
HOEPLI
TECNICA
PER LA SCUOLA
PAOLO CAMAGNI
RICCARDO NIKOLASSY
TECNOLOGIE
E PROGETTAZIONE
DI SISTEMI INFORMATICI
E DI TELECOMUNICAZIONI
Per l'articolazione INFORMATICA
degli Istituti Tecnici
settore Tecnologico
1
RISORSE
ONLINE
HOEPLI

PAOLO CAMAGNI RICCARDO NIKOLASSY

Tecnologie e progettazione di sistemi informatici e di telecomunicazioni

Per l'articolazione Informatica degli Istituti Tecnici settore Tecnologico

VOLUME 1

EDITORE ULRICO HOEPLI MILANO

UN TESTO PIÙ RICCO E SEMPRE AGGIORNATO
Nel sito **www.hoepliscuola.it** sono disponibili:
- materiali didattici integrativi;
- eventuali **aggiornamenti** dei contenuti del testo.

Via Hoepli 5, 20121 Milano (Italy)
tel. +39 02 864871 – fax +39 02 8052886
e-mail hoepli@hoepli.it

www.hoepli.it

Indice

Modulo 3 La codifica dei numeri

Modulo 4 il sistema operativo

Modulo 5 Fasi e modelli di gestione di un ciclo di sviluppo

Presentazione

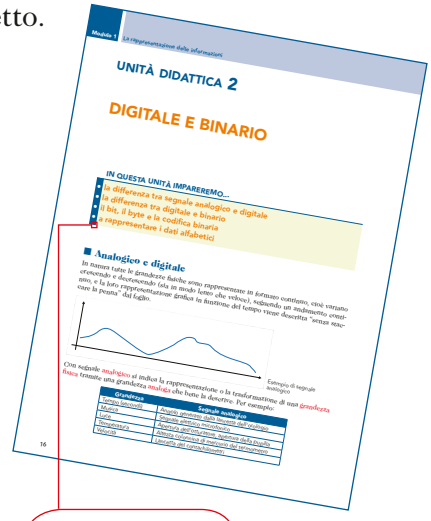
L'impostazione del presente corso in tre volumi è stata realizzata sulla base delle indicazioni ministeriali in merito a conoscenze ed abilità proposte per la nuova disciplina Tecnologie e progettazione di sistemi informatici e di telecomunicazioni. L'opera è in particolare adatta all'articolazione Informatica degli Istituti Tecnici settore Tecnologico, dove la materia è prevista nel secondo biennio e nel quinto anno del nuovo ordinamento.

Abbiamo ritenuto irrinunciabile fare tesoro della nostra esperienza maturata nel corso di numerosi anni di insegnamento che ci ha reso consapevoli della difficoltà di adeguare le metodologie didattiche alle dinamiche dell'apprendimento giovanile e ai continui cambiamenti tecnologici che implicano sempre nuove metodologie di comunicazione, per proporre un testo con una struttura innovativa, riducendo l'aspetto teorico e proponendo un approccio didattico di apprendimento operativo, privilegiando il "saper fare".

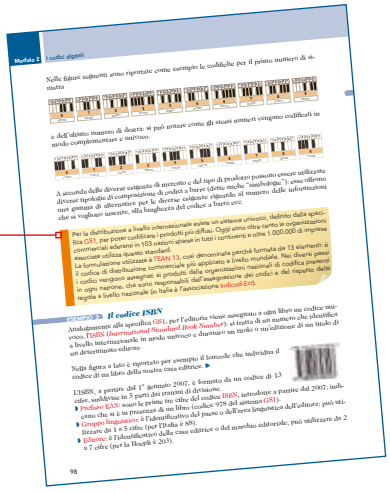
Il volume è strutturato in cinque moduli suddivisi in unità didattiche che ricalcano le indicazioni dei programmi ministeriali per il terzo anno di studio: lo scopo di ciascun modulo è quello di presentare un intero argomento, mentre quello delle unità didattiche è quello di esporne un singolo aspetto.

Nella pagina iniziale di ogni modulo è presente un indice delle unità didattiche trattate

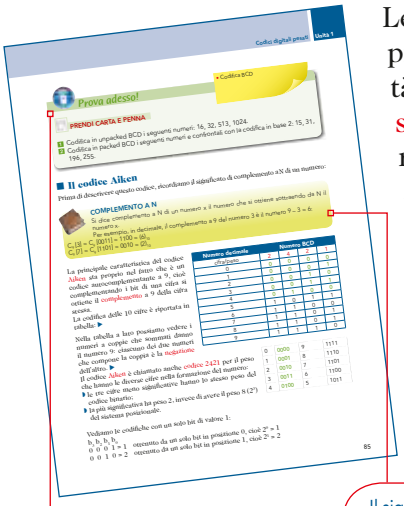
Indice degli obiettivi che si intendono raggiungere e delle attività che si sarà in grado di svolgere



All'inizio di ogni unità didattica sono indicati in modo sintetico i contenuti



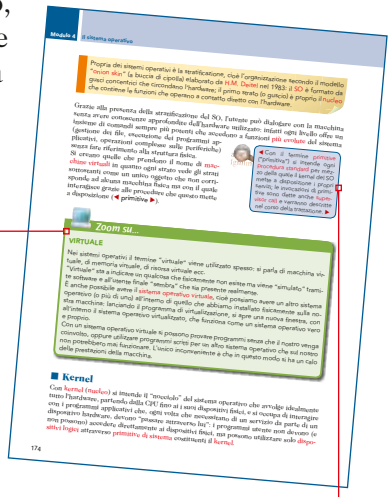
Le osservazioni aiutano lo studente a comprendere e ad approfondire



Lo studente può mettere in pratica in itinere quanto sta apprendendo nel corso della lezione

Il significato di moltissimi termini informatici viene illustrato e approfondito

Le finalità e i contenuti dei diversi argomenti affrontati sono presentati all'inizio di ogni modulo; in conclusione di ogni unità didattica sono presenti esercizi di valutazione delle conoscenze e delle competenze raggiunte, suddivisi in domande a risposta multipla, vero o falso, a completamento, ed infine esercizi di progettazione da svolgere autonomamente.



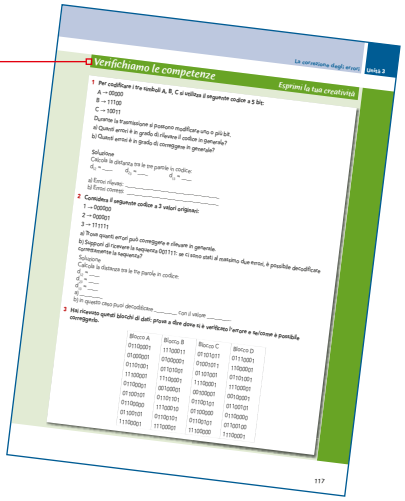
In questa sezione viene approfondito un argomento di particolare importanza

Le parole chiave vengono poste in evidenza e spiegate allo studente

RISORSE ONLINE

Alla pagina web **http://www.hoepliscuola.it** sono disponibili le risorse online, tra cui unità didattiche integrative, numerosi esercizi aggiuntivi per il recupero e il rinforzo, nonché schede di valutazione di fine modulo.

Per la verifica delle conoscenze e delle competenze è presente un'ampia sezione di esercizi



MODULO 1 LA RAPPRESENTAZIONE DELLE INFORMAZIONI

OBIETTIVI

- Acquisire il concetto di comunicazione
- Comprendere come viene gestita l'informazione nel calcolatore
- Conoscere il concetto di alfabeto, codifica e protocollo
- Comprendere la differenza tra segnale analogico e digitale
- Comprendere la differenza tra digitale e binario
- Conoscere l'origine dei sistemi di numerazione posizionale
- Conoscere il sistema decimale, ottale, binario ed esadecimale
- Rappresentare le immagini in binario
- Rappresentare i suoni in binario
- Rappresentare i filmati

ATTIVITÀ

- Saper codificare in binario
- Rappresentare i dati alfabetici
- Codificare i numeri nelle diverse basi
- Convertire un numero in base decimale
- Effettuare la conversione da basi pesate a decimale
- Effettuare la conversione da decimale a basi pesate di numeri interi e frazionali
- Convertire da binario a esadecimale
- Convertire da ottale a esadecimale
- Calcolare l'occupazione di memoria di immagini digitali
- Calcolare l'occupazione di memoria di suoni digitali

UNITÀ DIDATTICA 1

COMUNICHIAMO CON IL CALCOLATORE

IN QUESTA UNITÀ IMPAREREMO...

- il concetto di comunicazione
- l'informazione nel calcolatore
- alfabeti, codifiche e protocolli

Introduzione

Se risulta difficile comunicare tra essere umani è ancora più complesso avere a che fare con un dispositivo elettronico al quale dobbiamo impartire ordini, insegnare a eseguire compiti specifici e ricevere il risultato delle elaborazioni in un formato comprensibile per poterlo interpretare correttamente.

Il primo problema da risolvere quando due sistemi di natura diversa devono interferire tra loro, come nel caso di uomo e macchina, è quello di stabilire una forma di comunicazione, cioè fare in modo che un uomo possa trasmettere un messaggio a un elaboratore automatico ed essere in grado di interpretare le eventuali risposte.

INTERFACCIA

Definiamo interfaccia il dispositivo fisico che permette a due generici elementi, dello stesso tipo o di natura diversa, di interagire.
Le modalità con cui le due entità possono comunicare tra loro vengono stabilite mediante un insieme di regole, che prende il nome di protocollo di comunicazione.

Un semplice esempio di protocollo di comunicazione è quello che si stabilisce tra due persone di nazionalità diversa che non conoscono le reciproche lingue nazionali ma utilizzano una lingua comune, per esempio l'inglese, quando decidono la modalità con cui riuscire a comprendersi.

Nel nostro caso la comunicazione avviene tra uomo e macchina, quindi è l'uomo che deve stabilire le modalità di comunicazione, a partire dal linguaggio utilizzato e dalla sua rappresentazione simbolica.

La comunicazione

In generale la comunicazione prevede due specifici livelli, per cui dovremo cercare di superare tutti i problemi relativi a ciascuno di essi:
- livello A: trasmissione dei simboli;
- livello B: trasmissione del significato.

Per risolvere i problemi del primo livello, cioè quello della trasmissione dei simboli, il punto di partenza è individuare una "lingua comune" in modo da poter rappresentare il messaggio che dobbiamo trasmettere affinché venga ricevuto in modo corretto dalla macchina.

Il secondo livello riguarda il significato del messaggio, cioè la "semantica": il messaggio che viene ricevuto, costituito dai simboli trasmessi, deve contenere l'insieme di tutte le informazioni che il mittente ha necessità di comunicare.

Il messaggio viene trasmesso mediante un sistema di comunicazione.

Lo schema base di un sistema di comunicazione, sviluppato dal matematico Claude Shannon nella sua "teoria della comunicazione", viene così distinto nelle sue componenti essenziali.

SISTEMA DI COMUNICAZIONE

Un sistema di comunicazione è composto da:
- una sorgente di informazione, che sceglie un messaggio tra vari possibili;
- il messaggio viene inviato a un trasmettitore (o trasmittente, o emittente), il quale lo codifica in un segnale;
- il segnale viene inviato tramite un canale (o mezzo, per esempio l'aria in un messaggio verbale);
- fino a giungere al ricevitore (o ricevente).

Possiamo individuare i seguenti elementi fondamentali:
- un messaggio: composto da segnali ottici, acustici, elettrici ecc.;
- un trasmettitore: l'oggetto che invia il messaggio (telefono, computer, modem ecc.);
- un ricevitore: lo strumento che legge e decodifica il messaggio;
- un canale: il mezzo attraverso il quale viene trasmesso il segnale (fili, onde radio, luce);
- un codice: l'insieme dei simboli impiegati per adattare il messaggio alla trasmissione;
- un protocollo: l'insieme delle regole che definiscono il formato dei messaggi stessi, la loro lunghezza ecc.

In questa prima parte ci dedicheremo alla codifica del messaggio, cioè individueremo che cosa trasmettere o elaborare e lo trasformeremo in formato opportuno per essere elaborato da un agente di calcolo: ci occuperemo di come codificare l'informazione a partire dalla definizione dei simboli che ci permettono tale codifica (alfabeto).

ESEMPIO

Supponiamo che durante una verifica a scelta multipla vogliate comunicare i risultati a un compagno senza che il docente se ne accorga.
Supponiamo che sia un test a quattro distrattori e che vi dobbiate scambiare le informazioni costituite da A, B, C, D utilizzando simboli gestuali.
Per esempio vi accordate con il vostro amico sui gesti da utilizzare:

Risposta A

Risposta B

Risposta C

Risposta D

In questo esempio abbiamo:
- il messaggio: una delle quattro lettere (A, B, C, D) che indicano la soluzione del quesito;
- il trasmettitore: voi;
- il ricevitore: il vostro amico;
- il canale: l'aria, il messaggio è visuale diretto;
- il codice: la posizione delle mani e la corrispondenza con le lettere A, B, C, D;
- il protocollo: un simbolo che corrisponde a un carattere.

Tipologia dell'informazione

◀ Con il termine multimedialità si intende la presenza simultanea di più mezzi di comunicazione in uno stesso strumento informativo: per esempio una presentazione multimediale può contenere immagini in movimento (video), immagini statiche (fotografie e illustrazioni), musica e testo. ▶

Inizialmente, i calcolatori venivano utilizzati per elaborare dati numerici: se uno dei primi linguaggi di programmazione (il ForTran, Formula Translator) definiti a tale scopo si riduceva a raggruppare formule matematiche, la crescita esponenziale della tecnologia digitale (come indica la legge di Moore: "*la velocità dei processori raddoppia ogni 18 mesi*") ha portato dapprima all'elaborazione di informazioni di tipo testuale e poi a quella ◀ multimediale ▶.

Possiamo quindi elencare alcune categorie di informazioni che sono "trattate" dagli elaboratori elettronici e che analizzeremo nel dettaglio:
- numeri e operazioni numeriche;
- dati alfanumerici;
- immagini e filmati;
- suoni.

In particolare ci soffermeremo in modo approfondito sui primi due mentre degli ultimi due vedremo le caratteristiche generali.

Simbologia e terminologia

Concentriamo ora la nostra attenzione su due elementi estremi nella classificazione del sistema di comunicazione: il messaggio e il codice, che non devono essere confusi tra loro.

MESSAGGIO E CODIFICA

- con il termine messaggio si intende il contenuto dell'informazione, cioè il significato di quanto si vuole trasmettere;
- con il termine codifica si intende la modalità espressiva, ossia la forma con cui viene scritto il messaggio, cioè il significante.

ESEMPIO

Vediamo per esempio il numero 10:

10 dieci nella numerazione araba

X dieci nella numerazione romana

 dieci nella numerazione unaria

Il significante è costituito dalle tre modalità con cui viene rappresentato il messaggio attraverso dei simboli, mentre il contenuto, cioè il significato, è il numero 10: si dice che "il significante denota il significato".

L'insieme dei valori da rappresentare prende il nome di alfabeto sorgente ed è l'insieme di tutte le parole che il mittente, cioè l'origine, deve trasmettere: spesso viene indicato con

$$T = (x_1, x_2, \ldots, x_n).$$

Nell'esempio della "verifica in classe" l'alfabeto sorgente è costituito da A, B, C, D.
L'insieme dei simboli utilizzati per rappresentare le parole dell'alfabeto sorgente si chiama alfabeto in codice e si indica con

$$E = (e_1, e_2, \ldots, e_k).$$

Nell'esempio della "verifica in classe" l'alfabeto in codice è costituito dalla posizione delle mani.

Naturalmente non esiste alcuna relazione sia tra il numero dei simboli sia nella tipologia dei simboli dei due alfabeti.

Una qualunque sequenza di caratteri (che prende il nome di stringa) di lunghezza l_i di simboli di e_i è la trasformazione o codifica.

CODIFICA

La codifica è una tecnica con la quale un dato viene rappresentato mediante un definito insieme di simboli, o di dati, più elementari di qualsiasi natura (grafica, luminosità, acustica ecc.).
Con tali simboli è possibile formare sequenze che possono essere messe in relazione biunivoca con gli elementi costituenti l'informazione.

Alcuni esempi sono riportati di seguito:
- alfabeto Morse: sequenze di punti e linee rappresentanti caratteri;
- numero matricola: sequenza di cifre rappresentanti uno studente;
- codice articolo: sequenza di simboli rappresentanti un articolo di un negozio;
- codice fiscale: sequenza di caratteri rappresentanti una persona;
- parole della lingua italiana: sequenze di lettere {a, b, c, …, z};
- regolazione dell'incrocio: semaforo con luce di colorazioni diverse (G, V, R).

CODICE

L'applicazione che associa a ogni parola x_i dell'alfabeto sorgente una stringa di lunghezza l_i di simboli dell'alfabeto in codice e_i viene detta codice o tabella codice: ogni stringa della tabella codice viene detta parola codice.

Osserviamo che:
- il termine codice viene spesso usato per indicare una parola codice;
- la parola sorgente ha una lunghezza diversa da quella della parola codice.

Codifica a lunghezza fissa

CODICE A LUNGHEZZA FISSA

Supponiamo di avere:
- $T = (x_1, \ldots, x_n)$ *alfabeto sorgente*, cardinalità n, cioè composto da n elementi;
- $E = (a_1, \ldots, a_k)$ *alfabeto in codice*, cardinalità k, quindi composto da k elementi.

La parola codice ha una lunghezza $l_i = m = costante$ per tutti gli elementi di T se a ognuno degli elementi $x_i \in T$ si fa corrispondere una delle k_m disposizioni con ripetizione dei k simboli di E sugli m posti della sequenza.

Necessariamente sarà verificata la relazione $k_m > n$ in quanto gli n elementi dell'alfabeto sorgente devono trovare almeno altrettante disposizioni che li rappresentino con simboli dell'alfabeto in codice.

Il calcolo combinatorio ci ricorda che, avendo k simboli e dovendo avere n configurazioni diverse, è necessario avere una lunghezza minima m ottenuta dal numero intero eccedente al $\log_k n$, cioè:

$$n = INT_{max}(\log_k n)$$

Per esempio:
- con due simboli e $n = 7$ parole è necessario avere $m = INT_{max}\,[\log_2 7] = 3$;
- con tre simboli e $n = 12$ parole è necessario avere $m = INT_{max}\,[\log_3 12] = 3$;

- volendo codificare i caratteri stampabili di una comune tastiera per elaboratore (26 + 26 lettere, tra minuscole e maiuscole, 10 cifre decimali, 32 caratteri speciali, quali simboli di punteggiatura, operatori ecc.), cioè 94 caratteri distinti, saranno necessarie stringhe composte da:
  – $n_1 = \text{INT}_{max}[\log_2 94] = 7$ simboli di un alfabeto binario;
  – $n_2 = \text{INT}_{max}[\log_3 94] = 5$ simboli di un alfabeto di 3 simboli;
  – $n_3 = \text{INT}_{max}[\log_4 94] = 4$ simboli di un alfabeto di 4 simboli.

Zoom su...

LUNGHEZZA DI UNA CODIFICA

Le rappresentazioni più lunghe si ottengono con gli alfabeti più "poveri"; in particolare, la codifica di tipo binario, fondata solo su due simboli (tipicamente 0 e 1), necessita delle stringhe più lunghe ma si rivela la più idonea in relazione agli elaboratori elettronici e ai sistemi basati su logiche binarie e/o dispositivi bistabili (sistemi rappresentabili tramite algebre booleane o di commutazione).

Vediamo un esempio.

ESEMPIO

Codifichiamo i giorni della settimana utilizzando un alfabeto di due simboli (A, B):
- *T = (Lunedì, Martedì, Mercoledì, Giovedì, Venerdì, Sabato, Domenica) cardinalità* $n = 7$;
- *E = (A, B) cardinalità* $k = 2$.

Dato che $n = 7$ e $k = 2$ è necessaria una lunghezza $m = 3$ del codice per poterli codificare. Procediamo nel modo seguente:

1. dividiamo i giorni della settimana in due gruppi e assegniamo a entrambi un primo simbolo di differenziazione come primo carattere del codice:
   A: Lunedì, Martedì, Mercoledì, Giovedì
   B: Venerdì, Sabato, Domenica
2. allo stesso modo, dividiamo ogni gruppo così ottenuto in due sottogruppi, assegnando a ciascuno, alternativamente, un secondo elemento dell'alfabeto in codice:
   AA: Lunedì, Martedì
   AB: Mercoledì, Giovedì
   BA: Venerdì, Sabato
   BB: Domenica
3. continuiamo a dimezzare i sottoinsiemi finché non otteniamo tanti insiemi di singoli elementi e aggiungiamo, per ogni suddivisione, un nuovo simbolo alla codifica:
   AAA: Lunedì
   AAB: Martedì
   ABA: Mercoledì
   ABB: Giovedì
   BAA: Venerdì
   BAB: Sabato
   BBA: Domenica

Rimane libera la configurazione BBB, in quanto con 2 simboli e lunghezza 3 è possibile ottenere 8 disposizioni diverse: il codice si dice ridondante.

CODICE RIDONDANTE

Si ha un codice ridondante ogni volta che si ha un numero di simboli diverso dal numero massimo di simboli esprimibili.

Vedremo in seguito come viene sfruttata la ridondanza per introdurre il meccanismo di rilevazione o di correzione dell'errore.

ESEMPIO

Generiamo le parole in codice a lunghezza fissa nella seguente situazione:

- $T = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9)$
- $E = (a, b, c)$
- $m = [\log_3 9] = 2$

Avendo tre simboli nell'alfabeto suddividiamo in tre gruppi le parole sorgenti, assegnando a ogni gruppo un carattere dell'alfabeto in codice:

```
x1 = a
x2 = a
x3 = a
x4 = b
x5 = b
x6 = b
x7 = c
x8 = c
x9 = c
```

A ogni gruppo aggiungiamo una seconda parola dell'alfabeto: dato che ciascuno di essi è composto da tre elementi e abbiamo tre simboli diversi, questi sono sufficienti a differenziare ogni parola. Le parole codice sono le seguenti:

```
x1 = a a
x2 = a b
x3 = a c
x4 = b a
x5 = b b
x6 = b c
x7 = c a
x8 = c b
x9 = c c
```

Il codice ottenuto non è ridondante, dato che $k^m = n = 9$.

Codifica a lunghezza variabile

Nei codici a lunghezza variabile la parola codice ha una lunghezza costituita da un numero di cifre variabile in funzione del valore da rappresentare.

La corrispondenza viene decisa tenendo conto della frequenza con cui vengono usati i valori, in modo da ottenere come vantaggio il risparmio di spazio nella memorizzazione e di tempo nella trasmissione (per esempio viene utilizzata negli algoritmi di compressione dei dati, come il codice di Huffman, descritto in seguito).

Ne sono un esempio l'alfabeto Morse, la codifica dei prefissi telefonici della rete fissa ecc.

In pratica, però, tutti i codici utilizzati nelle trasmissioni tra computer sono a lunghezza fissa: i codici a lunghezza variabile sono usati per esempio dagli algoritmi di compressione dati.

ESEMPIO

Dato un alfabeto sorgente e un alfabeto in codice, definire due possibili codici di lunghezza massima 2, uno a lunghezza variabile e uno a lunghezza fissa:
- alfabeto sorgente: (picche, fiori, quadri, cuori);
- alfabeto in codice: (*, /).

Alfabeto sorgente	Codice a lunghezza variabile	Codice a lunghezza fissa
picche	*	**
fiori	/	//
quadri	*/	**
cuori	//	/*

È quindi possibile generare molti codici diversi anche partendo dal medesimo alfabeto sorgente e alfabeto in codice: inoltre, in ogni codice, ogni parola codice può avere lunghezza diversa anche all'interno del medesimo codice.

Vediamo un ultimo esempio:
- alfabeto sorgente: (Cucciolo, Eolo, …, Mammolo);
- alfabeto in codice: (A, B, C).

Alfabeto sorgente	Codice a lunghezza fissa	Codice a lunghezza variabile
Cucciolo	AA	A
Eolo	AB	B
Mammolo	AC	C
Gongolo	BA	AA
Dotto	BB	BB
Pisolo	BC	CC
Brontolo	CA	ABC

Prova adesso!

- Codifica
- Codice a lunghezza fissa
- Codice a lunghezza variabile

PRENDI UN FOGLIO

Codifica i mesi dell'anno con un alfabeto composto da:

1. @ # *
2. @ # * =
3. @ # * = %

generando rispettivamente per ogni alfabeto sorgente un codice:

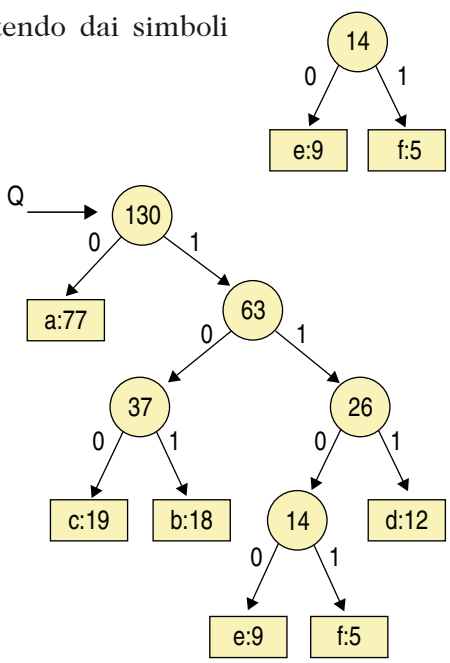
1. a lunghezza fissa minima
2. a lunghezza variabile minima

Codifica di Huffman

La codifica di Huffman genera un codice binario a lunghezza variabile assegnando ai caratteri che sono più frequenti nel messaggio una codifica più corta rispetto ai caratteri con minor frequenza, che avranno una codifica più lunga: in questo modo, a parità di messaggio da trasmettere, con questa codifica il numero di bit risulta notevolmente inferiore.
La costruzione del codice avviene innanzitutto individuando la frequenza dei singoli caratteri, come nell'esempio seguente:

carattere	*a*	*b*	*c*	*d*	*e*	*f*
frequenza	*77*	*18*	*19*	*12*	*9*	*5*

e successivamente costruendo un albero binario partendo dai simboli meno frequenti.



La costruzione dell'albero avviene ordinando in un primo tempo i simboli per frequenza di comparsa: successivamente i due simboli che appaiono meno di frequente sono collegati a un nodo il cui peso è uguale alla somma delle frequenze dei due simboli (14 = 9 + 5).

Il simbolo di peso minore è assegnato al ramo 1, l'altro al ramo 0 e così via, considerando ogni nodo formato come un nuovo simbolo, fino a ottenere un nodo genitore detto radice (peso 130).

Ogni simbolo viene codificato seguendo il percorso che parte dalla radice e arriva al simbolo stesso: quindi, più il simbolo è "profondo" nell'albero, più la parola del codice sarà lunga.

Individuiamo ora le parole in codice seguendo i diversi rami e la rispettiva occupazione come numero di bit del nostro messaggio, che era lungo 130 caratteri:

carattere	*a*	*b*	*c*	*d*	*e*	*f*
frequenza	*57*	*18*	*19*	*12*	*9*	*5*
codifica	*0*	*101*	*100*	*111*	*1000*	*1100*
occupazione	*1*57*	*2*18 = 54*	*3*19 = 57*	*12*3 = 36*	*9*40 = 36*	*5*4 = 20*

Se sommiamo il numero di bit trasmessi con questa codifica (nrbit = 260) e li confrontiamo con una qualunque codifica della stessa utilizzando un codice a lunghezza fissa (k = 3, quindi 390 bit), notiamo quanto il risparmio sia rilevante anche in un semplice esempio come questo (oltre il 30% di risparmio di spazio).

I codici di Huffman vengono ampiamente usati nella compressione dei dati (pkzip, jpeg, mp3) e normalmente permettono un risparmio compreso tra il 20 e il 90%, secondo il tipo di file.

Protocollo di comunicazione

Affinché possa effettuarsi la comunicazione tra due persone, due calcolatori ecc. è necessario definire il protocollo in modo che sorgente e ricevente possano essere in grado di effettuare il colloquio e scambiarsi messaggi reciprocamente comprensibili.
Per avere uno scambio corretto è necessario definire alcune caratteristiche della comunicazione e del messaggio: per esempio il momento di inizio e di fine della comunicazione, la lunghezza e il formato dei messaggi ecc.
Questi elementi costituiscono un insieme di regole che permettono di effettuare lo scambio delle informazioni, che prende il nome di protocollo.

PROTOCOLLO DI COMUNICAZIONE

Per protocollo di comunicazione si intende un insieme di regole che determinano le modalità con cui due soggetti (o apparecchiature) si scambiano i dati.

ESEMPIO

Riprendiamo l'esempio della comunicazione delle risposte ai quesiti tra due alunni:

- innanzitutto bisogna stabilire il momento iniziale nel quale cominciare a comunicare, in quanto i due soggetti devono essere pronti a scambiarsi i dati: devono essere sincronizzati;
- successivamente è necessario indicare a quale domanda stiamo suggerendo la soluzione;
- infine si deve indicare il termine della comunicazione.

Il nostro messaggio cambia quindi di aspetto e diviene composto da più parti:

1. inizio comunicazione;
2. numero quesito – soluzione al quesito;
3. termine comunicazione.

Il punto 2 può essere anche composto da una successione di coppie numeri/risposta finché il trasmettitore non indica al ricevitore la fine della comunicazione.

Stabiliamo per esempio il segnale di *inizio comunicazione*, *fine comunicazione*, *pronto alla ricezione* e *conferma ricezione* come indicato dai disegni che seguono.

inizio comunicazione

fine comunicazione

pronto alla ricezione

conferma ricezione

Il messaggio è composto da due segnalazioni: il numero della domanda, indicato solo con il numero delle dita, come nei disegni seguenti, accompagnato dalla risposta esatta.

domanda 2

domanda 3

Mittente:

Destinatario:

Traduciamo il messaggio attribuendo a ogni codifica il suo significato:
Mittente: "Sei pronto a ricevere?"
Destinatario: "Ok, inizia la comunicazione"
Mittente: "domanda 2-risposta C, domanda 3-risposta B, fine trasmissione"
Destinatario: "Ok, ricevuto"

Cenni sulla trasmissione e sul disturbo

Normalmente il canale, cioè il mezzo di trasmissione e di conseguenza il segnale, vengono disturbati da eventi casuali, cosicché il segnale ricevuto spesso è diverso da quello inviato. Si aggiunge quindi al nostro segnale un ulteriore segnale chiamato disturbo, che non è desiderato e che interferisce con l'intero processo.

Viene fatta una distinzione in base alla tipologia tra:
- disturbo, che ha caratteristiche prevedibili e quindi eliminabili;
- rumore, che ha caratteristiche non prevedibili.

La nostra codifica dell'informazione deve essere fatta in modo da tener conto di tale effetto e, se la capacità del canale è adeguata e il disturbo è probabilisticamente prevedibile, siamo in grado di ridurre questi ultimi a una piccola quantità a piacere.
Non ci sono limiti tecnici nella riduzione dei disturbi ma solo limiti temporali ed economici: quanto più si vuole ridurre l'errore, tanto più lunga e costosa deve essere la codifica. In pratica, ciò significa che si deve cercare una mediazione e ci si deve accontentare di precisioni "accettabili" avendo l'accortezza di non utilizzare canali "troppo" disturbati.
Una prima precauzione adottata per ridurre la possibilità di errore è quella di utilizzare un alfabeto in codice nel quale i simboli siano ben distinti tra loro in modo da avere un discreto margine di tolleranza prima che vengano scambiati tra loro per effetto di un disturbo.
Una seconda accortezza è quella di utilizzare codifiche particolari con elementi aggiuntivi (codici ridondanti) che integrino informazioni che consentano di riconoscere e correggere le eventuali stringhe errate: il ricevitore decodificherà il segnale ricevuto riconoscendo le diverse situazioni di errore e ricostruendo il messaggio iniziale mediante la correzione dell'errore, poi lo invierà a destinazione finalmente integro.

Verifichiamo le conoscenze

Esercizi a scelta multipla

1 Quali tra i seguenti non è un elemento fondamentale di un sistema di comunicazione?

- O messaggio
- O trasmettitore
- O ricevitore
- O canale
- O codice
- O alfabeto
- O protocollo

2 Con il termine "codifica" si intende:

- O una qualunque sequenza di simboli dell'alfabeto in codice
- O una qualunque sequenza di simboli dell'alfabeto sorgente
- O una qualunque sequenza di simboli da trasmettere
- O nessuna delle affermazioni precedenti

3 Nella codifica a lunghezza fissa:

- O le parole dell'alfabeto sorgente hanno lunghezza costante
- O le parole dell'alfabeto in codice hanno lunghezza costante
- O i caratteri dell'alfabeto sorgente hanno lunghezza costante
- O i caratteri dell'alfabeto in codice hanno lunghezza costante

4 Quali affermazioni sul codice di Huffman sono false?

- O genera parole in codice di lunghezza variabile
- O utilizza un alfabeto in codice di lunghezza variabile
- O viene utilizzato nella compressione dei dati
- O è un codice ridondante
- O le parole dell'alfabeto sorgente possono avere frequenze diverse

5 Quale componente non appartiene al protocollo di comunicazione?

- O segnale di inizio comunicazione
- O segnale di fine comunicazione
- O segnale di sincronismo
- O segnale di errore nella trasmissione
- O strutturazione del messaggio

Test vero/falso

1	Con interfaccia si definisce il dispositivo che permette a due generici elementi di interagire.	V F
2	Con protocollo di comunicazione si intende la modalità con cui due entità comunicano.	V F
3	Il significante denota il significato.	V F
4	L'insieme dei valori da rappresentare prende il nome di alfabeto sorgente.	V F
5	L'insieme dei simboli usati per rappresentare le parole dell'alfabeto sorgente si chiama alfabeto in codice.	V F
6	In un codice ridondante si ha un numero di simboli uguale al numero massimo di simboli esprimibili.	V F
7	Un disturbo ha caratteristiche prevedibili e quindi eliminabili.	V F
8	Un rumore ha caratteristiche prevedibili e quindi eliminabili.	V F

Verifichiamo le competenze — *Esprimi la tua creatività*

1 Avendo a disposizione i seguenti alfabeti:

- alfabeto sorgente: (Cucciolo, Eolo, …, Mammolo)
- alfabeto in codice: (A, B, C)

individua due codici che rappresentino i sette nani (Biancaneve compresa).

2 Codifica i mesi dell'anno con un alfabeto in codice composto da:

- tre simboli: @, #, *
- quattro simboli: @, #, *, =

Per ciascuno individua due possibili codici:

- con stringhe di lunghezza fissa minima
- con stringhe di lunghezza variabile minima

3 Costruisci un sistema di codifica con cinque simboli:

- scrivi le prime 16 codifiche
- indica il numero massimo di parole dell'alfabeto sorgente

4 Nell'alfabeto di Marte sono previsti 300 simboli: quale deve essere la lunghezza minima di ogni stringa codificata (con parole di lunghezza fissa) nel caso in cui si utilizzino solo due simboli nell'alfabeto in codice?

5 L'alfabeto A1 contiene i simboli {@,#,§}: quante informazioni puoi codificare con parole di lunghezza 4? Prova a codificarne 10.

6 L'alfabeto A2 contiene i simboli {a,b,c,d}: quante informazioni puoi codificare con parole di lunghezza 3? Prova a codificarne 12.

7 L'alfabeto A3 contiene i simboli {0,1,2,3,4}: quante informazioni puoi codificare con parole di lunghezza 3? Prova a codificarne 20.

8 L'alfabeto A4 = A3 contiene i simboli {x,y,z,k}: quante informazioni puoi codificare con stringhe di lunghezza 5? Prova a codificarne 25.

9 Individua l'espressione analitica (formula) che lega il numero di simboli dell'alfabeto in codice con lunghezza fissa e dell'alfabeto sorgente motivandone la risposta.

10 Dati due alfabeti A = {@,§,#} e B = {0,1}g, determina quanto deve essere la lunghezza *m* di una stringa nell'alfabeto B per trasferire tutte le stringhe di 4 caratteri scritte nell'alfabeto A.

Trova la codifica nell'alfabeto B della stringa A = §#@@.

11 Individua le frequenze di ogni carattere nella seguente frase da trasmettere:

"sopra la panca la capra campa sotto la panca la capra crepa"

Individua una codifica a lunghezza variabile che permetta di risparmiare almeno il 20% di spazio rispetto a una codifica a lunghezza fissa con un alfabeto di soli due simboli {0,1}.

UNITÀ DIDATTICA 2

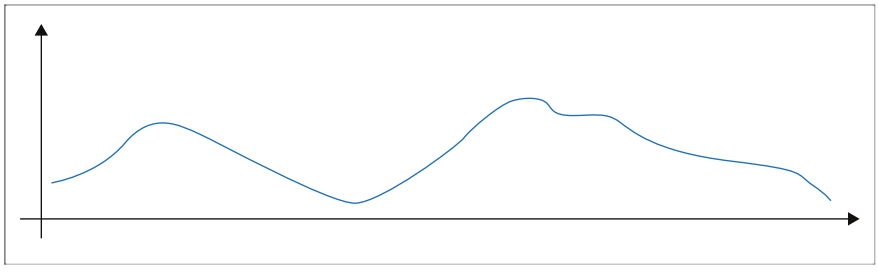
DIGITALE E BINARIO

IN QUESTA UNITÀ IMPAREREMO...

- la differenza tra segnale analogico e digitale
- la differenza tra digitale e binario
- il bit, il byte e la codifica binaria
- a rappresentare i dati alfabetici

Analogico e digitale

In natura tutte le grandezze fisiche sono rappresentate in formato continuo, cioè variano crescendo e decrescendo (sia in modo lento che veloce), seguendo un andamento continuo, e la loro rappresentazione grafica in funzione del tempo viene descritta "senza staccare la penna" dal foglio.



Esempio di segnale analogico

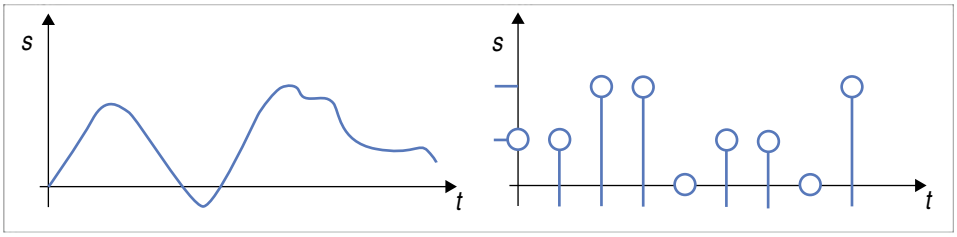
Con segnale analogico si indica la rappresentazione o la trasformazione di una grandezza fisica tramite una grandezza analoga che bene la descrive. Per esempio:

Grandezza	Segnale analogico
Tempo (secondi)	Angolo generato dalla lancetta dell'orologio
Musica	Segnale elettrico microfonico
Luce	Apertura dell'otturatore, apertura della pupilla
Temperatura	Altezza colonnina di mercurio del termometro
Velocità	Lancetta del contachilometri

La temperatura, la pressione e in generale tutte le grandezze fisiche sono per loro natura analogiche e vengono rappresentate mediante il sistema che maggiormente le identifica, generalmente modalità grafiche, dove la grandezza indipendente è il tempo.
La rappresentazione numerica di una grandezza analogica è quasi sempre data, istante per istante, da un numero reale (teoricamente con precisione infinita, ossia infinite cifre dopo la virgola), cioè è una sequenza di valori istantanei.

SEGNALI

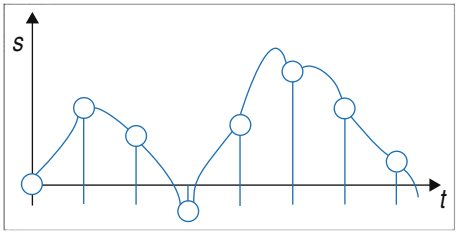
Segnale: grandezza fisica che varia nel tempo.
Segnale analogico: segnale che può assumere un insieme continuo (e quindi infinito) di valori.
Segnale *tempo-continuo*: segnale il cui valore è significativo (e può variare) in qualsiasi istante di tempo.
Segnale *tempo-discreto*: segnale il cui valore ha interesse solo in istanti di tempo prestabiliti, generalmente equidistanziati.



Segnale analogico tempo-continuo — Segnale digitale tempo-discreto

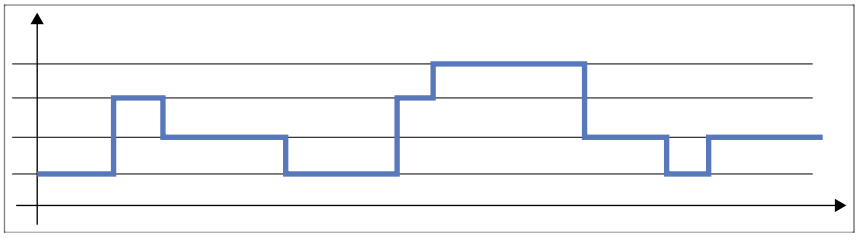
Il valore istantaneo di un segnale è un numero reale e come tale può essere codificato, come vedremo, utilizzando una notazione in virgola fissa o in virgola mobile.

È possibile effettuare dei "rilievi" parziali di un segnale analogico prelevando a particolari istanti di tempo (campionamento) il valore del segnale stesso. Si effettua così un'operazione di discretizzazione del segnale, cioè si passa da un insieme infinito di valori a un insieme discreto: tipicamente i segnali tempo-discreti hanno istanti equidistanziati in cui vengono campionati.



Campionamento

Se preleviamo il segnale precedente a intervalli di tempo prefissati e manteniamo il valore letto per tutto l'intervallo di tempo finché non effettuiamo una successiva lettura, otteniamo un segnale come quello riportato nella figura, cioè composto da un'onda rettangolare a scalini: abbiamo fatto la digitalizzazione del segnale, cioè abbiamo trasformato un segnale analogico in un segnale digitale.



Esempio di segnale digitale

Questo tipo di segnale prende il nome di digitale, termine che deriva da *digit*; a sua volta *digit*, che in inglese significa "cifra", deriva dal latino *digitus*, che significa "dito". Il significato di digitale è legato in definitiva a "ciò che è rappresentato con i numeri, che si contano appunto con le dita, inteso come insieme finito di elementi". Nella lingua italiana il termine digitale è sostituito dalla parola numerico, cioè che viene rappresentato con uno (o più) numeri.

ESEMPIO

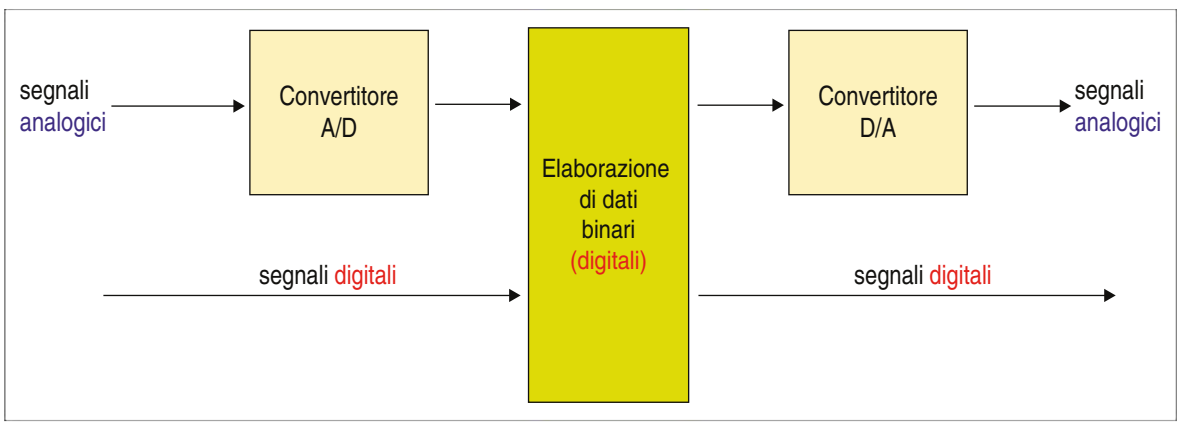
Pensiamo all'orologio digitale in contrapposizione con l'orologio analogico: su un display vengono visualizzate le cifre, che sono numeri interi, mentre nell'orologio analogico abbiamo la lancetta che si muove.

DIGITALE

Il segnale è detto digitale quando i valori utili che rappresentano una grandezza fisica sono discreti (finiti).

Il passaggio da analogico a digitale è chiamato digitalizzazione: vediamo un semplice schema che individua i blocchi fondamentali che permettono a un elaboratore digitale (per esempio un personal computer) di trattare i dati analogici.
Possiamo individuare tre elementi:

- convertitore A/D: dispositivo di interfacciamento che effettua la conversione analogico/digitale;
- unità di elaborazione: sistema a microprocessore (personal computer);
- convertitore D/A: dispositivo di interfacciamento che effettua la conversione digitale/analogico.



Un segnale digitale può invece essere direttamente elaborato da un personal computer in quanto i componenti elettronici del calcolatore sono stati progettati per trattare i segnali digitali codificati mediante il sistema binario, cioè sequenze di simboli 0 e 1.

Perché il digitale?

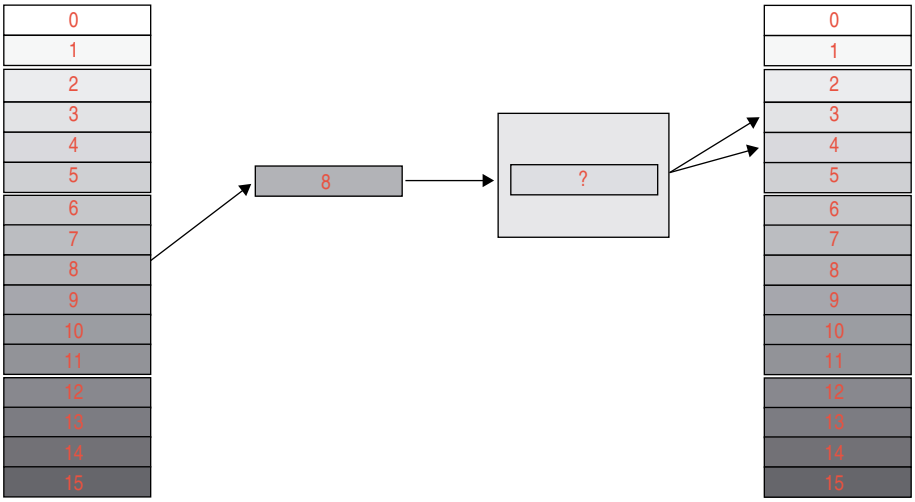
Sono molteplici le motivazioni per cui l'elaborazione digitale viene preferita all'elaborazione analogica. Riportiamo le più importanti di seguito:

- è possibile elaborare i dati con il computer, in quanto essi sono rappresentati in formato numerico;
- è possibile integrare diverse fonti, quali musica, parole, immagini, in un unico supporto come il CD-ROM;
- si possono condividere le informazioni attraverso la rete Internet.

I dispositivi per elaborare informazioni basate su un alfabeto binario sono semplici ed è sempre possibile aumentare l'accuratezza della rappresentazione del segnale incrementando il numero di elementi usati per rappresentare l'informazione; inoltre si possono costruire sistemi estremamente complessi di dimensioni ridotte, veloci e poco costosi.
Il sistema digitale è semplice e poco costoso quando occorre memorizzare l'informazione ed è inoltre particolarmente insensibile ai disturbi.

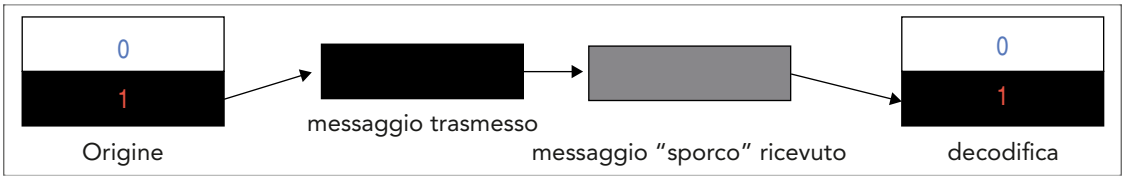
ESEMPIO

Vediamo con un esempio la capacità di un sistema digitale di essere poco sensibile ai disturbi. Si supponga di dover comunicare mediante un insieme di 16 simboli codificati con altrettante sfumature del nero, cioè con 16 diverse gradazioni di grigio, per coprire la gamma che va dal bianco al nero, disponendo per esempio due alunni ai vertici estremi di un'aula e dotando l'alunno trasmettitore di 16 palette colorate con 16 colori: il trasmettitore se deve comunicare il valore 7 alza la corrispondente paletta colorata. Supponendo per semplicità di essere in perfette condizioni di illuminazione, cioè di non avere elementi "ambientali" esterni di disturbo, chiediamo al secondo alunno, il "ricevitore", di indicarci il valore segnalato. Anche in condizioni perfette sarà sicuramente in difficoltà sia per individuare il giusto colore sia per classificarlo correttamente in quanto la poca differenza tra due colori adiacenti porta facilmente a un'errata interpretazione.

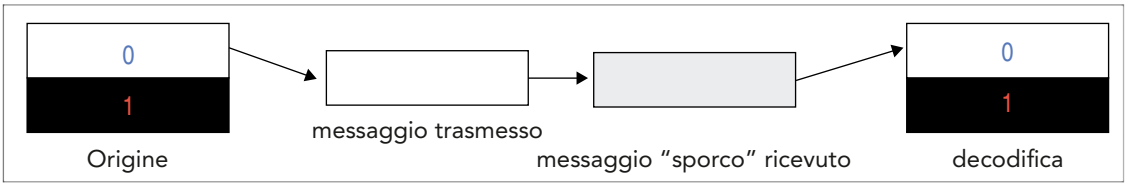


Lo stesso problema può essere risolto con un sistema digitale, dove i segnali hanno per esempio solo due possibili valori (binari). Se viene trasmesso per esempio il nero e per un qualunque motivo il segnale si "sporca" prima di essere ricevuto, abbiamo un margine mol-

to ampio per poterlo interpretare correttamente in quanto le due possibili configurazioni sono molto distanti tra loro: quindi nel caso pervenisse un messaggio alterato in diverse varianti di grigio si potrebbe tranquillamente considerare nero.



Analogo è il discorso in caso di un segnale bianco inviato: è facilmente interpretabile anche in presenza di un disturbo.



Digitale o binario?

Abbiamo detto che un calcolatore elettronico è costituito da circuiti digitali che realizzano operazioni tra segnali elettrici (di tensione e corrente) che assumono solo due valori (livello alto e livello basso): i segnali possono ammettere solo due stati distinti e perciò è necessario che nel sistema di numerazione e codifica utilizzato siano presenti solo due simboli.

Le informazioni che devono essere "trattate" possono essere suddivise in tre gruppi:
- dati alfabetici (o alfanumerici, cioè simboli lessografici);
- dati numerici (o numeri);
- dati multimediali (immagini, suoni e filmati).

È quindi necessario rappresentare ciascuno di essi mediante un meccanismo di codifica specifico per ogni tipologia di dato: in questa sede ci occuperemo dei primi due gruppi.

CODIFICA

Con il termine codifica intendiamo il processo che assegna un codice univoco a tutti gli oggetti di un insieme predefinito utilizzando i simboli dell'alfabeto.

I simboli 0 e 1 prendono il nome di *bit*, una contrazione di *binary digit*: spesso si usano *parole* di 4 bit (nibble) o di 8 bit (byte):
- attraverso 1 bit è possibile rappresentare due informazioni (una con 0 e l'altra con 1):
  – 1 bit → $2 = 2^1$ informazioni (0, 1);
- utilizzando una sequenza di 2 bit possiamo avere quattro codifiche distinte:
  – 2 bit → $4 = 2^2$ informazioni, ottenuti dalle combinazioni di più 0 e 1 (00, 01, 10, 11);

- utilizzando 3 bit possiamo rappresentare otto informazioni differenti:
  – 3 bit→ 8 = 2^3 informazioni (000, 001, 010, 011, 100, 101, 110, 111).

In generale, con n bit abbiamo la possibilità di avere 2^n informazioni (codifiche) differenti.

Il motivo per cui il sistema binario ha avuto tanta importanza nei sistemi di elaborazione è dovuto al contributo di George Boole, il quale dimostra come la logica possa essere ridotta a un sistema algebrico molto semplice, che utilizza solo un codice binario (zero e uno, vero e falso). Il codice binario è stato particolarmente utile nella teoria della commutazione per descrivere il comportamento dei circuiti digitali (1 = acceso, 0 = spento). Claude Shannon definì un metodo per rappresentare un qualsiasi circuito costituito da una combinazione di interruttori (e/o relais) mediante un insieme di espressioni matematiche, basate sulle regole dell'algebra booleana.

Bisogna prestare attenzione a non confondere digitale con binario, anche se nella vita quotidiana sono utilizzati come sinonimi:

- con digitale indichiamo un sistema discreto, cioè descritto da valori non continui;
- con binario si intende una codifica che utilizza un alfabeto di rappresentazione di due simboli.

Il motivo per cui digitale e binario vengono utilizzati allo stesso scopo è dovuto al fatto che nei personal computer i valori memorizzati sono digitali e sono rappresentati con codici binari: quindi, per esempio, si è soliti dire erroneamente che "in un hard disk abbiamo memorizzato dei numeri digitali", mentre l'espressione corretta è "abbiamo memorizzato in codice binario dei valori digitali".

Codifica in bit o binaria

Abbiamo detto che il dato a cardinalità minima è quello di valore 2 (per un dato di cardinalità 1 non ho scelta e dunque non ho informazione!), cioè il bit.

Se utilizzo il bit per codificare l'informazione ottengo la codifica binaria, che è la più "semplice" possibile, dove l'informazione è rappresentata da stringhe costruite con i simboli 0 e 1.

L'utilizzo di un sistema binario trova motivazioni di carattere tecnologico:

- due sono gli stati di carica elettrica di una sostanza;
- due sono gli stati di polarizzazione di una sostanza magnetizzabile.

Nella trasmissione dei segnali i due stati possono essere rappresentati con:

- passaggio/non passaggio di corrente in un conduttore;
- passaggio/non passaggio di luce in un cavo ottico.

In sostanza, è "comodo" rappresentare il "mondo" in binario.

MISURA

Il numero di bit necessari a codificare un dato di cardinalità N, cioè la quantità m = {[$\log_2$ N]}, è detta misura della quantità di informazione contenuta nel dato.

ESEMPIO

È possibile rappresentare qualunque informazione utilizzando due soli valori: vediamo come rappresentare un mazzo di carte da scopa.
Ogni carta ha un valore e un seme: il valore è compreso tra 1 e 10 mentre il seme può essere {fiori, quadri, cuori, picche}.

I numeri da 1 a 10 li codifichiamo con 4 bit, dato che $2^3 < 10 < 2^4$ ($n = \text{INT}_{max}(\log_2 10) = 4$)

1	2	3	4	5	6	7	J	Q	K
0001	0010	0011	0100	0101	0110	0111	1000	1001	1010

ai quali aggiungiamo altri 2 bit per codificare ciascuno dei quattro semi come indicato nella tabella a lato.

♥	00
♦	01
♣	10
♠	11

Quindi:
- il 7 di quadri viene codificato con 011101;
- il 4 di picche viene codificato con 010011;
- ecc.

Rappresentazione dei dati alfabetici

Con dati alfabetici intendiamo i 26 caratteri dell'alfabeto anglosassone (moltiplicato per due poiché sono necessarie sia le maiuscole sia le minuscole), le dieci cifre numeriche, le parentesi e gli operatori, oltre a un insieme di caratteri particolari composto dalla punteggiatura, dalle lettere accentate ecc.:

{a,b,c, A,B,C, %, &, (,), 0,1,2,3, ., ;, ?, +, -, *, ...}

Tutti questi elementi possono essere codificati usando 7 bit (27 = 128).

Il metodo di codifica più diffuso tra i produttori di hardware e di software prende il nome di codice ASCII (American Standard Code for Information Interchange).
Sebbene 7 bit siano sufficienti per codificare l'insieme di caratteri di uso comune, il codice ASCII standard utilizza 8 bit, il primo dei quali è sempre 0.

> Codici ASCII con 1 iniziale sono utilizzati per codificare caratteri speciali, ma la codifica non è standard: i byte fino al 256 costituiscono la tabella ASCII estesa che presenta varie versioni a carattere nazionale.

Nella tabella ASCII standard si trovano le cifre numeriche, le lettere maiuscole e minuscole (maiuscole e minuscole hanno codici ASCII differenti), la punteggiatura, i simboli aritmetici e altri simboli ($, &, %, @, # ecc.). Essendo stata concepita in America, la tabella ASCII standard non comprende le lettere accentate (sconosciute all'ortografia in-

glese). I primi 32 byte della tabella standard sono inoltre riservati per segnali di controllo e funzioni varie.

Byte	Cod.	Char
00000000	0	Nul
00000001	1	Start of heading
00000010	2	Start of text
00000011	3	End of text
00000100	4	End of transmit
00000101	5	Enquiry
00000110	6	Acknowledge
00000111	7	Audible bell
00001000	8	Backspace
00001001	9	Horizontal tab
00001010	10	Line feed
00001011	11	Vertical tab
00001100	12	Form Feed
00001101	13	Carriage return
00001110	14	Shift out
00001111	15	Shift in
00010000	16	Data link escape
00010001	17	Device control 1
00010010	18	Device control 2
00010011	19	Device control 3
00010100	20	Device control 4
00010101	21	Neg. acknowledge
00010110	22	Synchronous idle
00010111	23	End trans. block
00011000	24	Cancel
00011001	25	End of medium
00011010	26	Substitution
00011011	27	Escape
00011100	28	File separator
00011101	29	Group separator
00011110	30	Record Separator
00011111	31	Unit separator

Byte	Cod.	Char
00100000	32	Spc
00100001	33	!
00100010	34	"
00100011	35	#
00100100	36	$
00100101	37	%
00100110	38	&
00100111	39	'
00101000	40	(
00101001	41	)
00101010	42	*
00101011	43	+
00101100	44	,
00101101	45	-
00101110	46	.
00101111	47	/
00110000	48	0
00110001	49	1
00110010	50	2
00110011	51	3
00110100	52	4
00110101	53	5
00110110	54	6
00110111	55	7
00111000	56	8
00111001	57	9
00111010	58	:
00111011	59	;
00111100	60	<
00111101	61	=
00111110	62	>
00111111	63	?

Byte	Cod.	Char
01000000	64	@
01000001	65	A
01000010	66	B
01000011	67	C
01000100	68	D
01000101	69	E
01000110	70	F
01000111	71	G
01001000	72	H
01001001	73	I
01001010	74	J
01001011	75	K
01001100	76	L
01001101	77	M
01001110	78	N
01001111	79	O
01010000	80	P
01010001	81	Q
01010010	82	R
01010011	83	S
01010100	84	T
01010101	85	U
01010110	86	V
01010111	87	W
01011000	88	X
01011001	89	Y
01011010	90	Z
01011011	91	[
01011100	92	\
01011101	93	]
01011110	94	^
01011111	95	_

Byte	Cod.	Char
01100000	96	`
01100001	97	a
01100010	98	b
01100011	99	c
01100100	100	d
01100101	101	e
01100110	102	f
01100111	103	g
01101000	104	h
01101001	105	i
01101010	106	j
01101011	107	k
01101100	108	l
01101101	109	m
01101110	110	n
01101111	111	o
01110000	112	p
01110001	113	q
01110010	114	r
01110011	115	s
01110100	116	t
01110101	117	u
01110110	118	v
01110111	119	w
01111000	120	x
01111001	121	y
01111010	122	z
01111011	123	{
01111100	124	|
01111101	125	}
01111110	126	~
01111111	127	Del

Per esempio, codifichiamo le parole gatto rosso:

01100111	01100001	01110100	01110100	01101111
g	a	t	t	o
01110010	01101111	01110011	01110011	01101111
r	o	s	s	o

Proviamo ora a effettuare la decodifica di una stringa espressa in cifre binarie. È necessario effettuare due passi successivi:

1. per prima cosa suddividere la sequenza in gruppi di otto bit (un byte);
2. successivamente determinare il carattere corrispondente a ogni byte.

Per esempio, la seguente stringa

01100111011010010110000101101111100101110

viene scomposta in byte

01100111011010010110000101101111 00101110

e decodificata in

c i a o .

La decodifica è possibile (e facile) perché i caratteri sono codificati con stringhe binarie di lunghezza costante.

Esistono altri formati per la rappresentazione dei caratteri dell'alfabeto all'interno dei sistemi digitali. Tra questi ricordiamo ECBDIC e Unicode.

ECBDIC

Il codice EBCDIC (Extended Binary-Coded Decimal Interchange Code) è un codice alfanumerico a 8 bit ideato dall'IBM e ancora utilizzato su grandi macchine IBM, per cui ha una certa diffusione sebbene la maggior parte delle macchine presenti in rete utilizzi il codice ASCII.

Unicode

È il nuovo standard utilizzato in Internet: il World Wide Web è un oggetto globale, ma l'ASCII non può rendere tutti i caratteri accentati delle lingue europee e men che meno quelli dell'arabo, del bengalese, dell'ebraico, del thailandese...
Per questi motivi si è passati prima all'ASCII esteso di Latin-1 poi a Unicode 16 bit (2 byte) con 2^{16} possibili codici, ovvero 65.536 possibilità (sono stati a oggi definiti solo 40.000 codici Unicode, dei quali la metà viene usata per gli ideografi Han e 11.000 sono utilizzati per le sillabe coreane Hangul).
È importante sottolineare che i codici Unicode da 0 a 255 corrispondono ai codici ASCII standard e quindi c'è compatibilità tra ASCII e Unicode.

■ Prefissi binari per il byte

Apriamo una parentesi sulla definizione dell'unità di misura della capacità di memoria, cioè il byte (contrazione di *binary ochtette*) che, come detto più volte, è il raggruppamento di 8 cifre digitali (8 bit).

Il simbolo utilizzato per il byte come unità di misura della quantità di informazione è "B"; nonostante la lettera maiuscola sia riservata alle sole unità di misura tratte dai cognomi degli ideatori, l'IEC (*International Electrotechnical Commission*) ha deciso di fare un'eccezione, dato che "b" è generalmente usato per indicare il bit (il cui simbolo standard sarebbe "bit" per esteso).

I multipli comunemente usati sono Kbyte, Mbyte, Gbyte e i Tbyte (con sigle KB, MB, GB e TB): si è detto che 1 Kbyte è uguale a 2^{10} byte (1024 byte), 1 Mbyte corrisponde a 2^{20} byte e via di seguito, ma in queste affermazioni si sono commesse alcune inesattezze.
I prefissi mega, giga e tera sono quelli definiti nel Sistema Internazionale (ufficialmente chiamato *Système International d'Unités* e abbreviato in SI) che è il più diffuso tra i sistemi di unità di misura; assieme al Sistema CGS, oggi non più ufficiale, viene spesso indicato come sistema metrico, soprattutto nei paesi anglosassoni.

Come si vede dalla tabella che segue, i multipli del byte vengono generalmente calcolati come potenze del 2 partendo dall'osservazione che si sta operando in un sistema binario e non decimale. Il ragionamento è giusto ma adottare i simboli del SI è *formalmente sbagliato*; tale ambiguità ha portato l'IEC a definire nuovi prefissi per i multipli binari, chiamati prefissi binari, che a tutt'oggi sono, però, poco conosciuti (e pubblicizzati), forse anche perché ormai l'utilizzo dei prefissi del SI è piuttosto radicato.
Alcuni produttori di memorie magnetiche sfruttano questa ambiguità per effettuare operazioni poco corrette: se in un dispositivo di piccole dimensioni la differenza di capacità è minima (per esempio, in una memoria da 1 mega e 44 KB abbiamo 1.509.950 byte contro 1.440.000 byte), all'aumentare delle dimensioni indicare i gigabyte al posto dei gibibyte porta a differenze del 7% circa, che salgono a oltre il 10% nel caso dei terabyte, come si può vedere dalla seguente tabella riepilogativa.

Prefissi binari					Prefissi SI			
Quantità		Fattore	Nome corretto	Sigla	Fattore	Nome usato comunemente	Sigla	Errore
1 B	=	2^0 B	byte	B	10^0 B	byte	B	0
1024 B	=	2^{10} B	Kibibyte	KiB	10^3 B	Kilobyte	KB	+2,4
1024 KiB	=	2^{20} B	Mebibyte	MiB	10^6 B	Megabyte	MB	+4,9
1024 MiB	=	2^{30} B	Gibibyte	GiB	10^9 B	Gigabyte	GB	+7,4
1024 GiB	=	2^{40} B	Tebibyte	TiB	10^{12} B	Terabyte	TB	+10,0
1024 TiB	=	2^{50} B	Pebibyte	PiB	10^{15} B	Petabyte	PB	+12,6
1024 PiB	=	2^{60} B	Exbibyte	EiB	10^{18} B	Exabyte	EB	+15,3

Per esempio, un hard disk da 10 TB nominali potrà contenere effettivamente circa solo 9 TiB.

Verifichiamo le conoscenze

Esercizi a scelta multipla

1 **Quale tra queste grandezze non è analogica?**

- ○ musica
- ○ luce
- ○ secondi
- ○ temperatura
- ○ velocità

2 **Un segnale analogico:**

- ○ è continuo nel tempo
- ○ assume solo due valori
- ○ assume solo un insieme finito di valori
- ○ si codifica con un campionatore

3 **Un segnale digitale:**

- ○ è continuo nel tempo
- ○ assume solo due valori
- ○ assume solo un insieme finito di valori
- ○ si rappresenta con le dita

4 **Con 5 bit è possibile avere:**

- ○ 10 combinazioni diverse
- ○ 16 combinazioni diverse
- ○ 32 combinazioni diverse
- ○ 64 combinazioni diverse

5 **Il codice ASCII è l'acronimo di:**

- ○ *Automatic Standard Code for Information Interchange*
- ○ *Australian Standard Code for Informatic Interchange*
- ○ *Automatic Standard Code for Informatic Interchange*
- ○ *American Standard Code for Information Interchange*

6 **Indica le corrispondenze tra prefisso e quantità di bit:**

- ○ KiB _____ B
- ○ KB _____ B
- ○ MiB _____ KB
- ○ MB _____ KB
- ○ GiB _____ MB
- ○ GB _____ MB

Test vero/falso

1	Un segnale è una grandezza fisica che varia nel tempo.	V F
2	Un segnale analogico può assumere un insieme finito di valori.	V F
3	Un segnale tempo-discreto è un segnale il cui valore ha interesse solo in istanti di tempo.	V F
4	Il segnale è detto digitale quando i valori utili che lo rappresentano sono finiti.	V F

5 Un segnale analogico può sempre essere trasformato in un segnale digitale. V F
6 La "discretizzazione del segnale" fa passare da un insieme infinito di valori a un insieme discreto. V F
7 Tipicamente i segnali tempo-discreti hanno gli istanti in cui vengono campionati finiti. V F
8 Tipicamente i segnali tempo-discreti hanno gli istanti in cui vengono campionati equidistanziati. V F
9 Con il termine nibble si intendono byte di 4 bit. V F
10 La codifica Unicode utilizza 8 bit per rappresentare i caratteri. V F

Verifichiamo le competenze — *Esprimi la tua creatività*

1 Quanti bit sono necessari per codificare i giorni della settimana?
E i giorni del mese?
E i giorni di un anno?

2 Codifica in binario i sette nani e Biancaneve con due diverse configurazioni di bit.

3 Codifica in binario i tuoi compagni di classe, separando i maschi dalle femmine, e assegnando 0 = maschio e 1 = femmina come primo bit (bit più significativo).

4 Codifica in binario le carte di un mazzo da scala quaranta sapendo che metà mazzo ha il dorso rosso e metà il dorso nero. Quindi codifica anche i due jolly.

5 Codifica in binario i giorni più significativi della tua vita, indicando (000…000) il giorno della tua nascita: per esempio codifica il tuo primo giorno di scuola elementare, di scuola media e di scuola superiore.

6 Codifica in binario utilizzando la tabella dei codici ASCII le seguenti parole:
Mario
Cammello
Zio Pino
Abracadabra

7 Decodifica utilizzando la tabella dei codici ASCII le seguenti stringhe binarie:
01101101011000010110110101101101011000 01
01010000011000010110111101101100011011 11
01000001011101010110011101110101011100 1001101001
01000111011011000110111101110010011010 0101100001

8 Esegui le seguenti equivalenze:

1,4 KB = _____ KiB		980 KB = _____ MiB	
6,7 KB = _____ KiB		1900 KB = _____ MiB	
3,8 MB = _____ KiB		8,8 TB = _____ TiB	
8,2 MB = _____ KiB		39 TB = _____ TiB	

UNITÀ DIDATTICA 3

SISTEMI DI NUMERAZIONE POSIZIONALI

IN QUESTA UNITÀ IMPAREREMO...

- l'origine dei sistemi di numerazione posizionale
- i sistemi additivi/sottrattivi
- a rappresentare i numeri nelle diverse basi
- a convertire un numero in base decimale

Rappresentazione dei dati numerici

Un primo problema che affrontiamo è quello di rappresentare i numeri prestando attenzione a non confondere il *significato* con la sua *rappresentazione*.

I numeri sono entità matematiche astratte e vanno distinti dalla loro rappresentazione.

NUMERO E NUMERALE

Si dice numero un'entità astratta.
Il numerale è una stringa di caratteri (codifica) che rappresenta un numero in un dato sistema di numerazione.

Vediamo per esempio la rappresentazione del numero 10.

10 dieci nella numerazione araba

X dieci nella numerazione romana

 dieci nella numerazione unaria

 dieci nella numerazione maya

 dieci nella numerazione babilonese

Abbiamo cinque numerali che rappresentano lo stesso numero: lo stesso numero è quindi rappresentato da numerali diversi in diversi sistemi di numerazione.

La differenziazione tra ciò che dobbiamo codificare e come viene codificato deve essere sempre ben presente in ogni tipo di informazione, non necessariamente numerica; quando abbiamo a che fare in generale con l'informazione distinguiamo sempre:
- contenuto: il messaggio ⇒ significato;
- modalità espressiva: la sua codifica ⇒ significante;
- supporto materiale: l'elemento fisico su cui (o mediante il quale) viene resa disponibile l'informazione; può essere cartaceo, magnetico, elettronico ecc.

A ogni significato corrisponde un significante e viceversa: il significante è il mezzo fonico o grafico che veicola il significato della parola in questione.

Sistemi di numerazione

Dopo aver visto come è possibile rappresentare i numeri, vediamo come questi numeri sono tra loro connessi per poterli in seguito utilizzare mediante alcune operazioni.

SISTEMA DI NUMERAZIONE (1)

Definiamo con sistema di numerazione un sistema utilizzato per esprimere i numeri e le operazioni che si possono effettuare su di essi.

NOTAZIONI NUMERALI

Fin dai tempi antichi i numeri si sono rivelati strumenti necessari per affrontare problemi di importanza fondamentale (come contare, misurare, commerciare, amministrare, formulare e far rispettare leggi, sviluppare conoscenze scientifiche e tecniche ecc.).
Tutte le culture delle quali conosciamo qualche forma di organizzazione hanno sviluppato notazioni numerali.

La storia di questi sviluppi è piuttosto complessa e, purtroppo, si sono perse le tracce delle vicende che li riguardano. La storia, però, fornisce indicazioni che possono essere di grande interesse, in quanto sono collegate a temi culturali e conoscitivi di grande rilevanza.

Una seconda definizione di sistema di numerazione è la seguente:

SISTEMA DI NUMERAZIONE (2)

Un sistema di numerazione è un insieme di regole e di simboli il cui utilizzo permette di rappresentare delle quantità.

Sostanzialmente i sistemi di numerazione sono di due tipi:
- additivo/sottrattivo;
- posizionale.

Sistema additivo/sottrattivo

SISTEMA ADDITIVO/SOTTRATTIVO

Un sistema di numerazione è di tipo additivo/sottrattivo se a ogni simbolo è associato un valore e il numero rappresentato è dato dalla somma o dalla differenza dei valori dei simboli che vengono accostati tra loro.

Un numero viene indicato accostando una serie di cifre fino a quando la somma dei numeri corrispondenti alle diverse cifre è pari al numero che si vuole rappresentare.
Un primo esempio di questo sistema è il sistema numerale romano.
I numeri romani sono stringhe, cioè sequenze, costituite dai simboli riportati nella tabella a lato:

I	1
V	5
X	10
L	50
C	100
D	500
M	1000

Se per esempio il simbolo I viene posizionato a sinistra del simbolo V deve essere sottratto, mentre se viene posizionato a destra deve essere addizionato al valore che lo precede.

IV equivale a V – I, quindi a 5 – 1 = 4
VI equivale a V + I, quindi a 5 + 1 = 6

La data della scoperta dell'America in numeri romani è la seguente:

MCDXCII = 1000 – 100 + 500 – 10 + 100 + 1 + 1 = 1492

Sulla lapide rappresentata nella figura a lato, fatta apporre da Benedetto XVI sul luogo dell'attentato a Giovanni Paolo II, la data del 13 maggio 1981 è incisa in numeri romani:

XIII = 10 + 1 + 1 + 1
V = 5
MCMLXXXI = 1000 – 100 + 1000 + 50 + 10 + 10 + 10 + 1

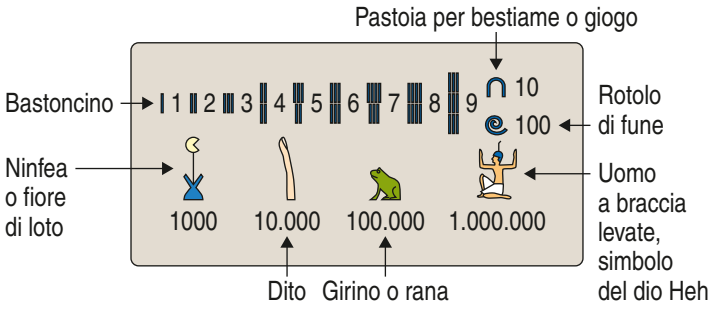
Zoom su...

SISTEMA NUMERALE ROMANO

In verità questo sistema non è proprio quello utilizzato nell'antica Roma ma è la sua modifica effettuata nel Medioevo: il sistema originale era "additivo" nel vero senso della parola, cioè i valori dei simboli venivano sempre addizionati, mai "sottratti". Era accettata cioè la ripetizione di un simbolo anche per quattro volte, così che il numero 9 si scriveva VIIII invece che IX come lo conosciamo oggi.

Sembra che la motivazione che ha portato al passaggio verso il sistema sottrattivo sia stata la difficoltà di incidere i numeri nelle epigrafi, in quanto la notazione originale risultava troppo lunga nel descrivere alcuni numeri.

Un secondo esempio di questo tipo di sistema è il sistema numerale egizio: esso impiegava simboli diversi (geroglifici) per rappresentare le diverse potenze del 10, da 1 a 10^6, come si può vedere dalla figura a destra:



Per esempio il geroglifico disegnato a lato

rappresenta il numero 3244, mentre quest'altro

rappresenta il numero 21.237.

Nei sistemi puramente additivi la posizione delle diverse cifre all'interno del numero rappresentato non è importante: per esempio il numero @@III potrebbe essere scritto anche come I@I@I.

Anche la numerazione attica (o *erodianica*) adottava un sistema puramente additivo ed esisteva un numero limitato di simboli di valore costante. Il numero 1 era rappresentato con un trattino verticale, ripetuto fino a quattro volte per rappresentare, appunto, i numeri da 1 a 4. A questo simbolo se ne aggiungevano altri adatti a rappresentare il 10, il 100, il 1000 e il 10.000.

Con questi sistemi di numerazione risulta abbastanza complesso eseguire le operazioni, anche le più semplici come l'addizione e la sottrazione. Inoltre è doveroso osservare che in essi non è presente il numero 0.

Sistema posizionale

Nei sistemi posizionali la posizione delle diverse cifre del numero è fondamentale: in essi viene scelta una base, ossia un numero naturale, e viene definita una serie di cifre che indicano tutti i numeri naturali più piccoli della base, compreso lo zero.
Tutti gli altri numeri vengono espressi in funzione di potenze della base.

Il sistema di numerazione che utilizziamo oggi è il sistema numerico decimale-posizionale (chiamato a torto sistema numerale arabo in quanto deriva dagli indiani), introdotto in Europa verso l'XI secolo, che è un sistema in base 10 (decimale).
I nostri numeri infatti vengono scritti utilizzando dieci cifre, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, che indicano tutti i numeri naturali più piccoli di 10: tutti gli altri numeri vengono espressi in funzione delle potenze della base, ossia in questo caso delle potenze di 10.

SISTEMA POSIZIONALE

Un sistema si chiama posizionale se una stessa cifra ha un valore diverso (peso) a seconda della posizione: la cifra all'estrema destra prende il nome di *cifra meno significativa* mentre la cifra all'estrema sinistra è detta *cifra più significativa*.

Le ragioni della superiorità di tale sistema di numerazione che, come abbiamo detto, si è diffuso in Europa dall'India, sono il principio posizionale (che di per sé denota i diversi ordini numerici) e l'uso di dieci simboli, comprensivi dello zero (gli arabi chiamavano lo zero *sifr*, che significa "vuoto", e proprio da esso deriva il termine *cifra*).

PRINCIPIO DI POSIZIONE

Già babilonesi, cinesi e maya furono capaci con il *principio di posizione* di rappresentare qualsiasi numero con una quantità ridotta di cifre di base, ma furono limitati nella rappresentazione di numeri e operazioni.

Il numero dei simboli che costituiscono l'alfabeto è pari al valore della base: la base e la posizione della cifra indicano il fattore moltiplicante (peso) di ogni cifra presente nel numero. Quindi ogni cifra assume un valore diverso a seconda della posizione che assume all'interno del numero, a differenza delle altre notazioni non posizionali che dovevano dare a ogni cifra un valore fisso a prescindere dalle posizioni.

Il numero di cifre necessario a rappresentare un numero dipende dalla base: più grande è la base, più corta è la codifica: il sistema binario, come sappiamo, è quello che utilizza il maggior numero di cifre per rappresentare i numeri dato che impiega il minimo numero di simboli.

Riassumendo, un sistema posizionale è un sistema naturale in cui vi sono quattro elementi fondamentali:

- una base fissa, che è il numero di cifre utilizzato in un sistema posizionale;
- l'ordine di una cifra, che è la posizione della cifra nell'intera rappresentazione numerica, contando, a partire da zero, dall'ultima cifra di destra verso sinistra;
- l'insieme dei simboli, che sono in numero pari al valore della base;
- il valore associato a una cifra, che si ottiene moltiplicando tale cifra per la base b elevato all'esponente dato dall'ordine della cifra stessa (cifra moltiplicata per il peso).

Il sistema decimale-posizionale consente anche una comoda esecuzione di operazioni aritmetiche: si mettono i numeri da sommare uno sotto l'altro e li si può addizionare colonna per colonna riportando i totali eccedenti il 10 (riporto) nella colonna a fianco (ordine superiore).

Definizioni e terminologia

Dato che utilizzeremo i sistemi posizionali per la rappresentazione dei numeri, riassumiamo le definizioni e la terminologia che adotteremo.

Indichiamo con N_b un numero naturale rappresentato in base b.

Per esempio:
- N_2 = 101 indica il numero 101 in base 2, cioè 101 è un numero binario;
- N_8 = 712 indica il numero 712 in base 8, cioè 712 è un numero ottale;
- N_{16} = 7B indica il numero 7B in base 16, cioè 7B è un numero esadecimale (generalmente viene indicato con pedice H).

Se non viene indicata alcuna base, cioè genericamente N, si intende base 10, cioè decimale.

Analogamente a quanto appena detto, una sequenza di cifre senza pedice rappresenta un numero naturale espresso in base 10 mentre una sequenza di cifre con pedice *b* rappresenta un numero espresso in base diversa da 10:
- 101_2 è binario;
- 712_8 è ottale;
- $7B_H$ è esadecimale;
- 8239 è decimale.

Indichiamo con c la generica cifra all'interno della stringa che rappresenta il numero e con il pedice m la posizione m-esima della cifra all'interno del numero, che è anche l'esponente da dare alla base per ricavare il peso alla singola cifra: per esempio la cifra c_{m-1} è quella più significativa all'interno del numero mentre la cifra meno significativa è c_0.

Utilizzando questa notazione, un generico numero è scritto simbolicamente con

$$N_b = c_{m-1}c_{m-2} \ldots c_2c_1c_0$$

Per ottenere da questo il corrispondente valore decimale sostituiamo a ogni termine il prodotto della cifra con la notazione

$$N_{10} = c_{m-1} \cdot b^{m-1} + c_{m-2} \cdot b^{m-2} \ldots c_2 \cdot b^2 + c_1 \cdot b^1 + c_0b^0$$

Zoom su...

NOTAZIONE COMPATTA

Il valore decimale di un generico numero rappresentato in base *b* in un sistema posizionale può essere scritto in forma più compattata con il simbolo di sommatoria:

$$N_{10} = \sum_{i=0}^{m-1} c_i \cdot b^i$$

e viene utilizzato per dare la definizione che segue.

SISTEMA NUMERICO POSIZIONALE

Un sistema numerico *posizionale* in base *b*, ovvero basato su un alfabeto Σ di *b* simboli distinti, consente di esprimere un qualsiasi numero naturale *N* di m cifre, mediante la seguente espressione:

$$N = \sum_{i=0}^{m-1} c_i \cdot b^i$$

Questa formula stabilisce una corrispondenza biunivoca tra i numeri naturali e la loro codifica in base *b*: ciascun numero naturale ha uno e un solo codice in base *b* e a ciascun codice in base *b* corrisponde uno e un solo numero naturale.
Qualunque sia la base *b* la formula costituisce anche un *metodo di conversione di un numero dalla base b in decimale*, dove ciascuna cifra del numerale rappresenta il coefficiente di una potenza della base e l'esponente è dato dalla posizione della cifra.

ESEMPIO

Nel sistema decimale un numero di 4 cifre ha l'espressione seguente:

$$b = \sum_{i=0}^{3} c_i \cdot 10^i$$

quindi il numero $N_{10} = 7354_{10}$ si scrive come la somma di quattro termini

$$7354_{10} = 7 \cdot 10^3 + 3 \cdot 10^2 + 5 \cdot 10^1 + 4 \cdot 10^0$$

e come risultato decimale dà, naturalmente, il numero stesso.

Possiamo vedere ogni singolo termine nel dettaglio:

moltiplicatore	7	3	5	4
posizione	4	3	2	1
potenze/peso	10^3	10^2	10^1	10^0
	1000	**100**	**10**	**1**

Nel sistema binario un numero di 4 cifre ha l'espressione seguente:

$$b = \sum_{i=0}^{3} c_i \cdot 2^i$$

quindi il numero $N_2 = 1001_2$ si scrive come la somma di quattro termini:

$$1001_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 1 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = 9_{10}$$

2^3 2^2 2^1 2^0 BASE$_2$ — BASE$_{10}$

e come risultato decimale dà il numero decimale 9.

Possiamo vedere ogni singolo termine nel dettaglio:

moltiplicatore	1	0	0	1
posizione	4	3	2	1
potenze/peso	2^3	2^2	2^1	2^0
	8	**4**	**2**	**1**

Sistema unario

Il sistema unario è praticamente il sistema di numerazione per comparazione: è presente un unico simbolo (per esempio l'"I" oppure il "sasso", un "legnetto", i "punti sul dado" ecc.) e una sola regola di rappresentazione che dice di affiancare un simbolo vicino all'altro per ottenere tutti i numeri.
Per esempio IIIIIIIIIII = 12.

Il sistema unario non può essere considerato un sistema posizionale in quanto la cifra ha sempre lo stesso valore indipendentemente dalla posizione che assume, ma d'altra parte "funziona" anche se lo consideriamo un sistema posizionale, in quanto la regola di composizione del numero viene rispettata.
Verifichiamolo con un esempio:
IIII = 4 può essere visto come

$$1111_1 = 1 \cdot 1^3 + 1 \cdot 1^2 + 1 \cdot 1^1 + 1 \cdot 1^0 = 1 + 1 + 1 + 1 = 4_{10}$$

Quindi la numerazione unaria è anche una numerazione posizionale, dove però non è presente il numero 0: generalmente il numero 0 si indica con l'"assenza" di ogni cifra.

moltiplicatore	1	1	1	1
posizione	4	3	2	1
potenze/peso	1^3	1^2	1^1	1^0
	1	**1**	**1**	**1**

È evidente che il sistema unario non è un sistema "economico", e quindi non viene usato: tutti i codici numerici utilizzati sono posizionali e hanno necessariamente base $b > 1$.

La base 2 e il sistema binario

Abbiamo già parlato del sistema binario e, dato che è il sistema che utilizziamo per rappresentare i dati nei sistemi digitali, sarà approfonditamente trattato nelle prossime unità didattiche.

Vediamo solo un esempio di codifica: il numero 10101 può essere visto come

$$10101_2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 1^0 = 16 + 0 + 4 + 0 + 1 = 21_{10}$$

2^4 2^3 2^2 2^1 2^0 BASE$_2$ — BASE$_{10}$

Possiamo vedere ogni singolo termine nel dettaglio:

moltiplicatore	1	0	1	0	1
posizione	5	4	2	2	1
potenze/peso	2^4	2^3	2^2	2^1	2^0
	16	**8**	**4**	**2**	**1**

La base 5 e il sistema quinario

La mano è senz'altro il primo strumento di computazione e quindi la base 5 è stata forse la prima base utilizzata nella storia. Avendo cinque dita si inizia a contare da 1 a 5, quindi con cinque elementi (o simboli); per rappresentare il numero 6 si utilizza un dito della seconda mano, quindi un dito in aggiunta a una mano, e via di seguito.

Un esempio di base 5 è la lingua Api delle Nuove Ebridi:

1 = tai	6 = otai	= nuovo uno
2 = lua	7 = olua	= nuovo due
3 = tolu	8 = otolu	= nuovo tre
4 = vari	9 = ovari	= nuovo quattro
5 = luna	10 = lualuna	= due mani (2 × 5)

Il fatto che la base 5 in questo caso sia fondata sull'utilizzo computazionale della mano può essere desunto dalla denominazione del numero '6' (NUOVO UNO), del numero '5' (MANO) e del numero '10' (DUE MANI).

La base 5 oggi è presente anche in Africa, Oceania e nel sud dell'India.

Il numero $N_5 = 2374_5$ si scrive come la somma di quattro termini

$$2374_5 = 2 \cdot 5^3 + 3 \cdot 5^2 + 7 \cdot 5^1 + 4 \cdot 5^0 = 2 \cdot 125 + 3 \cdot 25 + 7 \cdot 5 + 4 \cdot 1 = 364_{10}$$

5^3 5^2 5^1 5^0 BASE$_5$ — BASE$_{10}$

e come risultato decimale dà il numero decimale 489.
Possiamo vedere ogni singolo termine nel dettaglio:

moltiplicatore	2	3	7	4
posizione	4	3	2	1
potenze/peso	5^3	5^2	5^1	5^0
	125	**25**	**5**	**1**

La base 8 e il sistema ottale

I sistemi di elaborazione utilizzano la notazione binaria ma, generalmente, risulta scomodo trattare lunghe stringhe di bit: diventa utile poter usare sistemi numerici che consentano di esprimere in maniera più compatta lunghe stringhe di 0 e 1.
Uno di questi sistemi è il sistema di numerazione ottale, quindi con b = 8, che utilizza le cifre $\Sigma = \{0,1,2,3,4,5,6,7\}$.

Ogni numero è costituito da una stringa di cifre ottali il cui valore è determinato dal prodotto della cifra per una potenza della base il cui esponente è dato dalla posizione della cifra nella stringa.

Per esempio:

- la stringa $N_8 = 354_8$ si scrive come la somma di tre termini:
  $354_8 = 3 \cdot 8^2 + 5 \cdot 8^1 + 4 \cdot 8^0 = 3 \cdot 64 + 40 + 4 = 192 + 40 + 4 = 236_{10}$
- la stringa $N_8 = 5712_8$ si scrive come la somma di quattro termini:
  $5712_8 = 5 \cdot 8^3 + 7 \cdot 8^2 + 1 \cdot 8^1 + 2 \cdot 8^0 = 5 \cdot 512 + 7 \cdot 64 + 8 + 2 = 2560 + 448 + 8 + 2 = 3018_{10}$

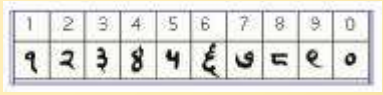
Possiamo vedere ogni singolo termine nel dettaglio:

moltiplicatore	5	7	1	2
posizione	4	3	2	1
potenze/peso	8^3	8^2	8^1	5^0
	512	**64**	**8**	**1**

La base 10 e il sistema decimale

Il sistema decimale viene da noi utilizzato per contare sin dai primi anni di scuola elementare, e quindi non entreremo nel dettaglio.

I dieci simboli hanno assunto forme diverse a seconda delle zone e dei periodi, ma quelli che usiamo oggi sono quelli che utilizzarono gli arabi e che, dalla forma araba, sono passati in Europa e hanno assunto la forma definitiva grazie alla diffusione della stampa nel XV secolo.



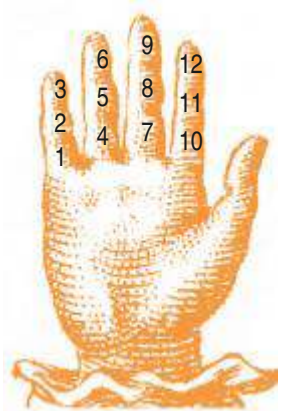
È doveroso fare un'osservazione sul nome che diamo ai numeri: per esempio, in italiano, il numero 4 corrisponde al nome quattro, il numero 6 al nome sei, ma se analizziamo i numeri a due cifre ci accorgiamo che questi sono costruiti in maniera additiva e non posizionale: per esempio 1274 viene chiamato milleduecentosettantaquattro e non uno-due-sette-quattro.

Il sistema a base 12

Il sistema numerico con il quale nell'antichità veniva misurato il cielo (e quindi il tempo) è quello a base 12 che ancora oggi utilizziamo nel calcolo dei mesi e delle ore.

L'origine della base 12 sta forse nel numero delle falangi (3 per ogni dito) computabili utilizzando il pollice come cursore (3 × 4 = 12).

Se pensiamo a ogni dito come a una delle quattro stagioni, le tre falangi individuano ciascuna un mese dell'anno.



Il sistema in base 12 era usato da sumeri e assiro-babilonesi come misura per lunghezze, superfici, volumi e capacità: la durata della giornata era suddivisa in 12 periodi detti danna di due ore ciascuno; a sua volta il cerchio, l'eclittica e lo zodiaco erano suddivisi da queste popolazioni in 12 beru (settori) di 30° ciascuno.

Anche per i romani l'unità di misura del peso era divisa in 12 once, che ritroviamo ancora oggi nei sistemi anglosassoni:
1 piede = 12 pollici
1 pollice = 12 linee
1 linea = 12 punti
1 libbra = 12 once

Nella numerazione in lingua inglese il suffisso "teen" è usato dal 13 in poi: questo presume che i valori da 1 a 12 siamo stati i numeri di base del sistema di numerazione.

Un'ulteriore motivazione che sta alla base dell'utilizzo della numerazione in base 12 è che essa ha un numero maggiore di divisori rispetto alla base 10: il numero 12 può essere diviso per 1, 2, 3, 4, 6 e 12 mentre il 10 solamente per 1, 2, 5 e 10; questo era utile soprattutto nell'uso monetario, e infatti la troviamo utilizzata anche nel Granducato di Toscana dove 1 lira = 12 crazie, oltre che in quella britannica, dove 1 scellino = 12 pence.

La base 16

Il sistema esadecimale ha b = 16 e utilizza generalmente un alfabeto di 16 cifre dove dopo la cifra 9 si "prosegue" con le prime 6 lettere maiuscole dell'alfabeto: Σ = {0,1,2,3,4,5,6,7,8,9,A, B,C,D,E,F}.

Per esempio:
- la stringa $N_{16} = 3B2_{12}$ si scrive come la somma di tre termini:
  $3B2_{16} = 3 \cdot 16^2 + B \cdot 16^1 + 2 \cdot 16^0 = 3 \cdot 256 + 11 \cdot 160 + 2 = 768 + 176 + 2 = 946_{10}$
- la stringa $N_{16} = 13FA_{16}$ si scrive come la somma di quattro termini:
  $13FA_{16} = 1 \cdot 16^3 + 3 \cdot 16^2 + F \cdot 16^1 + A \cdot 16^0 = 4096 + 3 \cdot 256 + 15 \cdot 16 + 10 \cdot 1 = 4096 + 768 + 240 + 10 = 5114_{10}$

Possiamo vedere ogni singolo termine nel dettaglio:

moltiplicatore	1	3	F = 15	A = 10
posizione	4	3	2	1
potenze/peso	16^3	16^2	16^1	16^0
	4096	**256**	**16**	**1**

Questo sistema di rappresentazione è particolarmente utilizzato nei calcolatori elettronici, in quanto, come vedremo in seguito, è strettamente collegato al sistema binario.

Generalmente i numeri esadecimali sono indicati con pedice H, cioè $N_{16} = 13FA_{16}$ viene indicato con $N_H = 13FA_H$.

La base 20

Il sistema in base 20 probabilmente è stata la naturale evoluzione del sistema in base 10 dove, oltre che al computo delle dita delle mani, si è passati a utilizzare anche quelle dei piedi per aumentarne la capacità di rappresentazione.

