA	4:3	4	0,48	234 kB
XGA	1024×768	Extended GA	4:3	8	0,79	768 kB
WXGA	1280×768	Wide XGA	5:3	32	0,98	3,75 MB
SXGA	1280×1024	Super XGA	5:4	32	1,31	5,00 MB
WSXGA+	1680×1050	Wide SXGA Plus	16:10	32	1,76	6,73 MB
UXGA	1600×1200	Ultra XGA	4:3	32	1,02	7,32 MB
WUXGA	1920×1200	Wide Ultra XGA	16:10	32	2,30	8,79 MB
QXGA	2048×1536	Quad XGA	4:3	32	3,15	12 MB
QSXGA	2560×2048	Quad Super XGA	5:4	32	5,24	20 MB
WQSXGA	3200×2048	Wide QSXGA	25:16	32	6,55	25 MB
WQUXGA	3840×2400	Wide Quad Ultra XGA	16:10	32	9,22	35,16 MB
HXGA	4096×3072	Hexadecatuple XGA	4:3	32	12,58	48 MB
WHXGA	5120×3200	Wide HXGA	16:10	32	16,38	62,5 MB
HSXGA	5120×4096	Hexadecatuple SXGA	5:4	32	20,97	80 MB
WHSXGA	6400×4096	Wide Hexadecatuple SXGA	25:16	32	26,21	100 MB
HUXGA	6400×4800	Hexadecatuple UXGA	4:3	32	30,72	117,19 MB
WHUXGA	7680×4800	Wide Hexadecatuple UXGA	16:10	32	36,86	140,63 MB
Sensore	8176×6132	Fotocamera Hasselblad H4D	4:3	32	50,14	191,25 MB

Figura 2.48: Principali standard di risoluzione per monitor e loro caratteristiche

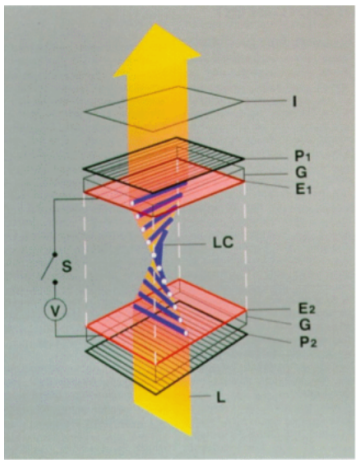
Tipo di immagine	Risoluzione in *Pixel*	*Bit* per *Pixel*
Scintillation camera (planar)	64×64 o 128×128	8 o 16
SPECT	64×64 o 128×128	8 o 16
PET	128×128	16
Digital fluoroscopy, cardiac catheter lab	512×512 o 1024×1024	8 o 12
Computed radiography, digitalized chest film	2000×2500	$10 - 12$
Mammography $(18 \times 24)(24 \times 30)$	da 1800×2300 a 4800×6000	$12 - 16$
X-ray CT	512×512	12
MRI	da 64×64 a 1024×1024	12
Ultrasound	512×512	8
Abbreviazioni: SPECT - *Single Photon Emission Computed Tomography*; CT - *Computed Tomography* PET - *Positron Emission Tomography*; MRI - *Magnetic Resonance Imaging*		

Figura 2.49: Valori tipici di risoluzione e di profondità del colore per immagini radiologiche

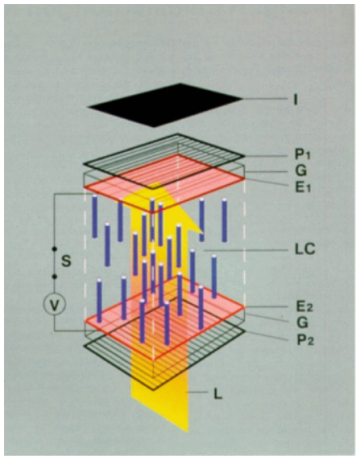
Quando si ha a che fare con le immagini i calcolatori devono essere dotati di adeguate caratteristiche tecniche per la lor gestione efficiente, cioè avere una memoria primaria (RAM) e una memoria di massa (HD) molto corpose, processori (o multiprocessori) e connessioni di rete molto veloci, altrimenti tutto si rallenta in modo inaccettabile. Per quanto riguarda le immagini in ambito radiologico non esistono degli standard specifici, e ogni costruttore propone apparecchiature con caratteristiche specifiche. Esiste tuttavia un ambito di risoluzioni che è tipico per la tecnologia impiegata per acquisire l'immagine, e la tabella di figura 2.49 ci mostra alcuni tra questi valori.