Il sistema a base 60 e il sistema sessagesimale

Un ultimo cenno merita il sistema a base 60 in quanto già utilizzato anticamente dagli assiri (ma anche da sumeri e altre civiltà mesopotamiche) e alla base della misurazione degli angoli (oggi misurata in 360°).
Sicuramente risulta difficile ricordare 60 simboli diversi e quindi probabilmente la base 60 veniva utilizzata combinandola a un'altra base, come per la misura del tempo in secondi.

Conclusioni

In sintesi, i principali vantaggi dei sistemi posizionali sono:
- lettura più immediata dei numeri;
- rappresentazione più concisa;
- maggiore efficienza nelle operazioni aritmetiche.

Nella prossima unità analizzeremo nel dettaglio le basi utilizzate nei sistemi automatici, cioè binaria, ottale ed esadecimale: di seguito riportiamo una tabella con alcuni esempi della codifica dei primi 16 numeri nelle basi più utilizzate.

Si può notare che la lunghezza della rappresentazione del numero è diversa nelle molteplici basi e, naturalmente, è più corta in quelle che hanno il maggior numero di cifre dell'alfabeto.

	Base 2	Base 3	Base 4	Base 5	...	Base 8	...	Base 10	...	Base 16
	0000	000	00	00		00		00		0
	0001	001	01	01		01		01		1
	0010	002	02	02		02		02		2
	0011	010	03	03		03		03		3
	0100	011	10	04		04		04		4
	0101	012	11	10		05		05		5
	0110	020	12	11		06		06		6
N_b	0111	021	13	12		07		07		7
	1000	022	20	13		10		08		8
	1001	100	21	14		11		09		9
	1010	101	22	20		12		10		A
	1011	102	23	21		13		11		B
	1100	110	30	22		14		12		C
	1101	111	31	23		15		13		D
	1110	112	32	24		16		14		E
	1111	120	33	30		17		15		F

Verifichiamo le conoscenze

Esercizi a scelta multipla

1 Quanti bit occorrono per codificare 18 valori diversi?
- ○ 3
- ○ 4
- ○ 5
- ○ 6

2 Quanti bit occorrono per codificare i numeri pari da 2 a 16?
- ○ 3
- ○ 4
- ○ 5
- ○ 6

3 Quanti byte occorrono per codificare 300 informazioni diverse?
- ○ 1
- ○ 2
- ○ 3
- ○ 4

4 Quante cifre occorrono nel sistema di numerazione a base 7?
- ○ 3
- ○ 6
- ○ 7
- ○ 8

5 Quale non può essere la base del numero 5732?
- ○ 7
- ○ 8
- ○ 9
- ○ 16

6 Qual è la base del numero 133201?
- ○ 2
- ○ 3
- ○ 4
- ○ 5

7 Qual è il valore nel sistema decimale del numero che, in base 4, si scrive 10324?
- ○ 34
- ○ 64
- ○ 82
- ○ 98

8 Qual è il valore numerico associato alla cifra 1 del numero 201357?
- ○ 1
- ○ 7
- ○ 14
- ○ 49

Test vero/falso

1 Il significante di un messaggio è la sua codifica e non il suo contenuto. V F

2 Il sistema di numerazione romano è di tipo additivo/sottrattivo. V F

3 In un sistema posizionale la cifra all'estrema destra prende il nome di *cifra più significativa*. V F

4 In un sistema posizionale più la base è grande più è corta la codifica. V F

5 In un sistema posizionale di base n sono necessari n–1 simboli. V F

6 La base e la posizione della cifra indicano il peso di ogni cifra presente nel numero. V F

7 Se un numero è dispari nel sistema binario ha come ultima cifra 1. V F

8 Con 4 bit il più grande numero intero rappresentabile è 2^4. V F

Esercizi di conversione

1 Quale valore decimale è rappresentato nel seguente geroglifico?

2 Converti le seguenti rappresentazioni binarie nel formato in base 10 equivalente:

1010_2 = _____ 10
1111_2 = _____ 10
10101_2 = _____ 10
11011_2 = _____ 10
100001_2 = _____ 10
101011_2 = _____ 10

3 Converti le seguenti rappresentazioni binarie nel formato in base 10 equivalente:

01101111_2 = _____ 10
11000011_2 = _____ 10
11101110_2 = _____ 10
11110001_2 = _____ 10
10101010_2 = _____ 10
11000011_2 = _____ 10

4 Converti le seguenti rappresentazioni quinarie nel formato in base 10 equivalente:

1012_5 = _____ 10
1331_5 = _____ 10
3411_5 = _____ 10
10402_5 = _____ 10
12341_5 = _____ 10
23423_5 = _____ 10

5 Converti le seguenti rappresentazioni ottali nel formato in base 10 equivalente:

44_8 = _____ 10
246_8 = _____ 10
1357_8 = _____ 10
20402_8 = _____ 10
23755_8 = _____ 10
46325_8 = _____ 10

6 Converti le seguenti rappresentazioni in base 12 nel formato in base 10 equivalente:

72_{12} = _____ 10
AA_{12} = _____ 10
532_{12} = _____ 10
$39A_{12}$ = _____ 10
$3BC_{12}$ = _____ 10
2681_{12} = _____ 10

7 Converti le seguenti rappresentazioni esadecimali nel formato in base 10 equivalente:

AB_H = _____ 10
FF_H = _____ 10
$12D_H$ = _____ 10
$31A1_H$ = _____ 10
1234_H = _____ 10
2288_H = _____ 10

8 Verifica le seguenti equivalenze e ricerca la base incognita:

$(555)_X = (365)_{10}$
$(121)_X = (300)_4$
$(121)_X = (201202)_3$
$(111)_X = (11)_6$
$(575)_X = (3BB)_{12}$
$(2342)_X = (375)_8$

UNITÀ DIDATTICA 4

CONVERSIONE DI BASE DECIMALE

IN QUESTA UNITÀ IMPAREREMO...

- la conversione da basi pesate a decimale
- la conversione da decimale a basi pesate di numeri interi
- la conversione da decimale a basi pesate di numeri frazionali

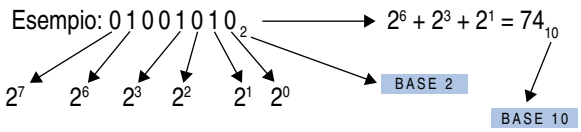
Introduzione alle conversioni di base

Il sistema di numerazione binario (sistema di numerazione in base 2) si compone di due simboli, dove ogni simbolo, denominato *bit* (*binary digit*), può assumere due valori rappresentati dai simboli logici 0 e 1.
Quando una variabile assume più di due possibili valori si ricorre a una configurazione formata da più cifre binarie (configurazione binaria) e i bit "estremi" sono denominati:

- il bit più a sinistra LSB (Least Significant Bit), o bit meno significativo;
- il bit più a destra MSB (Most Significant Bit), o bit più significativo.

Queste denominazioni dipendono dal fatto che in un sistema posizionale pesato hanno maggiore importanza le cifre a sinistra dato che hanno un peso maggiore e, in caso di errore, si avrà un danno maggiore se viene "alterata" la cifra più significativa rispetto a quella meno significativa.

Vediamo un esempio di codifica binaria: essendo un sistema di numerazione posizionale e data una cifra binaria è possibile determinarne per esempio il valore decimale interpretando i simboli che la compongono come i coefficienti del polinomio.



Questo procedimento non è altro che una conversione di base, cioè la trasformazione di un numero da una rappresentazione di partenza (in questo caso la base 2) a una seconda rappresentazione in una base diversa (in questo caso la base 10).

Conversione in decimale

CONVERSIONE IN DECIMALE

La conversione tra un numero espresso in una qualunque base e il formato decimale è immediata applicando la definizione, cioè da un generico numero scritto simbolicamente con

$$N_b = c_{m-1}c_{m-2} \ldots c_2c_1c_0$$

si ottiene il corrispondente valore decimale sostituendo a ogni termine il prodotto della cifra con il suo peso:

$$N_{10} = c_{m-1} \cdot b^{m-1} + c_{m-2} \cdot b^{m-2} \ldots c_2 \cdot b^2 + c_1 \cdot b^1 + c_0b^0$$

cioè eseguendo la somma delle potenze.

Affrontiamo la conversione dei numeri espressi nelle diverse basi con cifre dopo la virgola (frazionali), dato che ogni sistema posizionale permette di rappresentare non solo i numeri interi ma anche i numeri reali.
In questo caso le posizioni sono negative rispetto allo 0 e quindi l'esponente sarà anch'esso negativo.
Vediamo un esempio relativo a ciascuna delle tre rappresentazioni dei numeri che vengono utilizzate nell'elaborazione automatica, cioè binaria, ottale ed esadecimale.

Conversione da binario a decimale

Convertiamo il numero 10.101, scrivendolo sotto forma di somma di termini:

$$10.101_2 = 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 1^{-2} + 1 \cdot 1^{-3} = 2 + 0 + 1 \cdot \frac{1}{2} + 0 \cdot \frac{1}{4} + 1 \cdot \frac{1}{8} = 1.625_{10}$$

Possiamo vedere ogni singolo termine nel dettaglio:

moltiplicatore	1	0	1	0	1
posizione	2	1	–1	–2	–3
potenze/peso	2^1	2^0	2^{-1}	2^{-2}	2^{-2}
	2	**1**	**1/2=0,5**	**1/4=0,25**	**1/8=0,125**

Conversione da ottale a decimale

Convertiamo il numero 1531.24, scrivendolo sotto forma di somma di termini:

$$1531.24_8 = 1 \cdot 8^3 + 5 \cdot 8^2 + 3 \cdot 8^1 + 1 \cdot 8^0 + 2 \cdot 8^{-1} + 4 \cdot 8^{-2} = 512 + 320 + 8 + 2 \cdot \frac{1}{8} + 4 \cdot \frac{1}{64} = 857.314_{10}$$

Possiamo vedere ogni singolo termine nel dettaglio:

moltiplicatore	1	5	3	1	2	4
posizione	4	3	2	1	–1	–2
potenze/peso	8^3	8^2	8^1	8^0	8^{-1}	8^{-2}
	512	**64**	**8**	**1**	**1/8=0,125**	**1/64=0,016**

Conversione da esadecimale a decimale

Convertiamo il numero 1ABC.25, scrivendolo sotto forma di somma di termini:

$$ABC.25_8 = 10\cdot16^2+11\cdot16^1+12\cdot16^0+2\cdot16^{-1}+5\cdot16^{-2} =$$
$$= 10\cdot256+11\cdot16+12+2\cdot\frac{1}{16}+4\cdot\frac{1}{256} = 2748.142_{10}$$

Possiamo vedere ogni singolo termine nel dettaglio:

moltiplicatore	0	A = 10	B = 11	C = 12	2	4
posizione	4	3	2	1	–1	–2
potenze/peso	16^3	16^2	16^1	16^0	16^{-1}	16^{-2}
	4096	**256**	**16**	**1**	**1/16=0,063**	**1/64=0,004**

Conversione da decimale intero alle diverse basi

Esiste una tecnica generale per effettuare l'operazione inversa, cioè per convertire un numero decimale in un'altra base: si applica l'algoritmo della divisione ripetuta, cioè si divide ripetutamente il numero decimale per la base desiderata, si considerano i soli resti delle divisioni e li si prende al contrario. Il procedimento è iterato fino a ottenere uno 0 come nuovo dividendo. L'algoritmo è il seguente:

A) Se il numero da convertire è "0"
  allora non c'è altro da convertire,
  altrimenti
    a. si divide il numero per 2 e si individua il resto;
    b. si sostituisce il valore con il quoziente della divisione;
    c. si torna al passo A e si ricomincia la divisione.
B) Si leggono i resti nell'ordine opposto a quello con cui sono stati trovati.

Conversione da decimale a binario

Vediamo un primo esempio convertendo passo passo il numero N = 59_{10}.

Passo A: il valore di N è 59 quindi non è uguale a 1, perciò:

Ⓐ si divide per 2 e si trova il quoziente (29) e il resto (1)

passo	Valore	Divisore	Quoziente	Resto
1a	59	2	29	1

B si sostituisce al valore (59) il quoziente (29)

passo	Valore	Divisore	Quoziente	Resto
1a	59	2	29	1
1b	29			

Si continua a ripetere questo procedimento fino a che il quoziente diviene 1.

Iterazione	Valore	Divisore	Quozienti	Resti
1	59	2	29	1
2	29	2	14	1
3	14	2	7	0
4	7	2	3	1
5	3	2	1	1
6	1	2	0	1

Alla sesta iterazione il quoziente vale 0, quindi l'algoritmo termina eseguendo il passo B, cioè la lettura dei resti in ordine opposto, dall'ultimo al primo (dal basso verso l'alto):

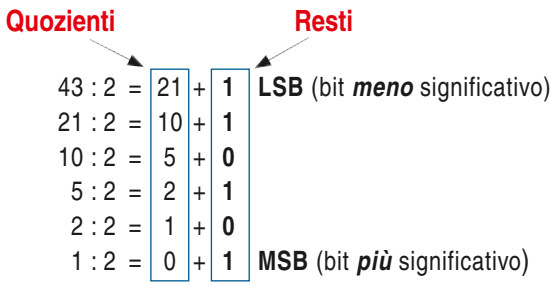
N = 1 1 1 0 1 1

Quindi il risultato è il seguente:

$$(59)_{10} = (111011)_2$$

Nel sistema numerico binario:
- un numero pari termina sempre con un bit 0;
- un numero dispari termina sempre con un bit 1.

Vediamo un secondo esempio convertendo il numero 43_{10} in base 2:



I resti ottenuti sono 1 1 0 1 0 1 e "rigirando" tali valori si ottiene 101011, che identifica il valore binario del numero 43.
Quindi il risultato è il seguente:

$$(43)_{10} = (101011)_2$$

Conversione da decimale a ottale

L'algoritmo è il medesimo, dove ora il divisore è il numero 8, cioè la base del sistema di numerazione. Convertiamo il numero 3157_{10}.

Iterazione	Valore	Divisore	Quozienti	Resti
1	3157	8	394	5
2	394	8	49	2
3	49	8	6	1
4	6	8	0	6

Alla quarta iterazione il quoziente vale 0, quindi l'algoritmo termina e la lettura dei resti in ordine opposto ci dà il risultato:

$$(3157)_{10} = (6125)_{8}$$

Vediamo un secondo esempio di come il numero 44 si converte in base 8:

Iterazione	Valore	Divisore	Quozienti	Resti
1	44	8	5	4
2	5	8	0	5

I resti ottenuti sono 4 5 e "rigirando" tali valori si ottiene 54, che identifica il valore ottale del numero 44.

CONVERSIONE TRA BASI

Se il numero è espresso in una base b1 e lo si vuole convertire in una base b2, basta dividere prima il numero e poi i successivi quozienti per la base b2 espressa nella base b1.

Il codice del numero nella nuova base sarà formato dai resti delle divisioni successive presi ordinatamente dall'ultimo al primo.

In questo modo il resto della prima divisione costituirà la cifra meno significativa, mentre quello dell'ultima la cifra più significativa.

Conversione da decimale a esadecimale

Anche per questo caso utilizziamo lo stesso algoritmo e convertiamo lo stesso numero 3157_{10}.

Iterazione	Valore	Divisore	Quozienti	Resti
1	3157	16	197	5
2	197	16	12	5
3	12	16	0	C

Alla terza iterazione il quoziente vale 0, quindi l'algoritmo termina e la lettura dei resti in ordine opposto ci dà il risultato:

$$(3157)_{10} = (C55)_{H}$$

Vediamo un secondo esempio, convertendo il numero 44157_{10}.

Iterazione	Valore	Divisore	Quozienti	Resti
1	44157	16	2759	13 (D)
2	2759	16	172	7
3	172	16	10	12 (C)
4	10	16	0	10 (A)

Alla terza iterazione il quoziente vale 0, quindi l'algoritmo termina e la lettura dei resti in ordine opposto ci dà il risultato:

$$(44157)_{10} = (AC7D)_{H}$$

Conversione da decimale frazionale alle diverse basi

La parte frazionaria del numero in base b è ottenuta moltiplicando ripetutamente la parte frazionaria del numero decimale per la base b e registrando la parte intera dei successivi prodotti, presi ora dall'alto verso il basso.

I numeri frazionari che hanno una rappresentazione esatta in base b daranno a un certo punto un risultato uguale a 0. Altrimenti ci si ferma alla precisione desiderata.

L'algoritmo è il seguente:

A) Si stabilisce la precisione desiderata (per esempio 3 bit dopo la virgola).
B) Si moltiplica la parte decimale per la base.
  Se si ottiene 0
    allora termine conversione,
    altrimenti
      a. si evidenzia la parte intera del risultato, cioè se supera o meno l'unità;
      b. si sostituisce al numero la parte decimale del prodotto ottenuto;
      c. si torna al passo B e si ripete la moltiplicazione.
C) Si leggono i numeri segnati nell'ordine in cui sono stati trovati.

Per essere precisi, la condizione di terminazione nel passo B è la seguente:

Se (si ottiene 0) oppure (superato il numero di cifre stabilite)

in quanto il numero ottenuto potrebbe essere periodico e dare sempre risultato diverso da zero.

Conversione da decimale a binario

Effettuiamo la conversione del numero 124.125.

A Parte intera: 124

124	: 2 =	62	0
62	: 2 =	31	0
31	: 2 =	15	1
15	: 2 =	7	1
7	: 2 =	3	1
3	: 2 =	1	1
1	: 2 =	0	1

I resti ottenuti sono 0 0 1 1 1 1 1 e "rigirando" tali valori si ottiene 1111100, che identifica il valore binario del numero 124.
Quindi il risultato è il seguente:

$$(124)_{10} = (1111100)_2$$

B Parte frazionale: 0.125
Seguiamo il procedimento passo passo.

Passo A: stabiliamo come precisione 3 bit dopo la virgola (quindi faremo 3 iterazioni).

Passo B: moltiplichiamo la parte decimale (0.125) per la base (2):

Iterazione	Passo	Valore	Moltiplicatore	Risultato	Parte intera
1	Ba	0.125	2	**0**.250	

A Il risultato ottenuto (0,250) non è uguale a 0, quindi si prosegue evidenziando la parte intera del prodotto ottenuto (0):

Iterazione	Passo	Valore	Moltiplicatore	Risultato	Parte intera
1	Ba	0.125	2	**0**.250	**0**

B Sostituiamo il prodotto (0,250) al valore del moltiplicando (0.125):

Iterazione	Passo	Valore	Moltiplicatore	Risultato	Parte intera
1	Ba	0.125	2	**0.250**	0
1	Bb	**0.250**	2		

C Ripetiamo l'istruzione B, moltiplicando tale valore per la base e, dato che il risultato è diverso da 0, proseguiamo con il passo Ba e quindi con il passo Bb:

Iterazione	Passo	Valore	Moltiplicatore	Risultato	Parte intera
1	Ba	0.125	2	**0**.250	**0**
1	Bb	0.250	2		
2	Ba	0.250	2	**0**.500	**0**
2	Bb	**0**.500	2		

Abbiamo individuato due bit e ripetiamo di nuovo il passo B:

Ⓐ moltiplichiamo il nuovo prodotto ottenuto per la base e, dato che il risultato è diverso da 0, proseguiamo con il passo Ba e quindi con il passo Bb (come terza iterazione):

Iterazione	Passo	Valore	Moltiplicatore	Risultato	Parte intera
1	Ba	0.125	2	**0**.250	**0**
1	Bb	0.250	2		
2	Ba	0.250	2	**0**.500	**0**
2	Bb	0.500	2		
3	Ba	0.500	2	**1**.000	**1**
3	Bb	0.000	2		

Ora l'iterazione termina in quanto il nuovo valore è uguale a 0. La parte decimale è 001, quindi il risultato è il seguente:

$$(.125)_{10} = (.001)_2$$

In questo caso sarebbe terminata comunque, dato che abbiamo individuato 3 bit dopo la virgola, come avevamo indicato al punto A.

Il risultato completo della conversione è il seguente:

$$(124.125)_{10} = (1111100.001)_2$$

ESEMPIO

Vediamo un secondo esempio, convertendo il numero $(0{,}21875)_{10}$ con precisione 6 bit.

Iterazione	Valore	Moltiplicatore	Risultato	Parte intera
1	0.21875	2	0.4375	0
2	0.4375	2	0.875	0
3	0.875	2	0.500	0
4	0.500	2	1.75	1
5	0.75	2	1.5	1
6	0.5	2	1.0	1

Quindi il risultato è il seguente:

$$(.21875)_{10} = (.000111)_2$$

Anche in questo caso la conversione è terminata per un doppio motivo:
- abbiamo individuato 6 bit dopo la virgola;
- la parte decimale dopo la sesta iterazione è uguale a 0.

ESEMPIO

Vediamo un ultimo esempio, convertendo il numero $(0.45)_{10}$ con precisione 6 bit.

Iterazione	Valore	Moltiplicatore	Risultato	Parte intera
1	0.45	2	0.9	0
2	0.9	2	1.8	1
3	0.8	2	1.6	1
4	0.6	2	1.2	1
5	0.2	2	0.4	0
6	0.8	2	1.6	1

Il risultato della conversione è il seguente:

$$(.45)_{10} = (.011101)_2$$

In questo caso la conversione è terminata solo per aver esaurito i 6 bit dopo la virgola: infatti la parte decimale dell'ultimo prodotto è 0.6 e, come si può vedere dalla tabella, si innesta un'iterazione senza fine, dato che il medesimo risultato si era già trovato nella terza iterazione.
Il numero risulta essere periodico di periodo 110, con 01 come antiperiodo. La conversione completa sarebbe:

$$(.45)_{10} = (.01110110110110110110...)_2$$

Conversione da decimale a ottale

L'algoritmo è il medesimo, dove ora il moltiplicatore è il numero 8, cioè la base del sistema di numerazione. Convertiamo il numero 0.45_{10} con precisione 6 bit:

Iterazione	Valore	Moltiplicatore	Risultato	Parte intera
1	0.45	8	3.6	3
2	0.6	8	4.8	4
3	0.8	8	6.4	6
4	0.4	8	3.2	3
5	0.2	8	1.6	1
6	0.6	8	4.8	4

Il risultato della conversione è il seguente:

$$(.45)_{10} = (.346314)_8$$

Anche in questo caso si innesta un'iterazione senza fine, dato che il medesimo risultato della sesta iterazione lo si era già trovato nella seconda iterazione.
Il numero risulta essere periodico di periodo 4631, con 3 come antiperiodo.

Conversione da decimale a esadecimale

Utilizziamo sempre lo stesso algoritmo, ora con il numero 16 come moltiplicatore. Convertiamo il numero 0.45_{10} con precisione 4 bit:

Iterazione	Valore	Moltiplicatore	Risultato	Parte intera
1	0.45	16	7.2	7
2	0.2	16	3.2	3
3	0.2	16	3.2	3
4	0.3	16	3.2	3

Il risultato della conversione è il seguente:

$$(.45)_{10} = (.7333)_{H}$$

Anche in questo caso si innesta un'iterazione senza fine di periodo 3, con 7 come antiperiodo.

Conclusioni

A conclusione di questa unità didattica sulla conversione tra basi diverse enunciamo il teorema della divisione che definisce la completezza dei sistemi di numerazione posizionali:

TEOREMA DI DIVISIONE

Il teorema di divisione garantisce che, per qualunque base b > 1, tutti i numeri naturali possono essere rappresentati in un sistema di numerazione (posizionale) a base b.
Non è invece garantita la rappresentabilità dei numeri frazionali.

Infatti:

- il numero 1.210 non è rappresentabile in binario, nel senso che sarebbero necessarie infinite cifre per rappresentarlo: cercando di rappresentarlo si avrebbe una stringa che comincia per 1.001100110011…
- il numero 0.1_3 non è rappresentabile in decimale: infatti è uguale a 1/3, che si scrive 0.3333…

Il concetto di numero periodico è quindi legato alla base di numerazione: un numero periodico in una determinata base può non esserlo in un'altra.

Verifichiamo le competenze

Esprimi la tua creatività

1 **Converti i seguenti numeri frazionari binari in decimale:**

1010.101_2 = _____ 10
1011.100_2 = _____ 10
11101.001_2 = _____ 10
10011.101_2 = _____ 10
110111.11_2 = _____ 10
111010.01_2 = _____ 10

2 **Converti i seguenti numeri frazionari binari in decimale:**

11101.11010111 = _____ 10
10001.00001101 = _____ 10
00011.10011001 = _____ 10
11000.11111001 = _____ 10
10011.00100100 = _____ 10
10010.01001001 = _____ 10

3 **Converti le seguenti rappresentazioni nel formato in base 10 equivalente:**

341.1_5 = _____ 10
234.23_5 = _____ 10
1312.42_5 = _____ 10
2040.2_8 = _____ 10
2375.5_8 = _____ 10
1463.25_8 = _____ 10

4 **Converti le seguenti rappresentazioni nel formato in base 10 equivalente:**

268.1_{12} = _____ 10
568.4_{12} = _____ 10
543.51_{12} = _____ 10
123.2_{16} = _____ 10
222.5_{16} = _____ 10
$2AB.19_{16}$ = _____ 10

5 **Converti le seguenti rappresentazioni in base dieci nel formato binario equivalente:**

32 = _____ 2
64 = _____ 2
96 = _____ 2
15 = _____ 2
127 = _____ 2
198 = _____ 2
245 = _____ 2
255 = _____ 2

6 **Converti le seguenti rappresentazioni in base dieci frazionali nel formato binario equivalente con precisione di 8 bit:**

23.466 = _____ 2
61.625 = _____ 2
13.543 = _____ 2
55.110 = _____ 2
19.999 = _____ 2
22.001 = _____ 2
41.700 = _____ 2

7 **Calcola i valori decimali dei seguenti numeri posizionali espressi in varie basi:**

N(2) = 10011.101_2
N(2) = 110111.1111_2
N(7) = 4625.32_7
N(6) = 36541.32_6
N(6) = 534.01_6

8 **Converti i seguenti numeri decimali nelle basi indicate:**

7516_{10} = _____ 8
108_{10} = _____ 2
108_{10} = _____ 3
115_{10} = _____ 7

9 **Converti il seguente numero esadecimale (D7A) (16) in base 4.**

10 **Utilizza la notazione esadecimale per rappresentare le seguenti configurazioni di bit:**

0100100000010111 _____
1010101011110000 _____
0110011011000000 _____
1110100001010101 _____
0101010101011001 _____

11 **Quali configurazioni di bit sono rappresentate dalle seguenti configurazioni esadecimali?**

5FD9 _____
610A _____
ABCD _____
0100 _____
CACA _____

12 **Effettua le seguenti conversioni:**

54.75_{10} = ?$_2$
123.625_{10} = ?$_2$
123_{10} = ?$_5$

UNITÀ DIDATTICA 5

CONVERSIONE TRA LE BASI BINARIE

IN QUESTA UNITÀ IMPAREREMO

- la conversione tra binario e ottale
- la conversione tra binario ed esadecimale
- la conversione tra ottale ed esadecimale

Introduzione

La conversione di un'informazione da una base a un'altra viene spesso utilizzata nei sistemi digitali e in questa unità affronteremo casi particolari che permettono di sfruttare alcune proprietà "matematiche" per passare tra basi con caratteristiche comuni.

Il sistema binario utilizza due soli simboli per rappresentare l'informazione, cioè a base $b = 2^1$.
Oltre al sistema binario e decimale i sistemi usati per la rappresentazione dell'informazione sono il sistema ottale e il sistema esadecimale:

- sistema binario: $b = 2^1 = 2$;
- sistema ottale: $b = 2^3 = 8$;
- sistema esadecimale: $b = 2^4 = 16$.

Ricordiamo che in ogni caso nel calcolatore le informazioni sono codificate in binario.

Con la notazione $B_1 \rightarrow B_2$ indichiamo la conversione di base, cioè dobbiamo rappresentare ogni cifra della rappresentazione in base B_1 nella rappresentazione in base B_2.

Tra i sistemi di numerazione appena elencati esiste un particolare legame tra le basi: sono tra loro legate dalla potenza, cioè la base ottale è terza potenza della base binaria e la base esadecimale è quarta potenza della base binaria. Abbiamo quindi una relazione del tipo $B_a = B_b^k$.

Base di partenza potenza della base di arrivo

Quando tra la base B_1 e la base B_2 esiste la corrispondenza del tipo $a = b^k$, cioè la base di partenza è una potenza *k*-esima della base di arrivo, ogni cifra della prima rappresentazione sarà composta da *k* cifre della seconda.

Quindi:
- tra base ottale e base binaria, cioè con $B_1 = 8$ e $B_2 = 2$, abbiamo k = 3 dato che $8 = 2^3$ e quindi ogni cifra in base 8 può essere rappresentata in base B_2 con tre cifre;
- tra base esadecimale e base binaria, cioè con $B_1 = 16$ e $B_2 = 2$, abbiamo k = 4 dato che $16 = 2^4$ e quindi ogni cifra in base 16 può essere rappresentata in base B_2 con quattro cifre.

Base di arrivo potenza della base di partenza

Quando tra la base B_1 e la base B_2 esiste la corrispondenza del tipo $b = a^k$, cioè la base di arrivo è una potenza *k*-esima della base di partenza, ogni cifra della seconda rappresentazione si ottiene raggruppando *k* cifre della prima.

Quindi:
- tra base binaria e base ottale, cioè con $B_1 = 2$ e $B_2 = 8$, abbiamo $k = 3$ dato che $8 = 2^3$ e quindi ogni cifra in base 8 può essere ottenuta raggruppando le cifre binarie tre alla volta e convertendo ogni gruppo in una cifra ottale;
- tra base binaria e base esadecimale, cioè con $B_1 = 2$ e $B_2 = 16$, abbiamo $k = 4$ dato che $16 = 2^4$ e quindi ogni cifra in base 16 può essere ottenuta raggruppando le cifre binarie quattro alla volta e convertendo ogni gruppo in una cifra ottale.

Conversione tra binari e ottali

Il sistema ottale si basa sulla numerazione in base 8, quindi con b = 8, che utilizza i simboli $\Sigma = \{0,1,2,3,4,5,6,7\}$ che corrispondono alle prime 8 cifre del sistema decimale: ogni numero è costituito da una stringa di cifre ottali il cui valore è determinato dal prodotto della cifra per una potenza della base, il cui esponente è dato dalla posizione della cifra nella stringa.
Viene usato principalmente come rappresentazione intermedia per la comunicazione con le macchine digitali in quanto, raggruppando terne di bit, è più vicino al sistema di numerazione decimale utilizzato dall'uomo rispetto al binario utilizzato dalla macchina.
Nella tabella seguente riportiamo la codifica nelle tre basi ottale, decimale e binaria:

ottale	0_8	1_8	2_8	3_8	4_8	5_8	6_8	7_8
decimale	0_{10}	1_{10}	2_{10}	3_{10}	4_{10}	5_{10}	6_{10}	7_{10}
binaria	000_2	001_2	010_2	011_2	100_2	101_2	110_2	111_2

Possiamo confrontare la codifica ottale con i primi otto elementi della codifica decimale e osservare che coincidono.
Tutte le otto combinazioni sono codificate con stringhe di 3 bit, dato che $2^3 = 8$.

Eseguiamo ora la conversione tra le basi ottale e binaria.

Da binario a ottale

Per passare dalla codifica binaria a quella ottale raggruppiamo le cifre binarie a gruppi di 3 e sostituiamole con la corrispondente cifra del sistema ottale.

ESEMPIO 1

Convertiamo il numero binario 110111_2. Per prima cosa separiamo i bit a gruppi di 3:

$110_2 \quad 111_2$

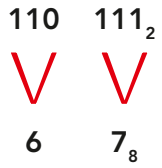
Quindi a ogni gruppo sostituiamo la cifra ottale corrispondente, cioè convertiamo in ottale ogni gruppo di 3 cifre:

$110_2 = 6_8$ e $111_2 = 7_8$

Quindi il numero ottale risultante è il seguente:

$110111_2 = 67_8$

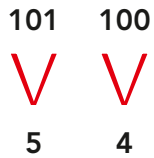
Sinteticamente, si può effettuare l'operazione con lo schema a lato: ▶



ESEMPIO 2

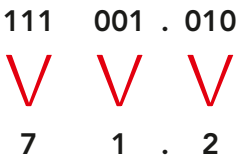
Vediamo un secondo esempio, dove convertiamo il numero 101100_2. Schematicamente abbiamo: ▶

Quindi il risultato è $101100_2 = 54_8$.



ESEMPIO 3

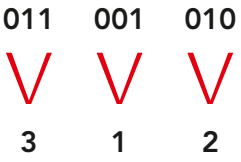
Lo stesso procedimento è applicabile anche ai numeri frazionali. Convertiamo per esempio in notazione ottale il numero binario 111010.010: ▶



ESEMPIO 4

Vediamo come convertire il numero 11001010_2: in questo caso il numero binario è composto da 8 bit, quindi non è costituito da un multiplo di 3.
Bisogna fare attenzione ad aggiungere a sinistra i bit mancanti, naturalmente con valore 0.
Il numero da convertire diviene quindi il seguente 011001010_2, suddivisibile in tre ottetti: ▶

Quindi il risultato è $11001010_2 = 312_8$.



ESEMPIO 5

Anche per le cifre frazionarie si deve fare attenzione a rendere raggruppabili a terzetti i bit di partenza aggiungendo eventualmente uno o più zeri come cifra più significativa per la parte intera e uno o più zeri come cifra meno significativa per la parte frazionaria.

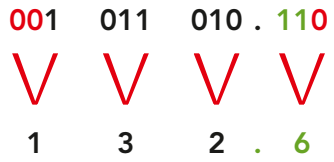
Convertiamo 1011010.11_2 e per prima cosa separiamo la parte intera da quella frazionaria:

parte intera : 1011010_2
parte frazionaria : 11_2

In entrambe le parti abbiamo un numero di bit insufficiente e quindi aggiungiamo gli zeri mancanti:

parte intera : $001\ 011\ 010_2$
parte frazionaria : 110_2

Ora effettuiamo la conversione: ▶



Quindi il risultato è $1011010.11_2 = 132.6_8$.

L'errore tipico che si commette in questa situazione è quello di convertire per esempio il numero 10111010.11_2 in:

101 110 10.11
5 6 2 . 3

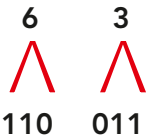
Si deve sempre partire dal punto, eventualmente completando le cifre con degli zeri per ottenere le terne: *xxx xxx . yyy yyy* …: il numero 10111010.11 viene quindi trasformato in 010 111 010.110 prima di eseguire la conversione.

Da ottale a binario

Per passare dalla codifica ottale a quella binaria a ogni cifra ottale sostituiamo la corrispondente cifra codificata in binario.

ESEMPIO 6

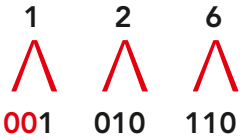
Convertiamo il numero ottale 63_8 in binario, utilizzando lo schema nella figura: ▶



Quindi il risultato è $63_8 = 111011_2$.

ESEMPIO 7

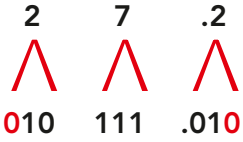
Convertiamo il numero ottale 126_8 in binario: ▶



Quindi il risultato è $126_8 = 001010110_2$, dove eventualmente gli zeri a sinistra possono essere eliminati ottenendo $126_8 = 1010110_2$.

ESEMPIO 8

Vediamo un ultimo esempio con la conversione di un numero frazionario. Convertiamo 27.2_8 in binario: ▶



Quindi il risultato è $27.2_8 = 010111.010_2$, ed eliminando gli zeri superflui otteniamo 10111.01_2.

Conversione tra binari ed esadecimali

Il sistema esadecimale si basa sulla numerazione in base 16, quindi con b = 16, che utilizza i simboli Σ = {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}, dove ogni numero è costituito da una stringa di cifre esadecimale il cui valore è determinato dal prodotto della cifra per una potenza della base il cui esponente è dato dalla posizione della cifra nella stringa.

Nella tabella seguente riportiamo la codifica nelle tre basi esadecimale, decimale e binaria. Essa viene usata come rappresentazione intermedia per la comunicazione con le macchine digitali in quanto, essendo ogni byte composto da 8 bit, viene suddiviso in due gruppi da 4 bit (nibble) e ciascuno di essi è codificato mediante il sistema esadecimale.

Esadecimale	0_{16}	1_{16}	2_{16}	3_{16}	4_{16}	5_{16}	6_{16}	7_{16}	8_{16}	9_{16}	A_{16}	B_{16}	C_{16}	D_{16}	E_{16}	F_{16}
Decimale	0_{10}	1_{10}	2_{10}	3_{10}	4_{10}	5_{10}	6_{10}	7_{10}	8_{10}	9_{10}	10_{10}	11_{10}	12_{10}	13_{10}	14_{10}	15_{10}
Binaria	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Zoom su...

NOTAZIONE PER I NUMERI ESADECIMALI

Generalmente nei testi che trattano le macchine digitali la rappresentazione dei simboli numerici viene indicata senza specificare la base di rappresentazione e mentre in alcuni casi è evidente l'individuazione della base per la presenza delle lettere alfabetiche come nei seguenti casi {1B, 9C, BA, 3FF, E12DFA} in altri c'è ambiguità quando contengono solo numerali come {12, 91, 172, 4354 ecc}: questi ultimi possono appartenere sia alla codifica decimale sia a quella esadecimale.
Per evitare confusione talvolta si trova il valore "39" in base esadecimale, scritto come "39 h" oppure "0x39".

Da binario a esadecimale

Per passare dalla codifica binaria a quella esadecimale raggruppiamo le cifre binarie a gruppi di 4 e sostituiamole con la corrispondente cifra del sistema esadecimale.

ESEMPIO 9

Convertiamo il numero binario 10101110_2.
Per prima cosa separiamo i bit a gruppi di 4:

$1010\ 1110_2$

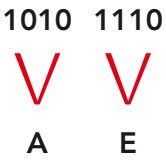
Quindi a ogni gruppo sostituiamo la cifra ottale corrispondente, cioè convertiamo in esadecimale ogni gruppo di 4 cifre:

$1010_2 = A_h$ e $1110_2 = E_h$

Il numero esadecimale risultante è il seguente:

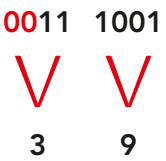
$10101110_2 = AE_h$

Sinteticamente si può effettuare l'operazione con lo schema a lato simile a quello utilizzato per la conversione in ottale: ▶



ESEMPIO 10

Convertiamo il numero binario 111001_2.
Per prima cosa completiamo l'"ottetto" aggiungendo due zeri: ▶



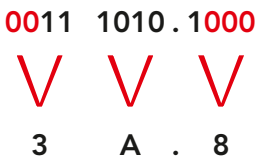
Il numero esadecimale risultante è il seguente:

$111001_2 = 39_h$

Un singolo byte è sempre rappresentato da un numero esadecimale con due cifre, la prima per i quattro bit più significativi e la seconda per quelli meno significativi:
0x0B, 0xAB, 0x5E, 0xFF ...

ESEMPIO 11

Come ultimo esempio convertiamo un numero binario frazionale 111010.1_2.
Per prima cosa completiamo il numero con gli zeri mancanti per definire i singoli nibble: ▶



Il numero esadecimale risultante è il seguente:

$111010.1_2 = 3A.8_h$

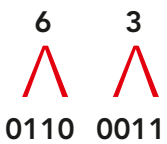
Fate attenzione quando aggiungete gli zeri e raggruppate i nibble:
- gli zeri si inseriscono sempre "all'esterno";
- i nibble si raggruppano a partire dal punto decimale xxxx . yyyy.

Da esadecimale a binario

Per passare dalla codifica esadecimale a quella binaria a ogni cifra esadecimale sostituiamo la corrispondente cifra codificata in binario.

ESEMPIO 12

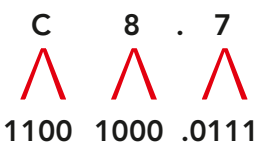
Convertiamo il numero esadecimale 63_h in binario, direttamente utilizzando lo schema nella figura a lato: ▶



Quindi il risultato è $63_h = 0110\ 0011_2$ ed eliminando il primo 0 si ottiene 1100011_2.

ESEMPIO 13

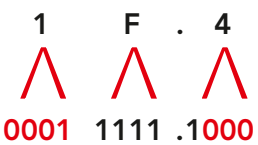
Convertiamo il numero esadecimale $C8.7_h$ in binario:



Il risultato è $C8.7_h = 1100\ 1000.0111_2$.

ESEMPIO 14

Come ultimo esempio convertiamo il numero $1F.4_h$ in binario:
Il risultato è $1F.4_h = 00011111.1000_2$ che, dopo aver eliminato gli zeri "superflui", diviene:

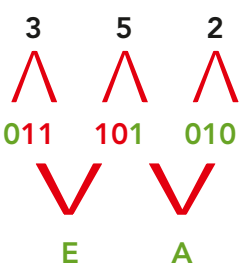


$1F.4_h = 11111.1_2$

Conversione tra ottali ed esadecimali

Il metodo più veloce per passare dal sistema ottale all'esadecimale o viceversa è quello di passare attraverso il sistema binario.

Vediamo alcuni esempi.



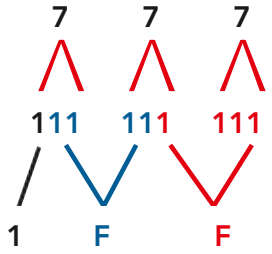
ESEMPIO 15

Convertiamo $(352)_8$ in esadecimale:

Quindi il risultato è $(352)_8 = (11\ 101\ 010)_2$ da cui si ottiene $(352)_8 = (11101010)_2 = (EA)_h$.

ESEMPIO 16

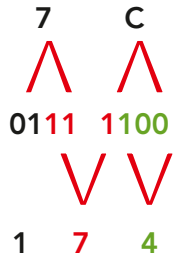
Convertiamo $(777)_8$ in esadecimale:



Quindi il risultato è $(777)_8 = (111\ 111\ 111)_2$ da cui si ottiene $(777)_8 = (0001\ 1111\ 1111)_2 = (1FF)_h$.

ESEMPIO 17

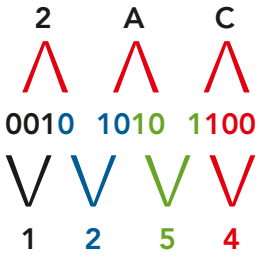
Convertiamo $(7C)_h$ in ottale:



Quindi $(7C)_h = (0111\ 1100)_2$ da cui si ottiene $(7C)_H = (001\ 111\ 100)_2 = (174)_8$.

ESEMPIO 18

Convertiamo $(2AC)_h$ in ottale:



Quindi $(2AC)_H = (0010\ 1010\ 1100)_2$ da cui si ottiene $(2AC)_H = (001\ 010\ 101\ 100)_2 = (1254)_8$.

Verifichiamo le competenze

1 Esegui le seguenti conversioni da binario a ottale:

010100_2 → _____
000111_2 → _____
101010_2 → _____
001110_2 → _____

110.011_2 → _____
1000.01_2 → _____
11.0011_2 → _____
1.00001_2 → _____

2 Esegui le seguenti conversioni da ottale a binario:

12_8 → _____
34_8 → _____
567_8 → _____
10_8 → _____

2.1_8 → _____
0.3_8 → _____
20.02_8 → _____
17.04_8 → _____

3 Esegui le seguenti conversioni da binario a esadecimale:

1111010_2 → _____
10101110_2 → _____
11111011_2 → _____
11010101_2 → _____

0111010.1_2 → _____
10111110.01_2 → _____
1011011.101_2 → _____
10010.001_2 → _____

4 Esegui le seguenti conversioni da esadecimale a binario:

$A7F_{16}$ → _____
$B4C_{16}$ → _____
$27E_{16}$ → _____
490_{16} → _____

32.2_{16} → _____
$A3.1C_{16}$ → _____
$2B.38_{16}$ → _____
$E3.64_{16}$ → _____

5 Esegui le seguenti conversioni da ottale a esadecimale:

72_8 => _____ $_2$ → _____ $_h$
54_8 => _____ $_2$ → _____ $_h$
321_8 => _____ $_2$ → _____ $_h$
610_8 => _____ $_2$ → _____ $_h$

2.7_8 => _____ $_2$ → _____ $_h$
0.45_8 => _____ $_2$ → _____ $_h$
21.32_8 => _____ $_2$ → _____ $_h$
47.024_8 => _____ $_2$ → _____ $_h$

6 Esegui le seguenti conversioni da esadecimale a ottale:

72_h => _____ $_2$ → _____ $_8$
18_h => _____ $_2$ → _____ $_8$
$AB2_h$ => _____ $_2$ → _____ $_8$
90_h => _____ $_2$ → _____ $_8$

34.5_h => _____ $_2$ → _____ $_8$
$67.FE_h$ => _____ $_2$ → _____ $_8$
$CD.63_h$ => _____ $_2$ → _____ $_8$
$27.9B_h$ => _____ $_2$ → _____ $_8$

UNITÀ DIDATTICA 6

IMMAGINI, SUONI E FILMATI

IN QUESTA UNITÀ IMPAREREMO...

- la rappresentazione delle immagini in binario
- la rappresentazione dei filmati
- la rappresentazione dei suoni in binario

Introduzione

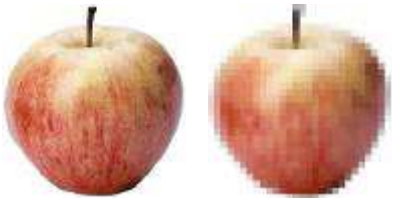
Lettere e numeri non costituiscono gli unici dati utilizzati dagli elaboratori in quanto sempre più applicazioni utilizzano ed elaborano anche altri tipi di informazione: immagini, suoni e filmati.

In questi casi si parla di applicazioni di tipo multimediale.

Queste informazioni sono di natura continua e devono essere trasformate in digitale: la rappresentazione discreta di informazioni continue comporta necessariamente una perdita di dati e a seconda della tecnica che viene utilizzata nella conversione otteniamo risultati più o meno "fedeli" all'origine analogica.

ESEMPIO

Un'immagine può essere rappresentata in digitale nei due formati diversi raffigurati sotto:



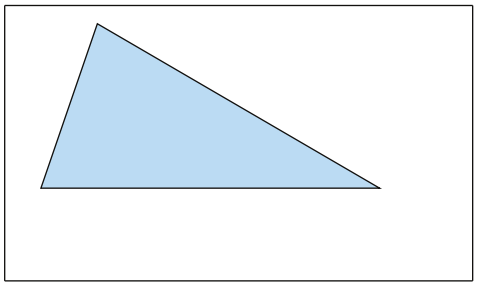
Il primo formato è perfettamente identico all'immagine analogica mentre nel secondo formato si nota che durante la conversione si sono perse delle informazioni.

In questa unità didattica verranno analizzati i formati digitali più utilizzati per le componenti multimediali.

Immagini digitali

Nella rappresentazione digitale di immagini sia pittoriche che grafiche è necessario trasformare in byte l'immagine analogica (per esempio una fotografia), cioè un insieme continuo di informazioni (luce, colore) in due dimensioni: la digitalizzazione avviene sovrapponendo idealmente una griglia fittissima di minuscole celle denominate pixel (*picture element*). A ogni punto viene associato un numero e tale numero corrisponde mediante una tabella di corrispondenza (tavolozza) a colori diversi o a sfumature diverse di un particolare colore: maggiore è la "rete di cellette", cioè più piccola è la dimensione della celletta, maggiore è il numero di pixel che viene utilizzato per definire l'immagine.

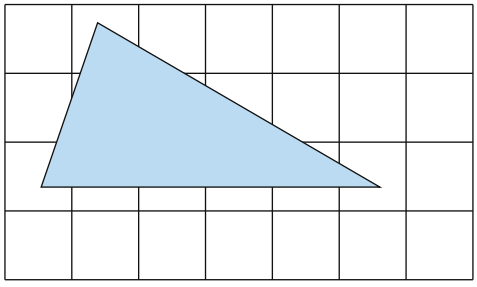
In riferimento all'immagine digitalizzata si parla di definizione facendo riferimento al numero di pixel e quindi alla qualità con cui si rappresenta l'immagine.



Bianco e nero

Vediamo per esempio come è possibile digitalizzare un triangolo in bianco e nero (e successivamente aggiungeremo i colori) rappresentato nella figura a lato:

Sovrapponiamo alla nostra immagine una griglia composta da righe orizzontali e verticali a distanza costante in modo da ottenere una griglia (matrice) di quadrati, come nella figura a lato:



Ogni quadrato così ottenuto prende il nome di pixel (*picture element*) e viene preso come unità di misura di riferimento delle immagini digitalizzate.

Codificare un'immagine consiste nel codificare i pixel in cui viene scomposta l'immagine: essa prende il nome di immagine bitmap o raster.

Codifichiamo ogni singolo pixel con un bit, in modo da assegnare:

- 0 se nel pixel il colore predominante è il bianco;
- 1 se nel pixel il colore predominante è il nero.

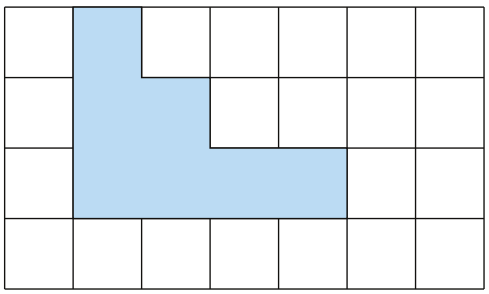
0 22	1 23	0 24	0 25	0 26	0 27	0 28
0 15	1 16	1 17	0 18	0 19	0 20	0 21
0 8	1 9	1 10	1 11	1 12	0 13	0 14
0 1	0 2	0 3	0 4	0 5	0 6	0 7

Stabiliamo ora un ordinamento tra i pixel in modo da salvare l'immagine come una sequenza di bit per poi poterla ripristinare identica: numeriamo i pixel a partire da quello in basso a sinistra fino ad arrivare a quello in alto a destra, come riportato nella figura.

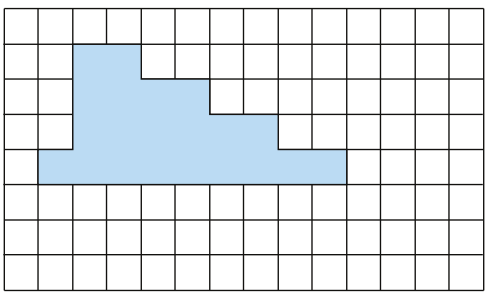
Otteniamo la sequenza bit seguente: 0000000 0111100 0110000 0100000

Effettuiamo ora il procedimento inverso, cioè dall'immagine memorizzata in bit ricostruiamo l'immagine nella griglia, ricordando che a ogni 1 corrisponde un pixel nero mentre a ogni 0 un pixel bianco.

Otteniamo il disegno rappresentato nella figura a lato, che è "abbastanza" lontano dall'immagine di partenza: ▶



Per avvicinarci alla realtà del disegno è necessario ridurre le dimensioni della griglia, cioè aumentare il numero di pixel; già dimezzando il lato del quadrato otteniamo il disegno raffigurato a lato: ▶



- nella prima codifica abbiamo rappresentato l'immagine con 7 × 3 = 21 bit;
- nella seconda codifica abbiamo una griglia composta da 14 × 6 = 84 bit.

Risulta evidente che all'aumentare del numero di pixel aumenta la qualità dell'immagine.

DEFINIZIONE DI UN'IMMAGINE

Con definizione di un'immagine si intende il numero di pixel che sono utilizzati per rappresentarla: più è alta la definizione, maggiore è la qualità dell'immagine stessa.

Livelli di grigio

Assegnando un bit a ogni pixel è possibile codificare solo immagini senza tonalità, in quanto il pixel o è bianco o è nero.
Per poter rappresentare anche i grigi e quindi l'insieme dei " livelli di chiaroscuro" è necessario utilizzare più bit per ogni pixel:

- con 4 bit possiamo codificare 16 toni di grigio (livelli di grigio);
- con 8 bit possiamo codificare 256 toni di grigio (livelli di grigio).

Calcoliamo per esempio l'occupazione di un'immagine codificata a 256 toni di grigio con qualità 640 × 480 pixel: essa necessita di 307.200 byte (1 × 640 × 480), quindi circa 300 KB di memoria.

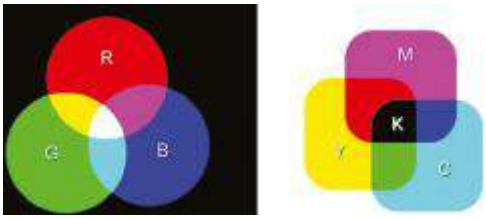
Immagini a colori

In modo analogo possiamo codificare le immagini a colori, cioè assegnando a ogni pixel un colore o una sua sfumatura. Il colore viene ottenuto da almeno tre colori di base, detti primari, secondo un modello cromatico: RGB (rosso, verde, blu) oppure CMYK (ciano, magenta, giallo, nero).

La composizione del colore avviene successivamente mediante una tecnica di sintesi che può essere sottrattiva o additiva, in funzione del tipo di dispositivo utilizzato (stampante, monitor o televisore):

- il metodo RGB utilizza la sintesi additiva per sommare i tre colori presenti e offrire tutta la gamma cromatica della foto ed è indicato per la visualizzazione a schermo in quanto i colori risultano più brillanti e saturi; inoltre il file in RGB è più piccolo del 25% rispetto allo stesso file in CMYK;
- il metodo CMYK utilizza la sintesi sottrattiva per sommare i suoi colori ed è la migliore soluzione per la stampa delle immagini in alta qualità (è il metodo che viene usato in tipografia stampando più volte lo stesso file, sovrapponendo colore per colore fino a ottenere l'immagine finale).

La figura seguente confronta la composizione additiva (RGB) e sottrattiva (CMYK) dei colori:



Per codificare un'immagine è possibile memorizzare il colore di ogni singolo bit oppure codificare ogni pixel mediante una tavolozza di colori (tecnicamente detta palette) per poi codificare i bit dell'immagine in riferimento al colore della tavolozza.
La tabella è un esempio di palette di 256 colori in RGB e per ogni componente di colore utilizza un byte.

	R	G	B
0	0	0	0
1	5	26	177
2	112	25	34
3	34	23	123
…			
254	181	123	24
255	255	255	255

Utilizzando la palette si risparmiano parecchi byte nella memorizzazione dell'immagine ma oltre all'immagine si deve memorizzare anche la palette con i colori utilizzati: nell'esempio precedente la palette occupa 256 × 3 = 768 byte.

CODIFICA BITMAP

La rappresentazione di un'immagine mediante la codifica dei pixel viene chiamata codifica bitmap.
Il numero di bit utilizzati per codificare ogni pixel prende il nome di profondità di colore.

L'immagine seguente mostra tre diverse profondità di colore:
- la prima è salvata a 16 colori e occupa 4 KB;
- la seconda è salvata a 256 colori e occupa 9 KB;
- la terza è bmp a 24 bit/pixel e occupa 21 KB.

Anche per le immagini a colori è abbastanza semplice effettuare il calcolo della dimensione di memoria occupata:
- se codifichiamo i singoli bit e utilizziamo 256 colori, quindi un solo byte per pixel, con una risoluzione di 640 × 480 pixel, l'occupazione è la medesima della precedente immagine a 256 toni di grigio (cioè 300 KB), ma se si aumenta il numero di colori portandoli a 64.000 (due byte per pixel) si ottiene il doppio, ovvero 600 KB, e per un'immagine true color (un byte per colore RGB, quindi 16 milioni di colori) la dimensione diventa di 900 KB;
- se utilizziamo invece una palette di 256 colori in true color si utilizzano 3 byte per definire ogni colore della palette (256 × 3 = 768 byte) e 8 bit per codificare ogni pixel, quindi otteniamo per un'immagine di 640 × 480 pixel una dimensione di (640 × 480 × 1) + (768 × 1) = 307.200 + 768 byte, circa 300 KB.

Osserviamo quanto sia notevole il risparmio di memoria a parità di numero di colori per pixel in caso di utilizzo di una palette di colori (900 KB contro 300 KB): ma dobbiamo sottolineare come nel primo caso si utilizza un range di 16 milioni di colori per ogni pixel mentre nel secondo caso un range di 256 colori scelti come sottoinsieme di 16 milioni.

Per ovviare alla grande occupazione di memoria si sono sviluppati formati compressi in grado di ridurre notevolmente il numero di kbyte utilizzati dalle immagini.
Ci sono due metodi fondamentali di compressione:
- *lossless*: senza perdita di informazione;
- *lossy*: con perdita di informazione, ed è il più utilizzato.

Il primo metodo si applica a qualunque tipo di informazione rappresentata in binario e si basa sul riconoscimento delle sequenze di bit che si ripetono con maggiori e minori frequenze: le sequenze più frequenti vengono sostituite con codifiche più corte appositamente codificate, in modo da risparmiare spazio.
Vengono applicate nei compressori WinZIP, WinRAR e nella rappresentazione delle immagini in formato GIF (Graphics Interchange Format), PNG (Portable Network Graphics) e TIFF (Tagged Image File Format).

Il formato GIF è particolarmente utilizzato per le applicazioni in Internet in quanto offre una buona rappresentazione visiva dell'immagine anche di grandi dimensioni occupando poco spazio e quindi consentendo rapide visualizzazioni. Un file GIF può contenere più immagini visualizzate da alcuni software (i browser) in sequenza, che permettono di ottenere semplici animazioni (GIF animate) e uno sfondo trasparente.
Purtroppo questa codifica ha dei limiti:

- le immagini GIF possono utilizzare al massimo 256 colori differenti;
- la stampa di tali immagini è di pessima qualità.

Zoom su...

ALGORITMI DI COMPRESSIONE

I principali algoritmi di compressione sono:

- RLE (Run-Length-Encoding): utilizzato esclusivamente per la compressione di immagini raster, sostituisce ogni sequenza di byte di valore identico con due soli byte, ovvero il numero di byte ripetuti e il relativo valore;
- LZ (Abraham Lempel e Jakob Ziv): utilizzato per la compressione di qualunque tipo di documento binario, memorizza in una tabella (dizionario) le configurazioni di valori che si ripetono frequentemente nella sequenza originaria, per poi sostituire ogni loro occorrenza con il relativo indirizzo nel dizionario.

Il secondo metodo si applica generalmente a dati multimediali, in quanto sfrutta la caratteristiche della biologia dei sistemi sensoriali umani: in base ai limiti delle capacità percettive dell'uomo è possibile alterare alcune caratteristiche del segnale originario, al fine di ridurre le dimensioni della sua rappresentazione binaria, mantenendo tuttavia una qualità accettabile. Per esempio la retina ha una maggiore insensibilità alle variazioni di luminosità rispetto a quelle cromatiche: questo permette di trascurare differenze sufficientemente piccole di colore tra pixel vicini riducendo la quantità di sfumature e quindi, di conseguenza, la dimensione dell'immagine.
Il formato JPEG (Joint Photographic Expert Group) sfrutta questa tecnica, e serve per visualizzare immagini con più di 256 colori, o di considerevoli dimensioni. In tale formato è possibile definire il rapporto di compressione in modo da determinare la dimensione (e la qualità) delle singole immagini.

Immagine JPEG con qualità al 100% = 87 Kbyte

Immagine JPEG con qualità al 90% = 30,2 Kbyte

Immagine JPEG con qualità al 50% = 6,7 Kbyte

Questa tecnica di compressione viene utilizzata anche per codificare immagini in movimento nei formati MPEG (Moving Picture Experts Group) e WMV (Windows Media Video).

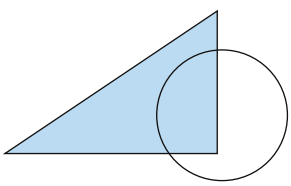
Riassumiamo nella tabella che segue le caratteristiche tipiche delle immagini secondo i tre principali utilizzi multimediali:

Tipo immagine	Definizione	Colori	Occupazione
Televisiva	720 × 625	256 (8bit)	440 kByte
Monitor di PC	1024 × 768	65.536 (16 bit)	1,5 Mbyte
Fotografica	15.000 × 10.000	16.000.000 (24 bit)	430 MByte

Immagini vettoriali

Una particolare rappresentazione delle immagini è quella vettoriale: utilizzata particolarmente per immagini di tipo geometrico, o per immagini riconducibili a insiemi di forme (punti, linee, rettangoli, cerchi), è caratterizzata dal fatto che non viene memorizzata l'immagine ma il procedimento per costruirla, conseguendo il doppio vantaggio di diminuire enormemente l'occupazione di memoria e di ottenere immagini facilmente ridimensionabili.

Ogni oggetto è quindi codificato attraverso un *identificatore* (per esempio polyline, circle ecc.) e alcuni *parametri* quali le coordinate del centro e la lunghezza del raggio (per la circonferenza) o le coordinate dei vertici (per il poligono), come riportato nella figura a lato. ▶



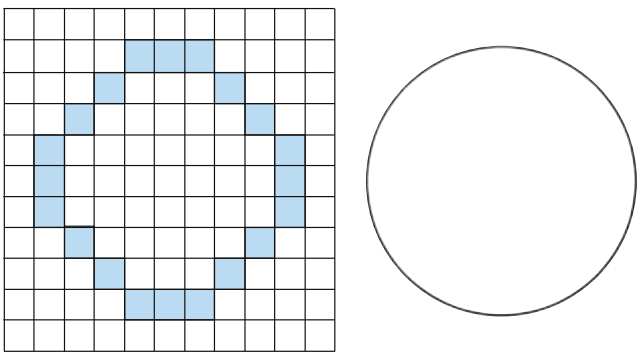
```
circle 98 66 50
polyline 0 48 88 152 88 48
```

Ricordiamo i seguenti formati:
- DXF (*Drawing Exchange Format*): strumenti di disegno tecnico;
- DWG: utilizzato da AutoCAD;
- CDR: utilizzato da Corel Draw;
- AI (*Adobe Illustrator*);
- WMF (*Windows MetaFile*);
- SVG (*Scalable Vector Graphics*): nativo in quasi tutti i browser moderni.

Il processo di visualizzazione di un'immagine vettoriale (cioè la trasformazione da una codifica matematica a una codifica raster) è detto rasterizzazione o rendering.

Questa rappresentazione non è però utilizzabile per immagini pittoriche, che non possono essere scomposte in elementi primitivi.

Le due immagini riportate a lato mostrano la differenza tra un'immagine raster e una vettoriale. ▶



Filmati digitali

Le immagini in movimento vengono rappresentate attraverso sequenze di immagini fisse (frame) visualizzate a una frequenza sufficientemente alta da consentire all'occhio umano di ricostruire il movimento.

Come per le immagini, anche per i filmati è necessario convertire in digitale ogni singolo fotogramma per poterlo memorizzare in digitale e quindi riconvertirlo in analogico per poi riprodurlo su uno schermo.
È possibile digitalizzare filmati provenienti da telecamere o da un sintonizzatore TV utilizzando apposite schede e software video, e successivamente ricostruire le animazioni di sintesi utilizzando sempre particolari programmi.
Lo spazio occupato da un'animazione dipende da molti fattori, tra i quali il più importante è il framerate: per comprenderne il funzionamento descriviamo brevemente come si ottiene l'animazione.
L'animazione è l'effetto che si ottiene sfruttando la tecnica cinematografica che si basa sul fatto fisiologico della persistenza di un'immagine sulla retina per un tempo relativamente lungo (1/10 di secondo). Scomponendo un movimento in un insieme di fasi successive (fotogrammi o frame) in modo che ciascuna fase succeda alla precedente in un periodo di tempo inferiore a quello di tale permanenza, ciascuna immagine si sovrappone alla precedente prima che questa scompaia dalla retina: ne consegue una visione continua del movimento che si traduce in un'unica impressione di moto, come avviene nella visione diretta.

IMMAGINI IN MOVIMENTO

Questa tecnica era già ben nota nell'antichità: Lucrezio, Leonardo e Newton conobbero e descrissero il fenomeno, disegnando immagini su dischi in movimento; via via si giunse poi a raggruppare le immagini su un nastro, fino ad arrivare alla moderna pellicola di celluloide.

FOTOGRAMMA

Si definisce fotogramma (frame) una singola immagine o disegno impiegato nella realizzazione di un'animazione.

Thomas Edison progettò il cinetoscopio dove le immagini si riproducevano alla frequenza di 48 fotogrammi al secondo, mentre i fratelli Lumière, nel loro cinematografo, ridussero tale numero a 16 immagini al secondo.
Più elevato è il numero di frame per secondo (*framerate*), migliore è la "qualità di movimento" del filmato: oggi il numero di fotogrammi varia da 4 al secondo per le moviole a 40 al secondo. Generalmente, viene usata la proiezione a 24 fotogrammi al secondo (la qualità TV è di 30 al secondo, sul Web scende a 15/10 al secondo).

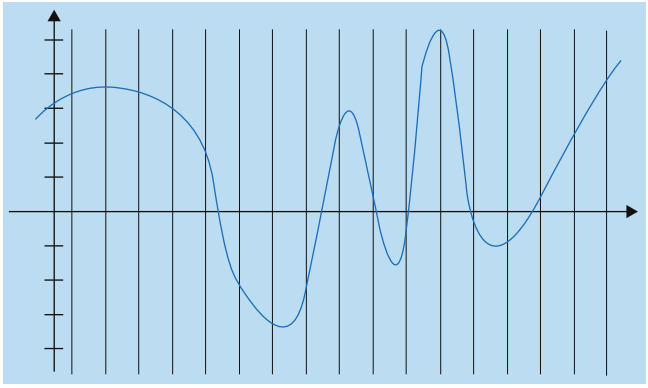
Per un filmato di qualità incidono, oltre al numero di frame, le seguenti proprietà:
- la dimensione dei fotogrammi;
- il numero di colori;
- la qualità dell'audio.

Infine incide anche la qualità del commento sonoro che viene codificato come un file audio e "sincronizzato" alle immagini.

Esistono diversi formati video: i più diffusi sono AVI (Audio Video Interleave), MPEG (Moving Picture Experts Group) e MOV (abbreviazione di *movie*), che si differenziano per la modalità, le caratteristiche di acquisizione e la tecnica di compressione utilizzata.

Suoni digitali

Oltre alle immagini è possibile rappresentare in modo digitale anche il suono: anche per esso deve essere effettuata una prima fase di conversione che trasforma il segnale analogico tipico dell'onda sonora in un segnale digitale, e successivamente si utilizzano codifiche particolari per la memorizzazione della musica digitale così ottenuta.



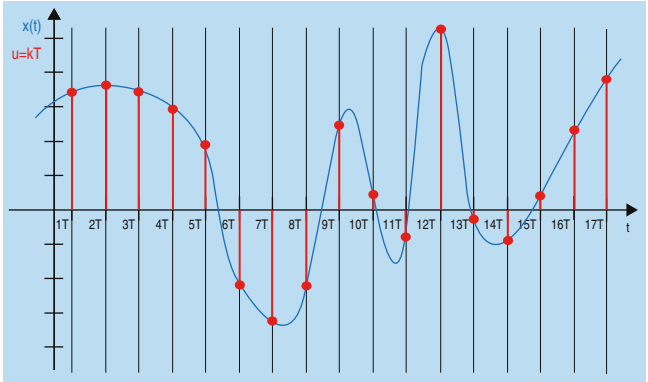
Il suono è un segnale analogico bidimensionale, cioè ha un'ampiezza in funzione del tempo.

In generale, per tradurre un qualunque segnale analogico nel corrispettivo digitale sono necessari i seguenti passaggi:

Il trasduttore (per esempio un microfono) trasforma l'onda sonora in un segnale elettrico e il campionatore effettua una discretizzazione, cioè a intervalli di tempo T regolari preleva campioni del segnale x(t).

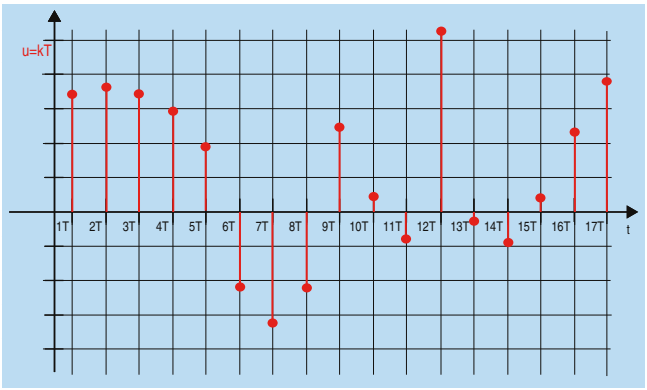
Il valore 1/T è detto frequenza di campionamento e rappresenta il numero di campioni del segnale x(t) acquisiti nell'unità di tempo.

$$f_c = \frac{1}{T}$$



Successivamente il quantizzatore converte il segnale acquisito dal campionatore in un segnale digitale introducendo sempre una perdita di informazione, che può essere definita sulla base del numero di bit a disposizione per rappresentare il valore letto (vengono definiti livelli di quantizzazione).

Nel nostro esempio introduciamo 16 livelli di quantizzazione, rispettivamente 8 per la semionda positiva e altrettanti per quella negativa, come si osserva nella figura seguente.



I sedici livelli di quantizzazione sono codificati con 4 bit: naturalmente all'aumentare del numero di bit aumenta il dettaglio e quindi la qualità del segnale digitalizzato.

DIGITALIZZAZIONE

Il procedimento di trasformazione da analogico a digitale finora descritto prende il nome di digitalizzazione.

La precisione con cui il segnale originario può essere ricostruito a partire dal segnale digitale, mediante una procedura detta di conversione digitale-analogica, può essere stabilita a priori e cresce all'aumentare della frequenza di campionamento e del numero di livelli di quantizzazione considerati.

Naturalmente all'aumentare della frequenza di campionamento aumenta il numero di informazioni lette così come la dimensione in bit e quindi l'occupazione di memoria: la frequenza minima di campionamento, cioè il numero minimo di "letture" al secondo che permette di non perdere segnale utile, è dettata dai teoremi di Shannon.
Il quantizzatore, invece, introduce sempre una perdita di informazione, che per essere contenuta richiede l'aumento del numero dei livelli di quantizzazione.

Le normali schede di digitalizzazione presenti nei personal computer permettono campionamenti che vanno da 8000 a 48.000 Hz e ogni valore campionato può essere rappresentato da 8, 16 o 32 bit.

Il formato WAV

Il formato WAV (contrazione di WAVEform audio file format) è lo standard utilizzato nei CD audio (Compact Disc Digital Audio, CDDA), per la registrazione audio digitale su com-

pact disc: il valore di campionamento è di 44.100 Hz con quantizzazione su 16 bit, quindi per ogni campione si possono avere 2^{16} valori di codifica con 44.100 campioni per ogni secondo di segnale.
Dato che in stereofonia si utilizzano due canali audio, per ogni secondo vi sono 88.200 campioni, ognuno dei quali richiede 16 bit di codifica: 16 × 88.200 = 1.411.200 bit, circa 1,4 Mbit per secondo.

BITRATE

Viene definita bitrate la quantità di bit al secondo necessaria per codificare un segnale.
Il bitrate di un segnale audio codificato con CDDA è 1,4 Mbit/s.

ESEMPIO

Calcoliamo la dimensione di un supporto per memorizzare un'ora di musica in digitale.
Un'ora è composta da 3600 secondi, quindi sono necessari 1,4 × 3600 = 5040 Mbit = 630 MB: questa motivazione è alla base della realizzazione dei CD-ROM, ovvero consentire di memorizzare un'ora di musica.
Oggi un CD-ROM ha una capacità di circa 700 MB, che equivalgono a quasi 70 minuti di registrazione audio.

Mp3

Verso la prima metà degli anni '90 venne presentato per la prima volta il formato Mp3 (implementazione del MPEG-1 Audio Layer III data) come formato di compressione dei dati particolarmente adatto per i file audio, ed ebbe subito un grande successo soprattutto in Internet in quanto presenta un rapporto di compressione altissimo che riduce di 12 volte l'ingombro dei file audio senza alterarne la qualità. L'algoritmo Mp3 si basa tra gli altri sul concetto di soglia di udibilità: poiché l'uomo percepisce i suoni entro un determinato spettro di frequenza, al di sotto e al di sopra del quale i suoni non vengono percepiti, si riesce a ottenere un notevole risparmio nella lunghezza dei file eliminando da un brano musicale tali suoni non udibili. Il bitrate di 128 kilobit al secondo è ritenuto di qualità accettabile per il formato Mp3, qualità che si avvicina a quella di un CD. Questo bitrate è il risultato di un tasso di compressione che si avvicina al rapporto di 11 a 1.

MIDI

Il MIDI (Musical Instrument Digital Interface) è uno dei formati in cui è possibile salvare la musica in forma digitale e con questo nome si indica sia il protocollo di trasmissione sia l'interfaccia hardware che serve a veicolare la trasmissione. Nasce nel 1983 per consentire la comunicazione e lo scambio di dati tra strumenti musicali di marche diverse, impensabile sino ad allora: tutti i sistemi analogici erano tra loro incompatibili e poco precisi.
L'avvento del personal computer ha favorito lo sviluppo di questo protocollo che, essendo in formato digitale, si presta naturalmente allo scambio di informazioni musicali tra strumenti, sintetizzatori musicali (reali e virtuali) e computer.
Proprio per la sua natura digitale il file MIDI ha dimensioni molto contenute: se confrontato con un file audio WAV di 30 MB il corrispondente file MIDI occupa 30 KB: infatti nel

file MIDI non è presente il suono (timbro), ma solamente il codice dello strumento e un numero che individua la nota e il suo valore musicale.

Invece di campionare il suono, un file MIDI registra gli eventi che generano il suono (per esempio quali tasti sono premuti su una tastiera, la durata, l'intensità ecc.).

La riproduzione avviene attraverso un sintetizzatore MIDI che genera i suoni corrispondenti alle informazioni che riceve: non è presente il cantato ma generalmente viene fatto eseguire da uno strumento solista oppure dal qualche emulatore che genera semplicemente "ooooh" oppure "aaaah", perciò un'ora di musica può occupare solamente circa 500 KB. La qualità di riproduzione dipende esclusivamente dal sintetizzatore (scheda audio) del computer che riproduce il suono.

Una nota di merito va ai produttori di file MIDI: non esiste infatti un programma che converta i file musicali WAV in MIDI, ma sono abili strumentisti (*midifilisti*) che senza partitura suonano a orecchio i pezzi su normali strumenti musicali dotati di sequencer, uno strumento che converte le note suonate in MIDI file.

Esempio riepilogativo

Calcoliamo l'occupazione di memoria di un filmato di un'ora non compresso: calcoliamo prima il video e successivamente aggiungiamo l'audio.

Video

Il framerate è 24 con 1920 × 1080 pixel in true color: quindi per ogni secondo servono 1920 × 1080 × 3 × 24 ~ 149 Mbyte (bitrate 149 × 8 ~ 1,2 Gbit/s), per un'ora otteniamo 149 × 3600 = circa 536 GB di spazio necessario.

Audio

In un sistema Dolby TrueHD l'audio è codificato con 8 canali, con frequenza di campionamento 96 kHz codificati su 24 bit: per ogni secondo di audio servono 24 × 96.000 × 8 ~ 18,4 Mbit (bitrate 18,4 Mbit/s), per un'ora di audio servono quindi 18,4/8 × 3600 ~ 8 GB di spazio.

Se non si utilizzasse la compressione, sapendo che un disco blu-ray può contenere 46 GB, dove potremmo memorizzare un film di un'ora che necessita di 544 GB di spazio?

Verifichiamo le conoscenze

Esercizi di completamento

1 Indica con una crocetta quali sono le codifiche dei formati audio, video o grafici.

Codifica	Audio	Grafici	Video
MP3			
GIF			
MIDI			
TIF			
AVI			
WAW			
BMP			
WMF			
MPEG			
MPG			
MOV			
PICT			

2 Il colore viene ottenuto da almeno tre colori di base, secondo un modello cromatico:
..... (....., ..., ...)
oppure da quattro colori base:
..... (....., ..., ..., ...)

3 Per avere una buona "qualità di movimento" del filmato il numero di frame per secondo (framerate) oggi varia da ..... al secondo per le moviole a ..... al secondo. Generalmente, viene usata la proiezione a ..... fotogrammi al secondo.

4 Un CD musicale di ..... MB contiene generalmente circa ..... minuti di musica in formato WAW e ..... ore in formato Mp3.

Test vero/falso

1	Un pixel (*picture element*) è l'unità elementare di memorizzazione di un'immagine.	V	F
2	Un pixel nelle immagini B/N viene memorizzato con 1 byte.	V	F
3	Con definizione di un'immagine si intende il numero di colori utilizzati per rappresentarla.	V	F
4	La sigla RGB si ottiene dalle iniziali dei colori rosso, giallo e blu.	V	F
5	Con la tecnica *lossless* si ha perdita di informazione.	V	F
6	Le immagini in formato JPEG (Joint Photographic Expert Group) sono compresse.	V	F
7	Il DWG ottimizza la rappresentazione bitmap per il disegno tecnico.	V	F
8	La trasformazione da analogico a digitale prende il nome di campionamento.	V	F
9	Con bitrate si intende la quantità di bit al secondo necessaria per codificare un segnale.	V	F
10	Il formato MIDI è un formato WAV compresso.	V	F
11	Un frame è una singola immagine o disegno di un filmato.	V	F
12	Con framerate si intende il numero di frame per secondo.	V	F

Verifichiamo le competenze

Esprimi la tua creatività

Esercizio guidato

Scatta una fotografia con una macchina fotografica digitale da 3 Mega Pixel (MP), quindi con una risoluzione pari a n = 2048 pixel per m = 1536 pixel.

Determina la quantità di memoria necessaria per salvare questa immagine nei casi in cui sia scattata in modalità:

- bianco/nero;
- scala di grigi;
- true color;
- con una palette da 256 colori.

Ne caso si voglia successivamente trasmettere WI-FI le immagini a un PC, determina il bitrate minimo necessario per trasferire queste immagini in 1, 4, 2 e 3 secondi.

Soluzione

a) bianco/nero

Lo spazio richiesto per la memorizzazione di un'immagine salvata in bianco e nero è m × n bit, ovvero

_____ × _____ circa = _____ byte

Il bitrate minimo necessario per trasferire quest'immagine in un secondo è

_____ / 1 = _____ bit/s

b) scala di grigi

Lo spazio richiesto per memorizzare un'immagine salvata in bianco e nero è m × n byte:

_____ × _____ circa = _____ byte

Il bitrate minimo necessario per trasferire quest'immagine in quattro secondi è

_____ / 4 = _____ bit/s

c) true color

Lo spazio richiesto per memorizzare un'immagine salvata in true color è

m × n × _____ byte, ovvero _____ × _____ × _____ circa = _____ byte

Il bitrate minimo necessario per trasferire quest'immagine in due secondi è

_____ / 2 = _____ bit/s

d) palette da 256 colori true color

Lo spazio richiesto per memorizzare un'immagine salvata in k = 256 colori in true color è

per la palette _____ byte + per l'immagine _____ byte = _____ byte

Il bitrate minimo necessario per trasferire quest'immagine in tre secondi è

_____ / 3 = _____ bit/s

2 MODULO I CODICI DIGITALI

UD 1 Codici digitali pesati

UD 2 Codici digitali non pesati

UD 3 La correzione degli errori

OBIETTIVI

- Comprendere le differenze tra codifica a lunghezza fissa e variabile
- Acquisire le tecniche di codifica con sistemi pesati
- Comprendere le motivazioni per l'utilizzo di codifiche non pesate
- Conoscere le codifiche per dispositivi dedicati
- Conoscere i sistemi di codifica in formato ottico
- Comprendere le tecniche di rilevazione e di correzione degli errori di trasmissione
- Conoscere i codici di Hamming

ATTIVITÀ

- Conoscere il codice ASCII e Unicode
- Codificare in codice BCD
- Eseguire somma e sottrazione in BCD
- Codificare in eccesso 3 e con il codice di Gray
- Codificare a sette segmenti e a matrice di punti
- Codificare e decodificare con QR Code
- Individuare l'errore con il codice di parità
- Correggere l'errore con byte di checksum
- Correggere l'errore con il codice di Hamming

UNITÀ DIDATTICA 1

CODICI DIGITALI PESATI

IN QUESTA UNITÀ IMPAREREMO...

- la codifica dei caratteri
- i codici BCD e packed BCD
- il codice eccesso 3
- il codice Aiken
- i codici quinario e biquinario
- il codice 2 su 5

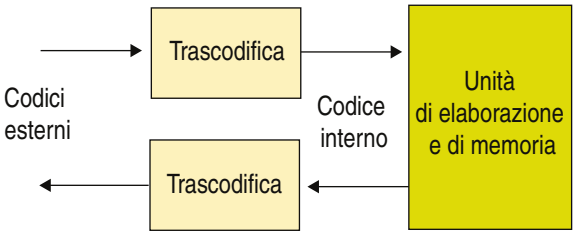
Introduzione

In un elaboratore elettronico devono essere memorizzati ed elaborati valori digitali, cioè valori che sono stati convertiti con segnali analogici da appositi convertitori analogici/digitali (A/D) a seconda della natura del segnale.
I valori digitali vengono codificati in una tabella che prende il nome di codice digitale, nella quale sono associati i simboli originari in maniera biunivoca a un nuovo simbolo che mantiene lo stesso contenuto informativo.
L'insieme dei simboli originari si chiama insieme sorgente (o esterno), l'insieme dei simboli codificati si chiama insieme codificato (o interno).

Dato che l'alfabeto dei simboli utilizzati per la codifica è binario, cioè è costituito da soli due simboli $E = (0, 1)$, il codice interno si dice codice binario.

CODICE BINARIO
Un codice si dice binario se l'alfabeto in codice è di tipo binario.

Una rappresentazione a blocchi del processo di codifica/decodifica è la seguente:



CODICE NON RIDONDANTE

Un codice si dice non ridondante se i simboli che vengono codificati sono in numero pari alle configurazioni ottenibili con le cifre binarie utilizzate.

Per i codici binari sappiamo che il numero di configurazioni è in funzione della lunghezza della parola codificata, cioè $2^{lunghezza}$.

Il codice interno è di norma non ridondante per minimizzare il numero di bit da elaborare e memorizzare.

Il codice esterno è di norma ridondante, per semplificare la generazione e l'interpretazione delle informazioni, ed è standardizzato, per rendere possibile la connessione di macchine (o unità di I/O) realizzate da costruttori diversi.

La possibilità di avere una ridondanza viene sfruttata per inserire nella codifica informazioni aggiuntive che consentono di individuare (e talvolta anche di correggere) eventuali errori causati da disturbi di trasmissione che hanno alterato il valore di qualche bit.

Possiamo classificare i codici anche in base alla lunghezza delle singole parole:
- codifica a lunghezza fissa: la lunghezza L delle parole codice associate ai valori dell'alfabeto sorgente è costante;
- codifica a lunghezza variabile: la lunghezza L delle parole codice associate ai valori dell'alfabeto sorgente è variabile.

Zoom su...

CODICI NELLA VITA DI OGNI GIORNO

Il codice può essere un'associazione del tutto arbitraria di parole codice a valori da codificare oppure può essere fondato su regole ben definite. Vediamo alcuni esempi di codice e regole di costruzione.

Tipo di codice	Modalità di costruzione
codice fiscale	algoritmo sui dati anagrafici
codice di avviamento postale	in base alla zona geografica
codice di matricola	progressivo generale
numero di targa	progressivo generale
partita IVA	in base alla tipologia e alla zona
numero di telefono	in base alla zona geografica e progressiva
codice sanitario	in base a un algoritmo

La codifica di caratteri: codici ASCII e Unicode

Il codice ASCII

Una parola o una frase in un calcolatore prende il nome di stringa di caratteri e viene rappresentata mediante una sequenza di bit. Sappiamo che i bit sono raggruppati in insiemi a lunghezza fissa e una prima codifica di tutto l'insieme dei simboli comunemente usati è stata effettuata utilizzando 7 bit. Una sequenza di 7 bit permette la rappresentazione di 2^7 = 128 configurazioni differenti. Il codice maggiormente usato che utilizza proprio 7 bit è il codice ASCII (*American Standard Code for Information Interchange*, codice standard americano per lo scambio di informazioni), riportato nella figura seguente:

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

In giallo vediamo i caratteri di controllo, in azzurro le cifre decimali, in verde le lettere maiuscole dell'alfabeto, in fucsia le lettere minuscole, in grigio tutte le rimanenti lettere.

Dalla specifica iniziale basata a 7 bit si è passati a un'estensione a 8 bit con lo scopo di raddoppiare il numero di caratteri rappresentabili. Il nuovo codice viene comunemente chiamato extended ASCII o high ASCII, nel quale sono state aggiunte le vocali accentate, simboli semigrafici e altri simboli di uso meno comune.

Vediamo di seguito alcuni esempi di codifica.

01001000	01100101	01101100	01101100	01101111	00101110
H	**e**	**l**	**l**	**o**	**.**

ESEMPIO

Proviamo ora a decodificare la seguente sequenza di bit:

011010010110110000000000011100000110111100101110

Per prima cosa è necessario separare la sequenza di bit in gruppi di 8 bit e successivamente assegnare a ogni gruppo il corrispondente carattere nella tabella ASCII:

01101001 01101100 00000000 01110000 01101111 00101110

Dalla tabella dei caratteri ASCII riconosciamo la frase **il po** .

Due semplici osservazioni:
- nella tabella i caratteri maiuscoli hanno una codifica diversa dai caratteri minuscoli;
- i caratteri O e I (o e i maiuscoli) hanno una codifica diversa dai numeri 0 e 1.

Bisogna prestare attenzione a non confonderli tra loro.

Il codice ASCII è non ridondante, perché i simboli che vengono codificati sono in numero pari alle configurazioni ottenibili con 7 cifre binarie.

Il codice Unicode

Un secondo codice per la codifica delle informazioni è il codice Unicode, che si basa sulla codifica del codice *ASCII esteso*, cioè del "vecchio" standard ASCII a 8 bit che consente la rappresentazione di 256 caratteri.
Con 256 configurazioni l'ASCII è sufficiente per gli alfabeti dell'Europa Occidentale e del Nord America, ma limitato per la rappresentazione di caratteri greci e latini, nonché per i simboli matematici, chimici, cartografici, per l'alfabeto Braille, gli ideogrammi ecc.
Si è quindi passati a un codice a 16 bit per cercare di soddisfare tutte le esigenze: attualmente lo standard Unicode non rappresenta ancora tutti i caratteri in uso nel mondo ma, essendo in evoluzione, forse in futuro arriverà a coprire tutti i caratteri rappresentabili.

Il codice Unicode, data la possibilità di codifica dei caratteri orientali, ha notevole diffusione nei linguaggi orientati al Web.
Per esempio, il linguaggio Java è un linguaggio che supporta Unicode.

ESEMPIO

Un testo di 400 caratteri che occupa 1600 bit è in formato Unicode?

Soluzione

Ogni carattere occupa 1600/400 = 4 bit, quindi non è Unicode: con 4 bit posso codificare 2^4 = 16 caratteri.
Se fosse codificato in Unicode con 1600 bit avrei solamente 1600/16 = 100 caratteri codificati.

È importante sottolineare che i codici Unicode da 0 a 255 corrispondono ai codici ASCII standard e quindi c'è compatibilità tra ASCII e Unicode.

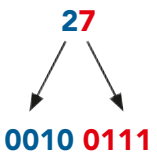
Il codice BCD (Binary Coded Decimal)

Un sistema che ci permette di rappresentare esclusivamente le dieci cifre decimali è la codifica *Binary Coded Decimal* (BCD).
Mediante questa codifica utilizziamo 4 bit per rappresentare le cifre decimali che vanno dallo 0 al 9 in codice binario ($2^4 = 16$, quindi abbiamo un codice ridondante in quanto 6 configurazioni sono inutilizzate).
Questa codifica è molto utilizzata in informatica codificando lo 0 con 0000 e il 9 con 1001. Per rappresentare altri simboli si possono usare sei combinazioni: a ogni cifra decimale sono associati 4 bit, come indicato nella tabella a lato:

Numero decimale	Numero BCD			
cifra/peso	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

ESEMPIO

Vediamo un esempio. La codifica del numero decimale 27 con il codice BCD è la seguente:



ESEMPIO

- Codifica in codice BCD il numero $(127)_{10} = (0001\ 0010\ 0111)_{BCD}$
- Codifica in codice BCD il numero $(379)_{10} = (0011\ 0111\ 1001)_{BCD}$

Non bisogna confondere il codice BCD con la codifica dei numeri nel sistema binario: i tre esempi riportati codificati in binario sono:
$27_{10} = 11011_2 = 0010\ 0111_{BCD}$
$127_{10} = 1111111_2 = 0001\ 0010\ 0111_{BCD}$
$379_{10} = 101111011_2 = 0011\ 0111\ 1001_{BCD}$
Il codice BCD codifica singolarmente ogni cifra, non l'insieme delle cifre: quindi solo le prime dieci configurazioni della codifica BCD coincidono con la codifica binaria.

Il codice BCD è un codice pesato: come possiamo vedere indicato nella tabella, ogni elemento del carattere, partendo da destra (bit meno significativo LSB) e procedendo verso sinistra (bit più significativo MSB), ha peso rispettivamente 1, 2, 4, 8.

Per esempio, $(0101)_{BCD} = 0*8 + 1*4 + 0*2 + 1*1 = (5)_{10}$.

Di tutte le possibili combinazioni ottenibili con quattro bit, solamente dieci vengono utilizzate. I restanti sei caratteri (1010, 1011, 1100, 1101, 1110, 1111) non hanno significato nel codice BCD: per questo motivo esso è ridondante.

Somma e sottrazione

Su questo codice anche le più semplici operazioni matematiche come la somma o la sottrazione, pur rispondendo alle normali regole aritmetiche, sono di difficile trattazione mnemonica, sia per la ridondanza del codice stesso che per le configurazioni non utilizzate che sono generalmente la causa di confusione e di errore.

Eseguiamo per esempio:

```
 5 +    0101 +
 8 =    1000 =
____   ______
13      1101
```

Con le normali regole della somma il risultato è 1101, ma tale valore non esiste in BCD: è infatti tra i caratteri privi di significato, mentre in codice binario puro equivale al numero 13. Il valore 13 espresso in BCD è invece 0001 0011.

La discordanza tra i due valori, dovuta alla ridondanza del codice, sta nel fatto che al risultato della somma, in BCD, occorre aggiungere $(6)_{10} = (0110)_{BCD}$ ogni volta che viene superato il numero 9, in quanto il salto che occorre fare per evitare caratteri privi di significato è di sei posizioni.

Riprendiamo l'esempio precedente: dato che la somma decimale supera il 10 è necessario correggere il risultato eseguendo la seguente operazione:

```
     1101 +        primo risultato
     0110 =        numero 6
___________
0001 0011
```

ESEMPIO

Vediamo un secondo esempio in cui sommiamo 5 + 5 in modo da ottenere 1010 che è una codifica non accettata:

```
25 +        0010  0101 +
 5 =               0101 =
___        ____________
30          0010  1010 +    1010 non ammesso
                  0110 =    sommo il numero  6
           ____________
            0011  0000
```

Lo stesso problema viene riscontrato nelle sottrazioni in BCD: è necessario correggere il risultato ogni volta che occorre un prestito dalla cifra precedente nello svolgimento della differenza decimale.

ESEMPIO

Vediamo un ultimo esempio:

33 –	0011 0011 –	
4 =	0100 =	
29	0010 1111 –	1111 non ammesso
	0110 =	sottraggo il numero 6
	0010 1001	ottengo 29

Packed BCD

Sappiamo che nei personal computer i dati sono memorizzati a gruppi di 8 (byte) e dato che il codice BCD utilizza solo 4 bit abbiamo due alternative:

1. memorizzare una cifra per byte e riempire i restanti quatto bit con zeri o uno (come nel codice EBCDIC dell'IBM), perdendo 4 bit per ogni byte: questa operazione prende il nome di *unpacked BCD*;
2. mettere due cifre per byte (questa modalità è chiamata *packed BCD*), aggiungendo un ulteriore quartetto di bit per indicare il segno. Il codice del segno è:
   - 1100 per il segno +
   - 1101 per il segno –

ESEMPIO

Codifichiamo il numero 127 in EBCDIC/IBM e in packed BCD.
La cifra 127 si rappresenta nel modo seguente:

- 11110001 11110010 11110111 in EBCDIC;
- 00010010 01111100 in packed BCD, dove 1100 è il segno +.

La codifica unpacked BCD in alcuni casi è preferibile nonostante comporti una notevole perdita di bit. Infatti in essa c'è una diretta corrispondenza tra i numeri e il codice ASCII: è sufficiente sostituire i primi quattro bit inutilizzati con 0011 per ottenere il corrispondente ASCII.

ESEMPIO

Passare dai seguenti numeri BCD a numeri ASCII:

Decimale	BCD	ASCII	POS. ASCII
3	0011	0011 0011	51
5	0101	0011 0101	53
7	1000	0011 1000	55

Il codice BCD è molto usato in elettronica, specialmente nei circuiti digitali privi di microprocessore, perché facilita la visualizzazione di lunghe cifre su display a cinque segmenti: infatti a ogni display fisico corrisponde esattamente una cifra.

Prova adesso!

• Codifica BCD

PRENDI CARTA E PENNA

1. Codifica in unpacked BCD i seguenti numeri: 16, 32, 513, 1024.
2. Codifica in packed BCD i seguenti numeri e confrontali con la codifica in base 2: 15, 31, 196, 255.

Il codice Aiken

Prima di descrivere questo codice, ricordiamo il significato di complemento a N di un numero:

COMPLEMENTO A N

Si dice complemento a N di un numero x il numero che si ottiene sottraendo da N il numero x.
Per esempio, in decimale, il complemento a 9 del numero 3 è il numero 9 – 3 = 6:

$C_9[3] = C_9[0011] = 1100 = (6)_{10}$
$C_9[7] = C_9[1101] = 0010 = (2)_{10}$

La principale caratteristica del codice Aiken sta proprio nel fatto che è un codice autocomplementante a 9, cioè complementando i bit di una cifra si ottiene il complemento a 9 della cifra stessa.
La codifica delle 10 cifre è riportata in tabella: ▶

Numero decimale	Numero BCD			
cifra/peso	2	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	0	1
8	1	1	0	1
9	1	1	1	0

Nella tabella a lato possiamo vedere i numeri a coppie che sommati danno il numero 9: ciascuno dei due numeri che compone la coppia è la negazione dell'altro. ▶

0	0000	9	1111
1	0001	8	1110
2	0010	7	1101
3	0011	6	1100
4	0100	5	1011

Il codice Aiken è chiamato anche codice 2421 per il peso che hanno le diverse cifre nella formazione del numero:

- le tre cifre meno significative hanno lo stesso peso del codice binario;
- la più significativa ha peso 2, invece di avere il peso 8 (2^3) del sistema posizionale.

Vediamo le codifiche con un solo bit di valore 1:
$b_3\ b_2\ b_1\ b_0$
0 0 0 1 = 1 ottenuto da un solo bit in posizione 0, cioè $2^0 = 1$
0 0 1 0 = 2 ottenuto da un solo bit in posizione 1, cioè $2^1 = 2$

0 1 0 0 = 4 ottenuto da un solo bit in posizione 2, cioè $2^2 = 4$
1 0 0 0 = 2 ha lo stesso peso del bit in posizione 1, cioè $2^1 = 2$: il valore di questa configurazione dovrebbe quindi essere 2, ma nella codifica esiste già una configurazione di valore 2 (0010) e quindi la codifica 1000 non è ammessa.

Come esempio convertiamo in decimale i seguenti numeri:
- 1011: sostituendo a ogni bit a 1 il corrispondente valore si ha 2 + 0 + 2 + 1 = 5
- 1110: sostituendo a ogni bit a 1 il corrispondente valore si ha 2 + 4 + 2 + 0 = 8
- 1111: essendo ogni bit al valore 1 otteniamo 2 + 4 + 2 + 1 = 9

Riassumendo, questo tipo di codifica gode della proprietà di autocomplementazione, cioè per passare dalla cifra decimale n al suo complemento a 9 (9 – n) basta passare al complemento a 1 della corrispondente codifica Aiken di n.
Si noti che per tale proprietà le stringhe equidistanti della tabella di codifica hanno i bit invertiti.

I codici quinario e biquinario

Quinario

Il codice quinario è un codice a lunghezza fissa di 4 bit con pesi 5-4-2-1, in grado di codificare 5 + 4 + 2 + 1 + 1 = 13 simboli: si tratta quindi di un codice pesato e ridondante.

Numero decimale	5421
0	0000
1	0001
2	0010
3	0011
4	0100
5	1000
6	1001
7	1010
8	1011
9	1100
———	———
10	1101
11	1110
12	1111

Osservando la tabella a lato si può notare quale sia la caratteristica principale di questo tipo di codifica: a parte la colonna del bit di peso 5, le prime cinque configurazioni sono identiche ordinatamente alle cinque sottostanti. L'unica differenza consiste appunto nell'inversione di tale bit nelle combinazioni da 5 a 9 in cui cambia solo MSB. ▶

Questa caratteristica potrebbe essere opportunamente sfruttata per agevolare alcune operazioni macchina (per esempio, dimezzare i comandi, ottimizzare le funzioni della tastiera ecc.).
Potrebbe anche essere utilizzata in applicazioni particolari dove l'alfabeto sorgente è di 12 cifre (per esempio i mesi dell'anno).

Biquinario

Un codice molto particolare è il codice biquinario in quanto utilizza 7 bit suddividendoli in due gruppi, uno da 2 bit e l'altro da 5.

È un codice pesato in entrambi i gruppi che sono rispettivamente di peso 5-0 e 4-3-2-1-0.

La principale caratteristica è che in ogni gruppo è presente uno e un solo bit impostato a 1: per ottenere questa particolarità sono stati introdotti due bit con peso 0, che servono

unicamente a "uniformare" il conteggio dei bit a valore 1 di entrambi i sottogruppi.
Lo possiamo vedere nella tabella a lato: ▶

Dec.	50	43210
0	01	00001
1	01	00010
2	01	00100
3	01	01000
4	01	10000
5	10	00001
6	10	00010
7	10	00100
8	10	01000
9	10	10000

Questo tipo di codifica si dice ad autoverifica con conteggio fisso oppure a rilevazione d'errore: per ogni cifra trasmessa o elaborata viene effettuata una verifica per controllare che gli "1" presenti nella stringa siano due, uno per ciascun gruppo in cui sono ripartiti i 7 bit del codice (conteggio fisso); in caso contrario sarà rilevata una condizione di errore.

Il codice 2 su 5

Simile al codice biquinario e indicato anche come 2/5, è ridondante a lunghezza fissa di 5 bit con pesi 6-3-2-1-0, dove il bit di peso nullo è non significativo e utilizzato al fine di mantenere nel codice esattamente due bit a "1" per ogni stringa come nel codice biquinario.
Con questo codice sarebbe possibile rappresentare 6 + 3 + 2 + 1 + 0 + 1 = 13 configurazioni, ma per mantenere la condizione di due soli 1 in ogni stringa del codice si riducono a 10.

Dec.	63210
0	00110
1	00011
2	00101
3	01001
4	01010
5	01100
6	10001
7	10010
8	10100
9	11000

La rappresentazione dello 0 è "anomala" in quanto, calcolando i pesi delle cifre a 1, dovrebbe avere come decodifica il numero 3: il codice 2/5 è quindi posizionale tranne per la codifica dello 0.

Anche questo codice, come il codice biquinario, è ad autoverifica con conteggio fisso, ovvero a rilevazione d'errore.

Conclusioni

Riportiamo nella tabella a lato la codifica dei numeri da 0 a 9 nei diversi codici descritti sino ad ora, in modo da poterli confrontare meglio.

Decimale	8421	AIKEN	Quinario	2 SU 5	Biquinario
0	0000	0000	0000	00110	0100001
1	0001	0001	0001	00011	0100010
2	0010	0010	0010	00101	0100100
3	0011	0011	0011	01001	0101000
4	0100	0100	0100	01010	0110000
5	0101	1011	1000	01100	1000001
6	0110	1100	1001	10001	1000010
7	0111	1101	1010	10010	1000100
8	1000	1110	1011	10100	1001000
9	1001	1111	1100	11000	1010000

Verifichiamo le conoscenze

1 Quale tra i seguenti codici ha lunghezza fissa?

- ○ il codice fiscale
- ○ il codice di avviamento postale
- ○ il codice di matricola
- ○ la partita IVA
- ○ il numero di telefono
- ○ il codice sanitario

2 Il codice ASCII:

- ○ ha lunghezza 7 bit
- ○ ha lunghezza 8 bit
- ○ ha lunghezza 16 bit
- ○ dipende dalle applicazioni

3 Il codice Unicode: (indica l'affermazione errata)

- ○ è compatibile con il codice ASCII
- ○ ha lunghezza 16 byte
- ○ è utilizzato nel Web
- ○ non è in stato ancora completamente definito

4 A quale numero corrisponde la codifica BCD 00011000?

- ○ 9
- ○ 11
- ○ 18
- ○ 24
- ○ nessuno dei precedenti

5 A quale numero corrisponde la codifica Aiken 00101100?

- ○ 26
- ○ 28
- ○ 32
- ○ 44
- ○ nessuno dei precedenti

6 A quale numero corrisponde la codifica quinario 01110001?

- ○ 26
- ○ 28
- ○ 32
- ○ 44
- ○ nessuno dei precedenti

7 A quale numero corrisponde la codifica biquinario 010001?

- ○ 2
- ○ 5
- ○ 11
- ○ 17
- ○ nessuno dei precedenti

8 A quale numero corrisponde la codifica 2 su 5 10001?

○ 2; ○ 5; ○ 6; ○ 7; ○ nessuno dei precedenti

Test vero/falso

1	Un codice si dice binario se l'alfabeto in codice è di tipo binario.	V F
2	Un codice si dice non ridondante se i simboli che vengono codificati sono in numero minore o uguale alle configurazioni ottenibili con le cifre binarie utilizzate.	V F
3	Il codice ASCII è un codice ridondante.	V F
4	Il codice Unicode si basa sulla codifica del codice ASCII esteso.	V F
5	Il codice BCD è un codice pesato con peso 2421.	V F
6	La codifica 1010 in codice Binary Coded Decimal corrisponde al valore 10.	V F
7	Nella codifica BCD è possibile effettuare le operazioni solo se non hanno riporto.	V F
8	Il codice EBCDIC è un codice BCD compattato.	V F
9	Il codice Aiken è un codice autocomplementante a 9.	V F
10	Il codice Aiken è un codice pesato con peso 8421.	V F
11	Il codice quinario è un codice pesato con peso 5421.	V F
12	Il codice quinario permette di rappresentare 16 configurazioni diverse.	V F
13	Il codice biquinario è un codice pesato con peso 5421.	V F

Verifichiamo le competenze

Esprimi la tua creatività

1 **Esegui la codifica delle seguenti stringhe in ASCII e Unicode.**

	ASCII	Unicode
Zio		
Alì Baba		
Fiat 127		
Zero0		
UNO1uno		

2 **Codifica i seguenti numeri in tutte le modalità conosciute.**

Decimale	8421	Aiken	Eccesso 3	2/5	Biquinario
5					
12					
21					
38					
69					

3 **Esegui le seguenti operazioni BCD.**

47 + ... +
12 = ... =
—— ————————
59 ....

47 + ... +
23 = ... =
—— ————————
70 ....
..... sommo
—— ————————
70 ....

78 – .... –
24 = ... =
—— ————————
14 .... –

22 – .... –
4 = ... =
—— ————————
19 .... –
..... = sottraggo
—— ————————
....

4 **Associa il codice corrispondente alle codifiche seguenti.**

Decimale		Codice
0	0000	
2	**0010**	
4	0100	
6	1100	
8	1000	

Decimale		Codice
1	00011	
3	0011	
5	1000001	
7	10010	
9	1100	

UNITÀ DIDATTICA 2

CODICI DIGITALI NON PESATI

IN QUESTA UNITÀ IMPAREREMO...

- il codice eccesso 3
- la codifica di Gray
- il codice 1 su N
- il codice a sette segmenti
- il codice a matrice di punti
- il barcode e il QR Code

Generalità

I codici non posizionali sono caratterizzati dal fatto che non esiste un peso per cui moltiplicare i singoli bit all'interno della stringa di codifica.
Le singole configurazioni delle diverse codifiche hanno una costruzione particolare definita per soddisfare particolari campi di applicazioni, generalmente in ambito industriale.

Il codice eccesso 3

Un primo codice che permette la rappresentazione dei valori decimali è il codice eccesso 3.

Ogni numero viene ottenuto dal corrispondente valore BCD sommando 3: in pratica la quantità 0 viene codificata come se si trattasse del valore 0 + 3 = 3, la quantità 1 viene codificata con il valore 4 e infine la quantità 9 viene codificata come se si trattasse del valore 12, e così via.

Le dieci cifre sono riportate nella tabella a lato: ▶

Numero decimale	Numero binario			
	b3	b2	b1	b0
0	0	0	1	1
1	0	1	0	0
2	0	1	0	1
3	0	1	1	0
4	0	1	1	1
5	1	0	0	0
6	1	0	0	1
7	1	0	1	0
8	1	0	1	1
9	1	1	0	0

Il codice eccesso 3 non è un codice pesato in quanto non esistono relazioni tra la posizione dei bit e il peso del bit stesso nella configurazione, come abbiamo visto per il codice binario (8421) e Aiken (2421).

Il codice eccesso 3 è un codice autocomplementante, così come il codice Aiken, ed è un codice ridondante in quanto non sono utilizzate tutte le 16 possibili configurazioni.

Nella tabella seguente possiamo vedere come i numeri a coppie, sommati, diano il numero 9: ciascuno dei due numeri che compone la coppia è la negazione dell'altro:

0	0011	9	1100
1	0100	8	1011
2	0101	7	1010
3	0110	6	1001
4	0111	5	1000

Somma

Come esempio di operazione in codice eccesso 3 vediamo solo la somma, dato che questa codifica raramente viene utilizzata nel caso in cui sia necessario effettuare operazioni algebriche.

La somma è simile alla BCD, ma bisogna sottrarre 3 (in binario) se la somma non ha riporto (carry), altrimenti sommare 3 (in binario) a entrambe le cifre.

ESEMPIO 1 ***Eseguiamo la somma di 2 + 5***

```
 2 +   0101 +
 5 =   1000 =
—      ————
 7     1101    risultato errato
```

Dato che non abbiamo avuto riporti (7 è minore di 9) sul MSB dobbiamo sottrarre 3 (in binario) al risultato:

```
 1101 –     (primo risultato )
 0011 =     (numero 3 in binario)
 ————
 1010       numero 7: risultato corretto
```

ESEMPIO 2 ***Eseguiamo la somma di 2 + 8***

```
 2 +   0101 +
 8 –   1011 =
—      ————
xx    10000   risultato errato dato che supera il 9
```

Il numero quindi deve essere rappresentato da due cifre e a entrambe devo sommare il valore 3:

```
    1  0000 +  (primo risultato)
0011  0011 =  (numero 3 in binario)
-------------
0100  0011    numeri 1 e 0 rappresentati in eccesso 3
```

Analoghi problemi li abbiamo con l'operazione di sottrazione, che lasciamo al lettore come approfondimento.

La codifica di Gray

È un codice numerico binario che prende anche il nome di codice ciclico in quanto due cifre successive differiscono solamente di 1 bit: si dice che hanno distanza unitaria (distanza 1).

DISTANZA

Con distanza in un codice si intende il numero di bit che "mutano" tra due configurazioni adiacenti.

È stato progettato e brevettato nel 1953 nei laboratori Bell dal ricercatore Frank Gray per poterlo utilizzare nell'acquisizione di lettura di posizione di particolari dispositivi elettronici (per esempio encoder di posizione utilizzati nei regolatori di volume digitali degli impianti hi-fi).

Zoom su...

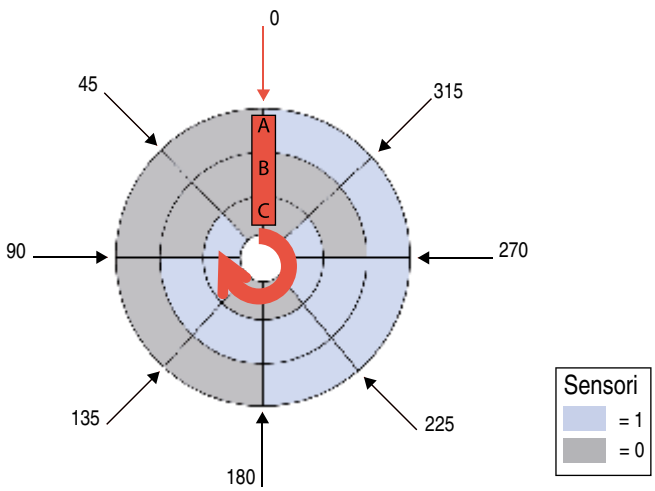
ENCODER

Il termine encoder è utilizzato in elettronica per indicare un dispositivo elettromeccanico che esegue la funzione di trasduttore di posizione angolare: esso converte la posizione angolare del suo asse rotante in numerici digitali che indicano il valore dell'angolo (chiamato resolver).
Esistono principalmente tre tipi di trasduttori angolari con uscita digitale: l'encoder incrementale, quello tachimetrico e quello assoluto.

A causa delle tolleranze meccaniche è improbabile che due o più bit di una cifra possano commutare esattamente nello stesso istante; viene quindi a crearsi un periodo intermedio in cui è codificato un valore indesiderato. Negli encoder che utilizzano questo codice il

passaggio da un valore al successivo (o precedente) comporta la commutazione di un unico circuito, eliminando ogni possibile valore equivoco.

Angolo	ABC
0-45	000
45-90	001
90-135	001
135-180	010
180-225	110
225-270	111
270-315	101
315-360	100



Per ottenere il codice di Gray è possibile ricorrere alla regola della specularità: a partire dalla combinazione (0 1) è possibile costruire il codice per successive operazioni speculari, aggiungendo poi uno 0 per ogni cifra al di sopra della linea di specularità e un 1, sempre per ogni cifra, nella parte inferiore.
Queste cifre vanno man mano aggiunte nella parte sinistra del numero che si ottiene.

Nel passare da una parola a quella successiva cambia un solo bit, vengono minimizzati gli errori nel passaggio da uno stato al successivo e aumenta la velocità dell'ALU.

1 bit	2 bit	3 bit	4 bit
0	0 0	0 00	0 000
1	0 1	0 01	0 001
		0 11	0 011
	1 1	0 10	0 010
	1 0		0 110
		1 10	0 111
		1 11	0 101
		1 01	0 100
		1 00	
			1 100
			1 101
			1 111
			1 110
			1 010
			1 011

Il codice eccesso 3 riflesso

Il codice eccesso 3 riflesso si ottiene dal codice eccesso 3 eseguendo due operazioni:
1. una operazione di shift a destra di una posizione;
2. la somma del numero ottenuto al numero di partenza, senza però sommare i riporti.

Vediamo due esempi:

1. codificando il numero 3_{10} partendo dalla codifica in eccesso 3

```
Dec.   Eccesso 3    Shift a Dx   Somma
 3       0110         0011       0110+
                                 0011=
                                 -----
                                 0101   <--- eccesso 3 riflesso
```

2 codifichiamo il numero 7_{10} partendo dalla codifica in eccesso 3

```
Dec.   Eccesso 3    Shift a Dx   Somma
 7       1010          0101       1010+
                                  0101=
                                  -----
                                  1111   <--- eccesso 3 riflesso
```

La seguente tabella riporta tutti i numeri codificati in eccesso 3 riflesso

Dec.	Ecc.3 R.
0	0010
1	0110
2	0111
3	0101
4	0100
5	1100
6	1101
7	1111
8	1110
9	1010

Possiamo osservare che il codice ottenuto è un codice progressivo come il Gray, cioè la distanza tra due configurazioni adiacenti è unitaria.

Codice BCD di Petherick

Il codice di Petherick è anch'esso un codice progressivo non pesato ed ha la caratteristica di non contenere le configurazioni 0000 e 1111.
Ne esistono due versioni, riportate nelle tabelle seguenti.

Dec.	Petherick
0	0010
1	0110
2	0100
3	0101
4	0001
5	1001
6	1101
7	1100
8	1110
9	1010

Dec.	Petherick
0	0101
1	0001
2	0011
3	0010
4	0110
5	1110
6	1010
7	1011
8	1001
9	1101

Codici progressivi: tabella riepilogativa

Oltre a quelli descritti esistono altri codici progressivi che come il codice di Pethernick non sono pesati e non hanno una "regola diretta" che ci permette di ricavarli ma sono ottenuti manualmente, togliendo e/o aggiungendo un bit a partire da una configurazioni.

Ricordiamo il codice Glixon , il codice O'Brien e il codice Tompkins che riportiamo nella seguente tabella.

parola in codice	Codici progressivi								
	codice Gray	codice Glixon	codice O'Brien		eccesso 3 riflesso	codice Tompkins		codice Petherick	
0	0000	0000	0000	0001	0010	0000	0010	0101	0010
1	0001	0001	0001	0011	0110	0001	0011	0001	0110
2	0011	0011	0011	0010	0111	0011	0111	0011	0100
3	0010	0010	0010	0110	0101	0010	0101	0010	0101
4	0110	0110	0110	0100	0100	0110	0100	0110	0001
5	0111	0111	1110	1100	1100	1110	1100	1110	1001
6	0101	0101	1010	1110	1101	1111	1101	1010	1101
7	0100	0100	1011	1010	1111	1101	1001	1011	1100
8	1100	1100	1001	1011	1110	1100	1011	1001	1110
9	1101	1000	1000	1001	1010	1000	1010	1101	1010
10	1111								
11	1110								
12	1010								
13	1011								
14	1001								
15	1000								

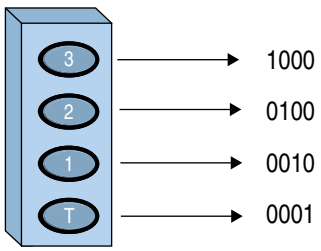
Il codice 1 su n

Tra i codici posizionali abbiamo visto il codice 2/5, la cui caratteristica fondamentale era quella di avere il numero di 1 costante all'interno di ogni configurazione.
Questa particolarità è presente anche nel codice 1 su n, che associa a ognuna delle n possibili configurazioni di una stringa di n bit avente un solo bit a 1.

Il fatto che non venga indicato il numero di cifre che compongono le stringhe della codifica ma che siano indicate genericamente con n indica la particolare flessibilità di questo codice, che viene utilizzato in dispositivi elettronici, quali per esempio i calcolatori tascabili.

Per esempio, viene utilizzato per immettere dati attraverso la tastiera numerica, decodificandoli all'interno mediante logica cablata.

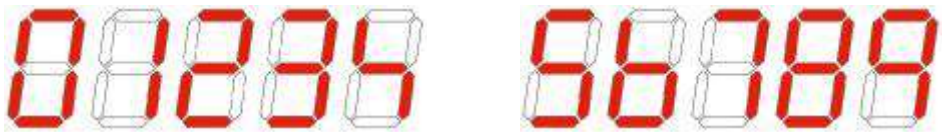
Come si può vedere nella figura a lato, viene anche utilizzato negli ascensori, impiegato per visualizzare la posizione dei piani raggiunti e per selezionare il piano da raggiungere: le configurazioni sono 4 e quindi il codice è 1 su 4.



Il codice a sette segmenti

Un codice particolare che viene utilizzato nei display per consentire la rappresentazione grafica di cifre decimali è il codice a sette segmenti. Utilizza 7 bit e a ogni bit è associato un segmento del display: dalla sua combinazione si ottengono le codifiche dei 10 simboli decimali.

Le 10 cifre sono ottenute dall'accensione dei segmenti riportati nella figura seguente:

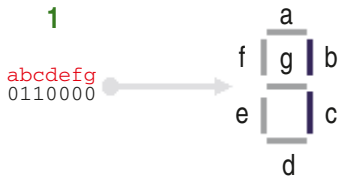


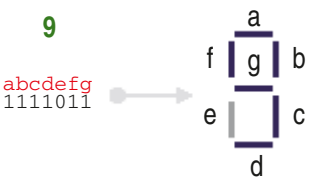
I sette segmenti sono indicati con le prime sette lettere dell'alfabeto (a, b, c, d, e, f, g) come indicato nella figura a lato:

Viene definita una corrispondenza tra il segmento da accendere e il bit che viene settato al valore 1 nella configurazione:

segmento	bit
a	1000000
b	0100000
c	0010000
d	0001000
e	0000100
f	0000010
g	0000001

Vediamo, per esempio, il numero 1 e il numero 9:



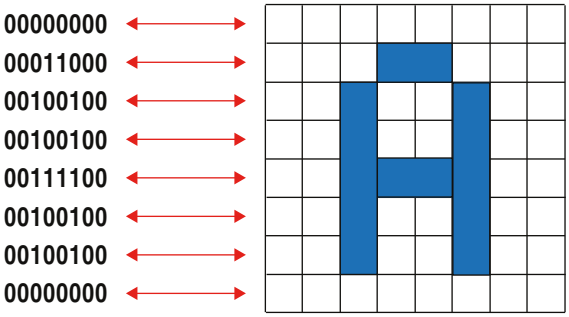


Il codice è ridondante in quanto ammette 32 configurazioni: generalmente viene esteso e utilizzato per rappresentare gli ulteriori 6 simboli del codice esadecimale: A, B, C, D, E, F.

Il codice a matrice di punti

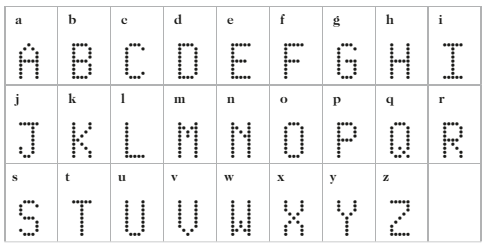
Il codice a matrice di punti è impiegato in tutti i dispositivi che utilizzano i pixel per visualizzare i dati, come i monitor, i display dei telefonini, i display delle calcolatrici ecc.
I caratteri sono rappresentati mediante una matrice M × N di bit in modo da consentire la rappresentazione di simboli grafici su una matrice di punti di M righe e N colonne.

Vediamo un esempio di come è possibile codificare la lettera A:



Nel nostro esempio abbiamo impiegato una matrice con 8 bit per ogni riga, in modo da utilizzare un intero byte e decodificarlo semplicemente: in sistemi dedicati la dimensione è comunque variabile e dipende unicamente dalla qualità delle informazioni che si devono visualizzare.

Nella figura seguente è riportato un esempio di alfabeto completo codificato con una matrice di punti 7 × 5.



Barcode e QR Code

Concludiamo questa unità con una breve trattazione dei due principali sistemi di codifica dell'informazione in formato ottico, che permettono di ottenere la lettura automatizzata mediante dispositivi di scansione (scanner) specialmente con tecnologia laser.

Barcode

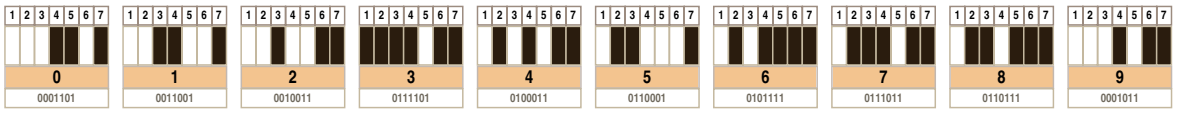
Un codice a barre (*barcode*) è la traduzione ottica di un codice numerico o alfanumerico che definisce e individua una particolare entità o prodotto.
Il codice a barre consente ai lettori ottici, collocati alle casse dei punti vendita, di registrare automaticamente i prodotti in uscita (marca, tipo, prezzo) scaricandoli quindi dalla contabilità del magazzino e fornendo il conto dettagliato della spesa al singolo acquirente.

ORIGINE DEL CODICE A BARRE

L'origine del codice a barre si deve all'esigenza di creare un sistema per accelerare il "check-out" al supermercato ed eliminare l'errore "umano" delle cassiere con il crescere, durante gli anni '70, della varietà e della tipologia dei prodotti alimentari in vendita nei negozi: nel 1973 è nato negli Stati Uniti il primo codice a barre, l'UPC (*Universal Product Code*).

La codifica del codice alfanumerico avviene mediante un'alternanza di barre verticali e di spazi disposti in modo tale da essere letti semplicemente da un dispositivo ottico.
I numeri estremi assumono una particolare codifica univoca, in modo che il lettore ottico sia in grado di riconoscere se un'etichetta viene letta in modo "diritto" o a rovescio.

Nelle figure seguenti sono riportate come esempio le codifiche per il primo numero di sinistra



e dell'ultimo numero di destra: si può notare come gli stessi numeri vengano codificati in modo complementare e univoco.

A seconda delle diverse esigenze di mercato e del tipo di prodotto possono essere utilizzate diverse tipologie di composizione di codici a barre (dette anche "simbologie"): esse offrono una gamma di alternative per le diverse esigenze riguardo al numero delle informazioni che si vogliono inserire, alla lunghezza del codice a barre ecc.

Per la distribuzione a livello internazionale esiste un sistema univoco, definito dalla specifica GS1, per poter codificare i prodotti più diffusi. Oggi sono oltre cento le organizzazioni commerciali aderenti in 103 nazioni sparse in tutti i continenti e oltre 1.000.000 di imprese associate utilizza questo standard.
La formulazione utilizzata è l'EAN 13, così denominata perché formata da 13 elementi: è il codice di distribuzione commerciale più applicato a livello mondiale. Nei diversi paesi i codici vengono assegnati ai prodotti dalle organizzazioni nazionali di codifica presenti in ogni nazione, che sono responsabili dell'assegnazione dei codici e del rispetto delle regole a livello nazionale (in Italia è l'associazione Indicod-Ecr).

ESEMPIO 3 *Il codice ISBN*

Analogamente alla specifica GS1, per l'editoria viene assegnato a ogni libro un codice univoco, l'ISBN (*International Standard Book Number*): si tratta di un numero che identifica a livello internazionale in modo univoco e duraturo un titolo o un'edizione di un titolo di un determinato editore.

Nella figura a lato è riportato per esempio il barcode che individua il codice di un libro della nostra casa editrice. ▶

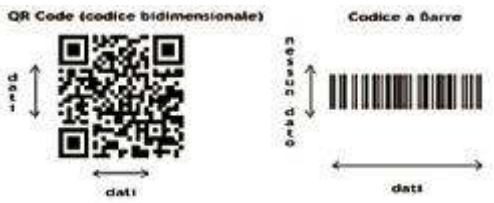
L'ISBN, a partire dal 1° gennaio 2007, è formato da un codice di 13 cifre, suddivise in 5 parti dai trattini di divisione.

- Prefisso EAN: sono le prime tre cifre del codice ISBN, introdotte a partire dal 2007; indicano che si è in presenza di un libro (codice 978 del sistema GS1).
- Gruppo linguistico: è l'identificativo del paese o dell'area linguistica dell'editore; può utilizzare da 1 a 5 cifre (per l'Italia è 88).
- Editore: è l'identificativo della casa editrice o del marchio editoriale; può utilizzare da 2 a 7 cifre (per la Hoepli è 203).

- Titolo: è l'identificativo del libro; può utilizzare da 1 a 6 cifre (in questo esempio è 4824).
- Carattere di controllo: è l'ultima cifra del codice ISBN e serve a verificare che il codice non sia stato letto o trascritto erroneamente (cosa che può sempre accadere, specialmente quando si usano strumenti automatici come i lettori di codici a barre).

QR Code

Nei codici a barre è possibile memorizzare pochi dati, in genere solo il codice del prodotto. Per superare questo limite nel 1994 in Giappone la compagnia Denso-Wave ha sviluppato la tecnologia QR Code (*Quick Response*, risposta rapida) utilizzando per la codifica dell'informazione un sistema a barre bidimensionale a matrice.



QURIFICARE

Con il neologismo qurificare si intende il processo di codifica di un'informazione in formato QR Code e l'operazione opposta, cioè la decodifica, prende il nome di dequrificazione.

I codici QR possono memorizzare fino a un massimo di 4296 caratteri alfanumerici e 7089 caratteri numerici. Hanno avuto una facile diffusione in quanto sono disponibili programmi gratuiti per *qurificare* qualsiasi testo o indirizzo web, come osserviamo nell'esempio riportato a lato, realizzato con un programma reperito in Internet:

I codici QR possono facilmente essere letti da qualsiasi telefono cellulare moderno in quanto sono disponibili molteplici applicazioni gratuite che scannerizzano e decodificano il messaggio.

Oggi sono spesso aggiunti alle pubblicità per memorizzare informazioni aggiuntive e il link al sito web di riferimento, in modo da agevolare la connessione da parte dell'utente che non deve più digitare l'indirizzo, ma semplicemente "puntare" la telecamera del telefonino per ritrovarsi istantaneamente collegato a Internet.

Zoom su...

CODICE REED-SOLOMON

Nei codici QR è utilizzato un meccanismo per rilevare e correggere i dati nel caso in cui il QR fosse in parte danneggiato, per esempio, per la presenza di macchie o rotture e graffi sul supporto cartaceo. Viene utilizzato il codice Reed-Solomon che consente di ricostruire fino al 30% delle informazioni danneggiate.

Verifica le competenze

Esprimi la tua creatività

1 **Esegui le seguenti operazioni in eccesso 3.**

3 + ... +
1 = ... =

_____
xx .....
(risultato errato dato che non supera il 9)

9 + ... +
7 – ..... =

_____
xx .....
(risultato errato dato che supera il 9)

..... –
..... =

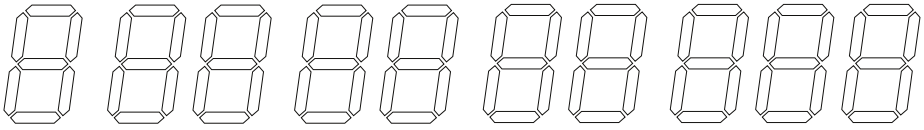
_____
.....

..... +
..... =

_____
.....

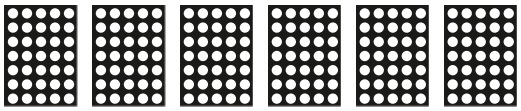
2 **Codifica i seguenti numeri con un codice a sette segmenti.**

3 24 71 60 859



3 **Codifica i seguenti numeri/lettere con una matrice di punti 7 x 5.**

4 C c I L 7



4 **Scarica un programma per leggere i QR Code e associali alle due frasi sotto riportate.**

I computer sono incredibilmente veloci, accurati e stupidi. Gli uomini sono incredibilmente lenti, inaccurati e intelligenti. L'insieme dei due costituisce una forza incalcolabile.
Albert Einstein (attribuito)

L'uomo più stupido è infinitamente più intelligente del miglior calcolatore, ma il calcolatore più stupido è infinitamente più logico dell'uomo più intelligente.

UNITÀ DIDATTICA 3

LA CORREZIONE DEGLI ERRORI

IN QUESTA UNITÀ IMPAREREMO...

- il concetto di distanza e distanza minima di un codice
- la differenza tra rilevazione e correzione degli errori
- la codifica con bit di parità
- il codice di Hamming

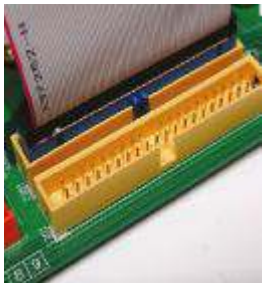
Introduzione

All'interno di un calcolatore tra unità di elaborazione e di memoria continuano ad avvenire scambi di informazioni in formato digitale: l'informazione che "si muove" nell'elaboratore consiste nella propagazione lungo un insieme di "fili paralleli" (il bus) di valori di tensione opportunamente settati con alto e basso, che rappresentano i segnali binari logici 1 e 0.

BUS

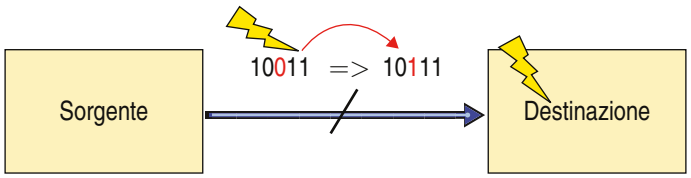
Con il termine bus si intende la connessione esistente tra due o più dispositivi di un PC per effettuare la trasmissione dei dati: la sua larghezza è costituita dal numero di conduttori che lo compongono e rappresenta il numero di bit che possono essere inviati contemporaneamente (in parallelo) da un componente all'altro (per esempio bus a 32 o 64 bit).

Nella figura a lato è riportato un bus parallelo utilizzato per il trasferimento dei dati dall'hard disk alle unità ottiche (CD e DVD).



Nella pratica vengono indicati come larghezza di un bus solamente i conduttori utilizzati per l'indirizzamento e i dati: come vedremo in seguito, un bus con architettura a 16 bit ha più di 16 fili in quanto sono necessari alcuni conduttori aggiuntivi per il controllo e la ridondanza.

Lo scambio digitale di informazioni tra sottosistemi è quindi costituito da un flusso di elettroni in movimento che generano un campo magnetico e sono soggetti a rumori proprio di natura elettromagnetica: questi rumori possono arrivare a sovrapporsi al segnale fino al punto di distorcere il valore iniziale, variando uno o più valori di tensione, e "consegnando" al destinatario un'informazione diversa da quella che era stata trasmessa.



Nell'esempio della figura un bit ha cambiato valore, cioè il valore iniziale 10011, trasmesso dal sistema mittente, viene interpretato 10111 dal sistema ricevente per effetto dei disturbi: in questo caso si dice che un bit si è sporcato.

Potrebbe inoltre verificarsi il malfunzionamento di un dispositivo o di un componente che potrebbe generare l'alterazione di un segnale.

È necessario introdurre un sistema che ci consenta innanzitutto di individuare la presenza di una situazione come quella descritta, in modo da accorgerci che l'informazione pervenuta è errata prima che possa provocare qualche "disastro". L'ideale sarebbe avere un meccanismo in grado di eseguire la correzione automatica di un errore, senza dover ritrasmettere tutta l'informazione.

In informatica è possibile ottenere entrambe le situazioni "arricchendo" il contenuto dei dati trasmessi con l'aggiunta di informazioni ridondanti specifiche per garantire la sicurezza della trasmissione e della corretta informazione.

Zoom su...

ERRORI E PROBABILITÀ

Qual è la probabilità che si verifichi un numero e di errori?
Detta p la probabilità che un singolo bit venga accidentalmente alterato, si può calcolare la probabilità che in un blocco di n bit vi siano contemporaneamente e errori.
Consideriamo una stringa di *n* bit (larghezza del bus) e supponiamo che l'evento di modifica di un bit (o *errore*) da parte di un disturbo:

- sia indipendente dalla posizione del bit nella stringa;
- si verifichi con probabilità pari a *p* (*tasso di errore*).

La probabilità che la stringa ricevuta contenga e errori è data da:

$$P_e = \binom{n}{e} \cdot p^e \cdot (1-p)^{n-e}$$

Se sostituiamo p = 1% otteniamo:

n	P_0	P_1	P_2	P_3
8	92,27%	7,46%	0,26%	0,005%
16	85,14%	13,76%	1,04%	0,049%

Osserviamo che per n = 8 le modifiche più probabili riguardano sostanzialmente un solo bit e per n = 16 riguardano in modo non trascurabile anche due bit.

In presenza di errore ci sono sostanzialmente quattro possibilità:
- scartare il messaggio;
- richiedere la ritrasmissione del messaggio;
- migliorare la qualità del canale per rendere uguale a 0 la possibilità di errore;
- effettuare la correzione.

Ci occuperemo della quarta possibilità, introducendo delle codifiche particolari che permettano al sistema digitale di "autocorreggersi".

Definizioni fondamentali

Abbiamo detto che all'informazione vengono aggiunti degli elementi che consentono di effettuare la rilevazione e la successiva correzione dell'errore; perché questo avvenga è necessario che il codice alla sorgente contenga *configurazioni non utilizzate*, disposte in modo tale che un errore trasformi una configurazione valida in una non utilizzata, e quindi sia riconoscibile in ricezione.

La condizione necessaria affinché si possa riconoscere un errore è la ridondanza del codice, ma vedremo che questa non sarà una condizione sufficiente.

DISTANZA

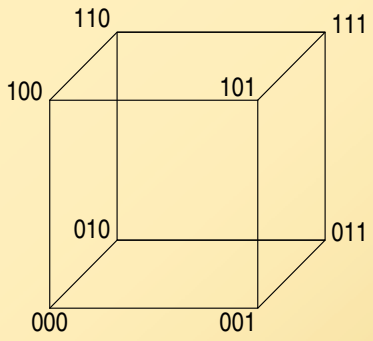
Con distanza di due configurazioni binarie si intende il numero di bit per cui le due configurazioni differiscono.

Vediamo alcuni esempi, dove indichiamo con D(A,B) la distanza tra due configurazioni binarie di 4 bit:

$$D(1000,1010) = 1$$
$$D(0110,0000) = 2$$
$$D(0100,1010) = 3$$
$$D(0100,0011) = 3$$

Essa prende il nome di distanza di Hamming.

Per avere chiaro il concetto di distanza è molto utile avvalersi del reticolo di Hamming; vediamo un esempio con un codice a 3 bit:



Visivamente la distanza tra due codici è data dal minor numero di lati che occorre percorrere per andare dal primo codice al secondo.
Per esempio per andare da 000 a 101 la strada più breve è passare da due segmenti, e infatti questi codici hanno distanza d = 2.

Operativamente per calcolare la distanza tra due codifiche si esegue l'operazione di XOR tra i due codici e si contano quanti numeri 1 ha il risultato.

Per esempio, se

s1	10001001
s2	10110001
s1 XOR s2	00111000

la distanza di Hamming tra s1 e s2 è pari a 3.

DISTANZA MINIMA DI UN CODICE

Si definisce distanza minima di un codice la minima distanza che intercorre tra due configurazioni qualunque del codice.

Per esempio, il codice ASCII ha

$$DMIN\ (Codice\ ASCII) = 1$$

in quanto esistono configurazioni che differiscono per un solo bit.

I codice non ridondanti hanno DMIN = 1 mentre i codici ridondanti devono avere DMIN > 1.

Un codice impiegato per la rilevazione di tutti i possibili errori singoli, o SEDC (*Single Error Detection Code*), deve non utilizzare le configurazioni che distano uno da ciascuna delle configurazioni utilizzate.

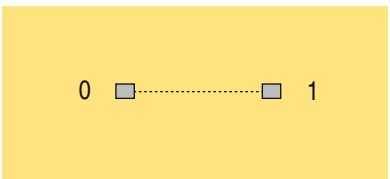
CODICE SEDC

Un codice SEDC deve avere almeno DMIN = 2.

Vediamo un esempio in cui dobbiamo memorizzare due informazioni, on e off.

A Codice irridondante

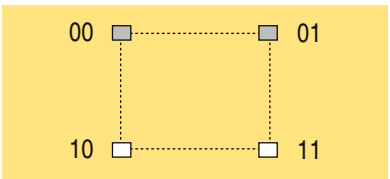
on = 0
off = 1



Se un bit si sporca si passa sempre in situazioni valide e quindi è impossibile riconoscere la presenza di una situazione di malfunzionamento.

B Codice ridondante con distanza minima 1

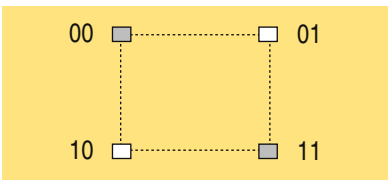
on = 00
off = 01



Se un bit si sporca si passa in situazioni sia accettate sia non valide: quindi non sempre è possibile riconoscere la presenza di una situazione di malfunzionamento.

C Codice ridondante con distanza minima 2

on = 00
off = 11



Se un bit si sporca si passa sempre in situazioni non valide: quindi è sempre possibile riconoscere la presenza di una situazione di malfunzionamento.

In generale, un codice per la rilevazione di modifiche su k bit deve avere almeno DMIN = k + 1.

Codici di Hamming

Per consentire la rilevazione e la correzione di errori è necessario utilizzare codici ridondanti, ovvero codici che utilizzano un numero maggiore di bit rispetto al numero strettamente necessario per codificare l'insieme sorgente.

Indichiamo per esempio:
- m bit di dati contenenti l'informazione da trasmettere;
- r bit di controllo o bit ridondanti.

Ciascuna parola in codice utilizza n = m + r bit, quindi ha lunghezza pari a n bit.

CODICI DI HAMMING

I codici di identificazione e correzione di errore con controllo di parità si chiamano codici di Hamming.

Controllo di parità

Il codice più semplice che ci permette di individuare un errore è il codice di parità, che consiste nell'aggiungere un bit in più alla codifica, detto bit di parità, e rende pari il numero degli 1 presenti in ogni configurazione.

Per esempio, aggiungiamo il bit di parità come LSB al seguente alfabeto in codice di 7 bit. Ora le parole hanno lunghezza 8 bit e se si verificasse il danneggiamento di un bit durante la trasmissione, il ricevitore sarebbe in grado di accorgersene riconoscendo la presenza dell'errore semplicemente effettuando il conteggio degli 1 presenti.

Parola in codice	Bit di parità	Numero di 1
0000000	**0**	0
0101010	**1**	4
0111011	**1**	6
0101011	**0**	4
1001100	**1**	4
1111111	**1**	8
1000100	**0**	2

Il destinatario è però in grado solamente di individuare la parola che arriva errata, ma non è in grado di correggerla in quanto gli è impossibile scoprire quale degli 8 bit si è sporcato e, quindi, riportarlo al valore originale. Una volta intercettato l'errore viene richiesta la ritrasmissione del messaggio.
Il codice di Hamming di identificazione di errore per ogni parola è il seguente:

Parola in codice	Codice di Hamming
0000000	00000000
0101010	01010101
0111011	01110111
0101011	01010110
1001100	10011001
1111111	11111111
1000100	10001000

dove ogni parola è composta dal contenuto informativo (m = 7) e, in aggiunta, la sua ridondanza (r = 1): ogni parola in codice ha lunghezza n = m + r = 8 bit.

CODICE LEGITTIMO

Si definisce codice legittimo una parola in codice che soddisfa l'algoritmo di parità.

Nel nostro esempio, oltre alle codifiche riportate nella tabella che segue, abbiamo per esempio:

Codifica	Numero di 1	
00011000	2	legittimo
01111110	6	legittimo
00000011	2	legittimo
00110000	2	legittimo
…		

che sono altre configurazioni non contemplate nella codifica ma che rispettano le condizioni di parità.

Supponiamo però che l'ultimo byte (10001000) arrivi a destinazione con tutti zeri (00000000), cioè che si siano sporcati 2 bit: il destinatario non si accorgerebbe dell'errore e lo interpreterebbe come dato valido.

Quindi il controllo di parità con ridondanza singola non si accorge di doppi errori, e comunque in caso di singolo errore non sa correggerlo, ma solamente identificarlo.

Questo sistema di rilevazione e correzione di un errore è quello normalmente utilizzato sui bus all'interno dei PC dato che la frequenza con cui si sporcano i bit sono dell'ordine di "*1 ogni qualche migliaia di ore di funzionamento*".
Sarebbe anche sufficiente la richiesta di ritrasmissione e quindi potrebbe bastare questo semplice controllo di parità come precauzione in caso di errori: diversa è l'ipotesi di trasferimento di dati verso HD o dispositivi elettromeccanici dove è necessario cautelarsi con sistemi più complessi in grado di effettuare direttamente l'autocorrezione.

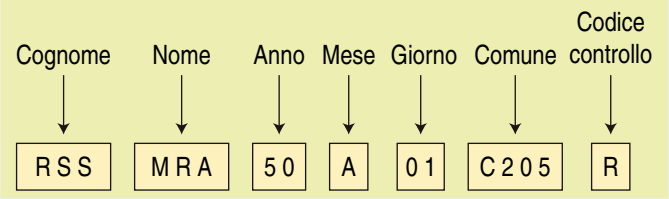
Zoom su...

CODICE FISCALE E IBAN

Sia il codice fiscale che l'IBAN sono codici con controllo di parità.

1 Codice fiscale

Il codice fiscale è composto da 16 cifre, di cui l'ultima è di controllo: si ottiene sommando i caratteri di posizione pari e quelli di posizione dispari secondo una tabella di corrispondenza numerica predefinita che individua in modo univoco da tale somma un carattere di controllo che identifica la correttezza dei primi 15 caratteri.

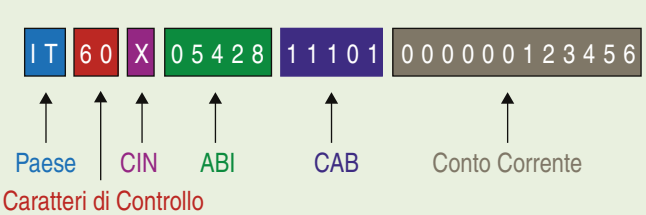


2 Codice IBAN

Il codice IBAN ha un doppio sistema di controllo con due CIN (Control Internal Number):

- il codice di controllo (o check digit) di 2 caratteri, che si ottiene da un algoritmo che elabora il paese, la banca e il conto (è anche chiamato CIN europeo);
- il CIN italiano, un carattere che si ottiene dal controllo delle successive 22 cifre, convertite in valore intero e sommate in modo da individuare in una tabella un carattere univoco.

La composizione del Codice IBAN



Identificazione e correzione degli errori

L'obiettivo che ci poniamo è quello di stabilire il numero massimo di errori che un codice è in grado di individuare e quindi di correggere.

Un errore è un cambiamento del valore di un bit in una posizione della parola: naturalmente un doppio errore sulla stessa posizione, oltre a essere ininfluente in quanto il bit "ritorna" al suo valore primitivo e quindi risulta essere corretto, non viene contato se non per fini statistici.

Identificazione della presenza di errori

CODICE RILEVATORE DI ORDINE N

Se un codice deve rilevare n errori, la sua distanza di Hamming deve essere almeno pari a $d = n + 1$.

In altre parole, se la distanza di Hamming tra codici legittimi è pari almeno a $d = n + 1$, allora è identificabile l'errore fino all'ordine n di errori (dove n è un intero).

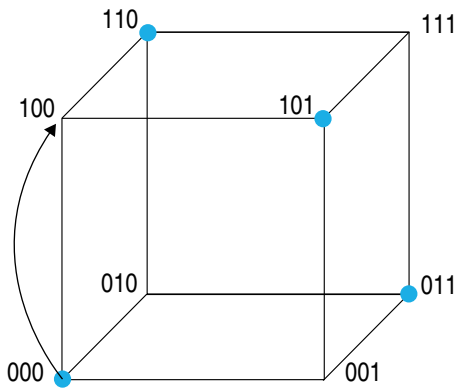
ESEMPIO 1

Vediamo nel caso precedente di un codice su 3 bit, dove un bit è di parità e quindi la distanza è $d = 2$, se è possibile identificare singoli errori, quindi $n = 1$:

- tra 101 e 100 identifico l'errore in quanto non è rispettata la parità;
- tra 101 e 000 non riesco a identificare l'errore dato che è rispettata la parità.

Con codici legittimi a distanza 2 non sarà riconoscibile il doppio errore, ma solo l'errore singolo.

Osserviamo visivamente sul reticolo di Hamming le due situazioni, dove indichiamo con un cerchio le parole legittime di un codice di Hamming per l'identificazione dell'errore con $d = 1$:



Se un bit si sporca si passa da un vertice a quello adiacente che non è legittimo e in seguito si individua una configurazione di errore: la distanza tra i due vertici è unitaria mentre la distanza tra due codici legittimi è d = 2.

La presenza contemporanea di d errori trasforma una parola di codice in una sequenza di bit corretta e quindi non permette di individuare la presenza degli errori.

Zoom su...

COMBINAZIONI

È possibile calcolare il numero di parole codice di 8 bit aventi distanza di Hamming 2 da una parola di codice assegnata avente la stessa lunghezza: il calcolo delle combinazioni ci viene in aiuto fornendoci le combinazioni semplici di 2 bit su 8:

$$\begin{matrix} n \\ k \end{matrix} = \frac{n!}{n(n-k)!} \Longrightarrow \begin{matrix} 8 \\ 2 \end{matrix} = \frac{8!}{2!\cdot 6!} = \frac{7\cdot 8}{2} = 28$$

La parità che abbiamo visto prende anche il nome di parità pari PE (*Parity Even*): esiste anche la parità dispari PO (*Parity Odd*) dove viene aggiunto un bit tra 0 e 1 in modo da avere un numero dispari di 1 nel blocco.

Correzione degli errori con checksum

Per riuscire a correggere l'errore mediante un codice di parità è necessario utilizzare il sistema di controllo della parità incrociata.

Si introduce in coda alla trasmissione di un blocco di n dati un byte, chiamato byte di checksum, che viene calcolato eseguendo l'operazione di XOR su tutti i byte costituenti il blocco-dati da trasmettere.

Dato 1	Dato 2	Controllo

Questo metodo prende anche il nome di controllo di parità longitudinale LRC (*Longitudinal Redundancy Check*) mentre il bit di parità descritto precedentemente è detto anche controllo di parità trasversale.
Vediamo un semplice esempio dove trasmettiamo 4 dati:

	b7	b6	b5	b4	b3	b2	b1	parità
Dato1	1	1	0	0	1	1	1	1
Dato2	0	1	0	0	0	0	0	1
Dato3	0	0	1	1	0	0	0	0
Dato4	1	1	0	0	1	1	0	0
checksum	0	1	1	1	0	0	1	0

Prima della trasmissione vengono aggiunti sia i controlli di parità trasversale (bit di parità "orizzontale") sia il byte di checksum, che esegue il controllo di parità "verticale".

Supponiamo che durante la trasmissione un bit del Dato3 si sporchi: alla ricezione il controllo di parità trasversale ci segnala la presenza dell'errore sul Dato3 senza però indicarne la posizione.

	b7	b6	b5	b4	b3	b2	b1	parità
Dato1	1	1	0	0	1	1	1	1
Dato2	0	1	0	0	0	0	0	1
Dato3	0	0	0	1	0	0	0	0
Dato4	1	1	0	0	1	1	0	0
checksum	0	1	1	1	0	0	1	0

Alla fine della trasmissione dei dati il controllo "verticale" ci segnala la presenza di un errore nel bit b5, dato che il numero di 1 è dispari.
Dall'intersezione tra la riga e la colonna individuiamo il bit che si è sporcato: con questo sistema siamo quindi in grado di correggere un errore.

Se gli errori crescono non sono più rilevabili ed è necessario aumentare la distanza minima di Hamming introducendo ulteriori bit di ridondanza.

Per esempio, in presenza di due errori ci troviamo in questa situazione:

	b7	b6	b5	b4	b3	b2	b1	parità
Dato1	1	1	0	0	1	0	1	1
Dato2	0	1	0	0	0	0	0	1
Dato3	0	0	0	1	0	0	0	0
Dato4	1	1	0	0	1	1	0	0
checksum	0	1	1	1	0	0	1	0

che però non siamo in grado di correggere perché entrambe le correzioni

	b7	b6	b5	b4	b3	b2	b1	parità
Dato1	1	1	1	0	1	0	1	1
Dato1	1	1	0	0	1	1	1	1

	b7	b6	b5	b4	b3	b2	b1	parità
Dato3	0	0	1	1	0	0	0	0
Dato3	0	0	0	1	0	1	0	0

portano a configurazioni legittime e quindi generano un'ambiguità nella correzione.

Se per esempio la distanza minima di Hamming è d = 3, il codice può essere usato come correttore di un errore (in una posizione qualsiasi della N-pla). Lo stesso codice può essere usato anche per rilerare due errori, senza però essere in grado di correggerli.
Vengono anche rilevate alcune configurazioni, ma non tutte, di tre o più errori.

Il codice di Hamming

Un secondo modo semplice per creare codici a correzione d'errore consiste nel replicare l'informazione.

CODICE CORRETTORE DI ORDINE N

Se la distanza di Hamming tra codici legittimi è pari almeno a $d = 2n + 1$, allora è correggibile l'errore sino all'ordine n di errori.

Nel 1950 Hamming propone il sistema di codifica a correzione d'errore che porta oggi il suo nome: il codice di Hamming è un codice che consente di correggere un singolo errore e funziona con qualunque dimensione del messaggio da trasmettere.

La differenza sostanziale rispetto al byte di checksum sta nel fatto che l'errore viene individuato e risolto direttamente dall'analisi del dato che lo contiene, quindi immediatamente.

L'idea di base del codice di Hamming è quella di *codificare la posizione dell'eventuale errore*: si aggiungono alcuni bit di controllo, tanti quanti sono necessari per codificare la posizione del bit errato ma, poiché anche i bit di controllo sono soggetti a errore, occorre includere anche le posizioni di questi bit nel conteggio totale.

ESEMPIO 2

◀ Un codework è il risultato della somma dei bit di dati + i bit di controllo. ▶

Data una stringa di 8 bit, si aggiungono 4 bit di controllo, per un totale di 12 bit di ◀ codework ▶: 4 bit di posizione bastano per codificare le 12 posizioni, dato che $2^4 = 16$ ci permette di codificare 16 posizioni.
In sintesi, per codificare la posizione di un errore in una parola di 12 bit è necessario avere a disposizione 4 bit: per esempio, se è errato il bit 11_{10} lo individuiamo come $1011_2 = 11_{10}$, cioè abbiamo bisogno di 4 bit per codificare la sua posizione di errore.

Esiste una regola che ci permette di individuare dal numero di bit la presenza o meno di un codice ottimo.

CODICE CORRETTORE OTTIMO

Un codice correttore è ottimo se vale la seguente relazione:
$bit\_dati + bit\_controllo + 1 <= 2^{bit\_controllo}$;

Esempio:
- con 3 bit di controllo e 4 bit dei dati (1 nibble) avremo: $3 + 4 + 1 <= 2^3$, quindi è ottimo;
- con 4 bit di controllo e 8 bit dei dati (ASCII) avremo: $4 + 8 + 1 <= 2^4$, quindi è ottimo.

Nel primo caso il codice generato prende il nome di codice di Hamming (7,4) mentre nel secondo caso avremo il codice di Hamming (12,8).

Costruiamo il singolo codework "mischiando" bit di informazione e bit di controllo.

A Innanzitutto vengono numerati i bit della parola da trasmettere a partire da sinistra verso destra, inserendo a ogni posizione avente indice una potenza del 2 il posto per un bit di controllo:

	b1	**b2**	**b3**	**b4**	**b5**	**b6**	**b7**	**b8**	**b9**	**b10**	**b11**
Dato	CTR	CTR	x	CTR	x	x	x	CTR	x	x	x

Su 11 bit saranno di controllo il bit1 (2^0), il bit2 (2^1), il bit4 (2^2) e il bit8 (2^3).

B Ogni bit di controllo è un bit di parità che esprime la parità di un sottoinsieme opportuno dei bit della stringa: il bit di controllo in posizione 2^j controlla la parità di tutti i bit la cui posizione, in binario, ha il j-esimo bit a 1.

Di seguito vediamo una tabella che ci aiuta nella costruzione dei bit di parità:

Posizione del bit di controllo	Posizione dei bit controllati
1	1,3,5,7,9,11
2	2,3,6,7,10,11
4	4,5,6,7,12
8	8,9,10,11,12

Ricordando la codifica binaria dei numeri da 1 a 12 analizziamo ogni bit di controllo.

Il primo bit di controllo si ottiene come parità di tutti i bit che sono in posizione tale per cui la loro codifica binaria ha 1 nel bit meno significativo (sono tutte le posizioni dispari): ▶

1 = 0001	sì
2 = 0010	no
3 = 0011	sì
4 = 0100	no
5 = 0101	sì
6 = 0110	no
7 = 0111	sì
8 = 1000	no
9 = 1001	sì
10 = 1010	no
11 = 1011	sì
12 = 1100	no

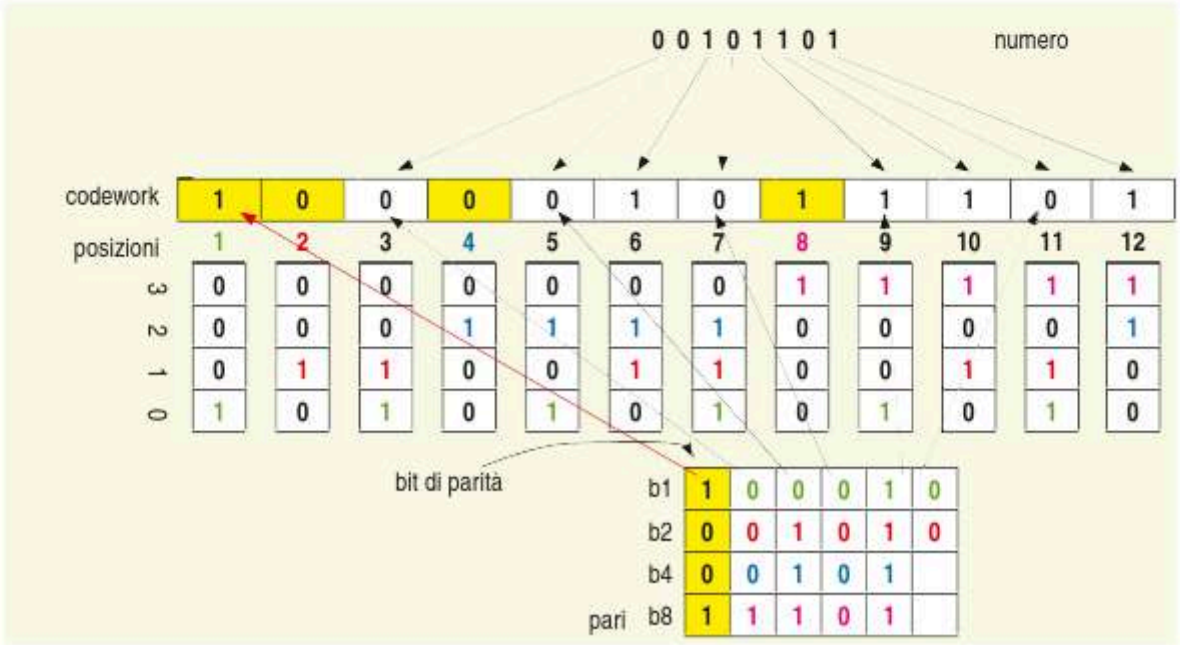
Il secondo bit è di parità per tutti i bit in cui la codifica binaria della posizione ha il 2° bit = 1, e cioè 2 3 6 7 10 11 . . .

	[I°]	[II°]	[III°]	[IV°]
1 = 0001	sì			
2 = 0010	no	sì		
3 = 0011	sì	sì		
4 = 0100	no		sì	
5 = 0101	sì		sì	
6 = 0110	no	sì	sì	
7 = 0111	sì	sì	sì	
8 = 1000	no			sì
9 = 1001	sì			sì
10 = 1010	no	sì		sì
11 = 1011	sì	sì		sì
12 = 1100	no		sì	sì

Il terzo bit è di parità per tutti i bit in cui la codifica binaria della posizione ha il 3° bit = 1, cioè
4 5 6 7 12 . . .

Il quarto bit è di parità per tutti i bit in cui la codifica binaria della posizione ha il 4° bit = 1 cioè
8 9 10 11 12 . . .

Vediamo un esempio di codifica della parola 00101101 nel codice di Hamming (12,8):



Costruiamo una tabella dove riportiamo i singoli bit di informazione relativi ai 4 bit di controllo che dobbiamo generare: in caso di parità pari abbiamo $b_1 = 1$, $b_2 = 0$, $b_4 = 0$, $b_8 = 1$. Otteniamo come codework:

100001011101

Potremmo anche effettuare la codifica con parità dispari, e quindi in tal caso il codework sarebbe:

010101001101

Nel caso di errore in trasmissione è molto semplice individuare la posizione del bit che si è sporcato dalla verifica di ogni singolo bit di parità:

- se tutti i valori dei bit di controllo sono corretti, la parola di codice viene accettata;
- se alcuni bit di controllo hanno valori non corretti, l'indice (la posizione) del bit in cui si è verificato l'errore è dato dalla somma degli indici dei bit di controllo con valore sbagliato: infatti essi intrinsecamente contengono nella loro posizione gli addendi che ci permettono di ottenere il valore della posizione del bit errato.

Vediamo un esempio:

dato	00101101											
3210	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100
codifica	1	0	0	0	0	1	0	1	1	1	0	1
trasmissione												
Bit	1	2	3	4	5	6	7	8	9	10	11	12
ricezione	1	0	0	0	0	0	0	1	1	1	0	1
parità	1	1		1				1				
discordanze	N	S		S				N				
Bit errato	2 + 4 = 6											

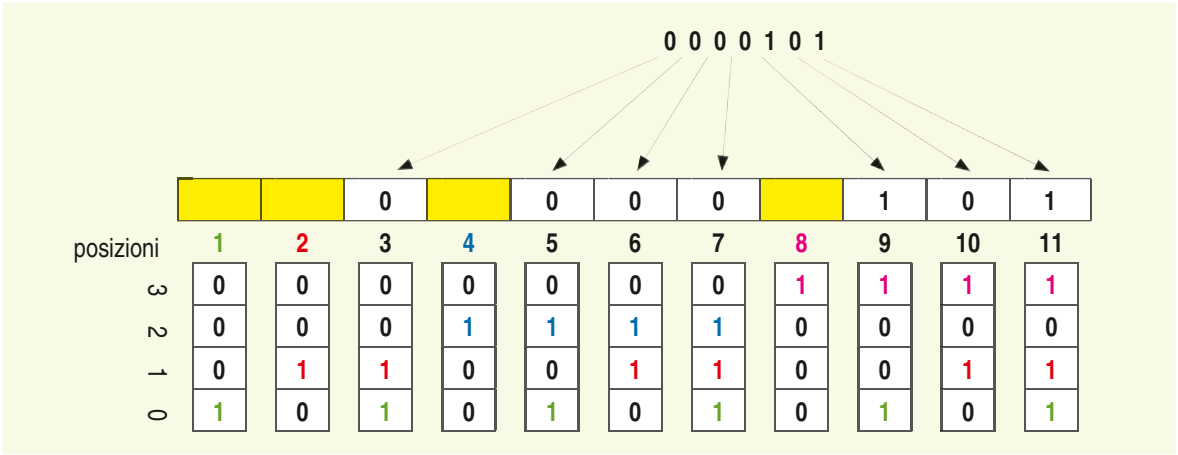
Eseguiamo per ogni bit il controllo di parità:

- 1 bit → somma bit 1 3 5 7 9 11 = 1 0 0 0 1 0 = pari ok
- 2 bit → somma bit 2 3 6 7 10 11 = 0 0 0 0 1 0 = dispari ERROR = POSIZ 2
- 4 bit → somma bit 4 5 6 7 12 = 0 0 0 0 1 = dispari ERROR = POSIZ 4
- 8 bit → somma bit 8 9 10 11 12 = 1 1 1 0 1 = pari ok

Sommando le posizioni dei bit con check errato, cioè il secondo e il quarto, otteniamo la posizione del bit del messaggio che si è sporcato: il bit numero 6.

Vediamo un secondo esempio, dove trasmettiamo con codice di Hamming dispari (11/4) la seguente stringa di 7 bit: 0000101.

Ⓐ Codifica in codice di Hamming



La tabella per il calcolo dei bit di controllo è la seguente:

b1	1	0	0	0	1	1
b2	0	0	1	0	1	0
b4	1	0	0	0	0	0
b8	1	1	0	1		

Il codework ottenuta è quindi: 10010001101.

Ⓑ Dopo la trasmissione la configurazione ricevuta è la seguente: 1001 0001 100.

Controlliamo i singoli bit di parità:
- 1 bit → somma bit 1 3 5 7 9 11 = 1 0 0 0 1 0 = pari ERROR = POSIZ 1
- 2 bit → somma bit 2 3 6 7 10 11 = 0 0 0 0 0 0 = pari ERROR = POSIZ 2
- 4 bit → somma bit 4 5 6 7 = 1 0 0 0 = dispari ok
- 8 bit → somma bit 8 9 10 11 = 1 1 0 0 = pari ERROR = POSIZ 8

Si è sporcato il bit in posizione 11.

Verifichiamo le conoscenze

1 Qual è la distanza di Hamming tra le seguenti due codifiche (a 8 bit)?

00101000 e 00011001

- ○ 0
- ○ 1
- ○ 2
- ○ 3
- ○ 4

2 Qual è la distanza minima di Hamming tra le seguenti due codifiche (a 8 bit)?

00101000, 00011001, 01111001, 00111000

- ○ 0
- ○ 1
- ○ 2
- ○ 3
- ○ 4

3 Quali delle seguenti configurazioni è errata in parità pari?

- ○ 1000100
- ○ 0000111
- ○ 0001100
- ○ 1010101
- ○ 1010101

4 Quali delle seguenti configurazioni è errata in parità dispari?

- ○ 0000100
- ○ 1000111
- ○ 1001100
- ○ 0010101
- ○ 1010011

5 Se hai ricevuto il byte 00001111 e un bit si è sporcato, qual è il messaggio inviato?

- ○ 0000100
- ○ 0000111
- ○ 0001100
- ○ 0010101
- ○ 0110101

6 Se hai ricevuto il byte 00001111 e due bit si sono sporcati, qual è il messaggio inviato?

- ○ 0010100
- ○ 0000111
- ○ 0001100
- ○ 0010101
- ○ 0011101

7 Qual è la codifica di Hamming (7,4) del messaggio 1 0 0 1?

- ○ 0 0 1 0 1 0 1
- ○ 0 0 1 0 0 1 1
- ○ 0 0 1 1 0 0 0
- ○ 0 0 1 1 0 0 1
- ○ 0 0 1 0 1 1 1

8 Quale messaggio è codificato nella parola in codice di Hamming (17,12) 1001 1010 1001 1010 1 dove un bit si è sporcato?

- ○ 0101 1001 1011
- ○ 0101 0001 1010
- ○ 0101 1011 1010
- ○ 0101 1001 1110
- ○ 0101 1001 1010

Verifichiamo le competenze

Esprimi la tua creatività

1 **Per codificare i tre simboli A, B, C si utilizza il seguente codice a 5 bit:**

A → 00000

B → 11100

C → 10011

Durante la trasmissione si possono modificare uno o più bit.

a) Quanti errori è in grado di rilevare il codice in generale?

b) Quanti errori è in grado di correggere in generale?

Soluzione

Calcola la distanza tra le tre parole in codice:

d_{12} = ____ d_{23} = ____ d_{13} = ____

a) Errori rilevati: _____

b) Errori corretti: _____

2 **Considera il seguente codice a 3 valori originari:**

1 → 000000

2 → 000001

3 → 111111

a) Trova quanti errori può correggere e rilevare in generale.

b) Supponi di ricevere la sequenza 001111: se ci sono stati al massimo due errori, è possibile decodificare correttamente la sequenza?

Soluzione

Calcola la distanza tra le tre parole in codice:

d_{12} = ____

d_{23} = ____

d_{13} = ____

a) _____

b) in questo caso puoi decodificare _____ con il valore _____.

3 **Hai ricevuto questi blocchi di dati: prova a dire dove si è verificato l'errore e se/come è possibile correggerlo.**

Blocco A	Blocco B	Blocco C	Blocco D
01100001	11100011	01101011	01110001
01000001	01000001	01001011	11000001
01101001	01101001	01101001	01101001
11100001	11100001	11100001	11100001
01100001	00100001	00100001	00100001
01100101	01101101	01100101	01100101
01100000	11100010	01100000	01100000
01100101	01100101	01100101	01100100
11100001	11100001	11100000	11100001

4 Invia il carattere ASCII M con un codice Hamming (11,7) di parità pari.

Il codice ASCII del carattere M è ..... quindi .....$_2$

Posizioniamo nel codework i bit di informazione:

Posizione bit 1 2 3 4 5 6 7 8 9 10 11

Valore bit x x ..... x ..... x .....

Calcoliamo i bit di check (controllo) in questo modo:

`bit 1 = x.....` → .....

`bit 2 = x.....` → .....

`bit 4 = x.....` → .....

`bit 8 = x.....` → .....

`risultato: ..... (parità pari)`

Inseriamo in ordine, al posto delle quattro x, i quattro bit:

Posizione bit 1 2 3 4 5 6 7 8 9 10 11

Valore bit .....

5 La seguente sequenza di 7 bit (1010100) rappresenta un codice di Hamming autocorrettivo a un bit. Ricava la parola codificata.

Il codice è:

*r*1	*r*2	*m*1	*r*3	*m*2	*m*3	*m*4
1	0	1	0	1	0	0

La sequenza ..... è .....

La sequenza ..... è .....

La sequenza ..... è .....

Il bit errato è il bit ____ per cui la parola codice corretta risulta:

*r*1	*r*2	*m*1	*r*3	*m*2	*m*3	*m*4
.....	.....	.....	.....	.....	.....	.....

La parola codificata è: $m_1m_2m_3m_4$ = ...

3 MODULO LA CODIFICA DEI NUMERI

OBIETTIVI

- Acquisire la nozione di complemento di un numero
- Acquisire il concetto di overflow
- Comprendere le modalità di rappresentazione dei numeri negativi
- Conoscere le motivazioni delle rappresentazioni a virgola mobile
- Acquisire il concetto di normalizzazione della mantissa
- Conoscere lo standard IEEE-P754 a 32 e 64 bit
- Comprendere le problematiche relative all'approssimazione e all'arrotondamento dei numeri periodici

ATTIVITÀ

- Eseguire il complemento a 1 e a 2 di un numero binario
- Effettuare le operazioni algebriche tra numeri binari
- Codificare i numeri in modulo e segno
- Rappresentare i numeri in complemento a 1, a 2 e a n
- Rappresentare i numeri decimali in virgola mobile
- Codificare e decodificare i numeri in IEEE-P754
- Codificare un numero periodico

UNITÀ DIDATTICA 1

OPERAZIONI TRA NUMERI BINARI SENZA SEGNO

IN QUESTA UNITÀ IMPAREREMO...

- le operazioni di complemento a 1 e a 2
- le operazioni di aritmetica binaria

Aritmetica binaria

Nonostante la rappresentazione dei numeri utilizzi delle tecniche che non sono la semplice codifica binaria, soprattutto per i numeri relativi e i numeri di grosse dimensioni, negli strumenti che elaborano dati binari è necessario eseguire le operazioni algebriche sui singoli byte binari; in particolare, in questa unità didattica, vedremo le quattro operazioni classiche della matematica e l'operazione di complemento, tipica dei sistemi binari.

Complemento a 1

La prima operazione che effettuiamo su numeri binari è il complemento a 1.

COMPLEMENTO A 1

Il complemento a 1 di un numero si ottiene scambiando gli 1 con gli 0 e viceversa.

Esistono diverse modalità di rappresentazione per indicare il complemento di un numero A:
- CP1(A) = complemento a 1 di A;
- ^{1}A = complemento a 1 di A;
- $\overline{A}$ = complemento a 1 di A (usato in elettronica come negazione).

Per il calcolo si procede nel modo indicato dalla definizione.

Vediamo di seguito alcuni esempi:

CP1(01110) = 10001
CP1(10001) = 01110
CP1(01101101) = 10010010

Complemento a 2

COMPLEMENTO A 2

Il complemento a 2 di un numero si ottiene scambiando gli 1 con gli 0 o viceversa e sommando 1 al risultato ottenuto.

Il complemento a 2 equivale quindi a sommare 1 al complemento a 1 di un numero. Vediamo alcuni esempi:

CP2(01110) = CP1(01110) + 00001 = 10001+ 00001 = 10010
CP2(01100) = CP1(01100) + 00001 = 10011+ 00001 = 10100
CP2(10001) = CP1(10001) + 00001 = 01110+ 00001 = 01111

Esiste un metodo pratico per effettuare il complemento a 2 di un numero N: si riscrivono i bit del numero stesso a partire da LSB lasciando inalterati tutti gli zeri fino a quando si incontra il primo 1, si ricopia anch'esso e quindi si invertono i bit successivi (0 in 1, 1 in 0). Per esempio:

- CP2(100101) => 011011: in questo caso il primo bit che troviamo è proprio un 1. Lo lasciamo quindi inalterato ma procediamo a invertire tutti gli altri bit alla sua sinistra;
- CP2(10010100) => 01101100: in questo caso i primi due bit vengono lasciati inalterati fino a quando incontriamo un bit uguale a 1, il terzo bit, che viene lasciato inalterato anch'esso. Procediamo invertendo i bit successivi alla loro sinistra.

Addizione

Riportiamo le quattro possibili combinazioni che otteniamo dalla somma tra due bit:

+	b_1		b_2			addizione
	0	+	0	=	0	
	0	+	1	=	1	
	1	+	0	=	1	
	1	+	1	=	0	riporto = 1

Il risultato della somma tra due bit di valore unitario è un bit di valore 0 e, contemporaneamente, si genera un riporto che viene indicato in un particolare bit dei microprocessori (bit di carry).

Per sommare due numeri procediamo come per l'operazione tra numeri decimali, disponendo i numeri in colonne ed eseguendo da destra verso sinistra la somma dei singoli bit, aggiungendo il riporto quando questo viene generato.

ESEMPIO

Vediamo di seguito alcuni esempi:

binario	
riporto	
	1 0 0 +
	1 1 =
	———————
	1 1 1

decimale	
riporto	
	4 +
	3 =
	———
	7

binario	
riporto	1 1
1	1 1 0 +
	1 1 =
	———————
	1 0 1 1

decimale	
riporto	
	6 +
	3 =
	———
	7

binario	
riporto	1 1 1
1	1 1 1 0 +
	1 1 =
	———————
	1 0 0 1 1

decimale	
riporto	
	1 4 +
	3 =
	———
	1 7

I microprocessori eseguono queste operazioni "istantaneamente", cioè con tempi dell'ordine di qualche nanosecondo (10^{-9}) con stringhe di 8, 16 o 32 bit.
Vediamo come viene generato un carry eseguendo operazioni su 8 bit (byte): quando il risultato dei due addendi è maggiore di 255 (massimo valore rappresentabile con 8 bit), si deve generare un riporto per il bit più significativo (l'eventuale nono bit).

binario	
riporto	1 1 1 1
	1 1 1 0 1 0 0 0 +
	0 1 1 0 1 1 1 0 =
	———————————
	0 1 0 1 0 1 1 0

decimale	
riporto	
	2 3 2 +
	1 1 0 =
	———————
	8 6 **atteso 342!**

Il valore 01010110_2 corrisponde a 86_{10} ma gli addendi 232_{10} e 110_{10} avrebbero dovuto dare come risultato 342_{10}, cioè 101010110_2: si è perso il bit più significativo, cioè $2^9 = 256$, che sommato a 86 dà come risultato proprio $256_{10} + 86_{10} = 342_{10}$, che era atteso.
Quando i microprocessori eseguono le somme che generano un overflow pongono al valore 1 un particolare bit, il bit di carry, per segnalare questa situazione.

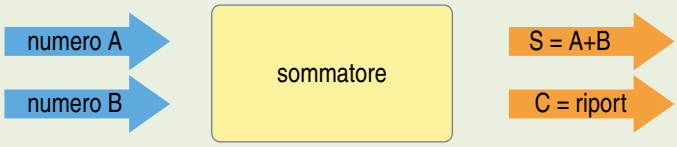
Zoom su...

CIRCUITO SOMMATORE

Nei calcolatori elettronici la somma bit per bit avviene con due dispositivi elettronici:

- mezzo sommatore: HALD ADDER, HA (senza riporto);
- sommatore completo: FULL ADDER, FA (con riporto).

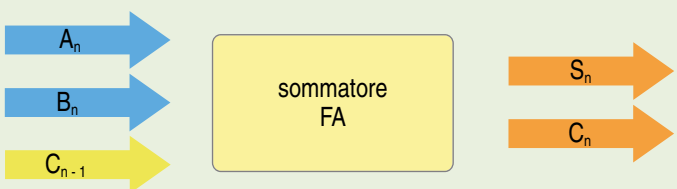
Osservando la seguente tabella della verità, vediamo lo schema di principio di un HA: il



dispositivo ha due linee di ingresso sulle quali vengono connessi i segnali dei due bit da sommare e due linee in uscita, la prima che riporta la somma dei valori in ingresso e la seconda per il riporto.

A	B		C	Sum
0	0		0	0
0	1		0	1
1	1		1	0
1	0		0	1

Come si può vedere dal disegno che segue, nel circuito sommatore completo le linee di ingresso sono invece tre: ai due bit da sommare si aggiunge la linea dove è presente il riporto del numero precedente, cioè il meno significativo rispetto a quello corrente.



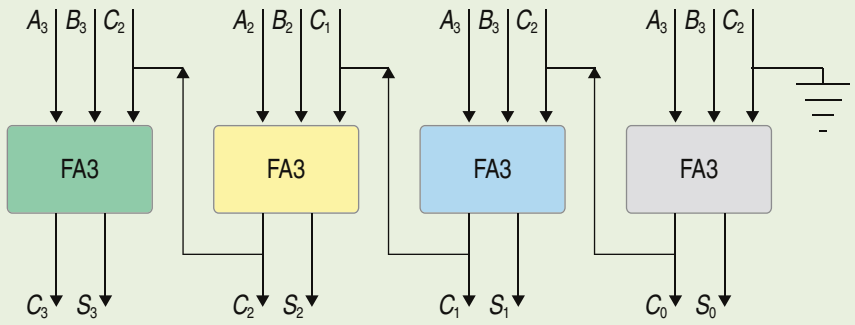
Le uscite sono generate dalla seguente tabella della verità:

Cin	A	B		Count	Sum
0	0	0		0	0
0	0	1		0	1
0	1	1		1	0
0	1	0		0	1
1	0	0		0	1
1	0	1		1	0
1	1	1		1	1
1	1	0		1	0

Viene eseguita la somma su tre bit in modo da avere in uscita:

$$Sum = A + B + C_{n-1}$$

Collegando tra loro più dispositivi possiamo realizzare i circuiti sommatori a più bit, come quello riportato nella figura che segue, che effettua la somma su due nibble (4 bit):



Il riporto generato dal primo dispositivo deve propagarsi in tutti gli altri. Avremo quindi un ritardo sull'FA3 ottenuto dalla somma dei tre precedenti tempi di calcolo: il riporto C_3 viene generato dopo tutti gli altri valori presenti sulle uscite.

Sottrazione

Riportiamo nella tabella della verità rappresentata a lato le quattro possibili combinazioni che otteniamo dalla sottrazione tra due singoli bit:

b_1 – b_2	sottrazione
0 – 0 = 0	
0 – 1 = 1	con prestito = 1
1 – 0 = 1	
1 – 1 = 0	

Il secondo caso, cioè 0 – 1, non può essere fatto su singoli bit, ma in una sottrazione tra byte o multipli di byte, in modo da poter prendere un prestito dal bit di sinistra. Dobbiamo anche ricordare che le nostre operazioni sono fatte su numeri privi di segno, cioè tra numeri che consideriamo in valore assoluto, e che non possono dare un risultato negativo.

Vediamo alcuni esempi.

Eseguiamo 7 – 3, che in binario è: ▶

binario	
	1 1 1 –
	1 1 =
	———
	1 0 0

In questo caso non abbiamo utilizzato nessun prestito.

Eseguiamo 5 – 3 seguendo passo passo ogni operazione, a partire dal bit meno significativo:

Entrambi hanno valore 1, quindi il risultato è 0. ▶

Il secondo bit del minuendo è uguale a 0, mentre il sottraendo è uguale a 1: avrò bisogno di prelevare un prestito dal bit di posizione due b_2.

b_2	b_1	b_0	
1	0	1	–
	1	1	=
	———		
		0	

Nella tabella aggiungiamo una riga dove indichiamo il prestito che abbiamo preso dal b_2, in modo tale da "ricordarci" quando andremo a fare la sottrazione su quel bit.

	b_2	b_1	b_0	
	1	0	1	–
prestito	1			–
		1	1	=
		1	0	

Il risultato della sottrazione dei bit b_1 è $0 - 1 = 1$.

Eseguiamo ora la sottrazione del bit b_2: non abbiamo nulla nel sottraendo, ossia 0, ma abbiamo il prestito dato al bit b_1, quindi il risultato è $1 - 1 - 0 = 0$.

	b_2	b_1	b_0	
	1	0	1	–
prestito	1			–
		1	1	=
	0	1	0	

Il risultato della sottrazione è il numero binario $010_2 = 2_{10}$.

Vediamo un esempio più complesso dove eseguiamo 14 – 3, e riportiamo i singoli passi in sequenza:

	b_3	b_2	b_1	b_0	
	1	1	1	0	–
prestito					–
	0	0	1	1	=
				1	

	b_3	b_2	b_1	b_0	
	1	1	1	0	–
prestito			1		–
	0	0	1	1	=
			1	1	

Bit b_0 $0 - 1 = 1$ con prestito da b_1, cioè $10 - 1 = 1$
Bit b_1 $1 - 1 - 1 = 1$ con prestito da b_2, cioè $11 - 1 = 1$

	b_3	b_2	b_1	b_0	
	1	1	1	0	–
prestito		1	1		–
	0	0	1	1	=
		0	1	1	

	b_3	b_2	b_1	b_0	
	1	1	1	0	–
prestito		1	1		–
	0	0	1	1	=
	1	0	1	1	

Bit b_2 $1 - 1 - 0 = 0$

Bit b_3 $1 - 0 = 1$

Il risultato della sottrazione è il numero binario $1010_2 = 11_{10}$.

Se il minuendo è più piccolo del sottraendo viene generato un errore. Vediamo un esempio, riportato a lato, eseguendo un'operazione su 8 bit: $104_{10} - 122_{10}$:

bit	7	6	5	4	3	2	1	0	
	0	1	1	0	1	0	0	0	–
prestito	1	1	1	1	1	1			–
	0	1	1	1	1	0	1	0	=
	1	1	1	0	1	1	1	0	

Il valore 11101110_2 ottenuto corrisponde a 238_{10} ed è il risultato della sottrazione tra 360_{10} e 122_{10}. Il numero 360 in binario è 101101000_2: si è quindi aggiunto un bit più significativo al minuendo, cioè $2^9 = 256$, che sommato a 104 dà come risultato proprio $256_{10} + 104_{10} = 360_{10}$. Quando i microprocessori eseguono le sottrazioni che generano un overflow, pongono al valore 1 sempre il bit di carry per segnalare questa situazione, che in questo caso ha il significato di prestito (borrow).

La sottrazione che abbiamo eseguito prende il nome di sottrazione diretta: esiste una seconda modalità per eseguire la sottrazione che utilizza la rappresentazione dei numeri negativi in complemento a 2, come vedremo nella prossima unità didattica.

Prodotto

La tabella della verità a lato riporta le quattro possibili combinazioni che otteniamo dal prodotto tra due bit:

X b_1		b_2		prodotto
0	x	0	=	0
0	x	1	=	0
1	x	0	=	0
1	x	1	=	1

Il procedimento è identico alla moltiplicazione tra numeri decimali: si parte dalla cifra meno significativa del moltiplicatore e si moltiplica per tutte le cifre del moltiplicando.

Eseguiamo per esempio 5 × 3 seguendo passo passo ogni operazione.

Il primo bit (b_0) del moltiplicatore è uguale a 1, quindi ricopiamo il moltiplicando così com'è sotto la linea di moltiplicazione:

b_2	b_1	b_0	
1	0	1	x
	1	1	=
1	0	1	+
			=

Anche il secondo bit (b_1) del moltiplicatore è uguale a 1, e ripetiamo la stessa operazione ricopiando il moltiplicatore ma spostandolo a sinistra di una posizione:

b_2	b_1	b_0	
1	0	1	x
	1	1	=
	1	0	1	+
1	0	1	–	=
1	1	1	1	

Come ultimo passaggio effettuiamo la somma dei due risultati parziali ottenendo $1111_2 = 15_{10}$.

ESEMPIO

Vediamo per esempio il prodotto di 1011 × 1010 , cioè 11 × 10:

b_6	b_5	b_4	b_3	b_2	b_1	b_0	
			1	0	1	1	x
			1	0	1	0	=
			0	0	0	0	+
		1	0	1	1	–	+
	0	0	0	0	–	–	+
1	0	1	1	–	–	–	=
1	1	0	1	1	1	0	

Quindi il prodotto di 1011 × 1010 è uguale a 0110 1110_2, che in binario rappresenta 110_{10}.

Vediamo un ultimo esempio: il prodotto di 11001010 × 01011000 ($202_{10} \times 88_{10} = 17.776_{10}$).

Nonostante sia un'operazione lunga non presenta grandi difficoltà: si moltiplicano per ogni termine del moltiplicatore, a partire da destra, tutti i termini del moltiplicando, spostando ciascun termine del primo di una cifra a sinistra. Alla fine si sommano i risultati.

Convertendo in decimale il risultato ottenuto verifichiamo che rappresenta il numero 17.776_{10}.

```
                1 1 0 0 1 0 1 0
                0 1 0 1 1 0 0 0
                ---------------
                0 0 0 0 0 0 0 0 0
              0 0 0 0 0 0 0 0 0 -
            0 0 0 0 0 0 0 0 0 - -
          1 1 0 0 1 0 1 0 - - - -
        1 1 0 0 1 0 1 0 - - - - -
      0 0 0 0 0 0 0 0 - - - - - -
    1 1 0 0 1 0 1 0 - - - - - - -
  0 0 0 0 0 0 0 0 - - - - - - - -
-------------------------------
1 0 0 0 1 0 1 0 1 1 1 0 0 0 0 0
```

Casi particolari: moltiplicazione per potenze del 2

Moltiplicazione per 2

Nel caso del prodotto per 2 non è necessario effettuare la moltiplicazione ma è sufficiente aggiungere uno zero (0) a destra. Infatti se moltiplichiamo per esempio 1011 × 10 otteniamo:

	b_4	b_3	b_2	b_1	b_0	
		1	0	1	1	x
				1	0	=
					0	+
	1	0	1	1	–	=
	1	0	1	1	0	

che è il numero di partenza con uno 0 aggiunto a destra.

Vediamo altri due esempi:

```
1011011  x
      10
---------
       0
 1011011–
---------
10110110
```

```
11011  x
   10
------
    0
11011–
------
110110
```

Moltiplicazione per 4

Analogo discorso nel caso del prodotto per 4: non è necessario effettuare la moltiplicazione ma è sufficiente aggiungere due zeri (00) a destra.
Infatti se moltiplichiamo per esempio 1011 × 100 otteniamo:

	b_5	b_4	b_3	b_2	b_1	b_0	
			1	0	1	1	x
				1	0	0	=
						0	+
					0	–	=
	1	0	1	1	–	–	=
	1	0	1	1	0	0	

Vediamo altri due esempi:

```
1011011  x
     100
---------
      00
1011011– –
---------
101101100
```

```
11011  x
  100
------
   00
11011– –
------
1101100
```

Moltiplicazione per 2^n

In definitiva dovendo moltiplicare per una potenza n del 2 è sufficiente aggiungere a destra un numero di 0 pari a n:

1011 × 2 n = 1 $2^1 = 2$ 10110
1011 × 4 n = 2 $2^2 = 4$ 101100
1011 × 8 n = 3 $2^3 = 8$ 1011000

MOLTIPLICAZIONE PER UNA POTENZA DI 2

Nel caso di moltiplicazione per potenza k-esima di 2 il risultato è uno shift a sinistra di k posizioni.

Divisione

Come per la moltiplicazione, anche per la divisione seguiamo l'algoritmo utilizzato per le divisioni decimali. Eseguiamo per esempio passo passo $110110_2/101_2$ ($54_{10}/5_{10}$):

b_5	b_4	b_3	b_2	b_1	b_0						
1	1	0	1	1	0	:	1	0	1	=	

Confrontiamo il divisore con le tre cifre più significative del dividendo. Dato che è minore, scriviamo 1 come quoziente e trascriviamo il divisore sotto queste tre cifre per eseguire la sottrazione che la effettuiamo ottenendo:

b_5	b_4	b_3	b_2	b_1	b_0						
1	1	0	1	1	0	:	1	0	1	=	1
1	0	1									
—	—	—									
0	0	1									

Abbassiamo una cifra, il b_2, e procediamo con la divisione aggiungendo uno 0 al quoziente, dato che il divisore è maggiore del dividendo (11 < 101):

b_5	b_4	b_3	b_2	b_1	b_0							
1	1	0	1	1	0	:	1	0	1	=	1	0
1	0	1	↓									
—	—	—	—									
0	0	1	1									

Abbassiamo una successiva cifra, il b_1, e procediamo con la divisione:

b_5	b_4	b_3	b_2	b_1	b_0							
1	1	0	1	1	0	:	1	0	1	=	1	0
1	0	1		↓								
—	—	—	—	—								
0	0	1	1	1								

Dato che 111 > 101 aggiungiamo 1 al quoziente, effettuiamo la sottrazione e successivamente aggiungiamo al resto l'ultimo bit, il bit_0:

b_5	b_4	b_3	b_2	b_1	b_0								
1	1	0	1	1	0	:	1	0	1	=	1	0	1
1	0	1											
0	0	1	1	1									
		1	0	1									
		0	1	0	0								

Otteniamo 100, che è minore di 101, quindi non divisibile: aggiungiamo al quoziente uno 0 e con questa operazione abbiamo concluso la divisione.

b_5	b_4	b_3	b_2	b_1	b_0									
1	1	0	1	1	0	:	1	0	1	=	1	0	1	0
1	0	1												
0	0	1	1	1										
		1	0	1										
		0	1	0	0	resto								

Il risultato ottenuto è 1010 con resto 100, infatti convertendo in decimale abbiamo:

dividendo: $110110_2 = 54_{10}$
divisore: $101_2 = 5_{10}$
quoziente: $1010_2 = 10_{10}$
resto: $100_2 = 4_{10}$

quindi $54_{10} : 5_{10} = 10_{10}$ con resto 4_{10}

Vediamo un altro esempio.

b_5	b_5	b_5	b_4	b_3	b_2	b_1	b_0									
1	0	0	1	1	0	1	1	:	1	0	0	1	=	1	0	0	0	1
1	0	0	1															
0	0	0	0	1	0	1	1											
				1	0	0	1											
				0	0	1	0	resto										

Convertito in decimale è il seguente:

dividendo: $10011011_2 = 155_{10}$
divisore: $1001_2 = 9_{10}$ quindi $155_{10} : 9_{10} = 17_{10}$ con resto 2_{10}
quoziente: $10001_2 = 17_{10}$
resto: $10_2 = 2_{10}$

È possibile effettuare come esercizio la verifica della correttezza dell'operazione appena eseguita effettuando la moltiplicazione tra 10001_2 e 1001_2 e aggiungendo il resto 10_2.

ESEMPIO

Eseguiamo come esempio le seguenti due operazioni: 110111 : 110 e 1010111 : 1010.