La tecnologia degli schermi digitali

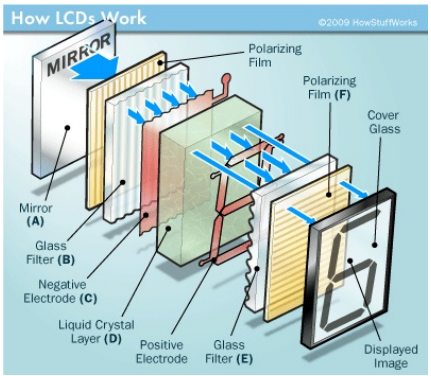
Il trapasso tra tecnologia analogica e digitale nell'ambito dell'elaborazione delle immagini è stato affiancato da una (quasi) contemporanea transizione tra monitor CRT a raggi catodici e schermi piatti basati sulla tecnologia LCD dei *cristalli liquidi* (*Liquid Crystal Display*). La tecnologia LCD sfrutta la proprietà di alcuni composti organici (bifenile) di modificare la propria trasparenza in funzione di un campo elettrico applicato. Più precisamente



(a) Stato OFF di un cristallo LCD *Twisted Nematic* (assenza di campo elettrico)



(b) Stato ON di un cristallo LCD *Twisted Nematic* (campo elettrico applicato)

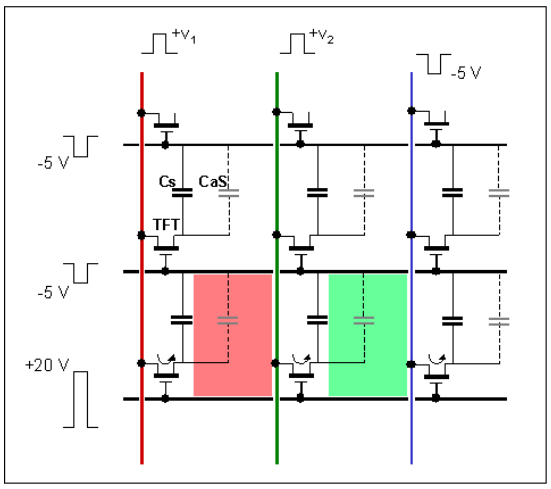


(c) Funzionamento di un cristallo LCD di tipo alfanumerico

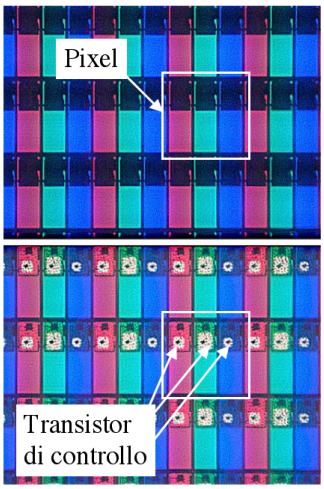
Figura 2.50: Funzionamento di uno schermo LCD a cristalli liquidi in modalità di cristallo normalmente chiaro

si sfrutta il cosiddetto *effetto nematico ritorto* o *Twisted Nematic* (TN), nel quale le molecole del bifenile risultano disposte lungo una configurazione a elica (bastoncini blu di fig.2.50a) e il passaggio della luce viene comandato da un campo elettrico. Il cristallo liquido (LC) è contenuto tra due elettrodi trasparenti (E_1, E_2), due vetri (G) e due schermi polarizzati (P_1, P_2). Quando nel cristallo non c'è un campo elettrico applicato si è nello stato OFF, la luce supera il primo polarizzatore P_2, passa attraverso il cristallo liquido e subisce una progressiva rotazione di $90°$ della polarizzazione, uscendo dallo schermo polarizzatore P_1 ortogonale a P_2. Quando invece viene applicato un campo elettrico (stato ON), le molecole del bifenile si orientano tutte lungo le linee del campo, la luce non subisce alcuna rotazione della polarizzazione e viene poi bloccata dal polarizzatore P_1, a essa ortogonale (fig.2.50b), facendo apparire il cristallo scuro. Questa modalità di funzionamento corrisponde a un cristallo *normalmente chiaro* (cioè in assenza di campo elettrico). Per cambiare logica di funzionamento (cristallo *normalmente scuro*) è sufficiente orientare di $90°$ uno dei due schermi polarizzatori, in modo che siano tra loro paralleli invece che ortogonali.

Gli schermi LCD, introdotti dalla NEC nel 1986, non generano una luce propria, ma sfruttano quella ambien-



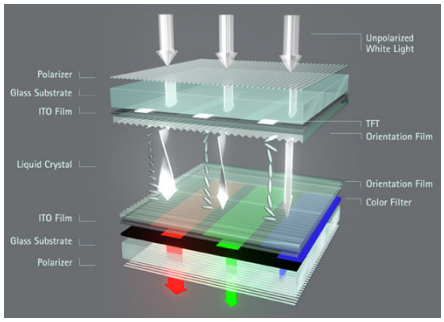
(a) Struttura della matrice TFT che comanda i *subpixel* RGB in un cristallo liquido *Twisted Nematic* normalmente scuro



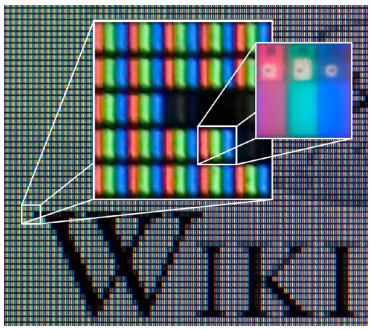
(b) Gruppo di *subpixel* con i relativi transistor di comando