A

```
110111 : 111  = 1001
  0111
     1
```

Risultato: 55 : 6 = 9 con resto 1

B

```
1010111 : 1010 = 1000
   0111
    111
```

Risultato: 87 : 10 = 8 con resto 7

Casi particolari: divisione per potenze del 2

Divisione per 2

Nel caso della divisione per 2 non è necessario effettuare la divisione ma è sufficiente togliere uno 0 a destra oppure, in ogni caso, il bit meno significativo.
Infatti se dividiamo per esempio 1010 : 10 otteniamo:

b_4	b_3	b_2	b_1	b_0							
	1	0	1	1	:	1	0	=	1	0	1
				1	resto						

che è il numero di partenza senza l'ultima cifra, che rimane come resto. Ad esempio:

b_4	b_3	b_2	b_1	b_0							
1	1	0	1	1	:	1	0	=	1	0	1	1
				1	resto						

b_4	b_3	b_2	b_1	b_0							
1	0	1	1	0	:	1	0	=	1	0	1	0
				0	resto						

Divisione per 4

Analogo discorso nel caso della divisione per 4: non è necessario effettuare l'operazione ma è sufficiente togliere gli ultimi due bit a destra, ovvero quelli meno significativi.

Vediamo alcuni esempi.

A

	b_4	b_3	b_2	b_1	b_0							
		1	0	1	1	:	1	0	0	=	1	0
				1	1	resto						

B

	b_4	b_3	b_2	b_1	b_0							
	1	1	0	1	1	:	1	0	0	=	1	1
				1	1	resto						

C

b_5	b_4	b_3	b_2	b_1	b_0								
1	1	0	0	1	0	:	1	0	0	=	1	1	0	0
				1	0	resto							

Divisione per 2^n

In definitiva dovendo dividere per una potenza n del 2 è sufficiente togliere da destra un numero di 0 pari a n, che diventano il resto della divisione:

10111011: 2 n = 1 $2^1 = 2$ 1011101 resto 1
10111011: 4 n = 2 $2^2 = 4$ 101110 resto 11
10111011: 8 n = 3 $2^3 = 8$ 10111 resto 011

DIVISIONE PER UNA POTENZA DI 2

Nel caso di divisione per potenza k-esima di 2 il risultato è uno shift a destra di k posizioni.

Verifichiamo le competenze

Esprimi la tua creatività

1 **Converti le seguenti rappresentazioni in complemento a 1:**

00011 _____
01111 _____
11100 _____
11010 _____
00000 _____
10000 _____

2 **Converti le seguenti rappresentazioni in complemento a 2:**

11100011 _____
11101111 _____
11111100 _____
11111010 _____
11100000 _____
11110000 _____

3 **Rappresenta i numeri seguenti in CA1 con n = 8:**

Dec.	binario	CA1
23	_____	_____
123	_____	_____
64	_____	_____
127	_____	_____
192	_____	_____
240	_____	_____

4 **Rappresenta i numeri seguenti in CA2 con n = 8:**

Dec.	binario	CA2
22	_____	_____
122	_____	_____
63	_____	_____
126	_____	_____
191	_____	_____
239	_____	_____

5 **Esegui le seguenti somme su 4 bit:**

0101 + 0010 _____
1110 + 0011 _____
0010 + 1110 _____
1100 + 0011 _____
0111 + 0001 _____
1000 + 0111 _____

6 **Esegui le seguenti somme in binario su 8 bit:**

32 + 27 _____
96 + 15 _____
64 + 11 _____
45 + 66 _____
100 + 92 _____
110 + 130 _____

7 **Esegui le seguenti sottrazioni su 4 bit:**

0101 – 0010 _____
1110 – 0011 _____
0010 – 1110 _____
1100 – 0011 _____
0111 – 0001 _____
1000 – 0111 _____

8 **Esegui le seguenti sottrazioni in binario su 8 bit:**

32 – 27 _____
96 – 15 _____
64 – 11 _____
45 – 66 _____
100 – 92 _____
130 – 110 _____

9 **Esegui le seguenti moltiplicazioni:**

0101 * 0010 _____
1110 * 0011 _____
0010 * 1110 _____
1100 * 0011 _____
0111 * 0100 _____
1000 * 0111 _____

10 **Esegui le seguenti moltiplicazioni in binario su 8 bit:**

42 * 22 _____
56 * 33 _____
24 * 55 _____
15 * 66 _____
11 * 110 _____
18 * 122 _____

11 **Esegui le seguenti divisioni:**

10000101 / 0010 _____
10011110 / 0011 _____
10010010 / 1110 _____
10001100 / 0011 _____
10010111 / 0100 _____
10001000 / 1000 _____

12 **Esegui le seguenti divisioni in binario su 8 bit:**

142 / 22 _____
256 / 16 _____
324 / 30 _____
415 / 22 _____
511 / 8 _____
618 / 32 _____

UNITÀ DIDATTICA 2

NUMERI BINARI RELATIVI

IN QUESTA UNITÀ IMPAREREMO...

- la rappresentazione in modulo e segno
- la rappresentazione in complemento a 1
- la rappresentazione in complemento a 2
- la rappresentazione in eccesso 2^{n-1}

Introduzione

Per rappresentare i numeri negativi è necessario aggiungere alla codifica un segno che, mentre può essere sottointeso per i numeri positivi, deve essere in "qualche modo" aggiunto per i numeri negativi.
Per rappresentare gli interi relativi introduciamo un bit che codifica il segno e, di fatto, dimezza l'intervallo dei valori assoluti a parità di cifre binarie utilizzate per la codifica.

Per esempio, con un byte si possono rappresentare i valori compresi tra:
- [0..255] numeri interi senza segno;
- [–128..0..+127] numeri interi con segno.

Analogamente, con una word (2 byte) si possono rappresentare i valori compresi tra:
- [0..65535] numeri interi senza segno;
- [–32768..0..+32767] numeri interi con segno.

Si utilizzano varie rappresentazioni:
- modulo e segno;
- in complemento a 1;
- in complemento a 2.

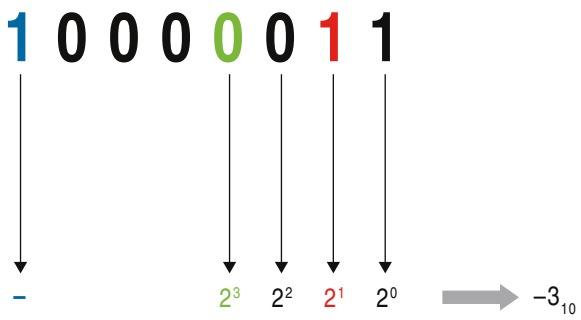
Modulo e segno

La rappresentazione con modulo e segno è un'estensione dei numeri naturali (1, 2, 3...) che include anche lo zero e tutti i valori negativi della forma –N, essendo N un naturale.

Per rappresentare il segno viene utilizzato il bit a sinistra: per convenzione si associa allo 0 il segno + e a 1 il segno – e si utilizzano i rimanenti bit per codificare il modulo del numero, cioè il valore assoluto del numero.

Per esempio con 8 bit l'MSB codifica il segno, dal bit 6 al bit 0 si ha il numero senza segno:

$(-2)_{10} = (1\ 0000010)_2$
$(+5)_{10} = (0\ 0000101)_2$
$(-15)_{10} = (1\ 0001111)_2$
$(+15)_{10} = (0\ 0001111)_2$



MODULO E SEGNO

Nella codifica in modulo e segno il bit che rappresenta esplicitamente i segni 0 = + e 1 = – è l'MSB (Most Significant Bit) e si utilizzano gli altri bit disponibili per rappresentare il valore assoluto come numero.
Il bit MSB prende anche il nome di bit di segno.

◀ Con range si intende il "campo" di rappresentazione dei numeri all'interno di una codifica: in un calcolatore, in base al numero di bit, è possibile calcolare il range di numeri rappresentabili.
Se durante un'operazione il risultato supera il range, viene generato un riporto (carry) sul bit più significativo che generalmente è indicato come overflow (trabocco sulla generazione del riporto). ▶

Vediamo ora come cambia il ◀ range ▶ di rappresentazione dei numeri con l'introduzione del segno. Avendo per esempio 3 bit a disposizione avremo la situazione della tabella a lato:

000	0
001	1
010	2
011	3
100	–0
101	–1
110	–2
111	–3

Abbiamo 3 bit, quindi 8 combinazioni:
- 3 numeri positivi;
- 3 numeri negativi;
- lo 0 codificato 2 volte (+0 e –0).

Quindi con 3 bit il range rappresentato è [–3, +3] anziché, come in binario puro, [0…7], cioè l'intervallo di rappresentazione viene più che dimezzato.

Esprimiamo con una formula l'intervallo dei numeri rappresentati con n bit:

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

ESEMPIO

n = 3 $[-2^{3-1} + 1, +2^{3-1} - 1] = [-3, +3]$
n = 4 $[-2^{4-1} + 1, +2^{4-1} - 1] = [-7, +7]$
n = 8 $[-2^{8-1} + 1, +2^{8-1} - 1] = [-127, +127]$

NUMERI IN MODULO E SEGNO

La rappresentazione dei numeri negativi con modulo e segno ha la caratteristica di avere un intervallo simmetrico e una doppia rappresentazione dello zero.

Questa rappresentazione presenta due difetti:
- il valore 0 ha due distinte rappresentazioni: 10000000 = "–0" e 00000000 = "+0";
- non permette di usare direttamente gli algoritmi già noti per eseguire le operazioni: in particolare, con le usuali regole di calcolo non è vero che $x - x = 0$.

Verifichiamo con un esempio e calcoliamo 5 – 5 = 5 + (–5):

		segno	modulo								
+5	+	0	0	0	0	0	1	0	1	+	
–5	=	1	0	0	0	0	1	0	1	=	

0		1	0	0	0	1	0	1	0		diverso da 0

La rappresentazione in modulo e segno non viene utilizzata quando è necessario fare operazioni aritmetiche in quanto risulta "scomodo" eseguire le sottrazioni.
Nei calcolatori la notazione più utilizzata per le operazioni è quella in complemento a 2, che vedremo nelle unità che seguono.

Per convertire un numero binario espresso in modulo e segno a numero relativo si eseguono due passaggi:
- si isola il segno, analizzando il binario espresso in MSB;
- si convertono gli n–1 bit come valore assoluto del numero.

Per esempio, convertiamo nei corrispondenti numeri relativi le seguenti rappresentazioni modulo e segno, 10101 e 01111:
- 10101:
  – 1: bit di segno = numero negativo;
  – 0100: valore assoluto = 5.
  Il numero relativo corrispondente è quindi $(10101)_2 = (-5)_{10}$.
- 01111:
  – 0: bit di segno = numero positivo;
  – 1111: valore assoluto = 15.
  Il numero relativo corrispondente è quindi $(01111)_2 = (+15)_{10}$.

Somma algebrica

La notazione in modulo e segno non è indicata per effettuare le operazioni tra i numeri, in quanto genera problemi soprattutto nella sottrazione. Vediamo alcuni esempi.

La regola generale per effettuare la somma algebrica tra due numeri codificati in modulo e segno è la seguente:

- se gli addendi sono concordi la somma è concorde con essi e ha come modulo la somma dei loro moduli. Si ha overflow nel caso di riporto sul bit di segno (carry-in) dovuto al traboccamento del penultimo bit (carry-out dal penultimo bit):

```
riporti          000          110          110
                0010  +      0111         1011  +
                0100  =      0010         1110  =
                ____         ____         ____
                0110         1001        11001
              corretto     overflow     overflow
```

- se gli addendi sono discordi la somma è concorde con l'addendo di modulo maggiore e ha come modulo la differenza tra il modulo maggiore e quello minore. In pratica si sottrae all'addendo di modulo maggiore l'opposto dell'addendo di modulo minore. Non è possibile avere overflow.

ESEMPIO 1

Eseguiamo –5 + 2 =
Convertiamo i due operandi in modulo e segno:

```
1101 +
0010 =
_____
```

Individuiamo il maggiore dei due operandi come valore assoluto (5).
Trasformiamo l'operando minore cambiandolo di segno:

```
-(0 010) => 1 010
```

Ora possiamo eseguire la somma come sottrazione:

```
1 101 –
1 010 =
_______
1 011
```

Il risultato è –3, come ci aspettavamo.

ESEMPIO 2

Eseguiamo –5 + 7 =
Convertiamo i due operandi in modulo e segno:

```
1101 +
0111 =
_____
```

Individuiamo il maggiore dei due operandi come valore assoluto (7).
Trasformiamo l'operando minore cambiandolo di segno:

```
-(1 101) => 0 101
```

Ora possiamo eseguire la somma come sottrazione:

```
  0 111 -
  0 101 =
  -------
  0 010
```

Il risultato è +2, come ci aspettavamo.

Per poter effettuare l'operazione di sottrazione, cioè l'addizione con segni discordi, è prima necessario eseguire operazioni preparatorie e solo successivamente è possibile effettuare l'operazione come addizione.

Complemento alla base

Il fatto che non sia possibile utilizzare la rappresentazione in segno e modulo per effettuare automaticamente le operazioni aritmetiche comporta che nei calcolatori si adottino per i numeri negativi altre rappresentazioni, come le rappresentazioni in complemento alla base.

Ne analizzeremo due:
- rappresentazione in complemento a 1;
- rappresentazione in complemento a 2.

In questo tipo di rappresentazione viene data una diversa attribuzione dei pesi associati alle cifre che codificano il numero:
- alla cifra più alta è associato un peso negativo;
- le cifre più basse hanno un peso positivo.

Complemento a 1

La rappresentazione dei numeri in complemento a 1 (CA1) con N bit si effettua nel seguente modo:
- numeri positivi: rappresentati come i primi 2^{N-1} numeri naturali;
- numeri negativi: "complemento a 1" della rappresentazione binaria del corrispondente intero positivo.

Per esempio:

```
CP1 (5)    => 5              = 0101
CP1 (–5)   => comp(0101)     = 1010
CP1 (15)   => 15             = 01111
CP1 (–15)  => comp(01111)    = 10000
CP1 (0)    => 0              = 00000
```

Vediamo in pratica come rappresentare con 5 bit i numeri in complemento a 1:
- codifichiamo su 4 bit i primi 16 numeri naturali ($2^5 - 1$);
- aggiungiamo uno 0 a sinistra e otteniamo i numeri positivi;
- otteniamo i numeri negativi complementando a 1 i numeri positivi.

Otteniamo come risultato la tabella riportata a lato: ▶

15	01111
14	01110
...	
2	00010
1	00001
0	00000
–0	11111
–1	11110
–2	11101
	...
–14	10001
–15	10000

Osserviamo che:
- i numerali positivi iniziano per 0, i numeri negativi per 1;
- l'intervallo di rappresentazione con *n* bit è dato da $[-2^{n-1} + 1, +2^{n-1} - 1]$;
- abbiamo una doppia rappresentazione dello zero.

Possiamo inoltre verificare che la notazione CA1 è una rappresentazione posizionale con i pesi delle cifre:
- la prima cifra: $(-2^{n-1} + 1)$;
- le altre cifre: 2^{n-1}, come nel binario.

Vediamo alcuni esempi:

$CP1(01101) = 0^* (-2^{5-1} + 1) + 1^*2^{4-1} + 1^*2^{3-1} + 1^*2^{2-1} + 1^*2^{1-1} = 0 + 2^3 + 2^2 + 0 + 2^0 = 13_{10}$

$CP1(11101) = 1^* (-2^{5-1} + 1) + 1^*2^{4-1} + 1^*2^{3-1} + 1^*2^{2-1} + 1^*2^{1-1} = (-16 + 1) + 2^3 + 2^2 + 0 + 2^0 = -2_{10}$

$CP1(00101) = 0^* (-2^{5-1} + 1) + 1^*2^{4-1} + 1^*2^{3-1} + 1^*2^{2-1} + 1^*2^{1-1} = 0 + 0 + 2^2 + 0 + 2^0 = 5_{10}$

$CP1(10101) = 1^* (-2^{5-1} + 1) + 1^*2^{4-1} + 1^*2^{3-1} + 1^*2^{2-1} + 1^*2^{1-1} = (-16 + 1) + 0 + 2^2 + 0 + 2^0 = -10_{10}$

Complemento a 2

La rappresentazione dei numeri negativi in complemento a 2 (CA2) si effettua nel modo seguente:
- numeri positivi: rappresentati come i primi 2^{N-1} numeri naturali;
- numeri negativi: si somma 1 alla rappresentazione in complemento a 1.

L'intervallo di numeri rappresentati con n bit è il seguente: $[-2^{n-1}, +2^{n-1} - 1]$.

Nella rappresentazione in complemento a 2 abbiamo l'intervallo asimmetrico con una sola rappresentazione dello zero.

Per esempio, effettuando la codifica con 4 bit di +5 e –5 e di +3 e –3:
- con n = 4 bit l'intervallo è il seguente: $[-2^{4-1}, +2^{4-1} - 1] = [-8, +7]$, con un numero negativo in più rispetto ai numeri positivi (si considera lo zero come "il numero positivo mancante");
- codifichiamo la prima coppia di numeri:
  $5 = 0101$
  $-5 = (-0101)_2 = CP1(1010) = CP2(1011)$

- codifichiamo la seconda coppia di numeri:
  3 = 0011
  $-3 = (-0011)_2$ = CP1(1100) = CP2(1101) ▶

Vediamo in pratica come rappresentare con 5 bit i numeri in complemento a 2:
- codifichiamo su 4 bit i primi 15 numeri naturali ($2^{5-1} - 1$);
- aggiungiamo uno 0 a sinistra e otteniamo i numeri positivi;
- otteniamo i numeri negativi complementando a 2 i numeri positivi.

Otteniamo come risultato la tabella riportata a lato, con range $[-2^{5-1}, +2^{5-1} - 1] = [-16, +15]$.
Vediamo alcuni esempi, evidenziando gli errori che tipicamente vengono commessi nell'utilizzo di questa rappresentazione.

15	01111
14	01110
	...
2	00010
1	00001
0	00000
–1	11111
–2	11110
–3	11101
	...
–13	10100
–14	10011
–15	10001
–16	10000

ESEMPIO 3

Rappresentazione in CA2 con 8 bit del numero decimale –67:
1. calcoliamo il valore binario di $67_{10} = 01000011_2$;
2. trasformiamolo in CA2: $CA2(-67_{10}) = 10111101_2$.

ESEMPIO 4

Rappresentazione in CA2 con 8 bit del numero decimale –3.

> Un errore tipico è quello di dimenticare il numero di bit con il quale deve essere fatta la rappresentazione.

$3_{10} = 11_2$ quindi –3 = 01, ma questo risulta essere un numero positivo!

La giusta codifica è la seguente:
1. calcoliamo il valore binario su 8 bit di 3 = 00000011_2;
2. trasformiamolo in CA2: –3 = 11111101.

ESEMPIO 5

Rappresentazione in CA2 con 6 bit del numero decimale 15.

> Un altro errore è quello di complementare sempre e comunque anche i numeri positivi.

La giusta codifica è la seguente:
1. calcoliamo il valore binario di $15_{10} = 001111_2$;
2. fine conversione: è già codificato in CA2.

Nella tabella riportata a lato è possibile confrontare la codifica su 4 bit nelle tre rappresentazioni: ▶

DECIMALE	M&S	CP1	CP2
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
–0	1000	1111	---
–1	1001	1110	1111
–2	1010	1101	1110
–3	1011	1100	1101
–4	1100	1011	1100
–5	1101	1010	1011
–6	1110	1001	1010
–7	1111	1000	1001
–8	---	---	1000

Aritmetica in complemento a 2

Le operazioni aritmetiche in CA2 che analizziamo sono l'addizione e la sottrazione: come già visto per la rappresentazione in modulo e segno è possibile che, in base al valore dei termini, si generino un riporto o un traboccamento.

OVERFLOW

Nelle operazioni aritmetiche in complemento a 2 si definisce overflow il traboccamento del riporto sulla cifra più significativa, l'MSB, che in questo tipo di rappresentazione individua il segno del numero (bit di segno).

Una semplice regola per la determinazione dell'overflow è la seguente:

- se i segni degli operandi sono diversi, non è possibile avere overflow, in quanto il risultato sarà sempre inferiore al maggiore dei termini di partenza;
- se i segni degli operandi sono uguali, si deve confrontare il segno del risultato:
  - + e +: deve risultare +, altrimenti overflow;
  - − e −: deve risultare −, altrimenti overflow.

Riportiamo nella tabella che segue le possibili combinazioni che portano a una generazione di overflow se il bit di segno del risultato è quello indicato:

Operazione	A	B	Bit di segno	Overflow
A + B	≥0	≥0	1	SÌ
A + B	<0	<0	0	SÌ
A – B	≥0	<0	1	SÌ
A – B	<0	≥0	0	SÌ

Possiamo anche formulare una regola che ci permette di individuare la presenza di un overflow semplicemente guardando i riporti.

RIPORTI E OVERFLOW

Una somma di due numeri di n cifre in complemento a 2 dà (errore di) overflow se e solo se i riporti in colonna n e n+1 sono diversi.

Addizione

L'addizione di due numeri rappresentati in CA2 dà il risultato corretto trascurando il riporto se il risultato è entro il range dei numeri rappresentabili. Per esempio, il numero positivo più grande rappresentabile con 8 bit in complemento a 2 è il seguente:

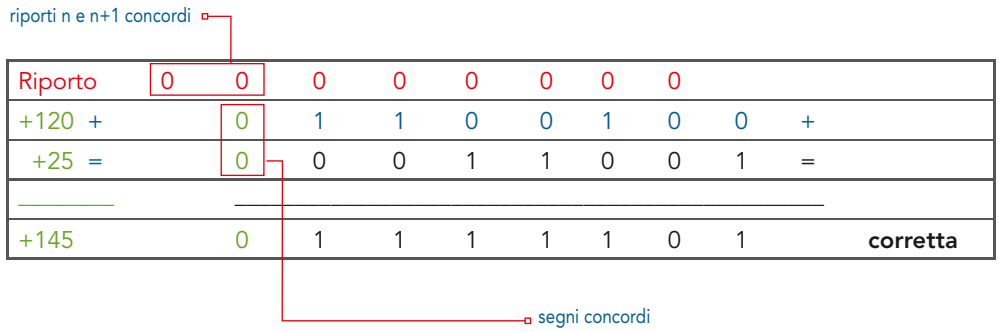
$$2^7 - 1 = 127$$

cioè la somma di tutti i pesi positivi; sommando 1 a 127 si ottiene −128, che è errato. Infatti

$$127_{10} = 011111111_{cp2}$$

I riporti nelle colonne n e n +1 sono diversi, quindi siamo in presenza di un overflow.

Vediamo per esempio la somma di 100 + 25:



I riporti nelle colonne n e n +1 sono uguali, quindi il risultato è corretto.

Sottrazione

La sottrazione viene effettuata come una semplice somma del CA2 del sottraendo: è proprio questo il principale vantaggio della rappresentazione CA2, cioè permettere l'esecuzione della sottrazione come fosse un'addizione (sottrazione indiretta).

Viene effettuata in due passaggi:

- si esegue la trasformazione del sottraendo in un numero in CA2;
- si sommano i due numeri ;
- in base al valore del riporto generato sull'MSB abbiamo tre situazioni:
  - se è uguale a 1 e il risultato è positivo il riporto si trascura;
  - se è uguale a zero e il risultato è negativo è necessario fare il CA2 del risultato per ottenere il modulo del numero finale: infatti vale la proprietà che CA2(CA2(A)) = A;
  - negli altri casi abbiamo avuto un overflow!

ESEMPIO 6

Eseguiamo la sottrazione tra +5 e +3 con 6 bit, cioè

$$(+5)_{10} - (+3)_{10} = \text{cioè } (000101)_2 - (000011) =$$

Innanzitutto determiniamo il range di rappresentazione, che è $[-2^{6-1}, +2^{6-1}-1] = [-32,+31]$; ora trasformiamo l'operazione in una somma:

$$(+5)_{10} + (-3)_{10} \Rightarrow (000101)_2 + (-000011)_2$$

e codifichiamo il numero negativo in complemento a 2 ottenendo b_0

$$001001^{2} + 111101^{cp2} =$$

I riporti nelle colonne n e n +1 sono uguali, quindi non siamo in presenza di overflow.

Essendo il riporto generato dall'MSB uguale a 1, si prende il numero risultante come positivo. Ricapitolando: $(+5)_{10} - (+3)_{10} = 001001_2 - 000011_2 = 001001_2 + 111101_{cp2} = 000010_2 = 2_{10}$.

Ricordiamo che per i numeri positivi la codifica in CA2 coincide con quella in modulo e segno, che a sua volta è la codifica in binario naturale: quindi le notazioni 001001_2 e 001001_{ca2} indicano lo stesso numero positivo.

ESEMPIO 7

Vediamo un altro esempio, dove eseguiamo la sottrazione tra +13 e +23 con 6 bit, cioè

$$(+13)_{10} - (+26)_{10} = \text{cioè } (001101)_2 - (011010)_2 =$$

Innanzitutto determiniamo il range di rappresentazione, che è $[-2^{6-1}, +2^{6-1}-1] = [-32,+31]$; ora trasformiamo l'operazione in una somma:

$$(+13)_{10} + (-26)_{10} \Rightarrow (001101)_2 + (-011010)_2$$

Dato che ora i due addendi hanno segno discorde, non avremo overflow.

Codifichiamo il numero negativo in complemento a 2 ottenendo

$$001101_2 + 100110_{CA2} =$$

I riporti nelle colonne n e n +1 sono uguali, quindi non siamo in presenza di overflow.

Essendo il riporto generato dall'MSB uguale a 0, il risultato sarà negativo e il valore si ottiene al complemento a 32 del risultato.

Ricapitolando: $(+13)_{10} + (-26)_{10} = -(110011)_{CA2} = -(001101)_2 = -13_{10}$.

ESEMPIO 8

Vediamo un altro esempio, dove eseguiamo la sottrazione tra –25 e +13 con 6 bit, cioè:

$$(-25) - (+13) = \text{cioè } (100111)_{CA2} - (001101)_2 =$$

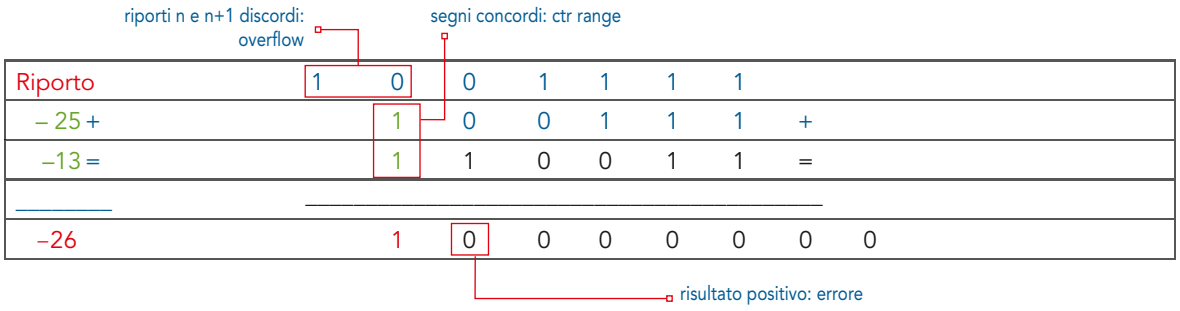
Innanzitutto determiniamo il range di rappresentazione, che è $[-2_{6-1}, +2_{6-1}-1] = [-32,+31]$; trasformiamo l'operazione in una somma:

$$(-25) + (-13) \Rightarrow 100111_{cp2} + (-001101)_2 =$$

Dato che i due addendi hanno segno concorde, esiste la possibilità di avere overflow.

Codifichiamo anche il secondo numero negativo in complemento a 2 ottenendo

$$100111_{cp2} + 110011_{cp2} =$$



I riporti nelle colonne n e n +1 sono diversi: siamo in presenza di overflow.

Il risultato ottenuto è errato: infatti avremmo dovuto ottenere –38, che non è contemplato nel range codificabile con 6 bit.

ESEMPIO 9

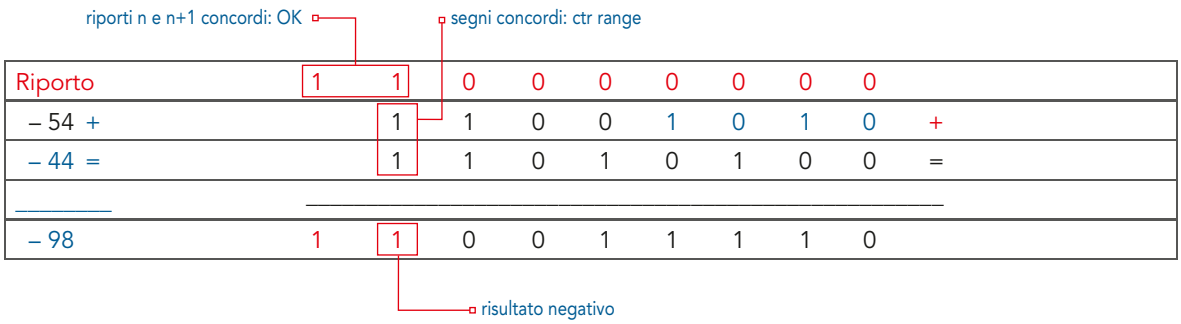
Vediamo un ultimo esempio, dove eseguiamo la sottrazione tra –54 e –44 con 8 bit, cioè

$$(-54) + (-13) = \text{cioè } (11001010)_{cp2} - (11010100)_{cp2} =$$

Innanzitutto determiniamo il range di rappresentazione, che è $[-2^{8-1}, +2^{8-1}-1] = [-128,+127]$.

Dato che i due addenti hanno segno uguale, esiste la possibilità di avere overflow.

Eseguiamo la somma in complemento a 2:



Riporto	1	1	0	0	0	0	0	0	0	
– 54 +		1	1	0	0	1	0	1	0	+
– 44 =		1	1	0	1	0	1	0	0	=
– 98	1	1	0	0	1	1	1	1	0	

I riporti nelle colonne n e n +1 sono uguali: non siamo in presenza di overflow.

Essendo il riporto uguale a 1, dato che i due numeri sono concordi e il risultato ha lo stesso segno, il risultato è corretto.

In effetti risulta un numero negativo pari a $-01100010_2 = -98_{10}$, che è proprio –54 –44, ed è compreso nel range di rappresentazione = [–128,+127].

Eccesso 2^{n-1}

L'ultima rappresentazione dei numeri relativi che analizziamo è quella che prende il nome dalla formula utilizzata per codificare i numeri: eccesso 2^{n-1}, dove n è il numero di bit che vengono utilizzati per la codifica.

Naturalmente al variare del numero di bit e quindi a seconda del valore di n otteniamo una rappresentazione diversa.

L'intervallo di numeri rappresentati con n bit è quello del CA2, cioè $[-2^{n-1}, +2^{n-1} - 1]$: è un intervallo asimmetrico e una semplice rappresentazione dello zero.

Per effettuare la codifica di un numero in eccesso 2^n-1 esistono due comode regole pratiche, da utilizzarsi a scelta caso per caso, a secondo della convenienza: la prima regola è comoda quando si hanno già i numeri in CA2.

Prima regola pratica

I numerali in eccesso 2^{n-1} si ottengono da quelli in CP2 complementando il bit più significativo.

Per esempio, con n = 4 bit e quindi con eccesso 8, l'intervallo è [–8, +7] e abbiamo:

–3: 0101, così ottenuto: $(+011)_2 \Rightarrow (1101)_{cp2} \Rightarrow (0101)_{EC8}$ (= –3 + 8 = 5);
+4: 1100, così ottenuto: $(-100)_2 \Rightarrow (0100)_{cp2} \Rightarrow (1100)_{EC8}$ (= +4 + 8 = 12);
+7: 1001, così ottenuto: $(-111)_2 \Rightarrow (0001)_{cp2} \Rightarrow (1001)_{EC8}$ (= +9 + 8 = 17).

Seconda regola pratica

I numerali in eccesso 2^{n-1} si ottengono sommando al numero x il valore 2^{n-1} e trasformandolo in binario.

Quindi per codificare ogni numero si effettuano due passaggi:
- si somma al numero relativo il valore 2^{n-1};
- si codifica in binario il numero così ottenuto.

Vediamo alcuni esempi.

Effettuiamo la codifica su 5 bit, n = 5, e quindi il valore da aggiungere a ogni numero è:

$$2^{n-1} = 2^{5-1} = 16$$

Codifichiamo i seguenti numeri:

5	$5 + 2^{n-1} = 5 + 16$	21	10101
−16	−16 + 16	0	00000
0	0 + 16	16	10000
15	15 + 16	31	11111

Completiamo la codifica di tutte le possibili configurazioni, come riportato nella tabella a lato: ▶

15	11111
14	11110
	...
2	10010
1	10001
0	10000
−1	01111
−2	01110
	...
−15	00001
−16	00000

Osserviamo che questa codifica non ha fatto altro che "traslare" i numeri posizionando come 0 il numero che aveva il maggior valore negativo (−16) e "trasformando" in questo modo tutti i numeri in numeri positivi.

Infatti il numero più grande che viene codificato, $(15)_{10}$, equivale come codifica al numero binario $11111_2 = 31_{10}$, che si ottiene sommando 16_{10} a 15_{10}: 16_{10}. È la quantità che viene sommata a tutti i numeri in questa rappresentazione con 5 bit.

La decodifica è immediata: si decodifica il numero considerandolo binario e quindi gli si sottrae il valore 2^{n-1}. Vediamo alcuni esempi:

111 => rappresenta 7 in binario su 3 bit, quindi $2^{n-1} = 2^2 = 4$: $\text{decod}(111)_{ec4} = 7 - 4 = 3$;
0111 => rappresenta 7 in binario su 4 bit, quindi $2^{n-1} = 2^3 = 8$: $\text{decod}(0111)_{ec8} = 7 - 8 = -1$;
1011 => rappresenta 11 in binario su 4 bit, quindi $2^{n-1} = 2^3 = 8$: $\text{decod}(1011)_{ec8} = 11 - 8 = 3$;
01011 => rappresenta 11 in binario su 5 bit, quindi $2^{n-1} = 2^4 = 16$: $\text{decod}(01011)_{ec16} = 11 - 16 = -5$.

Questa notazione è poco usata ma è alla base della rappresentazione in virgola mobile (*floating point*) trattata nella prossima Unità didattica.

Verifichiamo le competenze

Esprimi la tua creatività

Modulo e segno

1 Rappresenta in modulo e segno su 8 bit i numeri $+37_{10}$, -37_{10}, -100_{8}, $-3B_{16}$:

$+37_{10}$ -> _____
-37_{10} -> _____
-100_{8} -> _____
$-3B_{16}$ -> _____

2 Converti in decimale i seguenti numeri in codice modulo e segno su 4 bit:

0011 -> _____
0111 -> _____
1010 -> _____
1111 -> _____

3 Effettua le seguenti somme algebriche tra numeri in codice modulo e segno:

0010 +	1011 +	1101 +	1101 +
0100 =	1110 =	0010 =	0111 =
----	----	----	----
----	----	----	----

Complemento

1 Scrivi per i seguenti numeri le rappresentazioni, su 8 bit, in modulo e segno, complemento a 1 e complemento a 2:

	Modulo e segno	Complemento a 1	Complemento a 2
+18			
+115			
+79			
-69			
-3			
-100			
-126			
-111			
0			
99			
-99			
64			
-64			
+10			
-10			
+56			
-65			

Esercizi guidati

Dati due numeri decimali, trasformali in numeri in CA2 con 8 bit, poi esegui la somma algebrica e commenta il risultato.

1 $(-51)_{10} + (-98)_{10}$

Conversione in binario dei valori assoluti:
51 = _____$_2$
98 = _____$_2$

Rappresentazione in CA2:
−51 = _____$_{ca2}$
−98 = _____$_{ca2}$

Somma algebrica:
_____ +
_____ =

_____

Commento
C'è overflow?
Sì [ ] No [ ]

dato che _____

2 $(-54)_{10} + (-44)_{10}$

Conversione in binario dei valori assoluti:
54 = _____$_2$
44 = _____$_2$

Rappresentazione in CA2:
−54 = _____$_{ca2}$
−44 = _____$_{ca2}$

Somma algebrica:
_____ +
_____ =

_____

Commento
C'è overflow?
Sì [ ] No [ ]

dato che _____

3 $(96)_{10} + (69)_{10}$

Conversione in binario dei valori assoluti:
96 = _____$_2$
69 = _____$_2$

Rappresentazione in CA2:
96 = _____$_{ca2}$
96 = _____$_{ca2}$

Somma algebrica:
_____ +
_____ =

_____

Commento
C'è overflow?
Sì [ ] No [ ]

dato che _____

4 $(+50)_{10} + (-50)_{10}$

Conversione in binario dei valori assoluti:
50 = _____$_2$
50 = _____$_2$

Rappresentazione in CA2:
50 = _____$_{ca2}$
−50 = _____$_{ca2}$

Somma algebrica:
_____ +
_____ =

_____

Commento
C'è overflow?
Sì [ ] No [ ]

dato che _____

UNITÀ DIDATTICA 3

NUMERI REALI IN VIRGOLA MOBILE

IN QUESTA UNITÀ IMPAREREMO...

- a rappresentare i numeri in base 10 in virgola mobile
- la notazione floating point IEEE-P754

I numeri reali in virgola mobile

Ricapitoliamo brevemente due particolarità dei numeri reali che ci saranno utili per la trattazione dei numeri in virgola mobile:

1. sono un *soprainsieme* dei numeri interi, quindi alcune proprietà dei numeri interi possono non essere più verificate;
2. un numero reale può non essere finitamente rappresentabile, qualunque siano le cifre a disposizione:
   - numeri irrazionali (non rappresentabili finitamente in forma esplicita in nessuna base: $\sqrt{2}$, π, ...);
   - numeri razionali periodici nella base di rappresentazione utilizzata.

Abbiamo già visto come un numero razionale può risultare periodico o meno a seconda della base usata per rappresentarlo.
Infatti, i due numeri periodici in base 10
$1/3 = (0.333333333333...)_{10} = (0.1)_3$
$8/7 = (1.142857142857...)_{10} = (1.1)_7$
non lo sono rispettivamente in base 3 e in base 7.

La rappresentazione dei numeri in virgola fissa spesso comporta lo spreco di numerosi bit e non è in grado di descrivere tutte le casistiche di informazione numerica.
Per esempio, se utilizziamo una notazione con un numero di bit n_I per la parte intera e n_F per la parte frazionaria come indicato nello schema seguente:

Parte intera	Parte frazionaria
n_I bit	n_F bit

e dobbiamo memorizzare:
- il peso di una Ferrari F-2011, che approssimativamente è di 640 kg ~ 2^{19} grammi, dove la parte intera deve essere lunga 20 bit per poterlo rappresentare mentre la parte decimale è inutilizzata;
- il peso di una molecola di CO_2 ~ 10^{-23} grammi ~ 2^{-80} grammi, dove la parte intera non viene utilizzata mentre sono necessari 80 bit per la parte frazionaria.

La notazione in virgola fissa ci permette di fissare a piacere la posizione della virgola e quindi di scegliere il range di variabilità in modo di avere la massima qualità di rappresentazione.
Se nello stesso sistema dovessimo codificare entrambe le situazioni precedenti in virgola fissa avremmo bisogno di almeno 100 bit con un enorme spreco di spazio in quanto per la maggior parte sono inutilizzati oppure hanno valore uguale a 0.
La rappresentazione in virgola fissa non è quindi adeguata quando il range di variabilità non è noto a priori oppure assume qualunque valore, come nel caso che si deve codificare contemporaneamente sia numeri infinitesimi (peso dell'atomo) che enormi (volume della terra).

Per ovviare a questi problemi è possibile effettuare una rappresentazione dei numeri reali detta in mantissa e resto.

MANTISSA E RESTO

Dati un numero reale **N**, una base **B** e un naturale **n**, è sempre possibile rappresentare il valore reale **N** come somma di due contributi, di cui il primo costituito esattamente da n cifre.

In particolare, per ogni numero reale N si possono determinare:
- una mantissa m (reale) di esattamente n cifre
- un resto r (reale)
- un esponente esp (intero)

tali da esprimere N come:

$$N = m * B^{esp} + r$$

Esistono infinite triple (m r esp) in grado di rappresentare nella stessa base B, a parità di n, lo stesso valore reale.

Vediamo per esempio come è possibile codificare i numeri **N** = 31.4357, **B** = 10 e **n** = 4:

esp	scomposizione	m	r
−1	$314{,}3 * 10^{-1} + 570 * 10^{-1-4}$	314.3	.057
0	$31{,}43 * 10^{0} + 57 * 10^{0-4}$	31.43	.0057
1	$3{,}143 * 10^{1} + 5{,}7 * 10^{1-4}$	3.143	.00057
2	$,3143 * 10^{2} + ,57 * 10^{2-4}$	.3143	.000057
3	$,0314 * 10^{3} + ,357 * 10^{3-4}$	.0314	.00000357
4	$,0031 * 10^{4} + ,4357 * 10^{4-4}$	.0031	.0000004357
5	$,0003 * 10^{5} + ,14357 * 10^{5-4}$	.0003	.000000014357
6	$,0000 * 10^{6} + ,314357 * 10^{6-4}$	.0000	.00000000314357

Questa rappresentazione dei numeri reali prende il nome di rappresentazione in virgola mobile in quanto la virgola può essere "spostata" a piacimento in modo da ottenere per la mantissa una rappresentazione particolare.

NUMERO IN VIRGOLA MOBILE NORMALIZZATO

Quando la mantissa è minore di 1 e la prima cifra a destra della virgola è diversa da 0, il numero in virgola mobile si dice normalizzato.

Nella tabella precedente il numero normalizzato è quello ottenuto con esponente uguale a 2: $-0{,}3143570 \cdot 10^2$ è normalizzato perché la prima cifra a destra della virgola è "3".

La normalizzazione permette di avere, a parità di cifre usate per la mantissa, una maggiore precisione.

Vediamo due esempi dove utilizziamo cinque cifre per codificare la mantissa e variamo il valore dell'esponente:

1. codifichiamo il numero +45,6768 e otteniamo:
   - con esp = 2 $+45{,}6768 = +0{,}\mathbf{45676} \cdot 10^2$ con resto 0,0008, che **è** trascurabile;
   - con esp = 4 $+45{,}6768 = +0{,}\mathbf{00456} \cdot 10^4$ con resto 0,0768, che **non** è trascurabile;
   - con esp = 6 $+45{,}6768 = +0{,}\mathbf{00004} \cdot 10^6$ con resto 0,00056768, che è praticamente tutto il numero che dovevamo rappresentare;
2. codifichiamo il numero +532,4321 e otteniamo:
   - con esp = 3 $+532{,}4321 = +0{,}\mathbf{53243} \cdot 10^3$ con resto 0,0021, che **è** trascurabile;
   - con esp = 5 $+532{,}4321 = +0{,}\mathbf{00532} \cdot 10^5$ con resto 0,4321, che **non** è trascurabile;
   - con esp = 7 $+532{,}4321 = +0{,}\mathbf{00002} \cdot 10^7$ con resto 0,534321 (idem come sopra).

Tra tutte le possibili rappresentazioni in virgola mobile viene adottata la notazione normalizzata, in modo da avere una sola rappresentazione per ogni dato, che ci permette di trascurare il resto perché sia significativo solo il primo termine, cioè:

$$\mathbf{N} = m * \mathbf{B}^{esp}$$

La codifica binaria dei numeri reali in virgola mobile

Un numero in virgola mobile può essere scritto trascurando il resto secondo il seguente formato:

$$N = \text{mantissa} \times 2^{\text{esponente}}$$

È quindi necessario memorizzare due numeri, uno per la mantissa e uno per l'esponente, con il loro segno. Un esempio di formato del numero potrebbe essere il seguente:

dove indichiamo:

- S_M = il segno della mantissa;

- n_M = il numero di bit riservati alla mantissa;
- mantissa = la codifica binaria della mantissa;
- S_E = il segno dell'esponente;
- n_E = il numero di bit riservati all'esponente;
- esponente = la codifica binaria dell'esponente.

Sia la mantissa che l'esponente hanno un numero finito di cifre e quindi gli intervalli rappresentati saranno sempre limitati, in funzione del numero dei bit (32, 64 ecc.); in ogni caso, avremo degli errori dovuti all'arrotondamento.

Codifica della mantissa

In generale ci sono due modalità di codifica della mantissa:
- mantissa normalizzata;
- mantissa denormalizzata.

La mantissa normalizzata

La mantissa è normalizzata quando si "trasla" la virgola in modo da togliere gli zeri in prossimità di essa e centrare così il range dei valori rappresentabili.
Per esempio, nei casi seguenti normalizziamo la mantissa in modo da avere un solo bit uguale a 1 a sinistra della virgola.

ESEMPIO 1

Normalizziamo il seguente numero binario:

$$101.111_2$$

"spostiamo" la virgola a sinistra di 2 posizioni ottenendo il numero seguente:

$$1.01111 * 2^2$$

il numero ottenuto è normalizzato.

ESEMPIO 2

Normalizziamo il seguente numero binario:

$$0.001011_2$$

"spostiamo" la virgola a destra di 3 posizioni ottenendo il numero seguente:

$$1.0110 * 2^{-3}$$

il numero ottenuto è normalizzato.

ESEMPIO 3

Normalizziamo il seguente numero binario:

$$100100_2$$

"spostiamo" la virgola di sinistra di 5 posizioni ottenendo il numero seguente:

$$1.001 * 2^5$$

il numero ottenuto è normalizzato.

Dato che il numero a sinistra della virgola nella forma normalizzata è sempre uguale a 1 lo potremo "sottintendere": questa codifica prende il nome di codifica bit hidden (*nascosto*). Le codifiche con bit hidden codificano solo la parte decimale della mantissa, risparmiando un bit e quindi raddoppiando il range di rappresentazione a parità di bit.

Mantissa denormalizzata

Come vedremo in seguito, la codifica con mantissa denormalizzata viene utilizzata contemporaneamente a quella normalizzata per estendere la capacità di rappresentare numeri molto piccoli. In questo caso non vengono effettuate operazioni sui numeri da codificare, che sono del tipo

$$0{,}000x$$

cioè sono sempre numeri decimali minori di 0.

Vedremo nella rappresentazione IEEE-P754 come indicare convenzionalmente la situazione di mantissa denormalizzata rispetto alla normale codifica di mantissa normalizzata.

Zoom su...

CODIFICHE DELLA MANTISSA NORMALIZZATA

In generale ci sono quattro modalità di codifica, come evidenziato nei casi seguenti:
A) prima cifra significativa diversa da 0 alla destra del punto di radice:

1 senza bit nascosto

$$N = 0.1xxxxxxxx * 2^{esp}$$

2 con bit nascosto

$$N = 0.\underset{\text{sottinteso}}{1}xxxxxxxx * 2^{esp}$$

B) prima cifra significativa diversa da 0 alla sinistra del punto di radice:

1 senza bit nascosto

$$N = 1.xxxxxxxx * 2^{esp}$$

2 con bit nascosto

$$N = \underset{\text{sottinteso}}{1}.xxxxxxxx * 2^{esp}$$

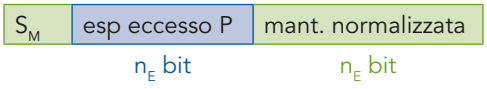
Quella utilizzata nella rappresentazione IEEE-P754 è l'ultima.

Codifica dell'esponente

L'esponente può essere codificato in una delle forma viste per la codifica dei numeri relativi, e cioè:

- modulo e segno;
- complemento a due;
- complemento a uno;
- in eccesso P.

La forma più utilizzata è quella in eccesso P, dove P è una costante che prende il nome di polarizzazione: in questo modo possiamo "risparmiare" il bit di segno dell'esponente in quanto la notazione eccesso P trasla i numeri negativi di P posizioni.



La struttura del numero in virgola mobile diviene quindi la seguente:
dove indichiamo:

- S_M = il segno della mantissa;
- n_M = il numero di bit riservati alla mantissa;
- mantissa = la codifica binaria della mantissa normalizzata;
- n_E = il numero di bit riservati all'esponente;
- esponente = la codifica binaria dell'esponente in eccesso P.

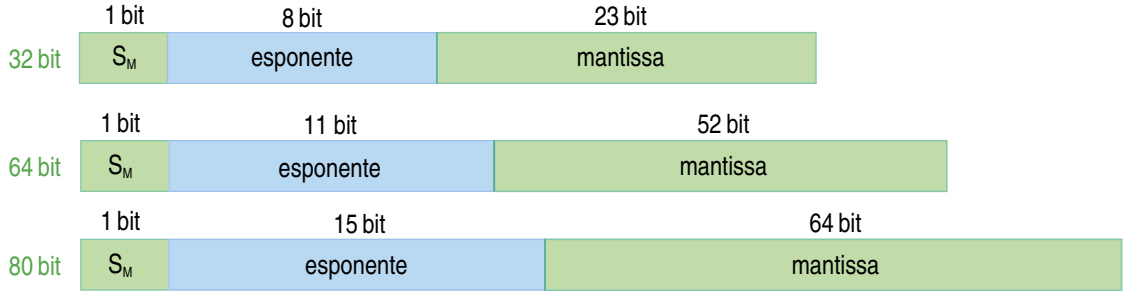
> Il valore P da aggiungere all'esponente è anche detto bias (dall'inglese *to bias*, "influenzare").

Float in singola precisione IEEE-P754

È la rappresentazione standard utilizzata per codificare i reali nei processori della famiglia Intel 80x86. Sono previsti tre diversi formati:

Ⓐ singola precisione: 32 bit;
Ⓑ doppia precisione: 64 bit;
Ⓒ precisione estrema: 80 bit.

Sono chiamati rispettivamente short real, long real e temporary real e hanno la seguente suddivisione dei bit:



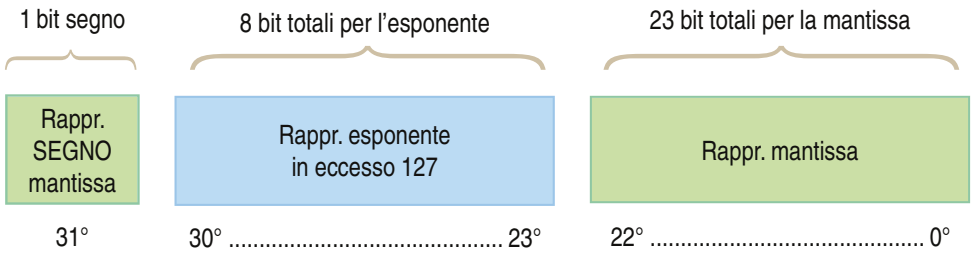
Come vedremo, non tutte le configurazioni sono utilizzate per rappresentare i numeri nella forma mantissa ed esponente, ma parte di queste vengono utilizzate per codifiche particolari, come per esempio:
- la codifica dello 0 e dell'infinito;
- la codifica con mantissa denormalizzata;
- altre codifiche speciali.

Float in singola precisione IEEE-P754 a 32 bit

Utilizza 32 bit, quindi 4 byte per rappresentare i numeri, mentre la mantissa viene rappresentata normalizzata con la prima cifra significativa diversa da 0 alla sinistra del punto di radice
N = 1.xxxxxxxx x 2^{esp} che viene sottointesa, codificando solo la quantità a destra della virgola.

I 32 bit sono così utilizzati:

- 1 bit (0 = positivo, 1 = negativo) per il segno del numero;
- 8 bit per l'esponente intero *esp*, codificato con *eccesso 127* (2^{8-1}-1);
- 23 bit per la codifica della mantissa *m* (cioè *n* = 23 bit effettivi, contando l'MSB) nell'ordine dal *meno* significativo al *più* significativo.



Vediamo alcuni esempi.

ESEMPIO 1

Codifichiamo il numero $\mathbf{V = 1.0_{10}}$:
1. il numero è positivo, quindi il bit di segno è uguale a 0;
2. come primo passo codifichiamo il numero in binario:
$$\mathbf{1.00_2}$$
3. procediamo con la normalizzazione, spostando a destra la virgola di 0 posizioni:
$$\mathbf{V = 1.00 * 2^0} \text{ (rappresentato senza errori)}$$
4. codifichiamo ora l'esponente con eccesso 127, al quale dobbiamo sommare 0, che è il valore ottenuto dalla normalizzazione:
$$0 + 127 = 127_{10} = 01111111_2$$

Otteniamo quindi:

Segno	Esponente	Mantissa normalizzata (23 bit MSB = 1 escluso)
0	0111 1111	000 0000 0000 0000 0000 0000

A questo punto lo scomponiamo in 4 byte

byte 1	byte 2	byte 3	byte 4
0011 1111	1000 0000	0000 0000	0000 0000

che in esadecimale viene scritto come **0x3F800000**.

ESEMPIO 2

Codifichiamo il numero $\mathbf{V = 5{,}875_{10}}$:

1. il numero è positivo, quindi il bit di segno è uguale a 0;
2. come primo passo codifichiamo il numero in binario:
   $\mathbf{101.111_2}$ (senza approssimazione)
3. procediamo con la normalizzazione, spostando a destra la virgola di 2 posizioni:
   $\mathbf{V = 1.}01111 * \mathbf{2}^2$ (rappresentato senza errori)
4. codifichiamo ora l'esponente con eccesso 127, al quale dobbiamo sommare 2, che è il valore ottenuto dalla normalizzazione:
   $2 + 127 = 129_{10} = 1000\ 0001_2$

Otteniamo quindi:

Segno	Esponente	Mantissa normalizzata (23 bit MSB = 1 escluso)
0	1000 0001	011 1100 0000 0000 0000 0000

A questo punto lo scomponiamo in 4 byte

byte 1	byte 2	byte 3	byte 4
0100 0000	1011 1100	0000 0000	0000 0000

che in esadecimale viene scritto come **0x40BC0000**.

ESEMPIO 3

Codifichiamo il numero $\mathbf{V = -29{,}1875_{10}}$:

1. il numero è negativo, quindi il bit di segno è uguale a 1;
2. come primo passo codifichiamo il numero in binario:
   $\mathbf{11101.0011_2}$ (senza approssimazione)
3. procediamo con la normalizzazione, spostando a destra la virgola di 4 posizioni:
   $\mathbf{1.}11010011 * \mathbf{2}^4$ (rappresentato senza errori)
4. codifichiamo ora l'esponente con eccesso 127, al quale dobbiamo sommare 4, che è il valore ottenuto dalla normalizzazione:
   $4 + 127 = 131_{10} = 1000\ 0011_2$

Otteniamo quindi:

Segno	Esponente	Mantissa normalizzata (23 bit MSB = 1 escluso)
1	1000 0011	110 1001 1000 0000 0000 0000

A questo punto lo scomponiamo in 4 byte

byte 1	byte 2	byte 3	byte 4
0100 0001	1110 1001	1000 0000	0000 0000

che in esadecimale viene scritto come **0xC1E98000**.

ESEMPIO 4

Codifichiamo il numero $\mathbf{V = -43{,}125_{10}}$:

1. il numero è negativo, quindi il bit di segno è uguale a 1;
2. come primo passo codifichiamo il numero in binario:

$$\mathbf{101011.001_2}$$

3. procediamo con la normalizzazione, spostando a destra la virgola di 5 posizioni:

$$\mathbf{1.}01011001_2 * \mathbf{2^5} \text{ (rappresentato senza errori)}$$

4. codifichiamo ora l'esponente con eccesso 126, al quale dobbiamo sommare 5, che è il valore ottenuto dalla normalizzazione:

$$5 + 126 = 132_{10} = 10000100_2$$

Otteniamo quindi:

Segno	Esponente	Mantissa normalizzata (23 bit MSB = 1 escluso)
0	1000 0100	010 1100 10000000 00000000

A questo punto lo scomponiamo in 4 byte:

byte 1	byte 2	byte 3	byte 4
1 1000010	0010 1100	1000 0000	0000 0000

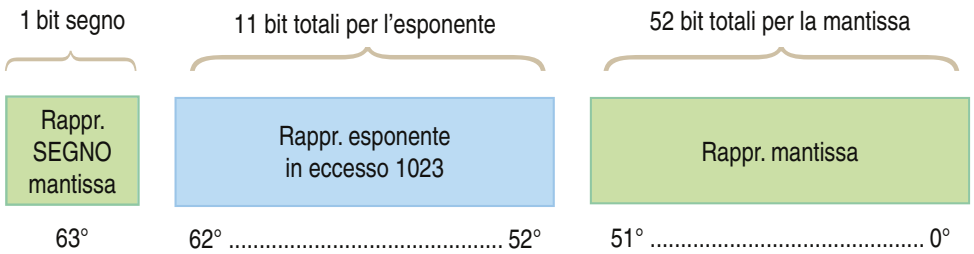
Il turbo C++ utilizza come rappresentazione interna per il tipo float la IEEE-P754: nella tabella seguente riportiamo a titolo di esempio alcune codifiche (che possono essere verificate come esercizio).

Decimale	3	2	1	0
1.0	3F	80	00	00
−1.0	BF	80	00	00
2.0	40	00	00	00
−2.0	C0	00	00	00
0.75	3F	40	00	00
−0.75	BF	40	80	00
100.25	42	C8	80	00
−100.25	C2	C8	80	00

Float in doppia precisione IEEE-P754 a 64 bit

Analogamente alla singola precisione abbiamo la codifica in doppia precisione: cambia solo il numero di bit dato che per rappresentare i numeri essa utilizza 8 byte.
I 64 bit sono così utilizzati:

- 1 bit (0 = positivo, 1 = negativo) per il segno del numero;
- 11 bit per l'esponente intero *esp*, codificato con *eccesso 1023* ($2^{11-1} - 1$):
  – i valori da 1 a 1022 rappresentano gli esponenti negativi da –1022 a –1;
  – i valori da 1023 a 2046 rappresentano gli esponenti positivi da 1 a 1023;
  – i valori estremi (0 e 2047) sono *riservati*;
- 52 bit per la codifica della mantissa *m* (cioè *n* = 52 bit effettivi, contando l'MSB) nell'ordine dal *meno* significativo al *più* significativo.

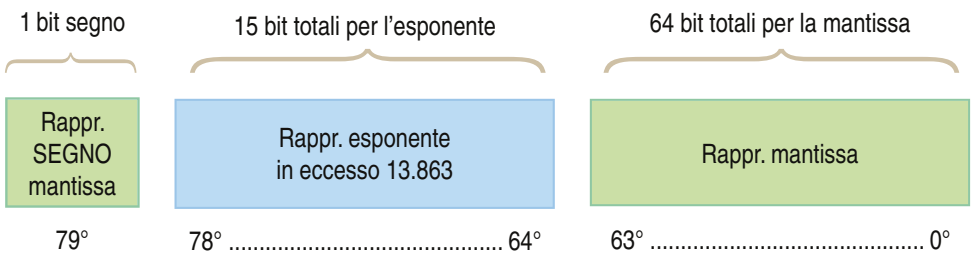


Float in doppia precisione IEEE-P754 a 80 bit

Analogamente alla singola precisione abbiamo la codifica in doppia precisione: cambia solo il numero di bit dato che utilizza 10 byte per rappresentare i numeri.

Gli 80 bit sono così utilizzati:

- 1 bit (0 = positivo, 1 = negativo) per il segno del numero;
- 15 bit per l'esponente intero *esp*, codificato con *eccesso 16.383* (2^{15-1}-1);
- 64 bit per la codifica della mantissa *m* dal *meno* significativo al *più* significativo.



Overflow e underflow

Cerchiamo di individuare quale sia il range dei numeri che si possono rappresentare con la notazione introdotta, per esempio con 16 bit, dove l'esponente è di "soli" 8 bit.

UNDERFLOW

L'undeflow avviene quando l'esponente è troppo piccolo e non è più rappresentabile con gli 8 bit assegnati.

Il valore più grande, sapendo che il bias è 127, sarebbe:

$$1111\ 1111_2 - 127_{10} = (2^8 - 1) - 127_{10} = 255 - 127 = 128$$

Nella realtà l'esponente $1111\ 1111_2$ viene riservato per indicare situazioni anomale, e quindi il più grande esponente utilizzabile è 127.

OVERFLOW

L'overflow avviene quando l'esponente è troppo grande e non è più rappresentabile con gli 8 bit assegnati.

Il valore più piccolo rappresentabile, sempre con bias 127, sarebbe:

$$0000\ 0000_2 - 127_{10} = -127_{10}$$

Anche in questo caso la rappresentazione $0000\ 0000_2$ viene riservata alla rappresentazione dello zero, quindi il più piccolo esponente utilizzabile è −126.
Ricapitolando, abbiamo che per l'esponente, codificato con *eccesso 127* ($2^{8-1} - 1$):

- i valori da 1 a 126 rappresentano gli esponenti negativi da −126 a −1;
- i valori da 128 a 254 rappresentano gli esponenti positivi da 1 a 127;
- i valori estremi (0 e 255) sono *riservati*.

È doveroso osservare che il numero:
0 0000 0000 00000000000000000… … … … … … ..0000000000000
rappresenta lo zero, quindi non c'è una forma normale per lo zero.

Conversione da float a decimali

L'interpretazione di un numero espresso in floating point è piuttosto complicata: chiamando s il segno, e l'esponente, ed m la mantissa si possono avere i seguenti casi:

1. e = 0, m = 0: il valore $(-1)^{8*}(0)$ è utilizzato per la codifica dello 0 (sia + 0 che −0);
2. e = 0, m<>0: il numero è nella forma non normalizzata (rappresenta valori molto piccoli);
3. 0 < e < 255: il numero è normalizzato nella forma $(-1)^s \cdot (1 + 0.\text{mantissa}) \cdot 2^{\text{esponente}}$
4. e = 255, m = 0: il valore è $(-1)^s + \infty$, cioè un numero infinitamente grande o piccolo;
5. e = 255, m < > 0: non è un numero valido (detto NAN, *Not A Number*) e queste configurazioni possono essere utilizzate per codificare informazioni di errore.

Per prima cosa è quindi necessario isolare l'esponente e la mantissa e in base ai loro valori si individua in quale delle cinque situazioni sopra elencate ci si trova: quando siamo nel terzo caso ricostruiamo il numero nel modo seguente:

$$N = (-1)^s \cdot (1 + 0.mantissa) \cdot 2^{esponente}$$

Vediamo alcuni esempi.

ESEMPIO 5

Deriviamo il numero decimale corrispondente alla seguente rappresentazione in singola precisione secondo lo standard IEEE 754 a 32 bit:

0 01111110 00000000000000… 00000000000

Scomponiamo i bit in segno, esponente e mantissa

Segno	Esponente	Mantissa
0	01111110	00000000000000000000000

e deduciamo che:
- segno: è positivo (e lo si ottiene anche dalla formula $(-1)^0 = 1$);
- esponente: $01111110_2 = 126_{10}$;
- posizioni di spostamento della virgola per la normalizzazione: 126 – 127 = –1 (1 a sinistra);
- mantissa: $1.0 * 2^{-1} => 0.1_2 = 0.5_{10}$

Quindi il numero 0 01111110 00000000000000 in float 32 bit corrisponde al numero decimale V = 0.5_{10}.

ESEMPIO 6

Deriviamo il numero decimale corrispondente alla seguente rappresentazione in singola precisione secondo lo standard IEEE 754 a 32 bit:

1 01111110 11000000000000000000000

Scomponiamo i bit in segno, esponente e mantissa

Segno	Esponente	Mantissa
1	01111110	11000000000000000000000

e deduciamo che:
- segno: è negativo (e lo si ottiene anche dalla formula $(-1)^1 = -1$);
- esponente: $01111110_2 = 126_{10}$;
- posizioni di spostamento della virgola per la normalizzazione: 126 – 127 = –1 (1 a sinistra);
- mantissa: $1.11 * 2^{-2} => 0.111_2 = -0.875_{10}$.

Quindi il numero 1 01111110 11000000000000 in float 32 bit corrisponde al numero decimale V = -0.875_{10}.

ESEMPIO 7

Deriviamo il numero decimale corrispondente alla seguente rappresentazione in singola precisione secondo lo standard IEEE 754 a 32 bit:

0xC3EA80000

Trasformiamolo in binario:

0xC3EA80000 = 0011 1110 1010 1000 0000 0000 0000 0000

Scomponiamolo in segno, esponente e mantissa

Segno	Esponente	Mantissa
0	01111101	01010000000000000000000

e deduciamo che:

- segno: è negativo;
- esponente: $01111101_2 = 125_{10}$;
- posizioni di spostamento della virgola per la normalizzazione: 125 – 127 = –2 (2 a sinistra);
- mantissa: $1.0101 * 2^{-2} => 0.010101_2 = 0.328125_{10}$.

Quindi il numero 0xC3EA80000 in float 32 bit corrisponde al numero decimale V = 0.328125_{10}.

Zoom su...

MANTISSA DENORMALIZZATA

I vincoli imposti dalla forma normalizzata non permettono di sfruttare appieno la gamma di valori rappresentabili con 32 e 64 bit.
Nella notazione IEEE-P754 a 32 bit il numero più piccolo rappresentabile è il seguente

$$1,0 \times 2^{-126}$$

in quanto la mantissa normalizzata deve avere valore 1 prima della virgola.

Togliendo questo vincolo il numero più piccolo sarebbe

$$0,000000\ldots1$$

cioè un numero con il valore 1 come ultimo bit, quindi:

$$1 \times 2^{-23}$$

Perciò il numero più piccolo rappresentabile diviene:

$$1,0 \times 2^{-126} \times 2^{-23} = 2^{-149}$$

Quando si vuole "denormalizzare" la mantissa convenzionalmente si utilizza la configurazione di tutti zeri per l'esponente. Quindi il numero più piccolo risulta essere così codificato

Segno	Esponente	Mantissa denormalizzata
0	0000 0000	000 0000 0000 0000 0000 0001

che in esadecimale viene scritto come 0x00000001.

Errori e arrotondamento

Rappresentando un numero reale n in una notazione floating point si può commettere un errore di approssimazione quando il numero è periodico o comunque irrazionale nella base 2: in realtà il numero che rappresentiamo è un "numero diverso" da quello di partenza, è cioè un razionale n′ con un numero limitato di cifre significative.

ERRORE ASSOLUTO

La differenza tra il numero che vogliamo rappresentare n e quello che codifichiamo n' prende il nome di errore assoluto:

$$e_A = n - n'$$

ERRORE RELATIVO

Il rapporto tra l'errore assoluto e il numero che vogliamo rappresentare n prende il nome di errore relativo:

$$e_R = \frac{e_A}{n} = \frac{n - n'}{n}$$

Si può facilmente dimostrare che se la mantissa è normalizzata, l'errore relativo massimo è costante su tutto l'intervallo rappresentato ed è pari a un'unità sull'ultima cifra rappresentata.

Per esempio, con 10 cifre frazionarie l'errore relativo è $e_R = 2^{-10}$.

Nelle notazioni non normalizzate l'errore relativo massimo non è costante.

Rappresentare numeri periodici IEEE 32

Vediamo due esempi di rappresentazione approssimata dei numeri in floating point nel caso in cui la mantissa risulta essere periodica.

Per esempio codifichiamo il numero V = $.1_{10}$, la cui codifica binaria è rappresentata a lato: ▶

1.	.1 × 2 = 0.2 riporto **0**
2.	.2 × 2 = 0.4 riporto **0**
3.	.4 × 2 = 0.8 riporto 0
4.	.8 × 2 = 1.6 riporto **1**
5.	.6 × 2 = 1.2 riporto **1**
6.	.2 × 2 = 0.4 riporto 0
7.	…

Quindi:

$$V = .1_{10} = .0001100110011001100110011001100110011\ldots_2 = . .0(0011)_2$$

Normalizziamo la mantissa ottenendo:

$$V = .(1100) * 2^{-3} = .\mathbf{1100\ 1100\ 1100\ 1100\ 1100\ 1100}\ 1100\ldots * 2^{-3}$$

Dato che il numero è periodico, la mantissa non potrà mai essere completamente rappresentata, quindi si introduce un errore di approssimazione.
Abbiamo due possibilità:

- troncare la mantissa ponendo LSB = 0 => m = .11001100 11001100 1100110;
- arrotondare la mantissa ponendo LSB = 1 => m = .11001100 11001100 1100111.

Il C arrotonda i numeri in floating point.

Vediamo un primo esempio, dove rappresentiamo $\mathbf{V = .15_{10}}$.

La codifica binaria è la seguente:

$$V = .15_{10} = .00(1001)_2$$

La normalizzazione trasforma il numero in:

$$V = (1.001) * 2^{-3} = 1.001(1001) * 2^{-3}$$

Con un esponente in eccesso 127 è:

$$127 - 3 = 124_{10} = 0111\ 1100_2$$

1 Con troncamento

Segno	Esponente	Mantissa normalizzata (23 bit MSB = 1 escluso)
0	0111 1100	0011 0011 0011 0011 0011 001

Lo scomponiamo in 4 byte

byte 1	byte 2	byte 3	byte 4
0011 1110	0001 1001	1001 1001	1 001 1001

e lo scriviamo in esadecimale come 0x3E399999.

2 Con arrotondamento

Segno	Esponente	Mantissa normalizzata (23 bit MSB = 1 escluso)
0	0111 1100	0011 0011 0011 0011 0011 010

Lo scomponiamo in 4 byte

byte 1	byte 2	byte 3	byte 4
0011 1110	0001 1001	1001 1001	1 001 1010

e lo scriviamo in esadecimale come 0x3E39999A.

Per ottenere la rappresentazione con arrotondamento basta sommare 1 alla rappresentazione con troncamento.

Vediamo un secondo esempio, dove rappresentiamo $\mathbf{V = -1/3_{10}}$.

La codifica binaria è la seguente:

$$V = -0,(3)_{10} = .(0101)_2$$

La normalizzazione trasforma il numero in:

$$V = .(0101)_2 = 1.(01)_2 * 2^{-2}$$

Con un esponente in eccesso 127 è:

$$127 - 2 = 125_{10} = 0111\ 1101_2$$

1 Con troncamento

Segno	Esponente	Mantissa normalizzata (23 bit MSB = 1 escluso)
1	0111 1101	01010101010101010101011

Lo scomponiamo in 4 byte

byte 1	byte 2	byte 3	byte 4
1011 1110	1010 1010	1010 1010	1010 1010

e lo scriviamo in esadecimale come 0xBEAAAAAA.

2 Con arrotondamento

Segno	Esponente	Mantissa normalizzata (23 bit MSB = 1 escluso)
1	0111 1101	01010101010101010101010

Lo scomponiamo in 4 byte

byte 1	byte 2	byte 3	byte 4
1011 1110	1010 1010	1010 1010	1010 1011

e lo scriviamo in esadecimale come 0xBEAAAAAB.

Vediamo un terzo esempio, dove rappresentiamo $\mathbf{V = 12.6_{10}}$.

La codifica binaria è la seguente:

$$V = 12.6 = 1100.(1001)_2$$

La normalizzazione trasforma il numero in:

$$V = 1100.(1001)_2 = 1.100(1001) * 2^3$$

Con un esponente in eccesso 127 è:

$$127 + 3 = 130_{10} = 10000010_2$$

1 Con troncamento

Segno	Esponente	Mantissa normalizzata (23 bit MSB = 1 escluso)
0	10000010	1001 0011 0011 0011 0011 001

Lo scomponiamo in 4 byte

byte 1	byte 2	byte 3	byte 4
0 100 0001	01001 001	1001 1001	1001 1001

e lo scriviamo in esadecimale come 0x41499999.

2 Con arrotondamento

Segno	Esponente	Mantissa normalizzata (23 bit MSB = 1 escluso)
0	10000010	1001 0011 0011 0011 0011 0110

Lo scomponiamo in 4 byte

byte 1	byte 2	byte 3	byte 4
0 100 0001	0100 1001	1001 1001	1001 1010

e lo scriviamo in esadecimale come 0x4149999A.

Verifichiamo le competenze

Esprimi la tua creatività

1 **Rappresenta il numero decimale –4.5 secondo lo standard in virgola mobile IEEE-P754 a 32 bit ed esprimilo in notazione esadecimale.**

Segno: _____
Rappresentazione parte intera: 4 = _____
Rappresentazione parte frazionaria: .5 = _____
Rappresentazione binaria: 4.5_{10} = _____
Forma normalizzata: N = 1 _____
Esponente: esponente = _____ quindi E = _____ + 127 = $_{10}$ = _____
IEEE-P754: _____
Esadecimale: _____ Hex

2 **Rappresenta il numero decimale 11.876 secondo lo standard in virgola mobile IEEE-P754 a 32 bit ed esprimilo in notazione esadecimale.**

Segno: _____
Rappresentazione parte intera: 11 = _____
Rappresentazione parte frazionaria: .876 = . _____ _____ _____
Rappresentazione binaria: 11.876 = _____ . _____
Forma normalizzata: 1. _____ _____ _____ _____
Esponente: esponente = _____ quindi E = _____ + 127 = $_{10}$ = _____
IEEE-P754: _____ _____ _____ _____
Esadecimale: ___ _____ Hex

3 **Rappresenta il numero decimale 12.6_{10} secondo lo standard in virgola mobile IEEE-P754 a 32 bit ed esprimilo in notazione esadecimale.**

Segno: _____
Rappresentazione parte intera: 12 = _____
Rappresentazione parte frazionaria: .6 = $_{2}$
Rappresentazione binaria: 12.6 = _____ $_{2}$
Forma normalizzata: 1, _____ _____ _____ _____ _____
Esponente: esponente = _____ quindi E = _____ + 127 = $_{10}$ = _____
IEEE-P754: _____ _____ _____ _____
Esadecimale: _____ _____ Hex

4 **Ricava il valore decimale del numero 3F400000 Hex in virgola mobile rappresentato secondo lo standard IEEE-P754 a 32 bit corrispondente a 3F400000 in base 16.**

Notazione binaria: _____ _____ _____ _____
Segno: _____
Esponente: E = _____ _____ _____
esp = _____ 127 = _____
N = _____ _____ _____ _____ =

5 **Ricava il valore decimale del seguente numero in virgola mobile rappresentato secondo lo standard IEEE-P754 a 32 bit: 0 10000000 10000000000000000000000.**

Segno: _____
Esponente: _____ _____ _____ _____
Mantissa: _____
N = _____ _____ _____ _____

6 **Considera la seguente rappresentazione floating point:**

- 1 bit per il segno;
- 4 bit per l'esponente codificato in complemento a 2;
- 19 bit per la mantissa, normalizzata con 1A cifra ≠ 0 a destra del punto di radice senza l'uso di hidden bit.

Rappresenta -20.025_{10}.

Formato: s yyyy xxxxxxxxxx con mantissa 0.1 … … …

Rappresentazione binaria: _____ _____ _____

Parte intera 20_{10} = _____ _____ _____

Parte decimale 0.025 = _____ _____ _____

Forma normalizzata: = _____ _____ _____

Esponente : 5_{10} = _____ _____ _____

Segno: _____

Rappresentazione floating point: _____ _____

7 **Considera la seguente rappresentazione floating point:**

- 1 bit per il segno;
- 5 bit per l'esponente codificato in complemento a 2;
- 15 bit per la mantissa, normalizzata con 1A cifra ≠ 0 a sinistra del punto di radice con hidden bit.

Rappresenta −351.534410.

Formato: s yyyyy xxxxxxxxxxxxxxx con mantissa 1.xxx...xxx

Rappresentazione binaria del valore assoluto: _____ _____

Forma normalizzata: _____ _____ _____ _____

Esponente: 8_{10} = _____ _____ _____

Segno: _____

Rappresentazione floating point: _____ _____ _____

8 **Considera la seguente rappresentazione floating point:**

- 1 bit per il segno;
- 5 bit per l'esponente in complemento a 2;
- 16 bit per la mantissa, normalizzata con 1A cifra ≠ 0 a sinistra del punto di radice senza hidden bit.

Rappresenta 1330.(4631)8.

Forma normalizzata del tipo 1.xxxxxxxxxxxxxxx

Rappresentazione binaria: _____ _____ _____ _____

Forma normalizzata: _____ _____ _____ _____

Esponente: 910 = _____ _____ _____

Segno: _____

Rappresentazione floating point: _____ _____ _____

9 **Considera la seguente rappresentazione floating point:**

- 1 bit per il segno;
- 4 bit per l'esponente in eccesso 2^{n-1};
- 7 bit per la mantissa, normalizzata con 1A cifra ≠ 0 a destra del punto di radice con hidden bit.

Trova il numero decimale corrispondente alla rappresentazione 111001011000.

1 – 1100 – 1011000

Segno: _____

Esponente: 1100_2 = 12 in eccesso $2^4 - 1 = 8$ E = esp + P esp = E − P = 12 − 8 = 4

Mantissa: _____ _____ _____

Numero rappresentato: _____ _____ _____

10 Converti i seguenti numeri decimali frazionari in codifica standard IEEE-754-SP ed esprimila in notazione esadecimale.

Decimale	Segno	Esponente	Mantissa	Esadecimale
1.5				
−1.5				
3.25				
−3.25				
5.125				
−5				
11.25				
−11.25				
213.125				
−213.125				
500.5				
−500.5				
1250.125				
−1250.125				
10250.125				
−10250.125				
1000000				
−1000000				
100000000				
−100000000				

11 Converti i seguenti numeri decimali frazionari periodici in codifica standard IEEE-754-SP ed esprimila in notazione esadecimale a 32 bit.

Decimale	Segno	Esponente	Mantissa	Esadecimale
−12.72				
+14.375				
−46.188				
+7.99				
−12.56				
+5.54				
−2.21				
+0.07				
+3.7				
–0.7				
156.0				
–2.9				

12 Converti i seguenti numeri decimali frazionari periodici in codifica standard IEEE-754-SP ed esprimili in notazione esadecimale a 32 bit.

Decimale	Segno	Esponente	Mantissa	Esadecimale
−12.72				
+14.375				
−46.188				
+7.99				
−12.56				
+5.54				
−2.21				
+0.07				

13 Converti i seguenti numeri decimali frazionari periodici in codifica standard IEEE-754-SP sia in eccesso sia per troncamento ed esprimili in notazione esadecimale a 32 bit.

Decimale	Segno	Esponente	Mantissa	Eccesso	Troncamento
0.(6)					
3.(3)					
4.1[6]					
3.[5]					
14.[6]					
3.[63]					
15.[5]					
−5.8[3]					
41.[6]					
−1.5[90]					
−6.[06]					
363.[63]					

4 MODULO IL SISTEMA OPERATIVO

OBIETTIVI

- Sapere che cosa succede all'accensione del PC
- Conoscere i compiti del sistema operativo
- Conoscere la storia dei sistemi operativi
- Riconoscere i meccanismi di caricamento del programma in memoria
- Conoscere le tecniche di virtualizzazione della memoria
- Descrivere le tecniche di realizzazione del file system
- Conoscere l'hardware dei dispositivi di I/O
- Apprendere il modello client-server

ATTIVITÀ

- Classificare i sistemi operativi
- Descrivere il ciclo di vita di un processo
- Individuare le problematiche per la cooperazione tra processi
- Scegliere le politiche di allocazione del processore
- Classificare le memorie
- Riconoscere il modello client-server
- Classificare le tecniche di gestione delle periferiche
- Affrontare i sistemi di protezione dei dati

UNITÀ DIDATTICA 1

GENERALITÀ SUI SISTEMI OPERATIVI

IN QUESTA UNITÀ IMPAREREMO...

- che cosa succede all'accensione del PC
- a che cosa serve il sistema operativo
- che cos'è il kernel di un sistema operativo

Accendiamo il PC

A differenza di tante altre macchine, il calcolatore non è in grado di funzionare da solo: sappiamo che è composto da un insieme di circuiti elettronici che eseguono istruzioni purché tali istruzioni, che costituiscono i programmi, siano scritte in linguaggio macchina binario, l'unico riconosciuto dal processore.

All'atto dell'accensione è necessario che un particolare programma scritto in binario "avvii" le elaborazioni: questo programma deve per prima cosa essere caricato in memoria centrale RAM (dato che per essere eseguito un programma deve essere presente in memoria centrale) e quindi successivamente mandato in esecuzione.

Questo particolare programma è stato chiamato programma di bootstrap (da *boot*, scarpone, e *strap*, "laccio"); in inglese americano con *bootstrap* si indicano le stringhe sul lato laterale o posteriore degli stivali da cowboy).

Seguiamo ora le operazioni che si susseguono all'accensione del PC. Innanzitutto è doveroso fare alcune premesse:

1. il programma di bootstrap (o semplicemente di boot) viene scritto dal produttore dell'hardware ed è memorizzato in un particolare chip di memoria, quello che prende il nome di memoria ROM (Read Only Memory), cioè una memoria "a sola lettura" in quanto l'utilizzatore la può solo leggere senza modificarne il contenuto;
2. all'accensione del PC l'hardware è predisposto per effettuare il caricamento del programma di boot a partire da un determinato indirizzo (i processori a 32 bit caricano la

prima istruzione dall'indirizzo esadecimale 0xFFFFFFF0 di quello che si chiama IPL, Initial Program Loader) che assume il controllo della CPU all'accensione;

3. ogni componente hardware, appena viene alimentato, manda in esecuzione un programma di autodiagnostica (POST, Power On Self Test), costituito da una serie di test che verificano il corretto funzionamento del dispositivo.

Zoom su...

BOOTSTRAP

La parola bootstrap deriva dall'espressione inglese "*pulling yourself up by your bootstraps*" e fa riferimento alla figura letteraria del Barone di Münchausen che era in grado di sollevarsi tirandosi i lacci degli scarponi; oggi questo modo di dire significa "aiutati da solo" e nel PC sta a indicare che solamente eseguendo questo programma il PC è in grado di attivarsi da solo e di predisporsi per essere operativo.

Accendiamo ora il PC e seguiamo passo passo i singoli eventi:

1. dopo che viene effettuato il POST della scheda madre, se tutti i test danno esito favorevole viene emesso un beep e si procede con l'analogo test della scheda video;
2. successivamente avviene il conteggio della memoria dinamica, che fino a pochi anni fa era possibile vedere (e seguire) in alto a sinistra sullo schermo ma che ora avviene talmente rapidamente che è possibile vedere soltanto la dimensione totale della RAM presente nel PC;
3. seguono i controlli sulla tastiera, sul mouse e su tutte le periferiche di input;
4. gli ultimi controlli sono quelli riferiti alle periferiche collegate, a partire dal disco fisso fino ad arrivare al check di funzionamento (o meglio, di collegamento) della stampante, del modem ecc.

Nei computer Apple il programma di bootstrap è tutto contenuto nella ROM mentre nei PC può essere memorizzato in parte su disco: in questo caso il disco prende il nome di disco di boot (i blocchi nei quali è memorizzato si chiamano boot sector).

ROM

La memoria ROM è costituita da un chip elettronico con modeste capacità di archiviazione (qualche decina di kbyte): dato che è molto lenta rispetto alla RAM (il tempo di accesso per una ROM è di circa 150 ns, mentre per la RAM è di circa 10 ns), le istruzioni contenute nella ROM sono talvolta copiate nella RAM all'avvio.

Al termine di queste operazioni il calcolatore può iniziare a operare e viene caricata in memoria una parte del sistema operativo, il kernel, o nucleo del sistema operativo, che sarà descritto in seguito.

Tutte queste operazioni sono effettuate eseguendo le istruzioni presenti nel programma di boot; nei sistemi IBM sono un insieme di routine software alle quali è stato dato il nome di BIOS (Basic Input-Output System): il BIOS è quello che comunemente viene indicato come ◀ **firmware** ▶, per distinguerlo dal software vero e proprio che viene installato dall'utente, in quanto è un "software stabile" (dall'inglese *firm*, "stabile"), cioè non modificabile.

◀ Il termine firmware indica un programma che non è né software né hardware, ma una "via di mezzo": è residente su un chip ROM, quindi non è modificabile dal programmatore in quanto predisposto e programmato dal produttore dell'hardware.
I calcolatori odierni hanno anche altri dispositivi con un loro firmware oltre al programma di boot; per esempio i dischi rigidi sono controllati da un apposito firmware, così pure i DVD o le schede particolari dedicate. ▶

Il sistema operativo

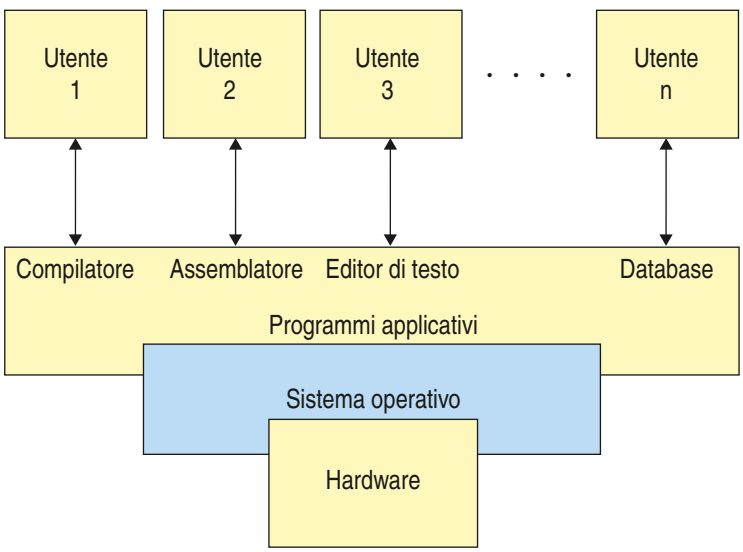
Dal momento della sua accensione il computer rimane in attesa di eseguire i programmi dell'utente: il sistema operativo apparentemente non fa nulla, ma è invece sempre attivo dal momento in cui viene caricato all'accensione della macchina fino allo spegnimento del computer.
Tutto quello che succede nella macchina è controllato dal sistema operativo: diamone una prima definizione.

SO (SISTEMA OPERATIVO)

Con sistema operativo (abbreviato in SO) intendiamo un gruppo di programmi che gestisce il funzionamento del computer agendo come intermediario tra l'utente e il calcolatore.

Il SO fa parte del software di base, termine con il quale si intende l'insieme dei programmi che consentono a un utente di eseguire operazioni base come costruire e mandare in esecuzione un programma.
Il software di base è costituito da:

- il sistema operativo;
- gli editor;
- i traduttori;
- i linker;
- i loader;
- i debugger.



In definitiva si può dire che il software di sistema serve alla macchina per funzionare, mentre il software applicativo serve all'utente per lavorare.

Un computer appena uscito dalla fabbrica non è in grado di funzionare, ma può solo eseguire il boot e arrestarsi con un messaggio di errore quando rileva l'assenza del sistema operativo: in questo caso occorre eseguire l'installazione del sistema prima di qualunque altra operazione.

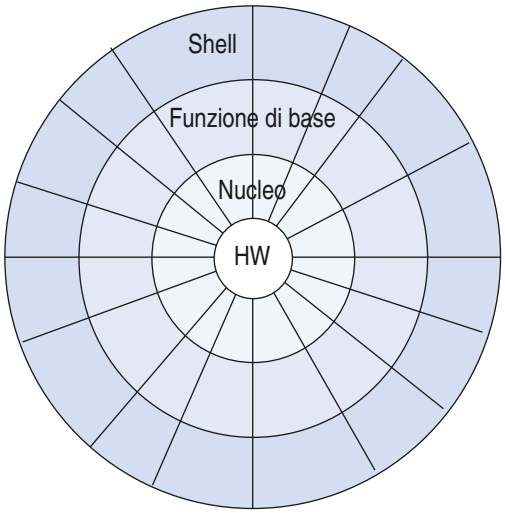
Su una macchina è possibile installare diversi sistemi operativi, per esempio possiamo installare un sistema operativo con licenza di utilizzo a pagamento, tipo Windows, oppure un sistema libero come Linux.
È anche possibile installare contemporaneamente due sistemi operativi sulla stessa macchina, suddividendo il disco fisso in due parti (facendo quelle che si chiamano partizioni del disco) e facendo scegliere all'utente, alla fine del programma di boot, con un menu, quale dei due sistemi deve essere caricato.

Il sistema operativo svolge principalmente due compiti:
- è il gestore delle risorse hardware (CPU, memoria, periferiche);
- fornisce il supporto all'utente per impartire i comandi necessari al funzionamento del computer (fa da interfaccia).

Nel dettaglio, sono gestite dal sistema operativo tutte le funzioni generali della macchina, come l'aspetto grafico delle visualizzazioni su monitor, la scrittura e la lettura dei dischi, la messa in esecuzione e la chiusura dei vari programmi, la ricezione e la trasmissione di dati attraverso tutti i dispositivi di I/O.

RISORSE
Le risorse sono gli elementi hardware o software del PC che vengono usate da specifici programmi per eseguire il proprio compito.

Il sistema operativo risiede sull'hard disk come tutti gli altri programmi e viene caricato nella memoria RAM all'accensione della macchina; abbiamo detto che è composto da un insieme di programmi e questi programmi vengono caricati solo quando l'utente ne richiede il funzionamento: solo una parte del SO, chiamata nucleo (o kernel), rimane sempre caricata in memoria.



Per poter comunicare con l'utente nei sistemi operativi è presente un programma apposito che prende il nome di shell (letteralmente "guscio"), in quanto ha la funzione di fare da interfaccia tra l'utilizzatore e il nucleo, isolandolo e "proteggendolo".

Propria dei sistemi operativi è la stratificazione, cioè l'organizzazione secondo il modello "onion skin" (a buccia di cipolla) elaborato da H.M. Deitel nel 1983: il SO è formato da gusci concentrici che circondano l'hardware; il primo strato (o guscio) è proprio il nucleo che contiene le funzioni che operano a contatto diretto con l'hardware.

Grazie alla presenza della stratificazione del SO, l'utente può dialogare con la macchina senza avere conoscenze approfondite dell'hardware utilizzato: infatti ogni livello offre un insieme di comandi sempre più potenti che accedono a funzioni più evolute del sistema (gestione dei file, esecuzione dei programmi applicativi, operazioni complesse sulle periferiche) senza fare riferimento alla struttura fisica.
Si creano quelle che prendono il nome di macchine virtuali in quanto ogni strato vede gli strati sottostanti come un unico oggetto che non corrisponde ad alcuna macchina fisica ma con il quale interagisce grazie alle procedure che questo mette a disposizione (◀ primitive ▶).

◀ Con il termine primitive ("primitiva") si intende ogni procedura standard per mezzo della quale il kernel del SO mette a disposizione i propri servizi; le invocazioni di primitive sono dette anche supervisor call e verranno descritte nel corso della trattazione. ▶

Zoom su...

VIRTUALE

Nei sistemi operativi il termine "virtuale" viene utilizzato spesso: si parla di macchina virtuale, di memoria virtuale, di risorsa virtuale ecc.
"Virtuale" sta a indicare un qualcosa che fisicamente non esiste ma viene "simulato" tramite software e all'utente finale "sembra" che sia presente realmente.
È anche possibile avere il sistema operativo virtuale, cioè possiamo avere un altro sistema operativo (o più di uno) all'interno di quello che abbiamo installato fisicamente sulla nostra macchina: lanciando il programma di virtualizzazione, si apre una nuova finestra, con all'interno il sistema operativo virtualizzato, che funziona come un sistema operativo vero e proprio.
Con un sistema operativo virtuale si possono provare programmi senza che il nostro venga coinvolto, oppure utilizzare programmi scritti per un altro sistema operativo che sul nostro non potrebbero mai funzionare. L'unico inconveniente è che in questo modo si ha un calo delle prestazioni della macchina.

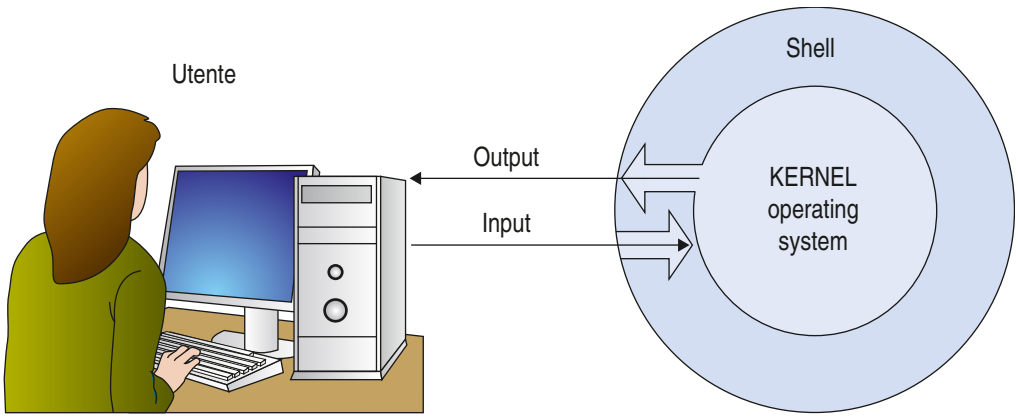
Kernel

Con kernel (nucleo) si intende il "nocciolo" del sistema operativo che avvolge idealmente tutto l'hardware, partendo dalla CPU fino ai i suoi dispositivi fisici, e si occupa di interagire con i programmi applicativi che, ogni volta che necessitano di un servizio da parte di un dispositivo hardware, devono "passare attraverso lui": i programmi utente non devono (e non possono) accedere direttamente ai dispositivi fisici, ma possono utilizzare solo dispositivi logici attraverso primitive di sistema costituenti il kernel.

Quando si sta eseguendo il codice del kernel si dice che il processore gira nel cosiddetto "modo supervisore": vedremo dettagliatamente le operazioni che vengono eseguite dal kernel e tra i compiti fondamentali ricordiamo quelli elencati di seguito, che saranno oggetto della prossima unità didattica:

- avvio e terminazione dei programmi (processi);
- assegnazione della CPU ai diversi processi;
- sincronizzazione tra i processi;
- sincronizzazione dei processi con l'ambiente esterno.

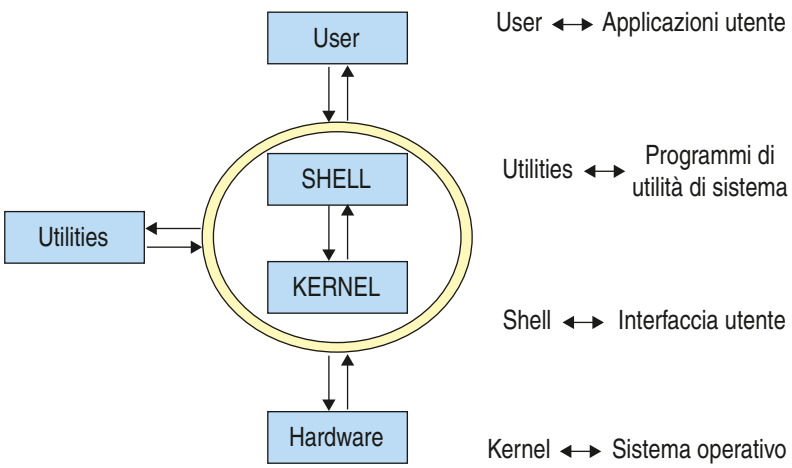
Il kernel "isola" quindi l'hardware dal resto del sistema operativo: questo meccanismo permette di installare lo stesso sistema operativo su piattaforme diverse, a patto che ciascuna di esse abbia a disposizione un kernel specifico (questa è la base su cui si fonda la portabilità di un sistema operativo).



Shell

I servizi offerti dal kernel per quanto di sua competenza e per accedere ai dispositivi e alle risorse disponibili sul computer si ottengono per mezzo di chiamate di funzione.
Alcune di queste funzioni vengono chiamate direttamente dai programmi utente, altre devono invece essere scelte o richiamate dall'utente stesso: per favorire la comunicazione e semplificare l'interfacciamento con il kernel è disponibile un apposito programma, la shell, che, come suggerisce il suo nome, avvolge il kernel come una conchiglia per "proteggerlo".

L'utente può accedere alle funzioni di sistema solo attraverso lo shell, che prende anche il nome di interfaccia utente.



INTERFACCIA UTENTE

Con interfaccia utente (o shell) si intende ciò che si frappone tra la macchina e l'utente, ciò che fa dialogare l'uomo con la macchina: è qualsiasi cosa permetta a un utente di gestire (più o meno) semplicemente le funzionalità di un sistema (anche non informatico).

L'interfaccia utente può essere di tipo CUI (Command User Interface) o GUI (Graphical User Interface):

- le interfacce grafiche di tipo CUI sono tipiche dei sistemi operativi a linea di comando, come per esempio MS-DOS, Unix e Linux (nelle vecchie distribuzioni), che presentano un invito, o prompt;
- le interfacce grafiche di tipo GUI sono tipiche dei sistemi operativi *friendly user,* come per esempio Windows, MacOS e Linux (nelle distribuzioni più recenti).

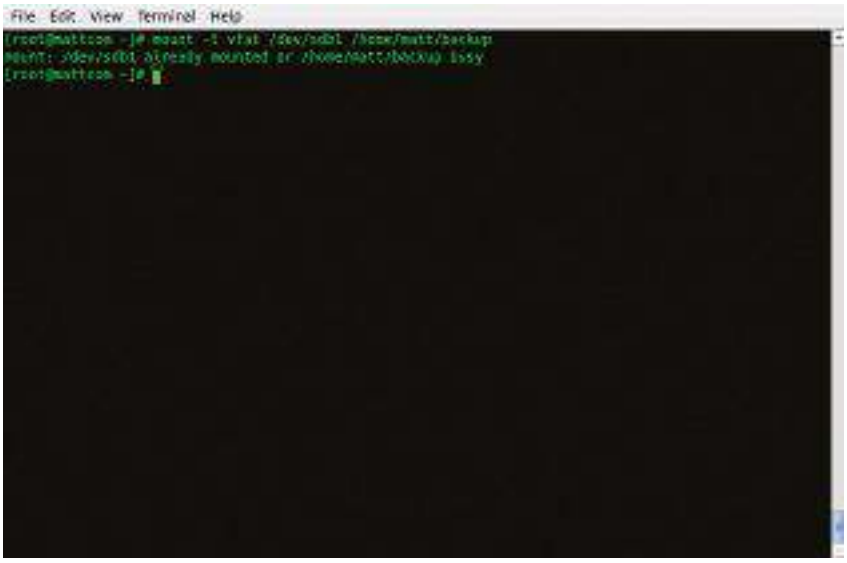


Figura 1 Interfaccia a linea di comando (Linux Red Hat)

Figura 2 Interfaccia grafica (MacOsX Lion10)

Figura 3 Interfaccia grafica (Windows 7)

Lo studio del software di base in generale e dei sistemi operativi in particolare si rivela complesso in quanto in parte è condizionato dall'architettura del calcolatore per cui è progettato tale software: noi affronteremo questo studio solo dal punto di vista concettuale, analizzando i diversi problemi che il sistema operativo deve risolvere, senza però curarci dell'implementazione dei programmi, in modo da mantenere il livello di trattazione valido per ogni tipo di sistema operativo. Nelle prossime unità didattiche partiremo da una veloce trattazione storica per seguire l'evoluzione di cui tali sistemi sono stati oggetto in funzione delle innovazioni tecnologiche.

I sistemi operativi in commercio

Sul mercato sono disponibili diversi SO: la scelta di quello più confacente avviene tenendo conto delle esigenze dell'utente e del campo di applicazione.
Alcuni sono infatti adatti per gestire reti di computer, altri invece per essere usati nel campo della grafica, altri ancora offrono maggiori garanzie di funzionamento in caso di guasti del sistema.
I SO più adatti per la gestione delle reti sono Linux (vedi figura 1) e Windows nella versione Server, per la sicurezza dei dati Unix, e infine, per la grafica, MacOsX Lion (vedi figura 2).

Un'altra suddivisione dei SO tiene conto del tipo di computer sul quale devono essere installati. Per esempio Unix è un SO adatto per computer molto potenti di tipo mini e mainframe, mentre Windows, Linux e MacOS vengono installati su personal computer.

Il primo SO idoneo a essere utilizzato sui personal computer è stato l'MS-DOS (MicroSoft Disk Operative System), in seguito spesso chiamato semplicemente DOS. Possedeva il pregio di occupare poca memoria RAM ma lo svantaggio di richiedere conoscenze tecniche abbastanza approfondite da parte dell'utente. Aveva inoltre un'interfaccia di tipo testuale a riga di comando. Ciò comportava che ciascun comando impartito doveva essere digitato sulla tastiera, costringendo l'utente a una conoscenza dettagliata del linguaggio di tale sistema operativo.

◀ La versione è molto spesso rappresentata da un numero che accompagna il nome di un programma per identificarne l'eventuale aggiornamento. Il numero può essere indicato da 1 a 10, oppure con l'anno di uscita del programma. Windows, per esempio, è stato prodotto nelle seguenti versioni: Windows 3.0 (1990), Windows 98 (1998), Windows 2000 (2000), Windows XP (2003), Windows Vista (2006), Windows 7 (2010), Windows 8 (2012). ▶

L'evoluzione di questo software è senza dubbio rappresentata da Windows (vedi figura 3), prodotto nei primi anni '90 e attualmente il SO più diffuso al mondo. Per tenere il passo con i continui progressi nel campo dell'elettronica, ogni due o tre anni circa escono nuove ◀ versioni ▶ di questo SO.

Verifichiamo le conoscenze

Esercizi a scelta multipla

1 **Il kernel di un sistema operativo:**

- ○ è scritto in assembler
- ○ contiene il programma di bootstrap
- ○ è caricato dopo il bootstrap
- ○ è caricato prima del bootstrap

2 **L'acronimo POST significa:**

- ○ *Power Off Self Test*
- ○ *Power On Security Test*
- ○ *PC On Self Test*
- ○ *Power On Self Test*

3 **Con firmware si intende (indicare l'affermazione errata):**

- ○ software non modificabile
- ○ software contenuto nel BIOS
- ○ componenti hardware come il BIOS
- ○ software a corredo dell'HW

4 **L'acronimo BIOS significa:**

- ○ *Binary Input-Output System*
- ○ *Basic Input-Output System*
- ○ *Binary Input-Output Software*
- ○ *Basic Input-Output Software*

5 **Le primitive possono essere:**

- ○ invocate esplicitamente dai processi
- ○ invocate da istruzioni macchina generate dai compilatori nella traduzione di costrutti di alto livello
- ○ invocate all'interno di funzioni di libreria
- ○ tutte le situazioni precedenti

6 **Quale di queste affermazioni in merito ai SO è vera?**

- ○ il software di base fa parte del SO
- ○ sono scritti in binario
- ○ sono residenti su disco
- ○ sono contenuti nella ROM

7 **Nel software di base è compreso:**

- ○ BIOS
- ○ traduttori
- ○ linker
- ○ caricatori
- ○ debugger

8 **Quale di queste affermazioni in merito alla shell è vera?**

- ○ fa parte del kernel
- ○ viene caricata con il BIOS
- ○ l'interfaccia utente fa parte dello shell
- ○ è residente nella ROM

UNITÀ DIDATTICA 2

EVOLUZIONE DEI SISTEMI OPERATIVI

IN QUESTA UNITÀ IMPAREREMO...

- la storia dei sistemi operativi
- le caratteristiche dei sistemi operativi
- a classificare i sistemi operativi

Cenni storici

Sappiamo che è praticamente impossibile per l'utente utilizzare direttamente l'hardware di un calcolatore: anche se scrivessimo i programmi in linguaggio binario ci troveremmo di fronte a due problemi:

- come immettere programmi nella macchina (Input);
- come estrarre dalla memoria del calcolatore i risultati dei calcoli eseguiti per comunicarli all'esterno (Output).

Queste problematiche sono state le prime a essere affrontate dai progettisti sin dagli anni '40 del secolo scorso: i primi tentativi di facilitare l'interazione uomo-macchina si fecero creando dei programmi e/o sottoprogrammi di uso comune messi a disposizione dei programmatori per essere richiamati nelle diverse applicazioni. Nacque così il software di base, cioè un corredo di programmi di utilità per poter integrare nei programmi l'utilizzo dell'hardware senza dover ogni volta riscrivere tutte le funzioni di basso livello.

La storia dei sistemi operativi segue quindi parallelamente l'evoluzione dell'hardware: i passi da gigante fatti dalla tecnologia e gli sviluppi nel campo delle architetture e delle apparecchiature periferiche rendono necessario lo sviluppo di nuovo software di base.

Possiamo fare una prima classificazione dei sistemi operativi sulla base della tipologia di colloquio uomo-macchina dalle origini dell'informatica fino a oggi:

- sistemi dedicati;
- sistemi a lotti (batch);

- sistemi interattivi:
  - conversazionali;
  - in tempo reale;
  - transazionali.

Sistemi dedicati (1945-1955)

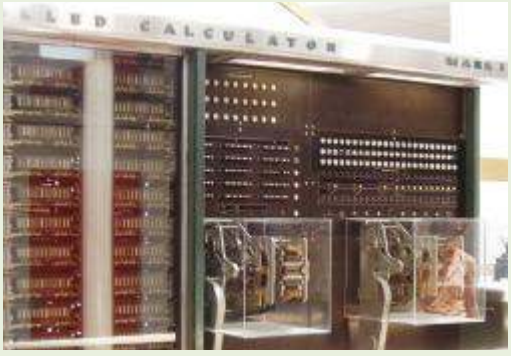
La prima fase, quella compresa tra il 1945 e il 1955, può essere definita come quella dei sistemi dedicati. I calcolatori erano valvolari, occupavano intere stanze, erano lenti, molto costosi e a uso quasi esclusivo di università o centri di ricerca.

Il tempo macchina, inteso come tempo di utilizzo della CPU, era una risorsa pregiata concessa a pochi utenti privilegiati per brevi periodi (turni), e durante il proprio turno i vari utenti avevano a disposizione l'intero sistema: il programmatore caricava i propri programmi (scritti in linguaggio macchina prima e successivamente in assembler) in memoria, li faceva eseguire e ne controllava l'esecuzione.
Non esisteva un sistema operativo vero e proprio: le poche funzioni disponibili per la gestione dell'hardware venivano inviate impostando interruttori e commutatori della console.
Non esistevano tastiera e monitor, e i risultati dei programmi si leggevano direttamente in alcuni registri rappresentati sul pannello tramite indicatori luminosi.

Zoom su...

MARK1, ENIAC E UNIVAC

Il primo calcolatore è forse quello costruito dall'università di Manchester e chiamato Universal Electronic Computer (successivamente rinominato Mark1) nel 1945: non esisteva alcun linguaggio assemblativo e veniva programmato con un alfabeto a 32 simboli, trasferiti come gruppi di 5 bit in memoria mediante un lettore/perforatore di nastro.



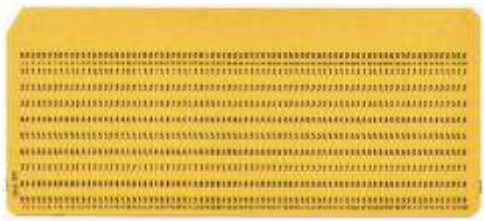
Negli Stati Uniti il primo calcolatore funzionante fu l'ENIAC (1945), una macchina costituita da 19.000 valvole e 1500 relays che occupava un intero locale; il peso complessivo superava le 30 tonnellate; la potenza consumata era di 200 KW. In origine l'ENIAC veniva programmato intervenendo manualmente su interruttori e connessioni dei cavi e solo in un secondo tempo, nel 1948, si iniziarono a scrivere primitive software conservate in tabelle di memoria precursori delle odierne ROM.

L'UNIVAC 1 (1951) è una macchina con un importante significato storico per l'industria informatica perché è stato il primo prodotto a essere venduto al pubblico, cioè creato con uno scopo commerciale. Era dotato di una tastiera di una macchina da scrivere per l'input ed equipaggiato con diverse unità a nastro magnetico. È stato usato per predire correttamente il risultato delle elezioni presidenziali del 1952.

In questi primi calcolatori il sistema operativo consisteva essenzialmente in alcuni programmi per il trasferimento di blocchi di dati tra la memoria principale a quella ausiliaria, costituita da nastri magnetici.

Gestione a lotti (1955-1965)

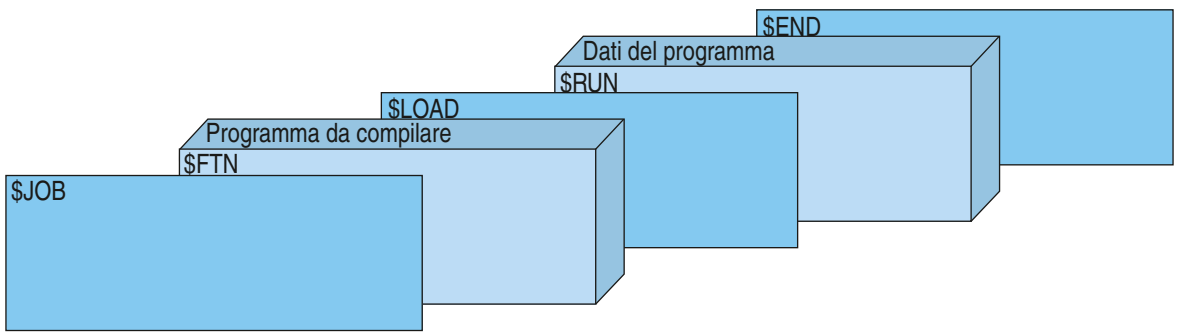
La gestione appena descritta permetteva un basso utilizzo della CPU in quanto la maggior parte del tempo veniva sprecata per caricare i programmi: in un sistema molto costoso sicuramente tale situazione non era soddisfacente.
Il passo successivo fu quello di introdurre un sistema automatico di caricamento dei programmi: si utilizzarono le schede perforate come supporti su cui memorizzare i programmi, i comandi con le chiamate di sistema operativo e i lettori di schede come dispositivi di input.



Ogni scheda perforata conteneva un'istruzione.

L'utente non utilizzava più la console del calcolatore, ma la sua interazione avveniva semplicemente inserendo un pacco di schede (lotto) nell'apposito lettore: questo tipo di gestione fu proprio chiamata gestione a lotti (batch processing) o elaborazione a lotti.

I lavori degli utenti erano costituiti da pacchi di schede perforate dove ogni lotto (chiamato lavoro o job) iniziava con una scheda di identificazione dell'utente e consisteva in una sequenza di passi o step, dove i singoli step erano le funzioni richieste dall'utente al sistema operativo, come la compilazione del programma, il caricamento, l'esecuzione ecc.



Il primo sistema operativo di questo tipo fu sviluppato da General Motors per IBM e prese il nome di "bath monitor" (o semplicemente monitor) e aveva come compito essenziale quello di trasferire il controllo da un job appena terminato a un nuovo job in partenza.

Le schede di controllo utilizzavano un particolare linguaggio per comunicare con il calcolatore: il JCL o Job Control Language.

Un job è delimitato da due schede speciali di controllo: $JOB e $END.

Per esempio il comando $FTN attivava il compilatore Fortran, $RUN richiedeva l'esecuzione, $END terminava l'elaborazione ecc.
Il processore alternava quindi l'esecuzione di istruzioni di controllo ai programmi: gli eventuali dati dovevano essere inclusi nel pacco di schede e quindi i singoli lavori prevedevano anche un'alternanza di schede di istruzioni, di controllo e di schede dati.
I risultati delle elaborazioni finivano stampati su un tabulato comune a tutti i job.

Le principali caratteristiche di questo tipo di sistema possono essere così elencate:
- il sistema operativo è sempre residente in memoria (monitor);
- in memoria centrale è presente un solo job alla volta;
- finché il job corrente non è terminato, il successivo non può iniziare l'esecuzione;
- non è presente interazione tra utente e job;
- se un job si sospende in attesa di un evento, la CPU rimane inattiva;
- ha una scarsa efficienza: durante l'I/O del job corrente, che sono per loro natura molto lente, la CPU rimane inattiva.

Il problema principale della scarsa efficienza di questo tipo di gestione era proprio legato al fatto che la CPU veniva sempre altamente sottoutilizzata dato che sia il lettore di schede sia la stampante, essendo componenti meccanici, erano di gran lunga più lenti del processore, che risultava per la maggior parte del tempo inattivo aspettando le "lentezze" delle sue periferiche.

Furono quindi introdotti perfezionamenti nella gestione a lotti per rimuovere tale collo di bottiglia:
- si passò dalle schede ai nastri magnetici, aumentando così la velocità di trasferimento;
- negli anni '50 apparvero i primi calcolatori con canali di I/O autonomi e con le prime memorie a disco di grande capacità.

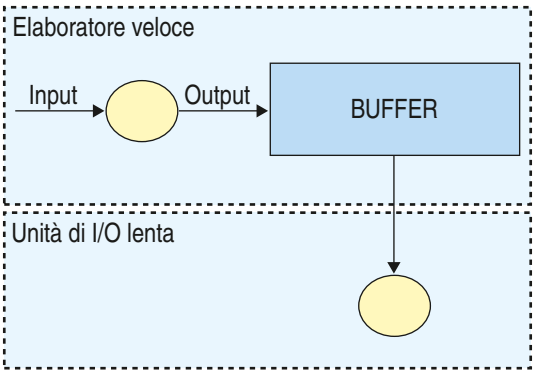
I calcolatori da strutture "monolitiche" iniziano a "scomporsi" in più componenti, come si può vedere dalla fotografia, che riprende un sistema IBM 1401 del 1959.

Buffering

Un primo rimedio per ovviare ai tempi di attesa delle periferiche fu l'introduzione della tecnica di buffering: un buffer è un'area di memoria intermedia dedicata al salvataggio temporaneo di informazioni, implementato direttamente nel device driver del dispositivo.

Con il buffering l'accesso alla periferica segue questa sequenza di operazioni:

- la periferica legge in Input/Output un blocco dati (blocco corrente) e lo passa alla CPU;
- mentre la CPU elabora il blocco dati corrente, la periferica legge il blocco dati successivo;
- la CPU produce output solo fino a quando il buffer è pieno e quindi si ferma;
- la periferica di uscita legge i dati prodotti dalla CPU prelevandoli dal buffer al suo ritmo.



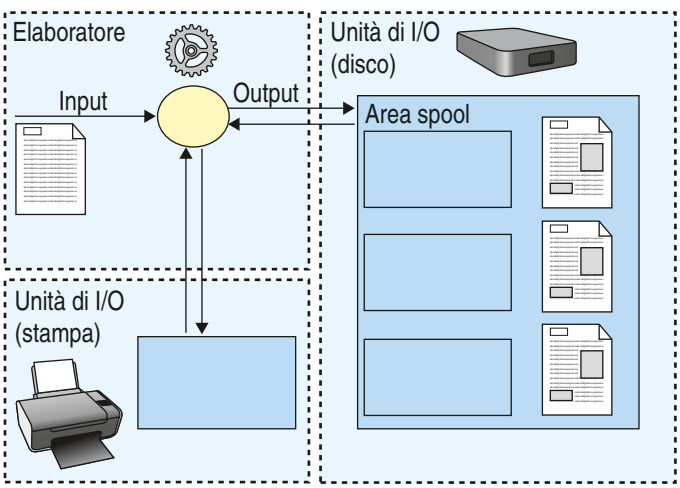
Spooling

Nonostante l'introduzione dei buffer, rimasero alcuni problemi da risolvere:

- i buffer non possono essere molto grandi per motivi di costi;
- il buffer tende a riempirsi velocemente e quindi il problema dell'attesa non viene risolto;
- con velocità di I/O << velocità CPU non produce giovamenti;
- se il buffer si riempie, la CPU si blocca.

Si arrivò all'introduzione di meccanismi chiamati dischi di spooling (Simultaneous Peripheral Operations On Line), creando contemporaneità tra le attività di I/O e quelle di computazione: il disco viene impiegato come un buffer molto ampio, dove leggere in anticipo i dati e memorizzare temporaneamente i risultati, in attesa che il dispositivo di output sia pronto.
Invece cioè di scrivere e leggere dai dispositivi fisici si legge e si scrive su due dispositivi virtuali:

- il disco di spooling di ingresso contiene le schede lette dal lettore e contenenti job non ancora eseguiti ("job virtuali");
- il disco di spooling di uscita contiene invece "output virtuali", ossia tabulati già prodotti relativi a job eseguiti ma non ancora stampati.



In questo modo è possibile sovrapporre i tempi di elaborazione con quelli di I/O: mentre la CPU elabora il programma, i canali di I/O provvedono a riempire o svuotare i dischi di spooling e a gestire i trasferimenti tra la memoria e i dischi stessi.

Sistemi interattivi (1965-1980)

◀ Il DMA (*Direct Memory Access*, "accesso diretto alla memoria") è un meccanismo che permette ad alcuni dispositivi, come unità a disco, schede grafiche, schede audio e musicali, di accedere direttamente alla memoria del calcolatore per trasferire i dati senza far "sprecare" tempo di elaborazione alle CPU: a trasferimento ultimato richiamano l'attenzione della CPU generando un interrupt, cioè mandando un segnale asincrono che indica alla CPU che la periferica ha terminato il proprio lavoro e il risultato è disponibile per l'utilizzo. ▶

Negli anni '60 IBM introduce i canali di I/O per ottenere il parallelismo effettivo delle operazioni: questi dispositivi, chiamati anche processori di I/O, hanno il compito di gestire totalmente le periferiche, così da lasciare alla CPU l'esecuzione dei programmi (oggi questo meccanismo si chiama ◀ DMA ▶ (Direct Memory Access).

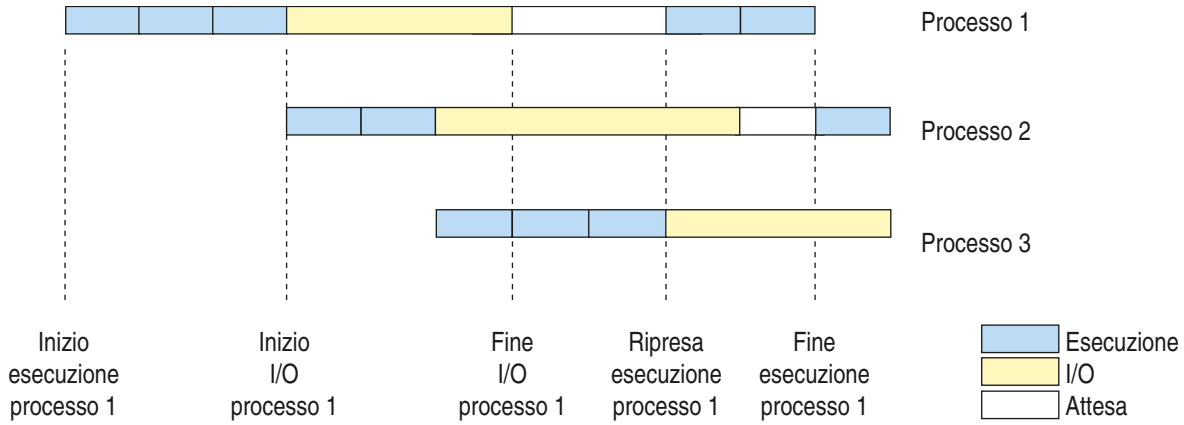
Multiprogrammazione

Con questo sistema si è data una prima risposta al problema costituito dai tempi morti di attesa: il secondo passo si è fatto introducendo la multiprogrammazione.
L'osservazione che sta alla base di questa tecnica è il fatto che spesso, durante l'elaborazione, un job è costretto a fermarsi in attesa di un evento esterno e quindi la CPU rimane inoperosa fino a quando questa situazione viene risolta (pensiamo per esempio a un errore nel codice in fase di test oppure a un programma che necessita che l'utente inserisca un dato dall'esterno per poter proseguire).
L'attesa può durare anche diversi minuti.
L'idea che ha fatto nascere la multiprogrammazione è quella di eliminare questi tempi tenendo pronti altri job in memoria che, nel caso uno si debba sospendere, prendano il suo posto nell'esecuzione.
In questo caso il "tempo perso" risulta minimo, dovuto solo al ◀ cambio di contesto ▶.

◀ Con cambio di contesto (context switch) si intendono le operazione tramite le quali il sistema operativo passa dall'esecuzione di un programma (processo) all'esecuzione di un altro processo, salvando lo stato del primo e predisponendo il sistema per poter mandare in esecuzione il secondo. Con processo si intende un'istanza del programma in evoluzione, cioè che è eseguito dal processore. ▶

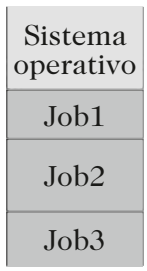
MULTIPROGRAMMAZIONE

Nella multiprogrammazione sono caricati in memoria centrale contemporaneamente più job: mentre un job è in attesa di un evento e sospende la sua esecuzione, il sistema operativo assegna la CPU a un altro tra quelli pronti all'esecuzione.



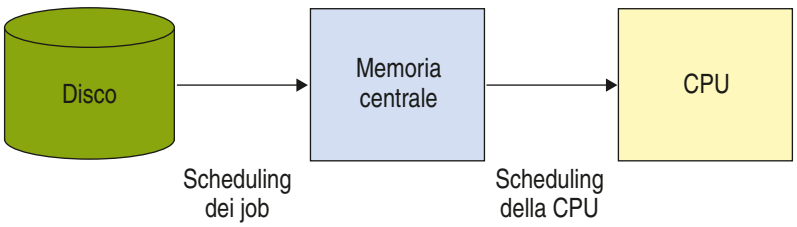
Avere in memoria più programmi, come vedremo, significa anche gestire lo spazio della RAM suddividendola in parti e caricando i nuovi programmi che devono essere eseguiti ma "scaricando" quelli che terminano l'esecuzione.

Il ruolo del sistema operativo diventa sempre più importante e complesso e le attività che deve svolgere divengono sempre più numerose: il sistema operativo deve *portare avanti* contemporaneamente l'esecuzione di più job.
Per poter evolvere simultaneamente, però, i programmi devono essere anche contemporaneamente residenti in memoria centrale: si parla ora di multiprogrammazione, nel senso che più job sono caricati in RAM e pronti a essere eseguiti.
La figura a lato riporta un possibile schema della memoria in un sistema multiprogrammato dove si può osservare la presenza contemporanea di tre job oltre al sistema operativo.



Dato che un solo job utilizza in un dato istante la CPU e più job attendono (in memoria centrale) di acquisire la CPU, quando il job che sta utilizzando la CPU si sospende in attesa di un evento, il SO *decide* a quale job assegnare la CPU ed effettua lo scambio (scheduling).
Il SO effettua delle scelte tra tutti i job:
- quali job caricare in memoria centrale: scheduling dei job o *long term scheduling*;
- a quale job assegnare la CPU: scheduling della CPU o *short term scheduling*.

Vedremo in seguito in base a quali criteri vengono scelti i job da caricare e da mandare in esecuzione.



Un ulteriore passo avanti viene fatto con l'esigenza di soddisfare nuove tipologie di programmi: i programmi interattivi, quelli cioè che richiedono un'alternanza tra elaborazione e inserimento dati da parte dell'utente (il caso limite sono i videogiochi). In questi programmi il job si sospende continuamente per le operazioni di I/O.

Grazie alla multiprogrammazione, inoltre, i sistemi sono diventati multiutente. All'unità centrale sono connessi molti terminali: più utenti interagiscono contemporaneamente con il sistema e il SO deve essere in grado di consentire a più persone l'utilizzo del calcolatore, come se ciascuna fosse l'unica proprietaria (virtualizzazione del calcolatore).

Time sharing

Nel 1963 il MIT (Massachussetts Institute of Technology) mise a punto un nuovo concetto di sistema operativo (il Compatible Time Sharing System, CTSS), che può essere conside-

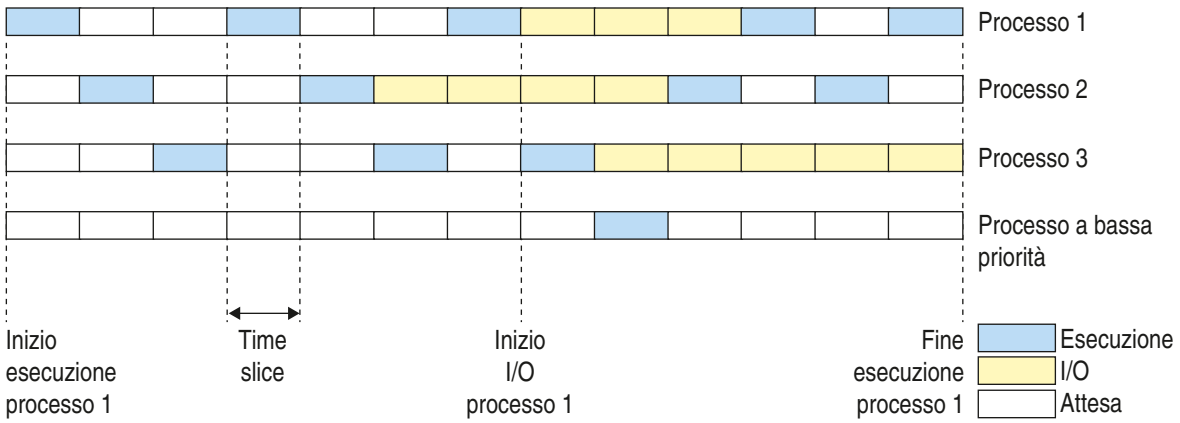
rato il primo dei sistemi operativi di tipo interattivo, operante su un sistema IBM7094 al quale erano connessi contemporaneamente più terminali.

Il termine time sharing (a partizione di tempo) sta a indicare il meccanismo di funzionamento: a ogni utente connesso con il suo terminale al sistema viene assegnata una porzione di tempo (quanto) di utilizzo CPU (time slice, tipicamente della durata di poche centinaia di millisecondi).

Se il job non si conclude nel tempo macchina a lui assegnato, viene temporaneamente sospeso e posto, assieme a tutti i job che sono nelle medesime condizioni, in una lista "di sospensione", e il processore viene assegnato a un altro programma che era stato, come lui, precedentemente sospeso e così via.

In questo modo vengono eliminati i tempi di inattività del sistema dovuti alle operazioni di I/O, di editing e messa a punto dei programmi propri dell'attività interattiva.

Nell'esempio di cui alla figura sottostante sono riportati 4 processi in memoria che si alternano nell'utilizzo della CPU secondo la suddivisione del tempo in "time slice": si può osservare che l'ultimo processo è "meno importante" dei primi, gli viene assegnata una priorità più bassa e accede con meno frequenza alla CPU rispetto agli altri.



Con il time sharing un job che sta utilizzando la CPU può terminare la sua elaborazione sostanzialmente per due motivi:

- ha terminato il tempo a lui assegnato, cioè il suo time slice;
- ha bisogno di qualche risorsa non disponibile (per esempio di un dato di input o di una periferica).

Questi meccanismi (politiche di gestione della CPU) saranno oggetto della prossima unità didattica.

Il concetto di time sharing fu reso famoso dal calcolatore PDP 10 costruito da Digital Equipment Corporation (DEC) e utilizzato in diverse università americane, tra le quali ricordiamo MIT AI Lab, Stanford AI Lab e Carnegie Mellon.

Alla fine degli anni '60 IBM presenta un progetto innovativo di computer, il System/370, con la memoria principale basata su circuiti integrati, l'aritmetica in floating point e il supporto per la memoria virtuale con il nuovo sistema operativo VM/370.

> L'utilizzo condiviso dell'elaboratore ha portato anche un nuovo problema: la sicurezza dei dati e dei programmi, questione ancora oggi centrale nell'informatica.

Digital risponde con la linea VAX, un'evoluzione del PDP, introducendo l'indirizzamento virtuale, il demand paging e il sistema operativo Unix, scritto per la parte a basso livello in linguaggio macchina e per la parte ad alto livello in C, linguaggio di programmazione progettato da Ken Thompson, Brian Kernighan e Dennis Ritchie.
Avere più utenti che condividono il calcolatore e le sue risorse ha creato l'esigenza di proteggere il lavoro di ogni utente in quanto un job potrebbe involontariamente (o deliberatamente) danneggiare il programma (o i dati) degli altri utenti: nasce quindi anche il problema della riservatezza, per garantire che ciascuno possa accedere solamente ai dati di propria competenza.

Home computing (anni '70)

Negli anni '70 il calcolatore non viene più visto solo come uno strumento di calcolo, ma anche di intrattenimento e divertimento: si punta al mercato casalingo e pioniere in questo settore fu Nolan Bushnell, fondatore del primo grande colosso dei videogiochi: l'Atari (1972).
Ad esso si affiancarono Commodore, Sinclair e altre case produttrici di home computer, macchine con architetture hardware poco costose (con un sistema operativo dedicato) programmabili generalmente in BASIC, che usavano come monitor il televisore domestico e avevano come periferiche di massa dapprima nastri magnetici (anche normali cassette audio) e in un secondo tempo unità con singolo floppy disk.



Si diffusero rapidamente sia per il costo modesto sia per la molteplicità di programmi disponibili: nasce il mercato dei videogiochi, un settore commerciale tuttora di grande importanza per l'informatica.

Sistemi dedicati (anni '80)

Una nuova tipologia di sistemi operativi dedicati si presenta negli anni '80 con l'avvento del personal computer: si ripercorre la storia dei sistemi operativi su macchine personali, cioè su piccoli sistemi di calcolo (piccoli di dimensioni, ma con potenza di calcolo superiore ai mainframe degli anni '70). Nel 1982 Microsoft rilascia MS-DOS, un sistema operativo per i microprocessori della famiglia x86 INTEL (PC IBM e compatibili). La tecnologia influenza notevolmente il sistema operativo: i monitor iniziano a essere grafici, cioè non più solo monocromatici a cristalli verdi (o ambra), e la loro risoluzione non è più a linee di caratteri (80 × 25) ma a matrice di punti (pixel).

Prende piede il software grafico interattivo, con nuovi strumenti di interazione con l'utente come il mouse, e quindi nuove tipologie di sistemi operativi: il primo esempio lo troviamo con il sistema operativo MacOS (acronimo di Macintosh Operating System) di Apple (1984), che è il capostipite di una nuova generazione di SO visuali dedicati (l'anno successivo, nel 1985, anche Microsoft rilascerà la prima versione del proprio sistema operativo Windows: la 1.0).

Questo tipo di sistemi operativi prende anche il nome di sistemi monoutente e il nome stesso indica che forniscono una macchina virtuale semplice, per un solo utente alla volta, che facilita l'applicazione di un vasto numero di pacchetti software, permettendo nel contempo all'utente di sviluppare e mettere in esecuzione programmi per proprio conto.

Nei sistemi operativi monoutente si dà maggior rilievo alla disponibilità di un linguaggio di comando facilmente utilizzabile, di un file system semplice e di risorse di I/O per dischi e dispositivi vari.

Sistemi odierni e sviluppi futuri

Nell'informatica è sempre difficile fare previsioni data la rapida evoluzione del settore: previsioni semplici sono quelle di un continuo e costante aumento dei pc personali, della connessione alla rete e dei computer indossabili (*wearable computer*) e i maggiori sforzi dei progettisti saranno rivolti alla progettazione di sistemi operativi per queste tre classi di macchine.

Possiamo individuare e illustrare alcune tipologie di sistemi operativi oggi disponibili per i diversi settori di applicazione dell'informatica:

- Sistemi Operativi per Server;
- Sistemi Operativi in Tempo Reale;
- Sistemi Operativi Embedded;
- Sistemi Operativi per Smart Card.

SO per server

Anche la più piccola organizzazione oggi ha i PC collegati in rete locale, quindi i SO per rete sono e saranno oggetto di sviluppo.
Hanno come caratteristica principale quella di servire molti utenti in contesti di rete, mettendo a disposizione accessi concorrenti e servizi, dall'utilizzo delle stampanti e dei dischi all'accesso a Internet e alla posta elettronica ecc.

SO real-time

Sono sistemi operativi altamente specializzati, sviluppati per quei dispositivi hardware sui quali le applicazioni hanno forti vincoli real-time, cioè interagiscono con l'ambiente esterno attraverso periferiche e garantiscono a dati di ingresso risposte in un tempo utile rispetto alle costanti di tempo proprie dell'ambiente esterno.
Riguardano generalmente le applicazioni di controllo di processo e dei sistemi di telecomunicazione e sono sostanzialmente di due tipi:

- hard real-time: hanno una deadline hard, cioè un limite temporale rigido che non deve mai essere superato perché le conseguenze di un eventuale fallimento potrebbero essere disastrose (per esempio controllo aereo);
- soft real-time: permettono che qualche volta si "sfori" la deadline, quindi i vincoli temporali non sono tassativi e le conseguenze di un eventuale fallimento non sono disastrose (per esempio rete telefonica).

Sistemi embedded

Non esiste una definizione universalmente riconosciuta per i sistemi embedded, e una tra quelle più utilizzate è quella riportata di seguito.

SISTEMA EMBEDDED

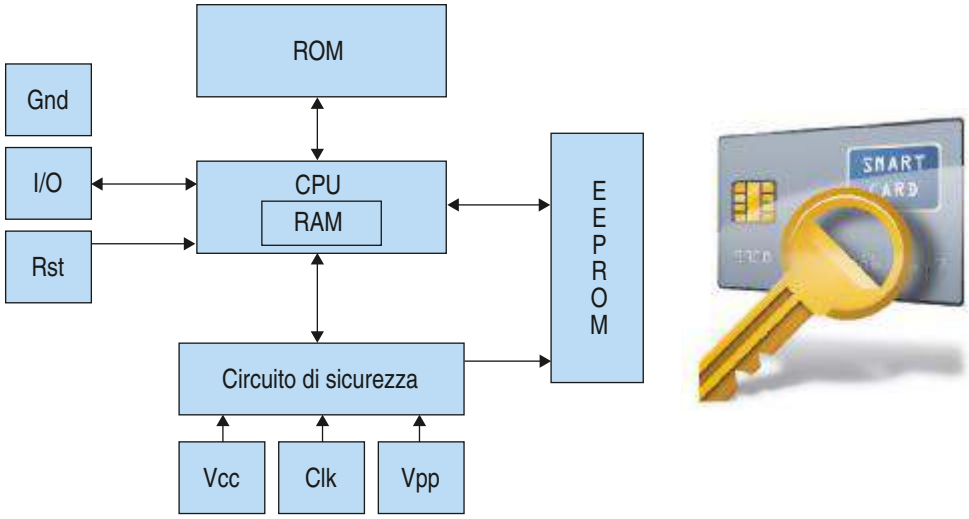
In generale con sistema embedded si intende un dispositivo incapsulato all'interno del sistema da controllare progettato per una determinata applicazione e supportato da una piattaforma hardware su misura.

Sono sistemi generalmente privi di interazione umana, capaci di resistere a eventi dannosi, a vibrazioni e shock e di ripartire in modo autonomo dopo un'eventuale interruzione; sono di dimensioni ridotte e realizzati con un hardware dedicato per svolgere un particolare compito, hanno poca memoria e generalmente devono avere un basso consumo.
Rientrano tra queste applicazioni i telefoni cellulari, gli elettrodomestici, i palmari ecc.

Sistemi su Smart Card

Possono essere visti come un caso limite dei sistemi embedded estremi: sono generalmente di plastica, della dimensione di una carta di credito e incorporano nel loro interno un microchip con notevoli capacità di elaborazione e di memorizzazione.
All'interno del chip possono essere integrati molteplici componenti, quali microprocessori, memorie, ROM, RAM, antenne ecc., come un vero personal computer.
Le Smart Card, oggi largamente diffuse, hanno bassi costi di realizzazione e vengono usate per applicazioni di ogni genere: per utilizzare i mezzi di trasporto pubblici (come le Oyster card, biglietto elettronico usato nell'area di Londra), come carte di credito e di debito, co-

me schede di sicurezza per l'accesso agli edifici e per applicazioni sanitarie o documenti di identità, fino alle raccolte punti dei supermercati.



Le carte di vicinanza, cioè quelle che utilizzano l'antenna per scambiare informazioni su distanze fino ai due metri, oggi non sono ancora di uso corrente, ma saranno utilizzate nel prossimo futuro per il ticketing e per persone con disabilità (per esempio per coloro che hanno limitate capacità d'uso di macchinari come le obliteratrici degli autobus).

I chip sono computer completi e sin dalle prime versioni di Smart Card i produttori stessi hanno sviluppato e proposto sistemi operativi dedicati: solo dal 1997 sono disponibili SO sviluppati da terze parti, cioè software house non produttrici anche dell'hardware (MULTOS e Windows per Smart Card).
Dai sistemi operativi scritti inizialmente in linguaggio assembler si è passati a una soluzione simile all'implementazione di una Java Virtual Machine sulla memoria di una Smart Card: oggi uno dei sistemi più utilizzati è il SmarTEC OS, che risiede sulla ROM della carta ma che permette modifiche al codice presente sulla memoria EEPROM da parte dello sviluppatore.

Verifichiamo le conoscenze

Esercizi a scelta multipla

1 Quale di queste affermazioni è vera?
- ○ Mark1 non aveva alcun linguaggio assemblativo
- ○ ENIAC non aveva alcun linguaggio assemblativo
- ○ UNIVAC non aveva alcun linguaggio assemblativo
- ○ Macintosh non aveva alcun linguaggio assemblativo

2 Con il termine monitor si intende:
- ○ un sinonimo dello schermo
- ○ un sistema operativo
- ○ una primitiva di sistema
- ○ una parte hardware

3 Uno step di un job in un sistema batch è per esempio:
- ○ il caricamento del programma
- ○ l'esecuzione del programma
- ○ la terminazione di un programma
- ○ la compilazione di un programma

4 Nei sistemi batch JCL significa:
- ○ *Java Control Language*
- ○ *Job Center Language*
- ○ *Job Center Line*
- ○ *Job Control Language*

5 Il time slice è dell'ordine di:
- ○ poche decine di millisecondi
- ○ poche centinaia di secondi
- ○ qualche secondo
- ○ pochi decimi di secondo

6 Windows è un sistema operativo:
- ○ multiprogrammato
- ○ multiutente
- ○ time sharing
- ○ bath

7 Quale di questi sistemi non è hard real-time?
- ○ controllo navale
- ○ forno a microonde
- ○ ascensore
- ○ air bag

8 Quale di queste affermazioni sui sistemi Smart Card è falsa?
- ○ è un sistema embedded
- ○ sono programmati in Java
- ○ sono usati come abbonamenti a mezzi pubblici (oyster card)
- ○ non hanno un SO dedicato

Test vero/falso

1	La prima fase compresa tra il 1945 e il 1955 è caratterizzata da SO dedicati.	V F
2	L' UNIVAC 1 era dotato di una lettore di schede perforate.	V F
3	La gestione a lotti è sinonimo di batch processing.	V F
4	Il buffering è una variante dello spooling.	V F
5	In un sistema multiprogrammato più processi sono in esecuzione contemporaneamente.	V F
6	Le console per videogiochi sono sprovviste di sistema operativo.	V F
7	Il sistema operativo MS-DOS è stato realizzato per PC IBM compatibili.	V F
8	I sistemi operativi per personal computer sono chiamati monoutente.	V F
9	Windows è il primo SO a finestre.	V F
10	Tra i sistemi embedded rientrano anche le carte a microchip.	V F

UNITÀ DIDATTICA 3

LA GESTIONE DEL PROCESSORE

IN QUESTA UNITÀ IMPAREREMO...

- che cosa si intende per processo
- il ciclo di vita di un processo
- le politiche di allocazione del processore
- le problematiche di cooperazione tra processi

Introduzione al multitasking

Tra i componenti del sistema operativo il primo a essere analizzato è il gestore del processore; nonostante l'evoluzione tecnologica abbia incrementato le capacità computazionali grazie all'aumento della velocità dei processori, ancora oggi deve essere ottimizzato l'utilizzo della CPU.

Tutti i moderni SO cercano di sfruttare al massimo le potenzialità di parallelismo fisico dell'hardware per minimizzare i tempi di risposta e aumentare il throughput del sistema, ossia il *numero di programmi eseguiti per unità di tempo*.

Finora abbiamo parlato indifferentemente di programma e di processo: è doveroso fare una precisazione prima di proseguire con la trattazione:

- il programma è costituito dall'insieme delle istruzioni, memorizzato su memoria di massa (è un'entità statica);
- il processo è un'istanza di un programma in evoluzione, cioè che è eseguito dal processore, quindi deve essere residente in memoria RAM (è un'entità dinamica);
- task è sinonimo di "processo" (letteralmente "compito").

Abbiamo visto che l'ottimizzazione del tempo di CPU avviene grazie alla multiprogrammazione, cioè alla contemporanea presenza di più programmi in memoria, e in questa unità didattica vedremo come è possibile migliorare la gestione:

- dello scheduling dei job, che consiste nell'insieme delle strategie e dei meccanismi utilizzati per la scelta dei programmi che dal disco devono essere caricati in RAM;
- dello scheduling della CPU, che consiste nell'insieme delle strategie e dei meccanismi che permettono di assegnare e sospendere l'utilizzo della CPU da parte dei vari programmi.

L'esecuzione di un programma, quindi un processo, è costituita da una successione di fasi di elaborazione sulla CPU e fasi di attesa per l'esecuzione di operazioni su altre risorse del sistema (operazioni di I/O, di caricamento dati, colloquio con periferiche ecc.) che di fatto lasciano inattiva la CPU.
Possiamo dare come definizione sintetica di processo quella riportata di seguito.

PROCESSO
Un processo è un'entità logica in evoluzione.

I sistemi operativi multitasking durante le fasi di attesa mandano in esecuzione sulla CPU altri programmi tra quelli caricati in memoria, "portando avanti" in parallelo più processi, riducendo al minimo l'inattività della CPU e migliorando così l'efficienza del sistema (numero di programmi eseguiti per unità di tempo).

Il parallelismo che si ottiene è virtuale (e non fisico) dato che il processore è uno solo. Possiamo fare quindi una distinzione di parallelismo in:
- multitasking: esecuzione di programmi indipendenti sulla CPU e sul processore di I/O;
- multiprocessing: multiprogrammazione estesa a elaboratori dotati di più CPU e processori di I/O.

Si noti come il multitasking non implichi la multiutenza, ossia l'uso simultaneo del sistema da parte di più utenti. Infatti anche nei sistemi monoutente (il personal computer) vengono mandati in esecuzione più processi simultaneamente, per esempio mentre editiamo un testo, sentiamo la musica, scarichiamo la posta... e un "antivirus ci protegge".

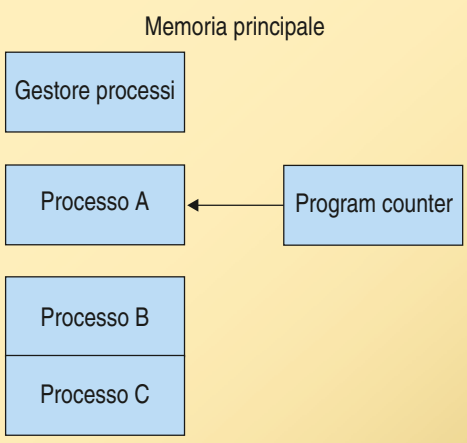
I processi

Come abbiamo visto, il programma è un'entità passiva (un insieme di byte contenente le istruzioni che dovranno essere eseguite) mentre il processo è un'entità attiva, che evolve man mano che le istruzioni vengono eseguite dalla CPU.

Noi prenderemo il singolo processo come unità di riferimento nei moderni sistemi operativi, che possiamo dire essere costituiti da due parti:
- il codice (composto dalle istruzioni);
- i dati del programma, a loro volta suddivisi in:
  - *variabili globali*, allocate in memoria centrale nell'area dati globali;
  - *variabili locali e non locali* delle procedure del programma, memorizzate in uno stack;
  - *variabili temporanee introdotte dal compilatore* (tra cui ricordiamo il *program counter*) caricate nei registri del processore;
  - *variabili allocate dinamicamente* durante l'esecuzione, memorizzate in uno *heap*.

L'insieme di tutti i dati di un processo prende anche il nome di contesto del processo che, naturalmente, varia istante per istante, a partire dal valore contenuto nel program counter, un registro che specifica l'istruzione successiva che la CPU deve eseguire.
È anche possibile avere in esecuzione contemporaneamente più istanze di un programma, quindi più processi originati dallo stesso codice: per esempio si possono avere aperte due finestre con lo stesso programma in esecuzione.
Inoltre i processi possono essere indipendenti oppure cooperare per raggiungere un medesimo obiettivo:

- nel primo caso, cioè di processi indipendenti, un processo evolve in modo autonomo senza bisogno di comunicare con gli altri processi per scambiare dati;
- nel secondo caso, due (o più) processi hanno la necessità di cooperare in quanto, per poter evolvere, necessitano di scambiarsi informazioni. Si pensi per esempio a un semplice videogame con due giocatori dove ogni giocatore è un processo: in questo caso nasce la necessità per i due processi di coordinarsi per poter comunicare e scambiarsi le informazioni (i processi si devono sincronizzare).



Il program counter (PC) è unico nel processore e contiene l'indirizzo di un'istruzione del processo che in quel momento è in esecuzione.

Oltre alla cooperazione esiste un'altra forma di interazione tra processi: due (o più) processi possono ostacolarsi a vicenda compromettendo il buon fine delle loro elaborazioni. È il caso in cui entrambi i processi competono per utilizzare la medesima risorsa, che magari è in quantità limitata nel sistema.
Questo tipo di interazione può portare a situazioni indesiderate per uno o per entrambi i processi (blocco individuale o critico).

Riassumendo, abbiamo quindi tre modelli di computazione per i processi:

- modello di computazione indipendente;
- modello di computazione con cooperazione;
- modello di computazione con competizione.

Prima di affrontare i problemi connessi all'interazione dei processi è necessario comprendere il ciclo di vita del processo, cioè analizzare che cosa succede da quando un programma diviene processo fino a quando ha terminato la propria esecuzione e… finisce di essere processo.

Il SO deve mantenere per ogni processo alcune informazioni che contengono la "fotografia" istante per istante del processo, cioè di cosa sta "facendo il processo": l'insieme di queste informazioni, con l'immagine del processo e i dati "anagrafici" dello stesso, prende il nome di descrittore del processo ed è memorizzato in un'apposita struttura (un record chiamato anche con la sigla PD, Process Descriptor, oppure PCB, *Process Control Block*). Vedremo in seguito dettagliatamente quali informazioni contiene.

Stato dei processi

Durante il ciclo di vita di un processo è possibile individuare un insieme di situazioni in cui il processo può trovarsi, che definiremo come gli stati di un processo associati alla sua evoluzione e alla sua "situazione" rispetto alla CPU.

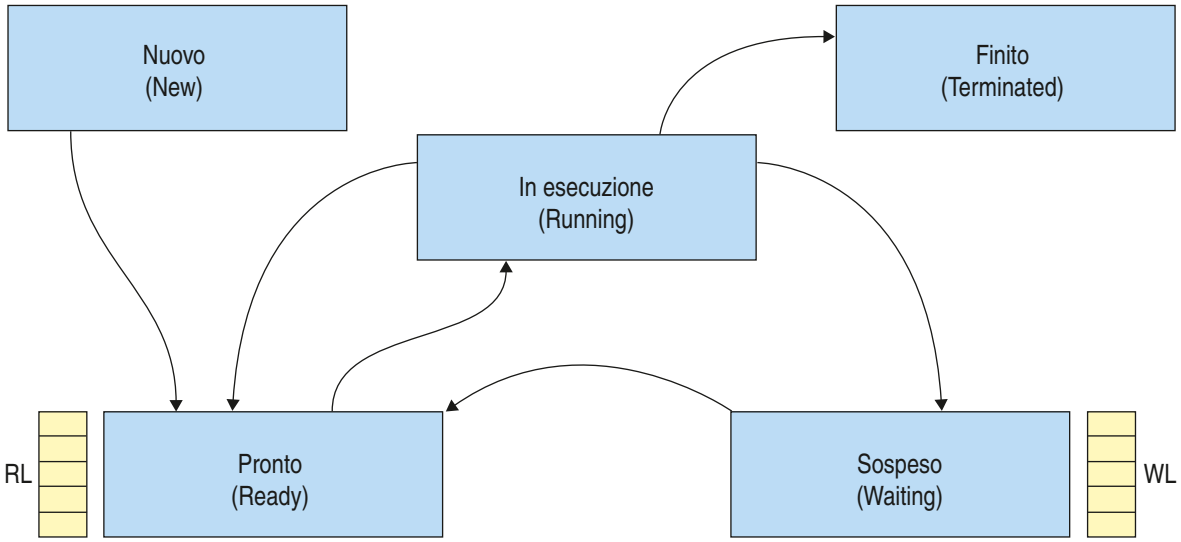
Vediamo dettagliatamente come può trovarsi un processo rispetto al processore:

- nuovo (*new*): è lo stato in cui si trova un processo appena è stato creato, cioè l'utente richiede l'esecuzione di un programma che risiede sul disco;
- esecuzione (*running*): il processo sta evolvendo, nel senso che la CPU sta eseguendo le sue istruzioni, e quindi ad esso è assegnato il processore. Nei sistemi a monoprocessore (quelli analizzati in questo testo) un solo processo può essere in questo stato;
- attesa (*waiting*): un processo è nello stato di attesa quando gli manca una risorsa per poter evolvere e quindi sta *aspettando* che si verifichi un evento (per esempio che si liberi la risorsa che gli serve e che il processore gliela assegni);
- pronto (*ready-to-run*): un processo è nello stato di pronto se ha tutte le risorse necessarie alla sua evoluzione tranne la CPU (cioè è il caso in cui sta aspettando che gli venga assegnato il suo time-slice di CPU);
- finito (*terminated*): siamo nella situazione in cui tutto il codice del processo è stato eseguito e quindi ha terminato l'esecuzione; il sistema operativo deve ora rilasciare le risorse che utilizzava.

STATO DI UN PROCESSO

Con stato di un processo intendiamo quindi una tra le cinque possibili situazioni in cui un processo in esecuzione può trovarsi: può assumere una sola volta lo stato di nuovo e di terminato, mentre può essere per più volte negli altri tre stati.

Vediamo come è possibile rappresentare mediante un grafico orientato i passaggi che un processo esegue tra i possibili stati sopra descritti: questo grafico prende il nome di diagramma degli stati.



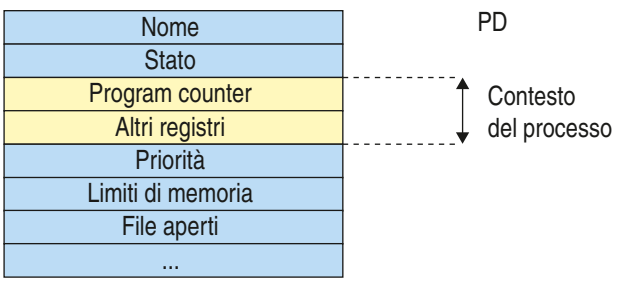
Seguiamo ora la vita di un processo dal principio: al nuovo processo viene assegnato un identificatore (PID, Process IDentifier) e viene inserito nell'elenco dei processi pronti (RL, Ready List) in attesa che arrivi il suo turno di utilizzo della CPU; quando gli viene assegnata la CPU, il processo passa nello stato di esecuzione, dal quale può uscire per tre motivi:

- termina la sua esecuzione, cioè il processo esaurisce il suo codice e quindi finisce (*exit*);
- termina il suo tempo di CPU, cioè il suo quanto di tempo, e quindi ritorna nella lista dei processi pronti RL (*ready list*);
- per poter evolvere necessita di una risorsa che al momento non è disponibile: il processo si sospende (*suspend*) e passa nello stato di attesa, insieme ad altri processi, formando la waiting list WL.

Dallo stato di sospeso, cioè dallo stato di attesa, un processo non può passare in quello di esecuzione: infatti, quando si rende disponibile la risorsa che sta "aspettando", viene spostato dalla WL ma viene inserito nelle RL, cioè nella lista dei processi pronti ad accedere alla CPU. Quando arriverà il suo turno, gli verrà assegnato il processore e solo allora potrà evolvere.

Possiamo ora entrare nel dettaglio del descrittore del processo (PCB) analizzando i principali dati che contiene:

- identificatore unico (PID);
- stato corrente;
- program counter;
- registri;
- priorità;
- puntatori alla memoria del processo;
- puntatori alle risorse allocate al processo.



Sappiamo che il program counter e i registri formano il contesto del processo: questi campi prendono anche il nome di area di salvataggio dello stato della CPU.

Oltre a queste informazioni sono anche presenti dati che riguardano *informazioni per l'accounting e per lo stato dell'I/O*, che riportano la lista dei file e delle periferiche associati al processo.
Il descrittore di processo viene allocato dinamicamente all'atto della creazione e opportunamente inizializzato. Viene rimosso dopo le operazioni di terminazione del processo.

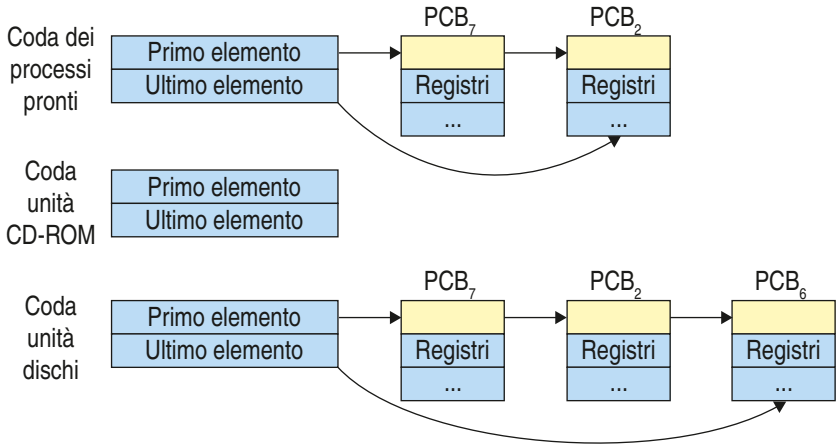
La schedulazione dei processi

Tra le operazioni che il sistema operativo deve effettuare rientra l'assegnazione della CPU ai processi che sono nella ready list, cioè nella lista dei processi pronti.
I meccanismi con i quali i processi vengono scelti prendono il nome di politiche di gestione o di schedulazione (scheduling) e il componente del SO che si occupa di questa gestione si chiama job scheduler.

In un SO con partizione del tempo un solo processo è in esecuzione e tutti gli altri sono posizionati in code, delle quali ricordiamo le due principali:

- la coda dei processi pronti (RL), in cui risiedono i processi caricati in memoria centrale che si trovano nello stato ready;
- la coda di attesa di un evento (WL), dove vengono inseriti i processi in attesa di una particolare risorsa.

La figura riporta la coda dei processi pronti e due code su due diversi dispositivi, un CD e un'unità a dischi. A seconda della modalità di gestione, lo scheduler sospende il processo che è in esecuzione mettendolo nella coda dei processi pronti e "risvegliando" un processo da tale coda per assegnargli la CPU: questo evento produce quello che si chiama cambio di contesto (context-switch).



CONTESTO DI UN PROCESSO

Il contesto di un processo è composto da alcune informazioni contenute nel suo specifico PCB, come il valore dei registri, il suo stato, il program counter, lo stack pointer ecc.

Il processo che viene sospeso dovrà successivamente essere ripristinato senza che rimanga traccia di quanto è successo: al processo deve "sembrare" di aver sempre posseduto la CPU, quindi il SO deve fare una "fotografia" di tutto quello che utilizza il processo, salvarlo e poi ripristinarlo.

Particolare attenzione deve essere posta quindi allo stato del processo, in base alle seguenti osservazioni:

- naturalmente non è necessario salvare il codice del programma;
- lo stack e la memoria heap non devono essere salvati, in quanto sarà lo stesso sistema operativo a preoccuparsi di NON modificarne i contenuti;
- sicuramente il contenuto dei registri verrà modificato dall'esecuzione del nuovo processo e quindi tutti questi devono essere salvati;
- lo stack pointer deve essere salvato insieme ai registri e al program counter.

Quando la CPU riattiva un processo che è stato sospeso, per prima cosa analizza il suo PCB per individuare il suo stack pointer e quindi da lì recupererà i valori dei registri e del program counter da ripristinare per poter riprendere l'esecuzione proprio dall'istruzione che era stata sospesa.

La parte del SO che realizza il cambio di contesto si chiama dispatcher.

Le operazioni eseguite per il cambio di contesto hanno generalmente la durata di 1 msec.

Zoom su...

PRE-EMPTIVE

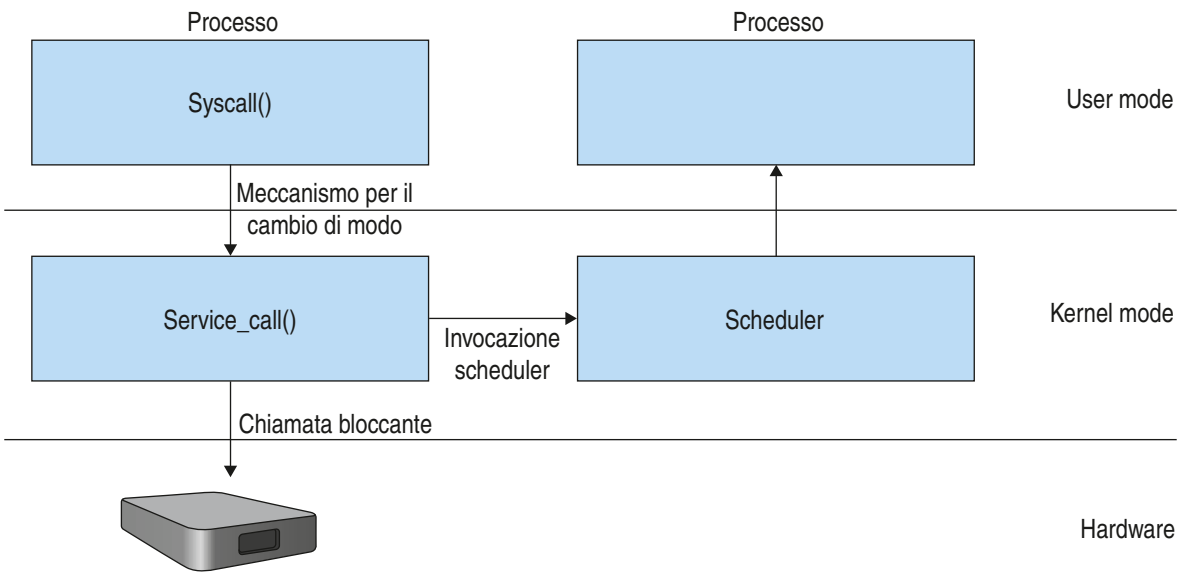
Non tutti i processi possono essere sospesi dal SO in ogni istante della loro esecuzione: per loro natura alcuni processi devono terminare la loro esecuzione (oppure un insieme di istruzioni che devono essere eseguite senza interruzione, come un'operazione di I/O) e solo quando sono in particolari situazioni possono essere interrotti.
Questi processi vengono chiamati non pre-emptive, a differenza di quelli che possono essere interrotti che sono i processi pre-emptive.
I sistemi a divisione di tempo (*time sharing*) hanno uno scheduling pre-emptive.

User mode e kernel mode

Durante la loro attività i processi, oltre che passare da periodi di esecuzione a periodi di attesa, si alternano anche in diverse modalità di esecuzione, caratterizzate da diversi livelli di privilegio: il livello user mode e il livello kernel mode.

Il livello user mode è quello di "normale stato di esecuzione" dei programmi applicativi dell'utente, mentre il livello kernel mode, detto anche supervisore, è quello nel quale sono in esecuzione i servizi del kernel.

Quando il processore è in kernel mode, cioè sta eseguendo un programma in modalità kernel, vengono opportunamente settati alcuni bit di stato del processore e in questo stato, generalmente, la sua esecuzione non può essere interrotta.



I criteri di scheduling

Senza entrare nel dettaglio vediamo brevemente i diversi criteri di scheduling: definiamo dapprima gli obiettivi che dobbiamo perseguire e successivamente confrontiamo tra loro le diverse politiche.

Gli obiettivi primari sono:
- massimizzare la percentuale di utilizzo della CPU (l'ideale sarebbe raggiungere una percentuale di utilizzo del 100%);
- massimizzare il throughput del sistema, cioè il numero di processi completati nell'unità di tempo.

Contemporaneamente è necessario ridurre al minimo i tempi di risposta del sistema quando un nuovo programma viene mandato in esecuzione, minimizzare i tempi di attesa tra un'esecuzione e l'altra e il tempo totale di permanenza di ciascun processo nel sistema.

Di seguito procediamo quindi alla definizione dei termini citati.

THROUGHPUT
Il numero medio di job, programmi, processi o richieste completati dal sistema nell'unità di tempo.

TEMPO DI COMPLETAMENTO (TURNAROUND TIME)
Il tempo dalla sottomissione di un job, programma o processo da parte di un utente nel momento in cui i risultati sono resi effettivamente disponibili all'utente stesso.

TEMPO DI RISPOSTA (RESPONSE TIME)
Il tempo dalla sottomissione di una richiesta da parte dell'utente nel momento in cui il processo risponde, chiamato anche **tempo di latenza**.

TEMPO DI ATTESA (WAIT TIME)
Si ottiene dalla somma degli intervalli temporali passati in attesa della risorsa.

Naturalmente non è possibile soddisfare tutti i criteri contemporaneamente: si dovrà trovare un compromesso, a seconda della tipologia di sistema:
- nei sistemi batch si cerca di massimizzare throughput e minimizzare turnaround;
- nei sistemi interattivi si cerca di minimizzare il tempo medio di risposta dei processi e il tempo di attesa.

Analizziamo di seguito i più comuni algoritmi di scheduling.

Algoritmo di scheduling FCFS

Il primo algoritmo che analizziamo è l'FCFS, acronimo di First-Come-First-Served, cioè "il primo arrivato è il primo a essere servito".
In questo caso i processi vengono messi in coda secondo l'ordine d'arrivo, quindi FIFO (First In First Out), e i processi pronti vengono schedulati secondo il loro ordine d'arrivo, indipendentemente dal tipo e dalla durata prevista per la loro esecuzione; inoltre questo algoritmo è di tipo non pre-emptive, quindi i processi non possono essere sospesi e completano sempre la loro esecuzione.

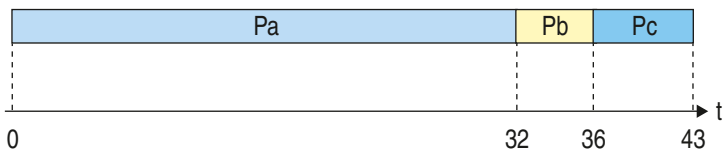
I principali difetti di questo algoritmo consistono nel fatto che se un processo ha un lungo periodo di elaborazione senza interruzione (CPU burst), non potendolo sospendere, tutti gli altri devono aspettare la sua naturale terminazione, e questo potrebbe produrre elevati tempi di attesa soprattutto in presenza di una sequenza di processi con un elevato grado di operazioni di I/O che quindi provocano un basso utilizzo della CPU (effetto convoglio).

ESEMPIO

Vediamo un esempio di come calcolare il tempo di attesa medio nel caso di tre processi e come tale risultato sia casuale, cioè dipendente solo dall'ordine di arrivo dei processi stessi, e quindi non significativo come parametro di qualità.

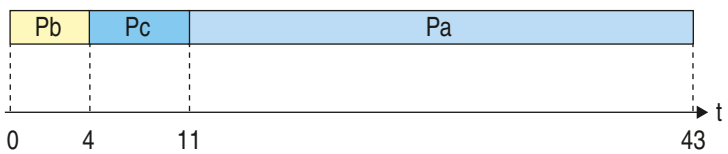
Supponiamo di avere tre processi che rispettivamente richiedono:
Pa = 32 unità di tempo/CPU
Pb = 4 unità di tempo/CPU
Pc = 7 unità di tempo/CPU

Nel primo caso i tre processi arrivano nell'ordine Pa, Pb e Pc. Calcoliamo il tempo medio di attesa in questa situazione:



$$ta_{m1} = \frac{ta_a + ta_b + ta_c}{3} = \frac{0 + 32 + 36}{3} = 22{,}7$$

Nel secondo caso l'ordine è diverso: Pb, Pc e Pa. Calcoliamo ora il nuovo tempo medio di attesa:



$$ta_{m2} = \frac{ta_a + ta_b + ta_c}{3} = \frac{0 + 4 + 11}{3} = 5$$

Evidentemente il parametro tempo di attesa non risulta essere significativo in quanto il suo valore è assolutamente casuale.

Algoritmo di scheduling SJF

Per migliorare l'algoritmo FCFS in modo da ottenere sempre il minor tempo di attesa, basta scegliere tra la lista dei processi pronti quello che occuperà per meno tempo la CPU e che quindi verrà mandato in esecuzione per primo (Shortest Job First).

È però necessario che il sistema operativo "in qualche modo" sia in grado di effettuare una stima dei tempi di utilizzo della CPU di tutti i processi che sono pronti nella RL, e questa operazione, oltre a essere onerosa, non sempre dà risultati attendibili.

Inoltre potrebbe verificarsi la situazione in cui mentre un processo è in esecuzione se ne aggiunge uno in coda con un tempo stimato minore; in questo caso possiamo avere due possibilità:

- nel caso di situazione non pre-emptive non si fa nulla;
- nel caso di situazione pre-emptive è necessario stimare per quanto tempo ancora il processo deve rimanere in esecuzione e confrontarlo con il tempo previsto per il nuovo processo: se questo è minore, si deve effettuare la sospensione e il cambio del contesto assegnando la CPU al nuovo processo.

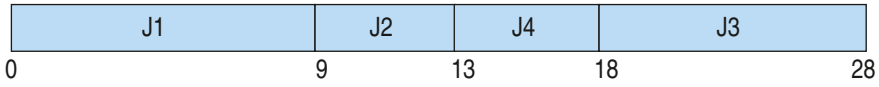
In questa seconda situazione l'algoritmo cambia anche nome e diviene SRTF, Shortest Remaining Time First, cioè si misura non il tempo del job intero, ma della parte rimanente che deve essere ancora eseguita.

ESEMPIO

Calcoliamo il tempo medio per la seguente situazione, dapprima utilizzando l'algoritmo SJF e quindi migliorandolo con il SRTF. ▶

Job	Durata	Arrivo
J1	9	0
J2	4	1
J3	10	2
J4	5	3

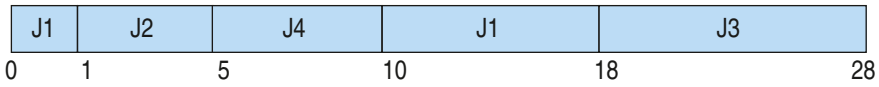
La sequenza di attivazione con l'algoritmo SJF è la seguente:



Il tempo di attesa è così calcolato:

$$Tw = (J1, J2, J4, J3) = [(0 + (9 - 1) + (13 - 3) + (18 - 2)] / 4 = 8,5$$

La sequenza di attivazione con l'algoritmo SRTF è quella che segue, dato che all'arrivo del secondo Job J2 il Job J1 in esecuzione viene sospeso in quanto ha un tempo restante di elaborazione maggiore:



Il tempo di attesa medio è:

$$Tw = (J1, J2, J4, J1, J3) = [(0 + (1 - 1) + (5 - 3) + (10) + (18 - 2)] / 4 = 7$$

Scheduling con priorità

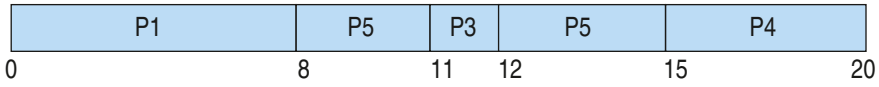
Nella procedura di scheduling con priorità a ogni processo viene associato un numero intero che corrisponde a un livello di priorità con il quale deve essere poi mandato in esecuzione: lo scheduler seleziona tra tutti i processi in coda quello a priorità più alta che avrà la precedenza di esecuzione su tutti.
Alla sua terminazione verrà scelto il processo che ha la massima priorità tra quelli rimasti e via di seguito; nel caso di due processi con medesima priorità si serve per primo il primo arrivato in coda, come nella FCFS.
La priorità viene assegnata ai processi dallo stesso sistema operativo, in base a criteri legati al tipo di processo e all'utente che lo ha mandato in esecuzione.

ESEMPIO

In questo esempio visualizziamo la sequenza dei processi in base alla seguente tabella di priorità:

Processo	Durata	Priorità
P1	8	1
P2	1	3
P3	5	4
P4	4	5
P5	3	2

La sequenza di attivazione è la seguente:



Il tempo di attesa medio è:

$$Tw = (P1, P5, P3, P5, P4) = (0 + 8 + 11 + 12 + 15) / 5 = 9,2$$

Gli algoritmi con priorità possono essere di tipo sia non pre-emptive sia pre-emptive: in questo caso, se si sta servendo un processo con priorità più bassa di uno nuovo appena giunto in coda, si cede la CPU a quello con priorità maggiore sospendendo il processo in esecuzione in quel momento.

Se continuano ad arrivare processi con alta priorità può avvenire il fenomeno della starvation dei processi, cioè i processi con priorità maggiore vengono sempre serviti a scapito di quelli con priorità bassa, che possono rimanere in coda anche per tempi indefiniti.

STARVATION

La starvation dei processi si verifica quando uno o più processi di priorità bassa rimangono nella coda dei processi pronti per un tempo indefinito in quanto sopraggiungono continuamente processi pronti di priorità più alta.

Una possibile soluzione è quella di introdurre le priorità variabili, cioè di modificare dinamicamente la priorità di un processo in base al tempo di attesa: se è da tanto tempo in coda, cioè è "invecchiato" (*aging*), gli viene alzato il livello di priorità mentre viene diminuito quello del processo in esecuzione man mano che aumenta il suo utilizzo di CPU.
La soluzione ottimale la si ottiene cambiando radicalmente politica, cioè adottando la Round Robin.

Algoritmo di Scheduling Round Robin

L'algoritmo classico utilizzato nei sistemi a partizione di tempo è il Round Robin (RR) dove tutti i processi pronti vengono inseriti in una coda circolare di tipo FIFO, cioè inseriti in ordine di arrivo, tutti senza priorità, e a ogni processo viene assegnato un intervallo di tempo di esecuzione prefissato denominato quanto di tempo (o time slice).
Se al termine di questo intervallo di tempo il processo non ha ancora terminato l'esecuzione, l'uso della CPU viene comunque affidato a un altro processo, prelevandolo sequenzialmente dalla coda e sospendendo il processo che era in esecuzione.

> Possiamo osservare che l'algoritmo RR può essere visto come un'estensione di FCFS con pre-emption periodica a ogni scadenza del quanto di tempo.

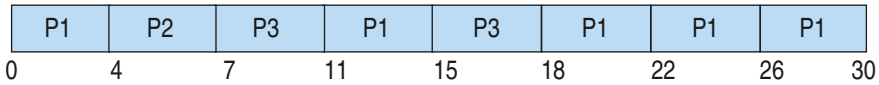
Con questo algoritmo tutti i processi sono trattati allo stesso modo, in una sorta di "correttezza" (fairness), e possiamo essere certi che non ci sono possibilità di starvation perché tutti a turno hanno diritto a utilizzare la CPU.

ESEMPIO

La sequenza di attivazione è la seguente, considerando il quanto di tempo = 4:

Processo	Durata
P1	20
P2	3
P3	7

Il tempo di attesa medio è:



$$Tw = (P1, P2, P3) = 11 / 3 = 3,6$$

È necessario però prestare molta attenzione al dimensionamento del time slice, in quanto le prestazioni del sistema sono direttamente legate alla sua durata:

- quando è piccolo abbiamo tempi di risposta ridotti ma è necessario effettuare frequentemente il cambio di contesto tra i processi, con notevole spreco di tempo e quindi di risorse (overhead);
- quando è grande i tempi di risposta possono essere elevati e l'algoritmo degenera in quello di FCFS.

Potrebbero poi sorgere problemi di decadimento delle prestazioni in quanto non sono presenti differenziazioni tra processi di sistema e processi utente, quindi anche i processi di sistema devono attendere il loro turno nella Round Robin.

La soluzione ottimale perciò non esiste: i moderni sistemi operativi combinano tra loro gli algoritmi qui presentati cercando in primo luogo di eliminare i problemi tipici di ogni algoritmo per ottenere una soluzione che possa essere mediamente buona per tutte le situazioni.

Come ultimo algoritmo trattiamo il MLFQ.

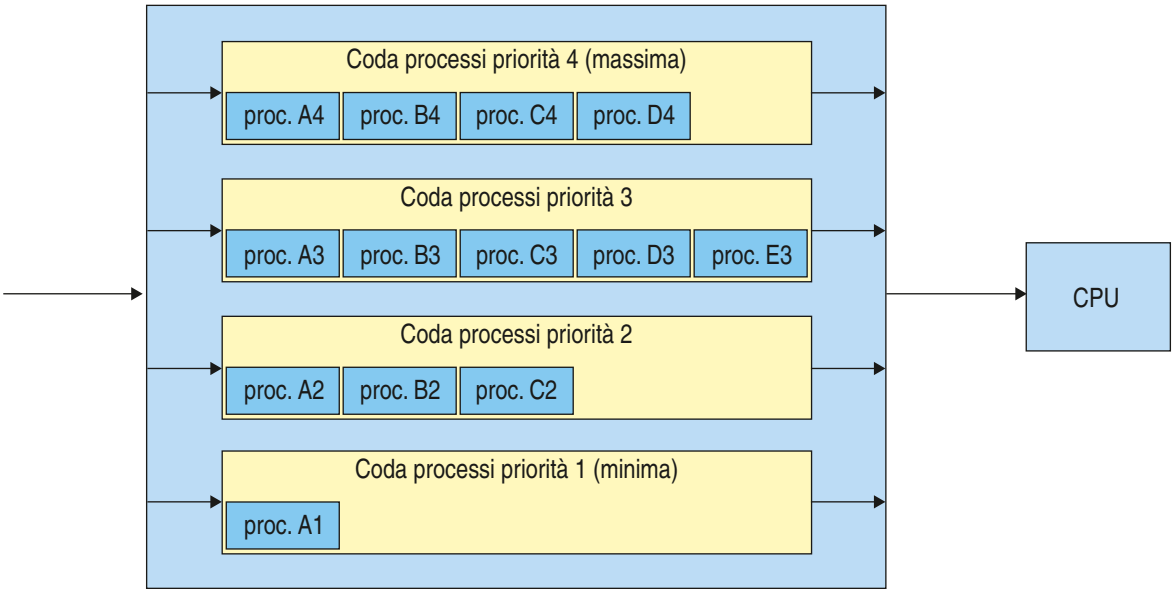
Algoritmo MLFQ

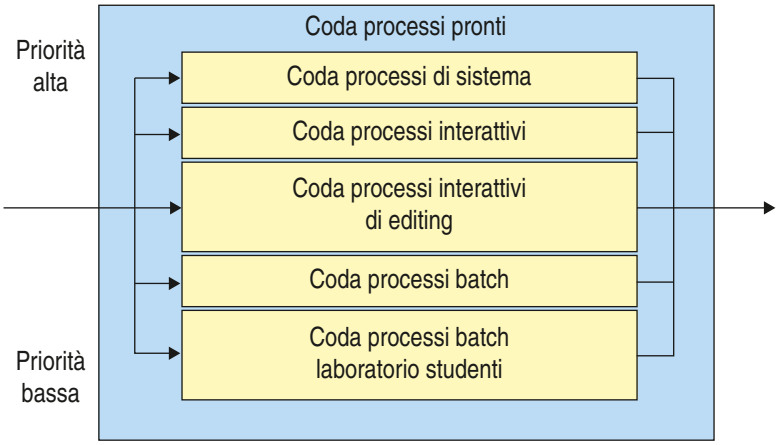
L'algoritmo MLFQ, acronimo di Multiple Level Feedback Queues, si ottiene dalla combinazione di un FCFS con un algoritmo a priorità.

Vengono determinate diverse code, una per ogni ordine di priorità, stabilite in base alla natura del job (di sistema, batch, interattivi, CPU-bound ecc.) e ogni coda viene gestita direttamente in FCFS oppure introducendo una politica di rotazione simile alla Round Robin.

Alle diverse code viene assegnato un numero di quanti di tempo diverso, e quando un processo ha esaurito i quanti a sua disposizione viene declassato e spostato in una coda con priorità più bassa: in questo modo si privilegiano i processi ad alta interattività rispetto ai processi lunghi.

Periodicamente vengono controllati i tempi di permanenza in coda e i processi più anziani vengono "promossi" aumentando la loro priorità.
Per esempio, una situazione di code a priorità in base alla natura dei processi potrebbe essere la seguente:





Scheduling a confronto tra sistemi operativi

Come esempio, riportiamo le caratteristiche di due sistemi operativi per personal computer.

Windows XP

- CPU Scheduling con prelazione.
- Trentadue livelli di priorità in Round Robin: si esegue per primo il task con priorità più alta.
- Due classi di priorità:
  - Real time: priorità nell'intervallo 16-32;
  - Variable: priorità nell'intervallo 1-15:
    - la priorità si abbassa se si esaurisce il quanto di tempo;
    - in seguito a uno sblocco, la priorità viene alzata di molto (boosted);
    - i processi in foreground sullo schermo hanno un livello più elevato di priorità.

Linux

- CPU Scheduling con prelazione.
- Centoquaranta livelli di priorità: si esegue per primo il task con priorità più alta.
- Tre classi di priorità:
  - Real time: priorità nell'intervallo 0-99 (FCFS non pre-empted oppure RR pre-empted);
  - System thread: priorità nell'intervallo 0-99 (FCFS non pre-empted);
  - User: priorità nell'intervallo 100-139 (RR pre-empted), con calcolo dinamico della priorità in base all'interattività del processo.

Cenni alle problematiche di sincronizzazione

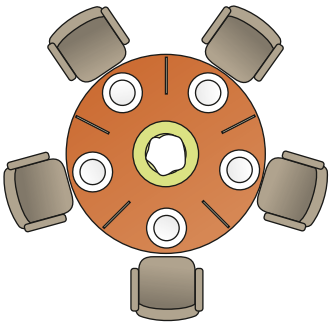
All'interno di un sistema operativo i processi possono essere classificati anche a seconda che evolvano in modo autonomo oppure che debbano scambiare dati con altri processi: nel primo caso parliamo di processi indipendenti mentre nel secondo caso i processi sono cooperanti.

I processi indipendenti potrebbero essere eseguiti anche su calcolatori diversi mentre quelli cooperanti devono condividere necessariamente delle risorse, anche solo per poter comunicare tra loro.

È necessario stabilire delle modalità di comunicazione in quanto, per esempio, il primo processo, dopo aver "prodotto" un dato, deve trasmetterlo al secondo che lo deve "consumare": questo è il tipico schema di coppia produttore-consumatore, che può però complicarsi quando i processi in questione sono più di due oppure ci sono più produttori o più consumatori o contemporaneamente sono tutti produttori e tutti consumatori.

Inoltre anche i processi indipendenti possono aver bisogno di sincronizzarsi tra loro in quanto potrebbero ostacolarsi a vicenda utilizzando contemporaneamente delle risorse provocando situazioni di blocco individuale o, addirittura, di starvation (dead-lock o abbraccio mortale, cioè blocco simultaneo di tutti i processi).

Per comprendere la necessità di sincronizzazione presente nei SO vediamo di seguito un famoso esempio proposto da Dijkstra nel 1965, conosciuto con il nome di "filosofi a cena".



Cinque filosofi stanno seduti intorno a un tavolo; ciascun filosofo ha di fronte un piatto e tra ogni piatto vi è una bacchetta per il riso (cinque filosofi, cinque piatti e cinque bacchette).
I filosofi alternano il loro tempo *pensando o mangiando*: quando un filosofo comincia ad avere fame, per poter mangiare deve avere due bacchette per poter prendere il riso, e quindi cerca di prendere possesso della bacchetta di sinistra e, successivamente, se va a buon fine, di quella di destra, una alla volta in ordine arbitrario.
Inizia a mangiare quando ha a disposizione entrambe le bacchette e quando si è saziato depone le bacchette e riprende a pensare.
Supponiamo che a un certo punto tutti i filosofi si trovino nella situazione di avere una bacchetta e di attendere la seconda, che non otterranno mai in quanto tutte le cinque bacchette sono in mano ad altrettanti filosofi: il loro destino è, purtroppo, la "morte per fame"!

In questo caso avviene quello che prende il nome di dead-lock, o abbraccio mortale: i processi (filosofi) si ostacolano a vicenda impedendo ciascuno l'avanzamento dell'altro trattenendo una risorsa (una bacchetta) e contemporaneamente danneggiando anche se stesso.

È quindi necessario che il sistema operativo gestisca anche questi casi, cioè la sincronizzazione dei processi tra loro indipendenti che però interagiscono in quanto utilizzano risorse in comune.

PROCESSI INTERAGENTI

Due o più processi si dicono invece interagenti se l'avanzamento di uno può condizionare l'avanzamento degli altri.

Abbiamo quindi due possibili esigenze di sincronizzazione dovute all'interazione tra processi:

- sincronizzazione per interferenza;
- sincronizzazione per cooperazione.

Per cooperare i processi possono comunicare tra loro: vediamo brevemente le due possibili forme senza entrare nei dettagli di come questi meccanismi vengono implementati nel sistema operativo.

Il primo metodo di comunicazione avviene utilizzando variabili comuni, condivise dai processi (share variables): è tipicamente impiegato nei SO aventi un'architettura dotata di una memoria comune a tutti i processi.

Il secondo metodo consiste nello scambio di messaggi lungo canali di comunicazione (message passing): questo metodo permette la comunicazione anche a processi residenti su macchine diverse, come è tipico delle reti di calcolatori.

Concludiamo osservando che nel primo metodo è possibile avere due situazioni diverse a seconda che i due processi debbano scambiarsi contemporaneamente delle informazioni e quindi si devono sincronizzare "aspettandosi" a vicenda, oppure se è sufficiente che un produttore produca e il consumatore consumi solamente (naturalmente il consumatore deve aspettare il produttore, quindi in un certo senso deve anch'esso sincronizzarsi, almeno temporalmente).

La sincronizzazione tra processi viene affidata ad appositi strumenti hardware e/o software, detti appunto meccanismi o primitive di sincronizzazione, il cui scopo è quello di garantire il rispetto di ordini temporali sul verificarsi di alcuni eventi: la loro trattazione viene fatta nei corsi specifici di progetto di sistemi operativi.

Verifichiamo le conoscenze

Esercizi a scelta multipla

1 Nei sistemi multitasking:
- ○ sono presenti più processori
- ○ più programmi evolvono contemporaneamente
- ○ più processi sono contemporaneamente in memoria
- ○ più processi sono in esecuzione contemporaneamente

2 Il PCB non contiene:
- ○ identificatore unico (PID)
- ○ stato corrente
- ○ stack
- ○ registri
- ○ priorità
- ○ puntatori alla memoria del processo

3 Dallo stato di pronto un processo può passare allo stato:
- ○ di esecuzione
- ○ di terminazione
- ○ di attesa
- ○ di inizio

4 Dallo stato di attesa un processo può passare allo stato:
- ○ di esecuzione
- ○ di terminazione
- ○ di pronto
- ○ di inizio

5 Nei sistemi batch si cerca di:
- ○ massimizzare il throughput
- ○ minimizzare il tempo medio di risposta
- ○ minimizzare il tempo di attesa
- ○ minimizzare il turnaround

6 Nei sistemi interattivi si cerca di:
- ○ massimizzare il throughput
- ○ minimizzare il tempo medio di risposta
- ○ minimizzare il tempo di attesa
- ○ minimizzare il turnaround

7 La SRTF è:
- ○ la FCFS con pre-emptive
- ○ la FCFS non pre-emptive
- ○ la SJF con pre-emptive
- ○ la SJF non pre-emptive

8 La starvation dei processi si verifica:
- ○ con politiche FCFS
- ○ con politiche SJF
- ○ con politiche SRTF
- ○ con politiche a priorità
- ○ con la Round Robin
- ○ sempre

Test vero/falso

1	I SO cercano di minimizzare il throughput del sistema.	V F
2	Lo scheduling dei job consiste nei meccanismi utilizzati per la scelta dei programmi disco.	V F
3	Un programma è l'evoluzione di un processo.	V F
4	Il multitasking si ottiene su architetture parallele di sistema.	V F
5	Con PCB si indica il *Program Control Block*.	V F
6	Il descrittore di un processo contiene anche il codice del programma.	V F
7	Dallo stato di pronto un processo può passare allo stato di attesa.	V F
8	Dallo stato di esecuzione un processo si sospende se gli manca una risorsa per evolvere.	V F
9	Le operazioni eseguite per il cambio di contesto hanno generalmente la durata di 1 ns.	V F
10	L'algoritmo SJF è un caso particolare dell'algoritmo FCFS.	V F
11	La starvation è dovuta alla mancanza di sincronizzazione tra processi.	V F

UNITÀ DIDATTICA 4

LA GESTIONE DELLA MEMORIA

IN QUESTA UNITÀ IMPAREREMO...

- la classificazione delle memorie
- i meccanismi di caricamento del programma in memoria
- le tecniche di virtualizzazione della memoria

Introduzione

La memoria centrale (RAM) è una risorsa importante per i calcolatori moderni che deve essere gestita al meglio, anche se rispetto al passato la sue dimensioni oggi sono molto aumentate, dell'ordine dei gigabyte.

MEMORIA CENTRALE

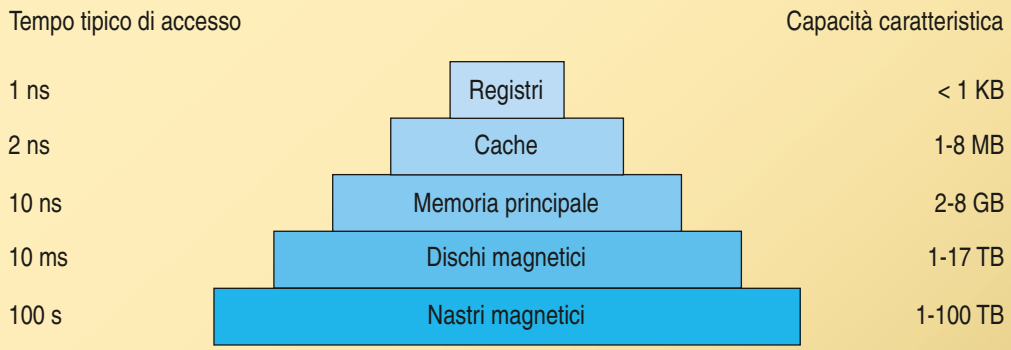
La memoria centrale è sempre stata una risorsa limitata:
– inizio anni '80: 128 kB;
– inizio anni '90: 1 MB;
– inizio anni 2000: 128 MB;
– attualmente: 8 GB.

Ricordiamo una celebre frase di Bill Gates, fondatore della Microsoft, pronunciata nel 1985, che si rivelò profondamente errata: "*Chi mai avrà bisogno di più di 640 K di memoria centrale?*".
In sostanza si può affermare che la memoria centrale è una risorsa che "non basta mai"!

Ogni utente desidera avere una "memoria infinita", veloce, poco costosa e possibilmente non volatile: naturalmente questi desideri sono contraddittori e quindi non possono essere esauditi tutti contemporaneamente.

Come vedremo il SO utilizza delle tecniche di gestione della memoria centrale, la RAM, cercando di "renderla infinita" (memoria virtuale) così da poter sempre esaudire ogni richiesta di spazio effettuata dell'utente.

Nel calcolatore sono presenti diversi tipi di memoria, classificati in base alle loro caratteristiche fondamentali, cioè velocità e capacità:
- nastro: molto capiente, magnetico, sequenziale (memoria di back-up);
- disco: capiente, lento, non volatile ed economico (memoria secondaria);
- memoria principale: volatile, mediamente grande, veloce e costosa;
- cache: volatile, veloce, piccola e costosa;
- registri: all'interno del processore, estremamente veloci e ridotti ad alcuni byte.