Figura 2.51: Struttura della matrice dei *subpixel*

tale che viene riflessa da una superficie speculare posta dietro al cristallo (il "mirror" di fig.2.50c); per questo motivo vengono detti anche schermi *a matrice passiva*. In tale tipo di schermo ogni elemento viene comandato direttamente dalla tensione che serve per farlo funzionare. Questa soluzione rappresenta però una limitazione alla costruzione di schermi con elevata risoluzione, poiché sarebbero necessarie milioni di connessioni per comandare correttamente tutti i singoli *pixel*. Per questo motivo a partire dagli anni '80 si sviluppò la tecnologia TFT-LCD a *matrice attiva* (Thin-Film-Transistor LCD), nella quale il potenziale elettrico di ciascun *pixel* è comandato da un transistor (fig.2.51a). Tutti i transistor (del tipo MOS-FET) sono collegati per righe, tramite l'elettrodo d'ingresso denominato *gate*, e per colonne, tramite l'elettrodo comune chiamato *source*. Le righe vengono alimentate secondo una sequenza temporale, una dopo l'altra, con un impulso $+20$ visibile in figura 2.51a, in modo da portare nella posizione ON tutti i transistor e consentire alle colonne dei dati di trasmettere le eventuali cariche agli elettrodi C_aS del cristallo liquido. Se la carica è presente (segnali $+V_1$ e $+V_2$) il cristallo TN (normalmente scuro) si polarizza, passa nello stato ON, diventa trasparente e la luce passa attraverso il filtro colorato corrispondente, rosso e verde per i *subpixel* della figura 2.51a, che genereranno quindi il colore giallo. Se la carica non è presente (segnale $-5V$) il cristallo TN rimane scuro. Una volta caricato, il singolo *subpixel* mantiene la carica grazie alla capacità C_s e alla capacità parassita C_aS, almeno fino a quando arriva il nuovo comando di riga. Se il pixel deve nel frattempo variare stato, il nuovo comando di colonna sarà del tipo $-5V$ e porterà all'oscuramento dello stesso. La figura 2.52 mostra un'esploso completo di uno schermo TFT-LCD e l'ingrandimento di un'immagine che evidenzia un gruppo di *pixel* e i transistor dei relativi *subpixel*. Ricordiamo infine che la tecnologia TFT-LCD necessita di una sorgente luminosa applicata sullo sfondo, che può essere basata su una luce fluorescente (*Cold Cathode Fluorescent Light* -

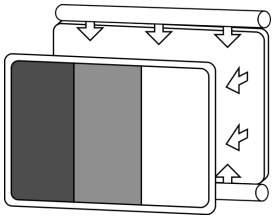


(a) Esploso completo della struttura di uno schermo a matrice attiva del tipo TFT-LCD con cristallo liquido *Twisted Nematic* normalmente scuro

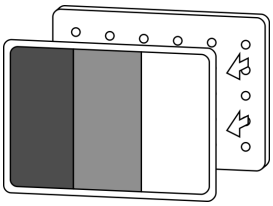


(b) Ingrandimento di un'immagine e relativi *subpixel*

Figura 2.52: Struttura di uno schermo TFT-LCD con particolore dei *subpixel*



(a) Luce di sfondo generata da tubi fluorescenti (CCFL)

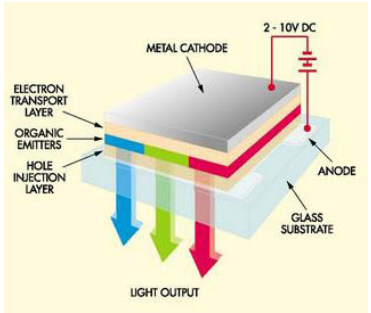


(b) Luce di sfondo generata da LED

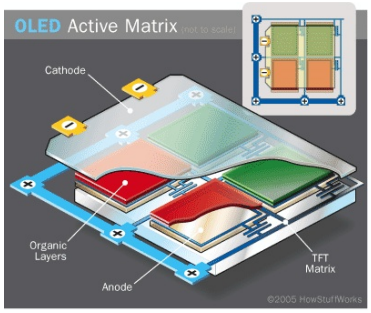
Figura 2.53: Due diverse tecniche per l'illuminazione della matrice TFT-LCD

CCFL) oppure su una luce generata da dei LED (*Light Emitting Diode*) (fig. 2.53).

Recentemente, a partire dal 2000, si è consolidata una nuova tecnologia a matrice attiva denominata OLED (*Organic Light Emitting Diode*); il principio fisico è legato al fenomeno della *elettroluminescenza* di alcuni composti organici e inorganici, vale a dire la loro capacità di emettere una radiazione luminosa quando sottoposti a un campo elettrico o percorsi da una corrente (fig.2.54). In questo modo si evita l'uso della sorgente di luce dello sfondo e il singolo *subpixel* si comporta di fatto come un diodo LED che genera una radiazione rossa, verde o blu, a seconda del composto usato.



(a) Principio di funzionamento di uno schermo OLED



(b) Struttura della matrice di uno schermo OLED

Figura 2.54: Struttura di uno schermo a matrice OLED (*Organic Light Emitting Diode*)