In questa unità didattica analizzeremo quella parte di SO che gestisce in particolare la memoria principale: il "memory manager" (gestore della memoria) che spesso utilizza anche la memoria disco per svolgere le sue funzioni.

La memoria centrale consiste in un ampio vettore di parole di memoria (o byte), ognuna delle quali ha un proprio indirizzo: la CPU preleva istruzioni e dati direttamente da essa per caricarli nei propri registri, in particolare carica l'istruzione presente nella posizione indicata dal *program counter*.
Ogni istruzione può a sua volta generare nuovi accessi alla memoria e quindi l'evoluzione di un programma è un susseguirsi di caricamenti (load) e archiviazioni (store) di istruzioni e di dati.

I compiti del gestore della memoria sono sostanzialmente tre:
- sapere sempre quali parti della memoria centrale sono in uso e quali sono libere;
- scegliere quale parte di memoria allocare ai processi che la necessitano e quindi deallocarla;
- gestire lo swapping tra la memoria principale e il disco quando la memoria principale non è sufficientemente grande per mantenere tutti i processi.

Caricamento del programma

Il programma eseguibile, in formato binario, risiede in un file su una memoria permanente, tipicamente un hard disk (memoria secondaria).

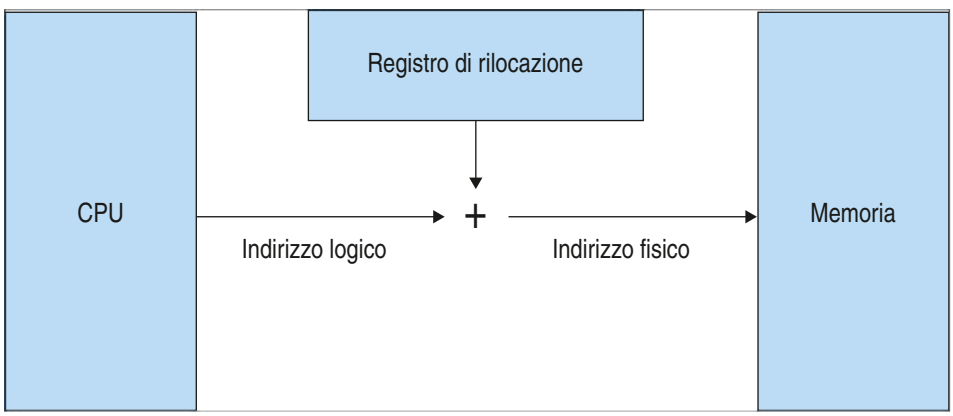
Il problema fondamentale che il gestore della memoria deve risolvere è trasformare il programma eseguibile (su memoria di massa) in un processo in esecuzione (in memoria di lavoro).

I programmi che "stanno per diventare processi", cioè per i quali è già stata fatta la richiesta di caricamento in memoria centrale, vengono messi in una coda di entrata, dalla quale ne verrà selezionato uno (o più) da caricare da parte del loader e quindi da collocare nella lista dei processi pronti (RL).
È opportuno fare una precisazione: durante la generazione del file eseguibile il compilatore e il linker generano all'interno del programma dei collegamenti tra istruzioni e indirizzi senza sapere dove il programma o i dati saranno caricati in memoria.
Inoltre è anche molto improbabile che un programma venga caricato a partire dalla stessa cella di memoria in esecuzioni diverse sullo stesso calcolatore o su due macchine differenti.

L'assegnazione degli indirizzi è quindi incompleta: vengono cioè generati degli indirizzi relativi (indirizzo logico) e all'atto del caricamento vero e proprio questi vengono trasformati in indirizzi assoluti (indirizzo fisico).

Un codice che ha tali caratteristiche si chiama codice rilocabile: vediamo un semplice esempio di funzionamento in due possibili situazioni.
Il compilatore genera gli indirizzi ipotizzando che il programma venga caricato nella memoria a partire dalla cella 0 e in base a questo valore vengono generati tutti gli indirizzi e i collegamenti dati/istruzioni:

- all'atto del caricamento in memoria viene individuato l'indirizzo iniziale, indirizzo di base, e viene sommato a tutti i riferimenti presenti nel programma (offset): questo modo di procedere prende il nome di rilocazione statica;
- il programma viene caricato in una zona libera di memoria e solo in fase di esecuzione viene inserito in un apposito registro, chiamato registro base, il valore dell'indirizzo effettivo della prima locazione di memoria centrale; quindi durante l'esecuzione, istruzione per istruzione, si calcola l'indirizzo assoluto sommando a ogni indirizzo relativo il contenuto del registro base. Questo modo di procedere prende il nome di rilocazione dinamica.



La formula generale è quindi:

indirizzo fisico = indirizzo logico + offset

dove l'offset è lo spiazzamento tra lo 0 logico del processo e il suo indirizzo di base, cioè la sua posizione effettiva della prima istruzione in memoria centrale (ed è contenuto nel registro di rilocazione).

Quindi gli indirizzi logici vanno da 0 a un valore massimo, mentre i corrispondenti indirizzi fisici vanno da R + 0 a R + valore massimo (dove R è il valore di offset presente nel registro di rilocazione).

Il passaggio dall'indirizzo logico all'indirizzo fisico si definisce address binding.

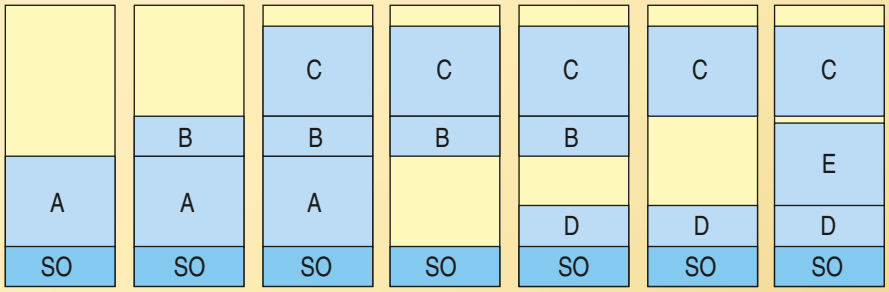
BINDING E LINKING

Il calcolo degli indirizzi logici viene effettuato nella fase di linking dei programmi; il passaggio dall'indirizzo logico a quello fisico avviene nella fase di binding che può essere fatta in momenti diversi:

- durante la compilazione (indirizzamento assoluto);
- nel momento del loading del programma (rilocazione statica);
- durante l'esecuzione, soprattutto se si hanno librerie dinamiche (rilocazione dinamica).

Sappiamo che per essere eseguito un programma deve risiedere in memoria centrale, ma per poterlo fare è necessario che lo spazio libero in memoria sia sufficientemente grande da poterlo contenere e inoltre che questo spazio sia contiguo.
Possiamo subito fare delle semplici osservazioni:

- oggi i programmi hanno dimensioni notevoli sicuramente superiori alla dimensione della memoria RAM: è impensabile che il programmatore scriva un programma cercando di risparmiare spazio, come si faceva negli anni '80!
- nei sistemi multiprogrammati il continuo caricamento e scaricamento dei programmi produce una frammentazione della memoria creando delle regioni libere di dimensioni spesso ridotte: magari la somma dello spazio totale libero sarebbe sufficiente a contenere il programma, ma le zone non sono contigue e il SO dovrebbe effettuare un'operazione di compattamento della memoria spostando tutti i programmi già caricati (ed effettuando per ciascuno una rilocazione dinamica, molto costosa in termini di tempo, quindi di efficienza);



- possiamo però osservare che non tutte le istruzioni che sono scritte in un programma devono contemporaneamente essere presenti in memoria anche perché magari buona parte di queste non verranno mai eseguite (si pensi a tutte le opzioni "inutilizzate" da programmi tipo Word o Excel!).

Procediamo, di seguito, all'illustrazione di alcune possibili soluzioni.

- Nei sistemi multiprogrammati è possibile effettuare lo "scaricamento" dei processi temporaneamente inattivi dalla RAM a disco per liberare spazio in caso di necessità: questa

operazione prende il nome di swapping e genera un insieme di operazioni che rallentano notevolmente il SO provocando un calo di prestazioni.
Nello specifico lo swapping si compone di quattro fasi:
– *identificare* i processi inattivi presenti in memoria (per esempio in stato di attesa);
– *salvare sulla memoria temporanea* i loro dati (dati globali, heap, stack);
– rimuoverli dalla memoria centrale (*scaricamento*);
– caricare nello spazio appena liberato il processo che deve essere eseguito (*caricamento*).

- Si possono caricare in memoria solo parti fondamentali del programma e lasciare su disco moduli che solo al momento di un'effettiva richiesta saranno caricati successivamente in memoria: questa tecnica si chiama caricamento dinamico e viene molto usata soprattutto nel collegamento alle librerie di sistema (◀ DLL ▶, Dynamic Link Library).
- Nel caso di processi con dimensione maggiore della memoria centrale è possibile effettuare a priori già un frazionamento del programma a opera del programmatore, individuando le parti che possono essere tra loro "alternative" e che verranno caricate in memoria nella stessa zona proprio alternativamente. La tecnica si chiama overlay, a indicare che nella stessa zona di memoria vengono sovrapposte più sezioni di programma, naturalmente una alla volta.
- Molti problemi nascono dalla differenza di dimensione tra i programmi; la soluzione ottimale sarebbe quella di riservare a ogni processo una dimensione fissa di memoria per semplificare le operazioni di swapping e ridurre la frammentazione: a tal fine viene utilizzata la tecnica di partizionamento della memoria.

◀ Le DLL (Dynamic Link Library, cioè librerie a collegamento dinamico) sono librerie che non vengono collegate staticamente a un programma eseguibile in fase di compilazione, ma vengono caricate dinamicamente in fase di esecuzione. Nel sistema operativo Microsoft Windows sono file con estensione DLL. ▶

Allocazione della memoria – partizionamento

Il sistema più semplice per allocare la memoria centrale nei sistemi multiprogrammati è quello di suddividerla in partizioni e assegnare una partizione a un processo indipendentemente dalla sua dimensione, partendo dalla prima (che viene generalmente riservata al sistema operativo) e cercando di riempire contiguamente tutte le partizioni. Naturalmente la scelta della dimensione della partizione è fondamentale per l'efficienza del sistema in quanto effettuare partizioni troppo piccole potrebbe provocare problemi di frammentazione, mentre segmenti troppo grandi rischiano di sprecare memoria con i processi di piccole dimensioni e di aumentare le richieste di swapping per mancanza di memoria.

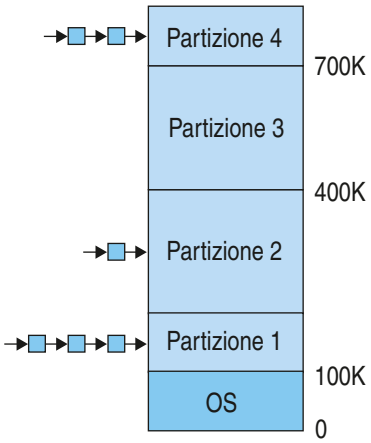
Di seguito vengono discussi due diversi schemi di partizionamento.

Schema a partizione fissa

PARTIZIONE FISSA

Nello schema a partizione fissa la dimensione della partizione viene definita all'atto dell'inizializzazione del sistema, quindi *staticamente*, e viene creata una tabella dove si memorizza lo stato delle partizioni, indicando quali sono libere e quali occupate (e da chi sono occupate).

Le partizioni possono anche essere di dimensione diversa tra loro e possono essere stabilite dall'operatore all'avvio del SO.



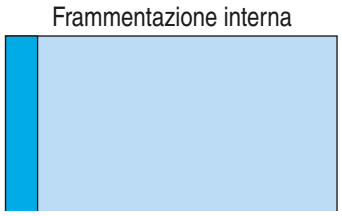
Per ogni partizione viene successivamente gestita una coda dei processi in attesa in base alle loro dimensioni (come nell'esempio della figura a lato): un nuovo job viene aggiunto alla coda relativa alla partizione più piccola che è in grado di contenerlo.

Nella frammentazione fissa si possono verificare due problemi:

- il problema della frammentazione interna, che si presenta quando le singole partizioni sono di grandi dimensioni;
- il problema della frammentazione esterna, che si presenta quando le singole partizioni sono di piccole dimensioni.

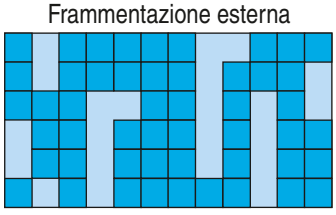
Frammentazione interna

Supponiamo che ogni processo occupi un'intera partizione e che venga schedulato un job "piccolo": se ogni partizione piccola ma sufficientemente grande per contenerlo ha la coda piena e invece la coda di una partizione grande è vuota, il job piccolo viene assegnato alla partizione grande: come si vede nella figura a lato potremmo sprecare quasi tutta la partizione grande per un job molto piccolo.



Frammentazione esterna

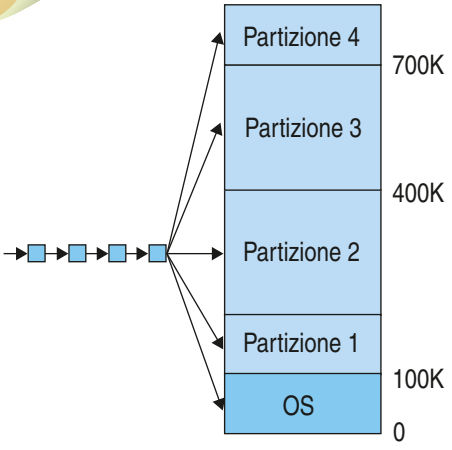
La dimensione di un processo può essere più grande di una qualunque partizione esistente anche se, nel complesso, la memoria associata a tutte le partizioni è sufficiente per contenerla.



Il grado di multiprogrammazione è limitato al numero di partizioni: non posso eseguire processi se non sono in grado di allocare una partizione.

Il problema della frammentazione interna si può risolvere mediante l'uso di una singola coda di ingresso; quando si libera una partizione, si possono avere due strategie di assegnazione ai job:

- si percorre la coda a partire dalla testa fino a individuare un job di dimensioni tali da poter essere contenuto;
- si percorre tutta la coda individuando il job più grande che può essere contenuto nella partizione che è libera: in questo modo, però, vengono discriminati i job di dimensione ridotta.



Schema a partizione variabile

PARTIZIONE VARIABILE

Schema a partizione variabile: in questo caso è possibile modificare dinamicamente sia il numero sia la dimensione di ogni singola partizione, per esempio unendo blocchi contigui precedentemente distinti o suddividendone uno particolarmente sovradimensionato, a seconda delle richieste di spazio all'atto del caricamento dei processi.
È anche possibile creare blocchi di dimensione specifica per il nuovo processo direttamente al momento del suo caricamento: avremo quindi tante partizioni quanti sono i processi attivi.

In questo caso il lavoro del SO è più complesso, in quanto lo spazio disponibile per i nuovi processi continua a variare e quindi deve decidere continuamente dove caricare i nuovi processi che attendono nella coda di entrata: utilizza gli algoritmi di allocazione dinamica della memoria centrale.

Il problema dell'allocazione dinamica della memoria centrale ha come strategia risolutiva tre possibili alternative:

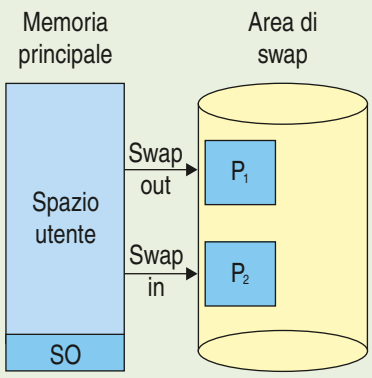
- first-fit: il gestore della memoria scandisce la tabella dei segmenti finché trova la prima zona libera abbastanza grande per contenere il programma; è molto veloce perché la ricerca finisce subito al primo matching;
- best-fit: il gestore della memoria scandisce tutto l'elenco dei segmenti e sceglie la zona libera più piccola sufficientemente grande da contenere il processo; è un metodo più lento del first-fit e spreca più memoria perché lascia zone di memoria troppo piccole per essere utilizzate;
- worst-fit: per risolvere il problema precedente sceglie la zona libera più grande ma, statisticamente, si è verificato che questo è il peggior algoritmo di allocazione.

Le simulazioni hanno dimostrato che le strategie migliori sono le prime due, in particolar modo la first-fit, che è la più veloce.

Zoom su...

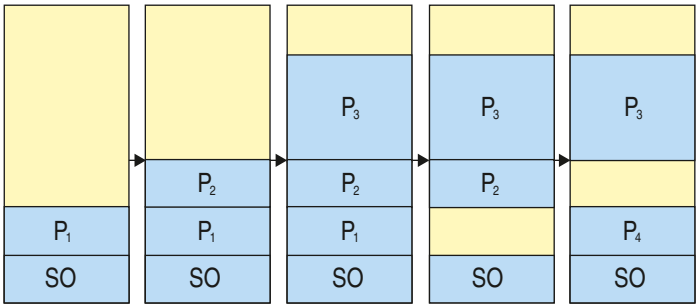
AREA DI SWAP

Per ottimizzare l'utilizzo della memoria generalmente il partizionamento dinamico fa uso del disco come area di appoggio (backing store o swap) per i processi che per un qualsiasi motivo non sono in grado di continuare e passano nello stato di wait: per esempio, un processo che effettua una chiamata di I/O viene bloccato e la sua immagine trasferita interamente su swap (swap out) e quando la chiamata di I/O è completata, e quindi il processo può riprendere la sua normale evoluzione, l'immagine del processo viene nuovamente trasferita in memoria centrale (swap in) per riprendere l'esecuzione.



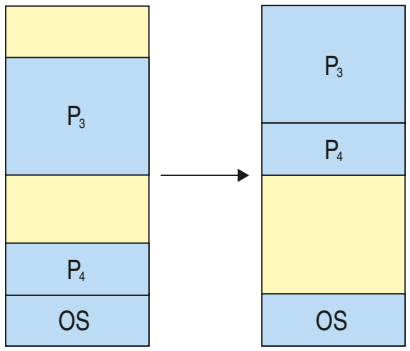
Come possiamo vedere dalla figura che segue, che simula lo stato della RAM in seguito all'esecuzione di una sequenza di processi, il partizionamento dinamico è generalmente soggetto a frammentazione esterna: infatti la terminazione dei processi P1 e P2 e il successivo caricamento del processo P4 hanno provocato la formazione di due frammenti di dimensioni modeste all'interno della RAM.

Una possibile soluzione è quella di spostare periodicamente i processi all'interno della RAM per fare in modo di "ricompattare le aree di memoria libere" (memory compaction) ottenendo quindi un'intera zona da riutilizzare. ▶



Questa procedura è simile a quella utilizzata sui dischi rigidi (defrag) per togliere la deframmentazione che, per esempio applicata nel caso sopra descritto, porterebbe ad avere la situazione riportata nella figura seguente. ▼

È però importante osservare che l'operazione di compattazione della memoria è estremamente onerosa dal punto di vista computazionale (per esempio, per ricopiare una memoria di 1 MB si impiega un secondo, dato che la velocità tipica della copia è di 1 byte/ms e durante questo arco di tempo nessun processo può ovviamente essere eseguito.
Potrebbe essere un problema di difficile soluzione soprattutto nei sistemi interattivi, dove provocherebbe il decadimento delle prestazioni.



Memoria virtuale: introduzione

Entrambe le tecniche descritte in precedenza determinano problemi di frammentazione della memoria lasciando dei blocchi di memoria liberi tra quelli occupati, blocchi che, se fossero uniti e contigui, potrebbero ospitare un altro processo.
Man mano che si utilizza il sistema, lo spazio perso diventa considerevole, provocando un lento ma inesorabile declino delle prestazioni.
La tecnica della compattazione, ovvero la fusione di tutti i blocchi liberi in uno solo, non sempre può essere applicata e in ogni caso è piuttosto onerosa in termini di computazione.

I moderni sistemi operativi introducono quindi il concetto di memoria virtuale, realizzata con le tecniche di paginazione e segmentazione che vedremo nelle pagine che seguono.

L'idea di base è quella di fare in modo che il SO mantenga in memoria principale solo le parti del programma in uso e trattenga il resto su disco (idea, tra l'altro, già utilizzata nelle tecniche di swapping e di caricamento dinamico, ma realizzata in modo diverso).

Statisticamente nei programmi vale il principio di località: quando viene eseguita un'istruzione, la probabilità più alta è che subito dopo ne venga eseguita una ad essa vicina: quindi è sufficiente avere caricato un "pezzo" di codice mentre il resto può rimanere sulla memoria di massa.

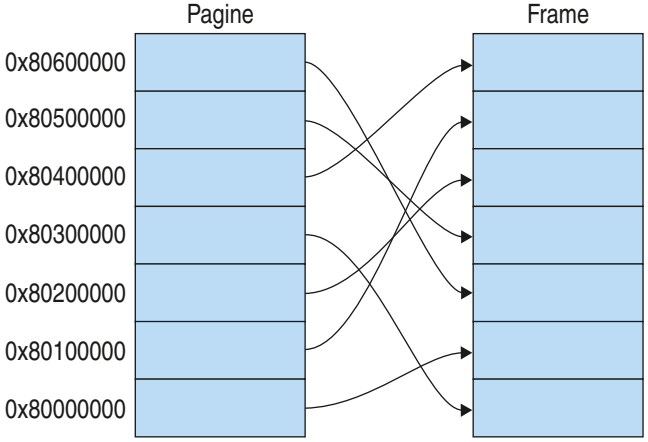
Caricando un pezzo di programma per volta è quindi possibile mandare in esecuzione programmi di qualunque dimensione: con una macchina a 32 bit ogni processo può indirizzare 4 GB e con questa tecnica può in effetti "credere" di averla tutta a sua disposizione, anche se praticamente gliene viene dato un "pezzo alla volta".

Memoria virtuale: paginazione

Il primo metodo che descriviamo per risolvere i problemi appena descritti è la paginazione.

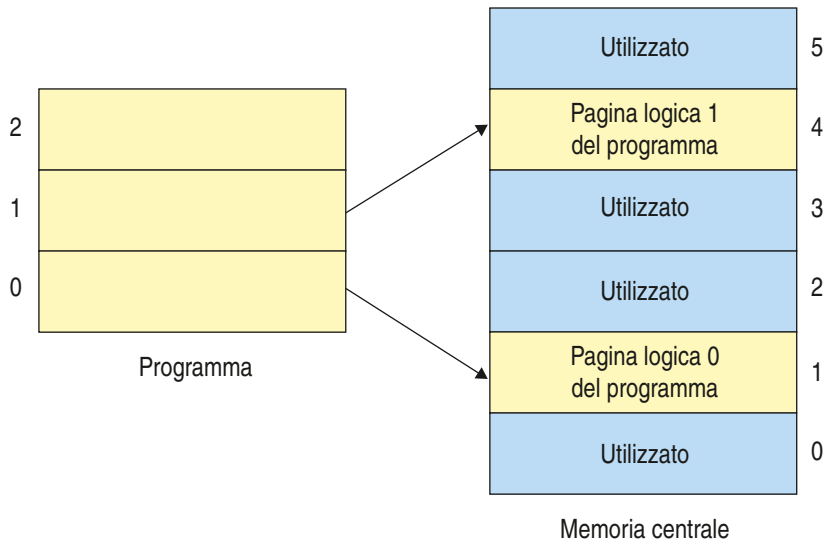
Gli obiettivi della paginazione sono i seguenti:
- mantenere in memoria solo le parti necessarie;
- gestire ogni volta solo piccole porzioni di memoria;
- non sprecare spazio evitando la frammentazione;
- poter utilizzare porzioni di memoria non contigue per lo stesso programma;
- non porre vincoli al programmatore (come nelle overlay).

Nella paginazione sia il programma sia la memoria centrale vengono suddivisi in pagine di dimensione fissa:
- la memoria fisica in blocchi chiamati frame o pagine fisiche;
- il programma in blocchi di uguale dimensione detti pagine (o pagine logiche).



Naturalmente il numero di pagine logiche (la cui somma costituisce la dimensione del programma) può essere diverso dal numero di pagine fisiche (la cui somma è la dimensione della memoria):
- se il numero delle pagine logiche è minore del numero delle pagine fisiche (libere) il programma potrebbe anche essere caricato tutto in memoria;
- se il numero delle pagine logiche è maggiore del numero delle pagine fisiche (libere) il programma verrà caricato in memoria parzialmente.

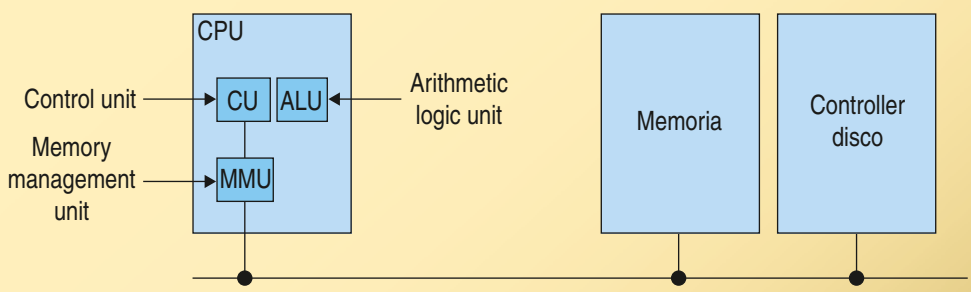
Non necessariamente le pagine fisiche devono essere contigue e quindi le pagine logiche possono trovarsi "mischiate" nella memoria centrale: naturalmente affinché un processo possa sfruttare questa tecnica di gestione condizione necessaria è che il codice sia rilocabile dinamicamente.
Un beneficio importante è che con la paginazione, per poter eseguire un programma all'atto del caricamento iniziale, è sufficiente che venga caricata in memoria la prima pagina, quella cioè che contiene la prima istruzione da eseguire: è quindi sufficiente che nella memoria fisica sia libera una sola pagina.



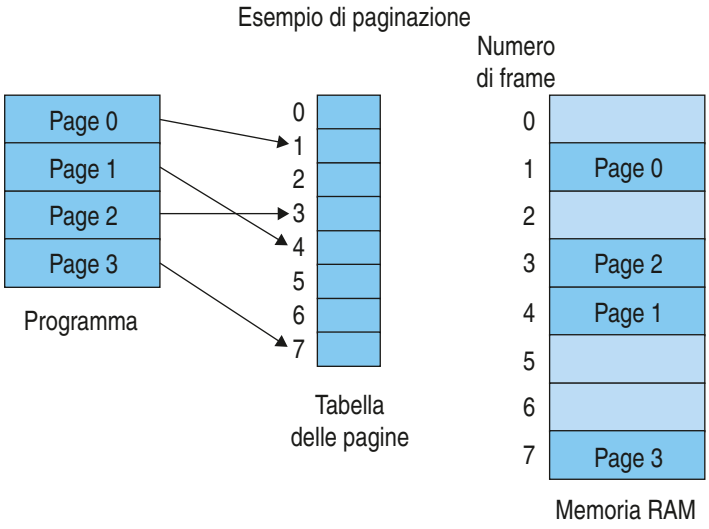
La gestione delle pagine rientra tra i compiti del SO mentre la traduzione da indirizzo virtuale a indirizzo fisico viene effettuata da un dispositivo hardware (MMU, Memory Management Unit), dove:

- lo spazio degli indirizzi logici rappresenta l'intero insieme degli indirizzi generabili dalla CPU;
- lo spazio degli indirizzi fisici rappresenta l'intero insieme degli indirizzi visibili dalla memoria.

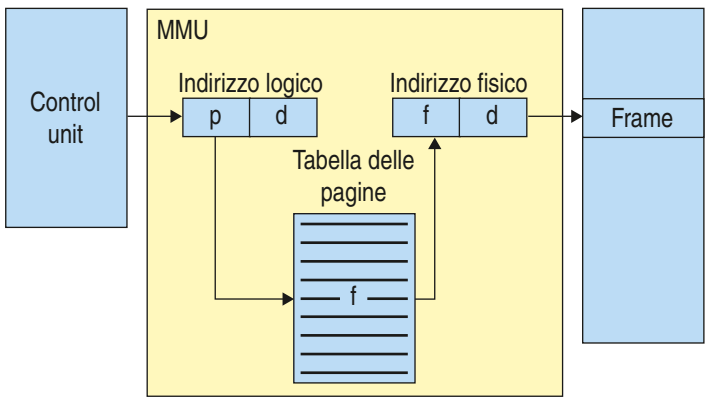
La MMU è parte integrante del processore e coopera con l'unità di controllo (CU): la CU genera indirizzi logici e li invia alla MMU, che li traduce nei corrispettivi indirizzi fisici e li scrive sul BUS (address).



Il SO cura una tabella delle pagine dove il numero stesso di pagina fisica viene utilizzato come indice: nella tabella sono memorizzati gli indirizzi fisici iniziali di tutti i frame presenti nella memoria fisica oltre alle indicazioni generali, cioè se la pagina è occupata o libera, ed eventualmente l'ID del processo che la occupa.



Inoltre, anche a ogni processo viene associata una tabella delle pagine specifica dove sono indicati quali segmenti di codice sono caricati in RAM e quali su disco.
Per ottenere un indirizzo di memoria dobbiamo ora avere a disposizione due elementi, ovvero il numero di pagina (p) e lo spiazzamento nella pagina o offset (d): la somma di questi due elementi permette di ricavare l'indirizzo fisico desiderato.
Dal numero di pagina la MMU preleva dalla tabella della pagine l'indirizzo fisico (f) al quale deve essere sommato l'offset (d).



All'atto del caricamento di una pagina il SO deve effettuare l'aggiornamento di tutti i parametri necessari per effettuare in seguito l'esatto indirizzamento dei dati e delle istruzioni.

Se durante l'esecuzione di un programma viene fatto riferimento a un'istruzione che non è presente in alcuna pagina tra quelle caricate in memoria, la MMU determina un'eccezione della CPU al SO detta page fault: il SO quindi deve provvedere a caricare quella pagina:

- se sono presenti in memoria frame liberi, la pagina viene caricata immediatamente;
- se tutte le pagine fisiche sono occupate, è necessario "fare spazio", cioè si sceglie un frame di pagina poco utilizzato, si salva il suo contenuto su disco e solo in quel momento è possibile caricare la pagina nel frame appena liberato.

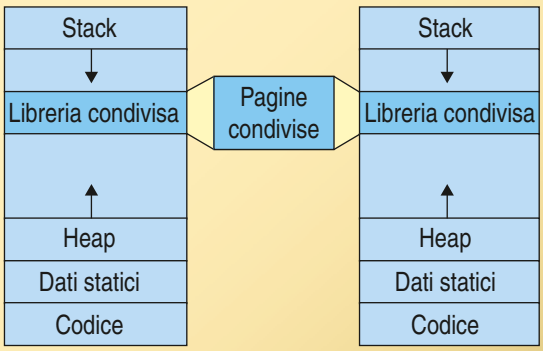
In entrambi i casi, naturalmente, si deve cambiare la mappa delle pagine e solo allora il SO riparte con l'istruzione bloccata.

Possiamo concludere dicendo che la paginazione elimina completamente il problema della frammentazione della memoria (frammentazione esterna) in quanto le dimensioni delle pagine e quelle dei frame coincidono e, inoltre, possono essere assegnate anche in modo non contiguo.

Abbiamo invece un nuovo problema, sempre dovuto all'occupazione parziale di parte della memoria: sicuramente ogni processo avrà almeno una pagina che non viene utilizzata completamente in quanto è improbabile che la sua dimensione sia proprio un multiplo esatto della dimensione di una pagina e quindi verrà a crearsi una frammentazione interna al frame, che risulterà parzialmente utilizzato.

Una soluzione immediata che ci permetterebbe di diminuire gli spazi che vengono sprecati nel frame è quella di ridurre la dimensione del frame stesso: così facendo, però, si complicherebbe notevolmente la gestione riducendo le prestazioni.

Oggi la paginazione è usata nella maggior parte dei sistemi operativi con alcune varianti ma pur sempre con questo meccanismo: è inoltre possibile condividere pagine di memoria comuni tra processi (padre e figlio generati da fork oppure da thread).
Per esempio due processi padre e figlio possono condividere delle librerie, come riportato nella figura. ▶



Una delle caratteristiche di Unix/Linux è che qualunque processo può a sua volta generarne altri, detti processi figli (*child process*). Ogni processo è identificato da un PID univoco assegnato in forma progressiva quando il processo viene creato tramite la chiamata di un'apposita funzione: la fork(). Dopo il successo dell'esecuzione di una fork sia il processo padre sia il processo figlio continuano nella loro esecuzione a partire dall'istruzione successiva alla fork; il processo figlio è però una copia del processo padre, riceve una copia dei segmenti di testo, stack e dati ed esegue esattamente lo stesso codice del padre.

Zoom su...

PROTEZIONE DELLA MEMORIA

A ogni frame vengono associati alcuni bit di protezione che possono indicare se l'accesso al frame è consentito in sola lettura, in lettura-scrittura oppure in sola esecuzione.
Un ulteriore bit aggiuntivo è detto bit di validità e permette di stabilire se una pagina fa parte dello spazio di indirizzamento logico di un processo e protegge dunque la memoria centrale da accessi illegali:

- 1: la pagina fa parte dell'indirizzamento logico del processo, quindi operazione valida;
- 0: errore di violazione di protezione della memoria.

I tentativi di accesso errati (o illegali) generano una segnalazione al sistema operativo (trap) che agirà di conseguenza generalmente sospendendo il processo che sta facendo un'operazione non corretta.

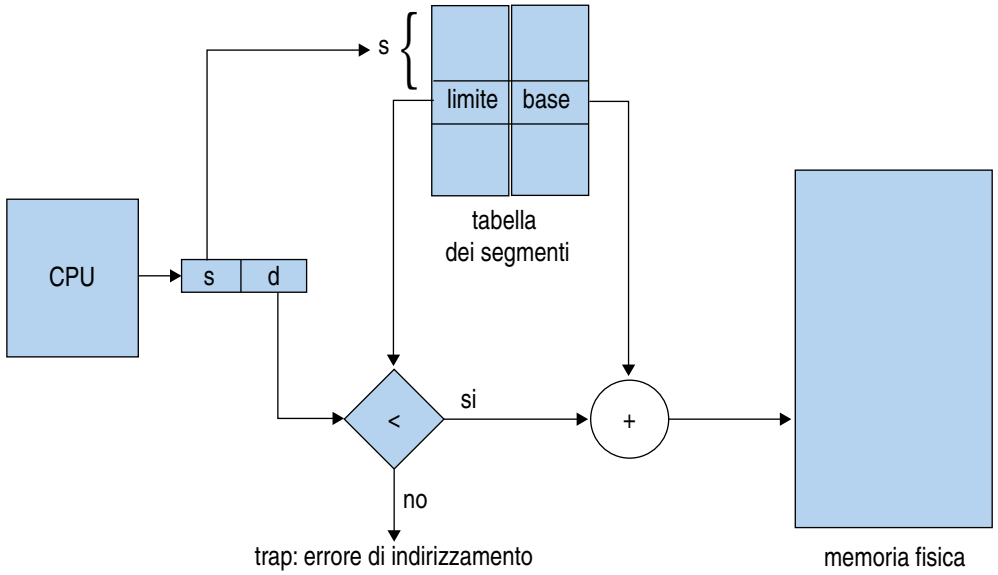
Memoria virtuale: segmentazione

Nella paginazione il programma viene suddiviso in pagine a prescindere dal suo contenuto, cioè, per esempio, senza distinguere l'area del codice da quella dei dati: questo impedisce di fatto che si possa condividere il codice su due istanze dello stesso programma, obbligando a caricare in memoria più volte lo stesso codice.
Per ovviare a questo problema si utilizza la segmentazione: è uno schema di gestione della memoria centrale che mantiene la separazione tra memoria logica e fisica come nella paginazione, ma suddivide quest'ultima in segmenti di dimensione variabile.
L'idea di base è quella di suddividere la memoria centrale nello stesso modo in cui sono divisi logicamente i programmi, cioè in entità con diverse funzioni, e quindi associare a ogni modulo software un segmento di memoria che può contenere una procedura, un array, uno stack o un insieme di variabili: un segmento è tipizzato ed è caricato in un frame di medesima dimensione.

FRAME E SEGMENTI

Nella segmentazione la memoria centrale fisica è divisa in segmenti fisici (*frame*) di dimensioni diverse, mentre lo spazio di indirizzamento del processo è diviso in segmenti logici (*segmenti*).

A ogni segmento viene associato un numero che permette di individuarlo.
Vengono tenute una o più tabelle dei segmenti dove il numero di segmento funge da indice e contiene anche l'indirizzo iniziale di memoria centrale e il limite del segmento (cioè il valore massimo che lo scostamento associato può assumere): per individuare l'indirizzo fisico assoluto è quindi sufficiente sommare tale valore allo scostamento.

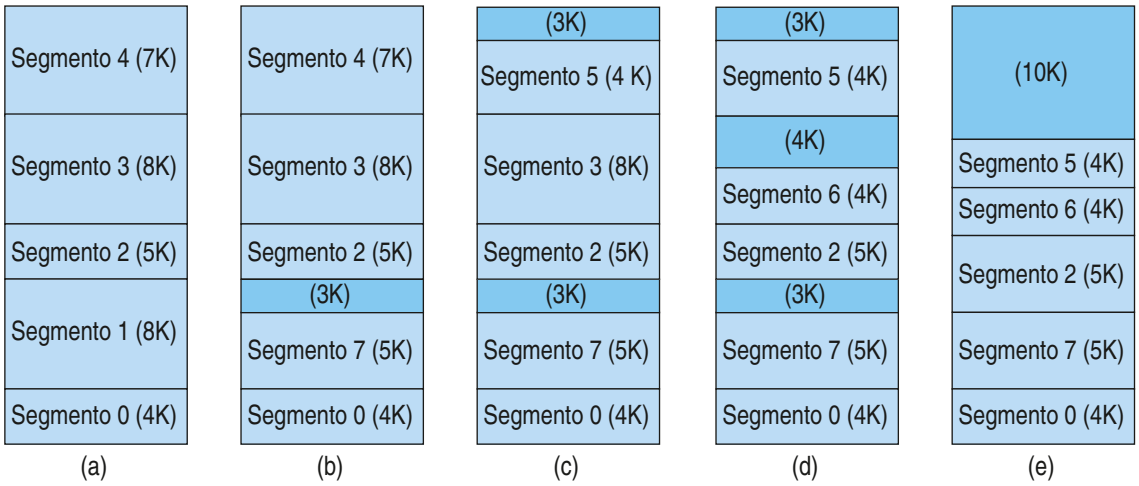


I segmenti di un processo possono essere caricati in frame non contigui in memoria centrale fisica, mentre quelli non caricati sono conservati nell'area di swap.

Con la segmentazione si ottengono i seguenti vantaggi:
- rispetto alla paginazione, la gestione di strutture dati dinamiche, che per loro natura crescono o diminuiscono, è semplificata;
- viene facilitata la possibilità di condivisione di segmenti tra alcuni processi;
- si semplifica il linking di procedure compilate separatamente;
- ogni segmento può avere un diverso tipo di protezione.

Un problema caratteristico della segmentazione è il frazionamento della memoria: questo avviene nel caso in cui i segmenti vengano continuamente caricati e sostituiti in memoria generando zone inutilizzate (frammentazione esterna chiamata anche checkerboarding).

Situazione iniziale: (d) Frammentazione eccessiva
Situazione finale: (e) Ricompattazione



L'hardware dedicato per il supporto alla segmentazione è anche in questo caso la MMU, stavolta orientata alla gestione dei segmenti.

Segmentazione con paginazione

Questa tecnica è stata proposta per sfruttare contemporaneamente i vantaggi della paginazione e della segmentazione:
- dalla paginazione si prende l'identificazione dei frame liberi, la scelta del frame libero in cui caricare una pagina, senza alcuna frammentazione;
- dalla segmentazione si prende la condivisione di porzioni di memoria, la verifica degli accessi e delle operazioni.

FRAME E SEGMENTI PAGINATI

La memoria centrale fisica è divisa in pagine fisiche (*frame*) di dimensione fissa, mentre lo spazio di indirizzamento del processo è suddiviso in segmenti logici (*segmenti*) di dimensioni diverse, ciascuno suddiviso in *pagine logiche*.

Introduciamo quindi la paginazione logica dei segmenti, dove le pagine logiche hanno la stessa dimensione dei frame.
A ogni segmento è assegnato un numero e ogni segmento è suddiviso in più pagine.
L'indirizzo logico è composto quindi da tre componenti: il numero di segmento, il numero di pagina e lo spiazzamento all'interno della pagina.

Indice di segmento	Numero di pagina	Offset interno alla pagina
18 bit	8 bit	10 bit

L'indirizzo fisico invece è analogo a quello della paginazione, cioè composto dal numero di frame e offset nel frame.

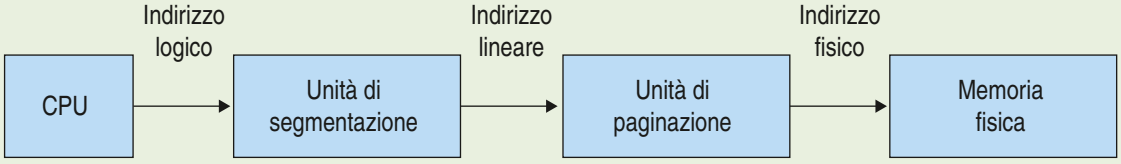
La gestione degli indirizzi è effettuata in modo completamente automatico dal SO e dalla MMU che effettua la traduzione dell'indirizzo logico in quello fisico.

Zoom su...

DA INDIRIZZO LOGICO A INDIRIZZO FISICO

Possiamo definire tre indirizzi:
- indirizzo logico: generato dalla CPU, viene consegnato all'unità di segmentazione;
- indirizzo lineare: prodotto dell'unità di segmentazione, viene affidato all'unità di paginazione;
- indirizzo fisico: prodotto dell'unità di paginazione, viene usato per accedere al dato.

La costruzione dell'indirizzo segue il percorso indicato nella figura che segue.



Verifichiamo le conoscenze

Esercizi a scelta multipla

1 Assegna a ogni memoria la sua velocità caratteristica:

a) registri ____ 10 ms
b) nastri magnetici ____ 1 ns
c) memoria RAM ____ 100 s
d) cache ____ 2 ns
e) dischi magnetici ____ 10 ns

2 Il memory manager ha tra l'altro i seguenti compiti:

- ○ sapere sempre quali parti della memoria centrale sono in uso
- ○ individuare il programma da caricare in memoria
- ○ scegliere quale parte di memoria allocare ai processi
- ○ gestire lo swapping tra la memoria principale e il disco

3 L'indirizzo logico:

- ○ viene assegnato a partire dalla cella 0
- ○ viene rilocato dal linker
- ○ viene trasformato in indirizzo fisico dal linker
- ○ viene trasformato in indirizzo assoluto dal loader

4 L'acronimo MMU significa:

- ○ *Memory Management Utility*
- ○ *Memory Maker Unit*
- ○ *Memory Management Unit*
- ○ *Memory Maker Utility*

5 Nella rilocazione statica:

- ○ viene sommato a ogni istruzione l'offset
- ○ viene utilizzato il registro base
- ○ l'indirizzo base viene sommato durante l'esecuzione
- ○ l'indirizzo base viene sommato durante la fase di link

6 Il page fault si verifica quando:

- ○ le pagine di memoria sono tutte occupate
- ○ in memoria non ci sono pagine libere
- ○ il sistema operativo non trova la pagina da caricare
- ○ un programma indirizza una pagina non presente in memoria

7 Nella paginazione (indicare l'affermazione errata)

- ○ la memoria fisica è in blocchi chiamati frame o pagine fisiche
- ○ il programma è in blocchi di uguale dimensione detti pagine
- ○ il codice deve essere rilocabile dinamicamente
- ○ non si risolve il problema della frammentazione esterna

8 Nella segmentazione (indicare l'affermazione errata):

- ○ la memoria fisica è in blocchi chiamati frame o pagine fisiche
- ○ il programma è diviso in segmenti logici (segmenti)
- ○ il codice deve essere rilocabile dinamicamente
- ○ non si risolve il problema della frammentazione esterna

Test vero/falso

1	La rilocazione statica avviene all'atto del caricamento in memoria.	V F
2	L'offset si ottiene sommando all'indirizzo di base i riferimenti.	V F
3	Nella rilocazione dinamica l'indirizzo assoluto viene determinato dal loader.	V F
4	Lo swapping avviene quando la memoria RAM libera principale non è sufficiente.	V F
5	Il gestore della memoria ha il compito di deallocare la memoria quando serve.	V F
6	Il loader preleva un processo dalla coda di entrata e lo colloca nella lista dei processi pronti.	V F
7	Il compilatore generalmente genera degli indirizzi relativi (indirizzo fisico).	V F
8	Nella rilocazione dinamica viene utilizzato il registro base per contenere l'offset.	V F

UNITÀ DIDATTICA 5

IL FILE SYSTEM

IN QUESTA UNITÀ IMPAREREMO...

- il concetto di file
- la struttura di una directory
- il modello client-server

Introduzione

Con il nome file system si intende quella parte del sistema operativo che gestisce la memorizzazione dei dati e dei programmi su dispositivi di memoria permanenti e mette a disposizione i programmi e i meccanismi necessari per la loro gestione.

Il file system, come indica anche il nome, è il "sistema di gestione dei file", ed è quindi composto dai file degli utenti e dai file di sistema che contengono dati o programmi, organizzati in directory e memorizzati in uno o più supporti che permettono di assicurare la permanenza delle informazioni: i file sono salvati su memorie secondarie (chiamate anche di massa), generalmente di tipo magnetico o magnetico/ottico, in grado di memorizzare grandi volumi di dati.

FILE SYSTEM

L'insieme di algoritmi e strutture dati che realizzano la traduzione tra operazioni logiche sui file e le informazioni memorizzate sui dispositivi fisici (dischi, nastri).

Nella realizzazione dei file system si deve tener conto in primo luogo delle esigenze dell'utente, che si possono riassumere nella necessità di:

- memorizzare informazioni in modo permanente;
- memorizzare enormi quantità di informazioni;
- accedere contemporaneamente agli stessi dati da parte di più processi;
- accedere velocemente ai dati.

Il file system è anche la parte di sistema operativo che maggiormente comunica con l'utente e spesso viene identificato con il sistema operativo: si colloca come interfaccia tra l'utente e i dispositivi dando una rappresentazione astratta e unificata dei dispositivi fisici effettivi presenti nel sistema.

Il concetto di file

L'elemento fondamentale del file system è proprio il file, il cui concetto è qualcosa di estremamente generale: diamone alcune definizioni.

FILE

Dal punto di vista dell'utente un file è un insieme di dati correlati tra loro e associato in modo univoco a un nome che lo identifica, memorizzato in un dispositivo di memoria secondaria.

Dal punto di vista del sistema operativo, un file è un insieme di byte (eventualmente strutturato).

Sappiamo che in un file possono essere memorizzati tipi diversi di informazioni: programmi sorgente o eseguibili, testi, immagini; ogni file ha quindi una particolare strutturazione a seconda del tipo di dato che memorizza.

Per ogni file il SO raccoglie un insieme di informazioni in un record, il descrittore del file, che ne è la carta di identità, e contiene:

- nome: è composto da una stringa di caratteri con lunghezza variabile a seconda del SO e viene detto anche nome simbolico; alcuni SO fanno distinzione tra lettere maiuscole e minuscole (◀ case sensitive ▶ naming);
- identificatore: un valore numerico che lo identifica in modo univoco nel file system; non è un dato leggibile dall'uomo e, di solito, è usato dal SO per referenziare a più basso livello il contenuto del file;
- tipo: informazione necessaria ai sistemi che gestiscono tipi di file diversi, generalmente incluso nel nome sotto forma di estensione, ovvero un codice solitamente di tre caratteri che lo identifica;
- locazione: il puntatore al dispositivo e alla locazione fisica del file nel dispositivo;
- dimensione corrente: espressa in byte, parole o blocchi;
- data e ora: informazioni sulla sua creazione e sull'ultimo accesso che ha portato qualche modifica ai dati in esso contenuti.

◀ Case sensitive significa letteralmente "sensibile alle maiuscole" e si riferisce al fatto che in alcuni sistemi operativi e linguaggi di programmazione viene fatta la differenza tra lettere minuscole e lettere maiuscole e quindi la dizione prova.txt individua un file diverso da Prova.txt. ▶

In SO multiutente sono previsti anche dati aggiuntivi come:

- utente proprietario: utente che ha creato il file e ne ha i massimi diritti;
- permessi: alcuni flag che mantengono le informazioni per il controllo di chi può accedere al file, modificarlo oppure solo leggerlo.

I descrittori dei file sono memorizzati in un contenitore, la directory, anch'essa memorizzata nella memoria secondaria.

Operazioni sui file

Elenchiamo brevemente le operazioni che il file system mette a disposizione per operare con i file:

- creazione: per poter creare un file il SO innanzitutto individua la posizione in cui memorizzarlo e quindi crea il suo descrittore nella directory dove sarà inserito registrando il nome, la locazione e gli altri attributi;
- ricerca: tramite il nome simbolico il file system ricerca nelle directory individuando dove il file è memorizzato e quindi carica in memoria il suo descrittore;
- scrittura: per scrivere su un file è necessario innanzitutto individuarlo e questo viene fatto attraverso la ricerca; successivamente vengono passate le informazioni che devono essere scritte. Partendo dal nome il SO ricerca la directory che contiene il file e dal suo descrittore legge la posizione fisica dove è memorizzato il file su disco e inizia quindi a scriverne il contenuto;
- lettura: analogamente alla scrittura, viene fatta una chiamata di sistema passando il nome del file da leggere e il riferimento di memoria in cui mettere le informazioni dopo che sono state lette;
- posizionamento: consiste nel posizionare un riferimento (◀ puntatore ▶) all'interno di un file nella posizione desiderata (generalmente l'operazione che effettua il posizionamento si chiama seek); in caso di fallimento ritorna un codice di errore;
- cancellazione: per cancellare un file solitamente basta cancellare il suo descrittore, in modo che venga rilasciato lo spazio su disco a lui assegnato così da poter essere utilizzato per altri file;
- troncamento: è simile alla cancellazione, ma mantiene il descrittore del file, quindi, in altre parole, si azzera il file, cancellando tutto il suo contenuto, mantenendo solo il descrittore con i suoi attributi, dove verranno azzerati la sua dimensione e l'indirizzo iniziale;
- accodamento: consiste nella scrittura di nuovi dati alla fine di un file già presente in memoria, e può essere visto come la combinazione delle operazioni di posizionamento e di scrittura (generalmente l'operazione che effettua il posizionamento si chiama append).

◀ Un puntatore è un particolare tipo di variabile di memoria in cui non viene memorizzato alcun dato ma che contiene un indirizzo di memoria. ▶

Oltre a queste operazioni il file system effettua sui file altre operazioni che lo coinvolgono direttamente:

- rinomina: consiste nel modificare il nome del file presente nel descrittore;
- spostamento: il file viene spostato in una nuova posizione del supporto, aggiornando il contenuto del suo descrittore;
- copia: viene creato un nuovo file identico a quello esistente ma, se si vuole mantenerlo nella stessa directory, gli si deve dare un nome diverso perché il nome simbolico è univoco per ogni directory.

Per poter operare sul file abbiamo visto come tutte le istruzioni necessitino di accedere al suo descrittore ed è quindi importante minimizzare i tempi per questa operazione; inoltre su un file vengono fatte sempre più operazioni contemporaneamente.
Per ottimizzare i tempi in molti SO si sono introdotte due operazioni che permettono al SO di mantenere in una tabella apposita i descrittori dei file che si stanno utilizzando:

- apertura: quando si vuole utilizzare un file lo si deve aprire: mediante l'operazione open viene caricato in memoria principale, nella tabella dei file aperti, il suo descrittore, per poi poterlo utilizzare direttamente senza dover nuovamente ricercarne le informazioni su memoria secondaria; con l'operazione di apertura viene effettuato anche un insieme di controlli:
  - l'identificazione del processo che ne richiede l'utilizzo;
  - la verifica delle autorizzazioni concesse all'utente;
  - la verifica e la gestione dello stato di uso condiviso;
- chiusura: al termine delle operazioni si richiama la primitiva close che provvede a rimuovere dalla tabella dei file aperti il nome del file da chiudere.

Zoom su...

CONDIVISIONE DI FILE

È necessario aprire una parentesi sulla condivisione dei file: nei sistemi multiutente o in caso di elaboratori connessi in rete, sia locale sia remota, la possibilità che un file serva contemporaneamente a più processi è abbastanza frequente.

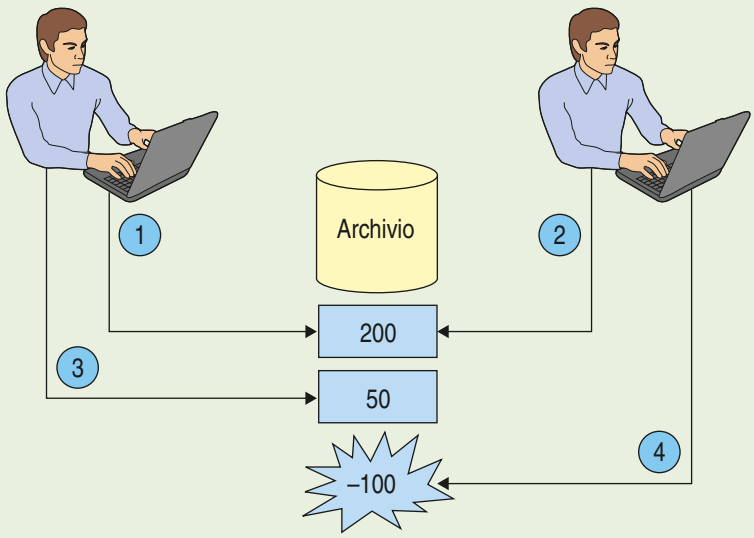
CONDIVISO

Un file si dice condiviso quando ad esso si può accedere contemporaneamente da più processi.

Si noti che questa definizione non fa distinzione tra operazioni di lettura e operazioni di scrittura, bensì accomuna tutte le operazioni come accessi all'archivio, anche se il problema nella condivisione delle informazioni racchiuse nell'archivio è determinato dalle operazioni di scrittura.
Se condividiamo un dato tra più utenti (siano essi programmi, processi o esseri umani che si interfacciano in qualche modo con l'archivio), dobbiamo assicurare che non ci siano errori nella sua condivisione.

Facciamo un esempio.
Supponiamo che in un magazzino siano presenti 200 pezzi di un determinato articolo e che due rappresentanti ricevano contemporaneamente un ordine per 150 pezzi. Entrambi i venditori si collegano all'archivio e controllano la quantità di pezzi stoccata in magazzino. Entrambi decidono che la quantità stoccata in magazzino è sufficiente e inoltrano l'ordine. Il risultato, chiaramente, è che uno dei due clienti si troverà senza merce perché non ci sono pezzi a sufficienza per entrambi gli ordini.

Passi:
1. l'utente A controlla la giacenza a magazzino e legge 200;
2. l'utente B controlla la giacenza a magazzino e legge 200;
3. l'utente A decrementa la giacenza di 150 unità e la porta a 50;
4. l'utente B decrementa la giacenza di 150 unità. La nuova giacenza è negativa!



L'esempio appena descritto rappresenta uno scenario tipico nella condivisione dei dati.

Per ovviare a questo problema alcuni sistemi operativi permettono di bloccare un file aperto (o parti di esso) per impedire ad altri processi di accedervi.

Esistono due tipi di blocco:
- blocco esaustivo o condiviso: è il caso di accesso in lettura, in quanto due o più processi possono leggere contemporaneamente le informazioni di un file;
- blocco esclusivo: in scrittura è necessario che un solo processo alla volta possa effettuare la scrittura.

Per la gestione dei blocchi i SO generalmente utilizzano una seconda tabella oltre a quella dei file aperti, una tabella che contiene i riferimenti a tutti i file aperti da ogni processo e le modalità con cui un processo ha aperto quel file.

Metodi di accesso

Per poter utilizzare le informazioni contenute nei file è necessario che queste vengano trasferite dalla memoria secondaria, generalmente da disco, alla memoria primaria (RAM).

Per recuperare un'informazione esistono due modalità, chiamate modalità di accesso:
- sequenziale: il metodo più semplice per leggere le informazioni memorizzate in un file è l'accesso sequenziale, nel quale le informazioni vengono elaborate in ordine nel file, un record dopo l'altro. Le operazioni ammesse in questa modalità di accesso sono le seguenti:
  – reset: posizionati all'inizio;
  – read next: leggi il prossimo record;
  – write next: scrivi nel prossimo record;
  – skip+/–n : avanza di +/– n record.
- diretto: è possibile effettuare l'accesso diretto o *relativo* solo sui file composti da record logici di lunghezza fissa; questo metodo viene utilizzato per accedere velocemente a

grandi quantità di informazioni (come per esempio nei database). Le operazioni ammesse in questa modalità di accesso sono quelle elencate di seguito, dove n indica l'ennesimo record del file a partire dal suo inizio:
– read n: leggi il record di posizione n;
– write n: scrivi il record di posizione n;
– position to n: posiziona il puntatore al record di posizione n;
– read next: leggi il prossimo record;
– write next: scrivi il prossimo record.

Alcuni sistemi operativi supportano sia l'accesso diretto sia quello sequenziale.

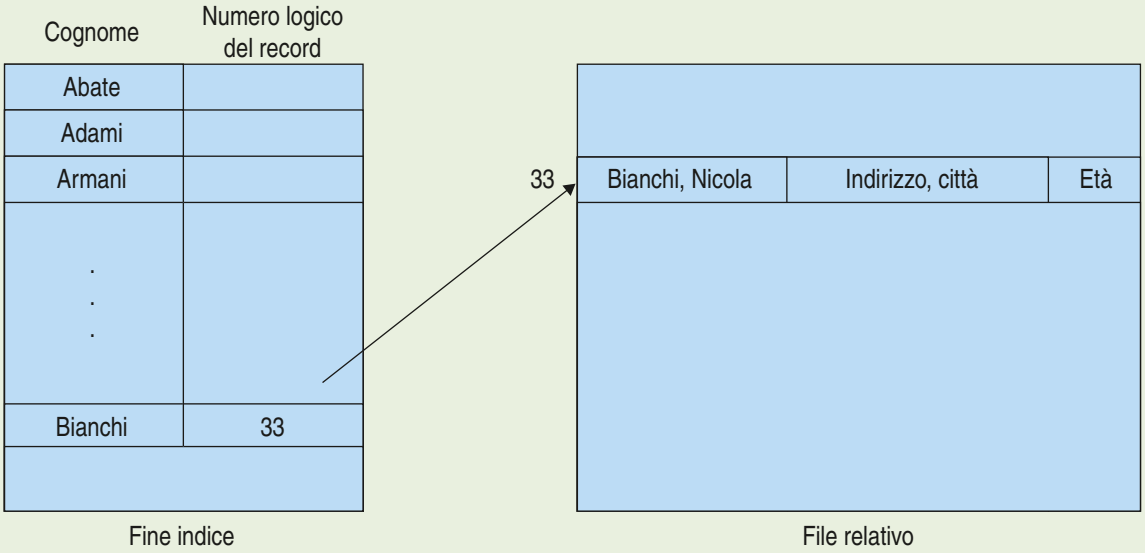
Zoom su...

ACCESSO INDICIZZATO

Utilizzando l'accesso diretto è possibile sviluppare delle tecniche che permettano un ulteriore tipo di accesso, chiamato accesso indicizzato, perché l'accesso al record desiderato avviene attraverso un indice, un file composto da record di due elementi (informazione, puntatore) dove viene creata la corrispondenza tra un dato (chiave) che permette di individuare un record nel file e la sua posizione all'interno del file (tale valore viene inserito nel puntatore).

La ricerca di un elemento avviene con due passaggi:
- si analizza l'indice e si trova l'elemento tramite la chiave di ricerca;
- si legge il puntatore che indica la sua posizione e quindi si accede direttamente ad esso nel file.

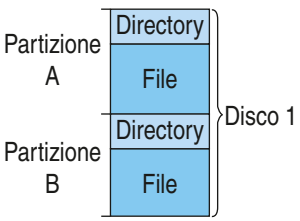
Con questa tecnica è possibile accedere velocemente a elementi in un file di grosse dimensioni con poche chiamate di I/O: è il metodo utilizzato dai database.



Struttura della directory

Prima di vedere come sono organizzate le directory è necessario analizzare come è possibile strutturare la memoria secondaria secondo un livello superiore di organizzazione che prende il nome di partizionamento.
È possibile suddividere un disco in una o più partizioni (volumi) e assegnare a ciascuna una specifica destinazione dei file; per esempio in una partizione si inserisce il SO, in una seconda i programmi di utilità, nella terza i file musicali e multimediali ecc., in modo da impostare un primo livello "logico" di separazione dei file.

Le partizioni sono porzioni indipendenti del disco che ospitano file system distinti.

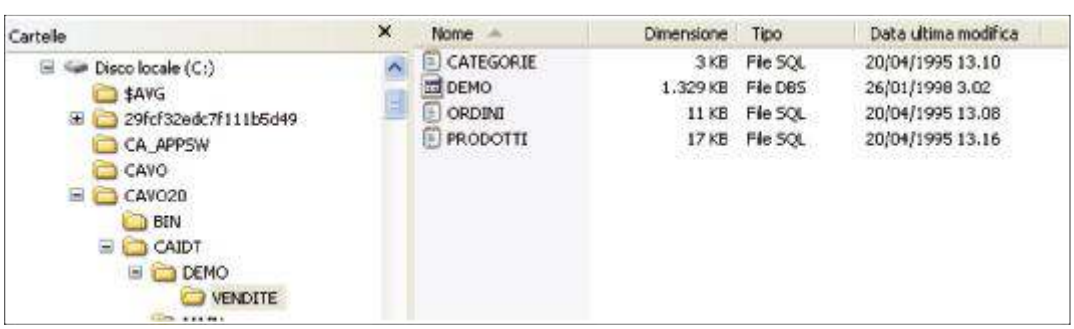


Il primo settore dei dischi è il cosiddetto *master boot record* (MBR), utilizzato per fare il boot del sistema e che contiene la partition table (tabella delle partizioni) dove viene anche indicato quale tra tutte le partizioni presenti è quella attiva.

A ogni partizione viene assegnato un nome simbolico oltre a un identificatore univoco (per esempio disco C, disco D, disco E ecc.).
Ogni partizione ha una tabella, la directory del dispositivo (o indice del volume), nella quale sono contenute tutte le informazioni sui file in essa memorizzati.
Per consentire agli utenti di raggruppare file tra loro affini sono state create le cartelle (in inglese, directory) e quindi all'interno di ogni volume i file sono organizzati in cartelle.
A loro volta le cartelle possono includere altre cartelle oltre ai file in modo da creare una struttura con molteplici livelli (albero), per permettere un'organizzazione dei file che semplifichi poi le operazioni per il loro reperimento.
Gli utenti possono così creare proprie sottodirectory e organizzare in esse i file, applicando una loro visione logica al file system.
L'albero ha una directory radice (root) dalla quale partono tutte le ramificazioni, e nel file system per ogni file è possibile definire:

- un percorso assoluto, che inizia dalla radice e attraversa tutte le sottodirectory fino al file specificato;
- un percorso relativo, che parte dalla directory corrente e comprende solo le sottodirectory.

La directory corrente è quella attualmente "aperta dall'utente" e dovrebbe essere quella che contiene i file che devono essere utilizzati.



ESEMPIO

In questo esempio si vede una rappresentazione grafica del file system di Windows, dove nel volume Disco locale C sono presenti diverse directory; selezionando una directory, nel nostro esempio VENDITE, vengono visualizzati tutti i descrittori dei file presenti in quella directory:

- il file ORDINI ha un percorso relativo alla cartella DEMO che è VENDITE\ORDINI.SQL;
- il file ORDINI ha un percorso assoluto a partire dalla root che è C:\CAVO20\CAIDT\DEMO\VENDITE\ORDINI.SQL.

La directory è una struttura di dati che può essere vista come una tabella che permette di tradurre i nomi dei file nei corrispondenti descrittori. In una directory è possibile:

- ricercare un file;
- creare e cancellare un file;
- elencare gli elementi contenuti;
- rinominare un file;
- spostarsi su un'altra cartella: ci si può posizionare su una cartella sia di livello superiore (padre) sia di livello inferiore (figlia).

Che cosa succede se rimuoviamo una directory? Se all'interno sono presenti altre directory e/o file come si deve comportare il file system se l'utente gli chiede di cancellare un intero sottoalbero?
Premesso che a livello operativo sono operazioni abbastanza semplici da realizzare, sono invece abbastanza complessi i controlli sui diritti posseduti dall'utente che richiede l'operazione, in quanto all'interno del sottoalbero potrebbero anche essere presenti file non di sua proprietà. Per esempio potrebbero esserci file condivisi.

Zoom su...

TIPI DI FILE

A ogni file, oltre al nome, viene associato un metadato: il tipo.
Quindi la struttura completa del nome di un file è la seguente:
<nome_file><separatore><estensione>
dove:

- nome_file è il nome che l'utente ha assegnato al suo file;
- separatore è un carattere che, appunto, separa il nome dall'estensione (di solito un punto);
- estensione è una stringa di 3-4 lettere rappresentante univocamente il tipo di dato contenuto nel file.

Nella tabella sono elencati i tipi di file comunemente usati.

Tipo di file	Estensione	Funzione
Eseguibile	exe, com, bin	Programma pronto per l'esecuzione
Codice oggetto	obj, o	Codice oggetto non ancora fuso in un eseguibile
Codice sorgente	c, cc, java, pas, asm, a	Codice sorgente in diversi linguaggi
Script batch	bat, sh	Comandi da far eseguire a una shell interprete
Documento di testo	txt, doc	dati in formato testuale, documenti
Libreria	lib, a so, dll	Librerie di funzioni statiche e dinamiche
Documento di stampa	ps, pdf	Linguaggi per la rappresentazione a video e per la stampa
Archivio	arc, zip, tar	Archivio di file
Multimedia	mpeg, mov, rm, mp3, avi	Formati binari contenenti informazioni audio e/o video

File nei sistemi multiutente

Nei sistemi operativi multiutente è necessario che per ogni file e cartella di sistema vengano memorizzati alcuni attributi aggiuntivi per poter implementare la condivisione e la protezione: viene introdotto il concetto di proprietario e di gruppo dove:

- proprietario: è l'utente che ha creato il file e può modificare gli attributi e garantire l'accesso;
- gruppo: sono un sottoinsieme di utenti che hanno un "motivo" per essere accomunati e hanno gli stessi diritti su un particolare file.

Naturalmente un utente può appartenere a più gruppi dato che può avere diritti diversi su più file.

Il sistema operativo associa al file l'identificatore dell'utente che lo ha creato come indicatore del proprietario.
Un accenno doveroso deve essere fatto ai file system remoti, cioè quelli che vengono realizzati mediante connessione di rete, sia private sia Internet, e permettono la condivisione di file anche su dischi fisicamente presenti su macchine diverse (DFS, file system distribuiti).

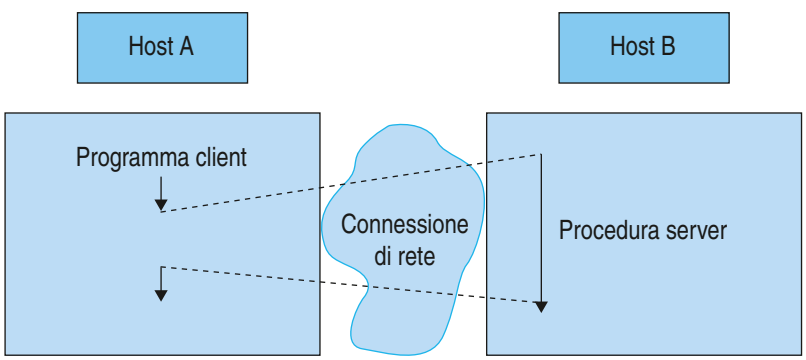
Il modello client-server

Un modello particolarmente usato nelle applicazioni distribuite come meccanismo di interazione è quello che prende il nome di client-server.

HOST, SERVER E CLIENT

Ogni calcolatore (nodo) connesso a una rete locale o remota viene chiamato **ospite** (in inglese host o end system).
La macchina che contiene i file (o, in generale, offre un servizio) è il *server*.
La macchina che chiede accesso a tali file (o richiede un servizio) è il *client*.

Il server generalmente si mette in attesa passiva di una richiesta da parte di un client: un host, quando ha bisogno di un file (o di un servizio) remoto, sospende la sua esecuzione e inizia la comunicazione richiedendo il servizio di cui necessita al server, che gli risponde esaudendo la sua richiesta; prima di effettuare la trasmissione del file viene naturalmente effettuato un insieme di controlli per garantire la "legalità" dell'operazione, cioè se l'utente che ha richiesto un accesso è autorizzato a compiere quelle operazioni.



Elenchiamo sinteticamente le caratteristiche di un client e di un server.

Client	Server
È eseguito sul computer locale È invocato dall'utente Richiede la comunicazione con il server Spedisce una richiesta Può accedere a più servizi, ma uno alla volta	È eseguito su un computer remoto Inizia l'esecuzione con l'inizializzazione del sistema Rimane in attesa di richieste dai client Spedisce una risposta (dati o file) Accetta richieste da più client

Client e server sono dei programmi, quindi non sono aggettivi riferiti al calcolatore, ma al servizio che viene offerto da un calcolatore: una macchina potrebbe essere client per alcuni servizi e contemporaneamente server per altri.
Questo tipo di modello si chiama anche peer to peer (P2P), dove tutti i nodi sono tra loro nodi equivalenti (l'inglese *peer* significa "pari, uguale").

Diritti e protezione dei file

Per tutti i file memorizzati in un calcolatore si devono attivare delle misure che garantiscano la sicurezza a fronte sia di problemi dovuti al malfunzionamento hardware (danni fisici, rotture ecc.) sia di accessi da parte di utenze non abilitate (protezione da malintenzionati o hacker).

In questa sezione non entriamo nel dettaglio su come si attuano le tecniche di back-up o di controllo degli accessi tramite procedure di login dato che non sono attività di competenza del file system, ma ne ricordiamo e sottolineiamo l'importanza dato che ci si preoccupa di proteggere i dati solo dopo che questi sono stati persi o danneggiati!

Il file system effettua l'accesso controllato alle diverse attività che si possono compiere sui file verificando il possesso dei permessi (diritti) per operazioni di lettura, scrittura, esecuzione, accodamento, cancellazione o elenco.
Ogni tipo di accesso richiede un particolare permesso e possono essere dati permessi multipli su un singolo file, permessi di accesso completo a tutte le operazioni su un file specifico oppure il medesimo permesso su tutti file presenti nella directory o nel volume.

Per controllare e proteggere gli accessi si associano i diritti all'identità degli utenti: viene associata a ogni file una lista di controllo degli accessi (ACL), che viene consultata dal sistema operativo per verificare che l'utente sia autorizzato all'accesso richiesto.

Una versione ridotta della ACL è quella che definisce solo tre tipi di utente:
- proprietario: chi ha creato il file;
- gruppo: utenti che condividono il file e hanno tipi di accesso simili;
- universo: tutti gli altri.

Con questo sistema sono necessari solo tre campi per definire la protezione, ciascuno composto da tre bit, uno per diritto.

ESEMPIO

Per esempio Unix adotta questo meccanismo e lo implementa con campi di soli tre bit ciascuno in quanto sono previste tre sole modalità diverse di accesso:
- accesso in lettura, flag r;
- accesso in scrittura, flag w;
- accesso in esecuzione, flag x.

Un esempio è il seguente : `rwxr-x--x`.
Il proprietario ha tutti i diritti (`rwx`), il gruppo ha i diritti di lettura ed esecuzione (`r-x`), gli altri solo esecuzione (`--x`).

Molto simile è anche il sistema GNU/Linux, che a ogni file assegna i permessi per 3 identità:
- il proprietario: è l'utente che ha creato il file o l'utente che root ha designato come proprietario;
- il gruppo: non necessariamente il gruppo del proprietario;
- gli altri: quelli che non fanno parte del gruppo.

Il comando ls -l permette di visualizzare i permessi di un file sotto GNU/Linux.

Anche i caratteri che indicano i permessi sono gli stessi (r,w e x) e la visualizzazione dei permessi viene rappresentata da una stringa di 9 caratteri, preceduta da un carattere che rappresenta il tipo di file.

Un esempio è: -rwxr-xr--

Si tratta di un file regolare (il primo carattere è un trattino -), il proprietario ha tutti i permessi, i membri del gruppo hanno i permessi di leggere e di eseguire(r-x), gli altri hanno solo il permesso di lettura (r--).

Sistemi più complessi permettono di combinare alcuni diritti e assegnare queste regole di accesso personalizzate per ogni utente o gruppo di utenti.

È anche possibile associare a ogni file o a ogni directory una password, ma poi l'utente è costretto a memorizzarla da qualche parte con il rischio che, per comodità, alla fine utilizzi sempre la stessa, vanificando quindi lo scopo per cui è stata introdotta questa protezione.

Verifichiamo le conoscenze

Esercizi a scelta multipla

1 Nel descrittore del file non è presente:
- O il nome del file
- O la dimensione
- O il tipo di file
- O la directory dove è memorizzato
- O i flag di protezione

2 In una directory sono presenti:
- O solo file dello stesso tipo
- O solo file con diversa dimensione
- O solo file dello stesso utente
- O solo file con diverso nome

3 Nella copia di un file il suo nome deve essere:
- O cambiato sempre
- O cambiato solo su un altro dispositivo
- O cambiato solo nella stessa directory
- O cambiato solo in altre directory

4 Quale tra le seguenti operazioni è primitiva, cioè non utilizza altre operazioni?
- O posizionamento
- O cancellazione
- O scrittura
- O lettura
- O ricerca

5 Un file si dice condiviso quando:
- O contiene dati di più utenti
- O è stato scritto contemporaneamente da più utenti
- O può essere utilizzato contemporaneamente da più processi
- O viene utilizzato da più calcolatori

6 Nel partizionamento dei dischi si ha:
- O un MBR per ogni volume
- O un MBR per ogni settore
- O un MBR nel primo settore
- O un MBR per ogni partizione

7 In un sistema client-server:
- O in ogni rete esiste un solo server
- O in ogni rete esiste un solo client
- O in ogni rete esiste una coppia client-server
- O ogni macchina può essere client o server

8 Con diritti su un file si intende:
- O la paternità del suo creatore
- O i permessi per compiere le operazioni
- O la rivendicazione economica su un file
- O nessuna delle precedenti affermazioni

Test vero/falso

1	Una memoria di massa è una memoria secondaria volatile.	V F
2	Il nome simbolico di un file è il nome espresso in simboli del file system.	V F
3	Due file con lo stesso nome possono essere memorizzati nella stessa directory.	V F
4	Due directory con lo stesso nome possono essere memorizzate nello stesso supporto.	V F
5	Con l'operazione di *seek* viene fatto il posizionamento all'interno del file.	V F
6	Con l'operazione di *append* vengono accodati nuovi dati a un file esistente.	V F
7	Un file condiviso può essere aperto contemporaneamente in lettura da più processi.	V F
8	Un file system logico è definito da ogni singolo utente.	V F
9	Un server è un calcolatore predisposto per offrire servizi.	V F
10	Un client è un programma che richiede servizi a un server.	V F
11	Un percorso assoluto va dalla directory corrente fino alla radice del volume.	V F
12	Ogni file ha associata una ACL specifica.	V F

UNITÀ DIDATTICA 6

STRUTTURA E REALIZZAZIONE DEL FILE SYSTEM

IN QUESTA UNITÀ IMPAREREMO...

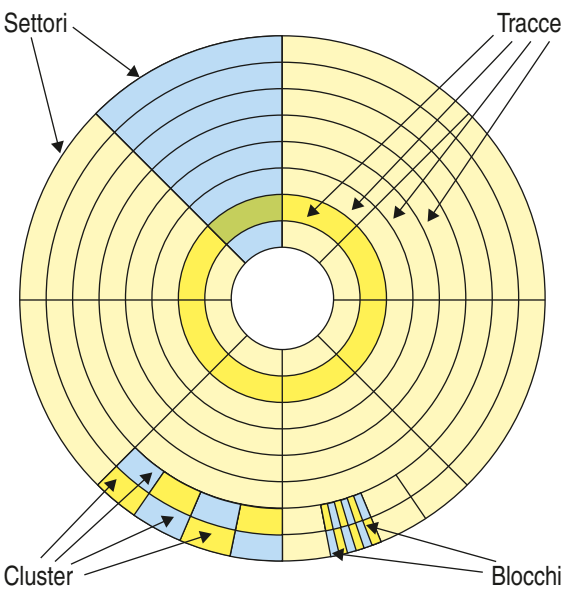
- l'organizzazione del file system
- le tecniche di realizzazione del file system

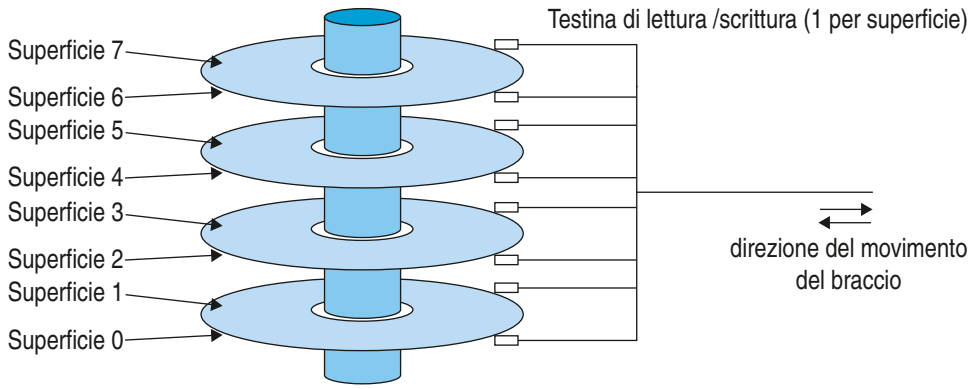
Struttura del file system

Il file system risiede permanentemente nella memoria secondaria, generalmente su uno o più dischi magnetici di dimensioni dell'ordine dei terabyte (1 TB = 1024 GB = 1.048.576 MB), quindi su un disco in grado di contenere una grande quantità di dati in modo permanente.

Il disco fisso

Lo spazio del disco fisso è logicamente suddiviso in tracce (cerchi concentrici, da 800 a 2000 tracce/cm) e in settori (spicchi).
Ogni settore è ulteriormente suddiviso in cluster, a loro volta suddivisi in singoli blocchi di dati aventi le seguenti dimensioni prestabilite: 256, 512 o 1024 byte: ogni blocco è numerato a partire da 0, per cui un disco può essere visto come un array in cui ogni elemento ha una dimensione di 512 byte. Il disco è in continua rotazione, con velocità costante da 5500 a 7200 RPM (rotazioni per minuto).
La densità di registrazione dei dati è variabile e dipende dal raggio: i cerchi più al centro hanno densità maggiore di quelli in periferia, mentre la velocità di trasferimento è indipendente dalla traccia (dai 10 ai 50 MB/s).





TEMPO DI ACCESSO

L'insieme delle tracce corrispondenti sulla stessa verticale appartenenti a superfici diverse prende il nome di cilindro.

Con t_{seek} si intende il tempo di posizionamento della testina sulla traccia (dai 5 ai 10 ms, in parte dipendente dalla distanza che deve compiere sul raggio).

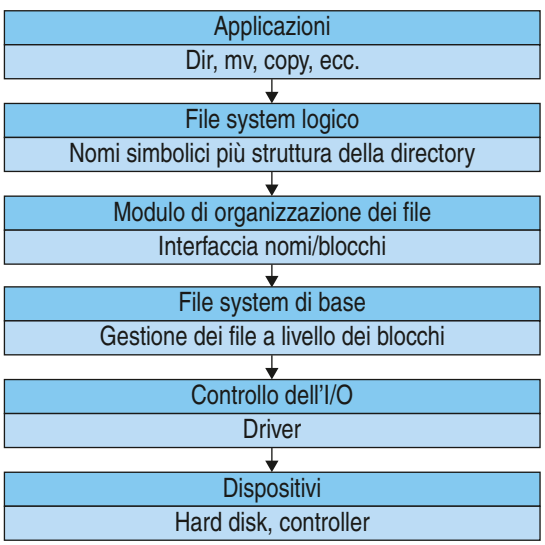
Con t_{lat} si intende il tempo di latenza, che è il tempo impiegato dalla rotazione del disco per raggiungere il settore desiderato (tempo medio 10 ms).

La somma dei due tempi viene definito come tempo di accesso $t_{acc} = t_{seek} + t_{lat}$.

Il file system

Per leggere un'informazione è necessario prima di tutto conoscere il file nel quale è memorizzata, quindi individuare il numero del blocco nel quale è scritta: successivamente il file system individua superficie, traccia e settore e solo allora il controller del disco può accedere in lettura e scrittura, che può essere fatta in modo sia sequenziale sia diretto (per ottimizzare i tempi vengono letti e scritti più blocchi simultaneamente per ogni lettura).

Inoltre sappiamo che i file sono organizzati in cartelle (directory) e queste hanno un'organizzazione complessa ad albero: sono quindi molteplici le attività che deve compiere il file system, alcune a diretto contatto con l'utente, altre direttamente sull'hardware.



La struttura del file system è perciò fatta a molteplici livelli, ognuno dei quali sfrutta le funzionalità di quelli sottostanti e crea un livello di file system virtuale, presentando agli applicativi un'astrazione indipendente dal dispositivo.

Gli obiettivi del file system sono quelli di:

- efficienza nella memorizzazione dei dati;
- efficienza nel recupero dati;
- coerenza delle informazioni;
- efficienza nell'organizzazione dello spazio libero.

La "stratificazione" del file system utilizza per tali scopi un notevole quantitativo di strutture dati, alcune localizzate su disco, altre in memoria centrale, che elenchiamo di seguito.

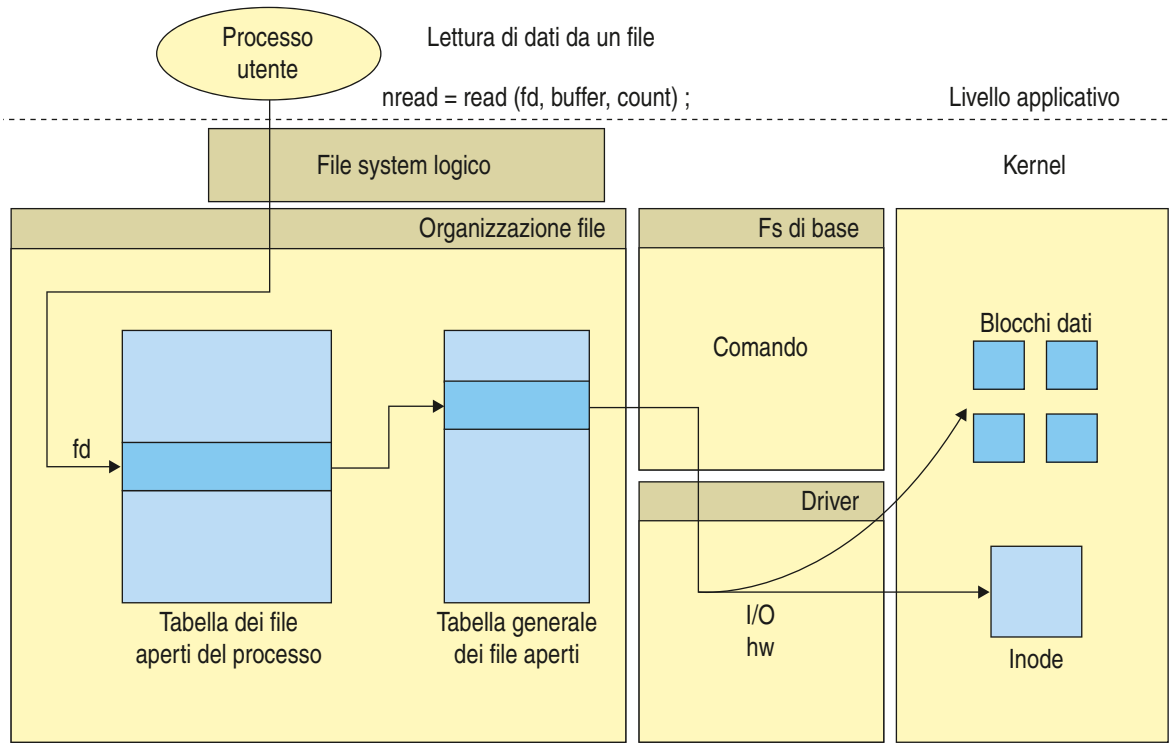
Strutture localizzate su disco

- Boot control block: è il primo blocco di una partizione e contiene le informazioni necessarie per far partire un sistema operativo da quella partizione.
- Partition control block: è noto anche con il nome di superblocco e contiene le strutture dati che gestiscono i file e lo spazio libero su disco.
- Strutture di directory: contengono l'elenco dei file presenti nella specifica cartella.
- Descrittori di file: contengono le informazioni associate a un file.

Strutture localizzate in memoria

- Tabella delle partizioni: contiene informazioni legate a ciascuna partizione presente sul disco (dimensione, locazione del blocco iniziale, locazione del blocco finale, tipo).
- Strutture di directory alle quali si è acceduto più di recente: gli accessi alle strutture di directory più recenti vengono mantenuti in memoria centrale, per evitare ulteriori accessi al disco.
- Tabella dei file aperti: è un array di descrittori dei file attualmente aperti.
- Tabella dei file aperti di ciascun processo: è un array di descrittori dei file attualmente aperti per il processo in questione.

Il disegno sottostante rappresenta le sequenza delle tabelle/strutture che vengono utilizzate a seguito di un'istruzione di lettura.



Vedremo di seguito, in dettaglio, ogni singola operazione partendo dal basso, cioè da come gestire i singoli dati nei blocchi di memoria per la memorizzazione di un file.

Allocazione di un file

Generalmente i file che devono essere memorizzati hanno una dimensione molto maggiore di quella di un singolo blocco: è quindi necessario suddividere il file in più blocchi, ed è possibile adottare diverse tecniche per allocare i blocchi del medesimo file sul disco:

- allocazione contigua;
- allocazione concatenata;
- allocazione indicizzata.

Allocazione contigua

Nell'allocazione contigua ogni file è allocato in un insieme di blocchi contigui, quindi uno di seguito all'altro, e per poter accedere a un file è necessario avere due informazioni:

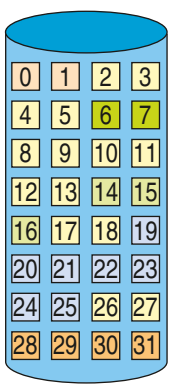
- il numero del primo blocco del file;
- quanti blocchi contigui sono occupati dal file.

Conoscendo il primo blocco b che occupa un certo file e il numero n che indica il numero di blocchi, cioè la dimensione del file, basta caricare in memoria i blocchi che vanno da b a b + n – 1.
La tecnica di allocazione contigua è stata usato nei vecchi sistemi IBM: nella figura che segue sono evidenziati i blocchi che memorizzano i file della directory descritta nella tabella.

Vantaggi

- L'accesso al file è molto semplice e veloce, così come è veloce il caricamento di tutti i blocchi che lo compongono dato che sono sequenziali.
- Sono poche le informazioni aggiuntive da salvare insieme al file, proprio perché si conosce la posizione dei blocchi sul disco.

File	Blocco iniziale	Lunghezza
Prova	0	2
Mio	14	3
Ali	19	7
Baba	28	4
Pippo	6	2



Svantaggi

- il primo problema si presenta al momento della memorizzazione del file su disco: è necessario trovargli uno spazio libero che sia contiguo, quindi potrebbe essere necessario spostare gli altri file per creare una zona abbastanza capiente;
- il secondo problema è quello della frammentazione esterna del disco, problema già incontrato nella RAM, e anche in questo contesto si utilizzano strategie di allocazione dello spazio di tipo first-fit, best-fit e worst-fit;
- il terzo problema lo riscontriamo quando un file cresce di dimensione e supera lo spazio in cui era precedentemente allocato: per risolverlo dobbiamo necessariamente spostare il file in una nuova area sufficientemente capiente riportandoci a uno dei primi due problemi.

Una soluzione che permette di ridurre la frequenza con cui si verificano le situazioni segnalate è quella di effettuare periodicamente offline un'operazione di ricompattazione del

disco (quella che prende il nome di defrag), ma è molto dispendiosa rispetto al compattamento della memoria primaria.

Allocazione concatenata

Una semplificazione dei problemi visti nell'allocazione contigua l'abbiamo con l'allocazione concatenata: a partire dal primo ogni blocco contiene un puntatore al blocco successivo del file, creando una *lista di blocchi*.

L'indirizzo del blocco successivo è memorizzato negli ultimi byte del blocco e in quello terminale si mette il valore particolare (per esempio un numero negativo, generalmente –1) per segnalare al sistema che quello è l'ultimo blocco del file.

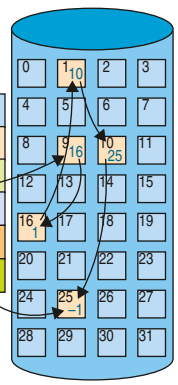
Il numero del blocco iniziale viene memorizzato nel descrittore del file, dove alcuni sistemi memorizzano anche il numero di blocchi usati e il numero del blocco finale.

Per aggiungere un blocco di dati si deve dapprima identificare un blocco libero sul disco, quindi lo si concatena alla lista dei blocchi associata al file e infine si scrivono i dati nel blocco.

Per modificare un blocco di dati esistente si deve effettuare la scansione della lista dei blocchi fino a quando non si identifica il blocco voluto e solo allora è possibile effettuare la modifica dei dati.

La figura a lato evidenzia un esempio di memorizzazione del file prova: a partire dal primo blocco (9) è possibile seguire tutta la catena fino al blocco terminale 25 che contiene –1. ▶

File	Blocco iniziale	Lunghezza
Prova	9	5
Mio	14	30
Ali	5	6
Baba	8	4
Pippo	11	31



Vantaggi

- Non si verifica la frammentazione esterna in quanto ogni blocco libero, anche singolo, può essere usato per memorizzare un nuovo file.
- Non occorre avere blocchi contigui e quindi non è necessario ricompattare il disco: questa operazione può essere fatta solo per motivi di efficienza, per migliorare le prestazioni nella lettura di grossi file, in quanto si impiega naturalmente meno tempo per leggere tanti blocchi sparsi piuttosto che blocchi contigui (tempo di posizionamento della testina di lettura/scrittura).

Svantaggi

- È necessario utilizzare dello spazio aggiuntivo in ogni blocco per memorizzare il puntatore al blocco successivo: si memorizza il numero di blocco successivo negli ultimi byte, e quindi, nel caso di blocchi da 512 byte, dovendo occupare 4 byte per puntatore, si "perde" circa lo 0,78%;

- l'accesso diretto all'interno della struttura è lento in quanto per raggiungere una particolare informazione è necessario scorrere tutti i blocchi precedenti (per leggere il blocco n sono necessari $n - 1$ accessi al disco);
- se si perde un collegamento si perde tutto il contenuto del rimanente segmento del file: quindi il sistema di allocazione è poco affidabile.

Per risolvere il problema di un'eventuale perdita di dati si introduce un sistema di doppi puntatori creando una lista bidirezionale: se si danneggia il collegamento a un blocco possiamo recuperare i successivi ripercorrendo la catena all'indietro.
Inoltre, per cercare di mantenere i vantaggi dell'allocazione contigua, si considera il disco formato da cluster di blocchi adiacenti e si aumenta la dimensione del blocco a 2 o 4 kb.

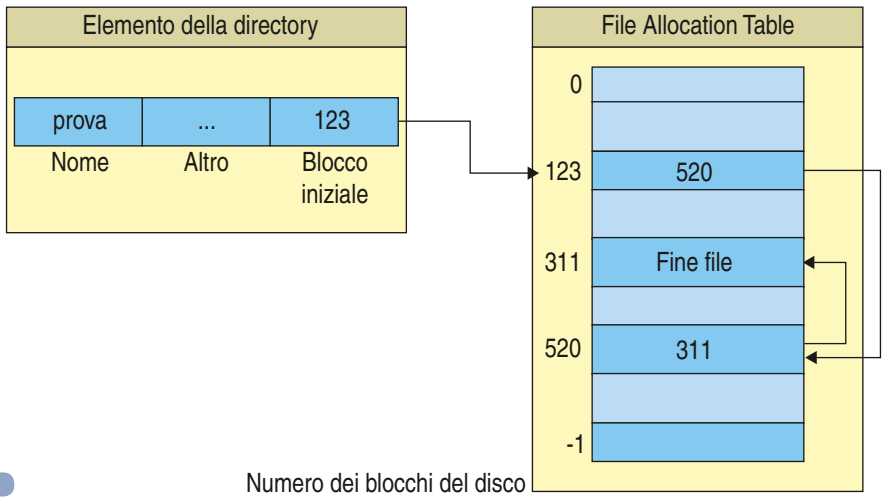
FAT – File Allocation Table

La soluzione migliore è quella che è stata adottata dai sistemi operativi MS-DOS, OS2 e Windows: è un'importante variante dell'allocazione concatenata, e utilizza una tabella chiamata tabella di allocazione dei file (FAT) che viene memorizzata all'inizio di ogni partizione. È una tabella che viene allocata all'inizio del disco ed è composta da tanti elementi (entry) quanti sono i blocchi presenti su disco: un elemento della FAT presente nella directory contiene il numero del primo blocco dove sono presenti i dati, mentre l'elemento della FAT corrispondente a quel blocco contiene il numero del blocco successivo e così via fino all'ultimo elemento che ha un valore speciale di end of file (FFFF), mentre i blocchi liberi hanno invece valore 0.

In altre parole si crea un collegamento a lista non dentro i blocchi ma dentro gli elementi della FAT, e quindi si ottiene un miglioramento del tempo di accesso diretto perché la testina del disco trova velocemente la locazione di qualsiasi blocco che appartiene al file desiderato.

ESEMPIO

Se il blocco i punta al blocco n, allora nella FAT l'i-esima entry contiene il numero n; se l'ultimo blocco del file è il numero j, la j-esima entry della FAT contiene un marker speciale di fine file (hFFFF).



Vantaggi

- La FAT viene mantenuta costantemente in memoria e quindi si riducono i tempi di accesso al file.
- Non ci sono rischi di perdita di file, ma eventualmente si perde un solo blocco perché è possibile ricostruire la sequenza del pezzo di file rimanente o manualmente o utilizzando appositi programmi di utilità presenti in commercio (per esempio PCTools, Norton utilities ecc.).

Svantaggi

- La dimensione della FAT non è costante, ma dipende dalla dimensione del disco in quanto per ogni blocco si deve avere a disposizione una corrispondente entry nella FAT che memorizza il numero di blocco successivo.
- Non appena vengono fatte modifiche alla FAT è necessario salvarla su disco per precauzione, dato che la perdita della FAT provocherebbe la perdita di tutti i dati del disco.

ESEMPIO

Proviamo a calcolare quanto spazio di memoria è necessario riservare per la FAT, per esempio per un disco fisso da 200 GB con 1 kb come dimensione di blocco.
200 GB corrispondono a 214.748.364.800 byte e conoscendo la dimensione del blocco di 1024 byte calcoliamo il numero di blocchi necessari:

$$Numero\ di\ blocchi = 200\ GB / 1024 = 209.715.200$$

Per indicizzare un blocco è necessario un intero a 32 bit, quindi 4 byte: possiamo quindi calcolare lo spazio occupato dalla FAT:

$$209.715.200 \times 4 = 819.200\ Kb = 800\ Mb$$

che corrisponde circa allo 0,4% della dimensione dell'intero disco.

Allocazione indicizzata

◀ Il blocco indice è una memoria aggiuntiva "di servizio", necessaria per gestire il resto della memoria, che viene indicata come overhead di memoria (letteralmente "spesa generale aggiuntiva", "bagaglio aggiuntivo"). ▶

Nell'allocazione indicizzata memorizziamo all'interno di un blocco chiamato ◀ blocco indice ▶ (o *i-node*) tutti i blocchi del file, eliminando il problema di frammentazione esterna e della dichiarazione della dimensione del file.
L'*i*-esimo elemento del blocco indice punta all'*i*-esimo blocco del file, mentre la directory contiene l'indirizzo del blocco indice stesso: all'inizio tutti i puntatori sono impostati a null e man mano che si allocheranno i blocchi, vengono aggiornati.
Per leggere l'*i*-esimo blocco di un file si accede al puntatore che si trova nell'*i*-esimo elemento del blocco indice e quindi si legge il blocco puntato: il tempo è costante indipendentemente dal numero di blocchi e dal blocco prescelto (complessità di calcolo O(1)).
Per modificare l'*i*-esimo blocco di un file si accede al puntatore che si trova nell'*i*-esimo elemento del blocco indice e si modifica il blocco puntato: anche per la scrittura il tempo è costante indipendentemente dal numero di blocchi e dal blocco prescelto (complessità di calcolo O(1)).
Per aggiungere un blocco a un file si deve innanzitutto individuare un blocco libero sul disco e si fa puntare ad esso un elemento che contiene null del blocco indice; quindi si scrive il dato e in questo caso il tempo richiesto è di tipo lineare O(n) rispetto al numero di blocchi.

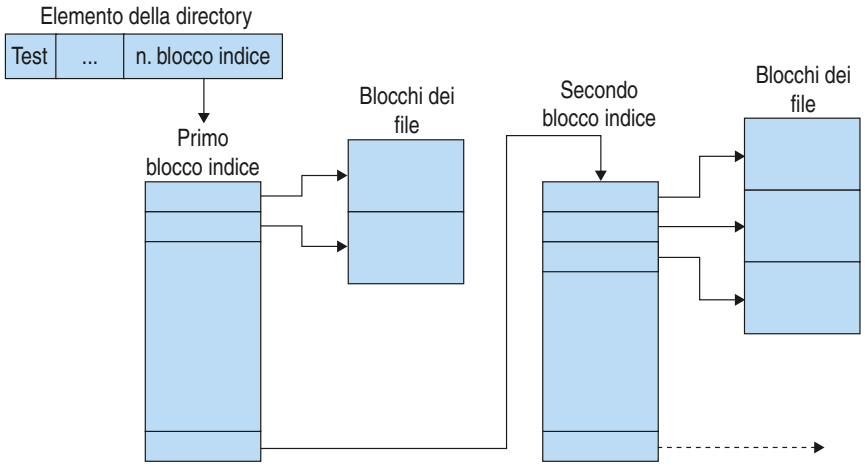
A differenza della FAT, che era unica per ogni partizione, con l'allocazione indicizzata ogni file ha la propria tabella indice e quindi per recuperare tutti i blocchi del file è necessario un solo accesso al blocco indice da cui ricaviamo gli indirizzi di tutti i blocchi memorizzati. L'unico accorgimento è quello di memorizzare tra gli attributi del file nel suo descrittore il blocco indice.

Vantaggi

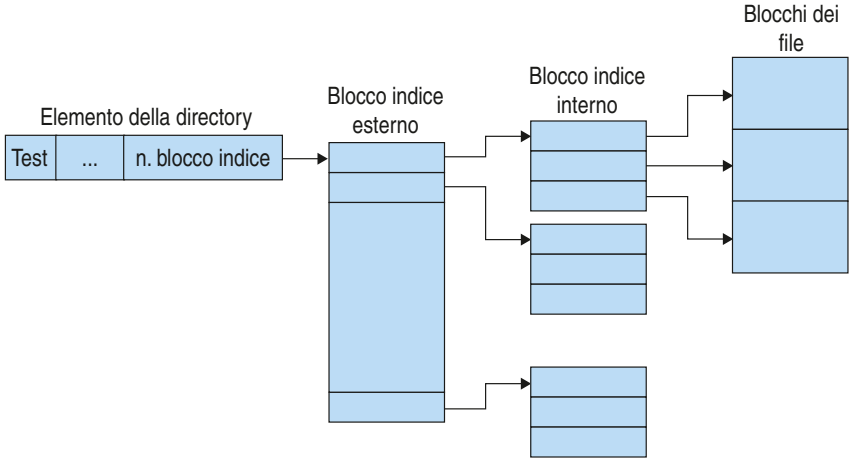
- Non si verifica la frammentazione esterna in quanto non sono necessari blocchi contigui.
- Sia l'accesso diretto sia l'accesso sequenziale sono efficienti.
- Per i file di piccole dimensioni il blocco indice è quasi vuoto.

Svantaggi

- I blocchi indice hanno una dimensione costante quindi ogni blocco indice può gestire file di una dimensione prefissata. I problemi sopraggiungono quando il file è di dimensione maggiore e in questo caso abbiamo due possibili soluzioni:
  – schema concatenato: l'ultima entry del blocco indice punta a un secondo blocco indice;



  – schema a più livelli: il blocco indice contiene solo puntatori ad altri blocchi indice e solo gli elementi dell'ultimo livello dell'indice puntano ai blocchi dei file.



Gestione dello spazio libero

Oltre a memorizzare dove sono salvati i file sul disco rigido, il SO deve tenere traccia di tutti i blocchi liberi del disco fisso, per poterli velocemente mettere a disposizione a fronte di una richiesta da parte dell'utente: viene gestita una lista dello spazio libero:

- in Windows XP queste informazioni si trovano nella *Master File Table* (MFT) che sta alla base dei volumi formattati NFTS (*New Technology File System*);
- in Unix queste informazioni sono racchiuse nel superblocco memorizzato sull'HD a partire dal blocco numero 1.

Ogni operazione di aggiornamento file, cioè inserimento, cancellazione e modifica, utilizza questa struttura e ne aggiorna di conseguenza il contenuto.

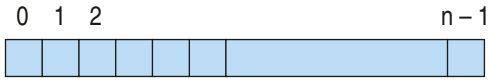
Può essere implementata secondo diverse tecniche illustrate qui di seguito.

Vettore di bit

Per ogni blocco usiamo un bit per indicare se è libero o occupato realizzando una mappa di bit (*bit map*) o vettore di bit (*bit vector*), dove abbiamo per il blocco i-esimo:

- bit[i] = 1 ⇒ blocco i-esimo libero;
- bit[i] = 0 ⇒ blocco i-esimo occupato.

Questa tecnica è estremamente semplice da realizzare e da utilizzare ed è molto efficiente nel trovare il primo blocco libero o n blocchi liberi consecutivi, ma il vettore deve essere tenuto costantemente in memoria centrale e salvato periodicamente su disco.

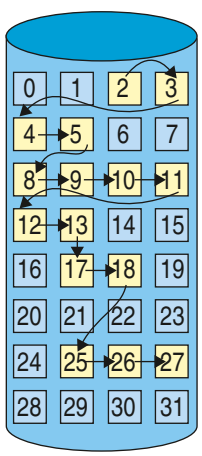


Per ridurre l'occupazione del vettore di bit è possibile usare un bit per dei cluster di blocchi, come avviene in MacOS.

Lista collegata

Può essere mantenuta una lista concatenata di blocchi liberi mediante un sistema di puntatori, così come vengono organizzati i file ad allocazione concatenata, mantenendo un puntatore al primo blocco in una locazione speciale del disco o in una memoria cache.

Nell'esempio della figura a lato il puntatore al primo blocco contiene il valore 2.
In questo modo non abbiamo alcuno spreco di spazio ed è immediato il recupero di un blocco libero; come alternativa si può usare una variante della FAT per gestire i blocchi liberi così da ridurre i tempi per cercare i blocchi liberi contigui.



Raggruppamento

È possibile raggruppare nel primo blocco libero i puntatori a n blocchi liberi. L'ultimo puntatore di questi n blocchi segnalati punta a un altro blocco di blocchi liberi come nell'allocazione indicizzata e così via.
A differenza della lista collegata standard, questo sistema consente di trovare più rapidamente molti blocchi disponibili contemporaneamente.

Conteggio

È un'estensione del metodo a lista, dove viene tenuto in memoria, oltre all'indice del primo blocco libero, anche un contatore che indica il numero di blocchi liberi contigui. In questo caso la lista dei blocchi che contengono i puntatori ai blocchi liberi è sicuramente più corta ma richiede maggiore memoria perché per ogni sequenza sono memorizzate più informazioni.

Realizzazione del file system

Realizzazione

Partiamo dal momento in cui viene aggiunto un disco alla macchina.

La *prima operazione* da fare è verificare se il disco è già stato formattato a basso livello creando cioè l'organizzazione a tracce e settori sulla superficie magnetica individuando le eventuali zone danneggiate presenti su di esso e "marchiandole" come bad cluster, cioè inutilizzabili successivamente dai sistemi operativi che verranno installati su quel supporto.

> A ogni cluster è associato un indirizzo fisico che è composto da *numero di cilindro*, *numero di traccia* all'interno del cilindro e *numero di settore* all'interno della traccia.

La *seconda operazione* consiste nella definizione del numero di volumi e delle partizioni desiderate per ogni volume: la creazione di *volumi logici* nella memoria di massa è un'operazione obbligatoria anche se non si intende suddividere ulteriormente il supporto, e nel caso di più partizioni deve essere indicata quale sarà la partizione principale (*primaria*).

Come *terza operazione* si procede alla formattazione ad alto livello: dato che ogni sistema operativo è strettamente legato a un tipo di file system, è necessario prima stabilire per ogni partizione quale sistema operativo (o versione) si intende installare e quindi si procede alla formattazione ad alto livello.

Windows indica ogni partizione con lettere seguite dai *due punti*, come nell'esempio della figura seguente, dove abbiamo creato in un volume due partizioni con file system diverso:

Disco fisico	Partizione	File system	Lettera del drive
Disco fisso 1	Partizione 1	NTFS	C:
	Partizione 2	FAT32	D:

Abbiamo cioè predisposto la tipologia di file system prima dell'installazione del sistema operativo in quanto, ovviamente, per poter installare i file del sistema operativo il disco deve essere in grado di memorizzarli.

Zoom su...

PARTIZIONI GREZZE

È possibile anche lasciare delle partizioni grezze, dove cioè non si vuole utilizzare un particolare file system ma si intende riservare quello spazio di disco per utilizzi o applicazioni particolari che lo gestiscono autonomamente, come per esempio un'area di swap, o un'area di memoria per un database, o ancora informazioni per dischi RAID ecc.

Mantenimento

Sappiamo che le informazioni sull'avvio del sistema sono mantenute nel primo settore del disco in una partizione separata chiamata master boot sector (MBR), e questo è il settore che il BIOS legge all'accensione della macchina. Le informazioni sul partizionamento di un hard disk si trovano nel MBR, così come le informazioni necessarie per caricare il sistema operativo, o più di uno se c'è un loader che permette il dual-boot.

La partizione primaria o partizione radice (*root partition*) contiene il kernel del sistema operativo e viene montata anch'essa all'avvio mentre le altre partizioni possono essere montate in seguito sia in modo automatico sia manualmente dall'utente al momento del loro utilizzo. A ogni avviamento del sistema viene sempre fatto un controllo di correttezza del file system per verificare la coerenza della partizione ed eventualmente intervenire correggendo le anomalie riscontrate.

Abbiamo detto che per gestire i file sono presenti molte strutture dati che possono variare da sistema operativo a sistema operativo, ma sempre suddivise in due gruppi: una parte è memorizzata su disco e una seconda deve essere sempre presente in memoria dinamica e quindi caricata in RAM al momento del boot di sistema.
È necessario che queste tabelle vengano periodicamente salvate sul disco in quanto un'improvvisa mancanza di alimentazione potrebbe far perdere tutte queste informazioni e ci troveremmo di fronte a un problema di perdita di coerenza tra le informazioni memorizzate sul supporto e le tabelle che ne descrivono i contenuti.

Un processo del file system, il controllore della coerenza, ha il compito fondamentale di confrontare i dati nelle strutture delle directory e i blocchi di dati su disco, cercando di ripristinare la situazione nel caso vengano individuate anomalie.

È comunque consigliabile l'utilizzo di gruppi di continuità (detti UPS, dall'inglese Uninterruptible Power Supply) che, grazie alle loro batterie tampone, sono in grado di sostituirsi alla rete di alimentazione in caso di un improvviso blackout.

Concludiamo ricapitolando di seguito le operazioni che vengono fatte dal sistema operativo nelle tre fondamentali richieste di operazioni su file.

- Creazione di un file: per creare un nuovo file il file system logico come prima operazione assegna un descrittore e quindi un suo identificatore, aggiorna la directory dove questo file verrà memorizzato caricandola da disco e aggiungi i dati del nuovo file. A questo punto si può iniziare a memorizzare le informazioni.
- Utilizzazione di un file: per poter effettuare qualunque operazione su un file questo deve essere aperto, cioè deve essere presente il suo descrittore in memoria centrale: se non è presente, allora il file system ricerca su disco il descrittore del file e lo carica nella tabella dei file aperti aggiornando un puntatore a questo riferimento nella tabella del processo che ha richiesto l'utilizzo del file. Nel descrittore del file è presente un contatore dei processi che lo stanno utilizzando, che viene quindi incrementato.
- Termine utilizzo: anche la cessazione di utilizzo di un file comporta una sequenza di operazioni da parte del SO: il contatore dei processi che stanno utilizzando il file, che è presente all'interno del descrittore del file stesso memorizzato nella tabella dei file aperti, si decrementa: se il suo valore raggiunge lo 0 significa che nessun processo ha più bisogno di quel file, che quindi può essere chiuso e il suo descrittore può essere rimosso dalla tabella generale dei file aperti.

Verifichiamo le conoscenze

Esercizi a scelta multipla

1 Indica quali strutture sono su disco (D) e quali in RAM (R):

__ tabella delle partizioni
__ *boot control block* (blocco di controllo del boot)
__ tabella dei file aperti di ciascun processo
__ *partition control block* (blocco di controllo della partizione)
__ strutture di directory cui si è più recentemente acceduto
__ strutture di directory
__ descrittori di file
__ tabella dei file aperti

2 L'unità più piccola su un disco fisso è:

○ il settore
○ il cluster
○ il blocco
○ la traccia

3 Ordina i livelli di file system dal più basso al più alto:

__ modulo di organizzazione dei file
__ controllo I/O
__ applicazioni dell'utente
__ file system di base
__ dispositivi fisici
__ file system logico

4 Quale di queste espressioni è vera?

○ allocazione contigua
○ allocazione concatenata
○ allocazione diretta
○ allocazione indicizzata

5 L'allocazione concatenata (indicare l'affermazione errata):

○ non ha frammentazione
○ nel descrittore ha memorizzato il blocco iniziale
○ ha l'accesso più lento rispetto all'allocazione contigua
○ ha memorizzato nel descrittore il numero del blocco finale

6 L'acronimo FAT significa:

○ *File Allocation Table*
○ *File Access Table*
○ *File Allocation Text*
○ *File Access Text*

7 In un entry della FAT è memorizzato:

○ il blocco corrente da leggere
○ il numero di blocco successivo
○ il numero di blocco appena letto
○ il riferimento all'ultimo blocco del file

8 Con UPS si intende:

○ *Universal Power Supply*
○ *Uninterruptible Power System*
○ *Unix Power Supply*
○ *Uninterruptible Power Supply*

Test vero/falso

1 Il settore generalmente ha una dimensione di 512 o 1024 byte. V F
2 L'allocazione contigua permette di accedere direttamente a un blocco. V F
3 L'allocazione concatenata memorizza nel descrittore il numero del blocco iniziale. V F
4 La soluzione FAT è tipica del sistema operativo Unix/Linux. V F
5 La soluzione FAT è una variante dell'allocazione concatenata. V F
6 Con il termine *overhead* si intende una memoria aggiuntiva di servizio "sprecata". V F
7 Nell'allocazione indicizzata c'è un apposito indice per ogni file. V F
8 La Master File Table serve per gestire i blocchi liberi. V F

UNITÀ DIDATTICA 7

LA SICUREZZA DEL FILE SYSTEM

IN QUESTA UNITÀ IMPAREREMO...

- le tecniche di back-up dei dati
- i sistemi di protezione dei dati

La sicurezza del file system

I dati memorizzati sui dischi non sono mai sicuri, sia perché gli utenti possono accidentalmente effettuare delle errate memorizzazioni o cancellazioni sia perché nessun dispositivo elettronico e meccanico è esente da guasti o malfunzionamenti.
È necessario effettuare periodicamente dei salvataggi di sicurezza (copie di back-up) dei dati su altri dispositivi, generalmente esterni, in modo da poter ripristinare situazioni preesistenti andate perse o danneggiate.

Esistono in commercio appositi programmi che effettuano i back-up offrendo all'amministratore del sistema tutte le opzioni possibili con numerosi parametri personalizzabili:

- utilità che permettono di pianificare il salvataggio:
  - da eseguirsi manualmente;
  - da eseguirsi periodicamente in automatico:
    - all'avvio o all'arresto del sistema operativo;
    - a ogni ora o giornalmente a una data ora;
- parametri di tipo "includere/escludere" file o cartelle;
- il back-up e ripristino di tutti i dati contenuti su singoli server e stazioni di lavoro;
- il back-up di file di un tipo specifico localizzati su un server o su particolari stazioni di lavoro:
  - file finanziari;
  - file di produttività;
  - immagini;
  - progetti;
  - ecc.

- la sincronizzazione dei file tra numerose stazioni di lavoro, server e computer portatili;
- la sincronizzazione di file tra server di back-up, consentendo la creazione di back-up ridondanti;
- la crittografia con sistemi di cifratura per proteggere informazioni sensibili;
- il back-up di file aperti/in uso;
- la compressione del back-up.

Inoltre, mentre eseguono il salvataggio i sistemi di back-up effettuano contemporaneamente i controlli di integrità dei dati e di ridondanza ciclici (CRC) su tutti i file per proteggerli da modifiche involontarie dovute a errori di comunicazione.

I back-up possono essere completi o incrementali:
- completo: si esegue il back-up di tutto il sistema;
- incrementale: si esegue il back-up solo dei file che sono stati salvati dopo l'ultimo back-up effettuato.

Zoom su...

LEGGE 196/03

La legge 196/2003 oltre alla privacy ha introdotto alcune norme sul trattamento dei dati a livello informatico. In particolare tra le misure minime da adottare prevede l'adozione di procedure per la custodia di copie di sicurezza, il ripristino della disponibilità dei dati e dei sistemi.
La normativa sulla sicurezza dei dati prevede che i dati sensibili debbano essere salvati con frequenza almeno settimanale (all. B, art. 18) in un luogo sicuro e in caso di disastro informatico deve essere possibile il loro recupero entro sette giorni (art. 23).
Quindi le operazioni da adottare per il salvataggio dei dati sono obbligatorie.

È necessario effettuare periodicamente (settimanalmente) un back-up completo su supporti esterni diversi, cioè a rotazione di settimana in settimana, in modo da avere per esempio sempre quattro copie diverse dei dati, oltre al back-up incrementale giornaliero, per salvare ogni giorno le modifiche effettuate.

Storicamente i back-up venivano effettuati su memorie magnetiche di tipo sequenziale, i DAT (Digital Audio Tape), così chiamati perché venivano utilizzati come supporti a normali audiocassette. Oggi le audiocassette sono "sparite", ma i sistemi a nastro sono ancora utilizzati.

La struttura RAID

Come abbiamo detto, i dischi magnetici hanno al loro interno parti meccaniche in movimento che con gli anni tendono a usurarsi, che potrebbero manifestare durante l'uso difetti di fabbricazione inizialmente non evidenti e che in ogni caso sono comunque soggetti a possibili guasti. La riduzione dei costi ha portato a sviluppare tecniche di protezione

dei dati basate sulla duplicazione fisica dei dispositivi: la tecnologia RAID (acronimo di Redundant Array of Inexpensive Disk, che in italiano può essere tradotto con "vettore ridondante di dischi economici").
L'idea di base si fonda proprio sul basso costo dell'hardware rispetto al valore dei dati che contiene e su analisi statistiche sulla probabilità che due o più dischi si possano rompere contemporaneamente.

ESEMPIO

Per esempio, se ho un solo disco e questo si rompe, ho un fault del 100% e il mio sistema è bloccato, ma se ho sei dischi e se ne rompe uno ho un fault del 16,66% e il sistema continua a funzionare: inoltre se due dischi hanno gli stessi dati in copia, non viene persa alcuna informazione.

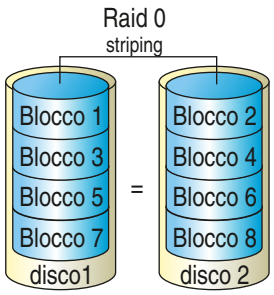
La prime applicazioni della tecnica RAID sono state utilizzate nei server di rete per effettuare copie di salvataggio dei dati, ma sono state applicate anche a singoli calcolatori e oggi trovano largo impiego anche nei PC domestici.

I primi sistemi di protezione consistevano nel fare copie speculari del disco sul disco stesso, creando due dischi logici sullo stesso disco fisico e copiando in entrambi i volumi le stesse informazioni.
Questo primo meccanismo era un valido sistema solo per rimediare alla perdita accidentale dei dati, ma non dava alcuna protezione in caso di rottura del disco.

Un passo successivo è stato quello di utilizzare una tecnica chiamata disk mirroring, che consiste nell'aggiungere un secondo disco fisico connesso allo stesso controller: ogni scrittura deve avvenire su entrambi, possibilmente in tempi diversi come precauzione contro l'incoerenza.
Il mirroring dei dati aumenta l'affidabilità a scapito delle prestazioni dato che le operazioni per unità di tempo sono raddoppiate.

Il passo successivo è stato quello fatto con il disk duplexing, introducendo una duplicazione completa dell'hardware, facendo quindi una copia di ciascuna partizione su un altro disco fisico connesso a un altro controller.

I livelli di RAID



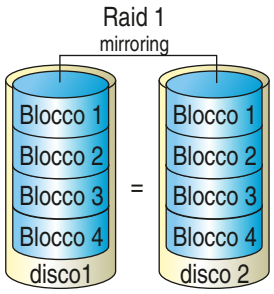
Esistono diverse configurazioni di dischi che cercano questo o quel compromesso tra il fattore affidabilità e quello delle prestazioni, e prendono il nome di livelli di RAID. Vediamoli brevemente.

RAID livello 0

Il sistema RAID-0 non è un vero sistema RAID in quanto non prevede ridondanza dei dati: consiste nell'utilizzo di almeno due dischi e utilizza un'operazione detta di striping per velocizzare l'equa distribuzione dei dati tra di essi, che sono visti dal sistema operativo come un'unica unità. Le prestazioni sono molto alte ma il sistema non offre alcuna protezione contro i malfunzionamenti.

RAID livello 1

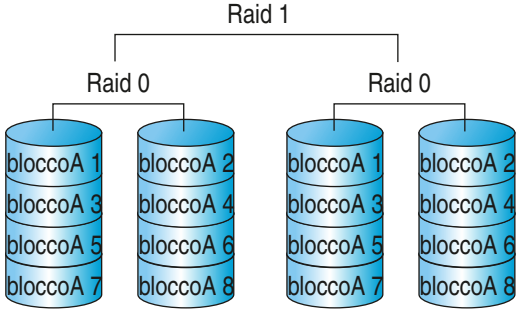
I sistemi RAID-1 sfruttano la tecnologia mirroring per il salvataggio dei dati senza interruzioni: sono costituiti da un numero pari di dischi nei quali vengono replicate le informazioni: il disco 2 replica il disco 1 oppure i dischi 3 e 4 replicano i dischi 1 e 2 ecc.
La capacità massima di memorizzazione è pari a quella del drive più piccolo presente nel sistema stesso, dato che ogni informazione deve essere duplicata.
Il sistema RAID-1 aumenta anche le prestazioni dato che molte configurazioni permettono di mettere in parallelo le operazioni di I/O leggendo da un'unità mentre l'altra è occupata.



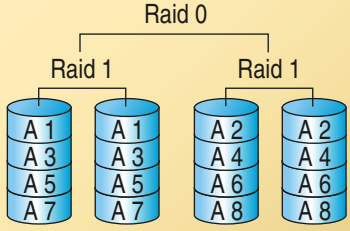
Nei sistemi RAID-1 i dati si perdono solamente per rottura di tutte le unità presenti nel sistema, caso piuttosto raro dovuto a un evento accidentale tipo un urto o una caduta, oppure ambientale come un allagamento o un incendio.

RAID livello 0+1

Questo sistema combina le tecniche utilizzate nei due livelli già descritti ottenendo i massimi benefici.
Si utilizzano due serie di dischi in bit-level striping messi in mirroring tra loro.

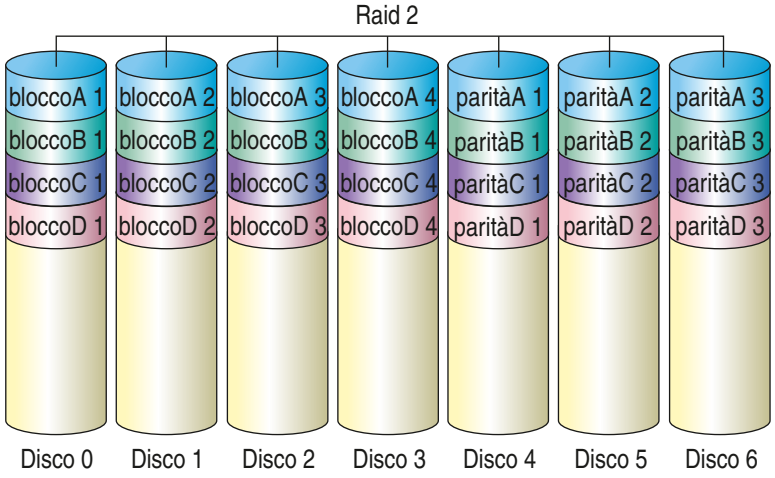


Esiste anche il livello 1+0, dove i due livelli sono scambiati: offre elevata affidabilità con alte prestazioni e può tollerare il guasto di due dischi in mirror diversi.



RAID livello 2

I sistemi RAID-2 introducono, oltre alla duplicazione dei dischi, anche sistemi aggiuntivi di codici per la correzione degli errori (Error Correction Code, ECC) con bit supplementari mantenuti su dischi separati.
Per un sistema con quattro dischi per le informazioni, ne sono necessari altri tre per l'ECC ottenuto mediante i codici di Hamming.



Zoom su...

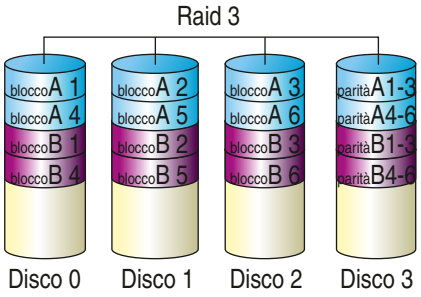
CODICI DI HAMMING

Il codice di Hamming è un sistema semplice di rilevazione e correzione degli errori: per esempio in una codifica di Hamming (7,4) tre cifre di parità sono combinazioni diverse delle quattro cifre d'informazione, seguendo lo schema riportato a lato. Si può facilmente verificare che la distanza minima di Hamming ottenuta con queste formule è d = 3 e quindi il codice può rivelare tutti gli errori semplici e doppi ed è in grado di correggere tutti gli errori semplici.

$$\begin{cases} p_1 = i_2 + i_3 + i_4 \\ p_2 = i_1 + i_3 + i_4 \\ p_3 = i_1 + i_2 + i_4 \end{cases}$$

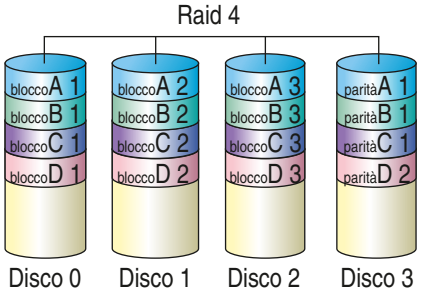
RAID livello 3

Il sistema RAID-3 prende anche il nome di *organizzazione di parità a bit alternati* ed è organizzato con quattro dischi, dove ogni file è diviso a gruppi di byte tra le prime tre unità mentre il quarto disco è di parità. Migliora il livello 2 dato che è sufficiente un unico disco supplementare e a parità di efficacia si possono migliorare le prestazioni introducendo un controller hardware dedicato alla parità e con una memoria cache di servizio.



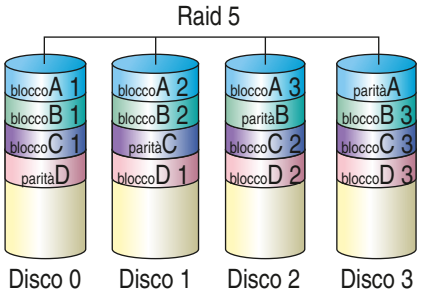
RAID livello 4

È chiamato anche di *organizzazione di parità a blocchi alternati* ed è molto simile al livello 3, ma con un unico disco di parità ed effettua una suddivisione fra i tre dischi a livello di blocco: se si rovina un blocco, grazie al blocco di parità e al contenuto degli altri dischi è possibile ripristinarlo.
La velocità generale di I/O è elevata in quanto è possibile effettuare il parallelismo delle operazioni di lettura/scrittura.



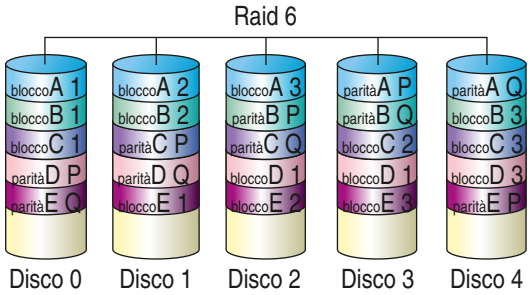
RAID livello 5

Il sistema RAID-5 prende anche il nome di *parità distribuita a blocchi alternati* perché distribuisce i dati a blocchi e le parità a livello di bit tra tutti i dischi invece che mantenere queste ultime in un disco separato. Unisce i vantaggi del RAID-3 e del RAID-4 evitando i colli di bottiglia dovuti al dover effettuare letture contemporanee da più dischi per ottenere la parità.



RAID livello 6

Si chiama anche *schema di ridondanza P+Q* ed è simile al RAID-5 ma molto costoso perché aumenta il numero di dischi in quanto introdu-



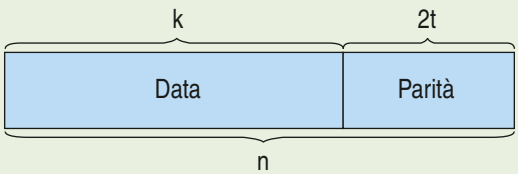
ce un doppio calcolo di parità utilizzando codici correttori di Reed-Solomon a livello di bit; offre però la massima sicurezza.

Nella scelta del tipo di RAID da utilizzare è necessario tenere presente innanzitutto le necessità che si hanno nel sistema:
- nel caso in cui il parametro richieda soltanto le prestazioni senza criticità sui dati si adotta il RAID-0;
- quando viene richiesta alta affidabilità e un sistema in cui sia possibile effettuare il recupero dei dati molto velocemente in caso di perdita, la soluzione migliore è il RAID-1;
- un ottimo compromesso lo si ottiene dal RAID 0+1 o dal RAID-5.

Zoom su...

REED-SOLOMON

I codici di Reed-Solomon sono codici a correzione d'errore basati sui codici a blocco: oltre che nei RAID-6 hanno una grande applicazione nei sistemi di trasmissione, nelle comunicazioni satellitari e nei modem xDSL.



Consideriamo simboli di s bit: un codificatore di Reed-Solomon suddivide i dati in gruppi di k bit e a ogni blocco aggiunge 2t simboli per formare una parola di codice di n bit. Dato un simbolo di dimensione s, la parola di codice può essere al massimo lunga $n = 2^s - 1$.
Un decodificatore Reed-Solomon può correggere fino a t simboli che contengono errori, con $2t = n - k$, come si può vedere dalla figura.

Struttura di memoria terziaria

Con memoria terziaria si intendono i sistemi di memorizzazione rimovibili (dischetti, CD-ROM, DVD ecc.) che possono comunicare con il calcolatore. La caratteristica principale di una memorizzazione terziaria è che deve avere un basso costo, e può essere classificata come descritto di seguito.

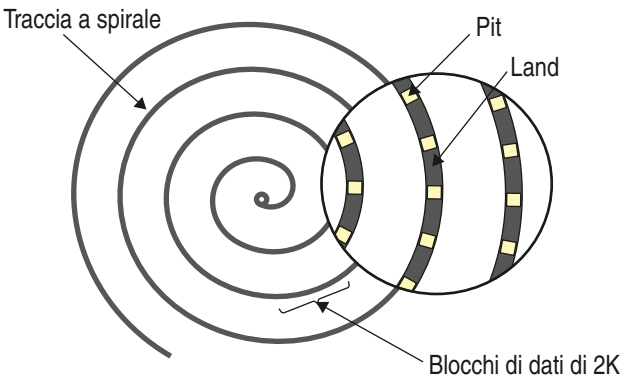
Dischi rimovibili

Ne abbiamo di diversi tipi:
- dischi magnetici: i floppy disk sono stati ormai sostituiti da dischi esterni connessi mediante porte USB, caratterizzati da alta velocità di accesso ma con un alto rischio di danneggiamenti;
- dischi ottici.

CD

Lo standard dei CD-ROM è stato originariamente concepito per i CD audio dalla Sony, nel 1982; ricordiamo le principali caratteristiche tecnologiche:
- l'incisione della traccia avviene a spirale da 22.000 a 600 giri/mm (lunghezza totale circa 5,6 km);

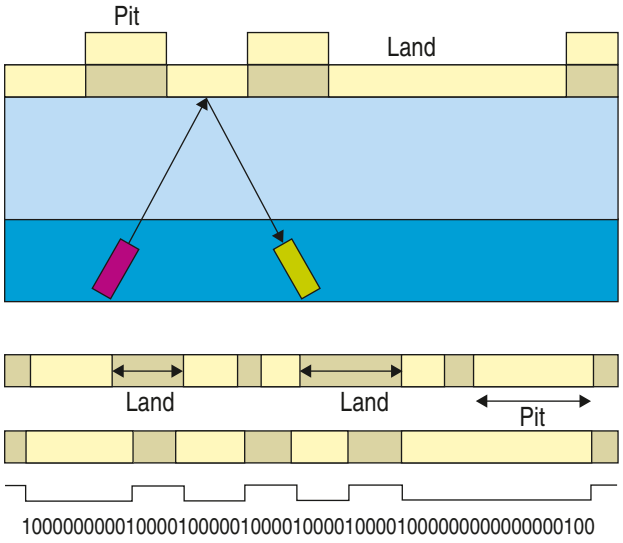
- la registrazione dei dati si basa su pit e land:
  - pit = area che viene bruciata dal laser (0,8 micron di diametro e 0,2 di profondità);
  - land = area di separazione tra due pit;
- la velocità di rotazione è tra 200 e 520 RPM (rotazioni per minuto).



Il CD vergine è trasparente: il laser ad alta potenza (8-16 mW) crea una macchia scura tra le superfici riflettenti che, in fase di lettura, verrà interpretata come pit/land.

Nei CD-R è possibile memorizzare i dati una sola volta, in quanto la scrittura è distruttiva: lettura e scrittura avvengono mediante due diverse intensità di laser:

- alta (scrive): brucia delle areole nello strato colorato;
- bassa (legge): uguale ai CD-ROM, cioè il raggio laser viene riflesso in controfase nei pit e quindi può essere rilevato come segnale negativo.



Non bisogna ovviamente identificare il pit con 0 e il land con 1: il valore digitale 1 viene associato al passaggio tra pit e land oppure tra land e pit, mentre l'assenza di un cambiamento è rappresentata da uno 0.

I CD esistono anche nella versione riscrivibile CD-RW: è come un CD-R in cui lo strato riflettente è costituito da una lega di argento, indio, antimonio e tellurio; la lega ha due stati di stabilità, cristallino e amorfo, con due diverse caratteristiche di riflessione.

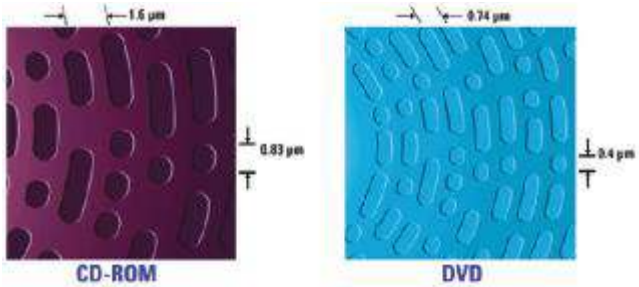
In questo caso, per avere la riscrittura, si utilizzano tre potenze diverse del laser:

- alta (scrive): la lega viene "fusa" e quindi passa dallo stato *cristallino* (stato riflettente - land) a quello di *amorfo* (opaco - pit);
- media (cancella): viceversa dallo stato *amorfo* la lega passa allo stato *cristallino*;
- bassa (legge): uguale ai CD-ROM, cioè il laser non provoca alcuna transizione di stato sul materiale.

DVD

DVD è l'acronimo di Digital Versatile Disc (cioè Disco Versatile Digitale) ma agli inizi della sua diffusione si prestava meglio l'interpretazione Digital Video Disc, cioè Disco Video Digi-

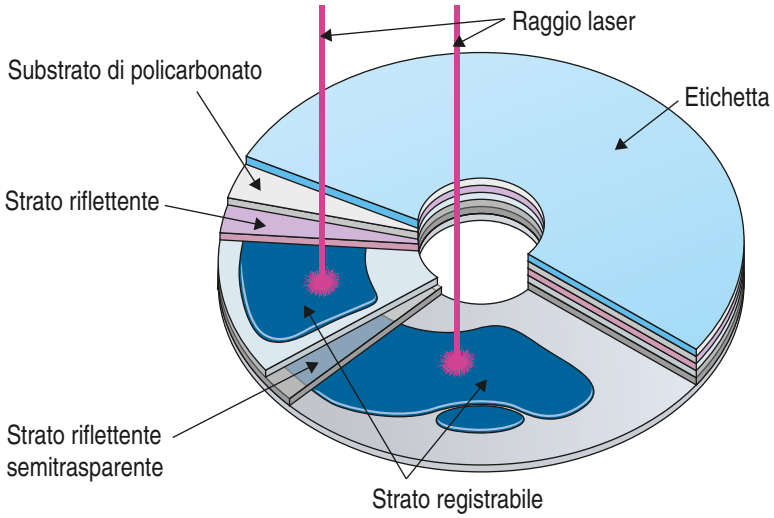
tale, dato che la sua prima applicazione era quella di supporto per video e solo in un secondo tempo è stato utilizzato come memoria digitale.
I DVD-R e i DVD-RW sono più sicuri ma più lenti di quelli magnetici e richiedono l'intervento umano per la sostituzione dei dischi che hanno dimensioni modeste: hanno le stesse dimensioni dei CD-R e sfruttano lo stesso principio (pit/land) per la scrittura/lettura.



Vantano però un aumento significativo di capacità, in quanto utilizzano una lunghezza d'onda del laser a $\lambda = 0{,}65\ \mu$ in modo da ottenere pit con dimensione dimezzata rispetto ai CD (pit = 0,4 µ invece di 0,83 µ).
Con il laser "più sottile" si riesce anche a ottenere una spirale più stretta: 0,74 µ (prima era 1,6 µ) in modo da memorizzare su un solo disco 133 minuti di film con risoluzione 720 × 480 pixel, compresso nel formato MPEG-2.

Tecnologicamente i primi DVD sono stati realizzati con un singolo strato di policarbonato (layer singolo), mentre gli attuali hanno il doppio strato (layer doppio) ottenendo le seguenti capacità di memorizzazione:

- singola faccia, layer doppio: 8,5 GB;
- doppia faccia, layer singolo: 9 GB;
- doppia faccia, layer doppio: 17 GB.



I nastri

I nastri sono il sistema più economico e veloce per effettuare il salvataggio di grosse masse di dati e sono utilizzati per i back-up completi dei sistemi quando è richiesto solo l'accesso sequenziale e non quello diretto.

Tecnologie future

Ricordiamo sinteticamente le due principali:

- memorizzazione olografica: utilizza la luce laser per registrare fotografie olografiche di grosse dimensioni su supporti speciali;
- sistemi meccanici microelettronici (MEMS): prevedono la creazione di circuiti integrati per memorizzare in modo non volatile i dati più velocemente che con i dischi magnetici.

Verifichiamo le conoscenze

Esercizi a scelta multipla

1 I back-up possono essere:
- ○ completi
- ○ parziali
- ○ incrementali
- ○ temporanei

2 I sistemi di back-up effettuano controlli di:
- ○ integrità dei dati
- ○ ridondanza ciclici
- ○ coerenza dei dati
- ○ duplicazione

3 La legge 196/03 in materia di privacy:
- ○ dà indicazioni su come effettuare il back-up
- ○ dà indicazioni minime sulle misure da adottare
- ○ impone il salvataggio giornaliero dei dati
- ○ indica che il ripristino dei dati deve essere fatto entro 24 ore

4 Il disk mirroring:
- ○ aggiunge un secondo disco allo stesso controller
- ○ prevede che la scrittura avvenga su entrambi in tempi diversi
- ○ aumenta l'affidabilità
- ○ aumenta le prestazioni
- ○ raddoppia le operazioni per unità di tempo

5 L'acronimo RAID è relativo a:
- ○ *Replicate Array of Inexpensive Disk*
- ○ *Redundant Array of Indexed Disk*
- ○ *Replicate Array of Indexed Disk*
- ○ *Redundant Array of Inexpensive Disk*

6 Quale tra queste affermazioni è falsa?
- ○ il RAID-0 non è un vero sistema RAID
- ○ il RAID-1 sfrutta la tecnologia mirroring
- ○ il RAID-2 introduce l'ECC
- ○ il RAID-3 prende anche il nome di organizzazione di parità a bit alternati
- ○ il RAID-4 prende anche il nome di organizzazione di parità a blocchi multipli
- ○ il RAID-5 prende anche il nome di parità distribuita a blocchi alternati
- ○ il RAID-6 prende anche il nome di schema di ridondanza P+Q

Test vero/falso

1	La perdita di coerenza è dovuta sostanzialmente a blackout elettrici.	V F
2	Nei back-up incrementali viene salvato solo il file qualora sia aumentata la sua dimensione.	V F
3	La legge 196/03 impone degli obblighi in merito al salvataggio dei dati.	V F
4	Nel disk duplexing, più controller possono leggere sullo stesso disco.	V F
5	Nel disk mirroring le coppie di dischi hanno lo stesso controller.	V F
6	Nel sistema RAID-0 la ridondanza è effettuata su due dischi.	V F
7	Lo striping è una tecnica che velocizza la copia in duplexing.	V F
8	I dischi ottici sono utilizzati come memorie terziarie.	V F
9	Nei CD-R sono presenti due strati di policarbonato.	V F
10	Il pit corrisponde allo stato amorfo, non riflettente.	V F

UNITÀ DIDATTICA 8

LA GESTIONE DELLA I/O

IN QUESTA UNITÀ IMPAREREMO...

- a conoscere l'hardware dei dispositivi di I/O
- come avviene il trasferimento dei dati
- le tecniche di gestione delle periferiche

Introduzione

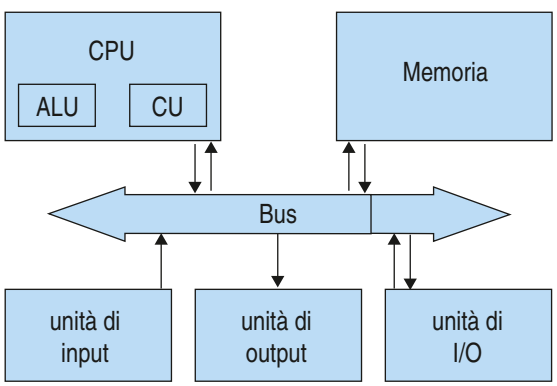
Ogni elaborazione eseguita da un calcolatore deve comunicare i risultati al suo esterno, con l'uomo o con altri calcolatori, e quindi necessita di meccanismi che permettano di interfacciarsi con il mondo esterno: questi sono i dispositivi di I/O, cioè di Input/Output. Il compito della gestione e del controllo di tutte le operazioni e dei dispositivi di I/O è una delle funzioni principali del sistema operativo.

SISTEMA OPERATIVO

Il sistema operativo è l'interfaccia tra l'hardware e i programmi che effettuano richieste di I/O.

Esiste un'incredibile varietà di dispositivi di I/O e inoltre la dinamicità dell'evoluzione tecnologica propone costantemente nuove tipologie di periferiche.

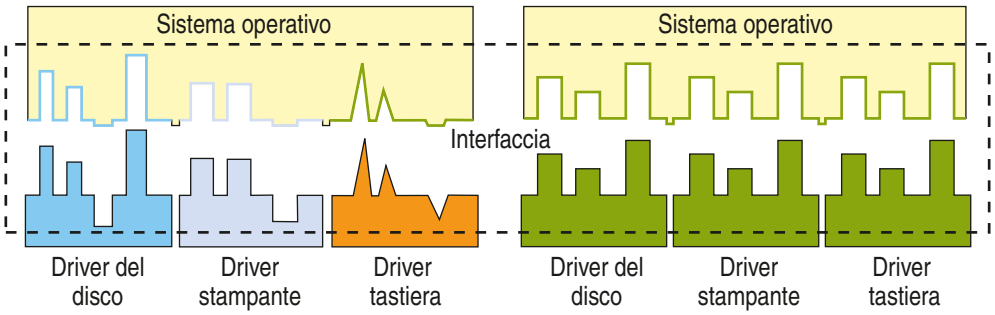
Per poter effettuare il corretto controllo dei dispositivi di I/O sono necessari diversi sistemi di gestione che costituiscono il sottosistema di I/O del kernel, strutturato in moduli chiamati driver, che sono presenti uno per ogni dispositivo.



I driver offrono dei meccanismi che permettono un trattamento uniforme di ogni dispositivo di I/O da parte delle applicazioni e del sistema operativo, cioè rendono uguali le modalità con cui i processi vedono i dispositivi e in cui i dispositivi sono gestiti dal SO.

Non è facile ottenere questa uniformità a causa della diversa natura e delle diverse modalità di funzionamento dei dispositivi, ma si cerca di usare un approccio modulare in modo da:
- nascondere nelle routine di basso livello gran parte dei dettagli dei dispositivi;
- distinguere i dispositivi fisici, cui sono associate le routine di basso livello, dai dispositivi logici (canali), cui sono associate funzionalità generali di alto livello (open, read, write, close);
- associare un dispositivo fisico a uno logico;
- consentire ai processi di interagire con i dispositivi logici.

Il raggiungimento di quest'ultimo obiettivo viene facilitato se anche l'interfaccia dei driver è standardizzata, e quindi gli scrittori dei driver di ogni specifica periferica sono tenuti a rispettare gli standard di interfacciamento imposti dal sistema operativo.



Zoom su...

DRIVER

Un driver è un programma che consente al sistema operativo di scambiare informazioni con i componenti hardware o con dispositivi esterni collegati al calcolatore: un insieme di driver per le periferiche standard viene fornito con il sistema operativo stesso e viene installato contemporaneamente ad esso; inoltre ogni produttore di periferiche solitamente allega un CD con i driver specifici e ottimizzati per il proprio prodotto e mette a disposizione anche una pagina web dove scaricare i programmi e gli aggiornamenti dei driver per le nuove versioni dei sistemi operativi.

Il sistema operativo classifica i dispositivi di I/O in:
- dispositivi di ingresso e puntamento: tastiera, joystick, penna ottica, mouse, touchpad, mouse ecc.;
- dispositivi di memorizzazione: dischi rigidi, nastri, CD-ROM, DVD ecc.;
- dispositivi di collegamento: scheda di rete, radio, linea seriale;
- dispositivi multimediali: videocam, microfono, casse ecc.

L'utilizzatore classifica i dispositivi di I/O prendendo come riferimento una loro particolare caratteristica:

- modalità di trasferimento: a blocchi o a caratteri;
- modalità di accesso: sequenziale o random;
- tipologia di trasferimento: sincrono o asincrono;
- condivisione: dedicata o condivisibile;
- velocità: per esempio tastiera 80 b/s, CD-ROM 6 Mb/s, video display 1 Gb/s;
- direzione di I/O: lettura, scrittura, entrambe.

Nella modalità di trasferimento a blocchi i dati sono letti/scritti tipicamente in blocchi di 512-1024 byte, mentre nel caso di comunicazione a caratteri i dati vengono letti/scritti un carattere alla volta.

Nella tabella seguente riassumiamo le modalità di classificazione riportando per ciascuna di esse un dispositivo come esempio.

Aspetto	Variazioni	Esempi
Modalità di trasferimento	Caratteri Blocchi	Terminale Dischi
Modalità di accesso	Sequenziale Random	Modem CD-ROM
Tipologia di trasferimento	Sincrono Asincrono	Nastri Mouse
Condivisione	Dedicato Condivisibile	Nastri Tastiera
Velocità	Pochi byte/s Gigabyte/s	Tastiera Schede di rete
Direzione di I/O	Sola lettura Sola scrittura Lettura/scrittura	Mouse Scheda video Dischi

Abbiamo detto che ogni dispositivo viene interfacciato al sistema operativo mediante un driver che viene fornito dal costruttore del dispositivo: la standardizzazione dei driver ha fatto in modo di offrire ai programmatori metodi di accesso uniformi che gli permettono di operare a un livello di astrazione tale per cui l'utente può gestire tutte le periferiche con le stesse modalità operative.

Sulle periferiche, data la loro eterogeneità, possono essere fatte molteplici operazioni, ma esistono tipologie di servizi comuni a tutte e alle quali il sistema operativo deve fare fronte:

- innanzitutto le periferiche sono in comune a tutti gli utilizzatori del sistema, quindi il sistema di I/O è condiviso tra i processi e deve essere gestita la concorrenza;
- ogni periferica di I/O deve essere gestita con "equità nell'accesso" alle risorse condivise ottimizzando il throughput;
- una delle modalità di comunicazione avviene mediante le interruzioni che, come vedremo, devono essere gestite;
- devono essere verificati i permessi di accesso di ogni processo ai diversi dispositivi;
- devono essere eseguite le procedure specifiche (estremamente complesse) per gestire le operazioni di basso livello sui dispositivi.

L'hardware di I/O

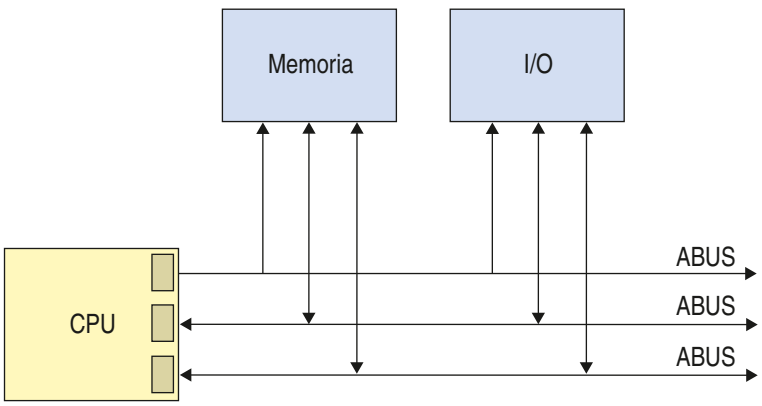
Tutte le periferiche di I/O si connettono al calcolatore e trasferiscono le informazioni mediante tre tipologie di componenti hardware: di comunicazione, di trasmissione e d'interfaccia.

Comunicazione: la porta

In generale una periferica comunica con un computer inviando segnali via cavo o via etere attraverso un punto di connessione chiamato porta (per esempio, la porta seriale, USB ecc.).

Trasmissione: il bus

Se uno o più dispositivi condividono lo stesso insieme di file e di protocolli che specificano come le informazioni possono viaggiare su di essi, questa connessione prende il nome di bus.
I bus sono largamente usati nelle architetture dei calcolatori e i messaggi si inviano tramite tensioni elettriche applicate ai fili.



In un elaboratore il bus, pur essendo fisicamente unico, può essere suddiviso in tre bus logici, a seconda della tipologia dei segnali che porta:

- ABUS, *Address Bus* (bus indirizzi): insieme di linee su cui la CPU scrive l'indirizzo del dispositivo a cui intende accedere;
- DBUS, *Data Bus* (bus dati): insieme di linee su cui viene scritto il dato che deve essere trasferito;
- CBUS, *Control Bus* (bus di controllo): insieme di linee che controllano la comunicazione tra la CPU e i dispositivi periferici.

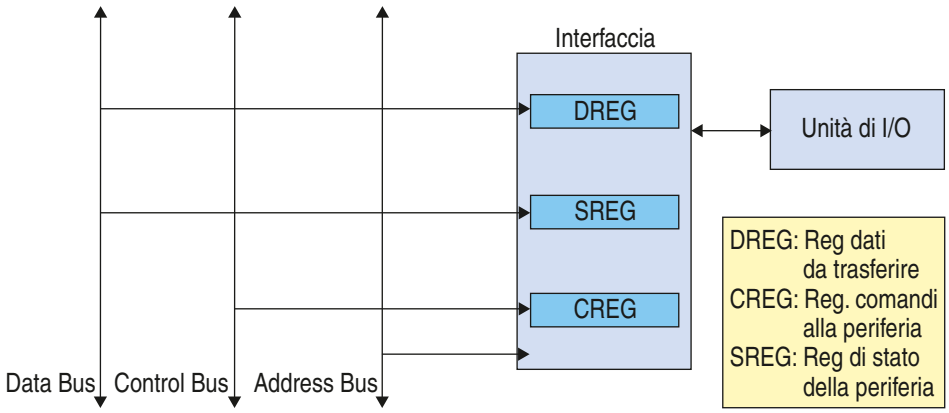
Interfaccia: il controller

I controller sono invece componenti elettronici che possono operare su una porta, un bus o una periferica e svolgono la funzione di adattamento sia elettrico sia logico tra le unità periferiche e il calcolatore.

A seconda dalla loro complessità possono essere singoli chip o circuiti separati su schede dedicate, come per esempio la scheda per l'hard disk SCSI dato che questo protocollo è molto complesso (la scheda SCSI contiene un'unità di elaborazione, microcodice e memoria privata per elaborare i messaggi del protocollo).

Ogni interfaccia ha generalmente tre registri:

- Status, registro di stato della periferica (SREG): può essere letto dal computer e il suo contenuto indica lo stato della porta, l'avvenuta esecuzione di un'operazione, la disponibilità di byte per la lettura nel registro data-in, errori ecc.
- Control, registro di controllo o di comando (CREG): viene scritto dal computer per lanciare un comando o cambiare il modo di funzionamento del dispositivo (per esempio, passare da half-duplex a full-duplex in una porta seriale).
- Data-in/Data-out, registro di dati in ingresso e registro dei dati in uscita (DREG): la CPU legge/scrive in questi registri i dati e a questi due registri possono essere associati dei chip FIFO integrati per estendere di parecchi byte la capacità del controller.



Trasferimento dati

Comunicare con il controller: indirizzamento dell'I/O

Il sistema operativo ha due modi per comunicare con il dispositivo, o meglio con il controller del dispositivo, per eseguire un insieme di istruzioni di I/O dedicate:

- indirizzamento dei registri di I/O separato tra Memoria e I/O: Isolated I/O;
- indirizzamento dei registri di I/O condiviso tra Memoria e I/O: Memory mapped I/O.

Istruzioni dirette di I/O: Isolated I/O

Nel primo sistema lo spazio degli indirizzi dei registri del controller è diverso da quello della memoria centrale (disgiunto) e si adottano apposite istruzioni di I/O destinate a manipolare i registri (IN e OUT): a ogni registro è quindi assegnato un numero di 8/16 bit, che specifica la porta di I/O corrispondente a cui ci si riferisce usando istruzioni di I/O.

◀ Una CPU con *n* linee di indirizzo è in grado di indirizzare 2^n locazioni differenti: questo insieme di indirizzi prende il nome spazio di indirizzamento. ▶

A livello hardware è però necessario prevedere opportuni meccanismi (I/O bus, linee dedicate) per discernere gli indirizzi di I/O da quelli relativi alla memoria (questa soluzione è adottata nell'Intel 8086).

La tabella seguente riporta gli indirizzi di memoria utilizzati dai processori Intel.

I/O address range (hexadecimal)	Device
000-00F	DMA controlle
021-021	interrupt controller
040-043	timer
200-20F	game controller
220-22F	sound card
230-23F	scsi host adapter
2FB-2FF	serial port (secondary)
320-32F	hard-disk controller
378-37F	parallel port
3D0-3DF	graphics controller
3F0-3F7	diskette-drive controller
3F8-3FF	serial port (primary)

Memory-mapped I/O

Un altro sistema è il mappaggio dell'I/O in memoria, con la CPU che esegue le richieste di I/O utilizzando le istruzioni standard di trasferimento dati in memoria centrale: ai registri corrispondono aree in memoria centrale, cioè sono connessi come le normali celle di memoria.
In questo modo viene ridotto lo spazio di indirizzamento per la memoria, con il vantaggio però di poter accedere ai registri delle periferiche usando tutte le istruzioni e i modi di indirizzamento utilizzabili per accedere alla memoria.
Questa soluzione è adottata per esempio dal Motorola 68000.

I personal computer odierni usano sia istruzioni di I/O sia istruzioni memory mapped I/O per controllare i dispositivi, e ogni porta ha un suo indirizzo.

ESEMPIO

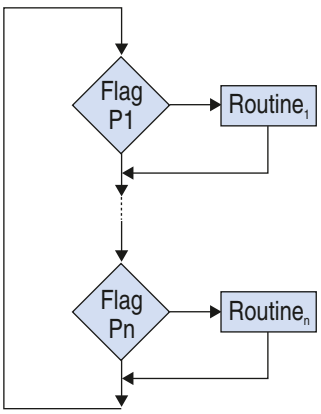
Il controller della scheda grafica ha delle porte per il controllo delle operazioni di base e una regione di memoria mappata in quella centrale (memoria grafica) che serve a mantenere i contenuti dello schermo.
Le scritture avvengono tramite quest'area, che offre maggiore facilità di lettura e viene utilizzata comunemente con le schede video per rendere veloci le operazioni di visualizzazione su schermo invece di avere milioni di operazioni di I/O.

Comunicare con il controller: protocolli

Vediamo ora quali sono le modalità di comunicazione tra controller e driver.
L'obiettivo è quello di bilanciare costo/prestazioni, per cui utilizzeremo come parametri di confronto il tempo di reazione, la velocità di trasferimento dati, l'efficienza nell'uso della CPU, la complessità, la flessibilità e il costo.

Le esigenze sono sostanzialmente quelle di sincronizzare il processore con la periferica e di trasferire i dati raggiungendo adeguate velocità senza sovraccaricare il processore.

I meccanismi sono tre:
- a controllo di programma, o attesa attiva: polling;
- controllo di interruzione;
- accesso diretto alla memoria DMA.



Attesa attiva: polling

Questo meccanismo prende anche il nome di I/O a controllo di programma perché la CPU inizia, dirige e termina l'operazione di I/O, rimanendo in attesa del completamento.
Il funzionamento è molto semplice: il processore interroga periodicamente tutte le periferiche (polling) per capire se vi sono dati da leggere o se è possibile scrivere su di esse.

Il protocollo per l'interazione tra computer e controller si basa sulla nozione di ◀ handshaking ▶.
Operativamente vengono effettuate queste operazioni:

1. il computer continua a leggere in polling il bit busy del dispositivo finché non lo trova sul valore 0 (disponibile);
2. se ha bisogno di tale dispositivo, il processore specifica il comando nel registro di controllo e setta il bit command-ready su 1, così che la periferica se ne accorga e lo esegua;
3. il controller mette a 1 il bit busy (occupato) ed esegue il comando;
4. tutti i dati vengono scambiati tramite una porta di lettura/scrittura;
5. terminata l'esecuzione vengono azzerati entrambi i bit.

◀ Il termine handshake (letteralmente "stretta di mano") sta a indicare che due dispositivi prima di iniziare a scambiarsi informazioni devono essere entrambi pronti e questo avviene aspettandosi e scambiandosi dei segnali prestabiliti: questo "periodo preliminare di trattative" viene chiamato handshaking e fa parte del protocollo di comunicazione.
La comunicazione vera e propria inizia quando entrambi sono pronti, cioè si sono "dati la mano reciprocamente". ▶

Questa modalità di operazione prende il nome di attesa attiva (busy wait) perché è un'attesa che presuppone un controllo periodico del bit e quindi un dispendio di operazioni, cioè si spreca tempo di CPU.

I vantaggi di questo meccanismo stanno nella semplicità, nella flessibilità e nel basso costo: questa tecnica dunque si può utilizzare per sistemi piccoli e poco complessi.

Gli svantaggi consistono nello scarso sfruttamento della CPU soprattutto in caso di eventi sporadici e periferiche lente; tutta la gestione dei dispositivi di I/O è totalmente demandata alla CPU che "spreca" la maggior parte del tempo dedicato al programma principale nell'esecuzione del ciclo di polling.
È inoltre difficile introdurre concetti di urgenza e quindi le latenze di risposta risultano a volte notevoli.

ESEMPIO

Proviamo a fare dei conteggi sul tempo perso in tre situazioni ipotizzando di avere un processore a 500 MHz e che siano richiesti 400 cicli di clock per l'operazione di polling.

Mouse: testato 30 volte/s per non perdere i movimenti dell'utente
Polling Clock/s = 30 * 400 = 12.000 = 12 * 10^3 clock/s
% Processore per polling = 12 * 10^3 / (500 * 10^6) = 0,002% ⇒ *Impatto limitato*

Floppy: trasferimento dati di 2 byte alla velocità di 50 kB/s per non perdere dati
Polling Clock/s = 50 kB / 2 * 400 = 10.000.000 = 10 * 10^6 clock/s
% Processore per polling = 10 * 10^6 / (500 * 10^6) = 2% ⇒ *Basso spreco*

Hard disk: trasferimento di blocchi di 16 byte alla velocità di 8 MB/s per non perdere dati
Hard Disk Polling Clock/s = 8 MB / 16 * 400 = 200 * 10^6 clock/s
% Processore per polling = 200 * 10^6 / (500 * 10^6) = 40% ⇒ *Inaccettabile*

Il polling è adatto con periferiche come il mouse, che per esempio può essere interrogato 30 volte al secondo.

Interrupt

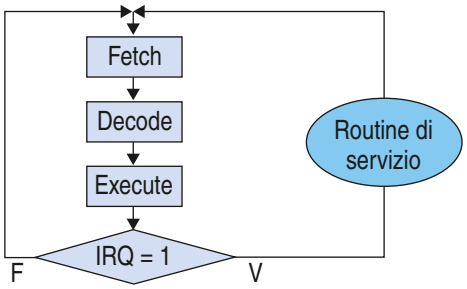
Per evitare che il processore sprechi tempo nell'attesa che la periferica sia libera è sufficiente che la procedura di lettura sia eseguita solo quando la periferica è pronta: così facendo si risparmia la maggior parte del tempo sprecato facendo busy waiting.
Una soluzione è quella di far "avvisare" il processore che la periferica è disponibile e/o ha bisogno di scambiare dati con la CPU mediante un messaggio particolare: la periferica richiede un'interruzione (manda un segnale di interrupt).

Vi sono più metodi per rilevare un interrupt, linee di richiesta multiple, scansione degli interrupt, vettorializzazione degli interrupt: descriveremo il primo.

Le interruzioni consentono una gestione asincrona dell'operazione di I/O: la CPU risponderà semplicemente alla richiesta di interruzione sospendendo il processo che stava eseguendo senza essere costretta ad attendere il suo completamento.

Il meccanismo base del funzionamento degli interrupt che utilizza le linee di richiesta (IRQ, acronimo di Interrupt ReQuest) della CPU, è il seguente:
- la CPU verifica lo stato delle linee di richiesta dopo l'esecuzione di ogni istruzione;

- se il controller gli ha mandato un segnale (INTREQ), il processore salva il valore del program counter (PC) e il contenuto del registro di stato (PSW) passa in modo kernel e segnala (INTACK) l'accettazione della richiesta;
- riconosce quindi il tipo di interrupt e passa il controllo alla relativa procedura di gestione dell'interrupt (ISR, Interrupt Service Routine) che la gestisce effettuando come prima operazione il salvataggio del contesto;
- eseguite le operazioni relative all'I/O, la CPU ripristina lo stato precedente all'interrupt: in pratica, si esegue l'istruzione successiva a quella interrotta.



Zoom su...

CAMBIO DEL CONTESTO

Il salvataggio del contesto è identico a quello che avviene durante la schedulazione dei processi al temine del time slice: viene salvato lo stato del processo in una struttura dati del SO riservata a quel processo nella tabella dei processi (registri, stack, stack pointer - SP).
Ci sono alcune operazioni, come l'operazione di salvataggio del contesto, che devono essere eseguite dal processore senza che venga interrotto, altrimenti si potrebbero verificare situazioni anomale indesiderate: ogni processore ha un meccanismo di "disabilitazione delle interruzioni" affidato a un apposito flag (IE, *Interrupt Enable*): in base al suo valore viene indicata l'ininterrompibilità del processore fino al termine delle operazioni che sta eseguendo; quando viene accettata la richiesta di interruzione il processore setterà opportunamente IE così che da quel momento la sua esecuzione diventa non interrompibile.

Gli interrupt sono usati anche per realizzare trap (interruzioni software) alle procedure del kernel in modalità supervisore, quindi per l'implementazione delle chiamate di sistema. Per esempio vengono utilizzati per gestire le eccezioni generate dai processi nel tentativo di eseguire operazioni illecite come divisioni per zero, accesso a indirizzi protetti, page fault, chiamate di sistema.

Zoom su...

RICONOSCIMENTO ECCEZIONE

Per individuare il dispositivo che ha richiesto un'interruzione esistono sostanzialmente le tre modalità operative illustrate di seguito.

- Salto a indirizzo fisso
  Nel sistema è presente un unico segnale INT per tutte le periferiche: il compito di individuare il dispositivo che lo ha richiesto viene demandato alla ISR che come prima istruzione esegue una scansione in polling dei registri di controllo delle periferiche per individuare chi ha sollevato l'interrupt.

È una soluzione semplice, economica, ma con problemi di latenza dovuti al polling.

- Segnali multipli di interruzione

  Nel sistema abbiamo un indirizzo specifico di memoria per ogni INT e questo contiene l'indirizzo della ISR.

  È una soluzione con maggior complessità HW sulla struttura dei bus ed è utilizzabile solo nei sistemi che hanno un numero limitato di periferiche.

- Interruzione vettorizzata

  Nel sistema il segnale INT è unico, ma contemporaneamente a questo ogni periferica invia sul bus dati un codice univoco con il quale si autoidentifica: prende il nome di vettore di interruzione, e questo valore viene usato come indice di una tabella contenente gli indirizzi delle varie ISR.

  È una tecnica molto efficiente, flessibile ma piuttosto costosa.

Esempio

Gli eventi da 0 a 31 non sono mascherabili e si usano per segnalare varie condizioni di errore.
Quelli da 32 a 255 sono mascherabili e si usano per le interruzioni generate dai dispositivi.

Numero interruzione	Descrizione
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19Ð31	(Intel reserved, do not use)
21Ð255	maskable interrupts

I problemi da risolvere nel meccanismo a interruzione sono i seguenti:

- occorre identificare la periferica e attivare la specifica ISR;
- deve essere stabilito un tempo limite per servire l'ISR;
- è possibile avere più richieste da esaudire, quindi è necessario introdurre delle priorità;
- è possibile ricevere la richiesta di un'interruzione mentre si sta già eseguendo una ISR.

Il vantaggio della gestione a interrupt sta nella maggior efficienza rispetto al polling e alla maggiore flessibilità offerta grazie anche alla possibilità di gestire priorità e annidamenti: inoltre le interruzioni sono eseguite in modo *trasparente* rispetto ai programmi, cioè senza che "questi se ne accorgano".

Lo svantaggio sta nel fatto che abbiamo ancora un overhead per la CPU in quanto può essere elevato il tempo di latenza dal momento in cui viene generato l'interrupt all'istante in cui si inizia a lavorare per l'I/O.
Inoltre, parte del tempo di CPU viene utilizzata per effettuare il salvataggio/ripristino dei registri: il salvataggio deve essere fatto all'inizio della ISR mentre il ripristino alla fine della procedura. Per minimizzare tale tempo si cerca di ridurre al minimo il numero di registri da salvare eventualmente aggiungendo registri dedicati e/o set di registri aggiuntivi per l'esecuzione della ISR.

DMA - Direct Memory Access

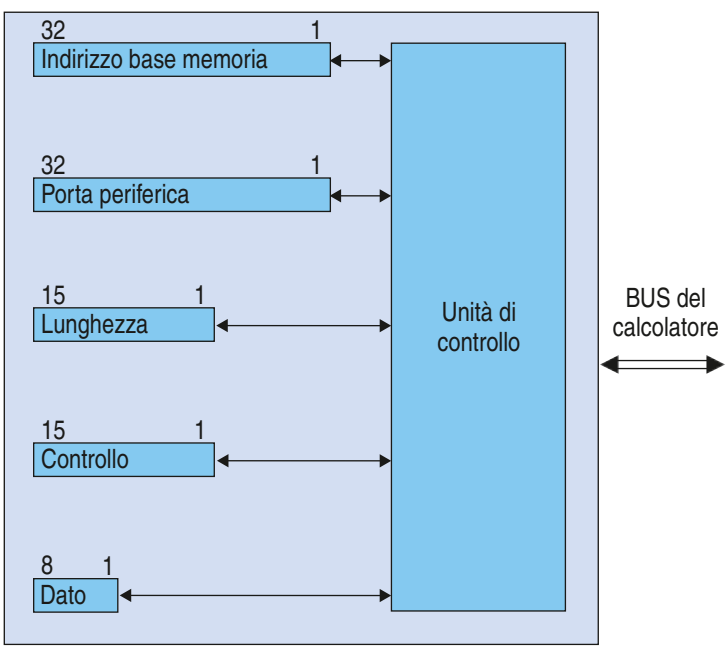
Alcuni dispositivi come gli hard disk richiedono trasferimenti di grosse quantità di dati contigui in tempi ridotti e le modalità fino ad ora descritte risultano largamente inefficienti: per evitare la latenza tipica delle interruzioni e l'intrinseca lentezza della gestione software si utilizzano nuove strategie in grado di interfacciare direttamente la memoria con i dispositivi di I/O: la tecnica che descriviamo prende nome di DMA, cioè di accesso diretto alla memoria (Direct Memory Access).

Viene realizzata mediante appositi dispositivi hardware contenenti processori specifici, i DMAC (DMA Controller), che sono capaci di trasferire un blocco di dati tra la memoria e una periferica operando direttamente sul bus di memoria senza alcun intervento della CPU, salvo poi lanciarle un interrupt a trasferimento compiuto.

In generale l'utilizzo di un DMAC consente di superare due limiti tipici sia dell'I/O SW sia di quello HW gestito tramite interrupt:

- la velocità di trasferimento non è più vincolata alla velocità con la quale il processore può verificare e servire una periferica;
- le istruzione di I/O non sono più eseguite dalla CPU.

Un blocco di comandi DMA contiene un puntatore alla sorgente del trasferimento, uno alla destinazione e un contatore con il numero dei byte da trasferire.
La figura che segue presenta lo schema di un DMAC.



I registri hanno specifici utilizzi, che sono:

- indirizzo base: contiene l'indirizzo iniziale della memoria in cui trasferire il blocco dati;
- porta periferica: indirizzo della porta periferica dove leggere/scrivere i dati da trasferire;

- lunghezza: contiene la dimensione del blocco di dati da trasferire;
- controllo: registro con campi per codici errore, abilitazione DMA, fine DMA ecc.;
- dato: registro tampone, se presente, usato durante il trasferimento.

Riassumiamo in uno schema le caratteristiche dei tre meccanismi che abbiamo descritto indicando la modalità con cui vengono eseguiti, cioè software (SW) oppure hardware (HW), ricordando che:

- SW: costoso da sviluppare, economico da riprodurre, elaborazioni flessibili, articolate ma lente;
- HW: costo di realizzazione e collaudo di ogni esemplare, elaborazioni veloci, semplici ma rigide.

	Controllo di programma	Interruzione	DMA
complessità	bassa	media	alta
costo	basso	medio	alto
prestazioni	basse	medie	alte
sincronizzazione	SW	HW	HW
trasferimento	SW	SW	HW
tempo di risposta	10 ms	50 ms	2 ms
efficienza	scarsa	media	ottima

Il sottosistema di I/O del kernel

Concludiamo questa trattazione delle modalità di I/O ricordando brevemente altri servizi che sono messi a disposizione dal sottosistema di I/O del kernel per garantire buone prestazione delle operazioni di I/O:

- I/O scheduling;
- buffering;
- caching;
- spooling.

I/O scheduling

La schedulazione di una serie di richieste di I/O consiste nel trovare un ordine efficiente di gestione delle chiamate di I/O migliorando le prestazioni globali del sistema. Per garantire buone prestazioni viene gestita una coda per ogni processo, e si sceglie in base a diversi principi quale richiesta soddisfare, prediligendo le richieste di I/O dei processi a priorità maggiore e decidendo inoltre in base a quanto l'operazione di I/O sia onerosa in termini di tempo.

Buffering

Per aumentare l'efficienza del sistema di I/O il SO prevede alcune aree di memoria (buffer) in cui trasferire i dati di input provenienti dai dispositivi prima di trasferirli ai processi utente e i dati di output prima di trasferirli ai dispositivi.

Il buffer è quindi una zona di memoria in cui vengono memorizzati temporaneamente i dati.

Si usa principalmente per tre motivi:
- per gestire una differenza di velocità tra il produttore e il consumatore di un flusso di dati;
- per gestire la differenza di dimensioni nell'unità di trasferimento dove sono presenti componenti che hanno blocchi di dimensioni diverse;
- per implementare la *semantica della copia*, che garantisce che la versione dei dati scritta su un dispositivo sia la stessa presente in memoria al momento della chiamata di sistema dell'applicazione.

Caching

La cache è una zona di memoria veloce che conserva in memoria primaria copie dei dati recentemente utilizzati così che, se la CPU vi deve accedere di nuovo, questi possono essere prelevati direttamente dalla cache senza dover accedere nuovamente alla memoria secondaria.

La differenza rispetto al buffer è che nel buffer si trova l'unica istanza di un'informazione mentre la cache mantiene la copia di un'informazione.

> Il programmatore e/o il compilatore può realizzare una cache utilizzando i registri programmabili della CPU.

Spooling e prenotazione dei dispositivi

Lo spool è un buffer su memoria di massa che viene utilizzato generalmente per i dispositivi di output non condivisibili, come per esempio le stampanti.

Per evitare che i processi effettuino lunghe attese su questo tipo di dispositivi, il SO utilizza la tecnica di spooling (Simultaneous Peripheral Operation On Line), che consiste nell'assegnare a ogni processo che lo richieda un dispositivo virtuale, realizzato da un file su disco:
- ciascun processo ottiene subito l'accesso al file che rappresenta il dispositivo richiesto;
- invece di trasferire i dati al dispositivo questi vengono scritti sul file assegnato;
- quando un processo chiude il proprio dispositivo virtuale, il suo file viene inserito nella coda dei file che saranno trasferiti al dispositivo fisico, uno alla volta, da un processo di sistema (spooler).

Lo spooler è l'unico programma che ha accesso diretto alla periferica e offre agli utenti diverse operazioni aggiuntive come la visualizzazione della coda di attesa dei processi o la cancellazione/sospensione di uno di essi, come riportato nella figura seguente.

Verifichiamo le conoscenze

Esercizi a scelta multipla

1 Che cosa sono i driver?

- O collegamenti fisici alle periferiche
- O sistemi di scrittura sulle periferiche
- O programmi che permettono l'interfacciamento delle periferiche
- O canali di comunicazione con la CPU

2 Quali di queste affermazioni sono false?

- O il joystick è un dispositivo di puntamento
- O la linea seriale è un dispositivo di collegamento
- O il mouse è un dispositivo di ingresso
- O il microfono è un dispositivo di collegamento
- O il nastro è un dispositivo di output dedicato
- O il joystick è un dispositivo sincrono

3 Che cosa si intende con porta?

- O il dispositivo che permette di accedere all'interno del PC
- O un sistema di connessione della periferica
- O un dispositivo per effettuare la trasmissione dei segnali
- O un'interfaccia per le periferiche

4 Quale tra i seguenti non rientra tra i BUS standard?

- O ABUS: bus indirizzi
- O BBUS: bus di blocchi di dati
- O CBUS: bus di controllo
- O DBUS: bus dati

5 Quali tra i seguenti non sono registri presenti su un'interfaccia?

- O AREG: registro di indirizzo
- O CREG: registro dei comandi
- O DBUS: registro dei dati
- O SREG: registro di stato

6 Il sistema Memory-mapped I/O (indicare l'affermazione errata):

- O utilizza le istruzioni standard di trasferimento dati in memoria centrale
- O prevede che i registri siano connessi come le normali celle di memoria
- O è utilizzato per esempio dalla scheda grafica
- O non influisce sullo spazio di indirizzamento per la memoria

7 Il protocolli di comunicazione con il controller sono:

- O polling, scheduling, DMA
- O scheduling, interruzione, DMA
- O polling, interruzione, scheduling
- O polling, interruzione, DMA

8 Nell'attesa attiva (indicare l'affermazione errata):

- O avviene l'I/O a controllo di programma
- O avviene l'handshaking
- O abbiamo alta flessibilità e basso costo
- O la velocità è buona perché la effettua la CPU

9 Con IRQ si intende:

- O l'abbreviazione di *Interrupt ReQuest*
- O l'acronimo di *Interrupt Request Query*
- O l'acronimo di *Interrupt Ready Query*
- O l'acronimo di *Internal Request*

10 Un processore riceve una richiesta di interruzione: quando parte il processo che porta all'attivazione della relativa procedura di servizio dell'interruzione?

- O immediatamente
- O al termine del corrente ciclo di clock
- O al termine del corrente ciclo di bus
- O al termine dell'istruzione corrente

Test vero/falso

1 Il mouse è un dispositivo asincrono. V F
2 Il modem è un dispositivo ad accesso sequenziale. V F
3 La porta è un componente del kernel del sistema operativo. V F
4 Il DBUS è costituito dall'insieme di linee su cui viene indirizzato il dispositivo. V F
5 Nell'*Isolated I/O* lo spazio degli indirizzi è diverso da quello della memoria centrale. V F
6 Il sistema *Memory-mapped I/O* riduce lo spazio di indirizzamento per la memoria. V F
7 Il sistema *Isolated I/O* è utilizzato nel Motorola 68000. V F
8 Nel polling tra gli svantaggi c'è anche lo scarso sfruttamento della CPU. V F
9 L'utilizzo del polling è sconsigliato per la gestione del mouse. V F
10 Con la gestione di un HD in polling si sprecherebbe circa il 40% del tempo CPU. V F
11 Le interruzioni consentono una gestione sincrona dell'operazione di I/O. V F
12 I segnali multipli di interruzione si utilizzano solo con un numero limitato di periferiche. V F
13 Il meccanismo di interruzione vettorizzata prevede un sistema di segnale INT unico. V F
14 Nel DMA le istruzioni di I/O sono eseguite velocemente dalla CPU. V F
15 La schedulazione delle richieste di I/O utilizza una coda per ogni processo. V F

Esercizi di completamento

1 I ..... offrono dei meccanismi che permettono il trattamento ..... di ogni dispositivo di I/O da parte delle applicazioni e del sistema operativo.

2 In generale una periferica comunica con un computer inviando segnali via cavo o via etere attraverso un punto di connessione detto .....

3 I controller sono componenti elettronici che possono operare su una ....., un ..... o una ..... e svolgono la funzione di adattamento sia elettrico sia logico tra le unità periferiche e il calcolatore.

4 Un sistema di trasferimento dati è il ..... dell'I/O in memoria, con la CPU che esegue le richieste di I/O utilizzando le istruzioni standard di trasferimento dati in memoria centrale.

5 Con ..... si indica la situazione in cui si ripete il controllo di una condizione sprecando tempo di calcolo.

6 Vi sono più metodi per rilevare un interrupt, linee di ..... multiple, ... degli interrupt, ... degli interrupt.

5 MODULO FASI E MODELLI DI GESTIONE DI UN CICLO DI SVILUPPO

OBIETTIVI

- Comprendere le necessità di una metodologia per lo sviluppo di sistemi informatici
- Conoscere gli elementi fondamentali dell'ingegneria del software
- Comprendere il concetto di astrazione
- Conoscere gli elementi teorici della progettazione a oggetti (OOP)
- Conoscere una metodologia di documentazione (UML)
- Capire l'utilizzo delle schede CRC per l'identificazione di classi

ATTIVITÀ

- Individuare e descrivere il problema complesso
- Scegliere le metodologie e le tecniche adeguate alle diverse situazioni
- Usare la progettazione orientata agli oggetti per programmi complessi
- Applicare il concetto di astrazione per modellare le classi
- Rappresentare classi e oggetti mediante diagrammi UML
- Usare i diagrammi UML per descrivere le relazioni tra gli elementi di un progetto
- Usare la progettazione orientata agli oggetti per sistemi informatici complessi

UNITÀ DIDATTICA 1

MODELLI CLASSICI DI SVILUPPO DI SISTEMI INFORMATICI

IN QUESTA UNITÀ IMPAREREMO...

- a individuare e descrivere il problema
- a costruire la soluzione di un problema
- ad affrontare correttamente la ricerca della soluzione
- a scegliere le metodologie e le tecniche adeguate alle diverse situazioni

Introduzione

Per realizzare con successo un qualunque sistema software, che si tratti di un semplice esercizio di laboratorio o di un progetto complesso come un nuovo sistema per la gestione di Malpensa 2000, è indispensabile utilizzare un approccio rigoroso in ogni fase della realizzazione, dalla pianificazione alla progettazione e infine al "collaudo sul campo".
Per i progetti di medie-grosse dimensioni la quantità di tempo che si dedica alla pianificazione deve necessariamente superare il tempo che si impiega per la programmazione e il collaudo: di fatto questo non avviene "sempre" e ci si accorge della scarsa (o totale assenza) di pianificazione quando si sta per la maggior parte del tempo davanti al computer, digitando codice sorgente e correggendo errori.
Questo è un sintomo che segnala la probabile carenza di una corretta metodologia di analisi e di progetto, che porta a una inutile (e "dolorosa") perdita di tempo, come già ricordato dalla Legge di Mayers, che riportiamo.

"È bene trascurare le fasi di analisi e progetto e precipitarsi all'implementazione allo scopo di guadagnare il tempo necessario per rimediare agli errori commessi per aver trascurato la fase di analisi e di progetto."

Sicuramente è possibile ridurre l'impegno complessivo dedicando più tempo e metodo alla fase di progettazione e pianificazione.
In questa unità didattica verrà descritto come affrontare in modo sistematico il progetto e la realizzazione di un sistema software.

Il mestiere del programmatore

Il lavoro dei programmatori si può riassumere semplicemente nelle due parole "**produrre programmi**" che però, come ben sappiamo, mascherano un insieme di attività complesse ben lontane dalla semplice codifica di un algoritmo in un linguaggio di programmazione (operazione effettuata dai cosiddetti *codificatori* o *manovali del software*).
Il programmatore programma, è cioè un "artista" della disciplina che prende il nome di programmazione, dove con programmazione intendiamo quanto segue.

PROGRAMMAZIONE

Programmazione è l'insieme di quelle attività che, a partire da un problema, conducono alla stesura di un programma la cui esecuzione da parte di un calcolatore ha come risultato la soluzione del problema dato.

Si parte quindi dal **problema** per arrivare al **programma**, individuando la **soluzione** corretta: questo è il principale compito del programmatore, cioè la ricerca della giusta soluzione. Ed è anche il principale problema: come si fa ad arrivare alla risoluzione di un dato problema?
Se consideriamo il problema come "*una questione da risolvere partendo da elementi noti mediante il ragionamento*" ne deriva che il programmatore dapprima si pone una serie di domande, necessarie proprio per avere gli elementi della conoscenza del problema, e solo successivamente può iniziare il lavoro di progetto.

Ma molte domande attendono una risposta.
- In cosa consiste il problema?
- Come si costruisce la soluzione di un problema?
- Qual è il giusto punto di partenza?
- Quali metodologie o tecniche utilizzare?

Il primo problema è proprio quello di "capire il problema"! Questo spesso non è solo di aritmetica, di geometria, di algebra ma, generalmente, è "*una questione, situazione difficile o complessa di cui si cerca la soluzione*" (dal greco *pròblema*, da *proballo* = metto avanti, propongo – Vocabolario Gabrielli).
Per poter iniziare a ricercare e quindi proporre una soluzione il programmatore deve "essere a conoscenza" di tutte le possibili forme e sfaccettature che può avere il problema, dai casi particolari a quelli imprevisti, dalle situazioni classiche a quelle improbabili; deve effettuare un'"indagine" approfondita alla "Sherlock Holmes" per diventare "proprietario della conoscenza" di ciò che dovrà risolvere: deve *analizzare la situazione* e produrre un modello esemplificativo della realtà.
Il secondo problema inizia quando finisce il primo: una volta individuato *cosa si deve risolvere* (e, come si vedrà, già questo non sempre è semplice!) "il bello viene adesso", cioè si deve progettare la soluzione che risolve il problema. Programmare non è quindi un'operazione semplice! Il programmatore non è comunque lasciato solo a se stesso ma supportato da tecniche, metodologie e strumenti che lo aiutano in ogni fase della progettazione del programma.
Nel seguito della trattazione verranno affrontate e proposte diverse tecniche, studiate e perfezionate nel corso degli anni: anche se da sole non assicurano sempre il raggiungimento

dell'obiettivo, ne sottolineiamo l'importanza, poiché spesso servono per "far imboccare" la strada giusta per raggiungere la risoluzione dei problemi; sono cioè un aiuto al programmatore e suggeriscono tattiche e strategie che permettono di iniziare il progetto della ricerca della soluzione; altre volte forniscono proprio la traccia da seguire per ottenere la soluzione.
Un primo aiuto ci viene dalle tecniche perfezionate e studiate nella disciplina che prende il nome di ingegneria del software, alcuni elementi della quale verranno trattati nei prossimi paragrafi. Questa disciplina è nata proprio per mettere ordine nella materia, mentre **un secondo aiuto** si chiama seniority (anzianità), cioè il frutto dell'esperienza "maturata sul campo", ottenuta con il perfezionamento delle metodologie classiche.
Questi aiuti vanno ad aggiungersi agli "strumenti personali" propri (e indispensabili) dei quali ogni programmatore è dotato: l'esperienza, la fantasia, l'intuito e l'ingegno.
Infatti, spesso il mezzo per accostarsi ai problemi non è solo quello razionale ma soprattutto quello creativo: sviluppare un pensiero creativo significa "dare spazio alle forme di espressione" e discipline che, come l'informatica, anche se dominate dalla razionalità e dalla logica hanno bisogno di fantasia e creatività.

Ingegneria del software e ciclo di vita

L'ingegneria del software, nata negli anni '60 per formalizzare e definire le tecniche e le strategie finalizzate all'ottimizzazione e al miglioramento della produzione dei programmi, ha indicato con "ciclo di vita del software" l'insieme di tutte le attività connesse alla "produzione di un programma" e le ha indicate come illustrato di seguito.

1. **PIANIFICAZIONE DEL SISTEMA**. Denominata anche analisi del sistema (o definizione delle specifiche del sistema), identifica la relazione tra l'ambiente e l'applicazione software individuando le parti del sistema da realizzare con il software.
2. **ANALISI**. Stabilisce le caratteristiche dell'applicazione informatica, delineandone gli aspetti di interfacciamento, di prestazione e di funzionalità; nella fase di analisi si decide *che cosa* il progetto dovrebbe realizzare, senza alcun riferimento diretto a come il programma realizzerà i suoi obiettivi.
   Il prodotto della fase di analisi è un elenco di requisiti che descrive in tutti i particolari che cosa il programma sarà in grado di fare una volta che sarà stato portato a termine. Generalmente viene prodotto un vero e proprio documento, noto come "specifiche dei requisiti", fondamentale per le fasi successive: omissioni, errori, sviste e "superficialità" commesse nella fase di stesura di tale documento sono causa di criticità e portano spesso al fallimento di tutto il progetto.

In questo documento sono presenti anche altre due parti:
- il manuale per l'utente, che indica in che modo l'utente utilizzerà il programma per ricavarne i vantaggi desiderati;
- le prestazioni richieste al sistema e le risorse utilizzate, cioè quanti e quali dati in ingresso il programma dovrà essere in grado di gestire e in quali tempi, la tipologia degli archivi o le connessioni in rete locale o remota, i fabbisogni massimi di memoria e di spazio su disco richiesti.

3. **PROGETTAZIONE**. Sulla base dei requisiti espressi dall'utente e focalizzati con l'analisi, incorpora tutte le attività connesse alla ideazione e definizione della strategia che porta alla soluzione e quindi all'individuazione, tra tutte le possibili strategie, di quella

più efficace ed efficiente. In particolare, nella fase di progettazione si individuano i seguenti due obiettivi:

- si sviluppa un piano di realizzazione del sistema, che comprende e descrive le diverse modalità operative da effettuare;
- si individuano le strutture dati e archivi necessarie alla soluzione del problema: se si ricorre alla progettazione orientata agli oggetti vengono stabilite le classi necessarie e i loro metodi più importanti, producendo il diagramma delle classi e di relazione tra di esse.

4 **CODIFICA (REALIZZAZIONE)**. Nella fase di realizzazione si scrive e si compila il codice sorgente eseguendo la vera e propria CODIFICA delle istruzioni in linguaggi di programmazione: questa fase è scomposta in una attività umana e in una serie di attività svolte dalla macchina:

- nella prima fase, prettamente umana, si traducono in un programma le informazioni fornite dalla formalizzazione;
- nella seconda fase il programma viene elaborato automaticamente per produrre il codice che l'esecutore è in grado di interpretare: il linguaggio macchina.

Si realizzano quindi compiutamente le classi complete di tutti i loro metodi individuati durante la fase di progettazione, ottenendo come risultato il programma finito.

5 **TEST e DEBUG**. Hanno come obiettivo quello di verificare l'assenza di errori; dato che è praticamente impossibile programmare senza errori, è necessaria una specifica fase, quella di test, nella quale viene verificata la correttezza dei risultati dell'elaborazione su un campione di dati di prova: l'eventuale presenza di errori innesca la fase di debug, cioè la ricerca della istruzione (o segmento di codice) errata per procedere alla sua correzione e all'eliminazione del "baco".

6 **INSTALLAZIONE, VERIFICA e COLLAUDO**. È un'unica fase composta da tre momenti:

- con installazione si intende la consegna del software al cliente mediante l'installazione fisica del programma sul sistema del cliente stesso;
- la verifica viene effettuata dagli utenti del programma e consiste nell'accertare che il software sia corretto rispetto alle specifiche individuate dell'analista e quindi conforme a quanto specificato nelle loro richieste, cioè se il risultato è un prodotto corrispondente allo scopo per il quale è stato richiesto;
- il collaudo verifica che, eseguendo prove con dati reali, il programma funzioni correttamente.

Al termine di questa fase si redige un documento sottoscritto da entrambe le parti, committente ed esecutore, che attesta la bontà del prodotto e la sua aderenza alle richieste del cliente, e descrive i collaudi che sono stati eseguiti e i loro risultati.

7 **MANUTENZIONE**. È la fase permanente di supporto al sistema dopo la consegna all'utente; le attività svolte in questo passo del ciclo di vita di una applicazione riguardano:

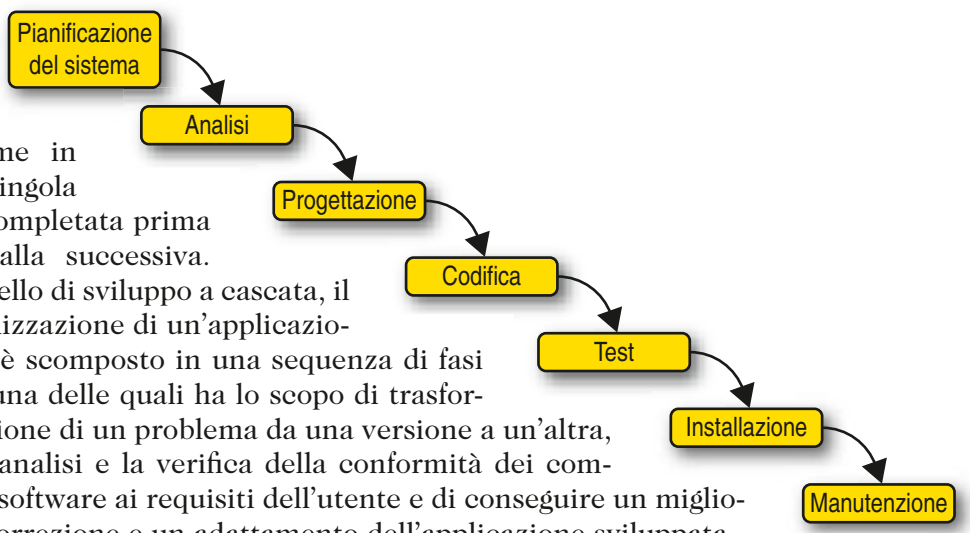
- l'aspetto correttivo, che consiste nell'eliminare gli errori identificati;
- l'aspetto adattativo, che ha l'obiettivo di apportare le modifiche necessarie per il trasferimento dell'applicazione informatica su altri sistemi, anche a seguito di innovazione tecnologica;
- l'aspetto migliorativo, che si propone di aggiungere nuove funzionalità o di ottimizzare quelle esistenti.

Questa fase risulta agevolata se nelle fasi precedenti si è adottato un buono stile di programmazione e si è prodotta un'adeguata documentazione interna (commenti) ed esterna (manuale d'uso e manuale tecnico) relativa al programma.

Questo modello dello sviluppo del software prende il nome di modello a cascata e i procedimenti formali di sviluppo vennero definiti per la prima volta agli inizi degli anni '70: gli allora progettisti del software avevano un modello molto semplice per queste fasi dando per scontato che, una volta portata a termine una fase, il suo risultato si sarebbe riversato sulla fase successiva, che a quel punto si sarebbe avviata. Purtroppo, l'esperienza "sul campo" ha constatato il fallimento di questa modalità operativa all'aumentare delle esigenze e delle dimensioni del progetto.

Sono stati sviluppati negli anni successivi altri modelli per definire un processo in grado di soddisfare tutte le esigenze di progettazione: di seguito sono riportati i principali modelli, e descritta sinteticamente la loro principale caratteristica, a partire dal modello a cascata a appena descritto.

Modello a cascata



Il modello a cascata (o modello waterfall) prende questo nome in quanto ogni singola attività viene completata prima del passaggio alla successiva. Secondo il modello di sviluppo a cascata, il processo di realizzazione di un'applicazione informatica è scomposto in una sequenza di fasi successive ognuna delle quali ha lo scopo di trasformare la descrizione di un problema da una versione a un'altra, di consentire l'analisi e la verifica della conformità dei comportamenti del software ai requisiti dell'utente e di conseguire un miglioramento, una correzione e un adattamento dell'applicazione sviluppata. In questo modello, ogni singolo passo viene svolto solo se i passi che lo hanno preceduto sono stati completati.

I limiti della metodologia a cascata sono evidenti: la semplice struttura del processo facilita il lavoro degli sviluppatori ma spesso ci si trova in una situazione nella quale ci si rende conto che sarebbe necessario tornare a quella precedente nella quale è stato trascurato qualche elemento ritenuto superfluo e rivelatosi poi indispensabile per poter procedere con il progetto: per esempio, in fase di sviluppo, potrebbe risultare necessario tornare alla fase di analisi per poter effettuare approfondimenti su qualche aspetto del sistema.

Questo modello non è di facile applicazione con progetti di medie dimensioni dove i requisiti utente non sono chiaramente definiti e sono suscettibili di modifiche ed evoluzioni.

Modello a prototipazione rapida

Si esegue un processo iterativo che raffina di volta in volta il risultato prodotto fino al raggiungimento di un prodotto che soddisfa sia l'utente sia lo sviluppatore.
Il processo di sviluppo inizia con la fase di raccolta dei requisiti che sono ritenuti *di rischio più elevato* e nella realizzazione di un progetto che rispetti tali requisiti: dato che il processo è iterativo, tutte le fasi si sviluppano per affinamenti successivi. Per esempio, la prima fase comincia con una prima iterazione dove si ottiene dall'utente una versione iniziale dei requisiti, che poi vengono rivisti nelle iterazioni successive sulla base di commenti e/o rettifiche ed integrazioni proposte dall'utente, fino a raggiungere la versione definitiva.
Successivamente, si costruisce un prototipo. Tale prototipo viene ottenuto, in genere, in tempi molto brevi; spesso si utilizzano degli strumenti automatici che lo costruiscono direttamente (strumenti "*case*"); il prototipo viene sottoposto all'attenzione dell'utilizzatore in modo che quest'ultimo lo visioni, provi a utilizzarlo ed esprima le differenze riscontrate rispetto alle sue aspettative. Sulla base delle indicazioni fornite, il prototipo viene modificato e rivisto dal progettista sino a quando le incongruenze tra prodotto e aspettative non sono ridotte al minimo: cliente e sviluppatore sono entrambi soddisfatti. Il prototipo ultimato costituisce il punto di partenza per la realizzazione del prodotto finale.

I problemi relativi allo sviluppo di una applicazione informatica basata sullo sviluppo e sul raffinamento di un prototipo riguardano, essenzialmente, due aspetti.

1. Il primo svantaggio riguarda il fatto che la valutazione del prototipo da parte del committente spesso non riguarda gli aspetti che il prototipo voleva evidenziare, ma piuttosto altre caratteristiche, come per esempio il layout, i colori, e altri dettagli di secondaria importanza molto lontani dallo scopo per cui è realizzato il prototipo, limitando gli effetti che il prototipo stesso si prefiggeva di raggiungere.
2. Il secondo problema è legato al fatto che il prototipo viene usato come base di sviluppo invece che come "chiarificatore di requisiti": quando il prototipo è operativo gli sviluppatori, spesso sotto la pressione del committente, lo utilizzano come base per la costruzione del prodotto, trascurando le carenze costruttive del prototipo stesso.

Modello incrementale

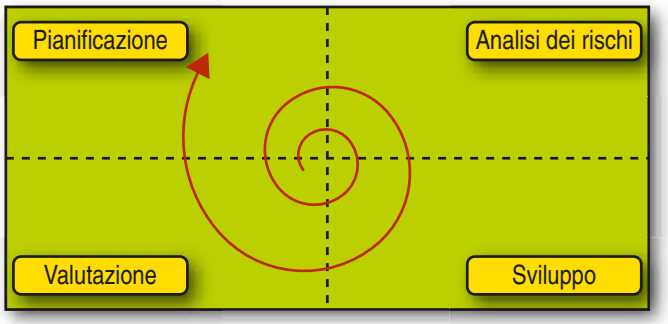
Il modello incrementale può essere considerato una evoluzione dei due modelli precedenti: si segue il principio del modello a cascata dove, però, una o più fasi sono eseguite per incrementi successivi al fine di creare versioni successive dello stesso prodotto; ogni versione soddisfa una parte dei requisiti (un sottosistema), iniziando a implementare dapprima quelli più critici fino ai meno pressanti (o marginali).

In particolare, si segue il modello a cascata nelle fasi di analisi e formalizzazione, anche se con qualche leggera differenza dal modello puro in quanto in tali fasi si devono anche identificare le priorità e le criticità dei sottoinsiemi individuando la "cronologia" da presentare all'utente finale e le modalità di interconnessione degli stessi. Ogni sottosistema soddisfa solo una parte dei requisiti: i più critici prima e, a seguire, i meno pressanti e la loro realizzazione viene effettuata in modo incrementale realizzandoli **uno alla volta**, cioè una parte per volta del sistema complessivo.
Una conseguenza di questo approccio è che, per ogni realizzazione di un sottosistema, deve essere prevista anche la coppia di fasi di codifica e di test dei sottosistemi software, l'integrazione di tali sottosistemi e il test dell'intero sistema prodotto.
I problemi a cui è soggetto il modello di sviluppo incrementale sono principalmente localizzati nel *riuso dei sottosistemi* e nella *integrazione* di tali sottosistemi.
Per quanto riguarda il primo aspetto, tutti i sottosistemi realizzati attraverso questo procedimento devono essere riusati nella costruzione degli incrementi successivi. Questo aspetto è determinante per contenere i costi di sviluppo che, in caso contrario, lieviterebbero fino a rendere il progetto complessivo poco competitivo rispetto ad altri metodi di sviluppo.
Il secondo punto riguarda il fatto che tutti i sottosistemi devono essere integrabili e, poiché la finalità dell'integrazione è quella di produrre l'intero sistema, devono avere lo stesso livello qualitativo poiché sono appunto frammenti dello stesso software. Questo secondo punto è strettamente legato al primo.
Si sottolinea, infine, che nel modello incrementale non è presente una fase di manutenzione separata dei sottosistemi: infatti l'aggiunta di un incremento corrisponde a una manutenzione e dunque le modifiche possono essere facilmente implementate quando si presentano.
Il ciclo di vita di un'applicazione informatica secondo il modello incrementale è riassuntonella tabella a lato. ▶

Modello a spirale

Per ovviare ai problemi dei modelli precedenti, è nata la metodologia a spirale (o iterativa) ancora oggi ampiamente usata.
Il modello a spirale è un modello generale che include i precedenti; proposto da Barry Boehm nel 1988, scompone il processo di sviluppo in quattro fasi multiple, ciascuna ripetuta più volte: la *pianificazione*, l'*analisi dei rischi*, lo *sviluppo* e la *verifica*.

Nel modello iterativo sono quindi presenti le stesse fasi del modello a cascata, ma i tempi sono più ristretti e dalla fase di testing si torna poi a quella di pianificazione per applicare eventuali correzioni al risultato dello sviluppo.



Ogni singola iterazione corrisponde a un modello "a cascata" ripetuto procedendo per affinamenti successivi dove però il numero di ripetizioni dell'iterazione viene preventivamente pianificato.
Sinteticamente, le attività delle quattro fasi sono le seguenti.

1. Pianificazione, in cui si determinano degli obiettivi, delle alternative e i vincoli associati al progetto. Il committente e il fornitore del sistema interagiscono allo scopo di definire in maniera sufficientemente univoca cosa deve essere realizzato e come. In questa fase è buona norma redigere dei documenti, in principio non eccessivamente dettagliati, che fissino i punti fondamentali della pianificazione del lavoro futuro.
2. Analisi dei rischi, in cui si identificano e si analizzano i problemi e i rischi associati al progetto, al fine di determinare delle strategie per controllarli. Tra i rischi che devono essere presi in considerazione si annoverano i fattori di costo, di tempo e di variazione delle specifiche.
   I rischi più evidenti da valutare sono quelli di carattere economico, facendo riferimento ai **costi** di realizzazione, di gestione e di esercizio. Al di là delle considerazioni strettamente economiche, un parametro critico durante l'analisi dei rischi è il **tempo**. Un altro dei rischi si riferisce alle **variazioni delle specifiche** che rende il prodotto non più adatto alle esigenze del committente; a questo proposito, un esempio è quello dell'obsolescenza prematura in cui il committente richiede un prodotto i cui servizi siano già stati superati in efficienza, affidabilità e rapidità da altri sviluppati più di recente.
3. Sviluppo, in cui si procede alla fase vera e propria della realizzazione: i tempi di realizzazione di quest'attività, che comprende sia la codifica sia la verifica, sono tra i più lunghi tra tutti quelli previsti all'interno del ciclo di vita del prodotto software e non è infrequente che il prodotto finale presenti dei difetti di codifica o di aderenza ai requisiti.
4. Valutazione, in cui il committente valuta se il sistema realizzato risponde alle sue esigenze. Attraverso questa fase il committente verifica che il prodotto soddisfi effettivamente i requisiti richiesti. Una logica conseguenza del fatto che un prodotto software non superi la fase di validazione dei requisiti è la necessità di impostare un nuovo ciclo di attività in cui definire più chiaramente – o ridefinire del tutto – i requisiti non realizzati e passare a una ulteriore sessione di analisi dei rischi, di sviluppo e di valutazione.

In base alla pianificazione e all'analisi dei rischi, si sceglie il modello di sviluppo che meglio si presta al soddisfacimento dei requisiti dell'utente.
Si sottolinea nuovamente che la forma della spirale del modello indica la possibilità di sviluppare il sistema in un solo colpo, fermandosi al primo giro e adottando quindi un modello "a cascata", oppure di sviluppare il sistema in modo incrementale anche adottando un modello basato sullo sviluppo di prototipi, "proseguendo" cioè per più di un giro.
La scelta del modello dipende sia dai requisiti del sistema sia dalle capacità dell'azienda produttrice; per esempio, un sistema di grandi dimensioni potrebbe amplificare le pro-

blematiche connesse a un ciclo di vita che segue il modello "a cascata" poiché i tempi di rilascio e di valutazione potrebbero essere inaccettabili.

Il modello a spirale è nato dall'esperienza e dalla constatazione che quasi sempre un prodotto software "apparentemente finito" ha reso necessaria una revisione o integrazione ancora prima di essere messo in funzione, cioè ha dovuto essere sottoposto a una pianificazione di un nuovo ciclo. I prodotti software più complessi vi passano moltissime volte e committente e fornitore devono essere preparati ad affrontare un'evenienza di questa natura tenendone conto durante l'analisi dei rischi. Si ricordi anche che un nuovo ciclo di attività può essere intrapreso anche dopo molto tempo dal rilascio di un prodotto finito, validato e perfettamente funzionante, quando si rendano necessari degli aggiornamenti di dimensioni e impatto più o meno rilevanti.

Metodologie agili

Nel corso degli anni '90 è emerso un certo numero di metodologie di sviluppo software, cosiddette "agili". Un sinonimo per questa accezione della parola agile, potrebbe essere "leggero" (*lightweight*): tali metodologie si contrappongono infatti a quelle cosiddette pesanti (*heavyweight*), come la waterfall o quella iterativa, proponendo modalità di lavoro fortemente orientate ai risultati.
La "leggerezza" di queste discipline è relativa all'impegno da profondere per il raggiungimento del risultato: invece che pianificare, progettare e sviluppare tutto il sistema software, che oltre alle funzionalità richieste cerchi anche di anticipare quelle future, le discipline agili si focalizzano sul raggiungimento di un risultato alla volta, piccolo e ben definito, costruendo con un processo iterativo il sistema completo.

L'idea di base nelle metodologie agili è che queste non sono predittive, non cercano di prevedere come evolverà il sistema software, ma sono adattative, ossia propongono valori e pratiche per meglio adattarsi all'evoluzione dei requisiti utente, prima che del sistema software.

Nei progetti software le esigenze del cliente sono in costante mutamento: sebbene l'avere in anticipo tutti i requisiti utente sia un aspetto desiderabile per realizzare progetti ben fatti, spesso ciò non è possibile e quindi la teoria si basa su un fondamento erroneo, cioè che "non è possibile o è estremamente difficile ottenere a priori tutte le specifiche in modo completo". In questi casi, non si può utilizzare una metodologia predittiva, perché il processo non è completamente prevedibile. Anche se fossero disponibili subito tutti i requisiti, e anche se fossero tutti coerenti, anche la fase di progettazione potrebbe dare dei problemi: è molto difficile, per esempio, catturare il sistema in un modello UML che sia completo e corretto e che possa essere utilizzato come base per lo sviluppo del codice. In contrapposizione alle metodologie pesanti, quelle agili si basano su un processo iterativo composto da fasi di progettazione, sviluppo e test molto più brevi: il progetto e il codice sono impostati per privilegiare l'adattabilità alle modifiche, piuttosto che per soddisfare pienamente e in modo pianificato tutte le specifiche.

Le tre principali caratteristiche delle metodologie agili sono:

- iterattività e incrementalità: le fasi di pianificazione, progettazione, sviluppo e test sono compresse in tempi molto più ridotti di quanto accade nelle metodologie tradizionali, concentrandosi sulla soluzione di pochi problemi, piccoli e ben definiti;
- rilasci frequenti: grazie al procedimento composto da piccoli passi, il team di sviluppo è in grado di produrre versioni del software in tempi più ridotti e i rilasci risultano quindi più frequenti;
- testing: una delle pietre miliari delle discipline agili è il testing, ossia la verifica del corretto funzionamento del sistema, che si applica sia al codice sia ai dati sia agli altri prodotti a cui le diverse discipline sono orientate (come i modelli o la documentazione).

Nelle metodologie agili vengono privilegiate le persone annziché i processi: il concetto si basa sul fatto che se una persona è felice di lavorare, produrrà un sistema di qualità superiore e in meno tempo; contrariamente, chi è obbligato a svolgere attività sgradite avrà problemi di concentrazione, qualità e risultati. Le pratiche e i valori delle discipline agili non sono stati inventati di sana pianta dai rispettivi autori, ma derivano dal comune buon senso, anche se vengono in certi casi portate all'estremo.

Le discipline agili non sono sempre applicabili a tutti i progetti: sono indicate in progetti dove i requisiti utente cambiano continuamente, e il team di sviluppo ne deve seguire in tempo quasi reale l'evoluzione.

Le metodologie agili fanno riferimento al testo pubblicato nel 1999 da Kent Beck, che definisce la programmazione estremizzata (*extreme programming*), una metodologia di sviluppo che insegue la semplicità eliminando la maggior parte dei vincoli formali di una metodologia di sviluppo tradizionale, dove vengono indicate le seguenti regole pratiche.

REGOLA	DESCRIZIONE
Pianificazione realistica	I clienti devono prendere le decisioni sulla funzionalità, i programmatori devono prendere le decisioni tecniche. Aggiornate il piano di sviluppo quando è in conflitto con la realtà
Piccoli stati di avanzamento	Fornite velocemente un sistema utilizzabile, e fornite aggiornamenti in tempi brevi
Metafora	Tutti i programmatori dovrebbero condividere un racconto che illustri il sistema in fase di sviluppo
Semplicità	Progettate ogni cosa in modo che sia la più semplice possibile, invece di predisporre tutto per future complessità
Collaudo	Sia i programmatori sia i clienti devono preparare casi di prova. Il sistema deve essere collaudato continuamente
Riprogettazione	I programmatori devono continuamente ristrutturare il sistema per migliorare il codice ed eliminare parti duplicate
Programmazione a coppie	I programmatori devono lavorare a coppie e ciascuna coppia deve scrivere codice su un unico calcolatore
Proprietà collettiva	Tutti i programmatori devono poter modificare qualsiasi porzione di codice quando ne hanno bisogno
Integrazione continua	Non appena un problema è risolto, mettete insieme l'intero sistema e collaudatelo

Settimana di 40 ore	Non usate piani di lavoro poco realistici, riempiendoli di sforzi eroici
Cliente a disposizione	Un vero utilizzatore del sistema deve essere disponibile in qualsiasi momento per la squadra di progettazione
Standard per la scrittura del codice	I programmatori devono seguire degli standard di codifica che pongano l'accento sul codice autodocumentato

Molte di queste regole pratiche vengono dal senso comune, mentre altre, come il requisito di programmare a coppie, sono sorprendenti: Beck afferma che la potenza della Programmazione estremizzata consista nella sinergia tra queste regole pratiche, cioè che la somma è maggiore degli addendi.

Conclusioni

Dall'analisi di ciascuno dei modelli sopra illustrati è possibile notare come il processo di sviluppo di un'applicazione informatica è sempre scomposto in una sequenza di fasi successive, ognuna delle quali ha lo scopo di trasformare la descrizione di un problema da una versione a un'altra, di consentire l'analisi e la verifica della conformità dei comportamenti del software ai requisiti dell'utente e di consentire un miglioramento, correzione e adattamento dell'applicazione sviluppata.
Il prodotto finale viene realizzato "per affinamenti successivi", cioè con ripetuti tentativi, e quindi un modello a spirale ha maggiori possibilità di successo in quanto a ogni iterazione avviene il confronto e il controllo con l'utente e le eventuali rettifiche in corso d'opera vengono introdotte gradualmente senza il rischio di dover rimettere mano solo a prodotto finito, per "rifare" parti del sistema con il rischio di danneggiare l'intero il progetto.
Il rischio del modello incrementale è quello "psicologico" dei progettisti che, sapendo di poter effettuare una successiva iterazione, a volte "rimandano" e posticipano la realizzazione di alcuni elementi "cruciali", ritardando i termini di consegna del prodotto con l'introduzione di numerose iterazioni a volte superflue.

Verifichiamo le conoscenze

Esercizi a scelta multipla

1 In cche osa consiste la *seniority*?

- ○ Nell'età dei programmatori
- ○ Nell'età degli analisti
- ○ Nell'esperienza di progettazione
- ○ Nell'uso delle metodologie

2 Ordina cronologicamente secondo il modello a cascata

- ○ Codifica (realizzazione)
- ○ Progettazione
- ○ Pianificazione del sistema
- ○ Test e debug
- ○ Analisi
- ○ Manutenzione
- ○ Installazione, verifica e collaudo

3 La manutenzione presenta i seguenti aspetti (indicare quello errato):

- ○ correttivo
- ○ migliorativo
- ○ qualitativo
- ○ adattativo

4 Quale tra le seguenti fasi non rientra nel modello a prototipazione rapida?

- ○ Raccolta e rifinitura dei requisiti
- ○ Progetto rapido
- ○ Costruzione del prototipo
- ○ Valutazione del prototipo
- ○ Rifinitura del prototipo
- ○ Definizione delle interfacce
- ○ Ingegnerizzazione del prodotto

5 Quale tra le seguenti fasi non rientra nel modello a spirale?

- ○ Analisi dei rischi
- ○ Analisi dei benefici
- ○ Sviluppo
- ○ Valutazione
- ○ Pianificazione

6 I rischi nel modello a spirale sono riferiti

- ○ all'aspetto economico
- ○ all'aspetto temporale
- ○ all'aspetto giuridico
- ○ all'aspetto tecnologico

Test vero/falso

1	Nel modello a cascata, ogni singola attività viene completata prima del passaggio alla successiva.	V F
2	Nel modello a cascata è prevista la fase di revisione del progetto.	V F
3	Il processo a prototipazione rapida di sviluppo inizia con la fase di raccolta dei requisiti.	V F
4	Nel modello incrementale ogni fase è un incremento della precedente.	V F
5	L'analisi dei rischi è fondamentale nel modello incrementale.	V F
6	Tra i rischi non è da trascurare la capacità del gestore del progetto.	V F
7	Il modello a spirale è stato definito nel 1998.	V F
8	Ogni modello di sviluppo migliora il modello precedente.	V F

UNITÀ DIDATTICA 2

UN NUOVO MODELLO DI SVILUPPO

IN QUESTA UNITÀ IMPAREREMO...

- il concetto di programmazione di sistema
- la motivazione della crisi del software
- il concetto di classe, oggetto, incapsulamento, ereditarietà e polimorfismo
- il concetto di astrazione, implementazione, interfaccia

Introduzione

La programmazione a oggetti (OOP, *Object Oriented Programming*) rappresenta, senza dubbio, il modello di programmazione più diffuso e utilizzato nell'ultimo decennio.
Nasce alla fine degli anni '80 come superamento della programmazione di tipo procedurale, ma oltre che essere una naturale evoluzione dei linguaggi di programmazione strutturati è anche una vera e propria rivoluzione in quanto sposta completamente il punto di vista dal quale si affronta il progetto di un'applicazione.
Come ogni rivoluzione, anche la "rivoluzione informatica" è conseguenza dell'insoddisfazione per l'inadeguatezza degli strumenti software a disposizione degli sviluppatori di fronte alla repentina evoluzione dei sistemi hardware degli anni '90.
Vediamo brevemente di comprenderne le cause, in modo da avvicinarci all'OOP seguendo le tappe che l'hanno generata.

Partiamo da un termine che ben conosciamo, ma che nel tempo ha modificato il suo significato: "programmare".
Riconsideriamo le fasi di realizzazione di un programma, e cioè:

- analisi del problema;
- definizione della strategia;
- codifica (programmazione);
- installazione;
- collaudo.

L'attività di programmazione si identifica con la fase di codifica, che è un'attività sicuramente poco stimolante, che segue le fasi creative e viene percepita come una vera e propria "manovalanza", compiuta da chi trascrive in un linguaggio di programmazione l'algoritmo progettato da altri.

Negli anni '80 ci si trova di fronte a un insieme di problematiche che causano la prima vera "crisi del software": l'evoluzione esponenziale dell'hardware e la riduzione dei costi sia delle macchine sia dei sistemi operativi ha reso *sproporzionati e preponderanti* i costi di sviluppo e manutenzione del software applicativo, nella componente sia correttiva (per eliminare errori) sia adattativa (per rispondere a nuove esigenze).

Inoltre, la metodologia di sviluppo orientata a una visione "algoritmica" del problema informatico (programmazione *in the-small*) si manifesta insoddisfacente, se non addirittura catastrofica se utilizzata e adattata alla progettazione, allo sviluppo e alla manutenzione di sistemi software complessi.

Zoom su...

CARATTERISTICHE DEL SISTEMA SOFTWARE

In un sistema, non basta che il software funzioni, deve essere anche "ben fatto" e presentare le seguenti caratteristiche principali:

1. ben organizzato
2. modulare
3. protetto
4. riusabile
5. riconfigurabile
6. flessibile
7. documentato
8. incrementalmente estendibile
9. a componenti
10. …

Si passa allora dalla "programmazione di un programma" alla "programmazione di un sistema", quindi al concetto di base del termine "programmare" si attribuisce un nuovo significato. Anche l'attività di pura codifica, intesa come scrittura di istruzioni, assume una nuova dimensione in quanto deve essere integrata al resto del progetto di sistema e quindi richiede ai progettisti nuove (e a volte maggiori) competenze, quali:

- la conoscenza di fondamenti teorici;
- la padronanza degli strumenti disponibili;
- la visione sistemistica del problema;
- la capacità di analisi dello spazio dei problemi;
- la capacità di ragionamento sul lavoro svolto;
- la capacità di comunicazione con gli utilizzatori;
- la capacità di rinnovarsi (aggiornarsi e modificare le proprie convinzioni).

Riassumendo, il nodo centrale è il fatto che cambia sostanzialmente la dimensione del problema: il termine programmare un elaboratore perde il significato che aveva avuto in precedenza e diventa sinonimo di progettare e costruire sistemi software.

Per progettare sistemi occorrono però linguaggi che offrano metafore e concetti sistemistici: infatti, in ogni linguaggio di programmazione, oltre alle regole sintattiche e semantiche e a strutture di dati, è intrinsecamente presente anche un insieme di metafore e concetti che agevolano, o meglio, "instradano" le scelte progettuali del programmatore che utilizza il linguaggio.

ESEMPIO 1

Ricordiamo per esempio il concetto di file: sicuramente è presente nei vari linguaggi ma viene concepito e utilizzato in modo diverso, dato che ogni linguaggio offre strumenti e approcci diversi per la sua gestione e l'utilizzo dei file, a volte anche in modi "trasparenti all'utente".

Si pensi poi, per esempio, come viene o meno considerata l'interfaccia grafica e quali modalità operative sono rese disponibili per essa nei diversi linguaggi: in un linguaggio che prevede solo la gestione dello schermo in modo testuale è improponibile (o meglio, "folle") pensare di implementare un'applicazione grafica con puntatori grafici da spostare con il mouse, pulsanti e finestre!

Ciascun linguaggio offre propri strumenti per migliorare l'espressività del programmatore (come l'organizzazione in funzioni, procedure, moduli ecc.), questi strumenti lo caratterizzano o lo rendono diverso dagli altri, anche se appartenente allo stesso paradigma.

Le diversità delle metafore e dei concetti intrinseci nei diversi linguaggi sono probabilmente la vera ragione dell'esistenza di tanti linguaggi di programmazione, indipendentemente dal paradigma al quale fanno riferimento: un linguaggio è sicuramente più adatto di un altro per risolvere applicazioni in un determinato ambito o settore, a volte anche per un solo specifico problema o parte di esso, offrendo specifiche primitive o di alto livello che permettono di "risparmiare tempo e fatica" allo sviluppatore; d'altra parte, purtroppo, troppo spesso un programmatore si "innamora" di un linguaggio (il primo!) e utilizza sempre lo stesso… anche perché *conosce solo quello* e quindi non è in grado di scegliere quello più adatto per quel particolare problema.

Perché tanti linguaggi di programmazione

Ogni linguaggio di programmazione ha quindi un "ruolo" o un settore di applicazione per il quale è più indicato; inoltre, tutti i linguaggi classici, cioè quelli noti come "Pascal like" in quanto si ispirano al linguaggio Pascal e alla programmazione imperativa, non offrono strumenti adatti alla progettazione di sistemi complessi.

Sono quindi necessari nuovi linguaggi, più aderenti alle nuove situazioni e più adatti a risolvere nuove problematiche: i linguaggi a oggetti sono tra questi e sono oggi utilizzati in tutte le fasi di sviluppo dei sistemi software (analisi, progetto, implementazione) perché introducono una nuova visione del sistema offrendo un insieme di concetti e metafore capaci di integrare e rinnovare le tre fasi classiche dello sviluppo.

Questi linguaggi aiutano non solo la codifica di particolari situazioni, ma modificano "il modo di pensare" e vedere i problemi, richiedendo quindi un approccio mentale diverso rispetto ai linguaggi procedurali.

Per tale motivo la programmazione a oggetti è da considerarsi un nuovo paradigma che va ad affiancarsi ai tre paradigmi classici, che sono:
1. imperativo (Modula-2, Pascal, Cobol, Ada, Basic, C, Fortran, Algol);
2. funzionale (Lisp, Scheme, ML);
3. logico (Prolog).

◀ Il termine legacy definisce tutti i casi in cui un'applicazione "obsoleta" continua a essere usata poiché l'utente (tipicamente un'organizzazione) non vuole o non può sostituirla in quanto i costi sarebbero sproporzionati rispetto alle migliori prestazioni che si otterrebbero. ▶

I linguaggi di questo nuovo paradigma presentano importanti caratteristiche:
- utilizzano e integrano le metodologie di sviluppo top-down e bottom-up;
- sono tra le sorgenti d'ispirazione del concetto di componente software;
- aiutano a integrare computazione e interazione;
- aiutano ad affrontare il problema della sostituzione del software obsoleto ◀ legacy ▶.

Il nuovo paradigma, il quarto, prende nome di paradigma a oggetti. Contrariamente a quanto si possa pensare, i primi linguaggi "a oggetti" furono SIMULA I e SIMULA 67, sviluppati da Ole-Johan Dahl e Kristen Nygaard all'inizio degli anni '60, ma ai quei tempi non riscossero particolare successo.

Il primo vero linguaggio a oggetti "puro" usato a tutt'oggi fu il linguaggio SmallTalk, introdotto negli anni '70 inizialmente da Alan Kay e successivamente ripreso da Adele Goldberg e Daniel Ingalls, entrambi ricercatori allo Xerox Park di Palo Alto, in California.
Oggi i linguaggi a oggetti più rappresentativi sono Java e C++.

I linguaggi a oggetti sono i principali artefici della transizione da progettazione e programmazione "in the small" a progettazione e programmazione "in the large".

Crisi, dimensioni e qualità del software

Abbiamo già accennato alla "crisi del software" che negli anni '80 ha rappresentato un serio problema, al punto di mettere in crisi più volte il "sistema informatica" (spesso ridicolizzando i progettisti e gli operatori del settore).

LEGGI CATASTROFICHE

In quegli anni sono nate leggi, corollari e teoremi ormai famosi, dalla "legge di Murphy" alla "legge dei mille programmatori", dalla "sindrome del 90%" alle leggi "di Mayers e di Mealy", che riportiamo di seguito per "dovere di cronaca".

Altro non si tratta che di "battute cattive" nate con una buona dose di cinismo e goliardia, che in comune avevano come fonte di ispirazione reale il fatto che i sistemi informatici erano "pieni zeppi di errori".
Legge di Murphy: se c'è una remota possibilità che qualcosa vada male, sicuramente ciò accadrà e produrrà il massimo danno.
Legge di Murphy (corollario 1): se c'è una possibilità che varie cose vadano male, quella che causa il male peggiore sarà la prima a verificarsi.
Legge di Murphy (corollario 2): gli stupidi sono sempre più ingegnosi delle precauzioni che si prendono per impedire loro di nuocere.
Legge dei mille programmatori: se assegnate mille programmatori a un progetto senza aver ben definito il disegno del sistema, otterrete un sistema con almeno mille moduli.
Sindrome del 90%: colpisce gli addetti ai lavori quando un progetto di un prodotto software è lì lì per essere completato, ma nonostante gli sforzi, continua a restare drammaticamente incompiuto: c'è chi afferma che il primo 90% di un lavoro viene svolto nel 10% del tempo e il restante 10% nel restante 90% (sigh!).
Legge di Mayers: è bene trascurare le fasi di analisi e progetto e precipitarsi all'implementazione allo scopo di guadagnare il tempo necessario per rimediare agli errori commessi per aver trascurato la fase di analisi e di progetto.
Legge di Mealy: se un progetto ritarda, aggiungendo una nuova persona al gruppo questa consuma più risorse di quante ne produce.

Volendo ricercare le motivazioni della crisi del software e prendendo spunto anche dalle "leggi" esposte sopra, è possibile individuare tre cause fondamentali:
1. crisi dimensionale;
2. crisi gestionale;
3. crisi qualitativa.

La crisi dimensionale ha origini di natura essenzialmente tecnologica: è dovuta cioè alla crescita della dimensione dei sistemi informatici con la diffusione di un numero sempre maggiore di computer e della maggiore connessione degli stessi in reti locali e remote.
I nuovi sistemi non hanno solo subìto un aumento delle "dimensioni fisiche", ma hanno provocato nuovi problemi e, di conseguenza, le entità delle astrazioni, dei modelli e degli strumenti per progettare si sono rivelati inadeguati.

La crisi gestionale ha origine dalla crescita sproporzionata dei costi per effettuare la manutenzione del software, sia correttiva sia adattativa, legata al variare delle dimensioni dei sistemi e quindi dei programmi. Nei sistemi di medie e grandi dimensioni trovare gli errori può essere molto difficile e oneroso e ogni modifica può coinvolgere tutto il team di sviluppo.

La crisi qualitativa è anch'essa legata sia alla crescita delle richieste sia al fatto che gli strumenti a disposizione dei programmatori sono risultati insufficienti per ottenere software di qualità e proprietà desiderate.
I linguaggi imperativi e la programmazione strutturata, con le loro strutture dati e le strutture di controllo, le funzioni e le procedure, non si sono dimostrati strumenti efficaci anche perché in essi non sono presenti modalità adatte a offrire supporto all'organizzazione del processo produttivo del software inteso come prodotto di qualità.

Il software di buona qualità deve essere protetto, modulare, riusabile, documentato, incrementalmente ed estendibile: cambiando le dimensioni del problema cambia la dimensione del progetto, ma soprattutto la modalità con cui si deve effettuare l'approccio alla soluzione; ne consegue la nascita dell'ingegneria del software, con nuovi strumenti progettuali e nuovi linguaggi implementativi.

La programmazione a oggetti si propone come una nuova metodologia, con un approccio risolutivo completamente diverso: l'approccio a oggetti. L'utilizzo di linguaggi orientati agli oggetti costringe il programmatore ad adottare nuove metodologie di programmazione, anche inconsapevolmente, sviluppando spesso in modo automatico le attitudini mentali adatte all'approccio sistemistico.
L'impiego di un linguaggio OOP impone l'utilizzo dei principi fondamentali della programmazione a oggetti, che si possono riassumere in modularità, incapsulamento, protezione dell'informazione, che sono le caratteristiche richieste al software di buona qualità: il programmatore "automaticamente" si trova a rispettare le tecniche fondamentali di organizzazione del software necessarie per superare tutte e tre le possibili cause di "crisi" prima descritte.

Astrazione, oggetti e classi

Con la programmazione a oggetti è possibile modellare la realtà in modo più naturale e vicino all'uomo, offrendo nuovi strumenti con cui poter descrivere tutti i livelli di astrazione nei quali viene "trasformato" il sistema software nelle varie fasi della sua progettazione.
L'astrazione è il primo importante strumento di lavoro nella fase di progettazione di un sistema software (oltre che essere un concetto importante in tutte le attività dell'ingegneria), e ne ricordiamo una possibile definizione.

ASTRAZIONE
L'astrazione è il risultato di un processo, detto appunto di astrazione, secondo il quale, assegnato un sistema, complesso quanto si voglia, si possono tenerne nascosti alcuni particolari evidenziando quelli che *si ritengono essenziali* ai fini della corretta comprensione del sistema.

Hoare, nel suo libro *Notes on Data Structuring*, sottolinea questo aspetto e associa il concetto di astrazione a quello di oggetto.

Nel processo di comprensione di fenomeni complessi, lo strumento più potente disponibile alla mente umana è l'astrazione. L'astrazione nasce dall'individuazione di proprietà simili tra certi oggetti, situazioni o processi del mondo reale e dalla decisione di concentrarsi su queste proprietà simili e di ignorare, temporaneamente, le loro differenze.

Gli oggetti simili hanno comportamenti uguali (metodi) e caratteristiche specifiche (attributi) che li differenziano tra loro: sono raccolti in classi, dove ogni classe è l'origine degli oggetti stessi e in essa vengono descritti i comportamenti e le proprietà degli oggetti.

Per essere precisi si dovrebbe parlare di programmazione per classi piuttosto che di programmazione per oggetti in quanto l'oggetto è un elemento di una classe, cioè ne è proprio una sua istanza, come vedremo meglio in seguito.

La classe è l'elemento base della OOP e un programma a oggetti è costituito da un insieme di classi che generano oggetti che tra loro comunicano con scambi di messaggi.
Spesso le classi non hanno bisogno di essere scritte in quanto sono di dominio pubblico, cioè disponibili come moduli di libreria, e il programmatore deve solamente utilizzarle come veri e propri "mattoni software" con cui costruire il programma.

Se una classe non soddisfa appieno le esigenze specifiche di un caso particolare è possibile completarla, aggiungendole tutto quello che necessita per la nuova esigenza: si parla di classe ereditata e l'ereditarietà è la vera potenzialità della OOP, che permette di modellare e specializzare le classi già esistenti e quindi di non dover mai "partire da zero" ma sempre da solide fondamenta.

Se riprendiamo alcuni concetti fondamentali già discussi in passato scopriamo che:
- è inutile riscrivere software che qualcuno ha già scritto (anche perché probabilmente lo ha scritto meglio di noi!);
- un programma deve essere in grado di adattarsi alla complessità del problema e del sistema senza dover intervenire radicalmente sul codice;
- un programma deve essere in grado di adattarsi alla tecnologia: per esempio, se viene modificata la rappresentazione in memoria occorre modificare solamente le operazioni e non l'intero programma.

Ci accorgiamo che la programmazione a oggetti è la risposta a ogni esigenza.

Sappiamo che cosa vuol dire modificare software già scritto: spesso non riusciamo neppure a capire quello che abbiamo scritto noi, immaginiamo che cosa può succedere se dobbiamo rielaborare programmi scritti da altri!

Il grande vantaggio della OOP è che le classi disponibili per gli sviluppatori "funzionano", cioè il codice è testato e completamente esente da errori, e sono "blindate", cioè non è possibile modificare il loro interno ma solo utilizzarle o estenderle in nuove classi più complete con aggiunte personali.

Le classi sono veri "circuiti integrati software" e il programmatore a oggetti a volte diventa un vero "assemblatore di software": naturalmente il sogno di ogni programmatore è quello di avere a disposizione tutti i componenti di cui ha bisogno ma, ovviamente, nella stragrande maggioranza dei casi questo non è possibile.

Il progettista, mediante l'astrazione, cerca di individuare negli oggetti che gli sono necessari comportamenti simili presenti e descritti in elementi di altre classi già esistenti, per poi completare e personalizzare la propria situazione.
Potremmo concludere con una frase apparentemente oscura ma che riassume l'intima essenza della programmazione per oggetti: "si astrae per specializzare".

Dov'è la novità?

Si potrebbe dire che di astrazione e modularità si è già parlato a lungo, così come di scomposizione top-down, approccio bottom-up, tecniche di *divide et impera* ecc., quindi non è chiaro che cosa realmente ci sia di nuovo nella OOP!
Si è inoltre già visto come sia possibile effettuare anche con i linguaggi della programmazione operativa l'implementazione di tipi di dati astratti (ADT) che costituiscano una rappresentazione concreta delle astrazioni concepite dal progettista.

La OOP va invece oltre, realizzando quello che comunemente prende il nome di *information hiding*, cioè l'incapsulamento, all'interno della classe, sia della rappresentazione dei dati sia delle elaborazioni che possono essere eseguite su di essi, fornendo all'utente un meccanismo di interfacciamento che garantisce il mascheramento del funzionamento interno dell'oggetto: solo attraverso l'interfaccia si interagisce con la classe e la classe stessa comunica all'esterno solo per "cosa fa" e non per "come lo fa".

> In tale modo si garantisce il rispetto della semantica delle astrazioni del dato: obbligando ad accedere a esso solo tramite apposite funzioni (metodi) si impedisce l'accesso diretto alla rappresentazione concreta del dato.

L'oggetto estende il concetto di dato astratto e per la definizione di una classe è necessario:
- definire tutte le operazioni che sono applicabili agli oggetti (istanze) del tipo di dato astratto;
- nascondere la rappresentazione dei dati all'utente;
- garantire che l'utente possa manipolare gli oggetti solo tramite operazioni del tipo di dato astratto a cui gli oggetti appartengono (protezione).

Un ulteriore vantaggio dell'incapsulamento è quello di obbligare il progettista a separare il "che cosa c'è dentro" dal "che cosa si vede dal di fuori" durante tutta l'attività di realizzazione di un'applicazione.
- Il sistema esterno è ciò che vede l'utente, quindi quello che deve soddisfare le aspettative definite dal contratto: al cliente non interessa che cosa c'è dentro, interessa che ciò che ha acquistato funzioni e come si utilizza, cioè come si interagisce con esso (interfaccia); il progettista deve quindi focalizzarsi sul funzionamento osservabile (astrazione).
- Il sistema interno non deve essere accessibile all'utente, per cui ci si deve focalizzare sull'implementazione, per fare in modo che tutti i dettagli restino invisibili all'esterno del sistema stesso (incapsulamento).

Ricapitolando, possiamo riassumere le caratteristiche dei due elementi fondamentali che costituiscono l'incapsulamento:
- interfaccia: esprime una vista astratta di un ente computazionale (funzione, procedura, componente software ecc.), nascondendone l'organizzazione interna e i dettagli di funzionamento;
- implementazione: esprime la rappresentazione dello stato interno ed è costituita dal codice che realizza il funzionamento richiesto.

Conclusione: che cos'è la programmazione a oggetti

Si è detto che la programmazione OOP nasce alla fine degli anni '80 come superamento della programmazione di tipo procedurale e per ovviare ai limiti propri che questa aveva.
Gli oggetti sono raccolti in classi, che definiscono campi e metodi comuni a tutti gli oggetti di un certo tipo, e la OOP offre la possibilità di estendere queste classi (nesting) creando vere e proprie gerarchie mediante l'ereditarietà (che permette di propagare automaticamente attributi comuni a più oggetti di una stessa classe).
Mediante il procedimento di astrazione, gli oggetti fisici vengono virtualizzati in oggetti software che offrono le stesse informazioni e servizi degli oggetti reali: si suddivide il sistema reale preso in esame in classi di oggetti, ognuna delle quali possiede proprie variabili e funzioni, che assumono il nome di attributi e metodi.

Zoom su...

METODI

I metodi possono essere di tre tipi, classificati in base al tipo di operazioni che eseguono:
1. visualizzatori per accedere ai valori delle variabili;
2. modificatori per modificare i valori delle variabili;
3. costruttori per creare gli oggetti della classe.

Nella OOP l'information hiding viene realizzato mediante l'incapsulamento ed è possibile accedere ai dati interni all'oggetto solo utilizzando i metodi previsti dal programmatore: l'utente ha a disposizione un'interfaccia che gli permette di interagire con gli oggetti non in modo diretto, ma esclusivamente mediante lo scambio di messaggi.
Un'ulteriore caratteristica della OOP è quella che prende il nome di polimorfismo: si tratta della possibilità di applicare lo stesso operatore a classi diverse, cioè mandare un messaggio con un "medesimo significato" a classi diverse che però lo interpretano e lo eseguono in maniera differente, compatibile con la loro struttura.

OOP IN SINTESI

Ereditarietà, incapsulamento e polimorfismo sono la "triade" di strumenti nuovi, da comprendere e sfruttare, per realizzare la programmazione di sistemi modulari, di buona qualità, con dimensioni scalabili, di facile manutenzione e riutilizzabili.

Verifichiamo le conoscenze

Esercizi a scelta multipla

1 L'acronimo OOP significa:

- ○ *Object Objective Programming*
- ○ *Objective Object Processing*
- ○ *Object Oriented Programming*
- ○ *Oriented Object Processing*

2 Quale tra le seguenti non è una caratteristica di un sistema software?

- ○ ben organizzato
- ○ modulare
- ○ protetto
- ○ semplice
- ○ riusabile
- ○ riconfigurabile
- ○ flessibile

3 Quale tra i seguenti non è un paradigma di programmazione?

- ○ imperativo
- ○ funzionale
- ○ matematico
- ○ logico

4 Quale fu il primo linguaggio a oggetti?

- ○ Ada
- ○ Java
- ○ Simula I
- ○ C++
- ○ SmallTalk

5 Le crisi del software furono di natura (indicare quella errata):

- ○ dimensionale
- ○ gestionale
- ○ qualitativa
- ○ quantitativa

6 Quale tra le seguenti affermazioni è errata?

- ○ i metodi corrispondono ai programmi
- ○ gli attributi corrispondono alle variabili
- ○ un oggetto è un'istanza di una classe
- ○ una classe è "un mattone software"

7 I metodi possono essere:

- ○ costruttori
- ○ distruttori
- ○ modificatori
- ○ visualizzatori

8 L'incapsulamento consiste:

- ○ nell'astrarre gli oggetti
- ○ nel fare in modo che i dettagli restino invisibili all'esterno
- ○ nel propagare automaticamente attributi comuni
- ○ nel realizzare il *divide et impera*

9 Quale tra i seguenti non è un concetto OOP?

- ○ incapsulamento
- ○ automatismo
- ○ polimorfismo
- ○ ereditarietà

Test vero/falso

1	Il primo linguaggio a oggetti fu sviluppato negli anni '90.	V F
2	Con il termine "legacy" si intende un'applicazione OOP.	V F
3	L'astrazione ha un ruolo fondamentale nelle OOP.	V F
4	SmallTalk fu introdotto negli anni '70 ed è il primo linguaggio "puro".	V F
5	La classe è un insieme di oggetti.	V F
6	L'oggetto è un'istanza di una classe.	V F
7	Nella OOP le variabili corrispondono agli attributi.	V F
8	L'interfaccia serve per interagire con un oggetto.	V F
9	Nella OOP l'information hiding viene realizzato mediante l'ereditarietà.	V F
10	L'ereditarietà permette di propagare automaticamente attributi comuni.	V F

UNITÀ DIDATTICA 3

DOCUMENTAZIONE DI UN PROGETTO

IN QUESTA UNITÀ IMPAREREMO...

- a classificare classi e relazioni tra di esse
- a individuare le relazioni di ereditarietà, aggregazione e dipendenza tra le classi
- a capire l'utilizzo delle schede CRC per l'identificazione di classi
- a usare al meglio i diagrammi UML per descrivere le relazioni tra classi
- a usare la progettazione orientata agli oggetti per programmi complessi

Assistenza

Nel progetto informatico si è visto che le attività non terminano con la consegna e l'installazione del prodotto finito: buona parte dei problemi inizia proprio da quel momento.

Possiamo riassumere le attività che iniziano dalla "messa in opera" con il termine di assistenza che può essere di due tipi.

- Operativa: consiste nella formazione e assistenza on line allo scopo di permettere l'utilizzo del programma da parte degli utenti e la soluzione di tutti i problemi operativi che dovessero manifestarsi nell'utilizzo quotidiano del software.
- Tecnica: sia hardware, cioè connessa a malfunzionamenti e/o aggiornamenti dei macchinari, sia software, cioè la manutenzione del prodotto, già individuata come fase permanente di supporto al sistema dopo la consegna all'utente.

L'utente ha sempre bisogno di un supporto operativo, anche una volta terminata la fase di formazione, ed è fondamentale corredare il prodotto software con un insieme di strumenti che permettano l'autonomia operativa dell'utilizzatore anche per evitare continui interventi di formazione nonché lo spreco di ore in assistenza telefonica: il supporto necessario è una completa documentazione del programma.

Possiamo individuare tre diverse tipologie di documentazione riservata all'utente:
- documentazione inerente all'installazione e all'avvio dell'utilizzo del prodotto;
- documentazione interna: *help on line* e *sensitive help*;
- manuale d'uso: manuale cartaceo con la descrizione delle singole videate, le istruzioni su come eseguire tutte le operazioni offerte dal prodotto, i casi tipici di errore e le indicazioni di come effettuare "il recovery" dello stesso.

Spesso ne viene sottovalutata e trascurata l'importanza e ci si limita a stendere qualche "pagina di documentazione" alla fine del lavoro con delle note "di scarsa utilità", fatte più per dovere che per convinzione di reale utilità delle stesse.

DOCUMENTAZIONE

La documentazione è invece "parte integrante del progetto" e deve seguire fase per fase la realizzazione dello stesso, deve essere iniziata a partire dall'analisi e affinata e integrata costantemente durante tutto il lavoro.

Contemporaneamente alla documentazione utente, deve essere redatta anche la documentazione tecnica, indispensabile per poter effettuare ogni intervento di manutenzione sul software.

Anch'essa si compone di due parti:
- documentazione interna: direttamente dipendente dallo stile di programmazione utilizzato, dalla atomizzazione delle funzioni e dei metodi e dal commento interno a essi;
- documentazione esterna: il *ben noto* manuale tecnico che deve essere particolarmente esauriente in modo tale da consentire ad altre persone di intervenire a diversi livelli sullo stesso progetto informatico per modificarne eventuali funzionamenti, integrando in esso nuove funzionalità o semplicemente modificando quelle esistenti.

Tutti i programmatori "snobbano" questi aspetti e finché non si trovano essi stessi a dover mettere mano al lavoro di altri non ne comprendono completamente la reale importanza e necessità: una prima "presa di coscienza" del programmatore sui benefici derivanti dalla documentazione avviene quando deve mettere egli stesso mano al proprio codice, magari scritto qualche tempo prima, di cui non ha prodotto documentazione e di cui non ricorda bene il funzionamento.
Come sempre è l'esperienza, soprattutto quella negativa, che persuade il progettista ad adottare regole e modelli di sviluppo standardizzati.

L'importanza della documentazione che correda un progetto software aumenta ancora nel momento in cui ci si prefigge l'obiettivo di rendere portabile e riusabile il software prodotto nell'ambito del progetto.
Il paradigma della programmazione a oggetti ha nelle finalità principali proprio questo scopo: garantire un livello elevato di riuso dei componenti software ("black box"), cioè la possibilità di isolare specifiche funzionalità o servizi all'interno di appositi package o classi per riutilizzarle in altri prodotti oppure semplicemente sostituirle con nuovi "tasselli",

senza doversi preoccupare dell'impatto che tali modifiche possono avere sulle restanti componenti software.
La documentazione di un progetto *object oriented* avviene a tutti i livelli della fase di sviluppo anche supportata dall'utilizzo del linguaggio UML, come di seguito descritto.

UML

UML (*Unified Modeling Language*) è un linguaggio di modellazione potente, completo, semplice e naturale ormai universalmente riconosciuto come uno standard *de facto* a livello mondiale: consiste in un insieme di notazioni sviluppate a partire dagli anni '80 e consolidate in un apparato standard alla fine degli anni '90 con il nome, appunto, di UML da Grady Booch, Ivar Jacobson e James Rumbaugh che hanno definito le linee guida per l'impiego del linguaggio di modellazione per ogni tipo di sistema software; la sua maggior diffusione e notorietà, tuttavia, deriva dall'essere particolarmente adatto all'analisi e alla progettazione di sistemi software *object oriented*.
La documentazione di un progetto *object oriented* avviene in diverse fasi e a diversi livelli, a seconda sia dell'aspetto specifico che si intende descrivere sia del differente punto di vista da cui lo si descrive: si è già detto come i diagrammi di flusso (o flow chart) tipici della programmazione strutturata non sono soddisfacenti nella descrizione delle classi ma consentiranno di documentare in modo efficace i singoli metodi delle classi appartenenti al progetto.
Per la descrizione delle classi si sono introdotti i diagrammi delle classi (o class diagram), primo diagramma del linguaggio UML, che permetteranno di elencare in modo sintetico le classi che compongono il progetto, mettendo in evidenza gli attributi che le caratterizzano e i metodi resi disponibili per operare su di esse.
Successivamente, è necessario esplicitare le relazioni esistenti tra le classi stesse e la struttura gerarchica che viene realizzata attraverso la realizzazione di sottoclassi.
Sempre nel diagramma delle classi sono disponibili gli strumenti grafici per mettere in relazione tutte le classi.

L'UML si basa proprio su diagrammi e rappresentazioni grafiche: viene anche chiamato Visual Modeling in quanto mediante l'UML si realizza proprio la visualizzazione grafica del modello, utilizzando 9 diagrammi di base, facilmente comprensibili a tutte le componenti coinvolte nel progetto:

- cliente: chiede la soluzione del problema/esigenza;
- analista: studia e documenta le esigenze del cliente per i progettisti;
- disegnatori (progettisti): definiscono il progetto;
- programmatori: producono, installano e testano il software.

Il Visual Modeling prevede una continua interazione, scambio di vedute tra programmatori e analisti, tra progettisti e programmatori e tra cliente e analisti in modo da ridurre le possibilità di errore di progetto e di realizzazione.

Elenchiamo di seguito i principali vantaggi dell'utilizzo delle metodologia UML:

- il sistema SW viene progettato professionalmente;
- è documentato prima che venga scritto il codice;
- la scrittura del codice è più agevole ed efficiente: è più facile scrivere software riutilizzabile e si ottiene una considerevole diminuzione dei costi di sviluppo;

- i diagrammi UML sono chiari a tutti: è più facile avere una visione d'insieme e quindi sfruttare in modo migliore le risorse hardware;
- è più facile prevedere eventuali "buchi", in modo da ottenere il software che si comporterà come ci si aspetta;
- è più facile effettuare modifiche successive grazie alla documentazione;
- i costi di manutenzione diminuiscono;
- la comunicazione e l'interazione tra tutte le risorse umane che partecipano allo sviluppo sono molto più efficienti e dirette, fatte in modo uniforme, quindi senza incomprensioni con notevole risparmio di tempo.

Si è detto che UML utilizza nove diagrammi base, che sono:

1. Class Diagram: uno per ogni classe, completato dalle associazioni esistenti tra di esse;
2. Object Diagram: uno per ogni oggetto;
3. Use Case Diagram: descrizione del sistema visto dall'utente;
4. State Diagram: rappresentano gli stati e i loro cambiamenti nel tempo di ogni oggetto del sistema (da Start a End State);
5. Sequence Diagram: Class e Object Diagram sono statici: in questo diagramma sono evidenziate le dinamiche, basate sul tempo, delle varie interazioni tra gli oggetti;
6. Activity Diagram: descrive la sequenza delle attività riscontrate negli Uses Cases;
7. Collaboration Diagram: descrive la cooperazione tra gli elementi del sistema;
8. Component Diagram: ognuno nel team lavora su un componente del sistema;
9. Deployment Diagram: mostra l'architettura del punto di vista fisico e logistico del sistema.

La completa trattazione della metodologia UML esula da questo testo, ma nel seguito viene riportata a titolo di esempio la trattazione completa di come produrre il diagramma delle classi, primo punto del Modello UML, in modo da offrire una valutazione dei benefici ottenibili dall'utilizzo di tecniche standardizzate di sviluppo e documentazione del software.

Classi e associazioni tra le classi

Diagramma delle classi

Il diagramma delle classi produce un'immagine "statica" del progetto, identificando le classi che lo compongono e le relazioni che intercorrono tra di esse. Viene utilizzato *a posteriori* per documentare un programma realizzato, ma anche *a priori* per supportare il processo di progettazione e di identificazione delle componenti (classi, oggetti, package ecc.) in cui è opportuno suddividere il programma che si intende realizzare.

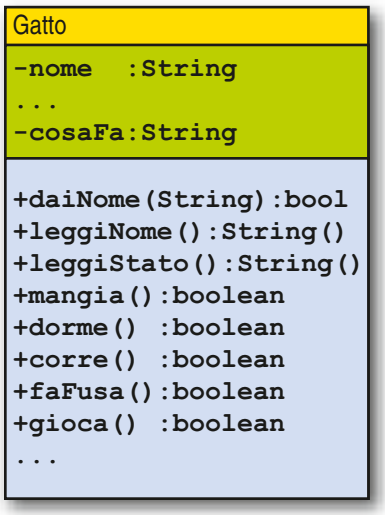
Il diagramma della classe è già stato descritto e ne indichiamo gli elementi principali.

- Nome della classe:
  - se è una singola parola ha l'iniziale maiuscola (per esempio, `Animali`);
  - se è composta da più parole, l'iniziale di ogni parola è maiuscola (per esempio, una classe di nome `UccelliRapaci`).
- Attributi:
  - in ogni caso con le iniziali in minuscolo (per esempio, `colore` e `formaOcchi`).
- Metodi:
  - in ogni caso con le iniziali in minuscolo (per esempio, `veloce()` e `mangiaPizza()`).

Graficamente lo abbiamo già utilizzato nella forma sintetica: ▶

Esiste anche la forma completa, in cui vengono aggiunti gli indicatori di visibilità semplicemente mettendo i caratteri + e – davanti ai metodi e ai campi. Si ha:
- – per `private`;
- + per `public`.

Un esempio è riportato a lato.



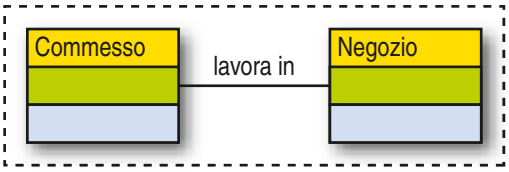
Associazioni

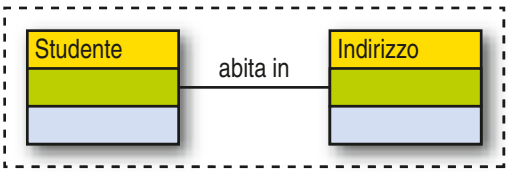
Le classi sono collegate tra loro con apposite linee, creando così un grafo, in modo da rappresentare le relazioni tra oggetti che caratterizzano ogni progetto Object Oriented. In primo luogo è necessario stabilire la differenze tra diversi tipi di relazione che possono intercorrere tra le classi, che possono essere sostanzialmente di cinque tipi:
- semplice;
- complessa;
- riflessiva;
- forte;
- debole.

Mediante le linee continue e tratteggiate, e simboli grafici quali frecce orientate, colorate o meno, eventualmente con numeri o qualche semplice indicazione, è possibile rappresentare ogni tipo di relazione.

Relazione semplice

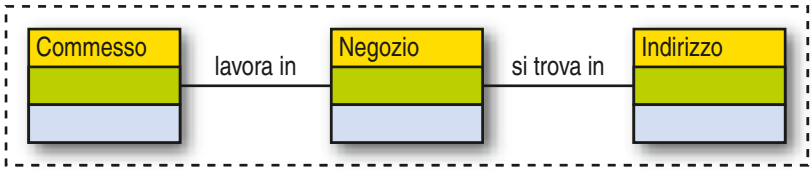
Il concetto di relazione è quello classico della matematica (in particolare della teoria degli insiemi), derivante dal termine inglese relationship, anche tradotto con associazione o correlazione.
La relazione semplice è quella che lega due classi quando i membri della prima classe possono associarsi concettualmente ai membri della seconda classe, cioè quando un attributo può essere un oggetto di una seconda classe. Viene denominata associazione e graficamente si rappresenta con una linea che connette due classi, con il nome dell'associazione appena sopra la linea stessa.



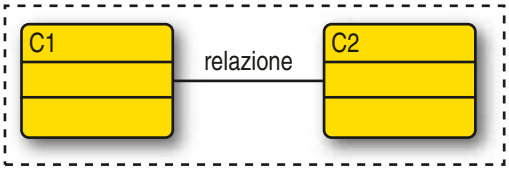
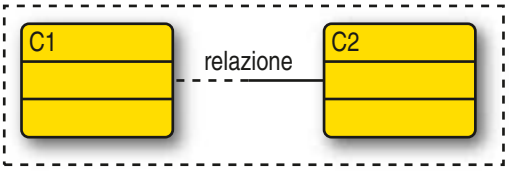
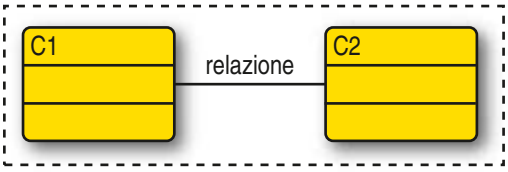
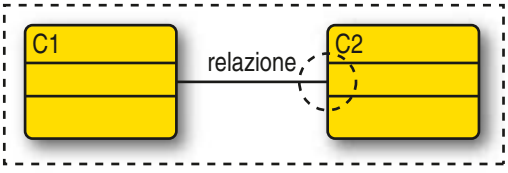
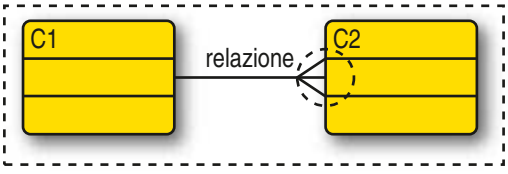


Relazione complessa

Un'associazione può essere più complessa della semplice connessione tra due classi.

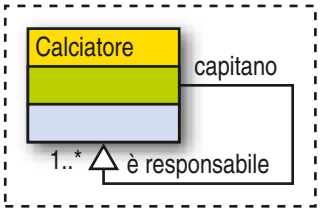


1. È possibile che più classi possano connettersi a una singola classe.
2. Più oggetti possono avere il medesimo valore dell'attributo di una classe associata: in questo caso si parla di cardinalità (o molteplicità) e può assumere le seguenti forme:
   - uno a uno;
   - uno a molti;
   - uno a uno o più;
   - uno a zero o uno;
   - uno a un intervallo limitato (per esempio: 1 a 2-20);
   - uno a un numero esatto *n*;
   - uno a un insieme di scelte (per esempio: 1 a 5 o 8).
3. La relazione può essere opzionale oppure obbligatoria:
   - opzionale se non tutti gli oggetti sono in relazione tra loro;
   - obbligatoria quando ogni oggetto della prima classe deve essere in relazione con quelli della seconda classe.

Simbolo	Descrizione
	Relazione Le relazioni vengono rappresentate con una linea che congiunge le classi
	Opzionalità L'opzionalità è rappresentata da una linea tratteggiata e indica che possono esistere istanze di C1 che non sono associate ad alcuna istanza di C2
	Obbligatorietà È rappresentata da una linea continua e indica che tutte le istanze di C1 devono essere associate ad almeno una istanza di C2
	Cardinalità Relativamente a quanto contenuto nel cerchio tratteggiato, per una o più istanze di C1, l'associazione individua una sola istanza di C2
	Cardinalità Relativamente a quanto contenuto nel cerchio tratteggiato, per una o più istanze di C1, l'associazione individua una o più istanze di C2

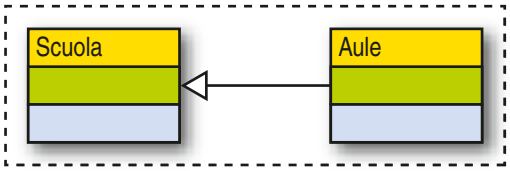
Relazione riflessiva

È un caso in cui gli oggetti della classe sono in relazione con se stessi, per esempio alunni e capoclasse, giocatori e capitano, particolarmente utilizzata nei database.



Aggregazione (o associazione debole)

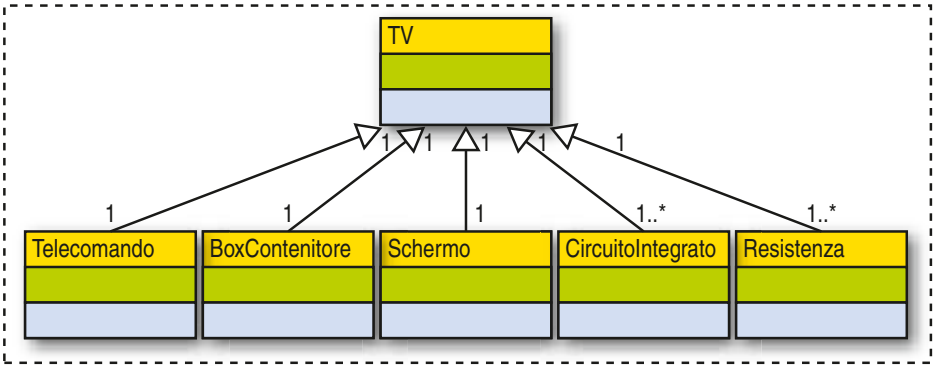
Una classe può rappresentare il risultato di un insieme di altre classi che la compongono: le classi che costituiscono i componenti e la classe finale sono in una relazione particolare chiamata part – whole (parte – intero).



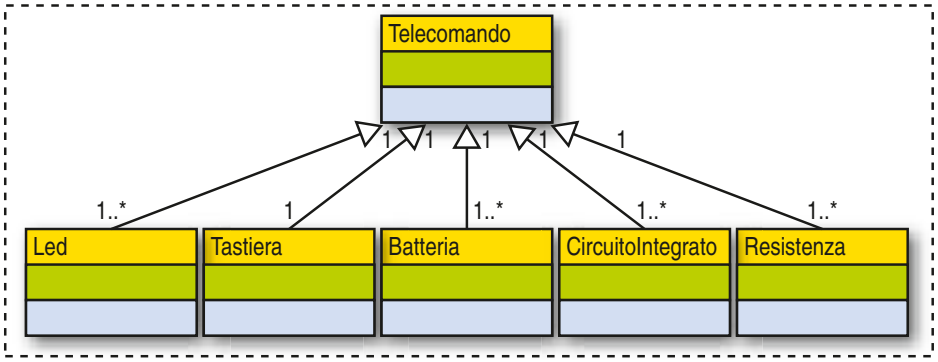
Una linea unisce "l'intero" per un componente con un triangolo raffigurato sulla linea stessa vicino all'"intero": nell'esempio della figura l'intero è la scuola che è composta di aule.
Nel caso di una aggregazione con diversi componenti è possibile effettuare una rappresentazione gerarchica in cui l'"intero" si trova in cima e i componenti ("parte") al di sotto.

ESEMPIO 1

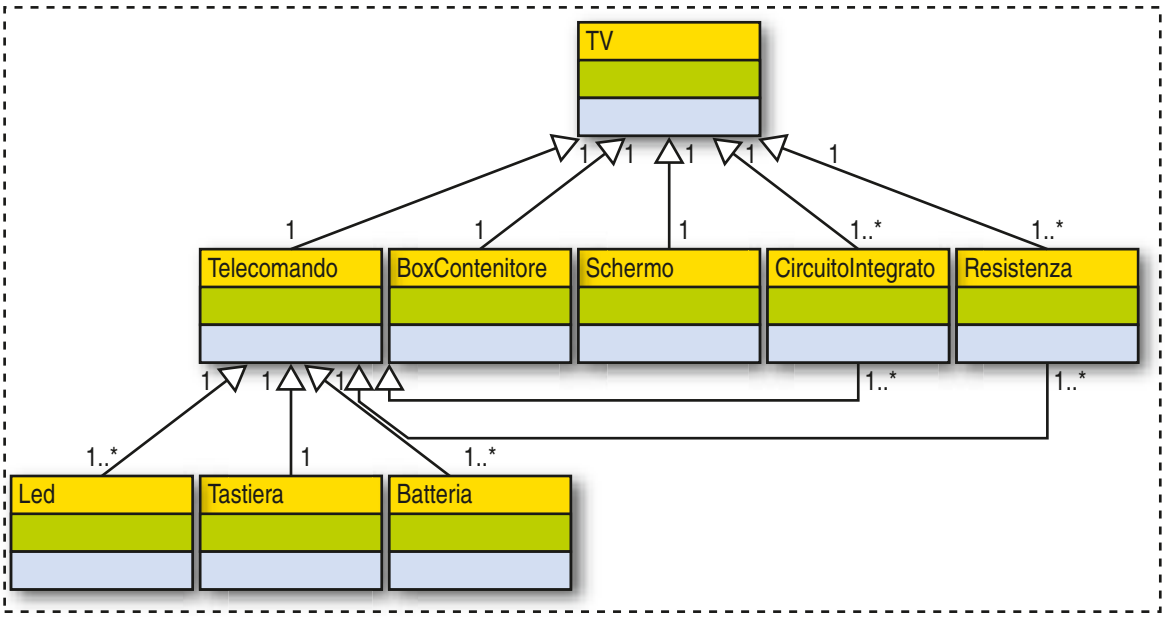
Vediamo un esempio completo e articolato di aggregazione: un televisore.
Innanzitutto individuiamo le "parti" che costituiscono un televisore (senza entrare nei dettagli): ogni TV ha un involucro esterno, uno schermo, degli altoparlanti, delle resistenze, dei transistor, alcuni circuiti integrati, un telecomando e altri componenti che trascuriamo in questo esempio. Il suo schema UML è il seguente:



A sua volta, il telecomando è costituito da diverse parti: resistenze, transistor, batterie, led ecc.

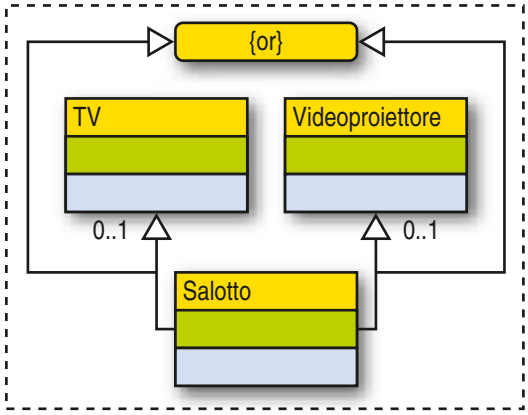


È possibile unificare i due modelli in un unico diagramma individuando le classi comuni ai due sottosistemi:



Qualche volta l'insieme di possibili componenti di un'aggregazione si identifica in una relazione OR (oppure OR esclusivo XOR): per modellare tale situazione è necessario utilizzare una constraint, che consiste nella parola OR all'interno di parentesi graffe su una linea tratteggiata che connette le due linee parte-intero.

Per esempio, supponiamo di voler modellare la situazione del riproduttore televisivo di un salotto: si ha a disposizione o un televisore oppure un videoproiettore, ma non tutti e due. Il diagramma è quello riportato nella figura a lato. ▶

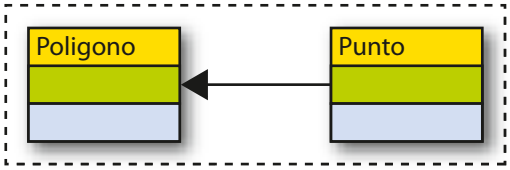


AGGREGAZIONE

Nella aggregazione gli oggetti possono far parte di classi più complete a livello superiore: l'oggetto di una classe condivide le proprietà della classe superiore, cioè ne eredita le proprietà.

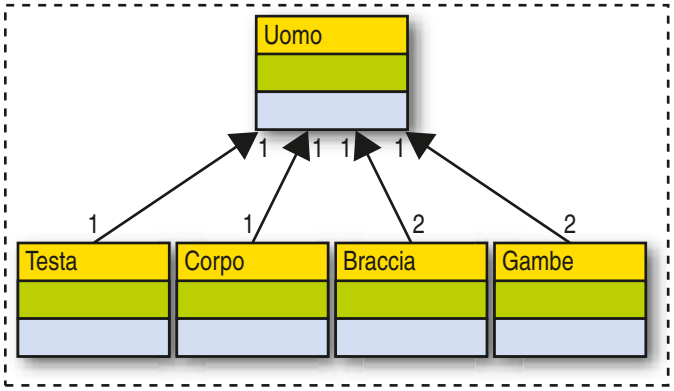
Composizione (o associazione forte)

Una composizione è un tipo più forte di aggregazione in quanto ogni componente può appar-



tenere soltanto a un "intero": nel caso per esempio di un Poligono, un Punto non può appartenere contemporaneamente a due Poligoni.

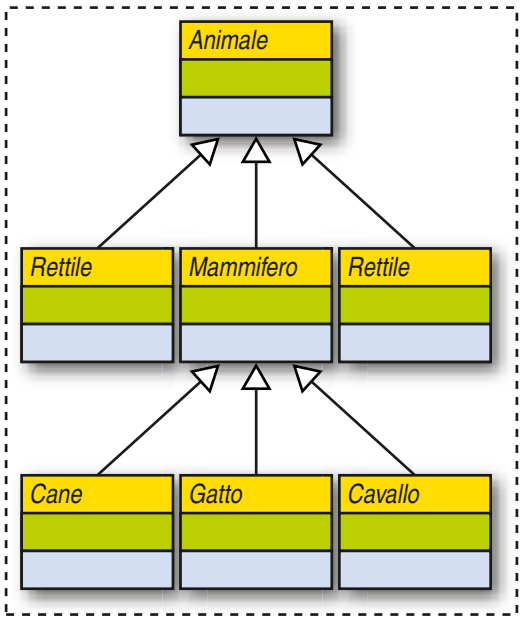
Il simbolo utilizzato per una composizione è lo stesso utilizzato per un'aggregazione, eccetto il fatto che la punta della freccia è colorata di nero. Come secondo esempio prendiamo l'anatomia di un uomo: ogni persona ha (tra le altre cose) una testa, un corpo, due braccia e due gambe e queste "parti" appartengono solo a una specifica persona": graficamente viene rappresentato con il grafico a lato. ▶



Esiste anche una forma "ibrida" tra l'aggregazione e la composizione: si chiama associazione composita, e in essa ogni componente appartiene *esattamente a un intero*.

Ereditarietà e generalizzazione

In UML l'ereditarietà viene rappresentata con una linea che connette la classe padre alla classe discendente e dalla parte della classe padre si inserisce un triangolo (una freccia bianca).
La rappresentazione di classi astratte viene indicata scrivendo il nome della classe in corsivo.



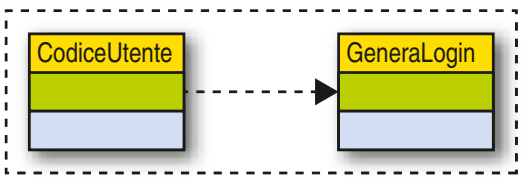
Dipendenza

Un ultimo aspetto di relazione tra le classi che necessita di essere riportato è la dipendenza, che avviene tra due classi quando la modifica di una classe crea (o potrebbe creare) qualche conseguenza sull'altra. La classe "autonoma" prende il nome di fornitore (o supplier) e la classe dipendente è chiamata client.

DIPENDENZA

La dipendenza parte solo dalla classe `client` e arriva alla `supplier` e viene indicata con una linea tratteggiata terminante con una freccia orientata verso la classe `supplier`.

Nell'esempio della figura la classe `CodiceUtente` dipende per esempio dalla classe `GeneraLogin`: se viene cambiata la modalità di assegnazione delle login, tutti i `CodiceUtente` devono essere riassegnati.



ESEMPIO 2

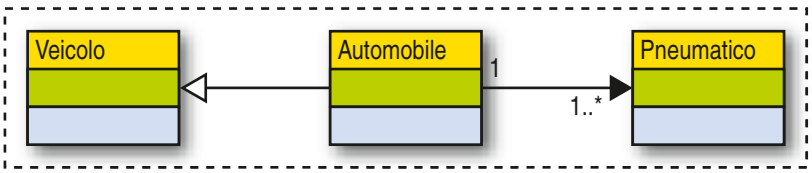
Realizzare il diagramma UML delle classi `Veicolo`, `Automobile` e `Pneumatico`.

Soluzione

Scriviamo "a parole" il legame esistente tra le classi:
"ogni automobile è un veicolo e ogni veicolo ha uno (o più) pneumatico":
quindi

- *è un* : ereditarietà;
- *ha uno* : associazione.

Il diagramma è a lato.



Un esempio completo: aule scolastiche

Situazione

Aula Scolastica: si modelli la situazione di un'aula scolastica.

Soluzione

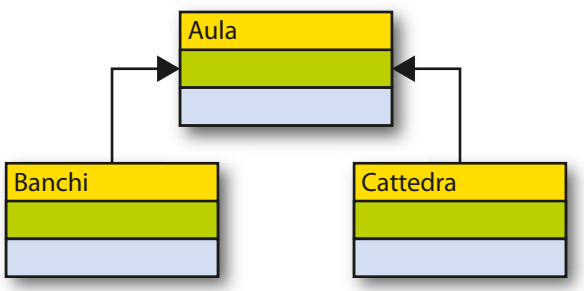
Descriviamo sinteticamente una generica aula scolastica per poi individuarne le classi: "Un'aula è un locale che ha un nome (IV°I3, laboratorio1 ecc.) ed è composta da una cattedra e *n* banchi; può ospitare *m* corsi (informatica, sistemi, inglese ecc.) e a ogni corso è assegnato uno e un solo docente ed è seguito da uno o più studenti. Il docente e gli studenti hanno un nome mentre gli studenti hanno anche un numero di matricola."
"Rivisitiamo" la descrizione applicando le regole empiriche che in prima approssimazione ci permettono di individuare classi, metodi e attributi. I sostantivi identificano potenziali classi o attributi di classi del nostro modello. I verbi identificano potenziali metodi o proprietà.

"Un'aula è un locale che ha un nome (IV°I3, laboratorio1 ecc.) ed è composta da una cattedra e *n* banchi; può ospitare *m* corsi (informatica, sistemi, inglese ecc.) e a ogni corso è assegnato uno e un solo docente ed è seguito da uno o più studenti.

Il docente e gli studenti hanno un nome e gli studenti hanno anche un numero di matricola."

Passo 1: analisi prima riga e determinazione prime classi

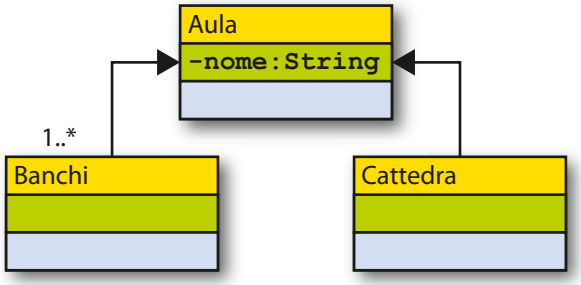
Dalla prima riga:
"Un'aula è un locale che ha un nome ed è composta da *n* banchi e una cattedra"
si ottengono tre classi (Aula, Banchi, Cattedra) che hanno tra loro relazione di associazione forte (composizione) in quanto ogni componente può appartenere soltanto a un "intero", cioè ogni cattedra appartiene a una sola aula così come i banchi. Il diagramma UML è riportato a lato. ▶



Passo 2: molteplicità classi individuate

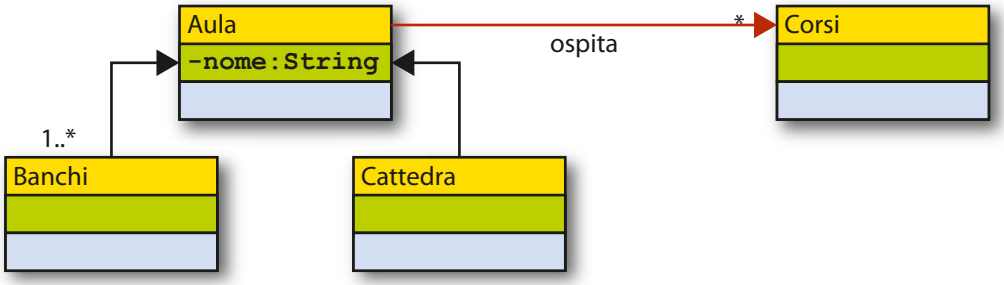
Sempre nella prima riga possiamo ora indicare la molteplicità delle relazioni, ricordando come vengono indicate in UML:

1. tra Aula e Banchi, relazione 1 o molti;
2. tra Aula e Cattedra, relazione 1 a 1.



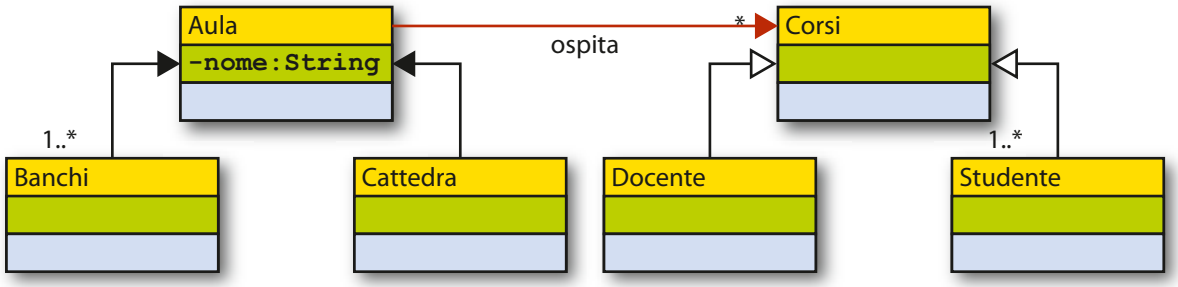
Passo 3: analisi seconda riga e completamento classi

"L'aula può ospitare *m* corsi..."
Si introduce una nuova classe, la classe `Corsi`, con molteplicità 0 o molti, che è in una relazione di associazione semplice con la classe `Aula`.
"... a ogni corso è assegnato uno e un solo docente ed è seguito da uno o più studenti."
Due nuove classi, la classe `Docenti` e `Alunni`, con molteplicità rispettivamente 1 a 1 e 1 o molti con la classe `Corsi`.



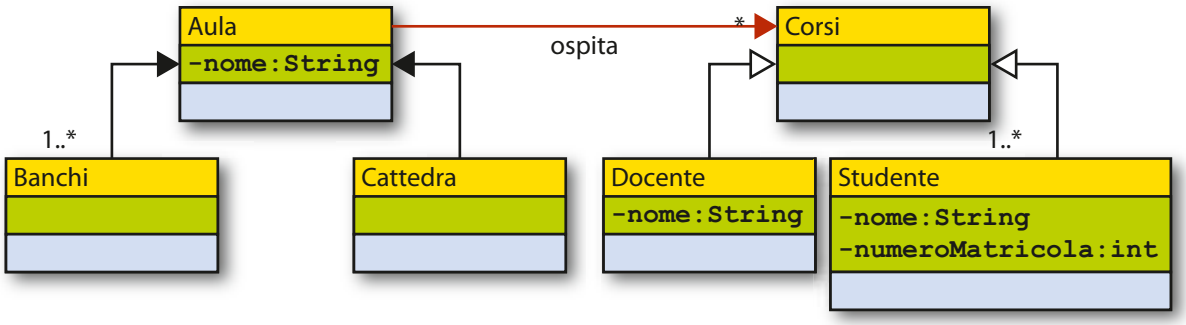
Passo 4: analisi ultima riga

"Il `docente` e gli `studenti` hanno un nome e gli studenti hanno anche una matricola."
L'ultima riga ci permette di aggiungere nelle classi alcuni attributi: `nome` e `numeroMatricola`.



Passo 5: analisi gerarchia tra le classi

Infine è possibile effettuare un'osservazione tra le classi `Docente` e `Studenti`; visto che hanno un attributo in comune, si può individuare una gerarchia di classi introducendo una classe `Persona` da cui discendono entrambe, dato che il Docente è una ("is a") Persona e lo Studente è una ("is a") Persona. Il diagramma UML completo delle classi rappresentato di seguito.



Conclusione

In questa unità didattica sono stati descritti i principali modelli di realizzazione di un'applicazione software e si è introdotto il linguaggio UML come esempio di stile standardizzato per la documentazione di un progetto. Concludiamo questa trattazione con un breve cenno a due fattori di qualità del software che sono semplici da valutare ma fondamentali da considerare in fase di progettazione orientata agli oggetti (OOD, *Object Oriented Design*) in quanto rappresentano un indicatore di riusabilità delle classi.

ACCOPPIAMENTO E COESIONE

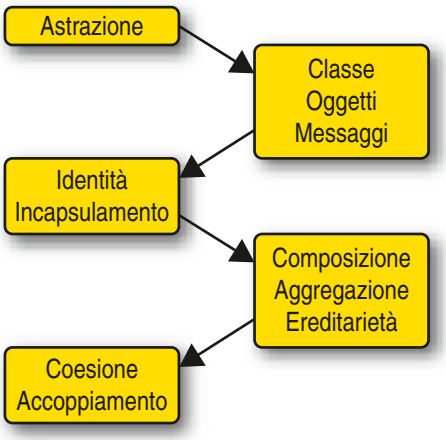
- Accoppiamento: indica quanto due classi di un modello siano dipendenti.
- Coesione: indica quanto una classe supporta il medesimo scopo all'interno del sistema.

Ogni classe deve rappresentare un singolo concetto e quindi i metodi pubblici e costanti elencati nell'interfaccia pubblica devono avere una buona coesione, ovvero devono essere strettamente correlati al singolo concetto rappresentato dalla classe.

Un "buon software" va scritto tenendo ben presente quelli che deve avere come obiettivi di progetto, che devono essere soddisfatti dalla relazione tra le classi, ovvero:

- basso accoppiamento: si deve essere in grado di capire il codice di una classe senza leggere i dettagli delle altre e, all'occorrenza, poter modificare una classe senza che le modifiche abbiano impatto sulle altre classi;
- alta coesione: si deve essere in grado di capire ciò che una classe o un metodo fanno in modo univoco e autonomo, in modo da poter effettuare il riuso delle classi e dei metodi.

Possiamo riassumere in uno schema la modalità di progettazione OOD per quanto riguarda l'individuazione delle relazioni tra le classi: dopo il processo di astrazione si individuano le classi con i loro metodi e le loro interazioni mediante i messaggi che possono "scambiarsi". Quindi si procede con l'individuazione delle classi simili e la loro fusione, poi con l'eliminazione dei duplicati e successivamente con l'incapsulamento e la descrizione delle associazioni tra le classi stesse. Per ultimo si valuta il grado di coesione e di accoppiamento tra di esse al fine di migliorare la qualità totale del progetto.



Verifichiamo le conoscenze

Esercizi a scelta multipla

1 L'assistenza può essere:

- ○ teorica/tecnica
- ○ teorica/operativa
- ○ operativa/tecnica
- ○ pratica/teorica

2 La documentazione riservata all'utente consiste in (indicare la risposta errata):

- ○ documentazione inerente all'installazione del prodotto
- ○ documentazione inerente all'avvio dell'utilizzo del prodotto
- ○ documentazione inerente alla struttura degli archivi
- ○ documentazione interna
- ○ manuale d'uso

3 La documentazione tecnica consiste in:

- ○ documentazione interna/esterna
- ○ documentazione on line/esterna
- ○ documentazione pratica/teorica
- ○ documentazione utente/programmatore

4 L'acronimo UML significa:

- ○ *Uniform Module Language*
- ○ *Unified Module Language*
- ○ *Uniform Modeling Language*
- ○ *Unified Modeling Language*

5 I diagrammi di base dell'UML sono:

○ 6; ○ 7; ○ 8; ○ 9

6 Quale tra i seguenti non è un diagramma di base dell'UML?

- ○ Class Diagram
- ○ Object Diagram
- ○ Method Diagram
- ○ State Diagram
- ○ Activity Diagram
- ○ Component Diagram
- ○ Deployment Diagram

7 Le associazioni possono essere (indicare quella errata):

- ○ deboli
- ○ forti
- ○ complesse
- ○ semplici
- ○ oggettive
- ○ riflessive

8 Quale tra le seguenti non è una forma di cardinalità della OOP?

- ○ zero a zero
- ○ uno a zero
- ○ uno a uno
- ○ uno a molti

Test vero/falso

1	La documentazione inerente all'istallazione fa parte della documentazione utente.	V F
2	La documentazione esterna all'istallazione fa parte della documentazione utente.	V F
3	L'UML viene anche chiamato *Virtual Modeling*.	V F
4	L'UML realizza la *visualizzazione grafica del modello*.	V F
5	Il *Collaboration Diagram* è un diagramma base di UML.	V F
6	L'*Use State Diagram* è un diagramma base di UML.	V F
7	L'associazione tra le classi avviene tramite linee continue o tratteggiate.	V F
8	Il simbolo utilizzato per una composizione ha la punta della freccia bianca.	V F
9	L'ereditarietà si indica con una freccia dalla parte della classe padre.	V F
10	La dipendenza tra le classi si indica tramite una linea tratteggiata.	V F

Verifichiamo le competenze

Esprimi la tua creatività

1. Descrivi il diagramma delle classi di un impianto HI-Fi, con livello di dettaglio fino ai componenti elettronici. Genera un diagramma UML dell'implementazione delle classi.

2. Descrivi il diagramma delle classi di un'automobile, ereditando dalla classe **`Automezzi`**, dalla classe **`Cerchio`** e dalla classe **`Sedia`**. Genera un diagramma UML dell'implementazione delle classi.

3. Descrivi il diagramma delle classi di un complesso musicale (comprensivo di musicisti, ereditando dalla classe **`Strumenti`**, dalla classe **`ApparecchiatureElettroniche`** e dalla classe **`Umano`**. Generare un diagramma UML dell'implementazione delle classi.

4. Descrivi il diagramma delle classi di un circo, comprendendo animali e artisti, ereditando proprio dalle classi **`Animali`** e **`Artisti`**. Genera un diagramma UML dell'implementazione delle classi.

5. Descrivi il diagramma delle classi di un videogioco tipo Battaglia Navale ereditando dalle classi **`MezzoDiTrasporto`** e **`Arma`**. Genera un diagramma UML dell'implementazione delle classi.

6. Progetta un programma per "l'algebra dei numeri complessi", dove è possibile calcolare le operazioni tra numeri complessi e rappresentarli graficamente sul piano di Gauss. Genera un diagramma UML dell'implementazione delle classi.

7. Progetta un programma per un semplice videogioco tipo "caccia all'orso": l'orso si muove casualmente sull'orizzonte cambiando velocità e direzione mentre il cacciatore spara modificando l'angolo di tiro. Genera un diagramma UML dell'implementazione delle classi.

8. Progetta un programma che emette lo scontrino al casello autostradale: gli automezzi sono classificati in categorie di pedaggio e i pedaggi sono differenziati nei giorni feriali e festivi. Genera un diagramma UML dell'implementazione delle classi.

9. Realizza un programma di gestione delle fatture prevedendo due tipi diversi di righe che descrivono il singolo articolo: uno che descrive prodotti che vengono acquistati in una determinata quantità numerica (come, per esempio, "tre tostapane") e un altro che descrive un addebito fisso (come "spedizione: euro 5,00" oppure "imballo euro 2,00"). Genera un diagramma UML dell'implementazione delle classi.

10. Realizza un programma che si possa utilizzare per tracciare una scena urbana, con case, strade e automobili. Gli utenti possono aggiungere a una strada case e automobili di vari colori. Dopo avere identificato le classi e i metodi, fornisci i diagrammi UML.

11. Progetta un semplice editor di grafica che consenta all'utente di aggiungere a un pannello un insieme misto di sagome (ellissi, rettangoli, linee e testo in colori diversi). Fornisci i comandi per salvare e ripristinare il disegno. Per semplicità,puoi usare un'unica dimensione per il testo e non sei tenuto a riempire le sagome. Individua le classi e fornire un diagramma UML del progetto.

12. Descrivi il diagramma delle classi che consentae di rappresentare la situazione di una pizzeria, dove per ogni pizza è necessario individuare ingredienti e calorie, oltre a descrivere ogni ingrediente in base alla sua origine (cioè animale, vegetale, latticino ecc.) per la scelta, per esempio, dei clienti vegetariani o intolleranti a qualche tipologia di alimento.

e-ISBN 978-88-203-5334-6

www.hoepli.it

Ulrico Hoepli Editore S.p.A.
via Hoepli, 5 - 20121 Milano
e-mail hoepli@hoepli.it